UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. Ingeniería de Sistemas Informáticos

Trabajo Fin de Máster

Máster Universitario en Software de Sistemas Distribuidos y Empotrados

Arquitecturas Software para Microservicios: Una Revisión Sistemática de la Literatura

Junio 2018

Autor: Elena Albertos Gómez

Tutor: Jorge Enrique Pérez Martínez

Co-Tutor: Jessica Díaz Fernandez

Agradecimientos

En primer lugar, quisiera agradecer la ayuda prestada a mi tutor del trabajo fin de master, Jorge Enrique Pérez Martínez, cuya orientación y conocimientos, así como sus valiosas sugerencias en relación a la arquitectura del software han sido indispensables para la finalización del estudio. Con igual énfasis, mi agradecimiento a Jessica Díaz Fernandez, por su implicación y apoyo en las fases de búsqueda y análisis de los estudios primarios, que han constituido la base de este proyecto. Ambos han sabido guiarme en la elaboración del trabajo, con mucha paciencia y entusiasmo.

También destacar la colaboración prestada por algunos autores, que me han facilitado de forma desinteresada sus artículos a través de la plataforma ResearchGate.

Por último y no menos importante, agradecer la paciencia mostrada por mi familia, que me ha apoyado durante todas las distintas etapas, facilitándome el espacio y el tiempo de estudio (en algunas ocasiones “a regañadientes”, especialmente mis hijos) que ha sido necesario para la realización del trabajo fin de master.

Resumen

La arquitectura de microservicios, o microservicios es un nuevo concepto que ha crecido en popularidad en los últimos años en la comunidad del desarrollo de sistemas software. A diferencia de los sistemas monolíticos, donde la lógica de negocio se encuentra en un único proceso con dependencias fuertes entre las unidades, los microservicios pretenden desacoplar la lógica de negocio en servicios independientes que se comunican entre ellos con un mecanismo ligero. Estos servicios pueden estar implementados en diferentes lenguajes de programación y usar distinta tecnología de almacenamiento. Son fácilmente escalables y desplegados en un sistema distribuido debido a su debil acoplamiento entre componentes y a su carácter distribuido.

La aparición del término microservicios ha originado "ríos de tinta" sobre este concepto y su aplicación. Entre otras cuestiones, algunos autores hablan ya de un estilo arquitectónico de microservicios. El objetivo de este trabajo de investigación es revisar toda la literatura que existe desde el año 2014, fecha en la que apareció el artículo de Martin Fowler: Microservices, (Fowler & Lewis, 2014) que traten específicamente sobre la arquitectura de microservicios, es decir, aquellos que presenten un nuevo modelo arquitectónico o arquitectura de referencia, desligado del concepto SOA. Por lo tanto, este estudio intenta dilucidar: a) si existe tal estilo y b) cómo está definido en términos de sus elementos constituyentes y restricciones topológicas y de comunicación, así como el lenguaje que lo describe.

Mediante la revisión de la literatura basada en la búsqueda sistematica en bases de datos cientificas, se han seleccionado aquellos artículos que responden a las preguntas de investigación previamente definidas. Una vez realizada la extracción y síntesis de los datos sobre los estudios primarios seleccionados se concluye que actualmente no hay estilos arquitectónicos propios ni lenguajes descriptivos asociados a los microservicios. Debido a las caracteristicas particulares, en el que un sistema se divide en numerosos servicios independientes y autonomos, resulta complicado tener una visión completa de la arquitectura. No obstante, este diseño es cada vez más utilizado por distintas empresas y organizaciones, con el fin de abordar una mayor complejidad empresarial con reglas de negocio cambiantes. Por lo tanto, se muestra la necesidad de definir un estilo arquitectónico que sirva de referencia en la construcción de sistemas informáticos reutilizables basados en microservicios.

Abstract

Microservices architecture or Microservices is a new concept that has gained quite some popularity just recently over the community-based software development. In contrast with the monolithic approach, where all business logic is placed in a single process with high coupling between modules, the microservices are built to decompose the business logic into independent services, whose communicate with each other through a lightweight mechanism. These small services can be implemented in different languages using different database technology. They can be easily scaled and deployed in a distributed system because of its distributed nature and loosely coupled design.

The apparition of the microservice term has provoked many discussions around this concept and its application. Among other issues, some authors talk about a microservices architectural style. The goal of this research study is to review the literature since 2014 , published date of the Martin Fowler's article: Microservices, (Fowler & Lewis) that mention explicitely a new architectural model or reference architecture, separated from the service-oriented architecture definition. Therefore, this research aims to figure out: a) if there is an own architectural style; b) and how it is defined in terms of components, connectors, and topological restrictions, as well as how the communication mechanism and the architectural description language is described.

Thus, a review of literature is perfomed, based on a systematic search on literature in the electronic databases. As a result, some articles are selected in order to answer the proposed research questions. Once the data are extracted and summarized, we can conclude that neither architectural styles nor architecture description languages based on microservices are currently distinguished. Since the microsevices are designed to be isolated, small and self-contained, it is difficult to have a large overview on system architecture. Nevertheless, the microservices architecture has become an approach to support fast changing business models for many companies. Therefore, it is very important to define an architectural style as a common reference for building reusable microservices-based systems.

Contenido

Índice de Figuras

Índices de Tablas

Capítulo 1. Introducción

Este capítulo presenta al lector la motivación del autor para la elaboración del presente trabajo de fin de máster, además detalla el contexto, metodología y estructura utilizada para su realización.

1.1 Motivación

Nuestra motivación en el tema sobre arquitectura en microservicios, surge por la creciente tendencia que existe en la adopción de sistemas basados en microservicios, y por contra la dificultad de encontrar modelos arquitectónicos que se ajusten a sus peculiaridades. Por ello, se justifica la necesidad de buscar artículos, publicaciones científicas que traten de una forma más concreta sobre sistemas software basados en arquitectura en microservicios.

El principal objetivo de los sistemas basados en microservicios es descomponer grandes proyectos de software en pequeñas unidades desacopladas que se comunican entre sí mediante una interfaz sencilla. Constituye un diseño arquitectónico contrario al enfoque tradicional y monolítico sobre las aplicaciones, en el que todo se crea en una única pieza.

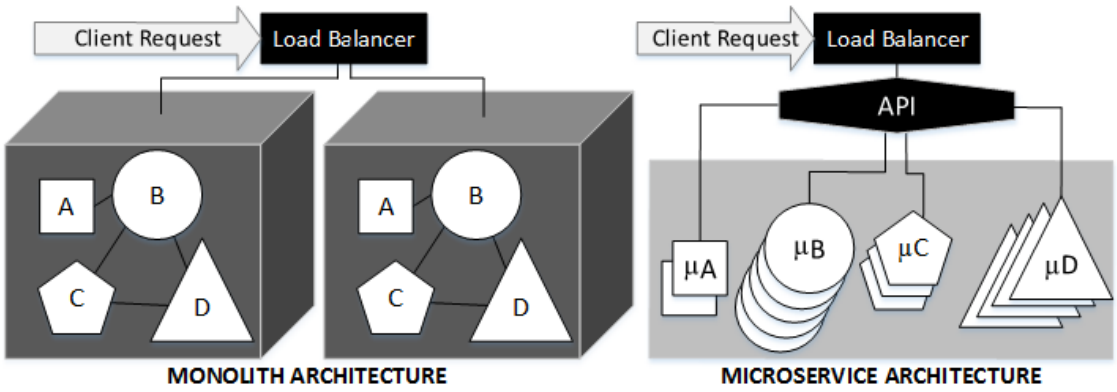


FIGURA 1 MICROSERVICES VS MONOLITIC ARCHITECTURE (Long, Yang, & Kim, 2017)

1.1.1 Objetivos

Objetivo General

El principal objetivo de este trabajo fin de master, es realizar un estudio sobre los artículos publicados en distintas bases de datos científicas y conferencias, sobre las arquitecturas en microservicios, es decir, aquellos que traten específicamente sobre un nuevo modelo arquitectónico o arquitectura de referencia de los microservicios.

Objetivos Específicos

- Realizar una revisión sistemática de la literatura, con el fin de analizar las publicaciones sobre arquitectura en microservicios.

- Caracterización de la tecnología en microservicios, mencionando las ventajas y los inconvenientes que presentan en la adopción de este modelo por distintas empresas y/o organizaciones.
- Identificar algunas tecnologías que subyacen en este puntero modelo arquitectónico.
- Realizar la comparativa sobre publicaciones que tratan únicamente de microservicios, y las que especifican estilos o referencias sobre la arquitectura.
- Proponer un nuevo reto sobre la creación de un modelo arquitectónico de base que sirva de referencias a las tecnologías en microservicios.

1.2 Metodología

La metodología utilizada en esta SLR es la definida por Kitchenham (Kitchenham & Charters, 2007), en dicha metodología se propone un proceso de tres componentes: Planeamiento de la revisión, conducción de la revisión y reporte de la revisión.

A continuación, se presentan las subsecciones de cada una de las etapas:

- Planeamiento de la revisión
    - Identificar las necesidades de la revisión.
    - Desarrollo de un protocolo de revisión.
- Conducción de la revisión
    - Identificación de la investigación.
    - Selección de estudios primarios.
    - Aseguramiento de la calidad de los estudios.
    - Extracción de datos.
    - Síntesis de los datos.
- Informe de la revisión

1.3 Estructura de la tesis

El presente trabajo de tesis se divide en cinco capítulos (junto a varios anexos) detallados a continuación:

El capítulo 1 expone la motivación de la realización del estudio de investigación asociado al Trabajo Fin de Master, los objetivos generales y específicos que se pretende conseguir, y por último la metodología utilizada para la consecución de la revisión sistemática de la literatura de la investigación.

El capítulo 2 amplia los conceptos sobre arquitecturas software que existen actualmente y sobre el término microservicios. Se ofrece una definición completa sobre los microservicios, así como los beneficios y retos a los que se enfrentan, incluyendo un análisis desde el punto de vista de los atributos de calidad.

La revisión sistemática de la literatura aplicada en el trabajo de investigación está definida en el capítulo 3, donde se describe el protocolo de búsqueda, y las estrategias de extracción y síntesis de la información obtenida de los estudios primarios.

El capítulo 4 presenta el informe de resultados respondiendo a las preguntas de investigación definidas, los hallazgos y finalmente la discusión de dichos resultados.

Por último, el capítulo 5 describe las conclusiones del presente estudio de revisión de literatura alrededor de la arquitectura de las aplicaciones basadas en microservicios, y muestra ideas para realizar trabajos futuros sobre el tema.

Los Anexos muestran todo el trabajo que ha servido de apoyo a la realización del proyecto. El Anexo A define la Kappa de Cohen, que es una medida de acuerdo entre jueces, y el Anexo B muestra los resultados obtenidos al aplicar esta medida durante la selección de los estudios primarios. Adicionalmente, los anexos C y D presentan el resumen de cada estudio primario, donde se recoge información detallada en base a unas preguntas, así como los de mapas conceptuales para la identificación de patrones comunes.

Capítulo 2. Conceptos Previos

Este capítulo presenta un conjunto de definiciones, características y clasificaciones que permitan al lector comprender el contexto en el cuál se desenvuelve la SLR planteada en el siguiente capítulo.

2.1 Arquitecturas Software

Existen varias definiciones en relación a una arquitectura software en sistemas informáticos, y una de ellas lo define como el diseño de alto nivel de un conjunto de estructuras que incluyen elementos software, las relaciones entre ellos y las propiedades de ambos, que deben satisfacer unos requisitos de negocio. Así también se entiende como arquitectura software o lógica a un conjunto de patrones y abstracciones que proporcionan un marco de referencia para guiar en la construcción de sistemas informáticos, donde programadores, analistas y todos los ingenieros compartan una línea de trabajo común.

Entre los conceptos ligados a la arquitectura de software es posible destacar, los estilos y los patrones arquitectónicos, que son una colección de decisiones de diseño arquitectónico aplicable a un contexto de desarrollo y a problemas de diseño recurrentes. (Taylor, Medvidović, Dashofy, 2010).

El resultado que captura dichas decisiones de diseño se denomina arquitectura software. El modelado trata de documentar y concretar dichos conceptos de diseño. Un mismo sistema puede tener varios modelos, que pueden variar en el detalle, siendo más o menos concreto, tratar sobre distintas perspectivas arquitectónicas que intentan capturar del sistema (estructural o de comportamiento, estática o dinámica, etc..), y en el tipo de lenguaje o notación que utilizan. Estos lenguajes de representación se denominan ADL (Architectural Description Language) (Bass, Clements, & Kazman, 2012).

2.1.1 Patrones arquitectónicos

Es un conjunto de decisiones de diseño arquitectónico específicas que es aplicable a un problema de diseño recurrente, y parametrizado para considerar distintos contextos de desarrollo software en los que se presenta el mismo problema. (Taylor et al., 2010)

Por lo tanto, un patrón arquitectónico ofrece buenas soluciones de diseño reutilizables para un mismo problema en un contexto que ha sido verificado, validado y aceptado por la comunidad. La solución debe describir una relación entre los elementos y pueden estar caracterizados según el tipo de elementos arquitectónicos que utilizan. En los siguientes subapartados, se emplean las definiciones de diversos patrones de diseño (Bass et al., 2012).

2.1.1.1 Patrones de módulos (*Module Patterns*)

Son un conjunto de decisiones sobre cómo el sistema debe estar estructurado, ya sea en capas o unidades de datos. Ofrecen descomposición en módulos de sistemas con restricciones topológicas en un dominio concreto.

- **CAPAS (*LAYERED*)**

Uno de los patrones más utilizados es el basado en capas o niveles jerárquicos. Cada capa es un grupo de módulos que ofrece un conjunto de servicios, y expone su funcionalidad

a través de una interfaz API. La relación entre capas es unidireccional siendo las capas de nivel más alto las que pueden usar los servicios de las capas adyacentes definidas más abajo, sin permitir comunicación circular. Este patrón está diseñado para satisfacer los requisitos de modularidad, portabilidad, mantenibilidad y reusabilidad del código en diversas aplicaciones.

2.1.1.2 Patrones C&C (*Component-and-Connector Pattern*)

Representan un conjunto de decisiones asociado a un tipo de estructuras que consideran los aspectos en tiempo de ejecución. Definen los componentes, que serían las principales unidades funcionales de computación, y los conectores, que posibilitan la comunicación entre ellos.

- **INTERMEDIARIO (*BROKER*)**

En sistemas donde los clientes no conocen la ubicación y/o identidad de los servidores, se utiliza un intermediario, llamado broker, que localiza un servidor para satisfacer las demandas de un cliente. El cliente realiza una petición mediante una interfaz de usuario hacia el intermediario, el cual, la redirige al servidor quien procesa dicha petición. La respuesta del servidor es comunicada igualmente al intermediario y devuelta al cliente, si no ocurre ninguna excepción.

Por lo tanto, la conexión entre los componentes, cliente y servidor, puede realizarse a través del intermediario, que puede además utilizar otros proxys para la traducción y reenvío de mensajes. Este patrón favorece la disponibilidad y el rendimiento del sistema, debido a que el broker puede sustituir un servidor en fallo y balancear la carga entre los servidores.

- **MODELO-VISTA-CONTROLADOR (*MODEL-VIEW-CONTROLLER*)**

El patrón modelo-vista-controlador (MVC) separa la funcionalidad en tres distintos conceptos: el modelo, que es una representación de los datos y el estado de la aplicación, la vista, que interactúa con el usuario ofreciendo una visualización de la información y recogiendo señales por distintos dispositivos o interfaz, y el controlador, que es el intermediario y traduce los cambios de usuario en cambios al modelo o a la vista, notificando los cambios de estado. Normalmente, existen varias vistas y controladores asociados a un único modelo.

Facilita el desarrollo y las pruebas, y minimiza el impacto de cambios en cualquier componente (MVC), debido a un acoplamiento débil entre ellos. La comunicación se realiza mediante notificación por eventos.

- **TUBERIAS Y FILTROS (*PIPE-AND-FILTER*)**

Este patrón está caracterizado por sucesivas transformaciones de cadena de datos, realizadas por los filtros, que transforman datos que llegan en sus puertos de entrada hacia sus puertos de salida. Todos los componentes o filtros son conectados mediante tuberías, que mantienen la secuencia sin alterar los datos que pasan, si bien limita el formato de los mensajes y la velocidad de transmisión. Cada filtro es responsable de una parte de la transformación de los datos de entrada, y pueden ejecutarse en paralelo de forma asíncrona.

Debido a la independencia en el proceso de cada etapa, permite la simplificación en el diseño del proceso, así como en la reusabilidad del sistema. Es utilizado como diseño de aplicaciones de procesamiento de señales (front-end), y/o flujos de datos.

- **CLIENTE-SERVIDOR (*CLIENT-SERVER*)**

Uno de los patrones más utilizados en aplicaciones web y acceso a base de datos, que define dos principales componentes, el cliente, que solicita servicios, y el servidor, que provee con un conjunto de servicios. Estos componentes pueden actuar tanto como cliente como servidor. Puede existir un servidor central o múltiples distribuidos. Utiliza un conector de tipo petición/respuesta (request/reply protocol) para la invocación de servicios por parte de los clientes y para el envío de la repuesta del servidor en modo síncrono o asíncrono.

Este patrón simplifica el diseño, agrupando servicios comunes que pueden ser reutilizados, y dando acceso instantáneo a los nuevos clientes que solicitan el servicio. Sin embargo, los servidores pueden convertirse en cuellos de botellas o en puntos de fallo únicos. Por ello, los servidores se replican en entornos distribuidos con el fin de aumentar la disponibilidad y la escalabilidad del sistema.

- ***PEER-TO-PEER***

Todos los componentes, llamados "peers", pueden interactuar con otros solicitando sus servicios mediante un protocolo de comunicación basado en petición/respuesta. Cada nodo o componente puede ser tanto consumidor como proveedor de servicios, existiendo unos nodos especiales que se encargan del enrutamiento.

Debido a su carácter descentralizado, este patrón es utilizado en sistemas distribuidos con alta disponibilidad. Los "peers" son creados y eliminados sin impacto, lo que aumenta la escalabilidad. Por otro lado, aparecen otros problemas a resolver, como la gestión de la seguridad, la consistencia de datos y la disponibilidad de los servicios y los datos.

- **ORIENTADO A SERVICIOS (*SERVICE-ORIENTED*)**

Los elementos en este patrón incluyen proveedores y consumidores de servicios que pueden ser implementados en distintos lenguajes y plataformas. Los componentes tienen interfaces que describen sus servicios y garantiza atributos de calidad a través de un SLA (Service Level Agreement). Otros componentes intermediarios son necesarios, tales como, un bus de comunicación que transfiere y transforma los mensajes (ESB en aplicaciones SOA), un servidor de registro/descubrimiento de servicios y un orquestador para la coordinación del flujo de control. La conexión entre los componentes puede ser de tipo SOAP (SOAP protocolo para comunicación síncrona), REST (HTTP protocolo petición/respuesta), y mensajería asíncrona (punto a punto y suscripción/publicación).

La ventaja principal es la interoperabilidad, y la integración de múltiples sistemas heterogéneos. Además, se permite la reconfiguración dinámica y actualización del sistema en tiempo de ejecución sin interrumpir el sistema. Si bien, se añade complejidad y sobrecarga debido al uso de los componentes intermediarios (ESB, Service Discovery, etc...).

- **SUSCRIPCION/PUBLICACION (*PUBLISH-SUBSCRIBE*)**

Una aplicación diseñada con el patrón suscripción/publicación es aquella en la que un servidor envía una serie de servicios a los usuarios y otros usuarios se suscriben a ella

para ser avisados de nuevas actualizaciones o informaciones. Cualquier componente puede hacer a la vez de suscriptor y publicador. El conector escucha los cambios de estado siendo el encargado de notificar el evento a todos los suscriptores cuando un evento es anunciado por un componente.

Surge este patrón de diseño como solución a sistemas de objetos o procesos independientes (no se conocen entre ellos) que reaccionan por eventos generados en el entorno, y asimismo provocan reacciones a otros componentes por las notificaciones de dichos eventos. Las redes sociales, las listas de distribución, e incluso ERP (Enterprise Resource Planning) son ejemplos de utilización de este patrón, donde los componentes ignoran la identidad de los otros. Como consecuencia, resulta sencillo la modificación del sistema añadiendo o eliminando componentes, aunque ese carácter multidireccional (sin previo conocimiento) aumenta el retardo en la comunicación.

- **DATOS COMPARTIDOS (*SHARED-DATA*)**

El modelo computacional asociado es un sistema de datos compartidos, donde los componentes que acceden a los datos almacenados pueden realizan operaciones en dichas bases de datos. Por lo tanto, los datos compartidos interactúan con los componentes que acceden a dichos datos mediante un conector que puede ser de escritura y/o lectura, transaccional o no transaccional.

La consistencia, seguridad y la privacidad de los datos, son aspectos relevantes para dicho patrón, donde se pone énfasis a la forma del diseño del almacenamiento de los datos que puede constituir un cuello de botella y un punto de fallo único en el sistema. Por consecuencia, la replicación de los datos compartidos o incluso el uso de bases de datos no transaccionales se implementan para paliar dichos problemas que afectan al rendimiento y a la disponibilidad.

2.1.1.3 Patrones de asignación (*Allocation Pattern*)

Engloba un conjunto de decisiones sobre cómo el sistema está ligado a estructuras no software, tales como CPU, sistemas ficheros, redes, equipos desarrollo, etc. Consideran los entornos de ejecución, mediante la asignación de los elementos software a elementos de computación físicos.

- **MAP-REDUCE**

Este patrón provee una infraestructura para análisis de una gran cantidad de datos que se ejecutan en paralelo en múltiples procesadores que favorecen la disponibilidad y reducen la latencia. Está compuesto de una función Map, con múltiples instancias en distintos procesadores, que extrae y transforma el subconjunto de datos, y una función Reduce, con una única o varias instancias, que reduce los datos extraídos anteriormente (compuestos por <clave, valor>) y produce el resultado final. La infraestructura es la encargada de la asignación de las instancias de las funciones map-reduce en el hardware, la recuperación ante fallos por el procesamiento paralelo mediante reasignación a otros recursos, más las utilidades para la ordenación de listas intermedias producidas durante todo el ciclo.

- **MULTI-CAPA (*MULTI-TIER*)**

El objetivo primordial es la separación o desacoplamiento de las partes que componen un sistema software en distintos niveles. Una arquitectura **n-Tier** se refiere a la distribución física de las capas, es decir donde corre el código y los procesos. Una arquitectura *n-Layer* se refiere a la distribución lógica de las capas, es decir, como está estructurado el código (lógica de negocios, capa de presentación y capa de datos). Normalmente, se aplica a arquitecturas cliente-servidor o en capas, y define las restricciones topológicas y los diversos tipos de comunicación en ambos sentidos.

Mediante la agrupación de un conjunto de componentes lógicos en distintos niveles, se optimiza el rendimiento y la disponibilidad del sistema, así como aspectos de seguridad. Los niveles son computacionalmente independientes, sin embargo, deben acordarse los protocolos de interacción entre ellos.

2.1.2 Estilos arquitectónicos

En el epígrafe anterior se definieron los patrones de diseño, que se utilizan para resolver problemas en un dominio y que son implementados y/o derivados en estilos arquitectónicos. Los conceptos de estilos y patrones arquitectónicos son similares y no siempre es posible identificar un límite entre ellos. Los estilos están diseñados para capturar conocimiento de diseños específicos para lograr objetivos dentro de un particular contexto de aplicación, mientras que los patrones focalizan el diseño en la resolución de un problema concreto.

Un estilo arquitectónico es el diseño de la aplicación a un alto nivel de abstracción, en concreto, un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer el sistema o subsistema. Una única arquitectura puede contener varios estilos arquitectónicos, y cada estilo puede utilizar varios patrones arquitectónicos.

En la siguiente tabla se muestra algunos ejemplos de estilos arquitectónicos comunes y vigentes, conforme a ciertos criterios basados en el análisis realizado en el artículo publicado (Reynoso CKicillof N, 2004). Muchos de ellos ya han sido definidos dentro del marco de los patrones arquitectónicos.

TABLA 1 ESTILOS ARQUITECTÓNICOS (Reynoso CKicillof N, 2004)

Estilo Arquitectonico	Sub-estilos	Observaciones
Flujo de datos	Tubería-filtros	Implementan transformaciones de datos en pasos sucesivos. **Reutilización** y la **modificabilidad.**
Centrados en Datos	Arquitecturas de Pizarra o Repositorio	Sistemas de acceso y actualización de datos en estructuras de almacenamiento Enfatiza la **integralidad** de los datos.
Estilos de Llamada y Retorno	Model-View-Controller (MVC) Arquitecturas en Capas Arquitecturas Orientadas a Objetos Arquitecturas Basadas en Componentes	Estilos más generalizados en sistemas en gran escala. Enfatiza la **modificabilidad y la escalabilidad**.
Estilos de Código Móvil	Arquitectura de Máquinas Virtuales	Enfatiza la **portabilidad**
Estilos Peer-to-Peer	Arquitecturas Basadas en Eventos (Invocación implícita) Arquitecturas Orientadas a Servicios Arquitecturas Basadas en Recursos	Procesos independientes o entidades que se comunican a través de mensajes. Enfatiza la **modificabilidad** por medio de la separación de las diversas partes que intervienen en la computación.
Estilos heterogéneos	Sistemas de control de procesos Arquitecturas Basadas en Atributos	Difícil categorizar estos estilos en familias de estilos existentes.

2.1.3 Lenguajes de Descripción de Arquitecturas

La descripción de la arquitectura se realiza con la utilización de lenguajes específicos para este propósito (*ADL*: Architecture *Description* Language) con el fin de generar automáticamente la arquitectura en algunas ocasiones. Una de las definiciones más simple, define ADL como una entidad consistente en cuatro "Cs": componentes, conectores, configuraciones y constraints, y una de las definiciones más tempranas sostiene que un ADL debe modelar o soportar los siguientes conceptos:

- ✓ Componentes, que representan los elementos computacionales primarios de un sistema.
- ✓ Conectores, que representan interacciones entre componentes.
- ✓ Composición jerárquica, en la que un componente puede contener una sub-arquitectura completa.
- ✓ Paradigmas de computación, es decir, semánticas, restricciones y propiedades no funcionales.
- ✓ Paradigmas de comunicación.
- ✓ Modelos formales subyacentes.

- ✓ Soporte de herramientas para modelado, análisis, evaluación y verificación.
- ✓ Composición automática de código fuente.

Véase en la siguiente tabla algunos ejemplos ADL, siendo la mayoría definidos en la década de los 90.

TABLA 2 LENGUAJES DESCRIPCION ARQUITECTURA (Reynoso CKicillof N, 2004)

ADL	Fecha	Organismo	Observación
Acme	1995	Monroe & Garlan (CMU), Wile (USC)	Lenguaje de intercambio de ADLs
Aesop	1994	Garlan (CMU)	ADL de propósito general, énfasis en estilo
C2 SADL	1996	Taylor/Medvidovic (UCI)	ADL específico de estilo
Darwin	1991	Magee, Dulay, Eisenbach, Kramer	ADL con énfasis en dinámica
Jacal	1997	Kicillof , Yankelevich (Universidad de Buenos Aires)	ADL - Notación de alto nivel para descripción y prototipado
MetaH	1993	Binns, Englehart (Honeywell)	ADL específico de dominio
Rapide	1990	Luckham (Stanford)	ADL & simulación
UML	1995	Rumbaugh, Jacobson, Booch (Rational)	Lenguaje genérico de modelado – No es ADL
UniCon	1995	Shaw (CMU)	ADL de propósito general, énfasis en conectores y estilos
Wright	1994	Garlan (CMU)	ADL de propósito general, énfasis en comunicación
xADL	2000	Medvidovic, Taylor (UCI, UCLA)	ADL basado en XML

2.2 Microservicios

En los últimos años, muchas empresas están adoptando la arquitectura de microservicios para transformar sus modelos de software existentes. Generalmente, estos sistemas son divididos en múltiples componentes independientes, heterogéneos, que corren en su propia plataforma y además representan una unidad lógica de negocio. Los equipos se organizan alrededor de la funcionalidad, y son capaces de elegir su propia tecnología o lenguaje de programación. Estos equipos son multifuncionales que se ocupan de cada fase del ciclo de vida, incluyendo el desarrollo, las pruebas y el despliegue en entornos de ejecución.

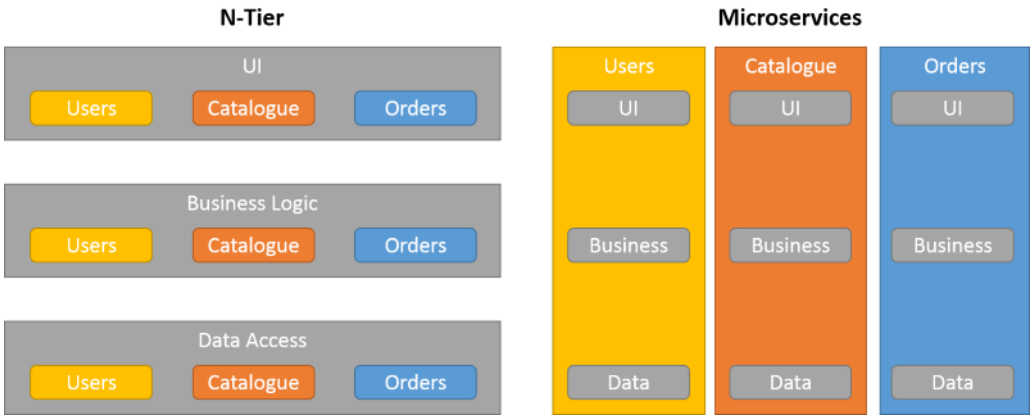


FIGURA 2 MICROSERVICES ARCHITECTURE APPROACH (Juan María Hernandez, 2016)

En la figura anterior se muestra una misma aplicación construida en base a una arquitectura en capas, y con un enfoque en microservicios. El primer diseño divide la aplicación en tres capas distintas: interfaz de usuario, lógica de negocio y datos, incluyendo cada una toda la lógica de negocio del sistema, existiendo unas restricciones topológicas que permiten únicamente la comunicación entre capas adyacentes. El segundo diseño orientado a microservicios, organiza la aplicación según las distintas funcionalidades de negocio, en tres distintos componentes, incluyendo cada uno las distintas capas mencionadas anteriormente.

Las principales ventajas que aporta es la alta escalabilidad, y sobre todo la independencia de las plataformas e infraestructuras tecnológicas, que facilita el desarrollo y el despliegue. Además, se favorece una descentralización en la organización, así como en la gestión de los datos. Por otro lado, los microservicios son desplegados y escalados en entornos distribuidos mediante replicación, de forma que el servicio es garantizado por fallos de sobrecarga o caída de algún componente, lo que aumenta la resiliencia.

2.2.1 Definición/Principios

Según Martin Fowler y James Lewis, en su artículo “Microservices” (Fowler & Lewis, 2014) lo definen como un estilo arquitectónico, es decir, una forma de desarrollar una aplicación, basada en un conjunto de pequeños servicios, cada uno corriendo en sus propios procesos y comunicándose mediante mecanismos ligeros, generalmente un recurso API de HTTP. Estos servicios están construidos alrededor de las capacidades de negocio, y son desplegados por herramientas automáticas de forma independiente. Además, hay una mínima gestión centralizada de servicios, y pueden ser escritos en diferentes lenguajes de programación y usar diferentes tecnologías de almacenamiento.

A continuación se muestran las características o principios de las dos definiciones más populares del término microservicios y las similitudes entre ellas (Zimmermann, 2017).

TABLA 3 COMPARISON OF DEFINITIONS MICROSERVICES

Characteristics by Lewis/Fowler	**Newman's Principles**
1. Componentization via services (running in own process and communicating with lightweight mechanisms)	1. Hide internal implementation details
2. Organized around business capabilities	2. Model around business concepts
3. Products not projects	
4. Smart endpoints and dumb pipes	3. Decentralize all the things
5. Decentralized governance (enabling polyglot programming)	
6. Decentralized data management (and polyglot persistence)	
7. Infrastructure automation (and decentralized management)	4. Adopt a culture of automation
	5. Independently deployable
8. Design for failure	6. Isolate failure
9. Evolutionary design	
	7. Highly observable

2.2.2 Atributos de calidad

Entiéndase los atributos de calidad de un sistema software como las propiedades o cualidades de calidad que la aplicación debe satisfacer. Existe una categorización de dichos atributos en el standard ISO/IEC FCD 25010: Systems and software engineering—Systems and software product Quality Requirements and Evaluation (SQuaRE)—System and software quality models, (Bass et al., 2012), que agrupa en las siguientes categorías principales: funcionalidad, compatibilidad, fiabilidad, usabilidad, eficiencia, seguridad, mantenibilidad y portabilidad.

