Universidad de la República
Facultad de Ingeniería
Instituto de Computación
Uruguay

Arquitectura de Microservicios para Plataformas de Integración

Ing. Andrés Nebel

Tesis de Maestría en Ingeniería de Software

Universidad de la República

Montevideo, Uruguay, Octubre de 2018

Director: MSc. Ing. Laura González
Co-Director: MSc. Ing. Guzmán Llambías
Universidad de la República

Resumen

La integración de sistemas heterogéneos se apoya comúnmente en Plataformas de Integración (PI). Estas plataformas consisten en infraestructura especializada, que provee mecanismos para resolver incompatibilidades entre sistemas con el fin de posibilitar su comunicación. En este ámbito, el avance y la expansión de la computación en la nube ha generado nuevos escenarios y requerimientos sobre las PIs en términos de escalabilidad y eficiencia, entre otros.

Por otro lado, la arquitectura de microservicios ha surgido recientemente impulsada por la industria, y está ganando creciente popularidad. Esta posee ventajas como el escalamiento independiente y la mantenibilidad que podrían beneficiar a las PIs en distintos escenarios. Por tanto, resulta de interés explorar las ventajas, desafíos y alternativas que introduce la arquitectura de microservicios en estas plataformas.

Esta tesis estudia la aplicabilidad de la arquitectura de microservicios para la construcción de PIs. Para esto se analiza, por un lado, el impacto de utilizar dicha arquitectura en PIs, determinando escenarios en los cuales es propicia su aplicación. Por otro lado, se proponen alternativas de arquitectura y diseño para la construcción de PIs basadas en microservicios, las cuales son evaluadas en base a distintos factores.

En relación al análisis de impacto, se plantea una metodología para determinar cómo es afectada una PI en base a sus atributos de calidad, obteniéndose del análisis dos resultados principales. Por un lado, se determina cómo impacta aplicar una arquitectura de microservicios en la calidad de la plataforma. Por otro, se determina un conjunto de características de los escenarios de integración (p. ej. cantidad alta de sistemas a integrar), que permite identificar si un escenario se beneficia al utilizar una PI basada en microservicios.

En relación a las propuestas de arquitectura y diseño, se presenta una propuesta principal aplicable a diversos escenarios, así como variantes aplicables a contextos específicos. La propuesta principal se evalúa en base a tres factores: i) patrones y buenas prácticas de microservicios, ii) atributos de calidad de la PI, y iii) la construcción de un prototipo.

Se concluye en este trabajo que la arquitectura de microservicios es aplicable para la construcción de PIs en escenarios con determinadas características y que las propuestas de arquitectura planteadas siguiendo este enfoque son técnicamente viables.

Palabras clave: plataformas de integración, arquitecturas de microservicios, atributos de calidad.

Contenido

1 Introducción

Esta tesis se posiciona en las áreas de Integración de Sistemas e Ingeniería de Software. En particular, estudia la aplicabilidad de arquitecturas de microservicios en Plataformas de Integración y propone soluciones de arquitectura en dicho ámbito.

Este capítulo presenta una introducción a la tesis. Concretamente, se describe el contexto y motivación de la problemática abordada, los objetivos trazados, los aportes del trabajo y cómo se organiza el resto del documento.

1.1 Contexto y Motivación

Los sistemas de software a gran escala, usualmente necesitan de la integración de sistemas heterogéneos. Su puesta en marcha se apoya comúnmente en Plataformas de Integración (PI), las cuales son infraestructuras informáticas especializadas que proveen mecanismos para facilitar la integración de sistemas [1]. Las PIs brindan soporte para la construcción de soluciones de integración basadas en los *Enterprise Integration Patterns* (EIP) [2][3] y aceptan pedidos en forma de mensajes, sobre los cuales realizan operaciones de mediación para resolver incompatibilidades entre sistemas [1].

Por otro lado, cada vez más organizaciones alojan sus datos y sistemas de software en la nube, utilizando sus distintas formas de servicio: Software como Servicio (SaaS), Plataforma como Servicio (PaaS) e Infraestructura como Servicio (IaaS)[4][5]. Esto ha originado nuevos escenarios de integración, motivando el surgimiento de PIs especializadas como las Plataformas de Integración como Servicio (iPaaS) [6]. La escala y complejidad de estos escenarios generan diversos requerimientos sobre la PI en términos de escalabilidad, mantenibilidad y eficiencia, entre otros [7].

En la Figura 1.1 se esquematizan los conceptos generales relacionados a la PI y su contexto.

Se observa que mediante soluciones de integración es posible integrar múltiples sistemas heterogéneos, los cuales pueden estar distribuidos en redes internas (integración interna) y en la nube (integración en la nube). Por otro lado, se observa que los sistemas a integrar pueden ser de múltiples empresas (escenario interorganizacional) o de una única empresa (escenario intraorganizacional). Además puede ser necesario integrar aplicaciones del tipo SaaS o productos de terceros en la nube (p. ej. *Salesforce CRM*[1]).

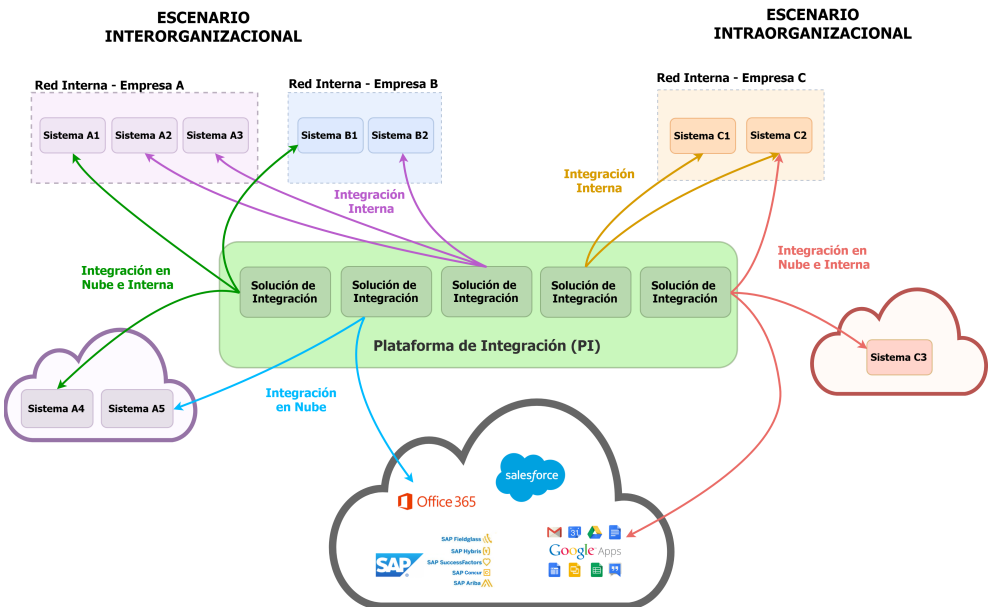

Figura 1.1: Conceptos generales relacionados a PI y su contexto.

Por otro lado, la arquitectura de microservicios ha surgido recientemente impulsada por la industria (p. ej. Amazon[2], Netflix[3] [8]) y está ganando creciente popularidad [9]. Esta consiste en un enfoque para desarrollar una aplicación como un conjunto de "pequeños" servicios, cada uno ejecutándose en su propio proceso y comunicándose con otros a través de mecanismos livianos como servicios web del tipo REST [9]. Los servicios se construyen en torno a funcionalidades de negocio y se pueden desplegar de forma individual, permitiendo automatizar completamente los mecanismos de despliegue [9].

La arquitectura de microservicios tiene varias ventajas, como la mantenibilidad y la posibilidad de escalar servicios de forma independiente, de acuerdo a la demanda concreta que tenga cada uno de ellos. Estas ventajas, entre otras, podrían resultar beneficiosas para abordar requerimientos de Plataformas de Integración. Sin embargo, también se ha señalado en varios

[1] https://www.salesforce.com
[2] https://www.amazon.com/
[3] https://www.netflix.com/

trabajos que la adopción de este tipo de arquitectura no es una solución general aplicable a todos los casos [10][11], ya que posee complejidades y problemáticas intrínsecas a sus propias características [12][13].

En base a lo anterior, resulta de interés explorar en qué contextos aplica utilizar una arquitectura de microservicios para la construcción de PIs; analizando los beneficios, desafíos y alternativas que surgen de dicho enfoque. Se plantea además investigar posibles propuestas de arquitectura de microservicios para PIs.

1.2 Objetivos

El objetivo general de la tesis es estudiar la aplicabilidad de una arquitectura de microservicios para la construcción de Plataformas de Integración. Para esto se plantean los siguientes objetivos específicos:

1. Relevar, analizar y caracterizar el conocimiento existente en las áreas de arquitectura de microservicios y Plataformas de Integración.
2. Analizar y determinar escenarios de integración donde pueda ser beneficioso utilizar una PI basada en microservicios.
3. Analizar cómo impacta aplicar una arquitectura de microservicios en las PIs y determinar concretamente cuáles escenarios son efectivamente beneficiados por la arquitectura.
4. Proponer alternativas de arquitectura y diseño para la construcción de PIs que estén basadas en microservicios.
5. Analizar las alternativas propuestas en relación a atributos de calidad de software (p. ej. escalabilidad, mantenibilidad) y evaluar la factibilidad técnica con la construcción de un prototipo.