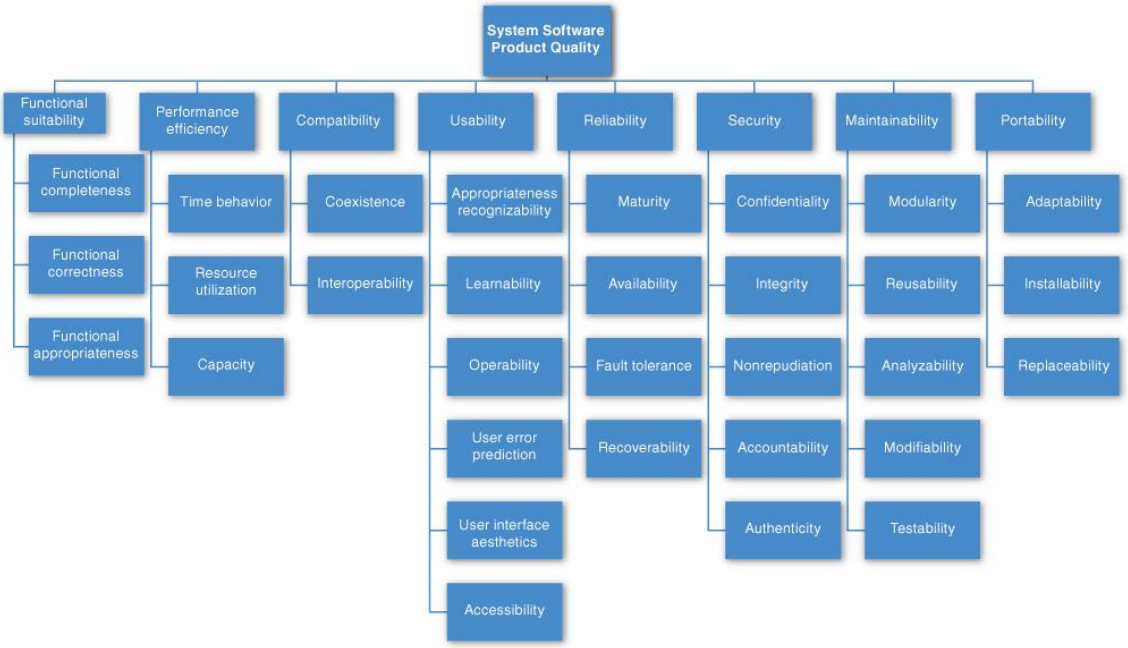


FIGURA 3 ISO/IEC FCD 25010 PRODUCT QUALITY STANDARD

Entre muchos atributos, se ha seleccionado aquellos que valoran de forma más precisa las cualidades de un sistema basado en microservicios.

- **Interoperabilidad (compatibilidad)**: Mide la facilidad de un sistema de intercambio de datos o servicios con otros sistemas internos y/o externos, para un funcionamiento correcto global.
  Uno de los puntos críticos en microservicios, que son implementados en distintas plataformas y lenguajes de programación, es la interoperabilidad. Debido a la estandarización del HTTP, los mecanismos de conexión entre ellos son definidos mediante RESTful interfaces o Internet APIs, garantizado que puedan intercambiar datos de una manera fácil, solo conociendo la interfaz ligado a cada servicio.

- **Fiabilidad:** Capacidad de un software de mantenerse operativo y funcionando en condiciones normales.
  La fiabilidad en microservicios es mayor puesto que el sistema está dividido en componentes independientes y aislados. Por esta razón, resulta sencillo solucionar un fallo en un microservicio sin afectar al resto del sistema. Al contrario que en aplicaciones monolíticas, donde al caer un servicio falla el sistema entero.

- **Disponibilidad**: Define la proporción de tiempo en que el sistema está disponible. Una nueva entrega software de un microservicio en un sistema en producción requiere menos tiempo que la actualización del aplicativo entero. De esta forma se favorece la disponibilidad, al reducirse los tiempos de inactividad.

- **Tolerancia a fallos***:* Es la propiedad que le permite a un sistema seguir funcionando correctamente en caso de fallo de uno o varios de sus componentes. Las aplicaciones complejas que tiene microservicios, están diseñadas para ejecutarse en sistemas distribuidos altamente disponibles. En su definición, un

sistema distribuido se ejecuta en distintas máquinas y/o particiones de disco, conectados por líneas de comunicación, lo que conlleva a riesgos en fallos hardware, software y caídas en la red, que deben ser solucionados mediante replicación, redundancia y otras técnicas para mantener un estado consistente del sistema global.

- **Flexibilidad:** Está relacionada con la habilidad del sistema para ser flexible frente a cambios, facilidad de reconfigurar y adaptarse a los nuevos requerimientos del sistema y del negocio.
  Debido a la división de la lógica de negocios en componentes o servicios independientes, se adquiere mayor flexibilidad frente a cambios en la lógica de negocio.

- **Mantenibilidad:** Habilidad de un sistema para permitir cambios en sus componentes o servicios para corregir errores, cambiar funcionalidad y/o satisfacer nuevos requerimientos a nivel de la lógica de negocio.
  Debido al desacoplamiento entre componentes es sencillo modificar un servicio, sin alterar el resto de la infraestructura. En los sistemas tradicionales, las modificaciones requieren de un mayor esfuerzo y coordinación entre los distintos equipos, exigiendo adicionalmente pruebas globales del sistema.

- **Portabilidad (Reusabilidad)**: Mide la capacidad del sistema para ser transferido a otro entorno, y para coexistir con otras aplicaciones compartiendo recursos comunes.
  Los microservicios son empaquetados de una manera estandarizada, permite introducir las dependencias y que se ejecute del mismo modo en cualquier infraestructura.

- **Escalabilidad:** Define la capacidad del sistema para crecer sin aumentar su complejidad ni disminuir su rendimiento.
  Quizás sea la característica más citada sobre esta arquitectura, la posibilidad de despliegue y escalado de los distintos servicios mediante múltiples instancias en contenedores y/o cambio de servidores. De esta forma, se consigue una alta escalabilidad horizontal para atender la demanda creciente/decreciente, siendo solo necesario dimensionar el servicio requerido sin tener que escalar la aplicación en su totalidad.

2.2.3 Conceptos y tecnologías asociadas

Entre los nuevos conceptos que están ligados al término microservicios, cabe destacar la tecnología de contenedores y la metodología DevOps. El despliegue y puesta en operación de cada microservicio se realiza mediante contenedores que facilitan la portabilidad, la independencia y la escalabilidad. También nacen nuevas metodologías que se ajustan a las características intrínsecas de una aplicación basada en microservicios. Por ejemplo, el termino DevOps, que consituye un nuevo método más adecuado para el desarrollo y el despliegue de los microservicios, de forma ágil y eficiente.

- **CONTENEDORES**

Una definición de un contenedor (entre muchas que existen) es la imagen de un sistema de ficheros diseñada para empaquetar una aplicación y ejecutarse en un sistema operativo compartido como un proceso. Por lo tanto, el contenedor permite empaquetar software de una manera estandarizada incluyendo todas las dependencias para ejecutarse en cualquier infraestructura favoreciendo notablemente la portabilidad y el escalado, creando contenedores según la demanda del negocio. Una forma sencilla y rápida de desarrollar, empaquetar y ejecutar aplicaciones en cualquier entorno.

Los contenedores tienen ciertas similitudes con respecto a las máquinas virtuales, ambas proporcionan aislamiento y una capa intermedia que posibilita la ejecución en un equipo anfitrión. Lo que constituye el punto fuerte en la tecnología de contenedores, es que a diferencia de las máquinas virtuales donde cada una ejecuta un sistema operativo, los contenedores comparten sistema operativo entre ellos, por lo que sólo se consumen los recursos realmente necesarios dando rapidez y ligereza al sistema. Por lo tanto, las máquinas virtuales comparten recursos hardware y los contenedores el mismo sistema operativo (procesos en la misma máquina).

Aunque favorece el despliegue y el tiempo de ejecución de aplicaciones tanto en desarrollo como en producción, existen ciertos inconvenientes aparejados al uso de contenedores. Por un lado, hay aspectos de seguridad que deben ser vigilados, ya que comparten los recursos con otros procesos del sistema operativo, por lo que no ofrece total aislamiento, y por otro, limita que la aplicación solo pueda correr en un solo sistema operativo. Es una solución idónea, para ejecutar múltiples copias de una sola aplicación, y una solución menos apropiada cuando se ejecutan múltiples aplicaciones en distintos sistemas operativos.

La tecnología de contenedores se basa principalmente en dos características del Kernel Linux avanzado, cgroups, para la gestión y asignación de recursos a un grupo de procesos y a los namespaces, que permiten el aislamiento y virtualización de dichos recursos.

Los microservicios se convierten en unidades de despliegue pequeñas, lo que facilita la encapsulación en contenedores aprovechando todas las ventajas de dicha tecnología, convirtiéndose en un sistema ágil para escalado de microservicios.

- **DEV&OPS**

Es una metodología de desarrollo ágil que permite adaptarse a los cambios desde el negocio mediante la entrega continua de versiones software, de mejor calidad y con menor coste, basándose en la automatización de las tareas de desarrollo (Development) y de sistemas (Operations).

Uno de los requisitos para que DevOps funcione en una organización es la integración entre el área de desarrollo y de sistemas, donde el nuevo perfil ingeniero DevOps, es un programador con conocimientos de administrador de sistemas, que le permita desplegar en producción el código generado de forma ágil, eficiente y flexible a través del conjunto de herramientas disponibles. Realmente, el desarrollador se convierte en el dueño del servicio, puesto que no solo lo implementa, sino que debe integrarlo y desplegarlo en una infraestructura, inclusive ponerlo en operación. La instauración de esta tecnología, provoca un cambio en la organización, y puede llegar incluso a prescindir de los equipos de infraestructuras y operaciones.

Normalmente las organizaciones que adoptan una arquitectura en microservicios, utilizan la tecnología DevOps, para aportar mayor flexibilidad, eficiencia y escalabilidad a los distintos servicios. Para que los servicios sean desplegados de una forma eficiente, es esencial la incorporación de la automatización y metodología DevOps.

Así mismo, aquellos que usan la citada metodología, se benefician de todas las ventajas de los microservicios, tanto en el desarrollo como en las operaciones. Los servicios son unidades que se manejan de forma aislada, lo que permite mayor fiabilidad en la implementación, pruebas y puesta en producción del software, resultando ágil su mantenimiento y despliegue por parte de un ingeniero de desarrollo y sistemas (nuevo concepto ingeniero) que utiliza dicha metodología.

2.2.4 Ventajas e Inconvenientes

La implantación de una arquitectura de microservicios en empresas como Netflix, Amazon y Google, les ha repercutido de una manera muy positiva y ha facilitado la escalabilidad y dimensionamiento de su negocio a nivel internacional. Sin embargo, la solución de migrar de sistemas monolíticos a una de microservicios, se ve frenada por la complejidad que presentan algunas funcionalidades y por ciertos inconvenientes a tener en cuenta, por lo que no se adapta a cualquier proyecto y organización.

Las ventajas más comunes por las cuales las empresas adoptan la arquitectura en microservicios, son la escalabilidad y la mantenibilidad, aunque también se valora la reducción de tiempo tanto en desarrollo como en despliegue, pruebas y operación, facilitando las distintas actualizaciones de software. Por otro lado, existe una independencia en plataformas y lenguajes de programación, ofreciendo la posibilidad de elegir entre distintas tecnologías que puedan ajustarse convenientemente a cada servicio.

Las capacidades asociadas pueden estar categorizadas de la siguiente manera:

- Capacidad de despliegue: Una arquitectura en microservicios ofrece la capacidad de implantación y despliegue de nuevas versiones software de forma automática, flexible y ágil, disminuyendo los tiempos de los ciclos en distintas plataformas. El despliegue puede ser realizado mediante asignación de cada servicio en la tecnología de contenedores, sin compartir recursos con otros componentes, tales como bases de datos (descentralización del almacenamiento), caché, etc.

- Capacidad en desarrollo: Los microservicios pueden ser desarrollados en un equipo pequeño, que facilita la comprensión y simplifica la gestión, mejorando de esta forma la productividad de nuevos miembros que se incorporan al equipo. Permite combinar diferentes lenguajes y tecnologías para poder implementar el desarrollo de la forma más conveniente, dando la libertad al desarrollador de utilizar distintos lenguajes de programación para ajustarse adecuadamente a las características del propio servicio.

- Capacidad en negocio: El código es organizado alrededor de la lógica de negocio, habiendo una separación de conceptos que son asociados a distintos componentes, los cuales, se implementan, despliegan y se prueban de forma independiente. De esta forma, se pretende paliar el problema de eficiencia en sistemas tradicionales IT, que son estables y robustos, pero no demasiado ágiles para adaptarse con rapidez a las necesidades cambiantes del negocio, en un marco cada vez más

competitivo.

- Capacidad en operación: Frente a los sistemas monolíticos, si un cambio afecta a un parte de la aplicación no es necesario modificar ni desplegar la aplicación entera. Permite mejor aislamiento a los fallos, el mantenimiento y la integración con otros servicios. Los servicios pueden ser escalados individualmente corriendo múltiples instancias, o incluso pueden ser movidos a otro servidor, sin tener impacto en los clientes. Por otro lado, evita el compromiso con las tecnologías presentes.

Si bien el uso de microservicios facilita cambiar, implementar y desplegar un único servicio rápidamente, también hace que el esfuerzo global de gestión y mantenimiento de un conjunto de servicios sea mayor de lo que sería en una aplicación monolítica correspondiente, y presenta algunos inconvenientes:

- Complejidad de gestión: Si el número de servicios aumenta, mayor número de nodos que mantener, por lo que la integración, las pruebas y la gestión del producto completo es más complicado. Por lo tanto, se requiere de una infraestructura para el independiente escalado de los servicios, que soporten tolerancia a fallos, y continua entrega. Esto puede incluir el uso de virtualización e infraestructuras cloud.

- Coordinación/Integración: Numerosos servicios que se desarrollan y operan de forma independiente deben ser integrados. Aunque cada microservicio pueda ser simple, la integración es mucho más complicada, por las dependencias y llamadas que existan entre ellos. Por otra parte, y con el fin de ofrecer un servicio común, deben definirse mecanismos de coordinación y control de las distintas versiones.

- Dependencia/Comunicación entre servicios: Las dependencias entre servicios necesitan una infraestructura de comunicación adecuada, ya que las llamadas a funciones locales (en otro tipo de sistemas), se convierten en llamadas remotas a otros servicios lo que afecta al rendimiento del sistema.

- Dimensionamiento servicios (*service tailoring*): Dificultad para determinar el tamaño y/o nivel de granularidad apropiado de los servicios para facilitar un diseño que se ajuste a la lógica de negocio, y al mismo tiempo, tener las menos dependencias posibles. Servicios demasiado detallados (*fine-grained*), aumenta el número de comunicaciones entre ellos y, por el contrario, servicios menos detallados (*course-grained*), son menos escalables, modificables, etc.

Capítulo 3. Revisión Sistemática de Literatura: Arquitectura en Microservicios

Este capítulo describe la parte central del trabajo de fin de máster y presenta la revisión sistemática de la literatura como un proceso sistemático de investigación.

3.1 Primera Fase: Planificación de la Revisión

La fase de planificación consta de la identificación del objetivo de la revisión y la definición de un protocolo de revisión. El protocolo de revisión especifica el proceso de búsqueda y análisis, reduciendo la posibilidad de que exista una revisión impulsada por las expectativas de la investigación.

3.1.1 Objetivo y preguntas de investigación

El objetivo del trabajo de investigación es conocer si existen publicaciones que traten sobre un estilo arquitectónico sobre microservicios en las diferentes bases de datos científicas. Previo a la búsqueda de estudios primarios, se ha intentado localizar tanto *Mapping study* como *Systematic Literature Review* (SLR), sobre el tema en cuestión, y se ha considerado que no es necesario un "*pre-review mapping study*" sobre el dominio que nos ocupa debido al porcentaje bajo de SMS localizados (en total 4).

Para ello se han definido las siguientes preguntas de investigación (*research questions*):

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

La siguiente figura muestra como el término arquitectura en microservicios ha ganado en popularidad desde el 2014.

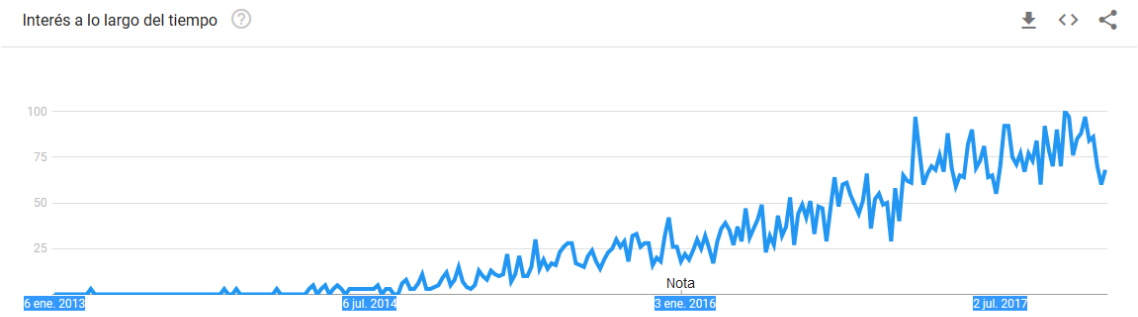


Figura 4 Búsqueda en Google a través del tiempo 2013–2018 del término Arquitectura en Microservicios. Fuente: Google Trends

3.1.2 Estrategia de búsqueda

La estrategia de búsqueda debe definir en primer lugar los motores y bases de datos de búsqueda. Han sido seleccionadas las siguientes bases de datos electrónicas (en orden de importancia) donde se realiza la búsqueda de los estudios primarios.

- **WOS (Web of Knowledge):** Es una base de datos genérica que normalmente incluye todas las revistas con impacto JCR *(Journal Citation Report)* independientemente de la editorial, además de conferencias cuyos *proceedings* se publican en las editoriales más importantes (IEEE, ACM, Springer, Elsevier, ...). El factor de impacto JCR permite determinar la importancia relativa de las mismas dentro de sus categorías temáticas. No muestra el texto completo del artículo, pero si el resumen, y otros datos útiles. URL: https://www.recursoscientificos.fecyt.es/ (redirige al http://wos.fecyt.es).

- **IEEE (Xplore Digital Library):** Es una base de datos de investigación académica que proporciona acceso al texto completo de artículos y trabajos sobre Ciencias de la Computación, Ingeniería Eléctrica y Electrónica. URL: http://ieeexplore.ieee.org

- **ACM DL**: Contiene revistas de relevancia mundial para las ciencias de la computación y tecnología de la información. Incluye la colección completa de publicaciones de la "*Association for Computing Machinery*" (*ACM*). URL: http://dl.acm.org

- **Springer**: Es una editorial global que publica libros, libros electrónicos y publicaciones científicas. SpringerLink, es una de las bases de datos científicas que ofrece a través Internet. URL: https://link.springer.com/

- **Science Direct**: Buscador de artículos científicos de la editorial Elsevier. URL: http://www.sciencedirect.com

- **Scopus**: Base de datos bibliográfica de resúmenes y citas de artículos de revistas científicas (también de la editorial Elsevier). URL: https://www.recursoscientificos.fecyt.es/. Para el acceso hay que entrar por FECYT (al igual que WOS) donde solicita login. Se debe seleccionar "*Other Institution Login"* y en el formulario introducir "*región -> View all*" y seleccionar "*FECYT"* y a continuación "*MADRID CONSORTIUM-Politecnic University of Madrid - Fecyt A/C Mbr#79, Library"*. Para extraer información a través de "*Document Download Manager*", con una versión Java con restricciones de seguridad mayor, debes añadir una excepción al siguiente programa "http://scddm.quosavl.com/ddm/scopus.jsp" para que permita su ejecución

Y las siguientes conferencias de interés, que tuvieron lugar a partir del 2014:

- IEEE SERVICES

- IEEE International Conference on Software Architecture(ICSA)

- European Conference on Software Architectures (ECSA)
- IEEE Software Special Issue on Microservices
- IEEE Software Special Issue on Release Engineering 3.0
- European Conference on Service-Oriented and Cloud Computing (ESOCC)
- First International Workshop on Microservices for Agile software development (WMSA17)
- International Workshop on Architecting with MicroServices (AMS)
- International Workshop on Quality-Aware DevOps

En segundo lugar, se definen los términos de búsqueda, para contestar a las preguntas de investigación:

1º Búsqueda de SLR & SMS *(Systematic Literature Review & Systematic Mapping Study*)

Los artículos científicos relevantes para responder las preguntas de investigación obtenidos a partir de la búsqueda en las bases de datos definidas se denominan *estudios primarios*. Previamente a la localización de estudios primarios, se debe buscar sobre posibles *estudios secundarios* (*Systematic Literature Review y Systematic Mapping Study*) posteriores al 2014 sobre la arquitectura de microservicios ya existentes.

En el caso de búsqueda de estudios secundarios, se eligen los patrones necesarios para la búsqueda, como se muestra a continuación (de la misma forma se ha realizado para SMS):

- Microservices AND (architecture style OR architectural style) AND (Systematic Literature Review)
- Microservices AND (architecture style OR architectural style) AND SLR
- Microservices AND (reference architecture) AND SLR

2º - Búsqueda de estudios primarios

Con la finalidad de no dejar ningún artículo publicado después del 2014, se definen los siguientes patrones, con la utilización de "*wildcards"* que sustituyen caracteres para abarcar más estudios, y operadores lógicos, para poder localizar aquellos que casan con los siguientes patrones:

- Microservices AND (architecture style OR architectural style)
- Microservices AND (architecture style OR architectural style)
- Microservices AND (reference architecture)

Todos los artículos seleccionados deben ser posteriores al año de publicación del artículo "Microservices" (Fowler & Lewis, 2014) hasta la actualidad. En las webs científicas que no aparecen la opción de filtrado por año, se descartan manualmente aquellos con fecha de publicación anterior a 2014.

3.1.3 Criterios de inclusión y exclusión

Los criterios de inclusión (CI) y los criterios de exclusión (CE) determinan la disposición de los estudios primarios para ser considerados o no parte de nuestro estudio.

A través de los criterios de inclusión y exclusión definidos se ha seleccionado un conjunto de estudios primarios descartando aquellos que no responden a nuestras cuestiones de investigación. Estos criterios se aplican sobre el título, "abstract" y conclusiones de los artículos seleccionados en la fase de búsqueda según los patrones ya dados.

Criterios de Inclusión (IC)

- **CI1:** SLR's & Mapping Studies sobre microservicios
- **CI2:** Estudios que traten de manera explícita de estilo arquitectónico basado/orientado a microservicios.
- **CI3**: Estudios que traten los microservicios desde el punto de vista arquitectónico (aunque no se mencione estilo).
- **CI4:** El material científico se ha publicado hasta la actualidad desde 2014.

Criterios de Exclusión (EC)

- **CE1:** Artículos que solo hablen de microservicios sin mencionar aspectos arquitectónicos.
- **CE2:** Estudios que sean casos de uso/aplicación del estilo arquitectónico de microservicios.
- **CE3:** Trabajos que traten sobre atributos de calidad del estilo arquitectónico de microservicios.

Cabe destacar que estos criterios finales de inclusión/exclusion han sido establecidos durante el proceso de clasificación de los estudios primarios por parte de dos observadores (tutores del proyecto). Es un proceso dinamico que es provocado por algún desacuerdo en la valoración de los estudios primarios, que conlleva a la redefinición de los criterios de inclusión/exclusión.

3.1.4 Estrategia de extracción y síntesis de datos

Durante la lectura reiterada de los estudios primarios seleccionados, se realiza una estrategia de extracción de datos mediante un resumen y una conclusión que responde a las preguntas de investigación, más los aspectos arquitectónicos que son tratados en cada artículo (véase Anexo C).

No se considera necesario un meta análisis puesto que la información a analizar es cualitativa y no cuantitativa, siendo importante detectar si los resultados de los estudios son consistentes entre sí (homogéneos) o inconsistentes (heterogéneos).

Si existen propuestas de estilos arquitectónicos orientados/basados en microservicios, se debería contrastar la bondad de cada una de las propuestas con un estándar de descripción de estilos arquitectónicos (por ejemplo: Shaw, IEEE). Si no hay ninguna propuesta, en el sentido indicado anteriormente, se sintetiza aquellas aportaciones que tienen más relación con el concepto de arquitectura, mediante un formulario a rellenar con preguntas sobre el tema en cuestión (véase anexo C). Adicionalmente, se diseña un esquema gráfico a través de mapas conceptuales por cada artículo, con el fin de observar patrones comunes arquitectónicos entre ellos. Más tarde, se agrupa toda la información extraída en base a unos temas comunes, para establecer los hallazgos y desafíos actuales.

3.2 Segunda Fase: Conducción de la Revisión

3.2.1 Búsqueda de los estudios primarios

Durante la búsqueda de los estudios primarios, se debe resaltar las peculiaridades por cada base de datos científica o web utilizada. La siguiente tabla muestra las diferencias que existe entre los motores de búsqueda, siendo necesario seleccionar los campos de búsqueda y los operadores lógicos que se utilizan para la concatenación de palabras clave en una búsqueda avanzada. Esta selección permite por un lado recoger la mayor cantidad de artículos sobre nuestro estudio, y por otro descartar los que no se ciñen a nuestros criterios o claves ya definidas.

TABLA 4 CAMPOS Y CLAVES DE BÚSQUEDA

BBDD	Campos Búsqueda	Operadores Logicos	Comentarios
JCR Science WoS	**Tema:** ("micro*service* architect* style") **Tema**: (micro*service*) AND **Tema:** ("architect* style") **Tema:** (micro*service*) AND **Tema**: ("reference architect*")	Las cadenas encerradas entre "" deben buscarse tal cual y la operación AND implica seleccionar solo aquellos trabajos en que aparezcan los dos strings	El plugin de Mendeley no funciona bien y hay que subir los PDF's manualmente o incluso buscarlos en las otras webs o google.
IEEExplore	**Metadata:** (((micro*service*) AND architect*) AND style) **Metadata:** (((micro*service*) AND reference) AND architect*)	Las cadenas encerradas entre () deben buscarse tal cual y la operación AND implica seleccionar solo aquellos trabajos en que aparezcan los dos strings	Mismos resultados si aplicas directamente (micro*service* architect* style)
ACM Digital library	**recordAbstract**:(+"microservice*" +"architect*" + style) **recordAbstract**:(+"micro*service*" + "reference" + "architect*")	Las cadenas encerradas entre "" deben buscarse tal cual y la operación + implica seleccionar solo aquellos trabajos en que aparezcan los dos strings	
Springer	micro*service* AND "architect* style" micro*service* AND "reference architect*"	Las cadenas encerradas entre "" deben buscarse tal cual y la operación AND implica seleccionar solo aquellos trabajos en que aparezcan los dos strings	Muchos resultados en la búsqueda no filtra por tema, solo por título y autor.
Science Direct	**Abstract + Title + Keywords:** "micro*service*" AND "architect* style" **Abstract + Title + Keywords:** "micro*service*" AND "reference architect*" **Year:** 2014 to Present	Las cadenas encerradas entre "" deben buscarse tal cual y la operación AND implica seleccionar solo aquellos trabajos en que aparezcan los dos strings. Además, permite en el motor búsqueda avanzada especificar el año de publicación.	
Scopus	**Article title, Abstract, Keywords:** "micro*service*" AND "architect* style" **Abstract + Title + Keywords:** "micro*service*" AND "reference architect*" **Year:** 2014 to Present	Las cadenas encerradas entre "" deben buscarse tal cual y la operación AND implica seleccionar solo aquellos trabajos en que aparezcan los dos strings	

En algunas ocasiones, el texto completo de los artículos seleccionados ha sido solicitado directamente a los autores a través de ResearchGate (previo registro de usuario), debido a que no es accesible o no es de libre distribución. Es una red social académica dirigida a la comunidad investigadora. El archivo digital de la UPM también ha constituido otro recurso para la búsqueda de los artículos completos.

En relación a las conferencias y workshops acontecidas desde el año 2014, se ha utilizado buscadores más generales, siendo más laxos en los criterios (generalmente buscando por microservices y/o architecture), debido a la ausencia de motores de búsqueda más refinados para filtrar los resultados. En la tabla siguiente se muestra el nº de artículos

encontrados en total por cada conferencia y/o workshop, que mencionan la arquitectura en microservicios. Muchos de ellos, ya habían sido referenciados en las anteriores bases de datos científicas, apareciendo otros nuevos de nuestro interés.

Entre las conferencias seleccionadas para la búsqueda de artículos, además de las elegidas en la primera fase de la planificación, se han incluido algunas de especial interés publicadas en IEEE (*Conference Publications*), más en ACM DL (a través de link https://dl.acm.org/icps.cfm), y Springer. También ha sido útil el listado de artículos por conferencia en la siguiente base de datos: http://dblp.org/db/conf/esocc/.

TABLA 5 NUMERO ARTÍCULOS EN CONFERENCIAS

Conference	Year	Numero artículos
IEEE International Conference on Software Architecture(ICSA)	2017 IEEE International Conference on Software Architecture (ICSA)	**3**
European Conference on Software Architectures (ECSA)	11th European Conference, ECSA 2017, Canterbury, UK, September 11-15, 2017	**3**
European Conference on Software Architecture Workshops (ECSAW)	ECSAW '16:Proccedings of the 10th European Conference on Software Architecture Workshops	**1**
IEEE Software Special Issue on Microservices	En el momento búsqueda no hay publicado "Paper Proceedings"	
IEEE Software Special Issue on Release Engineering 3.0	2017 IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS)	**0**
European Conference on Service-Oriented and Cloud Computing (ESOCC)	Service-Oriented and Cloud Computing - 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017 Service Oriented and Cloud Computing - 4th European Conference, ESOCC 2015, Taormina, Italy, September 15-17, 2015. Proceedings	**2** **1**
XP2017 (Scientific Workshops)	First International Workshop on Microservices for Agile software development (WMSA17)	**1**
AMS 2017	International Workshop on Architecting with MicroServices (AMS)	**5**
International Workshop on Quality-Aware DevOps	2nd International Workshop on Quality-Aware DevOps, QUDOS@ISSTA 2016, Saarbrücken, Germany, July 21, 2016	**0**
ECASE	2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering	**1**
FAS*W	2nd IEEE International Workshops on Foundations and Applications of Self* Systems, FAS*W@SASO/ICCAC 2017, Tucson, AZ, USA, September 18-22, 2017	**1**
SOSE (Service-Oriented System Engineering)	2017 IEEE Symposium on Service-Oriented System Engineering, SOSE 2017, San Francisco, CA, USA, April 6-9, 2017 2016 IEEE Symposium on Service-Oriented System Engineering, SOSE 2016, Oxford, United Kingdom, March 29 - April 2, 2016	**1** **2** **1**

Conference	Year	Numero artículos
	2015 IEEE Symposium on Service-Oriented System Engineering, SOSE 2015, San Francisco Bay, CA, USA, March 30 - April 3, 2015	
IISWC	2016 IEEE International Symposium on Workload Characterization, IISWC 2016, Providence, RI, USA, September 25-27, 2016	**0**
ICSSP	2016 International Conference on Software and Systems Process, ICSSP 2016, Austin, Texas, USA, May 14-15, 2016	**1**
Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)	2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA))	**0**
Big Data and Cloud Computing (BdCloud), IEEE International, Conference	2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)	**1**

Siendo el tema de investigación muy actual, cabe la posibilidad que el número de estudios primarios crezca en posteriores búsquedas, debido a la incorporación de nuevos artículos relacionados con el tema a partir del 2018.

En la Fgura 5 se muestran los artículos encontrados en las distintas conferencias organizadas por años consecutivos, que tienen relación con el tema de investigación (véase la tabla 5).

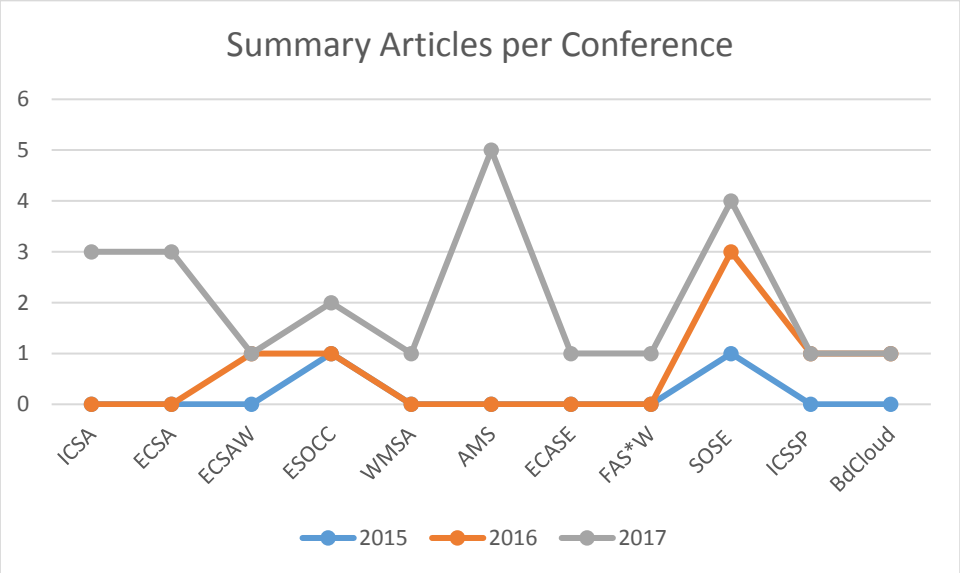


FIGURA 5 ARTÍCULOS ENCONTRADOS EN CONFERENCIAS Y/O WORKSHOPS

3.2.2 Selección de los estudios primarios

Tras la búsqueda sistematica en las bases de datos científicas se obtienen 93 artículos, que son nombrados mediante un identificador (desde P01 hasta P93). En una primera selección, se analiza el título, el resumen y las conclusiones de cada artículo, anotándose en un fichero las razones y/o criterios para su inclusión/exclusión. Los trabajos para cuya exclusión las razones sean obvias, no se documentan extensivamente. Una vez seleccionados se comparan resultados de dos personas (en este caso los tutores del proyecto), y si existe desacuerdo se obtiene la Kappa de Cohen. El método de la Kappa de Cohen mide la concordancia entre dos examinadores en sus correspondientes clasificaciones, y obtiene un coeficiente que determina la inclusión o exclusión de los artículos seleccionados en el trabajo actual (véase anexo A).

Durante este proceso, se van modificando dichos criterios de inclusión/exclusión, con el objetivo de que sean los mas apropiados posibles para nuestro estudio de investigación (véase sección 3.1.3). En resumen, se incluyen aquellos estudios que tratan sobre los microservicios desde el punto de vista arquitectónico y se excluyen aquellos que solo habla de microservicios sin mencionar aspectos arquitectónicos. Los artículos que tratan de manera explícita el estilo arquitectónico basado/orientado a microservicios se mencionan especialmente.