1.3 Aportes

Se considera que los principales aportes de la tesis son los siguientes:

- Análisis del conocimiento existente en las áreas de microservicios y Plataformas de Integración.
- Caracterización de escenarios de integración y principales preocupaciones sobre los atributos de calidad de la PI.
- Análisis del impacto de aplicar una arquitectura de microservicios en Plataformas de

Integración. Se destaca además el enfoque metodológico basado en el propuesto en [14], para analizar el impacto sobre un sistema a partir de sus atributos de calidad.

- Conjuntos de características de referencia que contribuyen a determinar para un escenario si es propicio utilizar una PI basada en microservicios.
- Propuestas de arquitectura de microservicios para plataformas de integración, las cuales se presentan como una propuesta principal y propuestas alternativas que aplican a contextos particulares.
- Evaluación de la propuesta de arquitectura en base a tres factores: i) patrones y buenas prácticas de microservicios, ii) atributos de calidad de la PI , y iii) prototipado que contribuye a la validación técnica.

1.4 Organización de la Tesis

El resto del documento se organiza de la siguiente manera:

En el Capítulo 2 se presenta un resumen del conocimiento existente relevante para la problemática abordada en la tesis

En el Capítulo 3 se analiza la aplicabilidad de arquitecturas de microservicios en PI. En particular, se estudia cómo impacta la arquitectura a la PI y en cuáles contextos aplica la misma.

En el Capítulo 4 se presenta una propuesta de arquitectura de microservicios para PIs y se comentan posibles variantes para la misma.

En el Capítulo 5 se evalúan las propuestas y decisiones de diseño del Capítulo 4 en base a atributos de calidad, y patrones y buenas prácticas de microservicios. También se evalúa la factibilidad técnica en base a la construcción de un prototipo.

En el Capítulo 6 se presentan las conclusiones de la tesis y posible trabajo a futuro.

2 Conocimiento Existente

Se presenta en este capítulo un resumen del conocimiento existente considerado relevante para la problemática abordada en este trabajo. La organización del capítulo se describe a continuación.

En la Sección 2.1 se describen conceptos generales asociados a microservicios. Se destacan ventajas y problemas de la arquitectura de microservicios, aspectos de diseño relacionados y buenas prácticas de la arquitectura. En la Sección 2.2 se plantean los conceptos de contenedores en el marco de virtualización, incluyendo plataformas de orquestación de contenedores sobre clústeres. En la Sección 2.3 se presenta una caracterización y conceptos de Plataformas de Integración, comentando distintos tipos de PI relevantes. En la Sección 2.4 se describen los patrones de integración "*Enterprise Integration Patterns*". En la Sección 2.5 se presentan algunos conceptos relacionados a arquitecturas y calidad de software utilizados en la tesis. En la Sección 2.6 se analiza trabajo relacionado. Por último, en la Sección 2.7 se presentan las conclusiones del capítulo.

2.1 Microservicios

En esta sección se describen los conceptos generales asociados a microservicios. Concretamente, se presenta una descripción general, su granularidad definida, la relación con *DevOps*, ventajas y problemas de la arquitectura de microservicios y aspectos de diseño relevantes para la misma. Por último, se presentan también patrones y buenas prácticas relacionadas a microservicios.

2.1.1 Descripción General

El concepto de arquitectura de microservicios ha surgido recientemente impulsado por la industria (p. ej. Netflix[1], Amazon[2]) y está ganado creciente popularidad [15]. El enfoque refiere a una forma de diseñar software como un conjunto de pequeños servicios independientes que cooperan entre sí mediante protocolos livianos de comunicación [9]. Los servicios se construyen y despliegan de forma individual, y usualmente se apoyan en procesos de integración y despliegue continuo, cuyos conceptos se explican en secciones posteriores.

El termino microservicios (MS), a pesar de la atención recibida, carece aún de una definición estandarizada. Una de las caracterizaciones más citadas pertenece a Fowler y Lewis, que lo definen como "una forma de construir una aplicación como un conjunto de pequeños servicios, donde cada uno ejecuta en su propio proceso y se comunica mediante protocolos livianos" [3] [9]. Los autores agregan que cada servicio da soporte a una funcionalidad de negocio puntual y se despliega de forma independiente y automatizada. Mencionan también que para la implementación de cada servicio, pueden usarse diferentes tecnologías y persistencia de datos, lo cual permite utilizar la tecnología apropiada para cada funcionalidad.

Newman en [15] destaca por otro lado que cada microservicio además de ser autónomo, se encarga de una única tarea, de la cual es responsable. Esto se alinea al enfoque de Robert Martin para un buen diseño, denominado principio de responsabilidad única (*Single Responsability Principle* [16]). Para determinar cuáles microservicios son construidos y su responsabilidad, se descompone el dominio del problema en funcionalidades de negocio (i.e. *business capability*), en base a las cuales se construyen los microservicios.

Fowler y Lewis afirman que la arquitectura de microservicios cambia el enfoque tradicional de comunicación entre servicios, en contraste con arquitecturas orientadas a servicios (*Service Oriented Architecture*, SOA). En particular, la arquitectura de microservicios plantea simplificar el protocolo de comunicación, y mover la complejidad y el ruteo a los propios microservicios. Esto se contrapone a protocolos habituales en SOA como WS-Addressing o WS-Choreography donde la complejidad está en parte resuelta por el protocolo. Fowler y Lewis refieren al enfoque como "componentes inteligentes, comunicaciones tontas" (*"Smart endpoints, dumb pipes"* [9]). En dicho enfoque, se propone utilizar protocolos livianos como HTTP y REST, o mensajería con intermediarios simples como RabbitMQ[4] [9].

El manejo de los datos se realiza de forma descentralizada, separando el modelo de datos de forma análoga a la separación en microservicios [9]. Cada microservicio maneja su propia base de datos y su contenido, lo cual requiere consideraciones respecto a problemas de consistencia de datos, usuales en los sistemas distribuidos. El paradigma de microservicios, se basa en consistencia eventual (i.e. características BASE: *Basically Available, Soft state, Eventual consistency*) y no en transacciones distribuidas para manejar la consistencia [9, 17].

[1] https://www.netflix.com/
[2] https://www.amazon.com/
[3] Traducido al español por el autor. La definición original puede encontrarse en [9].
[4] https://www.rabbitmq.com/

La forma de coordinación y ejecución general de los microservicios (p. ej. coreografía u orquestación), cuyos conceptos se explican en secciones posteriores, no está especificada en las principales referencias de la arquitectura [9, 18, 17] y es aún motivo de discusión [19]. Sam Newman en [15] afirma que prefiere la ejecución coreográfica por su bajo acoplamiento y capacidad de cambio. Netflix sin embargo, usa un enfoque de orquestación y ofrece en su framework (*Netflix OSS*[5]) un componente de orquestación llamado *Netflix Conductor*[6]. Richardson en [20] no incluye esta decisión de diseño en sus patrones y buenas prácticas de microservicios, y afirma que depende de la complejidad de las transacciones que ejecuten los microservicios, donde los casos complejos deberían resolverse con orquestación y los simples con coreografía [19].

Es útil para caracterizar en más detalle la arquitectura de microservicios, compararla con otras, como la tradicional arquitectura "monolítica". Se denomina a un sistema "monolítico" cuando consiste en una unidad de código singular, que ejecuta en un único proceso y espacio de memoria. En estos casos, la implantación del sistema se realiza en forma íntegra como unidad, y este ejecuta en el mismo ambiente y proceso [9]. Además, las llamadas entre componentes (como ser clases, funciones o demás) son hechas con invocaciones de funciones en memoria. Una comparación básica entre ambas arquitecturas puede verse en la Figura 2.1.

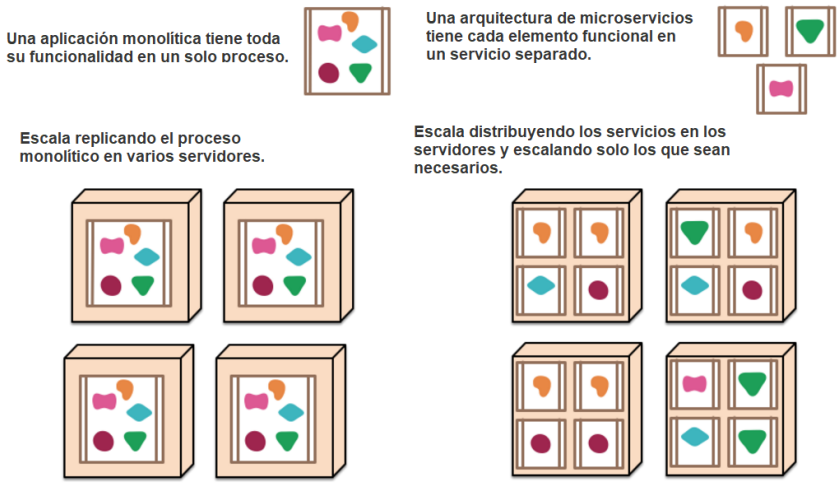

Figura 2.1: Comparación abstracta entre una arquitectura monolítica y una arquitectura de microservicios[7] [9].

Si bien la arquitectura monolítica es en determinados casos la opción más adecuada, la arqui-

[5] https://netflix.github.io/
[6] https://netflix.github.io/conductor/
[7] Imagen traducida al español por el autor. La original puede encontrarse en [9].

tectura de microservicios resuelve ciertas limitantes que presentan los tradicionales sistemas monolíticos. Se listan a continuación algunas de estas limitantes:

- Cambios en el sistema, sin importar su tamaño, requieren del despliegue del sistema entero [9].
- En general se utiliza la misma plataforma tecnológica para el sistema entero, en vez de tecnología especializada para cada servicio o tarea particular del sistema [9].
- Para realizar escalamiento, se debe escalar el sistema entero, incluso si la limitante (i.e. "cuello de botella") son componentes puntuales [9].
- Al aumentar el tamaño del sistema, la curva de aprendizaje para nuevos desarrolladores también aumenta, así como la complejidad de la gestión de configuración, pruebas y las tareas de mantenimiento, elevando los costos del sistema [17].
- Actualizar la tecnología en la cual se basa el sistema o sus dependencias, es a menudo problemático, lo cual evita que el sistema reciba beneficios de actualizaciones o tecnologías nuevas [21].
- Al aumentar el tamaño del sistema, también aumenta la complejidad del mismo, lo cual puede hacer del mantenimiento una tarea compleja [9].
- Al aumentar el tamaño del sistema, la coordinación entre los equipos de operaciones y desarrollo en una empresa, necesita ser alta [17].
- Al ser el sistema una unidad, se convierte en un único punto de falla [22].

Segun Fowler y Lewis: "Las aplicaciones monolíticas pueden ser exitosas, pero algunas de sus limitantes han generado frustraciones, especialmente al aumentar el despliegue de aplicaciones en la nube"[8] [9]. Por otro lado, construir el sistema como una conjunción de microservicios, resuelve varias de las limitaciones mencionadas [9, 17]:

- El impacto del mantenimiento es menor, ya que solo se actualiza y despliega los microservicios involucrados en el cambio.
- El escalamiento es más eficiente, ya que se hace un escalamiento selectivo de los microservicios, sin tener que escalar el sistema entero.
- Cada microservicio puede usar la tecnología más apropiada para su tarea, siempre que exponga una interfaz adecuada.
- No existe un único punto de falla y se usan patrones para controlar la comunicación a microservicios inoperantes (i.e. patrón *Circuit Breaker*).
- Los microservicios al ser ejecutados de forma independiente pueden ser recombinados y reutilizados para nuevos sistemas, reduciendo tiempos de desarrollo.

[8] Traducido al español por el autor. El comentario original puede encontrarse en [9].

- Gestión de la configuración de mayor granularidad y control.

En sistemas monolíticos, el desarrollador a menudo tiene un mal conocimiento del sistema en su totalidad, haciéndolo propenso a introducir defectos. En implementaciones de microservicios, el desarrollador conoce la tarea que el servicio tiene que hacer y tiene un conocimiento más completo de esta, ya que el microservicio tiene una tarea bien definida y desacoplada.

2.1.2 Granularidad del Microservicio

Para diferenciar microservicios de servicios web tradicionales, es importante definir el tamaño[9] que debe tener un microservicio para que se identifique como tal. Esta es una decisión de diseño relevante: si la separación es demasiado granular, entonces problemas identificados en secciones anteriores (p. ej. latencia) pueden agravarse y degradar la calidad del sistema. Por otro lado, si los microservicios son demasiado grandes, el sistema se comporta de forma similar a un monolítico.

Jeff Bezos fundador de Amazon, propone el tamaño ideal de un microservicio en base a la cantidad de integrantes del equipo encargado de su ciclo de vida. La propuesta, nombrada "*The two pizza team rule*", establece como medida que debe ser posible alimentar el equipo responsable con dos pizzas regulares [17]. Sin embargo, Fowler comenta que dicha definición es de las medidas de tamaño más grandes reportadas, y comenta que empresas poseen equipos de hasta seis personas manteniendo seis microservicios [9]. Lo cual implica la medida de una persona manteniendo un microservicio.

Sam Newman en [15] toma como referencia una caracterización de Jon Eaves, afirmando que debe ser posible reescribir un microservicio en dos semanas o de lo contrario su tamaño es demasiado grande.

IBM en [17] afirma que la forma de definir el tamaño de un microservicio es mediante las necesidades de negocio, asignando a cada microservicio una y solo una responsabilidad de negocio bien definida y limitada.

En [23] se estudia el impacto de esta decisión, contrastándola con la satisfacción de los requerimientos no funcionales. El trabajo propone además una adaptación automática de la granularidad de microservicios en tiempo de ejecución.

En base a la literatura existente, se concluye que no existe un consenso acerca del tamaño ideal de un microservicio. Existen algunas propuestas hechas por distintos autores, pero son en general poco precisas.

[9] Se hace referencia a tamaño como la cantidad de funcionalidades o lineas de código del servicio.

2.1.3 DevOps

El concepto de DevOps[10], refiere a cambios organizacionales que modifican el manejo del ciclo de vida de los sistemas. Concretamente, es un conjunto de prácticas que apuntan a reducir los costos de implantación y el tiempo transcurrido entre que un cambio en el software es realizado y luego puesto en producción [17]. El paradigma de DevOps, ha sido nombrado en diversos artículos como prerrequisito de la construcción de sistemas basados en microservicios [17, 24, 11].

DevOps se apoya fuertemente en automatizaciones de procesos cuyos conceptos se muestran en la Figura 2.2. Por un lado la "integración continua" consiste en automatizar las pruebas unitarias, la integración de los desarrollos al código principal y las pruebas de dicha integración [25]. La "entrega continua" considera los aspectos anteriores y los extiende, desplegando el sistema modificado a un ambiente de pruebas para efectuar pruebas de validación. Sin embargo el concepto de "entrega continua" no automatiza el pasaje a productivo final, ya que se valida previamente por el cliente, permitiendo control sobre los cambios productivos [25]. Por último, el caso donde el despliegue a productivo también se automatiza, se conoce como "despliegue continuo" [25].

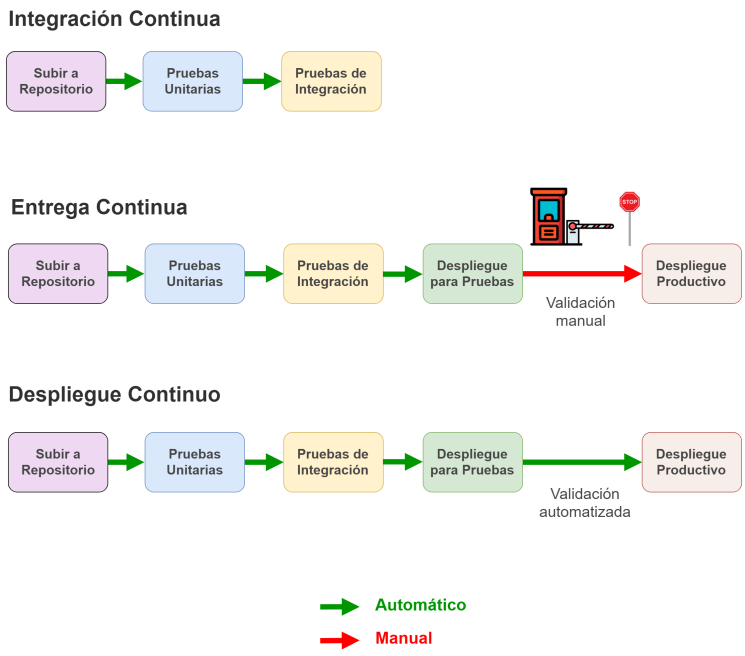

Figura 2.2: Comparación entre conceptos de automatización[11].

[10] Contracción de las palabras "*Development*" (desarrollo) y "*Operations*" (operaciones).
[11] Esquematizado por el autor en base a los conceptos de [25].

Las automatizaciones descritas anteriormente son importantes en arquitecturas de microservicios. Esto se debe a que la cantidad de partes del sistema que introduce el concepto de microservicios genera un aumento del esfuerzo de operaciones, que debe ser automatizado para evitar un impacto a nivel de proyecto.

Las prácticas de DevOps sugieren dividir el equipo de desarrollo en pequeños equipos de trabajo encargados del ciclo de vida entero de una parte del sistema [17]. Las prácticas de DevOps aplicadas a sistemas basados en microservicios implican que equipos pequeños se encargan del ciclo de vida entero de uno o más microservicios [24]. Al reducirse el equipo, la comunicación se simplifica, teniendo una gestión más efectiva. Aun así, se requiere una coordinación general de los equipos cuya complejidad se mantiene.

2.1.4 Problemas de la Arquitectura de Microservicios

Varios autores han afirmado que la arquitectura de microservicios no es una solución única aplicable a todos los requerimientos [9, 17, 20]. Esta presenta desafíos y problemas a considerar, que hacen que su elección para un sistema deba ser evaluada cuidadosamente.

Uno de los desafíos de la arquitectura es la complejidad operacional [21]. Esta se debe a la necesidad de gestionar y mantener una cantidad considerable de microservicios, lo cual motiva a las empresas a la aplicación de *DevOps*. También, impulsa al uso de estrategias relacionadas como despliegue continuo y entrega continua [24]. Dichos enfoques poseen una curva pronunciada de aprendizaje, debido al cambio que implican a nivel organizativo y también al uso de tecnología nueva que apoya los procesos.