Todo el proceso sobre la aplicación de la Kappa Cohen, así como la evolución de los criterios inclusión y exclusion se muestra en el apéndice C. También, se enumeran todos los estudios primarios analizados y el resultado de tal análisis por cada uno de los investigadores.

Adicionalmente, se utiliza Mendeley (Figura 6) como herramienta para almacenamiento y organización los estudios primarios seleccionados creándose la siguiente estructura de directorios:

- microservices architecture style SLR
- primary studies for SLR
  - included /trabajos primarios incluidos
  - excluded /trabajos primarios excluidos
  - candidates /trabajos del resultado de la búsqueda según los patrones
- mapping studies /mapping que ya existen
- SLR studies /SLR que ya existen
- methodology /metodología para mappings y SLR (Kappa Cohen), libros digitales

Mendeley es un gestor de referencias bibliográficas con plugins para Internet y Microsoft Word.

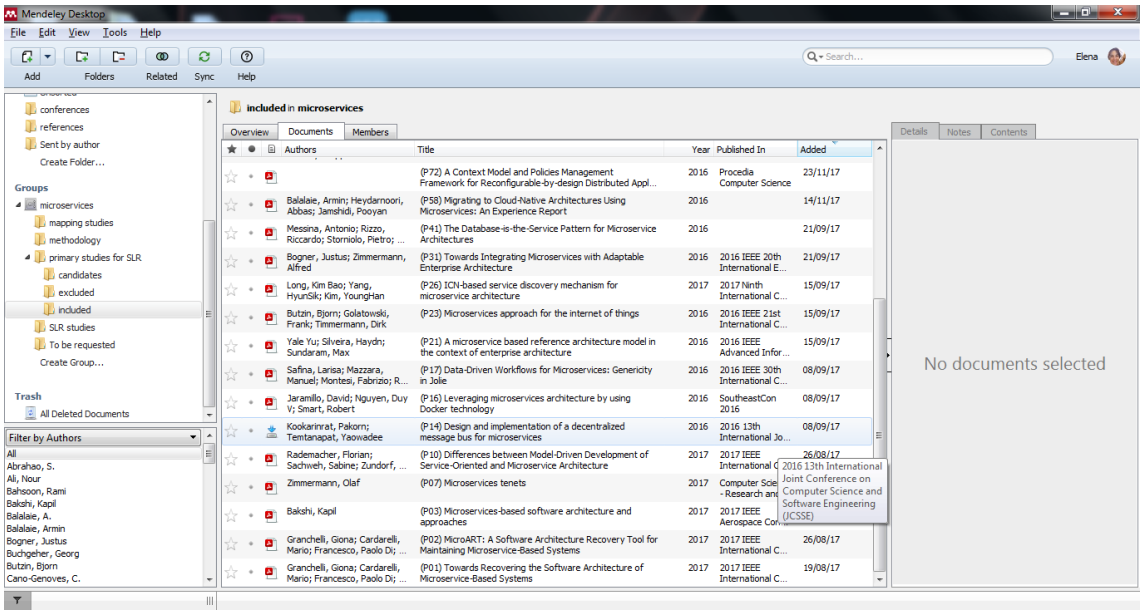


FIGURA 6 GESTIÓN DE LOS ARTÍCULOS CIENTÍFICOS MEDIANTE LA HERRAMIENTA MENDELEY

Una vez aplicados los criterios inclusión/exclusión quedan 21 artículos, que sumados a en 2 articulos nuevos encontrados en la búsqueda por Internet de las conferencias relacionadas, son 23 artículos.

Durante la extracción de datos, se comprueba que dos de los artículos ya seleccionados son "*short papers*" o similares a otros ya analizados. No obstante, se realiza la síntesis y extracción de datos con el objetivo de encontrar alguna contribución adicional no mencionada que pueda ser relevante.

Finalmente, el total de estudios primarios son 21 (publicados en los años 2016 y 2017), como se muestra en la siguiente figura:

TABLA 6 DESCRIPCIÓN ARTÍCULOS SELECCIONADOS

ID	Titulo	Tipo	Año	Contribución
P01	Towards Recovering the Software Architecture of Microservice-Based Systems	Conference Proceedings	2017	Software Recovery Tool for Microservices based systems
P03	Microservices-based software architecture and approaches	Conference Proceedings	2017	Several aspects of microservices-based architecture, including several potential use cases for the aerospace industry
P07	Microservices tenets	Journal Article	2017	Seven microservices tenets (similar to the ones defined by Lewis/Fowler & Newman)
P10	Differences between Model-Driven Development of Service-Oriented and Microservice Architecture	Conference Proceedings	2017	Differences between SOA and MSA, as well as implications in certain areas of MDD to consider when adapting service-oriented MDD to MSA
P14	Design and implementation of a decentralized message bus for microservices	Conference Proceedings	2016	Decentralized message bus to use as a communication tool between microservices

ID	Titulo	Tipo	Año	Contribución
P16	Leveraging microservices architecture by using Docker technology	Conference Proceedings	2016	Maximizing the use of Docker for leveraging mircoservices architecture
P17	Data-Driven Workflows for Microservices: Genericity in Jolie	Conference Proceedings	2016	Extension of Jolie, which is a programming language based on the microservices paradigm, for handling message types and data manipulation
P21	A microservice based reference architecture model in the context of enterprise architecture	Conference Proceedings	2016	Architecture model using building blocks for implementing and managing enterprise microservices in the context of enterprise architecture
P23	Microservices approach for the internet of things	Conference Proceedings	2016	Patterns and best practices for the microservices approach and how they use in IoT
P26	ICN-based service discovery mechanism for microservice architecture	Conference Proceedings	2017	Information-centric service discovery (ISD) flexible mechanism which applies the information-centric networking (ICN) concept
P31	Towards Integrating Microservices with Adaptable Enterprise Architecture	Conference Proceedings	2016	EA-Mini-Descriptions, a flexible architectural metamodel by extending enterprise architecture reference models
P41	The Database-is-the-Service Pattern for Microservice Architectures	Book Section	2016	Pattern for microservice architecture that uses a database as component
P58	Migrating to Cloud-Native Architectures Using Microservices: An Experience Report	Book Section	2016	Migrating a monolithic on-premise software architecture to microservices in cloud environment
P72	A Context Model and Policies Management Framework for Reconfigurable-by-design Distributed Applications	Journal Article	2016	ARCADIA project, a microservices-based software development paradigm and orchestration framework for reconfigurable-by-design distributed applications
P77	Incremental integration of microservices in cloud applications	Conference Proceedings	2016	Incremental integration of microservices with deployment and architectural reconfiguration scripts specific to the cloud environment
P82	Extraction of Microservices from Monolithic Software Architectures	Conference Proceedings	2017	Microservice extraction model to allow algorithmic recommendation of microservice candidates in a refactoring and migration scenario.
P83	Enabling Agile Business Process Modeling to Orchestrate Semantically-Annotated Microservices	Book Section	2017	Tool Microflows for automatically planning and enacting lightweight dynamic workflows of semantically annotated microservices using cognitive agents for replanning the workflow

ID	Titulo	Tipo	Año	Contribución
P87	Redefining a Process Engine as a Microservice Platform	Book Section	2017	Integrating a process engine in a microservice arhitecture
P91	Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity	Conference Proceedings	2017	Architecture approach to determine the granularity called microservice ambients which are defined by *Ambient-PRISMA Textual Language*
P92	Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures	Conference Proceedings	2017	Approach to improve the performance of a microservice architecture by workload-based feature clustering using a genetic algorithm
P93	Decision Models for Microservices: Design Areas, Stakeholders, Use Cases, and Requirements	Book Section	2017	Identify the areas of microservice design and create decision models (stakeholders and use cases)

La Figura 7 refleja el número artículos que permanecen en las distintas etapas durante la selección de los estudios primarios. Desde la búsqueda inicial de los artículos en bases de datos científicas aplicando los patrones ya establecidos, el filtrado de los artículos eliminando aquellos que son estudios secundarios, duplicados o no disponibles, hasta conseguir el resultado final mediante la aplicación de los criterios inclusión y exclusión.

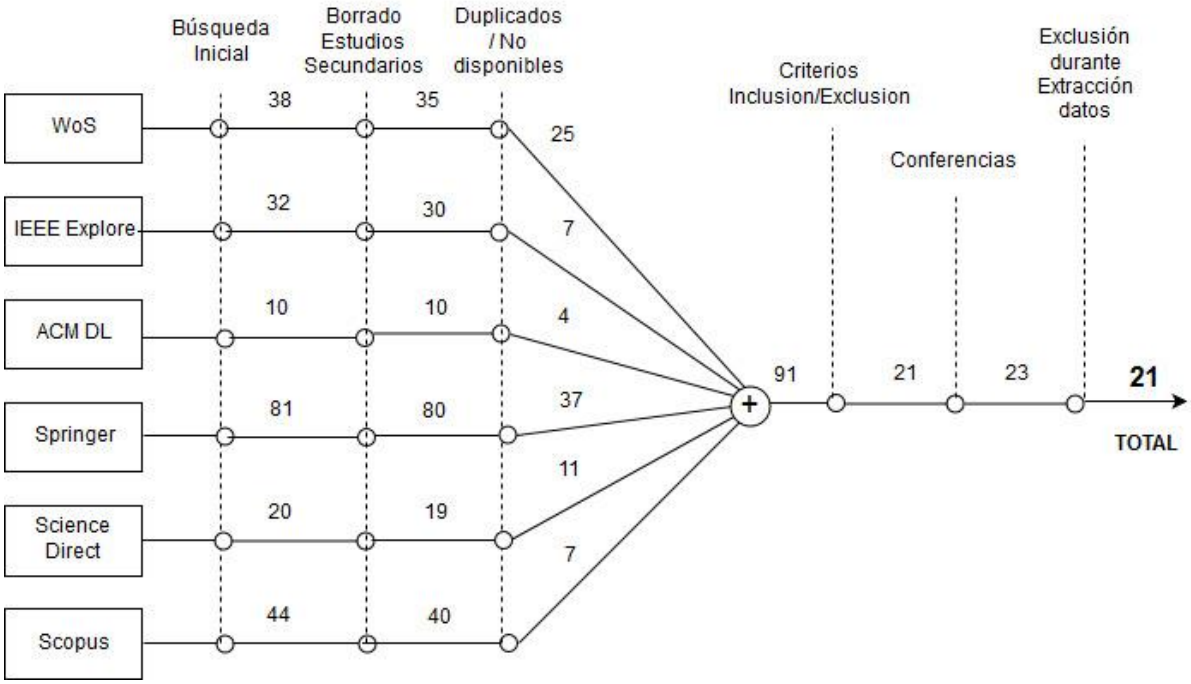


FIGURA 7 SELECCIÓN DE ARTÍCULOS DE LAS PRINCIPALES BASES DE DATOS

3.2.3 Extracción y síntesis de datos.

Como se ha mencionado anteriormente, y con el objetivo de facilitar la extracción de la información relevante, se rellena por cada artículo seleccionado, una serie de datos que se indican a continuación (véase Anexo C):

- Título
- Autores
- Datos de la publicación: nombre de revista, congreso, fecha, volumen, páginas, ISBN, DOI
- Si define un estilo arquitectónico basado/orientado a microservicios
    - Nombre dado al estilo
    - ¿define/caracteriza los tipos de componentes?
    - ¿Define/caracteriza los tipos de conectores/relaciones?
    - ¿Define/caracteriza las restricciones topológicas?
    - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
    - Qué elementos figuran en dicha arquitectura y cómo se caracterizan
    - Indicar cuál es el protocolo de comunicación entre los diferentes elementos
    - Qué estructura topológica guardan entre sí dichos elementos
    - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor…
- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Los tres primeros campos del formulario son mencionados a través de la referencia bibliográfica al final del documento. Además, se incluye la conclusión en base a las respuestas de investigación:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Como apoyo al proceso de síntesis y con el objetivo de encontrar patrones comunes, se crean mapas conceptuales en torno a los aspectos arquitectónicos encontrados en cada estudio (véase Anexo D).

Capítulo 4. Informe de Resultados

Este capítulo presenta los resultados de la revisión sistemática de la literatura de los 21 estudios primarios seleccionados, que responden a las preguntas de investigación. Para contrastar los resultados entre los distintos artículos, la información extraída se ha agrupado por temas comunes. De esta forma, se facilita la síntesis de los resultados obtenidos, y se vislumbra los desafíos actuales.

4.1 RQ1 ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

En respuesta a la primera pregunta de investigación podemos concluir que no existen estilos arquitectónicos para la construcción de sistemas en microservicios. Uno de los artículos coincide que no existe una arquitectura en microservicios, por carencia de principios, patrones y diseños, al estilo de la definición de SOA. Todo lo mencionado, se basa en conceptos operacionales, de desarrollo y procesos, así como en la organización en entornos distribuidos y usando servicios Cloud. Algunos autores creen que se trata de una evolución de una arquitectura orientada a servicios con ciertas peculiaridades. Generalmente, los microservicios poseen una orientación práctica mayor, debido a tener mecanismos más ligeros que SOA, y adoptan la filosofía "hacer solo una cosa, pero hacerla bien", es decir, de forma más eficiente. No obstante, se presenta una equivalencia con estilos arquitectónicos orientados a servicios, donde los microservicios son los componentes, y los conectores, representan las interacciones entre ellos.

Además de los microservicios (componentes), y las dependencias entre ellos, que en procesos de ingeniería inversa son analizadas a través de las llamadas en tiempo de ejecución, surge el "*Service Discovery*" para el descubrimiento de servicios en la red, y una pasarela "*API Gateway*" que enruta todas las comunicaciones (entrada y salida) entre los componentes y el usuario final.

En resumen, se muestra la arquitectura interna de un microservicio, compuesta principalmente de capa lógica, interfaz (a través de una API publica) y datos, y por el otro, la arquitectura que le rodea. Esta última, define todos los componentes que deben ser necesarios para el correcto funcionamiento de un sistema completo basado en microservicios, tales como *"Service Discovery", "API Gateway"*, y la comunicación entre ellos. También, se incluyen otros componentes para la puesta en operación en sistemas distribuidos, como los balanceadores de carga y los que evitan la sobrecarga de un nodo en fallo ("*Circuit Breaker" & "Health Status*") y para el despliegue se utilizan los contenedores.

4.2 RQ2 ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Debido a la ausencia de nuevos estilos arquitectónicos orientados a microservicios, tampoco existen lenguajes actuales para su definición. No obstante, algunos estudios primarios aportan lenguajes arquitectónicos o de programación de componentes que se adaptan a las características intrínsecas de la arquitectura en cuestión (véase más adelante en el apartado Hallazgos).

4.3 Hallazgos

El análisis de datos se realiza en dos fases, en la primera se elabora una lista inicial con los elementos y patrones comunes hallados en los estudios primarios, y en la segunda se forman categorías por los temas más relevantes. Finalmente, se establecen las siguientes categorías,

- Estilos Arquitectónicos
- Componentes, Conectores y Restricciones Topológicas
- Servicios Funcionales (Granularidad, Interacción entre servicios, Capa Datos)
- Servicios Infraestructura (Service Registry/Discovery, API Gateway)
- Coordinación (Coreografía, Orquestación)
- Despliegue/Operación (Containers, Bus mensajes, Circuit Breaker, Load Balancer)
- Equipos Desarrollo
- Lenguajes Descripción/Metamodelos

La Figura 8 muestra el número de artículos que mencionan las categorías anteriores. Los temas más citados son los que corresponden a elementos arquitectónicos, servicios de infraestructura y despliegue en entornos operacionales. Los equipos de desarrollo o desarrolladores no es un tema que se trate con énfasis, a pesar de ser una cacterística diferenciada en el ámbito de los microservicios.

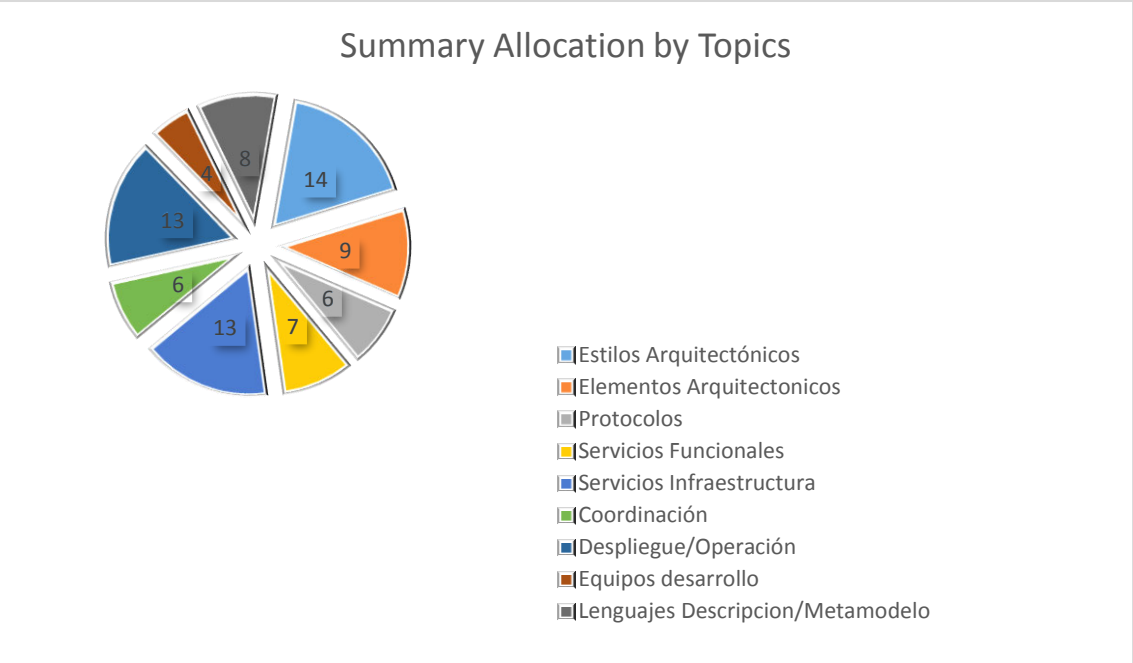


FIGURA 8 SUMAMRY ALLOCATION BY TOPICS

Por otro lado, la Figura 9 muestra una serie de claves (*keywords*) referidas en las distintas publicaciones (desde P01 hasta P93), que se incluyen dentro de las categorías anteriores. Puede observarse, que las claves más comunes son los que se incluyen dentro de la categoría elementos arquitectónicos (*Components & Connectors*), en despliegue y operación (*Container),* asi como los servicios de infraestructura (*Service Registry/Discovery*).

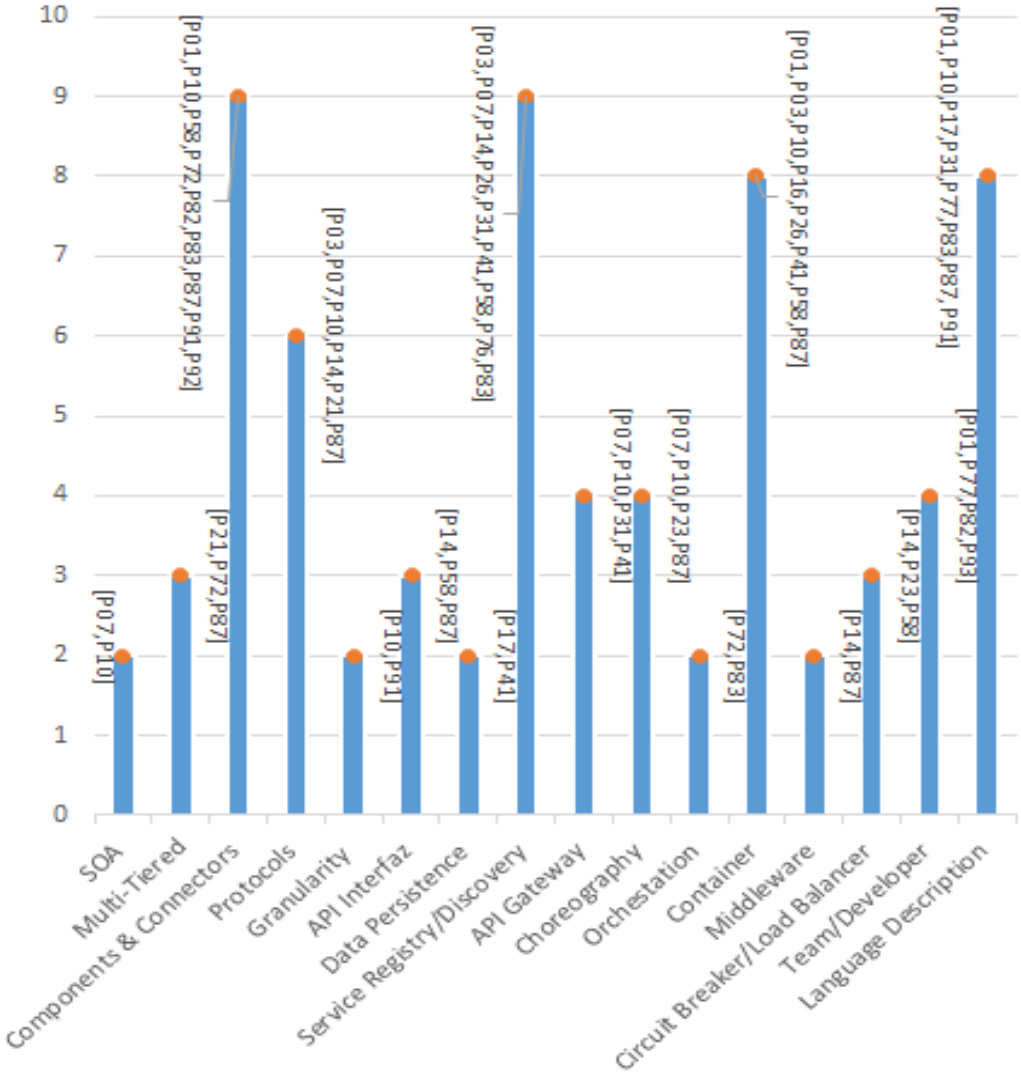


FIGURA 9 SUMMARY ALLOCATION BY KEYWORDS

4.3.1 Estilos Arquitectónicos

Una de las definiciones más vastas sobre el tipo de arquitectura se muestra en el [P31], donde clasifica la arquitectura en dos niveles: *inner architecture*, nivel arquitectónico que define los microservicios, y *outer architecture,* que engloba el anterior, añadiendo otras capas de comunicación con el usuario y con el entorno de operación. De esta forma, define cuatro capas arquitectónicas:

- ✓ Management: capa de comunicación con el cliente que ofrece servicios tales como Service Discovery, Routing (API Gateway), y Config.
- ✓ Inner Architecture.
- ✓ Messagging Channels: capa de comunicación entre microservicio que ofrece un middleware de la capa anterior (donde se definen y se instancian los servicios en su entorno de ejecución) con la capa operacional.

- ✓ Operational: capa que ofrece capacidades operacionales, tales como, automatización (ligado al concepto DevOps), y la monitorización de los microservicios.

Debido a las similitudes con las arquitecturas orientadas a servicios, hay dos estudios focalizados que establecen las diferencias con respecto a los microservicios. [P7] realiza una comparativa entre las características de ambas arquitecturas, en base a unos principios dados por el autor. Asimismo, [P10] establece las diferencias entre SOA (*Service Oriented Architecture*) y MSA (*Microservice Architecture)* desde una clasificación jerárquica en base a los conceptos SOA (*Service, Interaction, Execution Context).* SOA, tiene un modelo de referencia que abstrae los componentes en forma de modelos, a través de técnicas MDD (*Model-Driven Development*), llamado OASIS, y que ha originado una arquitectura de referencia. Por analogía, se muestra las necesidades y/o implicaciones de un modelado en conceptos y en lenguajes para MSA utilizando técnicas MDD.

Cabe destacar, el estilo arquitectónico *(3-tiered app)* definido en el artículo [P21], que divide el componente o microservicio en tres niveles o capas: *interface layer, business layer logic, data persistence layer.* Igualmente, la arquitectura presentada en el artículo [P87], incluye la lógica de negocio, los datos y la interfaz usuario (si existe como parte de una general), además de la interfaz API. Por un lado, existe *BPMS API*, que expone la lógica negocio del microservicio, y *Event API*, que gestiona el envío/recepción de eventos en un sistema con metodología suscripción/publicación. Adicionalmente existe un intermediario *Message Broker*, que traduce y enruta los mensajes, y un *Process Engine***,** Implementado dentro del microservicios, que se encarga de manejar los eventos, y conecta con los datos. También presenta un modelo de referencia compuesto de bloques clave (*key building blocks*), juntamente con otros bloques ya existentes en el marco empresarial:

Microservice, API Proxy, Domain Information Model (conjunto microservicios, cada uno de ellos en estilo arquitectónico 3-tier), Enterprise API Registry, Enterprise Microservice Repository, Enteprise Monitoring & Tracking

Existen otros artículos que hablan de una arquitectura multi-capa*,* como el artículo [P72]:

. *Component:* Son componente software o microservicios.

. *Service Graph*: Grafico de componentes que encapsula información sobre los microservicios.

. *Service Deployment*: Capa que guarda la asociación de componentes y sus recursos IaaS, principalmente su ubicación y configuración.

. *Service Runtime*: Encapsual información sobre las instacias en ejecución.

Dado que las arquitecturas en microservicios y IoT poseen similitudes, tales como la independencia en tecnologías y la existencia de un mecanismo ligero de comunicación, el estudio [P23] presenta un análisis de los patrones de diseño comunes en ambas arquitecturas. Sin mucho detalle, establece una *Arquitectura Bottom-up* para IoT, múltiples dispositivos heterogéneos que necesitan ser interoperables entre ellos, y una *Arquitectura Top-Down* para microservicios, puesto que constituye una descomposición de un sistema en distintas partes con comunicación entre ellas. Además, presenta una arquitectura hexagonal, que separa la lógica de negocio de los aspectos tecnológicos externos. Mediante varios puertos y una capa de mediación, se conecta a aspectos

transversales y a diferentes tecnologías a través de adaptadores.

4.3.2 Componentes, conectores y topología

Se enumeran algunos aspectos arquitectónicos, sobre todo en los componentes y comunicación entre ellos utilizando distintos protocolos. la mayoría de documentos establecen los componentes que son los microservicios, y los conectores mediante las interacciones entre ellos.

- Componentes

El estudio [P01] establece el tipo de los componentes o servicios, que puede ser funcional o de infraestructura, y atributos especiales, tales como IP o máquina donde se instancia cada servicio**.** Debe ser diseñado en base al patrón conocido como "*Bounded Context*" en la metodología DDD (*Domain-Driven Design*), que trata de dividir un modelo grande en unidades pequeñas y definir explícitamente sus relaciones. La agrupación entre servicios forma un cluster, y cada microservicio muestra una interfaz (siendo el conector), que presenta un *"endpoint"* en la comunicación, con puerto entrada/salida y relaciones de tipo *"expose/require"*. La conexión entre dos interfaces se hace mediante un enlace. Las dependencias y la comunicación entre los microservicios (componentes) se realizan mediante una interfaz de programación de aplicaciones (API), definida en términos de negocio. RESTful APIs proveen el modelo lógico para construir interfaces entre los componentes de una arquitectura en microservicios, que deben cumplir dos requisitos: *Message-oriented & Hypermedia-driven.*

El artículo [P83] establece los componentes y unas fases para determinar el camino óptimo de invocación de servicios en base a un plan inicial. Introduce el concepto de metaservicios que incluyen los microservicios abstractos, y el microservicio de registro y descubrimiento, así como agentes clientes dotados de inteligencia que pueden actuar según su conocimiento. Los microservicios con funcionalidad similar, pueden agruparse en un microservicio abstracto, de forma que el cliente no conozca los servicios concretos, puesto que pueden ser numerosos y cambiar dinámicamente. [P82], es otro artículo que agrupa los microservicios mediante un grafo. Cada grafo está compuesto por nodos (o clases) que están unidos por relaciones con un factor de dependencia asociado ("*Weight Coupling*").

Por otro lado, el trabajo [P91], establece una serie de componentes en la arquitectura llamado "*Microservice Ambients*", concebidos para el cambio dinamico de la granularidad del sistema, donde se especifica las interacciones internas y con el exterior a través de transacciones de tipo "*merge/decompose*".

El artículo [P72], describe los siguientes atributos de un componente software (microservicio):

- "*Exposed interface*", interfaz que describe el conjunto de funciones más granulares que son expuestas del componente.

- "*Required interface*", interfaz que describe el conjunto de funciones más granulares que son consumidas por el componente.

- "*Requirement*", conjunto de requisitos que se deben cumplir a la ejecución de un componente (CPU, memoria, etc.)

- "*Configuration*", atributo con aspectos de configuración para la fase de despliegue.

- Conectores (Protocolos de comunicación)

Si bien los microservicios son independientes, esta característica no significa que estén aislados. Deben proveer interfaces que sean conocidas por los clientes (ya sean otros servicios o usuarios finales) sin tener conocimiento de su ubicación ni de la tecnología subyacente. Por lo tanto, ofrecen API's que utilizan protocolos de comunicación estándares como HTTP, bajo las tecnologías REST e incluso SOAP (Web Services), así como los mecanismos de intercambio de datos como XML y JSON.

En primer lugar, se define el modo síncrono o asíncrono, y el tipo de elementos que intervienen en la comunicación, *one-to-one,* un mensaje se consume por un único consumidor, y *one-to-many*, un mensaje se consume por múltiples consumidores. Por lo tanto, caben las siguientes posibilidades expresadas en [P14], [P03]:

TABLA 7 IPC MECHANISM

	SYNC	ASYNC
One-to-one	*Request/Response*	*Notification (Async Response)*
One-to-many	------- (1)	*Publish/subscribe*

(1) El suscriptor y publicador no se conocen, por lo que no es posible recibir la respuesta.

Los estudios [P87], [P03] recomiendan el uso de un mecanismo IPC asíncrono orientado a eventos para la comunicación entre servicios (*event-based asynchronous communication*), utilizando el patrón suscripción/publicación. Para facilitar la comunicación asíncrona, se debe implementar un "*message broker*", donde los servicios consumidores se suscriban y además gestione todos los eventos o notificaciones desde el proveedor. De esta forma, se evita la dependencia entre publicador (proveedor) y suscriptores (consumidores).

Mediante mecanismos asíncronos de comunicación basados en eventos con patrón ***publish/suscribe*****,** los componentes pueden ser añadidos y/o eliminados sin demasiado impacto en el sistema.

Entre los protocolos de comunicación utilizados, [P87], [P03], [P07], [P14], coinciden en la existencia de los dos principales: **REST (***Representation State Transfer*, estándar con restricciones para la distribución de sistemas distribuidos hypermedia) **over HTTP o Thrift,** para mensajería síncrona y **AMQP,** en modo asíncrono. Este último es un protocolo a nivel aplicación con orientación a mensajes, encolamiento (*queuing*) y enrutamiento tanto punto a punto, como suscripción/publicación. En el estudio [P10] se indica que debe ser el mismo protocolo en ambas direcciones en la comunicación, ya sea para interacción externa con otros usuarios o aplicaciones, o para una interacción interna entre servicios (*protocol-aware comm*). Otro protocolo de comunicación es definido en [P14], **SWIM** para la detección y diseminación de fallos en un cluster, que se ajusta al

descubrimiento de los servicios por la red. Cada nodo le comunica al siguiente el fallo de uno de ellos.

- Restricciones Topológicas

[P92] presenta un modelo para componer microservicios, agrupándolos según sus funcionalidades, y reduciendo el número de dependencias entre servicios. Para ello, considera los siguientes componentes en la arquitectura (definiendo una notación formal):

- ✓ *Set of Microservices* (M): Conjunto de microservicios.
- ✓ *Set of feature instances* (I): Conjunto de características instanciadas de cada microservicio. Puede ser publica (contiene todas las propiedades) o interna (un subconjunto de propiedades).
- ✓ *Property Instantiation Function* (^: I -> 2P): Función que mapea cada "*feature instance*" con un conjunto de propiedades.
- ✓ *Public Instante Function* (h: F -> I): Función que define una instancia publica por cada característica.

Y las siguientes restricciones topológicas:

- ✓ Cada MS contiene todas las instancias necesarias para cumplir los requisitos de dependencias.
- ✓ Cada MS contiene cada característica al menos una vez.
- ✓ Cada instancia de una característica tiene un conjunto de propiedades de su característica.
- ✓ Cada característica tiene una instancia pública que contiene todas sus propiedades.
- ✓ Cada MS contiene al menos una instancia pública de una característica.

4.3.3 Servicios Funcionales

Aquellos servicios que responden a la lógica de negocio, donde está definido la interfaz API de comunicación y los datos. Estos servicios interactúan entre ellos, intercambiando información mediante protocolos de comunicación

- Capa Funcionalidad

Cada microservicio o componente debe estar asociado a una funcionalidad que defina una capacidad de negocio. Uno de los principios de la arquitectura de microservicios, es que los componentes estén ligeramente acoplados, para minimizar dependencias con el fin de que puedan ser ejecutados de forma independiente. Por lo tanto, tal y como se indica en [P10], es crucial determinar el tamaño y la granularidad del servicio o componente, de forma que responda a la lógica de negocio y que sea lo suficientemente independiente para no exceder en las interacciones o llamadas a otros servicios. Igualmente, el trabajo [P91] aborda el problema de diseño de la granularidad de los microservicios, basándose en una asignación dinámica y flexible de las granularidades en tiempo de ejecución. Proponen un modelo, *Microservice Ambients,* que establece las primitivas para adaptar los límites (*boundaries*).

- Capa Interfaz

Como hemos mencionado anteriormente la interfaz pública es el conjunto de funciones que expone el microservicio para ser consumido. Todos los artículos son coherentes con respecto a la necesidad de tener una interfaz programación de aplicaciones o API, por cada componente bajo estándares comúnmente conocidos como REST (o HTTP). En el artículo [P87], se diferencia entre la BPMS API y Event API, siendo la primera la que expone la lógica de negocio.