La arquitectura de microservicios también posee problemas relacionados a la consistencia eventual [21], ya que usa este enfoque para la persistencia descentralizada. Así, es necesario mitigar los problemas de datos desactualizados y controlar posibles decisiones que tome el código en base a ellas. También se deben contemplar las demoras en lograr la consistencia de datos y la propia complejidad de implementación del enfoque [21].

Por otro lado, se heredan problemas relacionados a los sistemas distribuidos [9, 21]:

- Latencia relacionada a comunicación por red en vez de invocaciones en memoria.
- Problemas de arrastre de fallas (i.e. fallas en cascada).
- Complejidad de supervisión del sistema (i.e. *monitoring*).
- Complejidad en la identificación de las fallas y problemas.

El problema de la latencia está dado por múltiples factores. Fowler recomienda disminuir la comunicación entre microservicios (p. ej. enviando datos en tandas de datos) y usar comunicación asíncrona [21]. El problema debe atacarse también a nivel de diseño: la separación excesiva en microservicios podría agregar problemas de latencia no solucionables mediante escalamiento [21]. Esto se debe a que cada microservicio nuevo adiciona:

- Latencia de una nueva comunicación por red.
- Latencia del empaquetado y procesamiento de mensajes para el uso de JSON (i.e. *marshaling*) .
- Latencia de aplicar seguridad a la información (p. ej. encriptación al usar SSL/TLS).
- Posibles demoras por fallas temporales o intermitencias a nivel del protocolo TCP/IP.

2.1.5 Tolerancia a Fallas

Los sistemas basados en microservicios deben diseñarse teniendo en cuenta posibles fallas causadas por la separación del sistema. Concretamente, los sistemas distribuidos implican problemas de arrastre de fallas (errores en cascada) y requieren más comunicaciones de red, las cuales pueden fallar [17]. Estos problemas deben ser tenidos en cuenta al construir dichos sistemas [26]. Existe una cobertura considerable en este tema, y varios enfoques y tecnologías han sido propuestos [18, 17, 26]. Se mencionan a continuación los mas destacables.

Un patrón existente para el manejo de fallas y la prevención de fallas en cascada es "*Circuit Breaker*" [27]. Este actúa de intermediario entre cliente y servicio, permitiendo evitar invocaciones innecesarias a un servicio que no está disponible o que se encuentra en estado de error [17]. Actúa de forma similar a una llave eléctrica que abre o cierra un circuito: cuando ocurre un número de fallas en un servicio el componente *Circuit Breaker* se "abre" y evita futuras invocaciones al servicio (ver Figura 2.3).

Las propiedades que usualmente se configuran en este patrón son [26]:

- *Tiempo de expiración*: Tiempo que espera el componente *Circuit Breaker* antes de dar una invocación por fallida si no hay respuesta.
- *Reintentos*: Cantidad de reintentos que puede hacer el componente *Circuit Breaker* antes de transmitir la falla al cliente.
- *Umbral*: Cantidad de fallas que provocarían que el componente *Circuit Breaker* pase a estado abierto.
- *Ventana*: Tiempo que permanece el componente *Circuit Breaker* en estado abierto antes de intentar nuevas invocaciones.

Por otro lado el patrón "*Bulkhead*", permite aislar los componentes de forma que no afecten el sistema entero [27]. Esto usualmente se realiza limitando la cantidad de llamadas concurrentes a un servicio, lo cual evita que este se extralimite y falle. Se suele utilizar la analogía de un barco para describir el patrón: si una parte del barco se daña, se cierran los compartimientos adecuados para aislar el daño y no hundir el barco entero [17].

Existen múltiples enfoques para el patrón *Bulkhead*, ya que en cierta forma es un concepto

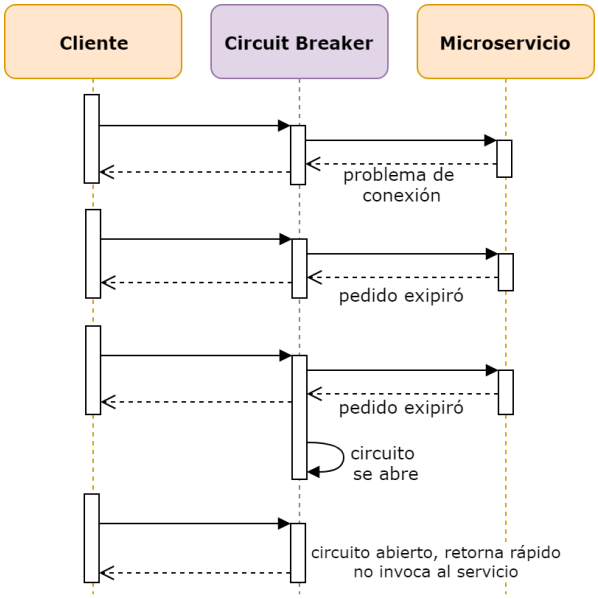

Figura 2.3: Interacción entre cliente, *circuit braker* y microservicio[12][26].

abstracto. Otra forma de implementarlo es simplemente escalar horizontalmente, para que si falla una réplica las otras mantengan la disponibilidad del sistema [27]. También se implementa limitando los recursos de *hardware* que puede consumir un servicio.

Desde el punto de vista tecnológico, existen bibliotecas pensadas específicamente para dar soporte a la tolerancia de fallas en sistemas distribuidos basados en servicios. Por un lado, se destaca "*Hystrix*"[13] del proyecto OSS de Netflix, el cual aporta tolerancia a fallas y latencia. Esta favorece la aplicación de los patrones de *Circuit Breaker* y *Bulkhead*, además de aportar un manejo general de fallas en cascada, expiración de llamados y reintentos. La biblioteca "*Polly*"[14] ofrece funcionalidades similares y tiene como ventaja que aporta manejo de caché.

2.1.6 Orquestación y Coreografía

En las arquitecturas de microservicios, las funcionalidades se implementan mediante la composición de invocaciones a distintos microservicios. Existen dos estrategias para coordinar esta composición: orquestación y coreografía [28].

La orquestación de servicios se basa en la existencia de una entidad centralizada (denominada

[12] Imagen traducida al español por el autor. La original puede encontrarse en [26].
[13] https://github.com/Netflix/Hystrix
[14] https://github.com/App-vNext/Polly

"orquestador") que coordina las invocaciones a los servicios combinando las respuestas de uno con las entradas de otro, para implementar las funcionalidades del sistema. La figura 2.4 ilustra este concepto.

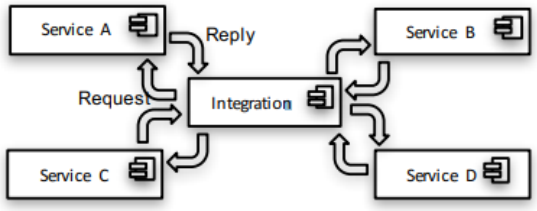

Figura 2.4: Orquestación de servicios [28]

La coreografía de servicios, como muestra la Figura 2.5, presenta un modelo descentralizado en el que no existe una entidad central encargada de coordinar la composición. Una coreografía se describe mediante el intercambio de mensajes entre servicios y reglas de interacción acordadas entre ellos. Cada microservicio conoce qué tarea debe hacer y a quién debe invocar, y ninguno conoce la composición completa.

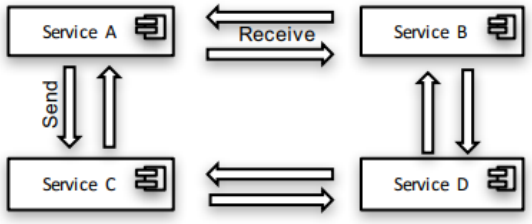

Figura 2.5: Coreografía de servicios [28]

2.1.7 Gestión de Logs

Uno de los problemas a resolver en sistemas basados en microservicios, es recolectar la información de ejecución de los mismos (i.e. *logs* de sistema) para proporcionar información que asista a la supervisión del sistema. Cuando la ejecución no es orquestada, la información de ejecución está dispersa y debe ser recolectada en cada microservicio. Esto puede hacerse de varias formas, según la tecnología y la plataforma elegida.

Netflix en el framework para microservicios *Netflix OSS*, sugiere el uso de Blitz4j[15]. Sin embargo esta solución implica que el lenguaje de escritura de los microservicios debe ser *Java*.

[15] https://github.com/Netflix/blitz4j

En [29] los autores de la PI de propósito general *Elastic.io* discuten el problema y sugieren algunas formas de almacenar y recolectar los logs, de forma independiente del lenguaje del componente. En concreto, se discuten las siguientes opciones:

- Montar un volumen en el contenedor (para almacenar los logs) desde donde luego recuperar los registros.
- Enviar los registros a *Syslog*[16] y que luego sean consumidos por un concentrador.
- Usar los canales estándares de salida *stdout/stderr*, concentrar dicha información y enviarla a persistencia externa.

En *Elastic.io* se utiliza la última opción con una solución de código abierto que envía los registros a una persistencia externa, ya que identifican problemas de acoplamiento y limitación de espacio en las otras soluciones. El análisis completo puede verse en [29].

Algunas plataformas de orquestación de contenedores como *Openshift*[17] proveen una solución análoga con herramientas complementarias como *fluentd*[18]. En dicho caso, la información de los canales estándar es concentrada en archivos JSON o mediante sustitutos de *Syslog* especializados como *journald*[19] y luego indizada y enviada a una persistencia externa especializada (p. ej. *elasticsearch*[20]).