- Capa Datos

Debido a la descentralización de la gestión de datos, cada microservicio debe manejar su propia base de datos de cualquier tipo, ya sea convencional o NoSQL. El artículo [P17] presenta el enriquecimiento de un lenguaje de programación orientado a servicios, para incluir nuevos operadores de selección que permitan el cambio en el flujo de control que considera previamente la estructura y tipo de mensaje (*data-driven approach*) evitando un procesamiento posterior a la recepción (*process-driven approach*). De esta forma, se crean funciones genéricas que se comportan según el argumento de datos que se pasa a la función.

El artículo [P41] ofrece un nuevo patrón arquitectónico llamado "*Database-is-a-pattern*", con el fin de utilizar la base de datos como servicio, mediante la unión de la lógica de negocio junto con las bases de datos. Esta solución ofrece ciertas ventajas como la simplificación de la arquitectura, eliminando una capa de servidor o contenedor.

4.3.4 Servicios Infraestructura

Son todos aquellos servicios complementarios que no representan la funcionalidad o lógica de negocio, pero que son indispensables para el correcto funcionamiento de un sistema basado en microservicios.

Principalmente, son el "*Service Registry & Discovery*", que es una base de datos que contiene la ubicación en la red de las instancias de los servicios (cada instancia se registra en el arranque y se elimina al apagarse), y la "*API Gateway*", que usa esta información para el enrutamiento de las peticiones del cliente durante el descubrimiento de los servicios.

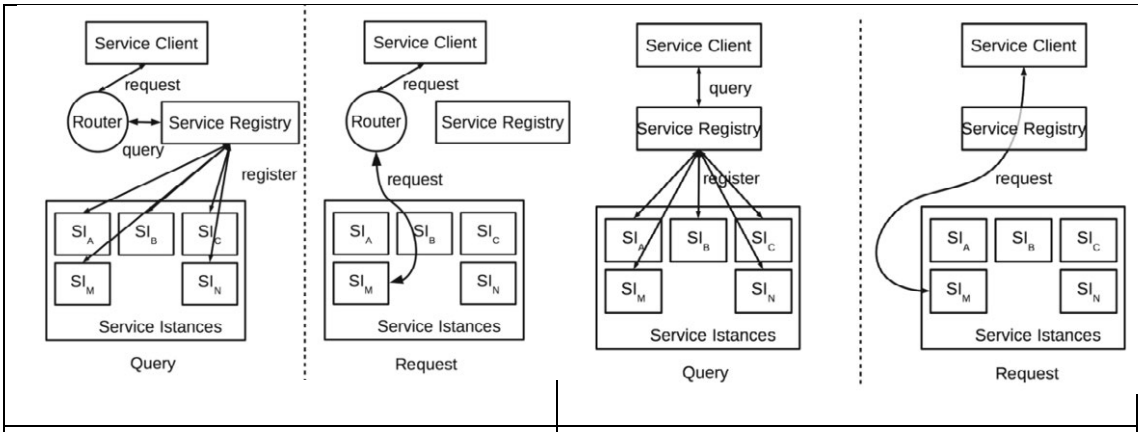
- Service Registry/Discovery

Dado que un microservicio puede cambiar su estado en entorno de ejecución ya sea por autoescalado, fallos y/o actualizaciones, existen mecanismos de descubrimiento de servicios en la red para mantener débil el acoplamiento entre los componentes. De forma que se presenta un nuevo componente en la arquitectura que cubre las dependencias existentes entre servicios (ratificado en la mayoría de los estudios), para localizar y registrar los servicios en el sistema. Este componente debe ser altamente disponible.

En efecto, es considerado como un meta-servicio en el estudio [P83], que construye un gráfico de nodos que contienen propiedades de los microservicios y de sus enlaces (f*ollowers*) a otros. [P26] propone "*ICN-Based Service Discovery"*, que utiliza tablas de enrutamiento FIB, donde se asocia el nombre del servicio (*ISD Name*) y el próximo punto más cercano (*NextHop*), para transmisión de la petición del cliente. Se utiliza la técnica "*longest-prefix matching*" para buscar la coincidencia del nombre del servicio. Diferencia entre dos tipos posibles, ISD (*Information-centric Service Discovery*) que es el más común, y otro basado en el concepto ICN (*Information-centric Networking*), que separa

las instancias de su ubicación a través de los nombres de servicios.

Hay dos tipos de implementación en el mecanismo para el descubrimiento de servicios clasificados en [P41]: *Server-Side Discovery & Client-Side Discovery.* El primero incluye un intermediario para el enrutamiento de las peticiones del cliente hacia las instancias de los servicios.



Server-side discovery pattern	Client-side discovery pattern

FIGURA 10 SERVICE DISCOVERY PATTERN

- API Gateway

Mediante la ***API Gateway o Proxy***, se define como los clientes acceden a los servicios, siendo capaz de ofrecer diferentes API's por cada cliente. En realidad, agrupa interfaces y realiza el enrutamiento de la petición entre servicios, en los dos sentidos de la comunicación. Es un elemento arquitectónico relevante en la mayoría de los estudios primarios.

4.3.5 Coordinación

- Coreografía

Tal y como se menciona en los artículos [P07], [P10], [P23] y [P87], la coreografía se impone a la orquestación comúnmente utilizada en arquitectura orientada a servicios tradicionales como SOA. En la coreografía, cada nodo escucha eventos (*event-driven),* y realiza su parte si es necesario originando "*Service Call Chains*", mientras que, en la orquestación, existe un nodo central que controla los servicios que son llamados. Para la interacción entre los servicios mediante coreografía, se requiere que cada microservicio conozca detalles del dominio incluyendo los puntos finales (*endpoints*), y el protocolo de comunicación. Toda esta información, sería almacenada por el orquestador, en el caso de aplicaciones SOA.

- Orquestación

En relación a los flujos de trabajo (*workflow*), [P83] ofrece un modelo operacional llamado *Microflows* para la integración y orquestación de microservicios en los procesos de negocio usando agentes inteligentes, métodos gráficos y una semántica basada en *JSON-LD (JavaScript Object Notation for Linked Data*), que es el formato de mensaje que se intercambian y *Hydra,* que representa el vocabulario entre ellos.

Otro modelo propuesto en [P72], llamado *ARCADIA Context Model*, facilita el despliegue de las aplicaciones distribuidas orientadas a microservicios mediante una infraestructura que provoca la ejecución de unas acciones basado en el análisis de los datos actuales y un conjunto de reglas. Frente a la coreografía, ofrece un orquestador de los servicios llamado *"Smart Controller"*.

4.3.6 Despliegue/Operación

Debido a la importancia del entorno de ejecución en la tecnología objeto de nuestra revisión, se deben considerar aspectos operacionales, que incluyan información sobre las instancias de los servicios en contenedores, y de su comportamiento. De esta forma, el estudio [P10] muestra las clases de un meta-modelo para orientación a servicios (*Service, Operation, Message, Port, Behaviour*), subrayando otras implicaciones en microservicios, tales como: (i) El modelado del comportamiento a nivel arquitectónico basado en la coreografía o *Service Call Chains*; (ii) Las técnicas DevOps para la integración y agrupación a nivel operacional de los servicios; (iii) Proveer datos técnicos en relación a los puertos, tales como el protocolo, el formato del mensaje y la tecnología contenedora. También, [P03] introduce varios componentes operacionales que deben formar parte de una arquitectura orientada a microservices con el fin de proveer servicios distribuidos: *Containerization, Service Communications, IPC (Inter-Process communication), Service Registry-Discovery* (los últimos ya tratados anteriormente).

Asimismo, el estudio [P92] ofrece un modelo de despliegue de arquitecturas en microservicios (desde el punto de vista de las dependencias entre microservicios y workload). MicADO, es la herramienta que optimiza un modelo arquitectónico en microservicios mediante un algoritmo genético, dado un flujo y un modelo despliegue original.

- Contenedores

Por el interés creciente sobre la tecnología contenedores para ser utilizado con los microservicios, [P16] es incluido para conocer la tecnología subyacente que existe en Docker. Describe una arquitectura cliente/servidor: "Docker Client" que se comunica mediante un conjunto de RESTfull API's a través de sockets con el "Docker Host", quien tiene el demonio que escucha las peticiones del cliente. Las características o atributos que deben ser asociados a los contenedores que ejecutan un microservicio, son mencionados en el estudio [P01]: *IP, Identifier, Network, Interface*. [P01], [P26], [P10], [P41], [P58] y [P03] resaltan la importancia de utilizar los contenedores donde se instancia la aplicación dentro de la definición de una arquitectura en microservicios. Un contenedor es similar a una máquina virtual que comparte el mismo sistema operativo de la máquina donde se ejecuta el contenedor. Si bien es dependiente de la maquina host, mejora el rendimiento y agiliza el arranque del sistema.

- Middleware (mensajería)

El estudio [P14] ofrece un middleware de comunicación descentralizado "***Decentralized Message Bus***" (llamado Serf) como medio de comunicación entre los servicios. También define los tipos de mensajes que son intercambiados entre los nodos, y los elementos arquitectónicos de bus descentralizado. También [P87] utiliza un componente intermedio para intercambio de mensajes "*Message Broker"*.

- Circuit Breaker/Load Balancer

Con el fin de mejorar los problemas de comunicación ligados a los sistemas distribuidos, como la latencia en la red o balanceo de carga, son necesarios algunos componentes para la distribución de tareas en los servidores disponibles y con menos carga, y evitar las llamadas a servicios fallidos. Para ello, se utilizan mecanismos como "*Health Status*" y "*Circuit Breaker",* mencionados en el estudio [P23]. El estado de salud del sistema es revisado por el "*Circuit Breaker*", quien recuerda el número de llamadas sin éxito y si se ha alcanzado un determinado limite. Si es lanzado devuelve un error en vez de reenviar la llamada para prevenir que el servicio que falla reciba más mensajes.

[P58] introduce varios componentes en operación para resolver los problemas añadidos de la integración y descomposición en microservicios: *Config Server, Edge Server, Circuit Breaker* (para aumentar la resiliencia en entornos distribuidos) y balanceador de carga. Así mismo, [P31] incluye una capa de balanceador de carga y un entorno de ejecución por cada componente, con el fin de comunicarse con la capa operacional.

4.3.7 Organización Equipos

Debido a que los servicios son pequeños y aislados, el esfuerzo en el desarrollo está divido en equipos, que pueden trabajar de forma independiente, lo que mejora la asignación de responsabilidades en los equipos de desarrollo. Por otro lado, los equipos son multifuncionales (*cross-functional*), reúnen conocimiento y habilidades para ser autónomos en el desarrollo del producto en todas sus fases.

Cabe destacar, la importancia del estudio [P93] ya que define una estructura sobre las principales áreas de diseño que deben ser consideradas al introducir arquitectura en microservicios y soportadas mediante modelos de decisión (*Decision Models*): (i) Diseño del sistema, (ii) Organización de los equipos y procesos, (iii) Infraestructura. Además del diseño del sistema, también destaca la importancia de analizar los aspectos sobre la organización de procesos y equipos, y de la infraestructura elegida. Se determina que los equipos deben ser multifuncionales y organizados por capacidades de negocio. Un equipo puede desarrollar varios microservicios, pero un microservicio solo puede ser desarrollado por un equipo. En consecuencia, se fomenta la relación interna mejorando su productividad y evitando pérdidas de tiempo con relaciones externas con otros desarrolladores de otras áreas lógicas de negocio o microservicios.

Por otro lado, se utiliza el factor desarrollador como clave para la agrupación en microservicios. [P77] y [P82], ofrecen unas herramientas para la migración de sistemas ya existentes a otros con orientación en microservicios y tienen en cuenta las llamadas entre los componentes, equipos de desarrollo y su semántica. Es decir, la composición de los microservicios se realiza en base a sus dependencias, funcionalidad y desarrollador.

4.3.8 Lenguajes de Dominio/Meta-Modelos

En relación a los lenguajes específicos de un dominio o de programación de los componentes, cabe destacar el estudio [P17] presenta el lenguaje *Jolie*, *(Java Orchestration Language Interpreter Engine),* asociado a la arquitectura orientada a servicios. Este lenguaje ofrece primitivas para la implementación y orquestación de servicios, que constituye una alternativa a lenguajes basados en XML, como *WS-BPEL* (*Business Process Execution Language*). Adicionalmente, presenta una abstracción de la

capa de comunicación, que es la encargada de la recepción/envío de mensajes, así como codificación/decodificación utilizando distintos protocolos y por distintos canales. Un programa escrito en Jolie, diferencia la parte de despliegue y la de comportamiento, de forma que un mismo comportamiento o funcionalidad puede ser reutilizada con distinta configuración en despliegue. El despliegue (*deployment*), contiene las directivas para la orquestación y la comunicación entre los servicios (*one-way, request/response*), incluyendo información sobre las interfaces, los puertos de comunicación y los tipos de mensajes. Por otro lado, el comportamiento (*behaviour*) define la funcionalidad del servicio.

Esta misma división, entre comportamiento y despliegue, es subrayada en el artículo [P10] en referencia a los lenguajes de modelado en arquitectura en microservicios. Se menciona que existe solamente un estudio (no objeto de nuestra investigación) que presenta un lenguaje de modelado en tiempo ejecución, infraestructura y dependencias. Otros lenguajes de modelado como *ArchiMate* y lenguajes de ontologías *OWL* son mencionados en el estudio [P31], para la descripción de la capa de los metamodelos. En dicho estudio define cuatro capas estructurales basado en el estándar MOF *(MetaObject Facility),* que son la base de la descripción de una arquitectura en microservicios: *(i) Run Time Data, (ii) Meta data & Inner Architecture model (components, com. channels), (iii) Architecture Metamodel & Ontology, (iv) Abstract Language*

A través del metamodelo definido en [P01] que contiene entidades especificas del dominio que están conectadas a través de relaciones, se define un DSL *(domain-specific language)* para un modelo de referencia asociada a una arquitectura en microservicios. Está compuesto de siete metaclases: *Product (root), Microservice, Cluster, Developer, Team, Interface, Link y ServiceType,* que se conectan a través de relaciones de tipo: *compose, require, expose, divide, workfor, owned*.

Un lenguaje que facilita la especificación para la integración de componentes en una arquitectura en microservicios para entornos cloud, es sugerido en el estudio [P77], basado en una extensión SoaML y UML (*DIARy-specification-profile*). Define metaclases que describen la lógica del proceso de integración y el impacto en la arquitectura mediante la definición de atributos, tales como, el nivel de escalabilidad y la inmediatez en la ejecución del componente o microservicio.

En el estudio [P91], *Ambient-PRISMA Textual Language,* concebido para realizar el cambio dinámico de la granularidad de los microservicios en tiempo de ejecución, es propuesto como un potencial lenguaje para definición de microservicios. Con el mismo fin, valora otros ADL's con sus características y limitaciones: MetaH y UniCon soportan evolución de componentes, así como ACME, Rapide y C2 que lo soportan por subtipado, pero no se ajustan a las transacciones *merge/decompose* definidas en el lenguaje sugerido.

En el contexto empresarial, BPMN (*Business Process Model notation*) y BPEL (*Business Process Execution Language*), son lenguajes mencionados en [P87], [P83]. El primero sirve para la definición de la lógica y procesos de negocio, y provee un mecanismo de comunicación orientada a eventos. BPEL constituye un lenguaje estándar para la

integración y automatización de procesos, y el manejo de eventos en la orquestación de servicios.

4.4 Discusión de los resultados y desafíos actuales

Después de analizar las diferentes publicaciones en bases de datos científicas, así como en reputadas conferencias internacionales, se comprueba que hay poca literatura o artículos que traten sobre una arquitectura de referencia o estilo arquitectónico en microservicios. Sin embargo, se ha comprobado que existen muchos artículos publicados sobre el tema en cuestión, que presentan ciertas discusiones y desafíos actuales.

En primer lugar, como se ha puesto de manifiesto en muchos de los artículos y concretamente en el estudio [P91], la arquitectura en microservicios debe lidiar con el problema de granularización de los componentes, para no exceder en las interacciones entre ellos, y se mantenga el principio de componentes débilmente acoplados.

Un sistema basado en microservicios no incluye una capa de abstracción o bus para la gestión de la comunicación para el intercambio, enrutamiento y transformación de mensajes de diferentes protocolos, de la misma forma que ofrece SOA a través de ESB (Enterprise Service Bus). Por lo tanto, la comunicación entre componentes debe utilizar el mismo formato y estructuras de mensajes, (protocol aware-communication), según indica el estudio [P10]. Contrariamente, se ofrece una capa intermedia de mensajería o bus de mensajes en el artículo [P14] y [P31].

Uno de los desafíos actuales es que los nuevos sistemas y aplicaciones puedan ser fácilmente compuestos mediante la combinación de la funcionalidad de una colección de servicios, o de refactorizar las aplicaciones existentes en microservicios, que sería la mejor opción para mantener funcionando dichos sistemas.

Son numerosos los estudios que muestran prototipos, herramientas y modelos, que abordan el problema de la adopción de los microservicios mediante la migración de aplicaciones ya existentes o incluso de la adaptación a sistemas Cloud [P77] con un enfoque empresarial. Por un lado, [P31] ofrece una propuesta basada en una mini descripción de la arquitectura empresarial llamada "*EA-Mini-Descriptions*". Surge para la integración de un largo número de estructuras micro-granulares con requerimientos de heterogeneidad, volatilidad y distribución. Adicionalmente, [P21] prioriza la transformación de un modelo empresarial para utilizar microservicios, ante la definición de una arquitectura propia, incluyendo nuevos componentes con las ya existentes. El estudio [P58] descompone un sistema monolítico en microservicios, en base a los propietarios de los datos y sustituyendo las dependencias del código por llamadas a servicios. Por último, [P01] presenta una herramienta para la recuperación de la arquitectura de una aplicación en microservicios, aplicando ingerinería inversa y técnicas de MDE (Model Driven Engineering). Teniendo en cuenta factores estáticos (desarrolladores y funcionalidad) que son extraidos a través del código fuente de un repositorio, y factores dinámicos (llamadas entre servicios) que son recopilados a través de los logs de ejecución, se origina un modelo de un nivel de abstracción mayor, que presenta la arquitectura del sistema. Igualmente, el estudio primario [P82], presenta un modelo de extracción de microservicios partiendo de sistemas monolíticos. Implementa estrategias de acoplamiento entre entidades (en base a sus dependencias, semántica y/o desarrollador)

que sirven para construir un gráfico de dependencias entre las clases del sistema. Este gráfico es dividido utilizando un algoritmo de agrupamiento, que va definiendo los distintos microservicios.

En consecuencia, se constata que una simple llamada a un procedimiento se convierte en una dependencia entre servicios, por lo que la reducción de llamadas entre los distintos servicios es clave para la construcción eficiente del sistema. Igualmente, la dependencia o asignación de un equipo de desarrollo y/o desarrolladores es esencial, para la correcta definición de un servicio.

Debido a la independencia de los lenguajes de implementación entre los servicios, un desafío es crear un lenguaje arquitectónico que tenga en cuenta otros aspectos asociados a los microservicios, tales como, las interfaces, el comportamiento y el despliegue. Es necesario incluir elementos técnicos y operacionales (asignación protocolos a servicios, configuración contenedores, aspectos dinámicos, etc...).

Capítulo 5. Conclusiones y Trabajo Futuros

Este capítulo resume los puntos principales del trabajo de fin de máster. Además, en este capítulo se plantea futuras líneas de investigación que podrían ser derivadas de este trabajo.

5.1 Conclusiones

En nuestra revisión sistemática de la literatura se pone de manifiesto que no existen estilos arquitectónicos ni arquitectura de referencia para las aplicaciones basadas en microservicios. Existe una vinculación al estilo arquitectónico SOA, que se basa en la integración de aplicaciones mediante servicios, que representan la medida más granular sobre la que se construyen otros artefactos, incluso algunos autores creen que se trata de una interpretación más refinada. Sin embargo, las características intrínsecas a los microservicios, muestran la necesidad de definición de una propia arquitectura que se ajuste sobre todo a las necesidades de sistemas heterogéneos y distribuidos donde se deben automatizar un conjunto de tareas (*workflow-based applications*), de sistemas altamente escalables e incluso de aplicaciones web con un mismo protocolo de comunicación (*no protocol transformation*).

Una de las causas por las que no se encuentra en la literatura propuestas de estilos arquitectónicos en microservicios, que definan formalmente los componentes, sus relaciones y restricciones topológicas, es la heterogeneidad que presentan frente a otros estilos, y su fuerte dependencia con el entorno de operación. Además, es un término que ha tomado relevancia a partir del año 2104, y aunque está en auge, las empresas todavía son reticentes en cambiar a una arquitectura en microservicios. Normalmente, la definición de arquitecturas de referencia o estilos arquitectónicos ligados a las propuestas de sistemas informáticos tardan tiempo en ser definidas, puesto que exige un rigor y una conformidad por todos los actores involucrados.

Tampoco existen SLR's (*Systematic Literature Review*) previas donde se trata un estilo arquitectónico o arquitectura de referencia de los microservicios, y los *Systematic Mapping* hallados son investigación de los trabajos primarios que tratan en general sobre el termino microservicios. Cabe destacar uno (Francesco, 2017), que en su tesis doctoral ya afirma que no existe ningún AL (*architecture language*) específico para microservicios y se propone revisar los AL existentes para SOA y analizar las extensiones y cambios necesarios.

Por otro lado, se muestra la ausencia de estándares para describir microservicios, y la importancia de definir correctamente los requisitos, independientemente de la capacidad de desarrollo, despliegue y escalado de dichas aplicaciones. Al contrario, en muchos artículos se pone énfasis en la capacidad de despliegue de los microservicios, dando muchas características generales y patrones ya existentes.

Varias soluciones arquitectónicas que aportan los estudios primarios seleccionados están centradas en la transformación de un modelo empresarial para utilizar microservicios, que en definir una arquitectura propia sobre ellos. Lo único interesante, a nivel arquitectónico, es la adaptación de la arquitectura de tres capas para definir un microservicio, y los nuevos

componentes arquitectónicos adicionales como el descubrimiento y registro de servicios. Tanto los aspectos operacionales, que van desde la instanciación de los servicios en los contenedores y los elementos propios de sistemas distribuidos, como la coordinación entre los componentes del sistema, son cruciales para el buen funcionamiento de un sistema en microservicios (véase la Figura 11)

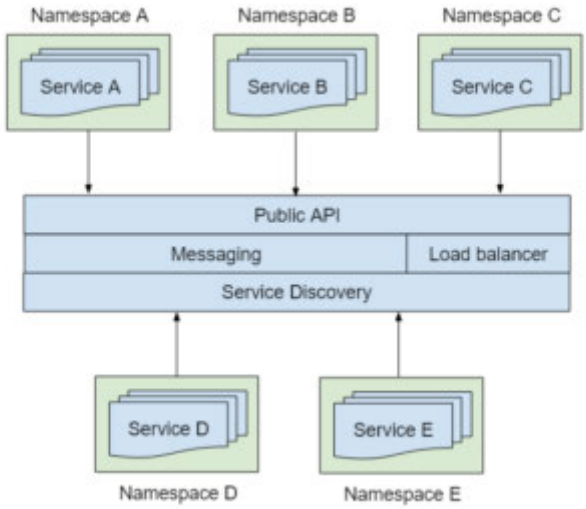


FIGURA 11 MICROSERVICE ARCHITECTURE COMPONENTS (Kookarinrat & Temtanapat, 2016)

5.2 Trabajos Futuros

Debido a la filosofía de dicha tecnología, en el que un sistema se divide en múltiples componentes individuales, resulta complicado tener una visión completa de la arquitectura. Aunque existen soluciones para la migración de sistemas tradicionales a la tecnología en microservicios, y algunas propuestas arquitectónicas para la adopción de este modelo reciente que ofrece ciertas ventajas, la conclusión del trabajo de investigación hace constar la inexistencia de estilos arquitectónicos propios de aplicaciones basadas en microservicios. Por lo tanto, se plantea como trabajo futuro una aproximación a la definición de un estilo arquitectónico, o la adaptación de alguno de los ya existentes.

Siendo el termino microservicios reciente, podría ocurrir que en los próximos años aparecieran otros estudios que se ocupen de las arquitecturas de referencias que han sido objeto de la investigación. Estas propuestas pueden ser planteadas desde la comunidad académica o incluso originadas en entornos empresariales, donde el término microservicios está en auge. En consecuencia, una valoración de los estudios a partir del 2018 podría mostrar cambios con respecto a la conclusión de este trabajo.

Si hubiera alguna propuesta de estilo arquitectónico publicada a partir de la fecha de la presentación del trabajo actual, también se propone realizar la evaluación de calidad, rigor y pertinencia, así como el análisis más exhaustivo por cada iniciativa.

Anexo A: Kappa de Cohen[i]

El **Coeficiente Kappa de Cohen** es una medida estadística que ajusta el efecto del azar en la proporción de la concordancia observada para elementos cualitativos (variables categóricas). En general se cree que es una medida más robusta que el simple cálculo del porcentaje de concordancia, ya que κ tiene en cuenta el acuerdo que ocurre por azar. Algunos investigadores han expresado su preocupación por la tendencia de κ a dar por seguras las frecuencias de las categorías observadas, lo que puede tener el efecto de subestimar el acuerdo para una categoría de uso habitual; por esta razón, κ se considera una medida de acuerdo excesivamente conservadora.

Otros discuten la afirmación de que Kappa "tiene en cuenta" la posibilidad de acuerdo. Para hacerlo con eficacia se requeriría un modelo explícito de cómo afecta el azar a las decisiones de los observadores. El llamado ajuste por azar del estadístico Kappa supone que, cuando no están absolutamente seguros, los evaluadores simplemente aventuran una respuesta (un escenario muy poco realista).

Calculo

El Coeficiente Kappa de Cohen mide la concordancia entre dos examinadores en sus correspondientes clasificaciones de N elementos en C categorías mutuamente excluyentes. La primera mención de un estadístico similar a Kappa se atribuye a Galton (1892), véase Smeeton (1985). La ecuación para κ es:

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)},$$

donde Pr (a) es el acuerdo observado relativo entre los observadores, y Pr (e) es la probabilidad hipotética de acuerdo por azar, utilizando los datos observados para calcular las probabilidades de que cada observador clasifique aleatoriamente cada categoría. Si los evaluadores están completamente de acuerdo, entonces κ = 1. Si no hay acuerdo entre los calificadores distinto al que cabría esperar por azar (según lo definido por Pr (e)), κ = 0.

El artículo pionero que introdujo Kappa como nueva técnica fue publicado por Jacob Cohen en la revista *Educational and PsychologicalMeasurement* en 1960. Hay que tener en cuenta que la Kappa de Cohen sólo mide el acuerdo entre dos observadores. Para una medida de acuerdo similar (Kappa de Fleiss) utilizada cuando hay más de dos observadores, véase Fleiss (1971). La Kappa de Fleiss, sin embargo, es una generalización para múltiples observadores del estadístico pi de Scott, y no de la Kappa de Cohen.

Ejemplo:

Se tiene un grupo de 50 personas que presentan una solicitud de subvención. Cada propuesta de subvención es analizada por dos evaluadores que anotan un "Sí" o un "No", según acepten o rechacen, respectivamente, la solicitud. El resultado del análisis de cada solicitud genera la tabla siguiente, en la que A y B denotan a cada uno de los dos evaluadores:

		B	
		Sí	No
A	Sí	20	5
	No	10	15

Los datos situados en la diagonal formada por los valores 20 y 15, representan el número de solicitudes en el que hay concordancia entre ambos evaluadores. Mientras que la diagonal formada por los valores de 10 y 5, representan los casos en los que hay discordancia entre los evaluadores.

Ahora pues, teniendo en cuenta que, de las 50 solicitudes, 20 fueron aceptadas y 15 rechazadas por ambos evaluadores. El porcentaje de acuerdo observado es:

$$\Pr(a) = \frac{20 + 15}{50} = 0.70$$

Para calcular Pr(e), es decir, la probabilidad de que el acuerdo entre evaluadores se deba al azar, se advierte que:

- El evaluador A acepta (dice "Sí") 25 solicitudes y rechaza (dice "No") 25. Es decir, el evaluador A dice "Sí" el 50% de las veces.
- El evaluador B acepta (dice "Sí") 30 solicitudes y rechaza (dice "No") 20. Es decir, el evaluador B dice "Sí" el 60% de las veces.

Por lo tanto, la probabilidad de que ambos evaluadores digan "Sí" al azar es:

$$\Pr(A) * \Pr(B) = 0.50 * 0.60 = 0.30$$

Y la probabilidad de que ambos lectores digan "No" al azar es:

$$\Pr(A) * \Pr(B) = 0.50 * 0.40 = 0.20$$

Teniendo en cuenta lo anterior, el valor de Pr(e) se calcula como la suma de las probabilidades de decir "Sí" y "No" al azar:

$$\Pr(e) = 0.30 + 0.20 = 0.50$$

Aplicando los valores de Pr(a) y Pr(e) en la fórmula de Kappa de Cohen se obtiene:

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)} = \frac{0.70 - 0.50}{1 - 0.50} = 0.40$$

Landis y Koch propusieron, y desde entonces ha sido ampliamente usada, la siguiente escala de valoración

TABLA 8 GRADO ACUERDO SEGÚN VALOR KAPPA

Kappa	grado de acuerdo
< 0,00	sin acuerdo
>0,00 - 0,20	Insignificante
0,21 - 0,40	Discreto/bajo
>0,41 - 0,60	moderado
0,61 - 0,80	Sustancial/bueno
0,81 - 1,00	casi perfecto/muy bueno

i https://es.wikipedia.org/wiki/Coeficiente_Kappa_de_Cohen

Anexo B: Resultados Aplicación Kappa Cohen y Establecimiento CI/CE

En este apéndice se indican los estudios primarios analizados (en bloques de diez), y el resultado de la clasificación de los investigadores para incluirlos/excluirlos del SLR según los criterios establecidos. Puede observarse que hay desacuerdo en tres casos (estudios 6, 8 y 9) en la Tabla 10. Se calcula el Kappa de Cohen como se indica a continuación.

		Jessica	
		Sí	No
Jorge	Sí	5	2
	No	1	2

Porcentaje de acuerdo, Pr (a)= (5 + 2) / 10 = 0.7
Probabilidad de que ambos digan que Sí al azar, P (Si) = Pr (JP) * Pr (JD) = 0.7 * 0.6 = 0.42
Probabilidad de que ambos digan que No al azar, P (No) = Pr (JP) * Pr (JD) = 0.3 * 0.4 = 0.12
Pr (e) = P (Si) + P (No) = 0.42 + 0.12 = 0.54
K = (Pr (a) – Pr (e)) / (1 – Pr (e)) = (0.7 – 0.54) / (1 – 0.54) = 0.16 / 0.46 = **0.35**

TABLA 9 ANÁLISIS DE LOS 10 PRIMEROS ESTUDIOS PRIMARIOS

#	Título	¿Se incluye?	Razón
01	Towards Recovering the Software Architecture of Microservice-based Systems	JP: SI	Habla de una herramienta, **MicroART**, capaz de recuperar la arquitectura de un sistema basado en microservicios y de un lenguaje específico del dominio capaz de representar los aspectos arquitectónicos relevantes. Es interesante saber cuáles son esos aspectos ya que nos dará una pista de qué consideran los autores que debe tener una arquitectura basada en microservicios
		JD: SI	Incluir porque abordan el estilo arquitectónico de microservicios. Los autores abordan la ingeniería inversa de la arquitectura de un sistema basado en microservicos. Han definido un metamodelo. Enfoque MDD
02	**MicroART**: A Software Architecture Recovery Tool for Maintaining Microservice-based Systems	JP: SI	Es el mismo que el #1. Vamos a leerlo para ver si existe algún aspecto añadido respecto al anterior
		JD: SI	Igual que P1 (mismos autores, misma contribución)
03	Microservices-Based Software Architecture and Approaches	JP: SI	This paper will discuss several aspects of microservices-based architecture. The characteristics of microservice-based architecture such as componentization, organization, endpoints and messaging mechanisms.
		JD: SI	Incluir, porque trata el estilo arquitectónico de microservicios y describe aspectos de su arquitectura
04	Conceptualizing a Framework for Cyber-Physical Systems of Systems Development and Deployment	JP: NO	En el abstract aparece “microservices architecture style”. Sin embargo, somos los autores del trabajo y sabemos que no se trata de describir los componentes de un estilo arquitectónico orientado a microservicios
		JD: NO	No incluir. No aborda el estilo de microserviciones, sino que se trata de un caso de aplicación del estilo en CPSoS
05	Towards microservices architecture to transcode videos in the large at low costs	JP: NO	En las conclusiones habla de “micro-services approach” a pesar de que en el abstract habla de ” microservice architecture style”. No obstante, esta segunda mención se hace en el contexto de principios de ingeniería del software. Entiendo que se habla de microservicios sin tener en cuenta aspectos arquitectónicos
		JD: NO	No incluir. No aborda el estilo de microserviciones, sino que se trata de un caso de aplicación del estilo para la codifición de vídeo
06	Designing a Smart City Internet of Things Platform with Microservice Architecture	JP: SI	Habla de estilo arquitectónico orientado a microservicios y las ventajas de este estilo frente a SOA. Habrá que leerlo pero me temo que hablará más de filosofía que de cosas concretas
		JD: NO	No incluir. No aborda el estilo de microserviciones, sino que se trata de un caso de aplicación del estilo para la implementación de una plataforma IoT para smart cities
07	Microservices tenets, Agile approach to service development and deployment	JP: SI	Compara el estilo arquitectónico SOA con microservicios y concluye que los microservicios no es un estilo aparte sino una implementación particular de SOA. Es interesante ver qué elementos

			de SOA asemeja con microservicios para deducir los componentes de este estilo arquitectónico para nuestra futura definición del mismo
		JD: SI	Incluir. Proporciona definición, conceptos, características y principios del estilo
08	Policy Enforcement upon Software Based on Microservice Architecture	JP: NO	Se habla de métricas para medir la calidad de aplicaciones basadas en microservicios. Es un trabajo centrado en la calidad del software.
		JD: SI	Incluir, pues define métricas y políticas del estilo arquitectónico
09	Benchmark Requirements for Microservices Architecture Research	JP: SI	Nos puede dar una idea de por dónde van los tiros en la investigación de arquitectura basada en microservicios
		JD: NO	Excluir. Este artículo aborda los requisitos de aplicaciones 'tipo' para ser usadas en artículo de investigación sobre microservicios. Aunque trata aspectos arquitectónicos no creo que sirva para responder las RQ
10	Differences Between Model-driven Development of Service-oriented and Microservice Architecture	JP: SI	Compara SOA con el estilo de microservicios, aunque desde la perspective de MDD. Nos puede dar indicios de los elementos constituyentes de un estilo arquitectónico basado en microservicios
		JD: SI	Incluir. Este paper caracteriza los estilos SOA y microservicios desde la perspectiva de su modelado

Como el valor de Kappa es inferior a 0.7, los dos investigadores contrastan sus análisis a la luz de la interpretación que da cada uno a los criterios de inclusión/exclusión. Resultado del mismo es que se incluyeron dos nuevos criterios, en este caso de exclusión:

- No se incluirán aquellos estudios que sean casos de uso/aplicación del estilo arquitectónico de microservicios.
- No se incluirán aquellos trabajos que traten sobre atributos de calidad del estilo arquitectónico de microservicios. Esto es una cautela para evitar que la muestra de estudios crezca demasiado.