2.1.8 Patrones y Buenas Prácticas

Es de interés considerar un conjunto de patrones y buenas prácticas de arquitecturas de microservicios. De esta forma se puede tomar una propuesta de arquitectura de microservicios y comparar si resuelve los problemas de una forma ya estudiada y validada.

Richardson presentó un conjunto de patrones para microservicios en [20] que abarca varios aspectos de diseño como: descomposición en microservicios, variantes de despliegue, persistencia, control de fallas y seguridad, entre otros. Un diagrama completo de los patrones a la fecha puede observarse en la Figura 2.6.

En el artículo de Fowler y Lewis [9] se destacan algunas buenas prácticas generales para la arquitectura de microservicios (p. ej. "componentes inteligentes, comunicaciones tontas" mencionado en secciones anteriores), pero son de escaso detalle.

Un conjunto pequeño de buenas prácticas usadas por Netflix es presentado en [30]. En particular, tiene puntos en común con Richardson, proponiendo una base de datos por microservicio y despliegue en contenedores. Difiere en proponer patrones referentes al ciclo de vida

[16] *Syslog* es un estándar para almacenar registros de ejecución. Permite desacoplar la entidad que genera los registros del almacenamiento.
[17] https://openshift.com/
[18] https://www.fluentd.org
[19] https://www.loggly.com/blog/why-journald/
[20] https://www.elastic.co/products/elasticsearch

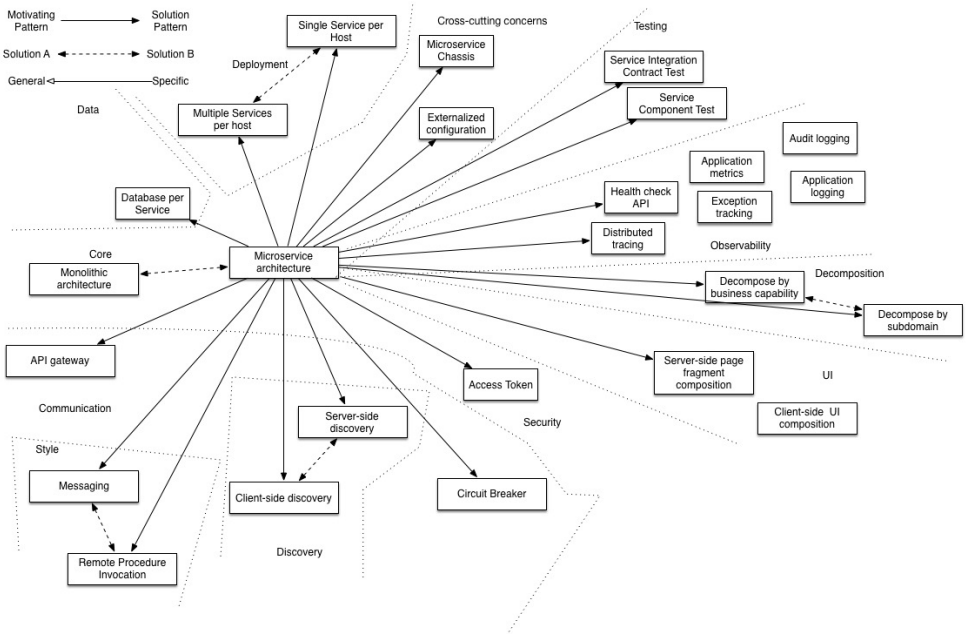

Figura 2.6: Patrones de Richardson, relacionados a arquitecturas de microservicios [20].

del microservicio, referentes al ensamblaje y gestión de nuevas versiones.

En el trabajo de Zdun et al. [31] se propone un conjunto de métricas y restricciones para evaluar el nivel de conformidad, por parte de una arquitectura, con algunos de los patrones de Richardson. Sin embargo, dicho trabajo se enfoca solamente en los patrones de descomposición y no en todos los propuestos en [20].

Un conjunto reducido de patrones para microservicios también es propuesto por Gupta en [32].

El equipo de Heroku en [33], propone una serie de prácticas a nivel de construcción de software para que servicios independientes sean exitosos y se adapten al despliegue en la nube y al concepto de Software as a Service (SaaS).

Finalmente, se entiende que los patrones de Richardson mencionados, corresponden a la fecha al conjunto más completo de patrones y buenas prácticas de microservicios. Se describen a continuación algunos de sus patrones más destacables:

- *Descomposición por capacidad de negocio*: Implica que la descomposición en microservicios debe ser realizada en base a capacidades de negocio identificadas en el dominio del problema. Corresponden usualmente a objetos del dominio.

- *Base de datos por servicio*: Cada microservicio debe tener su propia base de datos.
- *Instancia de servicio por contenedor*: Cada instancia de microservicio debe ser desplegada en un contenedor.
- *Circuit Breaker*: Usar la metodología de corte de circuito (i.e. *Circuit Breaker*), para prevenir fallos en cascada, además de evitar consumir instancias de servicio que ya no responden.
- *Health Check API*: Los microservicios deben exponer una interfaz que permita determinar si el servicio está operativo.
- *API Gateway*: La interfaz no debería acceder a múltiples puntos de entrada para consumir servicios, se debe usar un intermediario que actúe como único punto de entrada.

2.2 Contenedores y Virtualización

En esta sección se describen conceptos relacionados a contenedores y plataformas para gestionar y ejecutar los mismos.

2.2.1 Descripción General

El despliegue de software en la nube, a menudo se apoya sobre tecnologías de virtualización y las llamadas máquinas virtuales ("VM", del inglés *Virtual Machine*) [34]. Estas contienen no solo el sistema desplegado y sus dependencias, sino también virtualizaciones de hardware, un sistema operativo, servidores web, y las configuraciones hechas sobre estos [35].

Cada VM, posee su propio sistema operativo y ambiente de ejecución, lo cual implica cierta redundancia y produce un uso ineficiente de recursos [36], que se acentúa si la VM es escalada horizontalmente. Además, el tiempo de inicio[21] de una VM (y sus réplicas), es afectado por el tiempo de inicio de su propio sistema operativo [36]. Esto ha motivado la aparición de otras formas de virtualización más livianas, como los contenedores.

Los contenedores, a diferencia de las VMs, no requieren un sistema operativo dedicado para cada contenedor. En esencia, contienen solamente los binarios, bibliotecas y *middleware* necesarios para ejecutar la aplicación desplegada [36]. Estos son empaquetados en una "imagen" (i.e. *container image*) que es administrada y ejecutada por un gestor de contenedores (p. ej. Docker). La tecnología permite que el despliegue sea independiente del ambiente y del sistema opeartivo del servidor [37]. En la Figura 2.7 se muestra una comparación a alto nivel entre los conceptos de VM y contenedor.

[21] Tiempo en que una réplica inicia el sistema hasta que el software desplegado queda operativo.

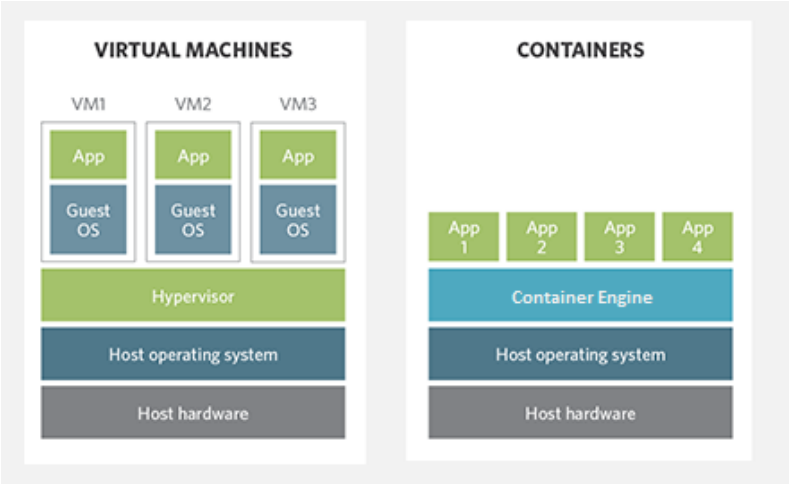

Figura 2.7: Comparación de VM y contenedores[22][38]

De la gestión y ejecución de los contenedores se encarga un gestor de contenedores (p. ej. Docker[23] [39]). Estos a menudo se apoyan en funcionalidades de compartimentación del sistema operativo. Por ejemplo en el caso de sistemas Unix se basa en *LXC* o *Linux VServer*, entre otros [36]. Los contenedores operan sobre una virtualización a nivel de sistema operativo, en contraste con la virtualización a nivel de hardware de las VMs [36].

El concepto de contenedores es importante para microservicios. Debido a su rápido tiempo de inicio [37], es posible escalar rápidamente, así como reiniciar contenedores frente a fallas, con bajo impacto en la disponibilidad. Además, disminuyen el consumo de recursos en comparación con las máquinas virtuales y aportan independencia del ambiente donde es desplegado.

2.2.2 Plataformas de Orquestación de Contenedores sobre Clústeres

Existen plataformas que combinan la gestión y orquestación de contenedores con manejo de clústeres (p. ej. Kubernetes[24], Docker Swarm[25]). Estas brindan funcionalidades importantes para las arquitecturas de microservicios [40]. En esta sección, se detallan algunas de ellas.