La aplicación de estos criterios implica que no se incluyen los trabajos 6, 8 y 9. Por tanto, la clasificación de los estudios queda de la siguiente forma:

- Incluidos: 1, 2, 3, 7, 10
- excluidos: 4, 5, 6, 8, 9

A continuación, los investigadores clasifican el siguiente bloque de 10 estudios primarios tal y como queda reflejado en la Tabla 11. Téngase en cuenta que ahora los criterios de inclusión/exclusión son los siguientes:

Se incluyen:

- Aquellos estudios que traten los microservicios desde el punto de vista arquitectónico (aunque no se mencione estilo). Los que además hablen de manera explícita de estilo arquitectónico basado/orientado a microservicios se mencionarán especialmente.

Se excluyen:

- Aquellos artículos que solo hablen de microservicios sin mencionar aspectos arquitectónicos.
- No se incluirán aquellos estudios que sean casos de uso/aplicación del estilo arquitectónico de microservicios
- No se incluirán aquellos trabajos que traten sobre atributos de calidad del estilo arquitectónico de microservicios. Esto es una cautela para evitar que la muestra de estudios crezca demasiado. Se podría eliminar este criterio en una segunda fase.

Hay dos desacuerdos: estudios 16 y 17. Se calcula el Kappa de Cohen como se indica a continuación.

		Jessica	
		Sí	No
Jorge	Sí	1	1
	No	1	7

Porcentaje de acuerdo, Pr (a)= (1 + 7) / 10 = 0.8

Probabilidad de que ambos digan que Sí al azar, P (Si) = Pr (JP) * Pr (JD) = 0.2 * 0.2 = 0.04

Probabilidad de que ambos digan que No al azar, P (No) = Pr (JP) * Pr (JD) = 0.8 * 0.8 = 0.64

Pr (e) = P (Si) + P (No) = 0.04 + 0.64 = 0.68

K = (Pr (a) – Pr (e)) / (1 – Pr (e)) = (0.8 – 0.68) / (1 – 0.68) = 0.12 / 0.32 = **0.375**

Como el valor de Kappa es inferior a 0.7, los dos investigadores contrastan sus análisis a la luz de la interpretación que da cada uno a los criterios de inclusión/exclusión. Resultado del mismo es que se incluye un nuevo criterio:

- Se incluirán aquellos estudios que versen sobre lenguajes de implementación para soportar microservicios

La aplicación de este criterio implica incluir el trabajo 17. Tras revisar la aplicación de los criterios, se decide incluir también el trabajo 16. Por tanto:

Incluidos: 14, 16, 17
Excluidos: 11, 12, 13, 15, 18, 19, 20

Como puede apreciarse del valor de Kappa, estamos frente a una de las dos paradojas que afectan al Kappa de Cohen (Feinstein & Cicchetti, 1990);(Lantz & Nebenzahl, 1996)). En este caso, nos encontramos con un acuerdo observado alto (0.8) siendo que el valor de Kappa es bajo (0.375). Experiencia similar ha ocurrido con el análisis de los primeros 10 estudios primarios (acuerdo observado de 0.7 mientras que el valor de Kappa ha sido de 0.35). Este se debe a un desequilibrio sustancial en los totales marginales (horizontales/verticales) de la tabla. Este desequilibrio se origina por el predominio de los "noes" frente a los" síes" (Tabla 11). En general, cuanto más cercano a 0,5 sea el predominio (cuanto más balanceados estén los totales marginales en la tabla) mayor es el Kappa para igual proporción de acuerdos observados; dicho de otro modo, predominios muy bajos, o muy altos, penalizan el índice Kappa, debido a que en ese caso la proporción de acuerdos esperados por azar es mayor que cuando el predominio es cercano a 0,5. Algebraicamente, el valor de Kappa es mayor, dado un acuerdo observado Pr(a), cuanto menor sea el valor del acuerdo por azar Pr(e). Por tanto, si el factor de corrección, Pr(e) de acuerdo por azar es alto, esto hace disminuir el valor de Kappa sea cual sea el valor del acuerdo observado.

Se puede resolver esta paradoja obteniendo una muestra aleatoria tal que dicha muestra no tenga "tendencias" hacia uno de los dos valores: incluir/excluir. En este estudio, los estudios primarios se obtienen de diferentes "fuentes" comparando el string de búsqueda contra una BBDD. Los primeros resultados ofrecidos por el motor de búsqueda corresponden a aquellos estudios que más se ajustan (estadísticamente) al string de búsqueda; es decir, los primeros resultados son los que tienen más probabilidades de ser "incluidos", mientras que los últimos tienen más probabilidades de ser excluidos. Por tanto, sería mala praxis presentar una muestra con los primeros estudios o con los últimos estudios puesto que dicha muestra aumenta el valor de Pr (e) y por tanto disminuirá el valor de Kappa, independientemente del acuerdo observado, Pr(a).

Otra solución pasa por poner Kappa en perspectiva; es decir, calcular un máximo (K_{max}), un mínimo (K_{min}) así como un valor normal (K_{nor}). La identificación de K_{min} así como de

K_{max} define un rango de posibles valores de k para cualquier nivel de acuerdo observado dado, Pr(a). (Lantz & Nebenzahl, 1996) establecen el cálculo de las tres variables anteriores de la siguiente forma (siendo P_0 el acuerdo observado):

$K_{nor} = 2P_o - 1$

$K_{max} = P_0^2 / [(1 - P_0)^2 + 1]$

$K_{min} = (P_0 - 1) / (P_0 + 1)$ para $P_0<1$

Desviaciones significativas de k respecto a K_{nor}, en cualquier dirección, sugiere la existencia de una asimetría predominante de alguna clase, bien en la categoría de acuerdos o en la de desacuerdos. Por ejemplo, para los 10 primeros estudios primarios los valores de Kappa son los siguientes:

Valores de k para los 10 primeros estudios primarios				
P_0	K	k_{nor}	k_{min}	k_{max}
0.7	0.35	0.4	-0.176	0.45

Mientras que para los estudios primarios del 11 a 20 se han obtenido los siguientes valores:

Valores de k para los estudios primarios del 11 a 20				
P_0	K	k_{nor}	k_{min}	k_{max}
0.8	0.375	0.6	-0.111	0.615

Puede observarse una mayor distancia entre k y k_{nor} en la segunda tabla lo que indica una asimetría mayor que en la primera tabla. Y, efectivamente, para el segundo caso hay una predominancia clara de los “noes”.

TABLA 10 ANÁLISIS DE LOS ESTUDIOS PRIMARIOS 11 A 20

#	Título	¿Se incluye?	Razón
11	A Framework for the Structural Analysis of REST APIs	JP: NO	No incluir. La palabra microservicios aparece en el abstract como contexto de REST APIs. No se trata ni de arquitectura ni de estilo arquitectónico de microservicios
		JD: NO	Excluir porque solo se menciona microservicios como contexto donde REST es un soporte fundamental. No se abordan aspectos arquitectónicos del mismo, excepto aquellos que podrían considerarse porque microservicios se basan fundamentalmente en REST
12	Requirements Reconciliation for Scalable and Secure Microservice (De)composition	JP: NO	NO incluir. Habla de una metodología para romper aplicaciones monolíticas en microservicios en pro de optimizar el balance entre la escalabilidad y la seguridad del sistema. No habla de arquitectura de microservicios sino de decisiones arquitectónicas (descomposición del sistema)
		JD: NO	Excluir. Este artículo aborda la descomposición de sistemas monolíticos en microservicios. La propuesta parte de un punto de vista de ing. de requisitos donde ya se pueden tomar decisiones arquitectónicas teniendo en cuenta tradeoffs en relación a seguridad y escalabilidad. Aunque la descomposición a nivel arquitectónico es importante, no es relevante para el estilo en sí (añadir criterio exclusión)
13	Towards a Scalable Framework for Artifact-Centric Business Process Management Systems	JP: NO	No incluir. Esto es un caso de uso: se usan microservicios y dockers para desarrollar un prototipo que soporta entrega continua de instancias de ingeniería de procesos
		JD: NO	Excluir porque es un caso de aplicación del estilo arquitectónico
14	Design and Implementation of a Decentralized Message Bus for Microservices	JP: SI	SI incluir. El paper propone una implementación de lo que podría ser un conector en el estilo arquitectónico de microservicios
		JD: SI	Incluir porque este paper cubre aspectos de comunicación entre servicios que son importantes para definir un estilo. En particular, creo que está proponiendo el estilo "message bus" para solventar los problemas de comunicación de los microservicios.
15	CIDE: An Integrated Development Environment for Microservices	JP: NO	NO incluir. Se trata del desarrollo de un entorno para mejorar el desarrollo de aplicaciones basadas en microservicios. No se habla de arquitectura para microservicios
		JD: NO	Excluir porque es un short paper de solo 5 páginas donde mezcla multitud de conceptos que están ahora de moda y no creo que llegue a desarrollar ninguno. Creo que se queda a nivel de desarrollo (con agentes) y no de estilo arquitectónico.
16	Leveraging microservices architecture by using Docker technology	JP: SI	SI incluir. En las conclusiones habla de que se han introducido algunos conceptos sobre la arquitectura de microservicios y cómo Dockers ayuda a soportarlos
		JD: NO	Excluir porque es un caso de implementación del estilo con Docker

17	Data-driven Workflows for Microservices	JP: NO	NO incluir. Habla de extender el lenguaje Jolie. Este lenguaje se basa en el paradigma de microservicios. No habla de microservicios desde el punto de vista arquitectónico ni de estilo arquitectónico
		JD: SI	Incluir porque aborda aspectos del estilo, quizás a bajo a nivel.
18	Scalable microservice based architecture for enabling DMTF profiles	JP: NO	NO incluir. Es un caso de uso/de aplicación de la arquitectura basada en microservicios
		JD: NO	Excluir porque es un caso de aplicación
19	Microservices: Lightweight Service Descriptions for REST Architectural Style	JP: NO	NO incluir. El concepto de microservicio no tiene nada que ver con el que nosotros trabajamos. Este estudio es de 2010, anterior a los primeros trabajos sobre microservicios (2014). Hay que modificar los filtros de búsqueda para que esto no ocurra
		JD: NO	Creo que excluir. No es el concepto de microservicio hasta ahora tratado
20	Adaptive Service-Oriented Architectures for Cyber Physical Systems	JP: NO	NO incluir. Describen un middleware para conferir adaptabilidad a SOA para CPS
		JD: NO	Excluir porque es un caso de aplicación del estilo arquitectónico

A continuación, los investigadores clasifican el siguiente bloque de 10 estudios primarios tal y como queda reflejado en la Tabla 12. Téngase en cuenta que ahora los criterios de inclusión/exclusión son los siguientes:

Se incluyen:

- Aquellos estudios que traten los microservicios desde el punto de vista arquitectónico (aunque no se mencione estilo). Los que además hablen de manera explícita de estilo arquitectónico basado/orientado a microservicios se mencionarán especialmente.
- Aquellos estudios que versen sobre lenguajes de implementación para soportar microservicios.

Se excluyen:

- Aquellos artículos que solo hablen de microservicios sin mencionar aspectos arquitectónicos.
- No se incluirán aquellos estudios que sean casos de uso/aplicación del estilo arquitectónico de microservicios.
- No se incluirán aquellos trabajos que traten sobre atributos de calidad del estilo arquitectónico de microservicios. Esto es una cautela para evitar que la muestra de estudios crezca demasiado.

En este caso los estudios primarios provienen de dos fuentes:

- 4 de ellos son los últimos indicados en las búsquedas sobre WoS
- 6 de ellos son los primeros indicados en las búsquedas sobre IEEE

Se espera que esto constituya una muestra más equilibrada en la que no haya predominancia clara de una de las facetas.
Para este caso hay un acuerdo observado de 1. Los estudios incluidos/excluidos son los siguientes:

Incluidos: 21, 23, 26

Excluidos: 22, 24, 25, 27, 28, 29, 30

Procedemos al cálculo del Kappa de Cohen:

		Jessica	
		Sí	No
Jorge	Sí	3	0
	No	0	7

Porcentaje de acuerdo, Pr (a)= (3 + 7) / 10 = 1.0

Probabilidad de que ambos digan que Sí al azar, P (Si) = Pr (JP) * Pr (JD) = 0.3 * 0.3 = 0.09

Probabilidad de que ambos digan que No al azar, P (No) = Pr (JP) * Pr (JD) = 0.7 * 0.7 = 0.49

Pr (e) = P (Si) + P (No) = 0.09 + 0.49 = 0.58

K = (Pr (a) – Pr (e)) / (1 – Pr (e)) = (1.0 – 0.58) / (1 – 0.58) = 0.42 / 0.42 = **1.0.** Es obvio ya que Pr(a) = 1.

El grado de acuerdo es perfecto**.** Por tanto, quedan fijados los criterios de inclusión/exclusión en los términos citados anteriormente.

Se incluyen:

- Aquellos estudios que traten los microservicios desde el punto de vista arquitectónico (aunque no se mencione estilo). Los que además hablen de manera explícita de estilo arquitectónico basado/orientado a microservicios se mencionarán especialmente.
- Aquellos estudios que versen sobre lenguajes de implementación para soportar microservicios

Se excluyen:

- Aquellos artículos que solo hablen de microservicios sin mencionar aspectos arquitectónicos.
- No se incluirán aquellos estudios que sean casos de uso/aplicación del estilo arquitectónico de microservicios.
- No se incluirán aquellos trabajos que traten sobre atributos de calidad del estilo arquitectónico de microservicios. Esto es una cautela para evitar que la muestra de estudios crezca demasiado. Se podría eliminar este criterio en una segunda fase.

TABLA 11 ANÁLISIS DE LOS ESTUDIOS PRIMARIOS 21 A 30

#	Título	¿Se incluye?	Razón
21	A Microservice Based Reference Architecture Model in the Context of Enterpise Architecture	JP: SI	Incluir, por motivos obvios. Se trata de una arquitectura de referencia
		JD: SI	Incluir. Este artículo parece que va a proporcionar bloques y componentes clave así como otros aspectos arquitectónicos y recomendaciones.
22	A Reference Architecture for Real-Time Microservice API Consumption	JP: NO	NO incluir. El término microservicio hace referencia a un módulo que se encarga de gestionar REST API. No tiene nada que ver con estilo arquitectónico orientados/basados en microservicio
		JD: NO	Excluir: Parece un caso de aplicación. Este artículo aborda la actualización de modelos de datos de bbdd NoSQL en tiempo real basándose en el estilo de microservicios
23	Microservices Approach for the Internet of Things	JP: SI	SI incluir. Habla de objetivos arquitectónicos, patrones y mejores prácticas tanto de microservicios como de IoT
		JD: SI	Incluir: aunque es un caso de aplicación de microservicios a IoT tanto el resumen como las conclusiones indican que se describen patrones y buenas prácticas de microservicios
24	Lessons Learned on Systematic Metric System Development at a large IT Service Provider	JP: NO	NO incluir. El término microservicio aparece como "idea moderna". Es un trabajo de métricas SW sin relación con los microservicios. En cualquier caso, si tuviera relación, uno de los criterios de exclusión lo descartaría (el que se refiere a atributos de calidad)
		JD: NO	Excluir: caso de aplicación
25	Experience on a Microservice-based Reference Architecture for Measurement Systems	JP: NO	NO incluir. No habla sobre arquitectura de microservicios sino sobre una arquitectura de referencia para EMI
		JD: NO	Excluir: caso de aplicación
26	ICN-based Service Discovery Mechanism for Microservice Architecture	JP: SI	SI incluir. Se trata de un trabajo para descubrir servicios con los que otros microservicios deben comunicarse. Es parte esencial del diseño de un conector software en el estilo basado en microservicios.
		JD: SI	Incluir: este artículo aborda el descubrimiento de servicios, así que es un aspecto arquitectónico a tener en cuenta para el desacoplamiento de servicios es un artículo de 3 páginas, tendremos que considerar e incluir short papers porque supongo que dará muy poquita información
27	Microservices	JP: NO	NO incluir. Es una entrevista en la que se habla de microservicios en general. No se habla de arquitectura
		JD: NO	Excluir: Es un artículo de opinión. No podemos seguir el criterio de leer abstract y conclusiones.

28	Docker Container Security via Heuristics-Based Multilateral Security- Conceptual and Pragmatic Study	JP: NO	NO incluir. Es un artículo sobre seguridad en cloud. Es un caso de uso de microservicios
		JD: NO	Excluir: aborda un aspect arquitectónico como la seguridad pero no centrado en el estilo de microservicios sino en cloud y containers.
29	Exploring the impact of situational context – A case study of a software development process for a microservices architecture	JP: NO	NO incluir. El término microservicios aparece de manera casual. Es un artículo sobre procesos de desarrollo
		JD: NO	Excluir: utiliza microservicios como caso de estudio
30	Exploiting Interoperable Microservices in Web Objects Enabled Internet of Things	JP: NO	NO incluir. Es un caso de uso de microservicios
		JD: NO	Excluir: es un caso de aplicación

A continuación, se aborda el análisis de los siguientes 20 estudios seleccionados (Tabla 13).

TABLA 12 ANÁLISIS DE LOS ESTUDIOS PRIMARIOS 31 A 50

#	Título	¿Se incluye?	Razón
31	Towards Integrating Microservices with Adaptable Enterprise Architecture	JP: SI	En la conclusión los autores indican: “In this paper, we presented architectural properties of microservices”. Aunque el artículo integra microservicios en una arquitectura de enterprise, las propiedades arquitectónicas pueden ser interesantes
32	Highly-Available Applications on Unreliable Infrastructure: Microservice Architectures in Practice	JP: NO	Excluir. Es un caso de estudio de cómo conseguir disponibilidad utilizando una arquitectura basada en microservicios
33	Performance Engineering for Microser vices: Research Challeng es and Directions	JP: NO	No, habla de “performance engineering for microservices”. Es un uso de los microservicios
34	Designed and Delivered Today, Eroded Tomorrow? Towards an Open and Lean Architecting Framework Balancing Agility and Sustainability	JP: NO	Excluir. Es un keynote
35	Cross-ISA Container Migration	JP: NO	No, habla de microservicios de pasada. Los autores indican en el abstract (es lo único que hay) que “Our work focuses on the mechanism required by the middle-ware to implement a power optimization policy”
36	Automation of the Incremental Integration of Microservices Architectures	JP: NO	Excluir. Es un caso de aplicación: general view of a method for the incremental integration of microservices into cloud applications

37	Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection	JP: NO	No, es un caso de uso. En el abstract, los autores indican: “We present a case study of a commercial product for counter-party credit risk implemented as a distributed microservice architecture.”
38	Investigation of Impacts on Network Performance in the Advance of a Microservice Design	JP: NO	Excluir. Es un benchmark para evaluar el impacto de rendimiento de una arquitectura de microservicios
39	Microservices Identification Through Interface Analysis	JP: NO	No. El paper habla de identificar microservicios. No tiene relación con aspectos arquitectónicos
40	Revisiting the Anatomy and Physiology of the Grid	JP: NO	Excluir. No tiene nada que ver con el tema que nos ocupa. debe ser porque aparece la palabra architecture en el texto
41	The Database-is-the-Service Pattern for Microservice Architectures	JP: SI	Incluir. Se propone que la Base de Datos sea un componente de la arquitectura de microservicios. Será interesante ver cómo lo hacen los autores.
42	A Microservice Architecture Use Case for Persons with Disabilities	JP: NO	Excluir. Es un caso de uso: A prototype of such infrastructure has been developed and its microservice architecture has been described in the paper
43	Decision Models for Microservices: Design Areas, Stakeholders, Use Cases, and Requirements	JP: NO	Excluir. El objetivo es identificar stakeholders, use cases, and requirements for decision models for microservices. Es un caso de aplicación sobre una arquitectura basada en microservicios
44	An Experience of Constructing a Service API for Corporate Data Delivery	JP: NO	Excluir. No hay mención, salvo en el texto, a microservicios y menos a arquitectura con ese estilo
45	Medley: An E vent-Driven Lightweight Platform for Service Composition	JP: NO	Excluir. No habla de arquitectura de microservicios. Habla de cómo componer/orquestar servicios
46	Visual modeling of RESTful conversations with RESTalk	JP: NO	Excluir. No hay mención, salvo en el texto, a microservicios y menos a arquitectura con ese estilo
47	Software Architecture for the Cloud – A Roadmap Towards Control-Theoretic, Model-Based Cloud Architecture	JP: NO	Excluir. Es un estudio sobre propiedades del cloud
48	Toward an Anti-fragile e-Government System	JP: NO	Excluir. No hay mención, salvo en el texto, a microservicios y menos a arquitectura con ese estilo
49	Towards a Framework for Building SaaS Applications Operating in Diverse and Dynamic Environments	JP: NO	Excluir. Tiene que ver con SaaS. Como indican los autores en las conclusiones: "In this paper, we presented an approach to offer quality attributes of a SaaS application as scriptable resources"
50	Anti-fragile Cloud-Based Telecom Systems	JP: NO	Excluir. No tiene nada que ver con el tema que nos ocupa. Además, la aparición del término microservices se asocia con SOA

TABLA 13 ANÁLISIS DE LOS ESTUDIOS PRIMARIOS 51 A 70

#	Título	¿Se incluye?	Razón
51	Python, PyGame and Raspberry Pi Game Development	JP: NO	Es un libro sobre lenguajes de programación. No tiene nada que ver con el tema
52	Conclusión. Capítulo del libro anterior	JP: NO	Mismo comentario que el #51
53	Enabling DevOps Collaboration and Continuous Delivery Using Diverse Application Environments	JP: NO	El término microservicio aparece en el texto de una manera lateral, asociada a DevOps.
54	User-Defined Functions. Capítulo del libro #51	JP: NO	Es un capítulo del libro #51
55	All the Services Large and Micro: Revisiting Industrial Practice in Services Computing	JP: NO	NO incluir. The goal of this study was to provide a peek into the current state of practice in services computing- Los microservicios son un ejemplo
56	FedUp! Cloud Federation as a Service	JP: NO	No incluir. Es un intento de que el proceso de unirse a una federación de clouds, se gestione como un servicio más. No tiene nada que ver con el tema de estudio
57	EcoData: Architecting Cross-Platform Software Ecosystem Applications	JP: NO	No incluir. La palabra microservices aparece de manera lateral. habrá salido en la búsqueda por “reference architecture”
58	Migrating to Cloud-Native Architectures Using Microservices: An Experience Report	JP: SI	Incluir. Es una migración de una arquitectura monolítica a una basada en microservicios. Por tanto, se espera que se defina la topología y la forma de los componentes/conectores de la arquitectura resultante así como los protocolos de comunicación
59	Approaches to the Evolution of SOA Systems	JP: NO	No incluir. Es un caso de estudio. We also presented emerging trends in supporting the maintenance and evolution of SOA systems, including microservices and knowledge-based support
60	The Integration Technologies and Tools for IoT Environments	JP: NO	Excluir. No está documentada su procedencia. Tampoco aparece el término microservice en el texto
61	Integrating Personalized and Accessible Itineraries in MaaS Ecosystems Through Microservices	JD: NO	Excluir: Es un caso de aplicación de MAS a Mobile app tipo Crowdsensing and crowdsourcing

62	Docker cluster management for the cloud – survey results and own solution	JD: NO	Excluir: aborda microservicios de forma transversal para el uso de Docker en la nube. De hecho no aparece la palabra microservicios ni el abstract ni en las conclusiones
63	Contextual Factors of Architectural Strategy for Complex Systems	JD: NO	Excluir: no aborda el estilo de microservicios, solo lo menciona
64	Principles of Continuous Architecture	JD: NO	Excluir: es una artículo generalista, solo se menciona microservicios como solución para "architect for change"
65	Real-Time HazMat Environmental Information System: A micro- service based architecture	JD: NO	Excluir: es un caso de aplicación
66	Introduction to Continuous Architecture	JD: NO	Excluir: solo se menciona microservices a modo de ejemplo
67	Continuous Architecture in Practice: A Case Study	JD NO	
68	A microservice-based middleware for the digital factory	JD NO	Excluir: es un caso de aplicación
69	S-InTime: A social cloud analytical service oriented system	JD NO	Excluir: es un caso de aplicación
70	Resource sharing in mobile cloud-computing with CoAP	JD NO	Excluir: es un caso de aplicación/evaluación del rendimiento

TABLA 14 ANÁLISIS DE LOS ESTUDIOS PRIMARIOS 71 A 90

#	Título	¿Se incluye?	Razón
71	Software defined P2P architecture for reliable vehicular communications	JD: NO	Excluir: es un caso de uso/implementación
72	A Context Model and Policies Management Framework for Reconfigurable-by-design Distributed Applications	JD: SI	Incluir: no estoy muy segura, pero desde luego no hace matching con ningún criterio de exclusión. Lo único que es un short paper
73	Automated deployment of a microservice-based monitoring infrastructure	JD: NO	Excluir: caso de aplicación
74	A performance comparison of container-based technologies for the Cloud	JD: NO	Excluir: Microservicio como tecnología, no como estilo arquitectónico
75	Auto-scaling and Adjustment Platform for Cloud-based Systems	JD: NO	Excluir: el término aparece en las referencias únicamente
76	Microservices Architecture Enables DevOps	JD: SI	Incluir: no va a proporcionar mucha información sobre el estilo, pero sí que trata la arquitectura de microservicios
77	Incremental Integration of Microservices in Cloud Applications	JD: SI	Incluir
78	An Agent-Based Composition Model For Semantic Microservices	JD: NO	Incluir: proponen un modelo y arquitectura de referencia para descubrir y componen microservicios
79	Apache Airavata as a Laboratory. Architecture and Case Study for Component-Based Gateway Middleware	JD: NO	Excluir: Microservicio como tecnología, no como estilo arquitectónico
80	An architecture for self-managing microservices	JD: NO	Excluir: caso de aplicación
81	Native Cloud Applications: Why Monolithic Virtualization Is Not Their Foundation	JD: NO	Excluir: es un caso de uso/implementación para explotar las ventajas de microservicos en app cloud nativas
82	Extraction of Microservices from Monolithic Software Architectures	JD: SI	Incluir: modelo formal de extracción de microservicios
83	Microflows: Enabling Agile Business Process Modeling to Orchestrate Semantically-Annotated Microservices	JD: SI	Incluir: orquestación de microservicios anotados semánticamente
84	Infrastructure Cost Comparison of Running Web Applications in the Cloud using AWS Lambda and Monolithic and Microservice Architectures	JD: NO	Excluir: comparación de varios estilos arquitectónicos en términos de costes de infraestructura
85	Multi-perspective Digitization Architecture for the Internet of Things	JD: NO	Excluir
86	Decision-Controlled Digitization Architecture for Internet of Things and Microservices	JD: NO	Excluir
87	Redefining a Process Engine as a Microservice Platform	JD: SI	Incluir: aborda la vista de proceso de la arquitectura (orquestación)
88	Dohko: An Autonomic System for Provision, Configuration, and Management	JD NO	Excluir: no aborda el estilo arquitectónico de microservicios

	of Inter-Cloud Environments based on a Software Product Line Engineering Method		
89	Using Microservices and Software Product Line Engineering to Support Reuse of Evolving Multi-tenant SaaS	JD NO	Excluir: re-engineering a legacy application to develop a multi-tenant SaaS using SPL techniques and microservices architecture
90	API governance support through the structural analysis of REST APIs	JD NO	Excluir

Anexo C: Extracción Datos Estudios Primarios

A continuación, se detalla los datos que han sido extraidos por cada estudio primario basándose en el formulario descrito anteriormente. Mediante este formulario, se intenta sintetizar los aspectos en arquitectura en microservicios.

P01

(Granchelli et al., 2017)

Estudio primario que presenta un prototipo llamado MicroART, para extraer información de un sistema basado en microservicios y realizar una agrupación de modelos en niveles más abstractos, que según el autor definen una arquitectura y un DSL.

El prototipo tiene dos fases, recuperación de la arquitectura (architecture recovery), partiendo de un GitHub repositorio con código fuente, más las referencias de los contenedores del sistema en ejecución, y otra de refinamiento (architecture refinement).

Define un metamodelo con 7 metaclases: Product (root), Microservice, Cluster, Developer, Team, Interface, Link, serviceType.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo

- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Con respecto a los componentes: Microservicios (Metaclase):

- Independientes unidades de trabajo
- Diseño utilizando “bounded context”, que combinan relativas funcionalidades.
- Atributos: Host (IP), Type (functional, infrastructural).

Cluster (Metaclase): Agrupar varios servicios (Ej: funcionales y infraestructura).

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos

Con respecto a las comunicaciones, se define dos metaclases

- Interface (Metaclase): Representa un endpoint en la comunicación, con puerto entrada/salida, con relaciones (expose/require) con la metaclase Microservice.
- Link (Metaclase): Conecta dos interfaces
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

Se presenta varios paradigmas que han sido utilizados por la herramienta MicroArt, para generar una arquitectura, con el sistema ya implementado.

1. MDRE (Model-driven Reverse Engineering): Técnicas de modelado y de ingeniería inversa, para convertir el ya existente código fuente en un nivel de abstracción mayor.

2. Extract-Abstract-Present: Como indica el nombre, extraer la info del código fuente, agrupar y filtrar dicha información, y presentarla de forma entendible.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Frameworks

1. ARMIN: Similar tool para reconstruir arquitecturas en microservicios desde el código fuente, sin tener en cuenta el análisis dinámico.

2. ARM: Técnica con el mismo objetivo, basada en cuatro fases, pero orientada a patrones diseño.

3. SOA-oriented arch: Parecida a la que propone el artículo, utilizando UML.

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No se menciona ningún estilo arquitectónico o arquitectura de referencia.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Salvo la definición de las metaclases y sus relaciones, tampoco menciona un lenguaje para describir MSA.

P03

(Bakshi, 2017)

Estudio primario que analiza los aspectos arquitectónicos desde el punto de vista de desarrollo y la operación: Containerization, Service Communications, y otros aspectos más operacionales, como IPC (Inter-Process communication) y Service Registry-Discovery.

También define una metodología en base a 12 factores a considerar para tecnologías microservicios: Codebase, Dependencies, Config, Backing Services, Build Release and Run, Process, Port Binding, Concurrency, Disposability Development & Production, Logs, Admin. Process,, etc..

- Si define un estilo arquitectónico basado/orientado a microservicios
    - Nombre dado al estilo
    - ¿define/caracteriza los tipos de componentes?
    - ¿Define/caracteriza los tipos de conectores/relaciones?
    - ¿Define/caracteriza las restricciones topológicas?
    - Dominio o contexto de aplicación del estilo

- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
    - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Microservicios (componentes arquitectura): independientes unidades de trabajo que definen una lógica de negocio.

1. Independientes en el despliegue -> Zero coordinación con el resto servicios

2. Múltiples clientes deben ser soportados por la arquitectura

3. Patron DDD (Domain Driven Design): Division de un dominio complejo (aplicación) en múltiples "bounded context" (microservicios) que define los limites aplicando la lógica de negocio y mapea las relaciones entre dichos "bounded context".

5. Elementos o características de las unidades de trabajo: CodeBase (uno por cada app), backing services (descentralización del almacenamiento), especificación explicita dependencias.

- Indicar cuál es el protocolo de comunicación entre los diferentes elementos (y aspectos de coordinación)

1. Dump pipes (redes simples que transmiten información sin discriminar)

2. Smart endpoints (Procesa los datos y aplica la lógica. Consume y produce msj)

3. Protocolos: Advanced Message Queuing Protocol (AMQP), HTTP based REST or Thrift)

5. Patrones Distribución Datos y Tareas: IPC (Inter-process communication)

- Request/sync response (HTTP based REST or Thrift) o Request/async response (1 to1):

Conecta clientes con servidores (RPC: remote procedure call)

-Publish–subscribe (1 to n): Conecta publishers con suscriptores

- Qué estructura topológica guardan entre sí dichos elementos
- Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

1. Contenedores: Componentes o instancia de la app donde se ejecuta el servicio.

2. Config: Configuracion por cada despliegue, y asignación de procesos a tipos de trabajo. Los logs son manejados por el entorno de ejecución, no por la instancia del servicio.

3. Puertos: Asignación puertos por donde se reciben las peticiones en cada unidad ejecución.

4. Infraestructura o Tecnología (Intercambio mensajes): RabbitMQ or ZeroMQ (async), Apache Kafka y Apache ActiveMQ

5. Service Registry-Discovery (Apache Zookeeper, etcd):

- Service Registry: API gestión para dar de alta y baja servicios (self-registration o a través de otro componente intermedio).