Por un lado, las plataformas mencionadas permiten ejecutar contenedores sobre clústeres, así como administrar el estado de los clústeres, gestionar los volúmenes de almacenamiento asignados, y la planificación de ejecución (i.e. *scheduling*) [40].

[22] Imagen modificada para generalizar el gestor de contenedores. La original puede encontrarse en [38]
[23] https://www.docker.com
[24] https://kubernetes.io/
[25] https://docs.docker.com/engine/swarm/

Por otro lado, permiten administrar en forma grupal los contenedores, los cuales pueden replicarse para responder a demandas de escalamiento. Usualmente las plataformas proveen también un balanceador de carga, y resuelven temas de conectividad y asociación de puertos [40]. A modo de ejemplo, en la plataforma Kubernetes, un grupo de contenedores operando en forma conjunta es denominado *Pod* [41]. Los *Pods* pueden ser replicados, dados de baja, o escalados automáticamente. Así, pueden existir muchos *Pods* que consistan en una réplica de un mismo servicio, balanceado por un balanceador de carga.

En base a las funcionalidades anteriores, es también posible configurar a las plataformas de orquestación de contenedores sobre clústeres, para que controlen que un cierto número de réplicas de un servicio estén operativas. De esta forma, si dicho número es reducido por motivos de falla, la plataforma inicia un contenedor nuevo (o grupo de contenedores) para cubrir la demanda. Esto contribuye a la alta disponibilidad del sistema y a la recuperabilidad frente a fallas [40].

Existen además plataformas similares a la descrita, pero orientadas a DevOps (p. ej. OpenShift[26], Heroku[27]) que se basan en funcionalidades de las plataformas de orquestación de contenedores sobre clústeres, y las complementan con herramientas de DevOps y funcionalidades de entrega continua. Estas se centran en el ciclo de vida de las aplicaciones y están orientadas a desarrolladores, permitiendo entre otros, integración con un repositorio y automatizar el despliegue y pruebas. Las plataformas son comercializadas en modalidad PaaS (p. ej. Heroku) o como software a instalar sobre un servidor o clúster propio (p. ej. OpenShift Origin).

2.3 Plataformas de Integración

2.3.1 Descripción General

En este trabajo, se usa la caracterización de Plataforma de Integración (PI) propuesta por González y Ruggia en [1], donde también se describe la arquitectura tradicional de las PIs. En esta, se describe a la PI como una infraestructura informática especializada, que brinda mecanismos de mediación y conectividad confiable, con el fin de facilitar la integración de sistemas heterogéneos en ambientes distribuidos. Para esto, las PIs aceptan pedidos en forma de mensajes, sobre los cuales se realizan operaciones de mediación para solucionar heterogeneidades entre sistemas [1].

Las PIs brindan soporte para la construcción de Soluciones de Integración (SI), las cuales están compuestas de una serie de componentes que efectúan operaciones de mediación (p. ej. trasformación) generalmente basándose en los *Enterprise Integration Patterns* (EIP) [2][3]. En algunos trabajos, el concepto de SI se lo considera bajo el nombre de "flujo de integración"

[26] https://www.openshift.org/
[27] https://www.heroku.com/

[6]. Las PIs complementan las funcionalidades de integración con servicios de infraestructura, aportando entre otros seguridad, supervisión (i.e. *monitoring*), conectividad y gestión.

En el trabajo de González y Ruggia [1] los componentes de las PIs que aportan funcionalidades de integración, se organizan en tres categorías: componentes de integración de base, avanzados e interorganizacionales. Los componentes de integración de base son tres: conectividad, mediación y mensajería; aunque pueden también desglosarse en mayor detalle [1]:

- *Conectores Simples*: Refiere a conectores que ofrecen una conectividad básica sobre protocolos estándar, como ser HTTP o SMTP.
- *Conectores Específicos*: Refiere a conectores que resuelven conectividad compleja. Esto puede ser interacciones particulares o protocolos específicos de sistemas (p. ej. RFC[28] en sistemas SAP[29]).
- *Virtualización de Servicios*: Permite la exposición de servicios web para obtener o enviar información a los sistemas a integrar vinculados a la PI.
- *Transformación*: Permite a la PI efectuar transformaciones de datos como las definidas en los EIP. Un ejemplo es la transformación de modelo de datos.
- *Ruteo*: Permite a la PI efectuar operaciones de ruteo de datos como las definidas en los EIP. Un ejemplo es el ruteo basado en contenido, donde se determina el destinatario en base a propiedades de los mensajes.
- *Supervisión*: Permite a la PI efectuar operaciones de supervisión (i.e. *monitoring*) sobre los datos procesados en la PI.
- *Mensajería*: Permite a la PI efectuar operaciones de mensajería como las definidas en los EIP. Usualmente es resuelto por *Middleware* orientado a mensajería (i.e. MOM, *Message-oriented middleware*).

Los componentes avanzados e interorganizacionales, no son considerados en esta tesis y refieren en su mayoría a extensiones de las prestaciones habituales de las PIs, como definición de SLA y como las extensiones propuestas por los trabajos [42, 43, 44, 45]. Las relaciones entre componentes pueden verse en la Vista Lógica de dicho trabajo, mostrada en la Figura 2.8 [1].

[28] *Remote Function Call* es un protocolo de comunicación propietario de SAP.
[29] https://www.sap.com

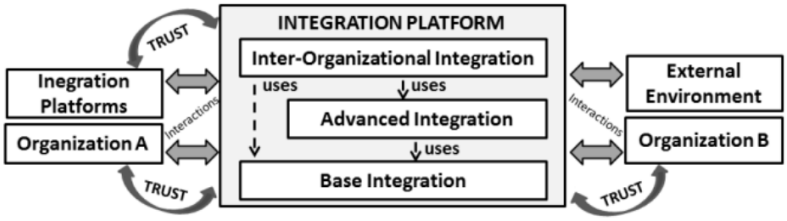

Figura 2.8: Vista Lógica en el trabajo de González y Ruggia [1]

2.3.2 Tipos de Plataformas de Integración

Existen varias categorizaciones para las PI, y nuevos tipos de PI han surgido recientemente (p. ej. "iPaaS", *Integration Platform as a Service*) que contemplan nuevos escenarios de integración [6]. En particular, la computación en la nube ha alcanzado un alto nivel de aceptación [5], acompañado de un aumento del uso de sus formas de servicio (véase Apéndice B), y de sistemas empresariales desplegados en la nube [5]. Se tienen por tanto nuevos escenarios, donde algunos (o todos) los sistemas a integrar pueden estar desplegados en la nube.

Gartner en [46] reconoce la aparición de nuevos escenarios, y lista nueve tipos de software de integración actuales que soportan la integración híbrida:

- *Api Management*
- *On-premise integration suites* (p. ej. ESB)
- Software de integración B2B (*Business to Business*)
- *Data integration tools* (productos de ETL, *Extract, transform and load*)
- *Integration Frameworks* (p. ej. Apache Camel[30])
- iPaaS (*Integration Platform as a Service*)
- iSaaS (*Integration Software as a Service*)
- BaaS y MBaaS (*Backend as a Service*, *Mobile Backend as a Service*)
- Plataformas IoT

En la Figura 2.9 Gartner describe, para los conceptos principales de escenarios híbridos, el nivel de satisfacción de requerimientos que brinda cada tipo de software de integración. Aún así, la mayoría no se alinea a la caracterización de PI descrita en la sección anterior (basada en [1]). Por ejemplo, Apache Camel es un framework de integración basado en los EIP, pero que no posee prestaciones de plataforma como supervisión (i.e. monitoring), gestión

[30] http://camel.apache.org/

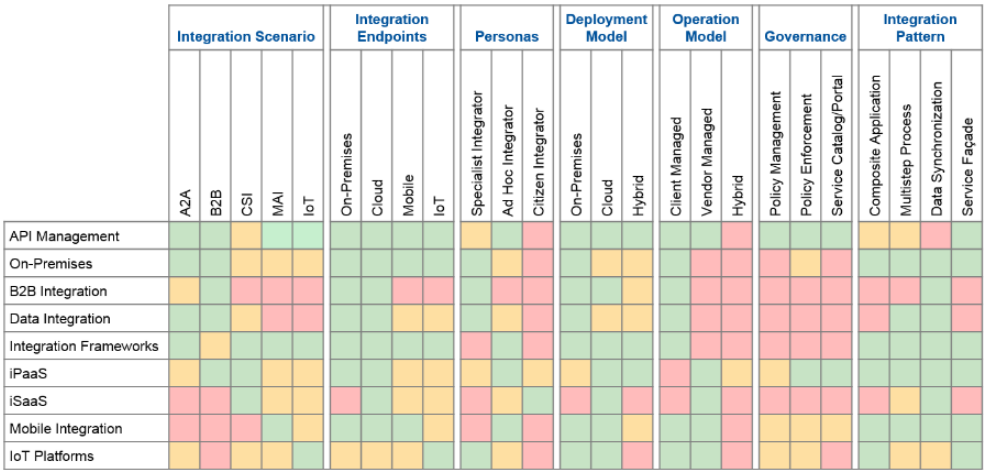

Figura 2.9: Tipos de software y satisfacción de requerimientos para conceptos de escenarios híbridos [46]

centralizada, entre otros.