- Service Discovery: API para consultar servicios en la base de datos disponibles (Client-Side : Cliente consulta el registro de servicios, selecciona una instancia y realiza la petición, o Service-Side: Se realiza a través de un router)

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

En este artículo, se habla de aspectos arquitectónicos, pero al mismo tiempo subraya la ausencia de estándares.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No menciona ningún DSL.

P07

(Zimmermann, 2017)

Estudio primario que realiza una comparativa entre las características de aplicaciones en microservicios con las aplicaciones que utilizan SOA. Define siete principios que caracterizan a los microservicios, y realiza un estudio comparativo sobre las dos principales definiciones acerca de microservicios (Martin-Fowler & Newman).

Si define un estilo arquitectónico basado/orientado a microservicios

- Nombre dado al estilo
- ¿define/caracteriza los tipos de componentes?
- ¿Define/caracteriza los tipos de conectores/relaciones?
- ¿Define/caracteriza las restricciones topológicas?
- Dominio o contexto de aplicación del estilo

- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

API Gateway***:*** Ofrece diferentes API's por cada cliente, enrutando la petición en uno o dos sentidos en la comunicación entre servicios.

Microservices, Service Discovery.

- Indicar cuál es el protocolo de comunicación entre los diferentes elementos

RESTful *(Representation State Transfer) over HTTP*: Describe cualquier interfaz (métodos y reglas) entre sistemas que utilizan directamente *HTTP (Hypertext Transfer Protocol)*

AMQP *(Advanced Message Queuing Protocol): Protocolo a nivel aplicación con* orientación a mensajes, encolamiento ("queuing") y enrutamiento tanto punto a punto, como publish/subscribe.

- Qué estructura topológica guardan entre sí dichos elementos
- Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

SOA: Service Provider, Consumer, Contract, ESB

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Plataformas de desarrollo y despliegue en plataformas Cloud: Sprint Boot y Cloud

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

Para el autor no existe una arquitectura en microservicios, de la misma forma que está diseñada la arquitectura orientada en servicios SOA. Son estilos arquitectónicos similares pero con ciertas diferencias.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No define lenguaje.

P10

(Rademacher, Sachweh, & Zundorf, 2017)

Estudio primario que establece las diferencias entre SOA y MSA desde una clasificación jerárquica en base a principales conceptos SOA (Service, Interaction, Execution Context). Intenta dar una aproximación de un modelo arquitectónico para MSA utilizando MDD (Model-Driven Development).

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombres dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

. Service (componente)

. Service Granularity: MSA debe lidiar con el problema de granularización de los componentes, para no exceder en las interacciones entre ellos.

. Interface Abstraction: MSA impone que todos los servicios usen el mismo formato y estructura del mensaje.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos (interfaz y interacciones)

. Protocols: REST over HTTP (one to one), Publish/Subscribe (one to many)

Mismo protocolo por ambas direcciones en la comunicación (ya sea para interacción externa y/o interna) -> Protocol-aware communication

. Choreography: Service Call Chains

. API Gateway: Agrupa interfaces de los microservicios.

Execution Content: Infraestructura que provee la interacción entre los servicios.

  - Qué estructura topológica guardan entre sí dichos elementos

- Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

Estilos orientado a eventos

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

MDD, es un paradigma que usa modelos para el desarrollo de sistemas informaticos. Utiliza lenguajes (DSL) para definir los distintos niveles de abstracción (modeling languages) usando modelos que pueden ser automáticamente transformados en objetos y/o codigo ejecutable (model transformation).

Implicaciones MDD para MSA:

- Metamodelos ->

SOA-MDD metamodelo se basa en: service, operation, message, port and behavior.

MSA se espera modelado en:

- Comportamiento: Service Call Chains (Coreography)

- Integración y Operación.

- Puertos: Añadir conceptos técnicos (protocolo, formato mensaje, tecnología contenedora)

- Modelling tools -> Necesidad de soportar cambio de modelos (API Gateway, por el cambio de la interface de los servicios, Microservicio, por el ajuste a un área de negocio) y modelos distribuidos que se apoyan en repositorios y sistemas de versiones de los MS.

- Model transformation -> No es relevante para MSA por las pocas lineas de código, pero si es importante la service operation engineering

- Modelling languages -> Por la independencia de los lenguajes entre los servicios, un modelado debe considerar solo aspectos como las interfaces, comportamiento o despliegue. Necesidad de incluir elementos técnicos y operacionales (asignación protocolos a servicios, configuración contenedores, etc.).

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No existe lenguaje arquitectónico orientado en microservicios.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No, además se menciona explícitamente que no existe publicaciones acerca del modelado de lenguajes para MSA-MDD.

P14

(Kookarinrat & Temtanapat, 2016)

Ofrece un middleware de comunicación descentralizado “Decentralized Message Bus” como medio de comunicación entre los servicios dispersados por la red. Define los tipos de mensajes que son intercambiados entre los nodos, y los elementos arquitectónicos incluyendo el bus.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Interfaz publica API: Define el interfaz de un servicio al bus de mensajes. Utiliza HTTP based RESTful como protocolo de comunicación, y es llamada mediante una URL, con información sobre: namespace (register/unregister service), topic (subscribe a topic to the message bus) y message (tipo: reqRes, pubSub, reqNoti)

Load Balancer**:** Distribuye la carga entre servicios, ofreciendo cierta disponibilidad ya que controla todas las instancias creadas de un servicio que comparten el mismo namespace. Usa el algoritmo round-robin.

Messaging: Define el intercambio de mensajes ofreciendo la infraestructura que soporta los tres tipos de mensajería.

Service Discovery Decentralized (Serf): Servicio descentralizado de descubrimiento de servicios, donde todos los nodos (que actúen como Service Discovery) pueden ser reemplazados por otros, para evitar tener un único punto de fallo (como los Service Registry en los de tipo centralizados). Pueden provocar inconsistencia en los datos, por lo que es necesario tiempo para sincronizar entre los distintos nodos.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos

SWIM protocol: Mantiene memberships, facilita el descubrimiento de servicio mediante la detección/diseminación de fallos en un cluster. Es una implementación de “gossip” protocolo, donde cada nodo le comunica al siguiente el fallo de uno de ellos. Tiene menos tráfico de red que el mecanismo Broadcast, donde se envía un mensaje a todos los nodos de la red.

HTTP protocol: Para la comunicación con las API públicas de los distintos servicios (Metodos: put, delete, etc…)

  - Qué estructura topológica guardan entre sí dichos elementos

Inter-services Communication: Según la relación entre los servicios y el modo de comunicación (síncrono o asíncrono) se presentan los siguientes tres tipos de comunicación entre ellos:

. One-to-One Sync Request/Response

. One-to-One Async Notification

. One-to-Many Publish/Subscribe

  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

Publish/Subscribe mecanismo, implementado mediante el Message Bus Decentralized.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

Habla de una arquitectura en microservicios donde existe una capa intermedia para la comunicación entre los distintos nodos o microservicios "Message Bus Decentralized". Se centra en la descripción del funcionamiento del middleware cuando se registra o actualiza un servicio, se detecta un fallo y cuando se comunican entre ellos. No menciona aspectos estructurales o sintaxis de un estilo arquitectónico.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No menciona un lenguaje para estilo arquitectónico.

P16

(Jaramillo, Nguyen, & Smart, 2016)

Estudio primario que describe la tecnología Docker y sus componentes, así como sus principales características que se adaptan como solución para las aplicaciones en microservicios.

- Si define un estilo arquitectónico basado/orientado a microservicios
    - Nombres dado al estilo
    - ¿define/caracteriza los tipos de componentes?
    - ¿Define/caracteriza los tipos de conectores/relaciones?
    - ¿Define/caracteriza las restricciones topológicas?
    - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
    - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Describe una arquitectura cliente/servidor:

- Docker Client: Se comunica con el docker host a través de una serie de comandos.

- Docker Host: Está compuesto de un demonio, que escucha las peticiones del cliente, de uno o varios contenedores, que han sido instanciados a través de unas imágenes que son almacenadas en un repositorio.

- Comunicación: Se comunican a través de sockets utilizando un conjunto de RESTful API

    - Indicar cuál es el protocolo de comunicación entre los diferentes elementos
    - Qué estructura topológica guardan entre sí dichos elementos
    - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor…

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Describe el proceso de un modelo operacional basado en contenedores.

1. Alertas Webhook GitHub: Cambio código fuente en el repositorio lanza notificaciones a los suscriptores (developer, tester, etc.)

2. Travis Test Framework: Tests automáticos son ejecutados para probar los cambios en el código fuente asociado al servicio (Tools: jslint, istanbul, mocha).

3. WebhookListener: Escucha peticiones de Webhook Github y desencadena una serie de acciones: crea una imagen, registra la imagen para ser usada como contenedor de dicho servicio.

4. Chageback-orchestrate: Conjunto de scripts para replicar y configurar nuevos entornos de ejecución.

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No menciona ningún estilo arquitectónico sobre microservicios.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Tampoco define ningún lenguaje.

P17

(Safina, Mazzara, Montesi, & Rivera, 2016)

Jolie (*Java Orchestration Language Interpreter Engine*), es un lenguaje de programación basado en el paradigma orientado a servicios, y que se ajusta tanto en el desarrollo como en el despliegue de sistemas basados en microservicios. Cada programa es un servicio que se comunica a través de mensajes (ofrece una capa abstracción de comunicación).

Enriquecimiento de Jolie a través de la inclusión de nuevos operadores, como el operador de selección (choice operator), para permitir el cambio en el flujo de control que considera previamente la estructura y tipo de mensaje (data-driven) evitando un procesamiento posterior a la recepción (process-driven). De esta forma, se crean funciones genéricas que se comportan según el argumento de datos que se pasa a la función.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombres dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan
  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Define los componentes del lenguaje de programación:

- Deployment: Contiene las directivas para la orquestación y la comunicación entre los servicios (Interfaces, Message Type, Ports).

-Behaviour: Funcionalidad del servicio (Procedure: Init, Define)

- Communication: One-way, Request/Response

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No habla de estilo arquitectónico.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Jolie, lenguaje para la implementación y orquestación de servicios, que constituye una alternativa a XML-based lenguajes para la orquestación de servicios web como WS-BPEL (*Business Process Execution Language*). Es un intérprete implementado en Java formado de tres componentes: un parseador que produce OOIT (*Object Oriented Interpretation Tree*), un entorno de ejecución que ejecuta dichos objetos y un core para la comunicación entre componentes. La capa de comunicación ofrece ”*channels*” , identificados por “*resource path, operation, msg content”*, y se encargan de la recepción/envío de mensajes así como codificación/decodificación utilizando distintos protocolos y por distintos medios.

P21

(Yale Yu, Silveira, & Sundaram, 2016)

Estudio primario que define los conceptos de arquitectura y los componentes de un sistema basado en microservicios en el marco de una empresa.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Diseño del microservicio en arquitectura en 3 capas (3-tiered app): interface, business logic, data persistence.

*Microservicios proveedores API’s NO tienen interfaz usuario.

Establece “key building Blocks”: Microservice Consumer, API Proxy, Domain Model (conjunto microservicios), API Registry, y otras que tienen más significado en un entorno empresarial.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos

Interfaz microservicio: API

Protocols: HTTP (REST, SOAP)

Data Exchange: XML, JSON

API Proxy: Establece una capa intermedia entre el cliente o consumidor y la API

microservicio. Enruta, gestiona sesiones, control acceso, seguridad, etc.,

API Registry: Registro y descubrimiento de los microservicios (comunicación con API Proxy)

    - Qué estructura topológica guardan entre sí dichos elementos
    - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…
- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No se menciona ningún estilo arquitectónico propio. Más bien, intenta cubrir la carencia de estilo arquitectónico mediante una arquitectura ya existente (3-tiered app).

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Nada sobre DSL ni ADL.

P23

(Butzin, Golatowski, & Timmermann, 2016)

Estudio que analiza los distintos patrones de diseño en sistemas de microservicios que pueden ser aprovechables para sistemas IoT (servicios independientes y pequeños: sensor, actuador, etc..), dado a las características comunes que presentan (*Lightweight communication, Independent deployable SW, Minimum of centralized management, Independent development techniques and technologies*).

Si define un estilo arquitectónico basado/orientado a microservicios

    - Nombre dado al estilo
    - ¿define/caracteriza los tipos de componentes?
    - ¿Define/caracteriza los tipos de conectores/relaciones?
    - ¿Define/caracteriza las restricciones topológicas?
    - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
    - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Arquitectura Bottom-up para IoT: Múltiples dispositivos heterogéneos (distintos vendedores) que necesitan ser interoperables entre ellos.

Arquitectura Top-Down para MS: Descomposición de un sistema en partes que se comunican mediante un mecanismo liviano.

Hexagonal Arquitectura (utilizando el patrón ports & adapters*): App Domain, Mediation, Ports: rontend Port, Integration Port, Management Port, Database Port, Cloud Port*.

    - Indicar cuál es el protocolo de comunicación entre los diferentes elementos

DPWS (Devices Profile for Web Services): Provee un servicio mensajería, el descubrimiento, la descripción y el evento de servicios web seguros en dispositivos con recursos limitados.

CoAP (Constrained Application Protocol): Protocolo a nivel de aplicación que permite la conexión web entre dispositivos IoT (diseñado para comunicación M2M sin intervención humana).

    - Qué estructura topológica guardan entre sí dichos elementos
    - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor.
- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Self-containment: Servicios (o dispositivos en IoT) deben contener todo lo que necesitan para cumplir su objetivo (logic, front-end, back-end, libraries). Facilita el escalado individual mediante la ejecución de múltiples instancias, el mantenimiento.

Service versions: Se tratan patrones "Inmutable Server Pattern", que una vez que la aplicación es puesta en operación no es modificable, u otros patrones más alternativos donde coexisten dos distintas versiones por un periodo de tiempo. Podría ser aprovechado para IoT para actualizar versiones con mínimo riesgo.

Monitoring &. Fault Handling: A través de patrones tales como *Health Status y Circuit Breaker*.

Container: Facilidad para el despliegue, pruebas y escalabilidad de microservicios y fuera de alcance para IoT, por la limitación de recursos HW en dispositivos (salvo para routers, smartphones o small PC's)

Orquestation vs Coreography: Coreografia para microservicios, donde cada nodo escucha eventos (event-driven), y realiza su parte si es necesario. Orquestación para IoT, donde existe un nodo centralizado que controla los servicios que son llamados, ya que es más fácil de implementar. Como excepción existe el MQTT (*Message Queue Telemetry Transport Protocol),* que fomenta el uso de una comunicación orientada a eventos.

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No existe una arquitectura en microservicios, salvo la arquitectura hexagonal donde se aplica el concepto de puertos para la conexión con distintos aspectos relacionados.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No define lenguaje.

P26

(Long et al., 2017)

Estudio primario que describe un mecanismo más eficiente para el descubrimiento de servicios por la red. Realiza la comparativa entre los sistemas ISD's existentes con el propuesto ISD (*Information-centric Service Discovery)* basado en el concepto ICN (*Information-centric Networking*), que separa las instancias de su ubicación a través de los nombres de servicios. De esta forma se consigue una reducción en los registros de los nombres de servicios, al mantener solo el nombre y no la ubicación (que se provee mediante el balanceador de carga), y en consecuencia mayor agilidad en la búsqueda y en los cambios.

- Si define un estilo arquitectónico basado/orientado a microservicios

  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan.

Service Registry & Discovery basado en tecnología ICN, para búsqueda de componentes.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos (y aspectos de coordinación)
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

1) Tecnologías ISD:

- RAFT (etcd): Demonio en cada nodo del cluster que contiene pares key-value para descubrimiento servicios en contenedores Linux.

- Consul: Basado en agentes que se comunican con los servidores Consul, los datos son almacenados (key-value) y replicados para mantener el estado de los servicios.

- Synapse: Contiene un componente HAProxy que enruta peticiones de servidores de aplicaciones a proveedores de servicios del cluster. Además, tiene "watchers" para detectar cambios en la ubicación de los servicios y actualizar el HAProxy.

- Docker Swarm: Cada Swarm server almacena key-value registros para almacenar ubicación servicios (parecido DNS). En cada nodo existe un agente que notifica los cambios en la ubicación.

Todos los mecanismos anteriores se basan en la búsqueda de servicios por ubicación más que por técnicas basadas en el nombre.

- ICN-Based Service Discovery: Utiliza tablas de enrutamiento FIB, donde se asocia el nombre del servicio (ISD Name) y el próximo router cercano (NextHop), para enrutar la petición del cliente. Se utiliza la técnica "longest-prefix matching" para buscar la coincidencia del nombre del servicio.

2) Componentes ICN:

NDN System: Nodos donde se guarda la tabla FIB.

FIB: Tabla que asocia ISD Name-uCost-NextHop.

uSLB: Balanceador de carga, que monitoriza las instancias de los servicios y devuelve una instancia con su ubicación como respuesta a una búsqueda de un nombre del servicio propagada por los nodos de la red.

Protocolo: Name-based routing protocolo, para enrutar la petición del cliente a la instancia del servicio disponible.

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No menciona ningún estilo arquitectónico ni arquitectura referencia.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No hay ningún lenguaje asociado puesto que no se menciona nada sobre arquitectura en microservicios.

P31

(Bogner & Zimmermann, 2016)

Define una aproximación para tratar el problema de arquitecturas micro-granular, basado en EA-Mini Descriptions (metamodelos, modelos y datos), que son basado en el estándar MOF (Meta Object Facility), definiendo cuatro capas estructurales para una arquitectura en microservicios. También menciona un método de integración de microservicios en un metamodelo de una arquitectura empresarial.

Hay una referencia del autor Gary Olliffe, que separa la arquitectura global en dos grupos: "inner architecture" y "outer architecture".

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombres dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan
  - ✓ Inner Architecture: Es el nivel arquitectónico que define los microservicios, con una capa de balanceo de carga, y otra de entorno de ejecución.
  - ✓ Outer Architecture: Es el nivel arquitectónico que engloba el anterior aportando otras capas de comunicación con el cliente y con el entorno de operación.
    - Management: Es la capa de comunicación con el cliente que ofrece servicios tales como "Service Discovery", Routing (API Gateway), Config)
    - Inner Architeture
    - Messagging Channels: Es la capa de comunicación entre microservicios. Ofrece un middleware de la capa anterior (donde se definen y se instancian los servicios en su entorno de ejecución) con la capa operacional.
    - Operational: Es la capa que ofrece capacidades operacionales, tales como, automatización (ligado al concepto DevOps), y la monitorización de los microservicios.
  - ✓ Propuesta EA-Mini descriptions: Define las capas siguientes (basado en MOF)
    - M0 -> Run Time Data
    - M1 -> Meta data & Inner Architecture model (components, com. channels)

  Estos dos anteriores construyen un "Cell Metaphor"

    - M2 -> Architecture Metamodel & Ontology (tipos, propiedades y relaciones entre entidades)

"Body Metaphor" -> Combinación de varias celdas "Cell"

- M3 -> Abstract Language & Semantic representation
  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos (interfaz y interacciones)
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

Estilos orientados a eventos.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

I. *Enterprise Architecture Management* (EAM): Define un conjunto de estándares, frameworks, tools, modelos de negocio para adaptar las empresas a la digitalización.

II. *Enterprise Services Architecture Reference Cube* (ESARC): Es una arquitectura de referencia para las empresas digitales, dividida en ocho dominios. Un refinamiento de EAM que se ajusta más a servicios cloud y micro-granular arquitecturas.

Los dominios donde se define los microservicios, son ISA (Information System Architecture) y TA (Technology Architecture).

III. *Enterprise Services Architecture Model integration* (ESAMI): Es un método para la integración de metamodelos de microservicios basado en matrices de correlación. Cada modelo de referencia tiene asignado un índice de correlación con otros servicios, y una opción de integración, cuyos valores pueden ser compulsorio o rechazados.

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

Ofrece una propuesta para la arquitectura en microservicios en el contexto del entorno empresarial. Se define cuatro capas, siendo la más abstracta la definición de los metamodelos y la más concreta, la definición de los datos en ejecución. Siendo una aproximación bastante genérica, sin detallar aspectos arquitectónicos, no ha sido ni validada ni evaluada.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Se mencionan lenguajes ya existentes para la definición de la capa con nivel de abstracción mayor ArchiMate & OWL. No hay ninguna propuesta adicional para ADL.

P41

(Messina, Rizzo, Storniolo, Tripiciano, & Urso, 2016)

Propone una arquitectura enfocada a utilizar la base de datos como servicio, que une la lógica de negocio junto con los datos. Esta solución ofrece ciertas ventajas como la simplificación de la arquitectura, eliminando una capa de servidor o contenedor, y presenta dos clusters: "Cluster Client" y "Database Cluster".

Este nuevo patrón arquitectónico se prueba introduciendo ebXML (electronic Exchange information in XML) a una base de datos noSQL. Es un prototipo que deriva de una extensión OrientDB, plataforma de desarrollo de aplicaciones de servidor que interactúa con objetos de

bases de datos gestionando otros objetos Java: POJOs (Plain Old Java Objects). Los resultados del uso de este nuevo patrón arquitectónico en el prototipo, presenta una mejora en el rendimiento con relación a una base de datos SQL estándar.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Define cuatro patrones arquitectónicos:

- API Gateway: Define como los clientes acceden a los servicios.

- Client & Server-side Discovery: Enruta peticiones cliente a una instancia servicio, ya sea a través del cliente (client-side), o a través de un router, que funciona como load balancer(server-side). Esta última opción tiene más saltos (network hops) por nodos aunque simplifica la parte de código de cliente en relación a la localización de servicios.

- Service Registry: Registra las instancias y la ubicación de los servicios). Es un componente que debe ser altamente disponible.

- Database per service: Cada servicio tiene su propia base de datos (solo accesible a través del, en tres formas distintas: conjunto de tablas, esquema BBDD, o database server.

- Database-is-a-service: La lógica de negocio está integrada con el motor de bases de datos.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.
- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

Aunque presenta un patrón de diseño orientado a las bases de datos, no menciona un estilo arquitectónico.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No define ningún lenguaje.

P58

(Balalaie, Heydarnoori, & Jamshidi, 2016)

Estudio primario que enumera los pasos seguidos para migrar una aplicación SSaaS a una arquitectura en microservicios utilizando metodologías DDD, transformando los componentes en servicios que se comunican a través API. Para solventar los problemas añadidos de la integración y descomposición en microservicios, se agregan nuevos componentes, como *Service Discovery, Config Server, Edge Server, Load Balancer y Circuit Breaker*, necearios para una arquitectura distribuida. Además, utiliza las tecnologías de contenedores, para fomentar la capacidad del sistema para la entrega continua SW (Continuous Delivery).

Por otro lado, menciona las necesidades de dicha migración (reusabilidad, descentralización datos, despliegue automático, y escalabilidad), así como las dificultades encontradas en el proceso.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Service (business capability), API (interfaz servicio), SSaaS (Service-side as a service).

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Conceptuales: DDD (Domain-Driven Design), Bounded Context

Operacionales: Configuration Server, Service Discovery, Load Balancer, Circuit Breaker, Container

Metodologías: DevOps (conjunto de prácticas que reducen el tiempo entre desarrollo y puesta en operación) y Continuous Delivery

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No se menciona un estilo arquitectónico, sin embargo, aplica patrones de diseño para transformar los componentes en servicios, usando las tecnologías orientadas a App basadas en microservicios.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No existe DSL ni ADL.

P72

(Gouvas, Fotopoulou, Zafeiropoulos, & Vassilakis, 2016)

Estudio primario que propone un modelo contextual para ARCADIA apps (*Reconfigurable-by-design Distributed Applications*), y un framework para la gestión de normas y/o políticas "*The Policies Management Framework*" (PMF), que provoca la ejecución de unas acciones basado en el análisis de los datos actuales y un conjunto de reglas, que al ser aplicadas afectan al despliegue de servicios, que son orquestados por "Smart Controller".

Si define un estilo arquitectónico basado/orientado a microservicios

  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

ARCADIA Context Model: Un modelo para representar los aspectos de desarrollo, despliegue y orquestación de ARCADIA App, la cual, responde a una arquitectura de capas, compuesta de componentes y/o servicios anidados (service chains) y otras entidades software. Tiene cuatro distintas facetas:

- *Component model (CM):* Representa las unidades de ejecución (microservicios), que encapsula la información sobre la configuración de los componentes para su despliegue, los requisitos en operación (CPU, memoria, etc..), y las interfaces expuestas y/o requeridas.

- *Service graph model (SGM)*: Combina varios modelos de componentes (CM), que contiene tres elementos: un '*GraphNode Descriptor'*: detalla los nodos del gráfico, *'VirtualLinkDescriptor'*: info sobre los links entre los nodos, y '*GraphMonitoringDescriptor*': define las métricas y la monitorización.

- *Service deployment model (SDM):* Asociación de los components con la ubicación en recursos IaaS. Compuesto de un 'ServiceGraphModelReference', 'ConfigurationDescriptor', configuracion de cada componente en el SGM, 'ComponentPlacementDescriptor', info sobre la ubicación en multi-IaaS.

- *Service Runtime Model (SRM*): Representa una instancia SDM que sigue las normas impuestas por el orquestador (Smart Controller). Compuesto por '*DeploymentModel*' instanciado y '*RuntimeBindings*' información sobre los componentes en ejecución.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No habla de estilo ni arquitectura de referencia, solamente propone un modelo que facilita el despliegue de las aplicaciones distribuidas orientadas a microservicios y un conjunto de reglas

que facilita la gestión de dichos servicios en la infraestructura.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No menciona ningún ADL ni DLS.

P77

(Zúñiga-Prieto, Insfran, Abrahao, & Cano-Genoves, 2016)

Utiliza un modelo arquitectónico para la integración de un microservicio en una plataforma Cloud, considerando nuevos atributos como el impacto arquitectónico. Presenta técnicas de transformación, modelado de lenguajes, y principalmente dos modelos arquitectónicos basados en UML que representan, por una parte, el modelo arquitectónico de la aplicación incluyendo los microservicios, y por otro el modelo arquitectónico de artefactos diseñados para la plataforma Cloud.

Si define un estilo arquitectónico basado/orientado a microservicios

- Nombre dado al estilo
- ¿define/caracteriza los tipos de componentes?
- ¿Define/caracteriza los tipos de conectores/relaciones?
- ¿Define/caracteriza las restricciones topológicas?
- Dominio o contexto de aplicación del estilo

- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Presenta dos metamodelos, con definición clases/componentes en diagrama UML, con la relación entre ellas (véase gráficos en estudio primario).

1st Model: Extended Increment Architecture Model (DIARy-specification-profile: basado en una extensión SoaML y UML)

- ✓ *Participant:* Microservice/Component para ser integrado. *ParticipantUse,* es una referencia a esta clase, y define tres atributos -> ***ElasticityLevel***: Nivel de escalabilidad del MS; ***DelayLevel*:** Si el MS requiere proceso inmediato
- ✓ *ServiceContract*: Describe interacción entre microservicios o *Participant. ServiceContractUse,* es una referencia a esta clase, y define los mismos atributos.
- ✓ *ServicesArchitecture,* partes del MS para que desempeñan el mismo rol
- ✓ *Role:* Tienen un nombre y un tipo de interfaz
- ✓ *Rolebinding:* Mapea los roles definidos en un *ServiceContract* con el *Participant*.
- ✓ *ExtendedIncrementArchitecture:* Especifica la lógica de integración con los elementos arquitectónicos agrupados en las clases: *Participant, RoleBinding y ServiceContract*.
- ✓ *Interface:* Especifica las operaciones y los eventos.

- ✓ *Interaction:* Definen interacciones entre varios componentes.

2nd Model: Increment Implementation

- ✓ *Project:* Metaclase que agrupa los artefactos que pueden ser implementados y desplegados en entornos cloud.
- ✓ *Implementation/Iteraction/Deployment Project:* Define artefactos relativos a la implementación, interacción y despliegue que podrían agrupados teniendo en cuenta ciertas semejanzas (Ej: tecnología, balanceadores, , etc..)
- ✓ *Artifacts:* Componentes SW que se agrupan en proyectos para ser implementados o desplegados conjuntamente según características comunes.
- ✓ *FrontEndService, HostedService:* Implementa la lógica de negocio.
- ✓ *EndPoint(Invoked/Exposed):* Especifica el tipo de servicio SOAP/REST y el formato de mensajes y protocolos.
- ✓ *Interface:* Implementa el inferfaz.
- ✓ *InteractionService:* Implementa protocolos de interacción.
- ✓ *ServiceData:* Estructura y formato de mensajes.
- ✓ *DynamicConfiguration*, está definida por *Settings* y por *EndPoints (invoked/exposed)* que cambian en entorno de ejecución (debe estar almacenados fuera de la unidad de despliegue)

  - o Indicar cuál es el protocolo de comunicación entre los diferentes elementos
  - o Qué estructura topológica guardan entre sí dichos elementos
  - o Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

  M2T (Model-to-text), transforma las interfaces de los microservicios (Participants), previamente chequeo de compatibilidades con el modelo arquitectónico de la aplicación.

  M2M (Model-to-model), transforma un modelo a otro mediante ATL (Atlas Transformation Language), y utilizando el framework de modelado en Eclipse (EMF). Modela los elementos internos de la arquitectura con artefactos en cloud.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

Se habla de un estilo arquitectónico incremental que engloba clases en diagramas UML, y las asociaciones entre ellas, en relación a la integración de nuevos componentes o participantes. No obstante, está asociado a la generación de artefactos en entornos Cloud, lo que dificulta enormemente la comprensión del estilo, ya que aparecen muchos componentes sin relación con la vista estructural de una arquitectura en microservicios.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Menciona lenguajes para el modelado de la arquitectura, tales como:

UML -> Unified Modelling Language

SoaML -> Service-Oriented Architecture Modelling Language

ATL -> Atlas Transformation Language (para generar el *Increment Cloud Artifact Model* desde *Extended Increment Architecture Model.*

P82

(Mazlami, Cito, & Leitner, 2017)

Estudio primario que presenta un modelo de extracción de microservicios partiendo de sistemas monolíticos. Este modelo es evaluado en base a unas métricas definidas de calidad y rendimiento en una serie de proyectos. Presenta unas mejoras con respecto al sistema monolítico: una disminución del tamaño de los equipos de desarrollo, y de la duplicación de código o funcionalidad. Presenta las siguientes fases:

1) Fase Construcción: Transforma el sistema monolítico (código fuente o repositorio con control versiones) en una representación gráfica, utilizando estrategias de acoplamiento (coupling strategies).

- Input: Siendo el sistema monolítico M, un conjunto de C (Clases), H (Historico de cambios - Eventos) y D (Desarrolladores).

- C -> File name, file path, file contents.
- H -> Conjunto de eventos definidos por una tupla de tres elementos.

o Hi -> (Ei, ti, di), donde “E”, es el conjunto de clases añadidas o modificadas por el evento hi; “t”, marca de tiempo cuando ocurre el evento; “d”, desarrollador.

- Estrategia Acoplamiento: Aplicando una función (Weight function) calcula un peso que define el grado de acoplamiento entre los nodos vecinos (o clases).

- Logical Coupling Strategy: Clases afectadas por los mismos cambios deben pertenecer al mismo microservicio.
- Semantic Coupling Strategy: Entidades de un mismo dominio (bounded context) en base al contenido y la semántica de su código fuente deben pertenecer al mismo microservicio.
- Contributor Coupling Strategy: Clases que han sido desarrolladas por el mismo equipo desarrollo y/o desarrolladores deben pertenecer al mismo microservicio.

- Output: Un gráfico compuesto por nodos (E, V)

- E -> Cada borde o línea con un peso que define el nivel de acoplamiento entre las dos clases que une, según las estrategias anteriores.
- V -> Cada vértice o nodo representa una clase que pertenece a C

2) Fase Clustering o Agrupamiento: Descompone la representación gráfica anterior en posibles candidatos a microservicios.

- Input: Gráfico anterior de dependencias entre nodos o clases, con un nivel de acoplamiento determinado.

- Algoritmo Agrupamiento: Aplicando un algoritmo de agrupamiento (Graph Clustering Algorithm) identifica los candidatos a convertirse en microservicios.

Divide el gráfico en subconjuntos considerando los pesos entre los vértices (cuanto más peso mayor dependencia entre entidades o vértices)

- Output: Conjuntos de microservicios "S", que son un subconjunto de clases del sistema monolítico con dependencias entre ellas.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan.

Conjunto de microservicios y relaciones dependencia entre ellos.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos (y aspectos de coordinación)
  - Qué estructura topológica guardan entre sí dichos elementos

Nivel de acoplamiento en base a sus dependencias, semántica y/o desarrollador del conjunto de microservicios.

  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No representa estilo arquitectónico, aunque presenta un modelo formal para la migración de sistemas monolíticos a microservicios.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No define ningún lenguaje.

P83

(Oberhauser, 2017)

Ofrece un modelo operacional llamado Microflows para la integración y orquestación de microservicios en los procesos de negocio usando agentes inteligentes, métodos gráficos y una semántica basada en JSON-LD (*JavaScript Object Notation for Linked Data*). Este método de

codificación de datos, está basado en JSON, que es un formato de serialización usado en la comunicación entre cliente y servidor para integrar datos estructurados en páginas web, y Hydra, que representa el vocabulario entre ellos.

Microflow presenta un mecanismo para determinar distintos flujos de trabajo realizado mediante la invocación de los microservicios en el orden establecido de un plan inicial, así como para la replanificación y recuperación de errores. Para ello, se establecen unos principios que seguir y definen fases del ciclo de vida.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Microservice semantic self-description: Microservicios proveen metadata para que sea invocado de forma autónoma por parte del cliente.

Client-Agent: Construcción de agentes inteligentes que usan el modelo BDI (Believe-Desire-Intention), son capaces de elegir que hacer en base a un objetivo y ejecutar los planes, a diferencia de otros métodos que ejecutan las acciones de forma rígida acorde a un plan (workflow schema).

Graph of microservices: Los microservicios son mapeados en un gráfico de nodos que representa el flujo de trabajo (workflow). Puede almacenarse en una base de datos de gráficos que sea compartida por todos los agentes. Cualquier especificación de un flujo de trabajo, posee un objetivo (Ej: inicio y fin en el flujo) y unas restricciones secuenciales o lógicas de la rama.

Microservice Discovery service: Sirve para el registro y descubrimiento de los servicios (Metaservice), que puede ser de tipo centralizado, distribuido o embebido en el cliente, además puede activarse el registro de forma voluntaria por cada servicio o mediante mecanismos multicast.

Abstract microservice: Los microservicios con funcionalidad similar, pueden agruparse en un microservicio abstracto que sea conectado al cliente, de forma que no conozca los microservicios concretos, puesto que pueden ser numerosos y cambiar dinámicamente.

Path Weighting**:** Se aplica un coste dinámico a los distintos caminos del flujo de los servicios, para ayudar a elegir o descartar aquellos que sean más costosos, en otras iteraciones.

  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos

Aunque no indica protocolo de comunicación, se comenta que el Microservice Discovery, construye a través de un metaservicio MS discovery (Eurkeka), un gráfico de nodos que contienen propiedades de los microservicios y de sus enlaces (followers) a otros microservicios.

Durante la fase de planificación un agente inteligente (BDI) crea un plan llamado Microflow, a partir de un objetivo y unas restricciones extraídos, encontrando un nodo inicial y final y aplicando un algoritmo (Ej: shortest path).

  - Qué estructura topológica guardan entre sí dichos elementos

  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

El objetivo y la especificación de restricciones son modeladas (JSON), mediante la extracción de datos de BPMN (*Business Process Modelling Notation*) o procesos de minería de los ficheros de log en ejecución (véase descripción al principio página).

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

El artículo está más enfocado a establecer y/o recuperar un flujo de trabajo en un sistema basado en microservicios, que de estilos arquitectónicos propios. Establece los componentes y unas fases para establecer el camino óptimo de invocación de servicios en base a un plan inicial. Introduce patrones de diseño, llamados metaservicios que incluyen los microservicios abstractos y el microservicio de registro y descubrimiento, así como agentes clientes dotados de inteligencia que pueden actuar según su conocimiento.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Menciona lenguajes de comunicación entre servicios basados en JSON-LD que incluye JSON. Este método de codificación de datos, está basado en JSON, que es un formato de serialización usado en la comunicación entre cliente y servidor para integrar datos estructurados en páginas web, y Hydra, que define el vocabulario o la semántica. También menciona el lenguaje BPMN (Business Process Model Notation) para los procesos de negocio.

P87

(Gutiérrez–Fernández, Resinas, & Ruiz–Cortés, 2017)

Propone una forma de comunicación y despliegue en sistemas basados en microservicios, utilizando un motor de procesos "*engine process*". Redefine el rol del motor de procesos en SOA, sustituyendo la orquestación por la coreografía, y utilizando "*event-based async communication*".

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombre dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

Microservicio: Componente que incluye la lógica de negocio, los datos, interfaz usuario (si existe como parte de una general), y API (dividida en BPMS API, y Event API, que gestiona el envío/recepción de eventos en un sistema con metodología Publish/Subscribe).

BPMS API: Interfaz que expone la lógica negocio del microservicio.

Event API: Interfaz que controla el envio/recepción de eventos para el microservicio.

Message Broker: Un intermediario que traduce y enruta los msj. Recibe msj de productores y los reenvía a los consumidores.

Process Engine: Implementado dentro del microservicios, se encarga de manejar los eventos, y conecta con los datos.

Event Mapper: Es un componente dentro del process engine que mapea los eventos de entrada con los procesos que ofrece el motor y encargado de los eventos de suscripción.

- Indicar cuál es el protocolo de comunicación entre los diferentes elementos

El protocolo asíncrono orientado a eventos "Event-based Async communication" para la comunicación entre servicios. También menciona mecanismos síncronos basados en HTTP REST.

Advanced Message Queuing Protocol: protocol de mensajes que comunica el cliente con el middleware message broker.

- Qué estructura topológica guardan entre sí dichos elementos
- Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor,…

Publish/Subscribe mecanismo, donde los microservicios que producen se suscriben a eventos que son provocados por los consumidores, de manera que se desencadene un envío/recepción de mensajes entre los distitnos servicios.

Coreography, para la interacción entre los servicios, de forma que los componentes deban conocer el endpoint (definido por un microservicio), protocolos y los detalles del dominio. Toda esta información, es almacenada por el orquestador, en el caso de aplicaciones SOA.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

Propone el uso de un "*engine process*" como plataforma para implementar sistemas en microservicios, añadiendo varios componentes para cumplir las propiedades de estos sistemas (organizado a través de lógicas de negocio, descentralización gestión datos, despliegue individual, interacción con otros microservicios y el desarrollo de la interfaz de usuario), así como el manejo de la interacción entre ellos.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Menciona un lenguaje BPMN (*Business Process Model notation)* para la definición de la lógica de negocio, que provee un mecanismo de comunicación orientada a eventos.

Dentro de la orquestación de servicios, BPEL constituye un lenguaje estándar para la integración y automatización de procesos, y manejo de eventos.

P91

(Hassan, Ali, & Bahsoon, 2017)

Estudio primario que realiza una asignación dinámica de la granularidad de una aplicación basada en microservicios en tiempo de ejecución. La herramienta llamada *Microservice Ambients* establece las primitivas para adaptar los límites o *boundaries*, y que define cuatro Aspects:

. Granularity Adaptation Aspect: Define el comportamiento para que un MSAmbient adapte sus granularidad (Primitivas: newAmbient, add-removeChild, add-removeAttachment)

InMonitorPort/ExMonitorPort: Publish/Request el monitoring service tanto de los elementos dentro y fuera del Ambient.

. Mobility Aspect: Permite MSAmbient para entrar o salir de los límites de su padre.

. Coordination Aspect: Comunicación entre elementos arquitectónicos dentro del Ambient, hacia el exterior.

. Distribution Aspect: Informa sobre la estructura jerárquica almacenando el nombre del padre en cada Ambient.

Especifica un lenguaje Ambient-PRISMA Textual Language, para definir plantillas para la creación Aspects.

- Si define un estilo arquitectónico basado/orientado a microservicios
    - Nombres dado al estilo
    - ¿define/caracteriza los tipos de componentes?
    - ¿Define/caracteriza los tipos de conectores/relaciones?
    - ¿Define/caracteriza las restricciones topológicas?
    - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
    - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

- Componentes (Microservice Ambients): Modelado de las interfaces de los componentes, que define reglas para monitorizar a través de Sources (que es un conjunto de componentes arquitectónicos que son monitorizados por el MSAmbient), y para control, a través de transacciones MERGE/DECOMPOSE.

- Conectores: Tiene dos tipos de conectores:

. Attachments: Relaciones entre los distintos MSAmbients (definiendo varios puertos: EC, IC, ES, IS, ExM, InM, IR)

. Weavings: Relaciones entre los distintos Aspects

    - Indicar cuál es el protocolo de comunicación entre los diferentes elementos (interfaz y interacciones)
    - Qué estructura topológica guardan entre sí dichos elementos

- Transacciones: MERGE, une componentes (MSAmbients) y DECOMPOSE, descompone un componente en varios.

    - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

Estilo orientado a eventos

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No se menciona un estilo arquitectónico, aunque se muestra un diseño de una arquitectura que resuelve el problema del cambio de los limites (granularidad) de los microservicios, mediante una estructura jerarquizada de *MicroServiceAmbients* (componentes), que se comunican a través de distintos puertos con otros componentes, teniendo la capacidad de cambiar su contenido (elementos arquitectónicos) de forma dinámica.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Se propone la definición Ambient-PRISMA Textual Language, como un potencial lenguaje para definición de microservicios, evaluando sus características como ADL.

También menciona otros ADL's y sus características/limitaciones (MeiaH y UniConno, ACME, Rapide, C2).

P92

(Klock, Werf, Guelen, & Jansen, 2017)

Estudio primario que define un modelo de arquitectura en microservicios, basado en la distribución de las características del sistema (funcionalidad) entre los distintos microservicios considerando las dependencias entre ellos. Según este gráfico de dependencias, se establece que propiedades de las características dependientes son incluidas para obtener un microservicio independiente, con el objetivo de reducir el nº de llamadas entre ellos. Sin embargo, la mantenibilidad del sistema se ve afectada, puesto que se duplica funcionalidad, y los tiempos de espera del cliente aumentan debido al incremento de los mensajes propagados en las colas.

Aporta una herramienta de libre código, MicADO, para optimizar un modelo de despliegue de una arquitectura en microservicios en base a su carga (workload). A través de un algoritmo genético, que recibe como entradas, un "Microservice Arquitecture Model" original y un "Workload", ofrece como resultado otro ""Microservice Arquitecture Model" con mejor rendimiento y escalabilidad. Se utilizan unas métricas para la valoración, como los tiempos de respuesta, el número medio de peticiones procesadas, y tamaño de los microservicios (según el nº instancias de características que se han incluido de otros dependientes). Se realiza un caso de estudio en un ERP software usando la tool MicADO.

- Si define un estilo arquitectónico basado/orientado a microservicios
  - Nombres dado al estilo
  - ¿define/caracteriza los tipos de componentes?
  - ¿Define/caracteriza los tipos de conectores/relaciones?
  - ¿Define/caracteriza las restricciones topológicas?
  - Dominio o contexto de aplicación del estilo
- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan

I. Feature Model: Compuesto de una tupla de 3 elementos (3-tuple)

a. Set of features (Fi): Conjunto de características del negocio que son instanciadas en los microservicios (Ej: Invoice, Order, Customer)

b. Set of properties (Pi): Conjunto de propiedades por cada característica anterior.

c. Dependency Graph: Grafico con las dependencias entre las propiedades de una característica con las otras definidas en otras características.

II. Microservice Architecture: Implementa un modelo anterior "Feature Model", mediante la instanciación de características en los microservicios. (4-tuple)

a) Set of Microservices (M): Conjunto de microservicios.

b) Set of feature instances (I): Conjunto de características instanciadas de cada microservicio. Puede ser publica (contiene todas las propiedades) o interna (un subconjunto de propiedades).

c) Property Instantiation Function (^: I -> 2P): Función que mapea cada "feature instance" con un conjunto de propiedades.

d) Public Instante Function (h: F -> I): Función que define una instancia publica por cada característica.

- Indicar cuál es el protocolo de comunicación entre los diferentes elementos
- Qué estructura topológica guardan entre sí dichos elementos

- Cada MS contiene todas las instancias necesarias para cumplir los requisitos de dependencias.

- Cada MS contiene cada característica al menos una vez.

- Cada instancia de una característica tiene un conjunto de propiedades de su característica.

- Cada característica tiene una instancia pública que contiene todas sus propiedades.

- Cada microservicio contiene al menos una instancia pública de una característica

- Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Tools: MicADO para optimización de un modelo de despliegue de una arquitectura en microservicos.

I. Inputs:

a. Microservice Architecture Model: Muestra el modelo arquitectónico original.

b. Workload: Volumen de trabajo, que puede ser bajo o máximo, provocando distintos modelos arquitectónicos de salida.

II. Proceso: Mediante un algoritmo genético.

III. Valoración: A través de métricas es posible valorar

a. Feature Usage Metrics -> Los patrones de uso frecuente a través de análisis de log "process mining".

b. Performance Metrics -> Miden el rendimiento y la escalabilidad del sistema, añadiendo metadata (id, feature) a la métrica, para establecer las medidas de uso del sistema por característica (tened en cuenta que las métricas de rendimiento están generalmente asociadas a procesos).

c. Fitness function -> Función que determina la calidad del despliegue basado en la teoría de colas M/M/1.

IV. Output: Modelo arquitectónico optimizado, con mejoras en los tiempos de respuesta o proceso de peticiones reduciendo el nº de comunicaciones entre servicios. Se consigue a través de la inclusión de las características de otros servicios dependientes, aunque intentando minimizarlas.

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

En este documento se habla de un modelo arquitectónico basado en la distribución de características/funcionalidad entre los microservicios. Al combinar características con dependencias en un mismo microservicio, se reduce el nº de llamadas, aunque se duplica funcionalidad.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

Se explica la arquitectura con una notación formal.

P93

(Haselböck, Weinreich, & Buchgeher, 2017)

Estudio que presenta las principales áreas de diseño (diseño del sistema, organización de los equipos y procesos, infraestructura) que deben ser consideradas al introducir arquitectura en microservicios y soportadas mediante modelos de decisión (*Decision Models*). A través de un proceso definido TAR *(Technical Action Research*) define los importantes actores (s*takeholders*), casos de uso y requisitos de los modelos de decisión. El objetivo del estudio conocido como TAR es el diseño de artefactos que son validados bajo condiciones reales.

Si define un estilo arquitectónico basado/orientado a microservicios

- Nombres dado al estilo
- ¿define/caracteriza los tipos de componentes?
- ¿Define/caracteriza los tipos de conectores/relaciones?
- ¿Define/caracteriza las restricciones topológicas?
- Dominio o contexto de aplicación del estilo

- Si no define un estilo arquitectónico basado/orientado a microservicios y solo habla de arquitectura de microservicios:
  - Qué elementos figuran en dicha arquitectura y cómo se caracterizan
  - Indicar cuál es el protocolo de comunicación entre los diferentes elementos
  - Qué estructura topológica guardan entre sí dichos elementos
  - Qué paradigmas u otros estilos aparecen/se citan: orientación a objetos, publish/subscribe, cliente/servidor, etc.

- Cualquier comentario de relevancia que no quepa en los epígrafes anteriores

Las siguientes áreas de diseño son identificadas para la arquitectura en microservicio: *System Design, Organizational Structures & Processed, Infraestructure*. En el área de diseño de sistemas, se tienen en cuenta los distintos componentes necesarios en un sistema de microservicios, tales como *Service Discovery, Registration*, etc., y en la integración ocurre en distintos niveles: datos, interfaz de usuario y servicio.

En relación a las research questions:

RQ1: ¿Podemos hablar de un estilo arquitectónico orientado/basado en microservicios?

No habla de una arquitectura en microservicios. No obstante, define los aspectos a considerar en la adopción de una arquitectura en microservicios, que abarcan el diseño del sistema, la organización de los equipos (más procesos) y la infraestructura que soporta el desarrollo y el despliegue del sistema.

RQ2: ¿Existe un lenguaje para describir este estilo? Y si no existe ¿se puede extender/ampliar alguno de los existentes o se debe crear uno nuevo?

No menciona lenguaje de descripción del estilo.

Anexo D: Diseño Mapas Conceptuales

En este apéndice se muestra los mapas conceptuales que son diseñados por cada estudio primario, con el objetivo de facilitar la labor de síntesis de la información útil para el trabajo de investigación. Se utiliza la herramienta *"CMapTools"* para la elaboración de dichos mapas conceptuales.