En [3] se hace una categorización similar de sistemas de integración, en donde se enfatizan las diferencias entre ESB, suite de integración y framework de integración. En la misma no se considera el término de plataforma de integración, aunque los términos de ESB e *Integration Suite* se alinean al de PI descrito en secciones anteriores. En la Figura 2.10 se puede ver un diagrama representativo de los conceptos de dicho artículo.

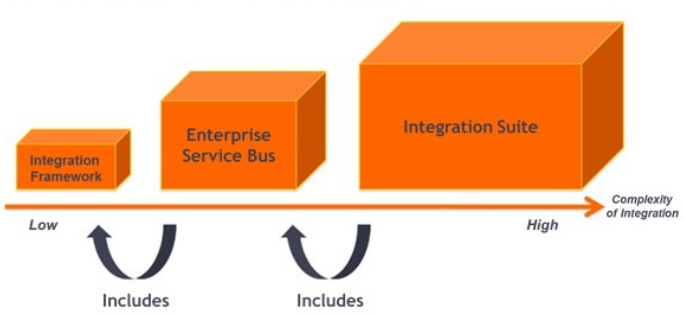

Figura 2.10: Comparación entre ESB, framework de integración y suite de integración [3].

De los tipos de PI que existen en el mercado, se describen a continuación dos en partícular, que se apegan más a las caracterizaciones del trabajo de González y Ruggia [1], sobre el cual se apoya este trabajo: iPaaS y ESB.

2.3.2.1 Plataformas de Integración como Servicio (iPaaS)

Las plataformas de integración como servicio (iPaaS), son un tipo de PI que han surgido recientemente como una evolución de la integración como servicio [6]. Los conceptos principales se esquematizan en la Figura 2.11. Gartner las define como:

> "(...) una suite de servicios en la nube que permite desarrollar, ejecutar y gestionar flujos de integración, para conectar procesos, servicios, aplicaciones o datos alojados tanto en redes internas como en la nube, para una o múltiples organizaciones[31]." [6].

De acuerdo a Gartner, los flujos de integración, son una conjunción de software personalizado y metadatos que permiten integrar múltiples sistemas heterogéneos, mediante operaciones como transformación, ruteo, conversiones de protocolo, virtualización de servicios, entre otros [6].

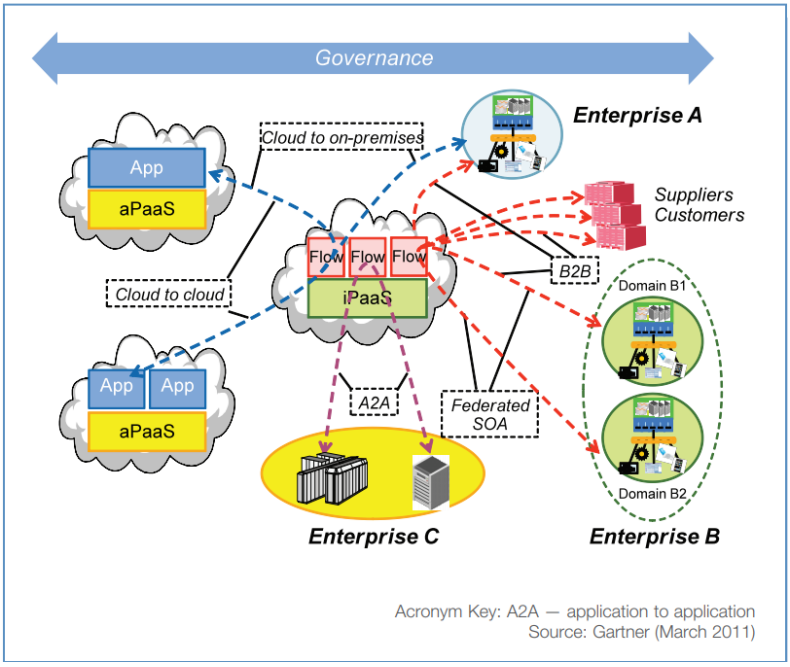

Figura 2.11: Resumen conceptual de las iPaaS y posibles escenarios de integración [6].

El servicio de iPaaS es ofrecido en modalidad *multitenant*[32], donde se asegura integridad

[31] Traducido por el autor, el original puede encontrarse en [6].

[32] *Multitenancy* es una forma de operación de un sistema donde múltiples instancias independientes de un mismo software operan en un ambiente compartido. Las instancias están lógicamente aisladas entre sí y esto es usualmente utilizado para soportar usuarios u organizaciones en forma aislada.

y seguridad de datos, así como escalabilidad elástica de la plataforma. Las iPaaS proveen además servicios de administración general (i.e. *governance*), supervisión (i.e. *monitoring*), manejo de repositorios, manejo de ciclo de vida de los flujos, manejo de políticas, entre otros.

Los proveedores de productos de iPaaS suelen ofrecer la plataforma en dos formas: como servicio y de forma independiente, para ser desplegado en forma interna (p. ej. SnapLogic). Gartner denomina como "CEIP" (*Cloud-enabled integration platform*) a esa alternativa [6]. Gartner en [6] recomendó inicialmente abordar las integraciones con iPaaS cuando al menos uno de los sistemas fuera alojado en la nube.

2.3.2.2 Enterprise Service Bus

El *Enterprise service bus* (ESB) es definido por Chappell como:

> "una plataforma de integración basada en estándares, que combina mensajería, servicios web, transformación de datos y ruteo inteligente, para coordinar y conectar las interacciones de números significantes de aplicaciones, en empresas extendidas [refiere a empresas y sus proveedores] y con integridad transaccional." [47]

También ha sido considerado como patrón de arquitectura por algunos autores [48]. En la Figura 2.12 se muestra un diagrama conceptual básico.

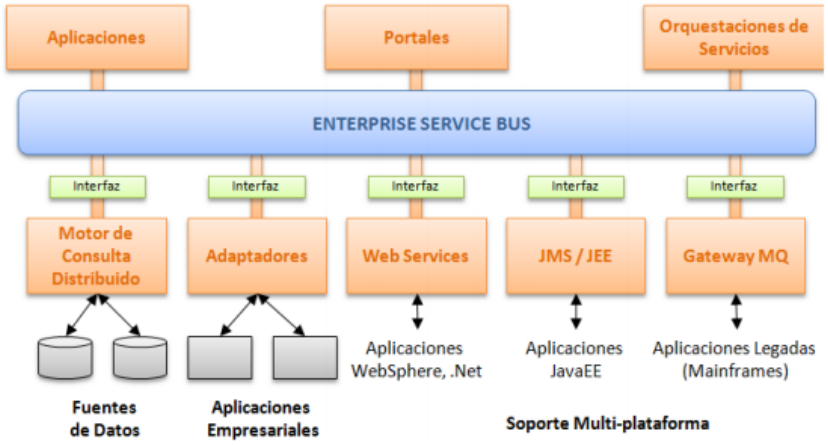

Figura 2.12: Diagrama conceptual de ESB[33]. [49]

Tradicionalmente, el ESB ofrece capacidades de infraestructura y mediación, actuando como punto intermediario en la integración de sistemas distribuidos, especialmente en arquitecturas

[33] Adaptación hecha en [49] de la imagen de Papazoglou en [50].

SOA [50]. En estos escenarios, los sistemas envían mensajes al ESB en lugar de interactuar directamente entre sí. Estos mensajes, son procesados por un flujo de mediación (p. ej. rutear según el contenido y luego transformar el modelo), que resuelve las disparidades entre los sistemas. También el ESB permite centralizar la configuración, supervisar los mensajes (i.e. *monitoring*), así como exponer servicios web y proveer seguridad de acceso.

Papazoglou en [50] añade que el ESB está implementado con tecnologías de *middleware* y que usa adaptadores e interfaces basadas en estándares para implementar la interoperabilidad entre sistemas. Estos pueden potencialmente estar desplegados en distintas plataformas y usar diferentes protocolos para comunicarse.

En la actualidad la computación en la nube ha generado que los ESB añadan soporte para la integración de software desplegado en la nube. Además, existen variantes a los ESBs tradicionales denominadas "ESBs livianos" (*Lightweight ESB*) [51]. Estas reducen las funcionalidades multipropósito de los ESBs como BPM y se basan en JSON y REST, en lugar de XML y SOAP.

2.3.3 Perfiles de Usuario de las PI

El uso de software para la integración de sistemas históricamente requería de conocimiento técnico elevado por parte de los usuarios de PI [52]. En la actualidad, los productos de integración apuntan a un rango más amplio de perfiles de usuario, ofreciendo funcionalidades variadas [46, 53].

El cambio en materia de quienes realizan tareas de integración[34], permite mayor fluidez en las mismas, evitando costos innecesarios y optimizando recursos humanos. Así, ingenieros de sistemas pueden enfocarse en proyectos de integración compleja, delegando a usuarios menos técnicos tareas de integración simple como integrar sus datos con herramientas diarias de negocio, como el correo.

Los perfiles de usuario, varían según el producto y proveedor de software de integración. Aún así, existe consenso en algunos perfiles, como el perfil técnico y el llamado "*Citizen Integrator*" [53, 52]. Este último refiere en particular a usuarios de negocio que requieren efectuar una actividad de integración simple.

Gartner en [46] define los siguientes perfiles:

- *Specialist Integrator*. Refiere al personal con conocimiento técnico elevado (p. ej. ingenieros de sistemas). Estos poseen acceso a todas o la mayoría de las prestaciones de la PI.
- *Ad-Hoc Integrator*. Refiere a especialistas de TI. Estos realizan tareas puntuales de integración o mantenimiento sobre integraciones ya existentes.