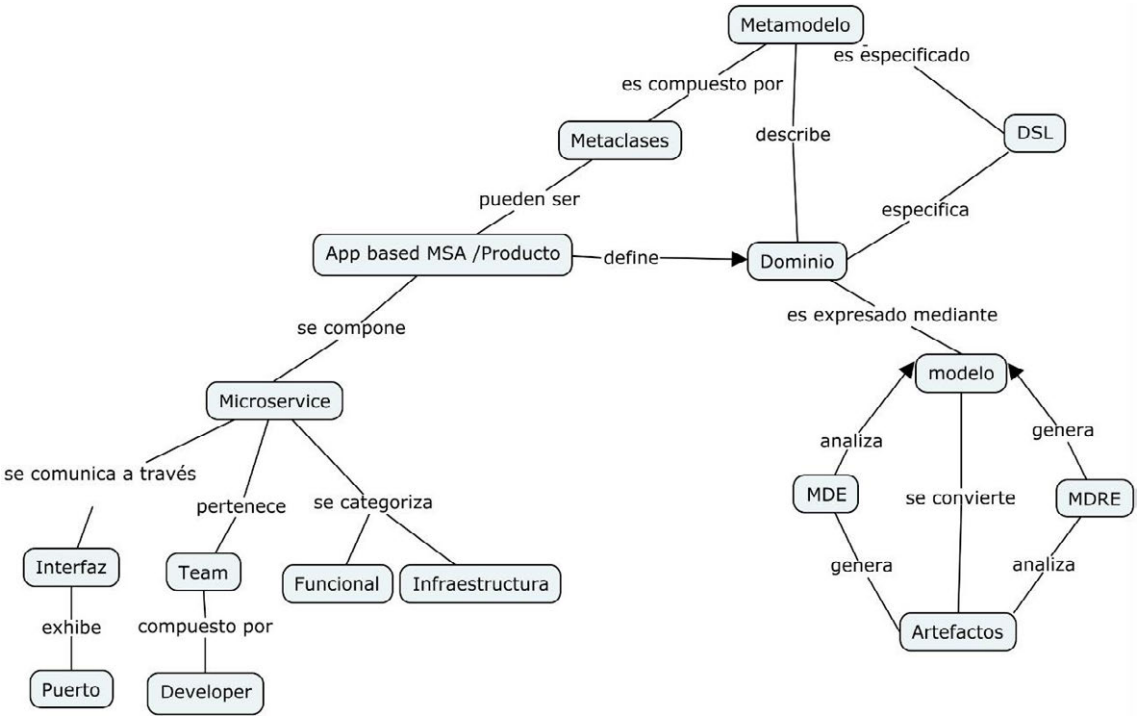


FIGURA 12 MAPA CONCEPTUAL ESTUDIO PRIMARIO P01

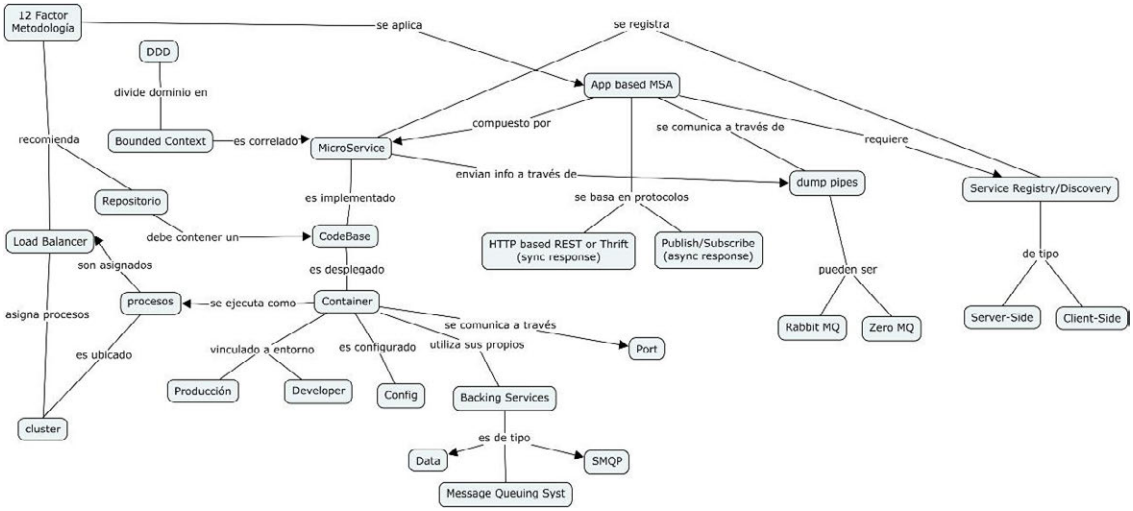


FIGURA 13 MAPA CONCEPTUAL ESTUDIO PRIMARIO P03

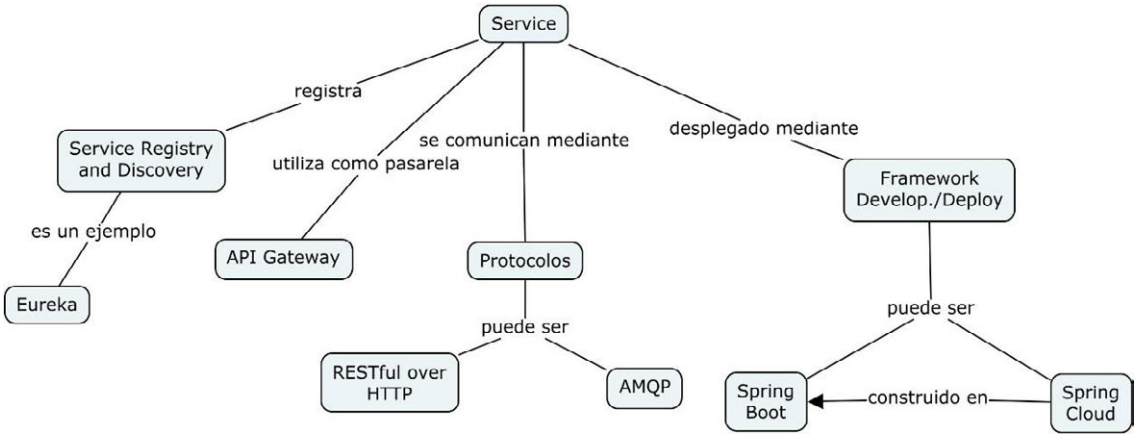


FIGURA 14 MAPA CONCEPTUAL ESTUDIO PRIMARIO P07

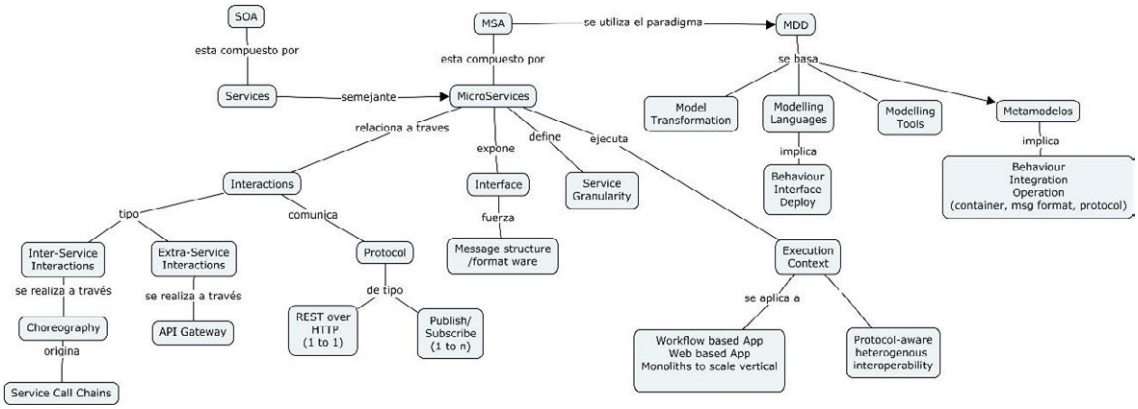


FIGURA 15 MAPA CONCEPTUAL ESTUDIO PRIMARIO P10

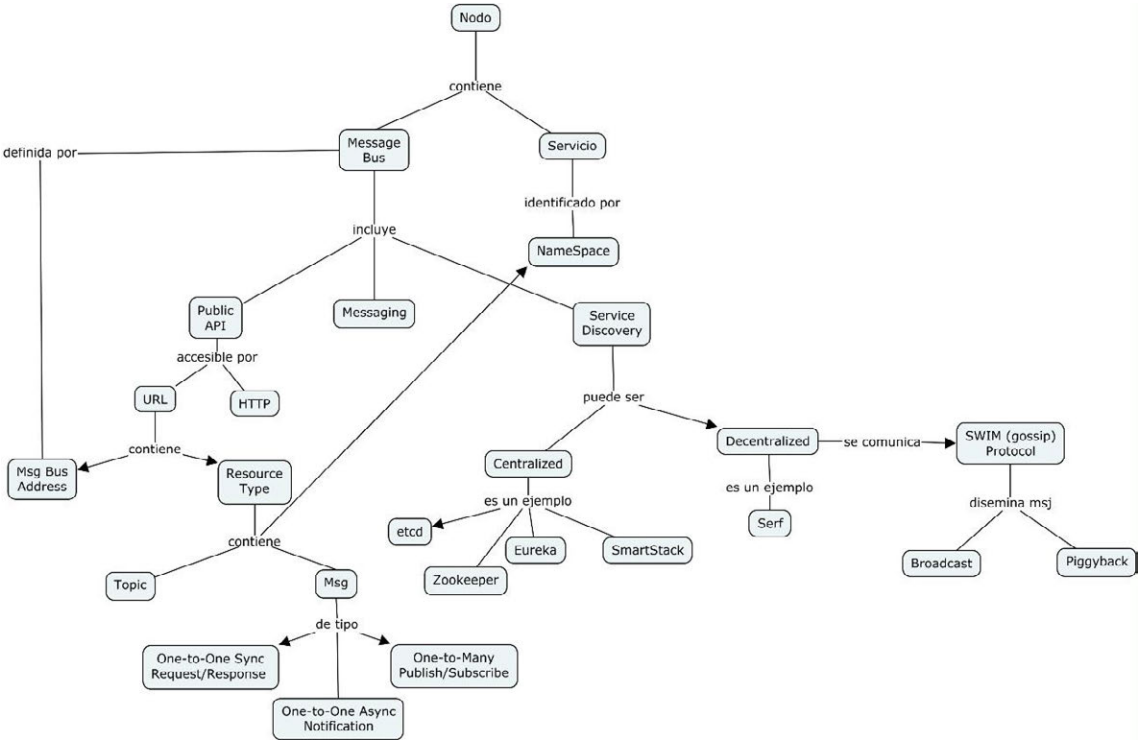


FIGURA 16 MAPA CONCEPTUAL ESTUDIO PRIMARIO P14

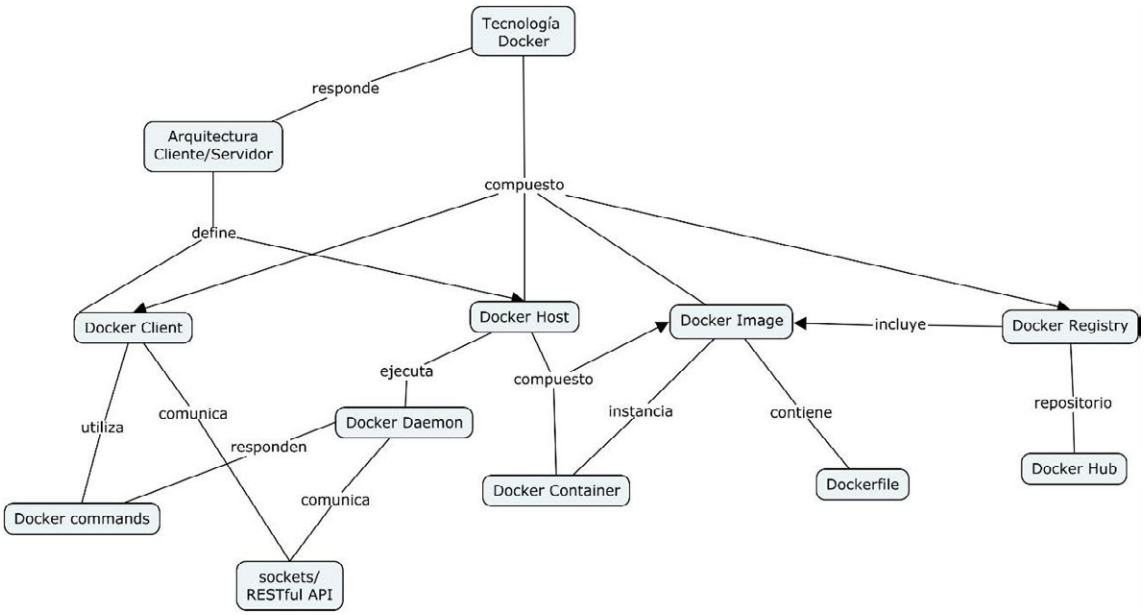


FIGURA 17 MAPA CONCEPTUAL ESTUDIO PRIMARIO P16

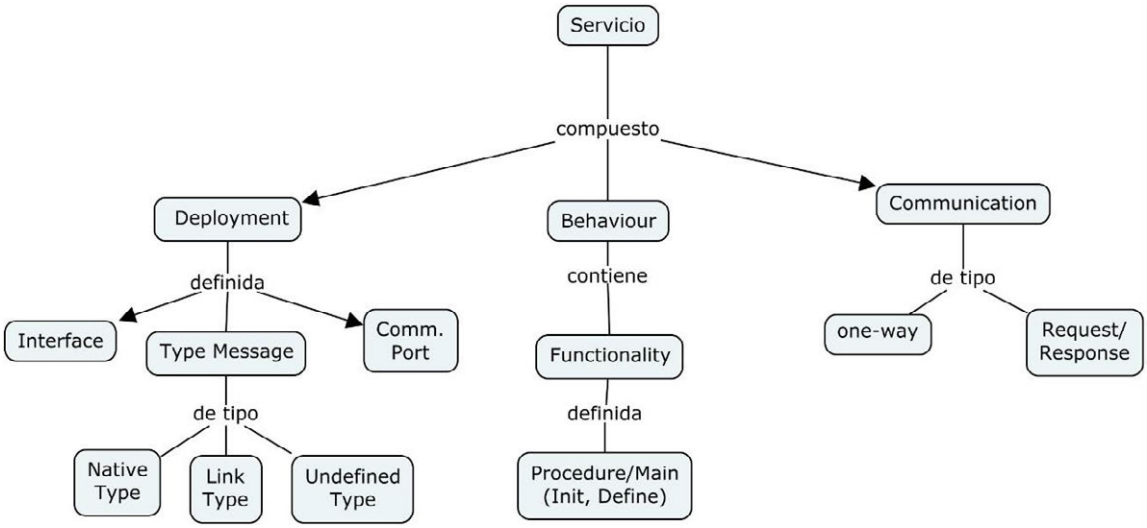


FIGURA 18 MAPA CONCEPTUAL ESTUDIO PRIMARIO P17

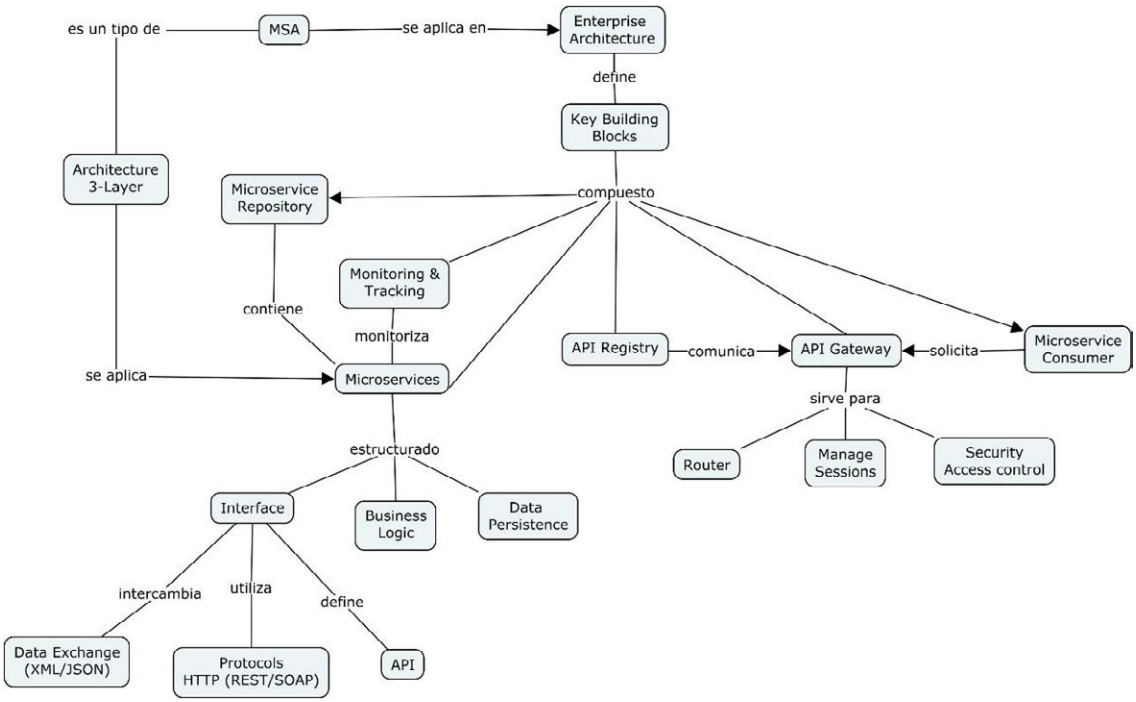


FIGURA 19 MAPA CONCEPTUAL ESTUDIO PRIMARIO P21

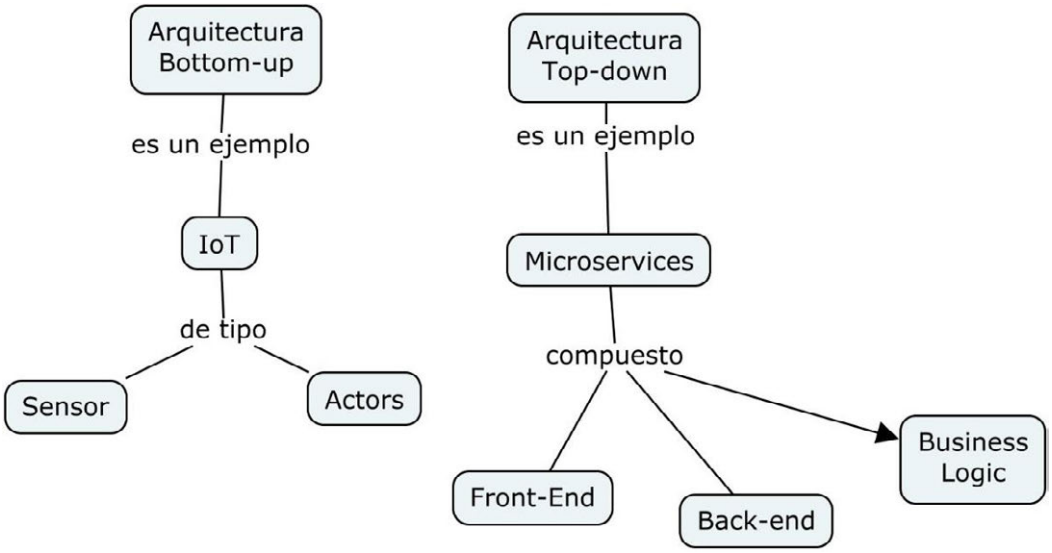


FIGURA 20 MAPA CONCEPTUAL ESTUDIO PRIMARIO P23

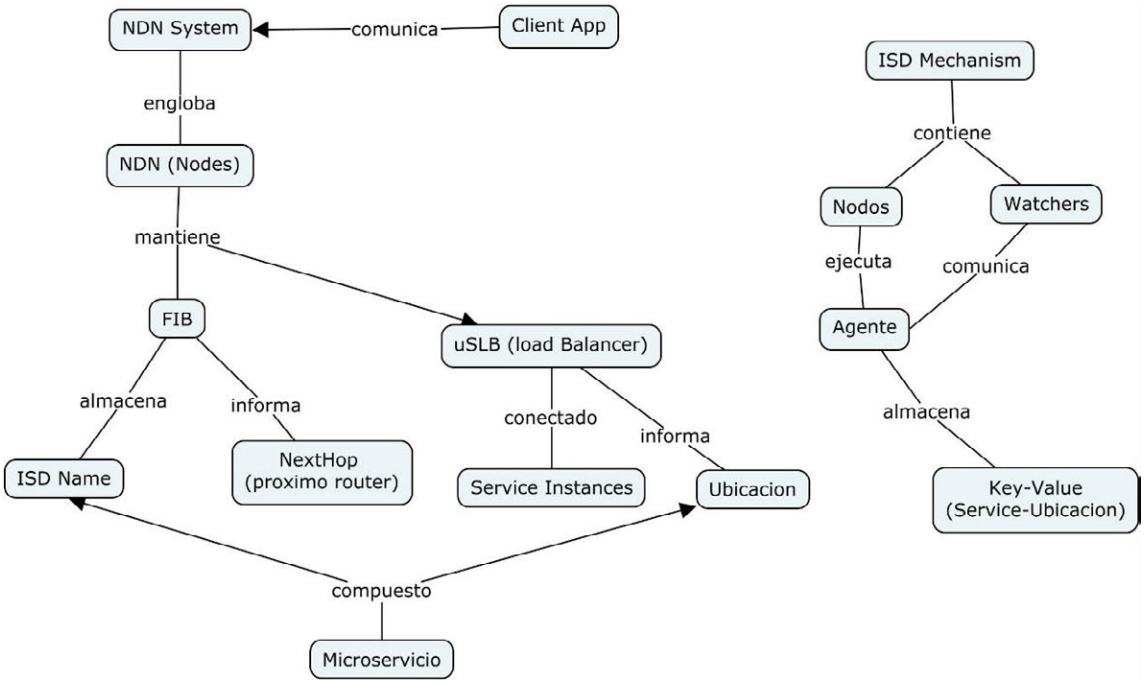


FIGURA 21 MAPA CONCEPTUAL ESTUDIO PRIMARIO P26

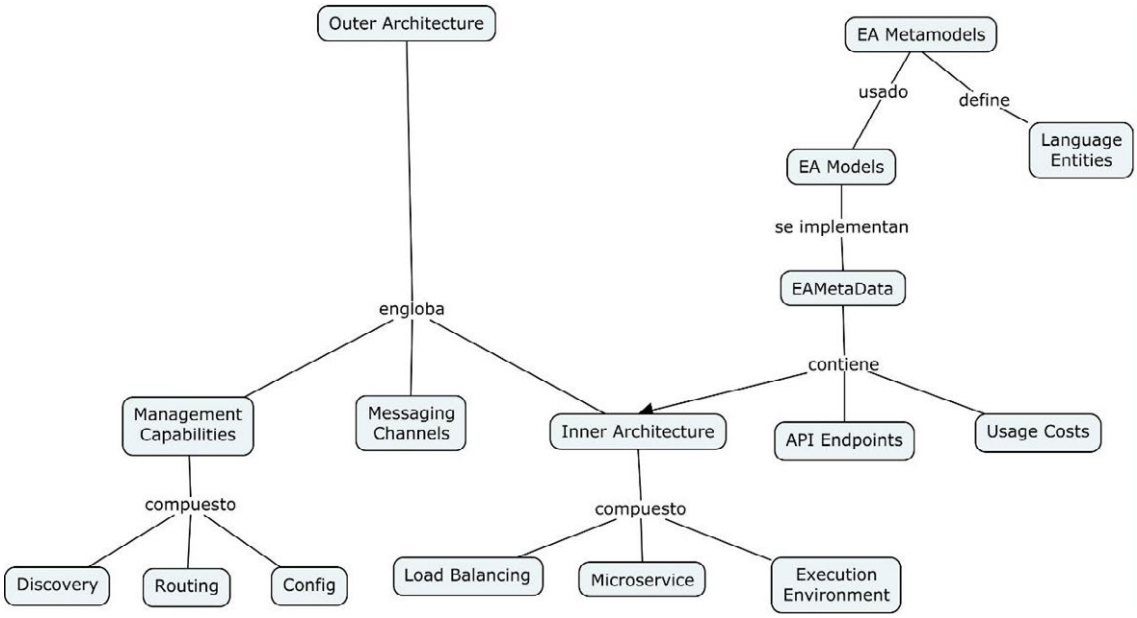


FIGURA 22 MAPA CONCEPTUAL ESTUDIO PRIMARIO P31

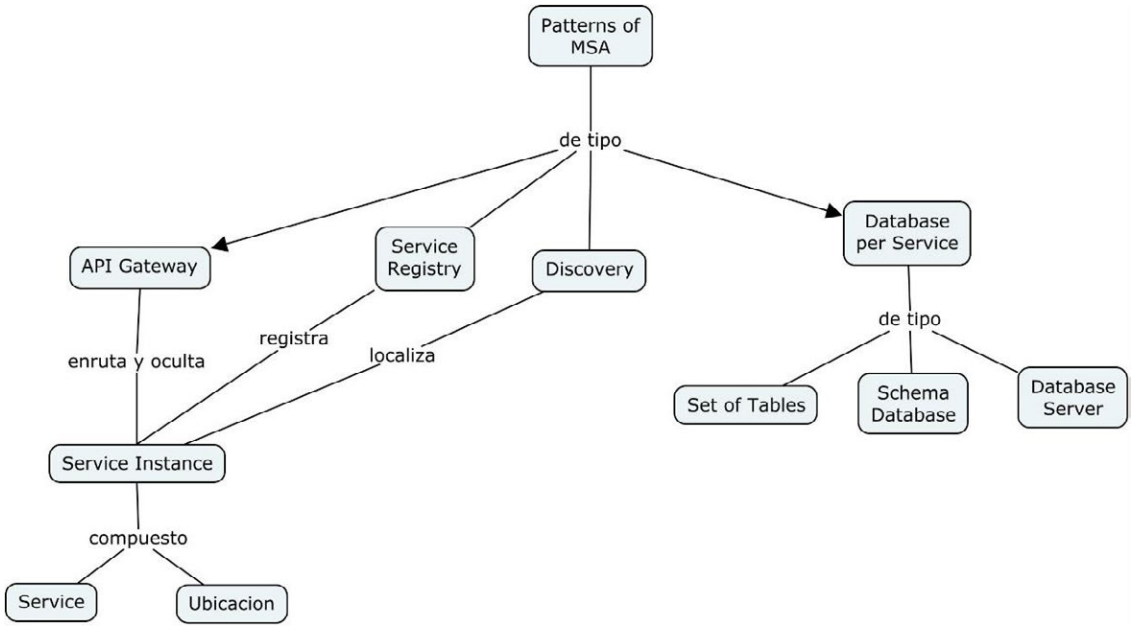


FIGURA 23 MAPA CONCEPTUAL ESTUDIO PRIMARIO P41

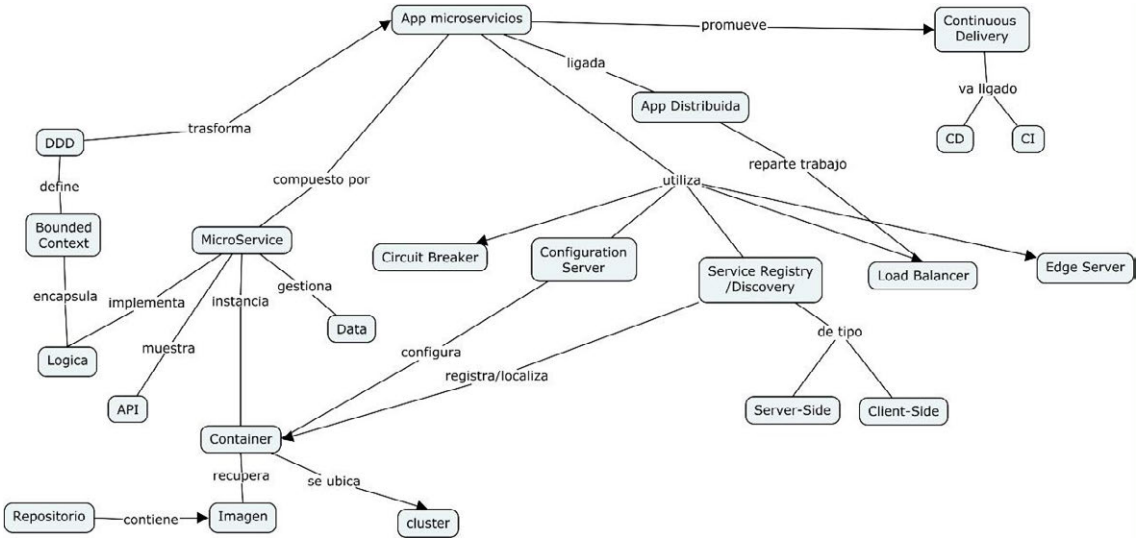


FIGURA 24 MAPA CONCEPTUAL ESTUDIO PRIMARIO P58

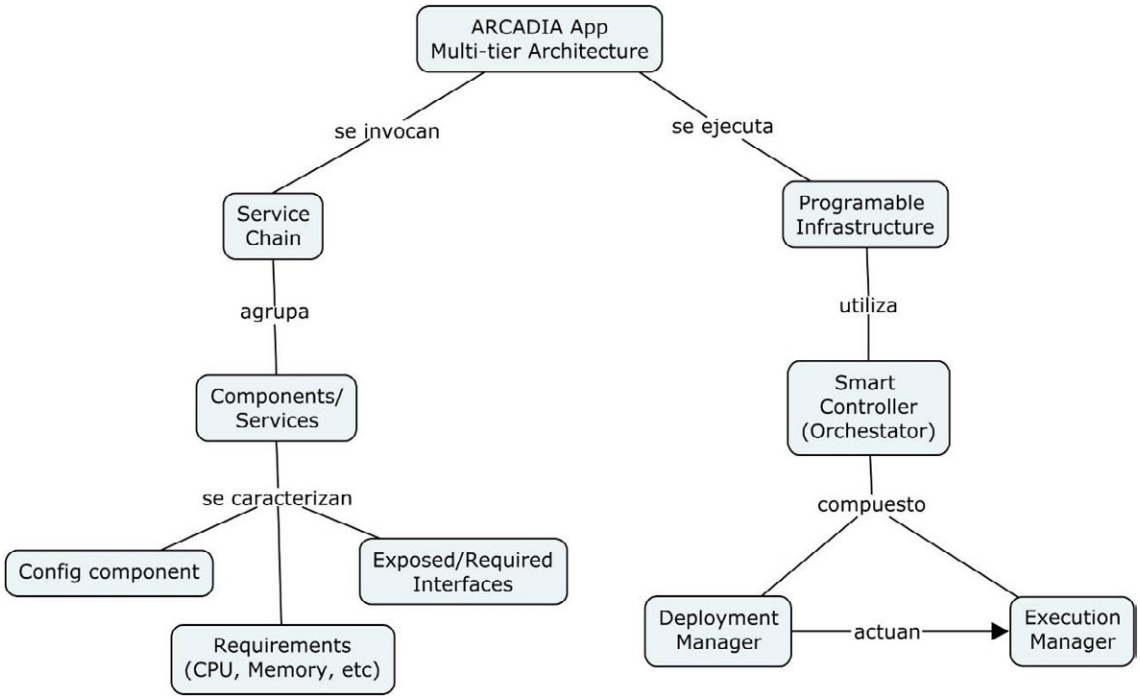


FIGURA 25 MAPA CONCEPTUAL ESTUDIO PRIMARIO P72

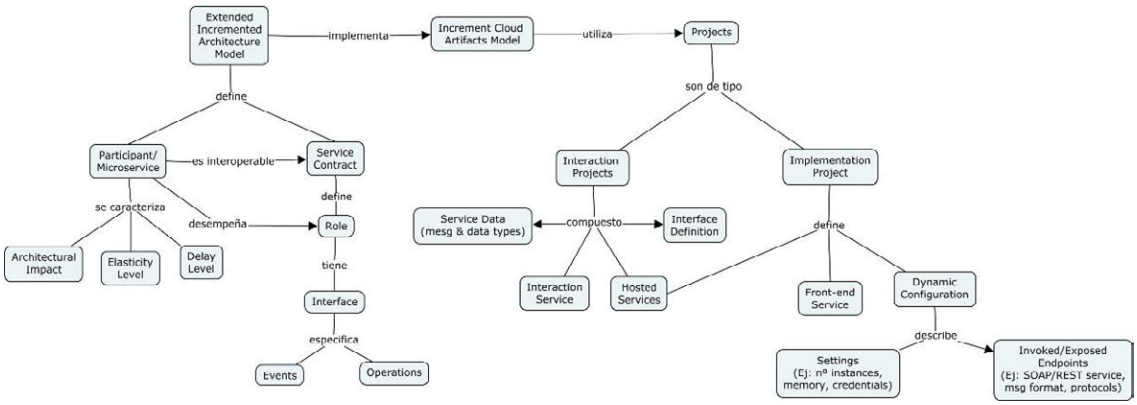


FIGURA 26 MAPA CONCEPTUAL ESTUDIO PRIMARIO P77

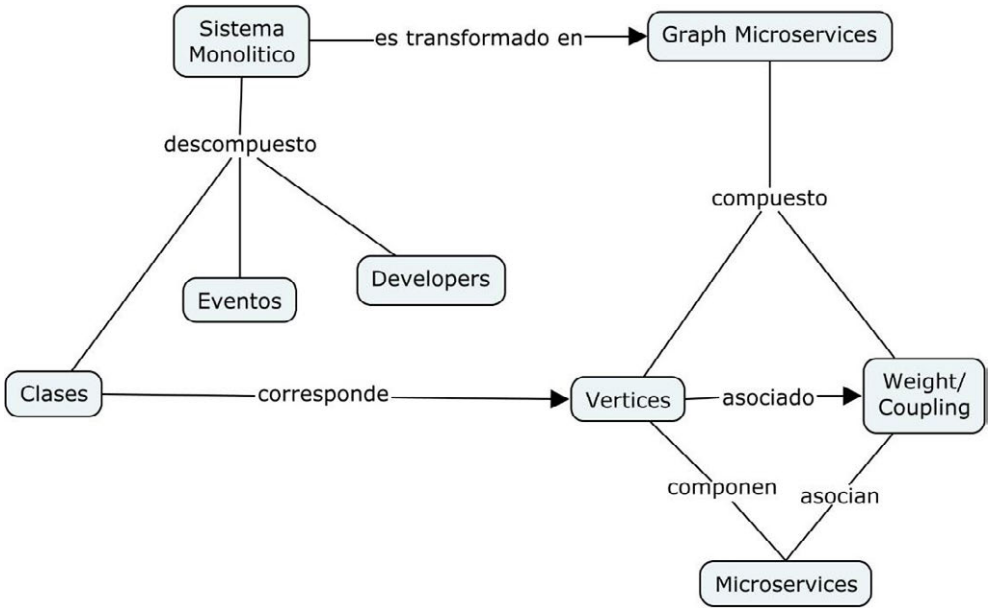


FIGURA 27 MAPA CONCEPTUAL ESTUDIO PRIMARIO P82

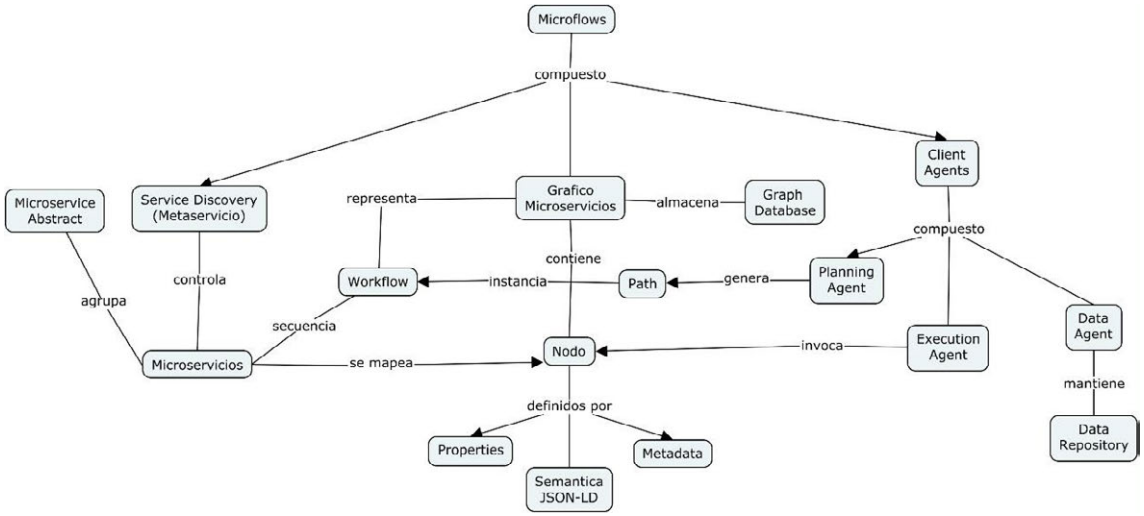


FIGURA 28 MAPA CONCEPTUAL ESTUDIO PRIMARIO P83

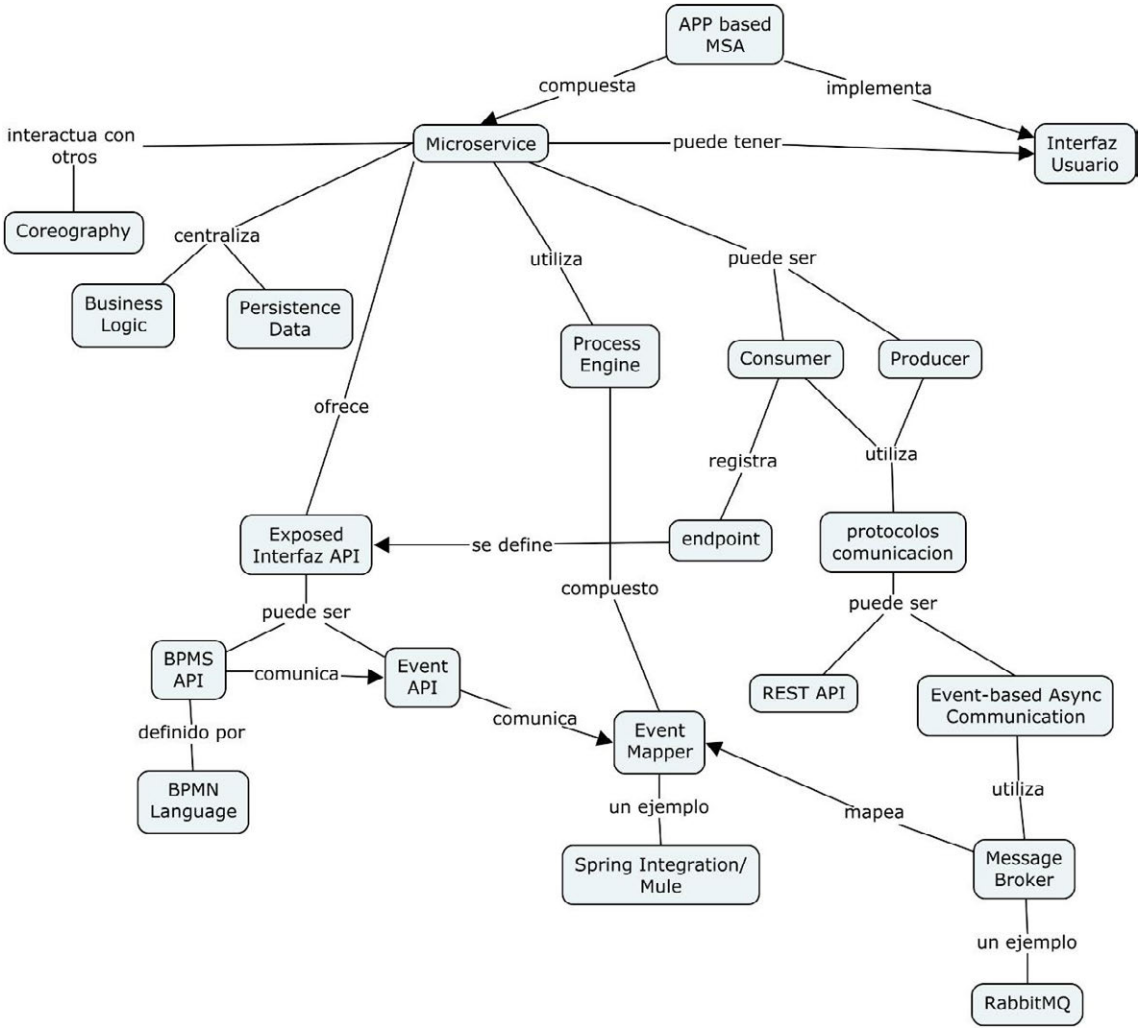


FIGURA 29 MAPA CONCEPTUAL ESTUDIO PRIMARIO P87

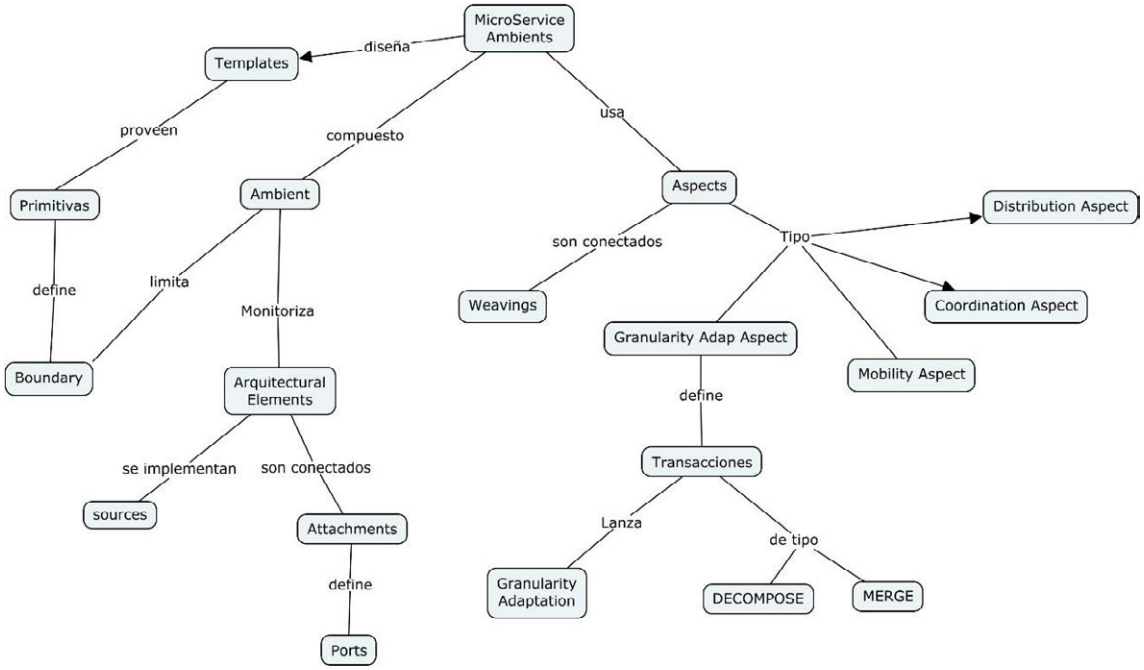


FIGURA 30 MAPA CONCEPTUAL ESTUDIO PRIMARIO P91

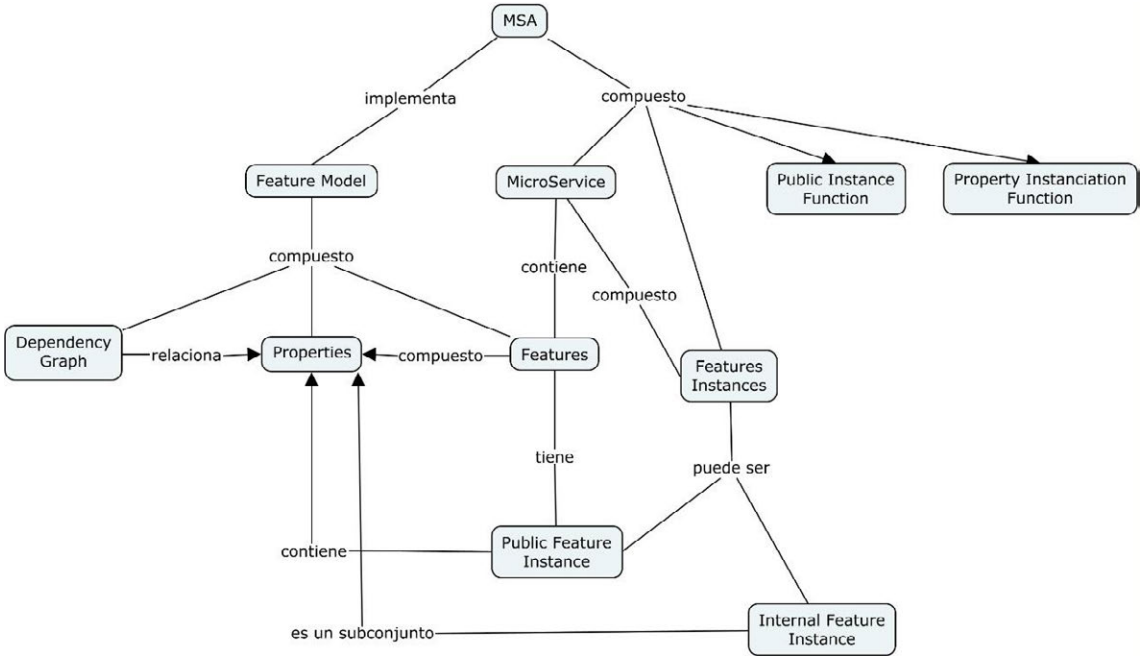


FIGURA 31 MAPA CONCEPTUAL ESTUDIO PRIMARIO P92

Referencias

Bakshi, K. (2017). (P03) Microservices-based software architecture and approaches. In *2017 IEEE Aerospace Conference* (pp. 1–8). IEEE. ttps://doi.org/10.1109/AERO.2017.7943959

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). (P58) Migrating to Cloud-Native Architectures Using Microservices: An Experience Report (pp. 201–215). Springer, Cham. https://doi.org/10.1007/978-3-319-33313-7_15

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice*. Retrieved from http://proquest.safaribooksonline.com/book/software-engineering-and-development/9780132942799/14dot-quality-attribute-modeling-and-analysis/ch14lev1sec2#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODAxMzI5NDI3OTklMkZjaDEybGV2MXNlYzQmcXVlcnk9

Bogner, J., & Zimmermann, A. (2016). (P31) Towards Integrating Microservices with Adaptable Enterprise Architecture. In *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)* (pp. 1–6). IEEE. https://doi.org/10.1109/EDOCW.2016.7584392

Butzin, B., Golatowski, F., & Timmermann, D. (2016). (P23) Microservices approach for the internet of things. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1–6). IEEE. https://doi.org/10.1109/ETFA.2016.7733707

Feinstein, a. R., & Cicchetti, D. V. (1990). Hig agreement but low kappa: II. Resolving the paradoxes. *Journal of Clinical Epidemiology*, *43*(6), 551–558. https://doi.org/DOI: 10.1016/0895-4356(90)90159-M

Fowler, M., & Lewis, J. (2014). Microservices. Retrieved May 1, 2018, from https://martinfowler.com/articles/microservices.html

Francesco, P. Di. (2017). Architecting Microservices. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 224–229. https://doi.org/10.1109/ICSAW.2017.65

Gouvas, P., Fotopoulou, E., Zafeiropoulos, A., & Vassilakis, C. (2016). (P72) A Context Model and Policies Management Framework for Reconfigurable-by-design Distributed Applications. *Procedia Computer Science*, *97*, 122–125. https://doi.org/10.1016/J.PROCS.2016.08.288

Granchelli, G., Cardarelli, M., Francesco, P. Di, Malavolta, I., Iovino, L., & Salle, A. Di. (2017). (P01) Towards Recovering the Software Architecture of Microservice-Based Systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)* (pp. 46–53). IEEE. https://doi.org/10.1109/ICSAW.2017.48

Gutiérrez–Fernández, A. M., Resinas, M., & Ruiz–Cortés, A. (2017). (P87) Redefining a Process Engine as a Microservice Platform (pp. 252–263). Springer, Cham. https://doi.org/10.1007/978-3-319-58457-7_19

Haselböck, S., Weinreich, R., & Buchgeher, G. (2017). (P93) Decision Models for Microservices: Design Areas, Stakeholders, Use Cases, and Requirements (pp. 155–170). Springer, Cham. https://doi.org/10.1007/978-3-319-65831-5_11

Hassan, S., Ali, N., & Bahsoon, R. (2017). (P91) Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity. In *2017 IEEE International Conference on Software Architecture (ICSA)* (pp. 1–10). IEEE. https://doi.org/10.1109/ICSA.2017.32

Jaramillo, D., Nguyen, D. V, & Smart, R. (2016). (P16) Leveraging microservices architecture by using Docker technology. In *SoutheastCon 2016* (pp. 1–5). IEEE. https://doi.org/10.1109/SECON.2016.7506647

Juan María Hernandez. (2016). Los microservicios no son para mí | Koalite. Retrieved May 20, 2018, from http://blog.koalite.com/2016/05/los-microservicios-no-son-para-mi/

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature reviews

in Software Engineering Version 2.3. *Engineering*, *45*(4ve), 1051. https://doi.org/10.1145/1134285.1134500

Klock, S., Werf, J. M. E. M. Van Der, Guelen, J. P., & Jansen, S. (2017). (P92) Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures. In *2017 IEEE International Conference on Software Architecture (ICSA)* (pp. 11–20). IEEE. https://doi.org/10.1109/ICSA.2017.38

Kookarinrat, P., & Temtanapat, Y. (2016). (P14) Design and implementation of a decentralized message bus for microservices. In *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)* (pp. 1–6). IEEE. https://doi.org/10.1109/JCSSE.2016.7748869

Lantz, C. A., & Nebenzahl, E. (1996). Behavior and interpretation of the κ statistic: Resolution of the two paradoxes. *Journal of Clinical Epidemiology*, *49*(4), 431–434. https://doi.org/10.1016/0895-4356(95)00571-4

Long, K. B., Yang, H., & Kim, Y. (2017). (P26) ICN-based service discovery mechanism for microservice architecture. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)* (pp. 773–775). IEEE. https://doi.org/10.1109/ICUFN.2017.7993899

Mazlami, G., Cito, J., & Leitner, P. (2017). (P82) Extraction of Microservices from Monolithic Software Architectures. In *2017 IEEE International Conference on Web Services (ICWS)* (pp. 524–531). IEEE. https://doi.org/10.1109/ICWS.2017.61

Messina, A., Rizzo, R., Storniolo, P., Tripiciano, M., & Urso, A. (2016). (P41) The Database-is-the-Service Pattern for Microservice Architectures (pp. 223–233). Springer, Cham. https://doi.org/10.1007/978-3-319-43949-5_18

Oberhauser, R. (2017). (P83) Enabling Agile Business Process Modeling to Orchestrate Semantically-Annotated Microservices (pp. 183–199). https://doi.org/10.1007/978-3-319-57222-2_9

Rademacher, F., Sachweh, S., & Zundorf, A. (2017). (P10) Differences between Model-Driven Development of Service-Oriented and Microservice Architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)* (pp. 38–45). IEEE. https://doi.org/10.1109/ICSAW.2017.32

Reynoso CKicillof N. (2004). Estilos y Patrones en la Estrategia de Arquitectura de Microsoft Estilos arquitectónicos. Retrieved from http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF

Safina, L., Mazzara, M., Montesi, F., & Rivera, V. (2016). (P17) Data-Driven Workflows for Microservices: Genericity in Jolie. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)* (pp. 430–437). IEEE. https://doi.org/10.1109/AINA.2016.95

Taylor, R. N., Medvidović, N., & Dashofy, E. M. (Eric M. (2010). *Software architecture : foundations, theory, and practice*. Wiley. Retrieved from http://proquest.safaribooksonline.com/book/software-engineering-and-development/9780470167748/applied-architectures-and-styles/service-oriented_architectures_and_web_s

Yale Yu, Silveira, H., & Sundaram, M. (2016). (P21) A microservice based reference architecture model in the context of enterprise architecture. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (pp. 1856–1860). IEEE. https://doi.org/10.1109/IMCEC.2016.7867539

Zimmermann, O. (2017). (P07) Microservices tenets. *Computer Science - Research and Development*, *32*(3–4), 301–310. https://doi.org/10.1007/s00450-016-0337-0

Zúñiga-Prieto, M., Insfran, E., Abrahao, S., & Cano-Genoves, C. (2016). (P77) Incremental integration of microservices in cloud applications. In *25th International Conference on Information Systems Development, ISD 2016*.