[34] Por tareas de integración, se refiere a implementación y configuración de soluciones de integración.

- *Citizen Integrator*. Refiere a usuarios de negocio. Realizan actividades simples de integración de datos, como integrar datos propios con el correo (p. ej. integrar Jira[35] con Gmail[36]).

2.4 Enterprise Integration Patterns

En 2004, Hohpe y Woolf presentaron en [2] una serie de patrones para la integración empresarial basada en mensajería. Cada patrón, resuelve a nivel de diseño (abstrayéndose de la tecnología) un problema de integración determinado. A este conjunto de soluciones se les denomina "patrones de integración empresarial" (*Enterprise Integration Patterns*, EIP) y han tenido una importante repercusión [54], además de que continúan utilizándose a la presente fecha [55, 56, 57].

Los patrones se agrupan fundamentalmente en seis categorías: *construcción de mensajes*, *ruteo de mensajes*, *transformación de mensajes*, *terminales de mensajería*, *canales de mensajería* y *administración de sistemas*. Un resumen de ellos puede verse en la Figura 2.13.

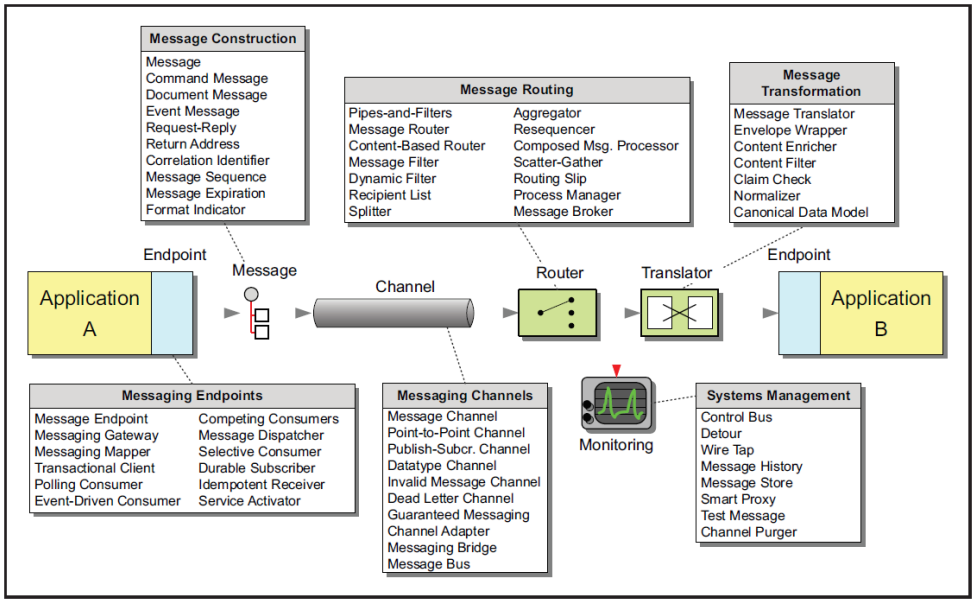

Figura 2.13: Resumen de los patrones de integración empresarial [54]

[35] https://www.atlassian.com/software/jira
[36] https://gmail.com

Las PIs a menudo permiten la construcción de soluciones de integración a partir de los EIPs. Esto se verá en la sección de trabajo relacionado.

Desde el lanzamiento de los EIPs, nuevas tendencias y escenarios han surgido (p. ej. computación en la nube) [58]. Estos llevan a reflexionar sobre la necesidad de inclusión de nuevos patrones, y también de determinar si los patrones originales solucionan problemas de integración que surgen de nuevas tendencias en materia de software. Algunos ejemplos son computación en la nube, macrodatos (i.e. *Big Data*), internet de las cosas (i.e. *Internet of Things*, IoT), microservicios y arquitecturas dirigidas por eventos (*event-driven architectures*, EDA), los cuales aportan complejidades nuevas.

En 2016 Hohpe y Woolf publicaron un artículo a modo de entrevista, reflexionando acerca del presente y futuro de los patrones de integración [54]. Afirmaron que los patrones continúan vigentes, y argumentaron que algunas tendencias (entre las que incluyen explícitamente microservicios, IoT y aplicaciones híbridas) necesitan mensajería basada en colas de mensaje e invocaciones asíncronas entre sus componentes, razón por la cual están sujetas a los problemas ya resueltos por los EIP.

En 2017, Ritter et. al. publicaron un artículo en el cual profundizaron en la problemática del posible desfasaje de los EIPs con nuevas tendencias y cambios de la computación [58]. En dicho trabajo, se concluye que se requieren adicionar nuevos EIPs a los presentados en [2] y se detallaron propuestas al respecto [58]. Aunque se coincide con Hohpe y Woolf en que los EIPs originales mantienen validez.

2.5 Arquitectura y Calidad del Software

En estas sección se describen algunos conceptos relacionados a arquitecturas de software y calidad que son relevantes a la tesis. Concretamente, se describe el modelo "4+1" de representación de arquitecturas, estándares y atributos de calidad, y un trabajo relevante que evalúa una arquitectura de software en base al impacto sobre los atributos de calidad.

2.5.1 Modelo de Vistas 4+1

El modelo de vistas 4+1, es un modelo propuesto por Kruchten en [59] para describir arquitecturas de software a partir de múltiples vistas, que describen diferentes aspectos del sistema.

El modelo, se compone de cuatro vistas principales, más una quinta que presenta casos de uso o escenarios importantes del sistema. No existe una notación definida para el modelo de vistas. En el trabajo original de Kruchten los ejemplos se diagramaron con la notación Booch [59], aunque es posible también utilizar otras notaciones como UML. En la Figura 2.14 se muestra un esquema de las vistas, así como también una correspondencia posible a diagramas

UML para usar como notación.

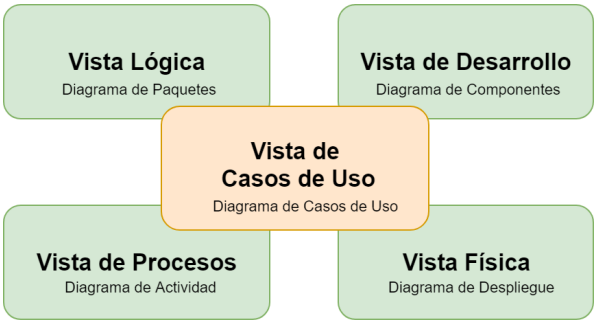

Figura 2.14: Esquema del modelo 4+1 y posibles diagramas UML a utilizar como notación

A continuación se describen las vistas y su propósito en el modelo:

- **Vista lógica**: Se enfoca en los requerimientos funcionales, mostrando una descomposición abstracta en clases o entidades que permitan soportar dichos requerimientos.
- **Vista de desarrollo**: Describe la modularización del sistema en capas, subsistemas y componentes. Detalla la organización estática del sistema.
- **Vista de procesos**: Comprende los aspectos dinámicos del sistema y de tiempo de ejecución. Se enfoca también en los aspectos de concurrencia, procesos y comunicación.
- **Vista física**: Detalla el despliegue del sistema, cómo son hospedados los módulos y la correspondencia entre hardware y software.
- **Vista de casos de uso**: Muestra los casos de uso principales del sistema.

2.5.2 Atributos de Calidad de Software

Existen múltiples definiciones de los atributos de calidad de software y estándares que los utilizan. En este trabajo, se considera el estándar ISO/IEC 25010:2011 que define propiedades de calidad de un producto de software y la definición del *Software Engineering Institute* (SEI)[37] de atributo de calidad:

> "Un atributo de calidad es una propiedad medible o comprobable de un sistema, usada para determinar qué tan bien cumple las necesidades planteadas por las partes interesadas." [38] [60]

[37] https://www.sei.cmu.edu/
[38] Traducido al español por el autor. La original puede encontrarse en [60]

En la Tabla 2.1, se muestran estándares de calidad y los atributos que definen. Estos además especifican subatributos (no mostrados en la Tabla 2.1), que refieren a cualidades diferenciables dentro del propio atributo de calidad.

Tabla 2.1: Estándares de calidad y atributos definidos.

IEEE Std. 1061	ISO/IEC 25010:2011	MITRE Guide to Total Software Quality Control
Eficiencia Funcionalidad Mantenibilidad Portabilidad Confiabilidad Usabilidad	Eficiencia de desempeño Adecuación funcional Mantenibilidad Portabilidad Confiabilidad Usabilidad Compatibilidad Seguridad	Eficiencia Integridad Confiabilidad Supervivencia Usabilidad Correctitud Mantenibilidad Testabilidad Expansibilidad Flexibilidad Interoperabilidad Portabilidad Reusabilidad

En el Apéndice A se detallan los atributos y subatributos especificados en el estándar ISO/IEC 25010:2011 usado en este trabajo.

En el libro SWEBOK (*Guide to the Software Engineering Body of Knowledge*), se listan tres formas de evaluar la calidad del diseño de software: revisiones del diseño (formales e informales), análisis estático (formal o semi formal) y simulación o prototipado. Las dos primeras, refieren a técnicas estáticas (no se ejecuta código) mientras que la última refiere a pruebas directas de ejecución [61]. Se comenta en la Sección 2.5.3, un trabajo que considera una revisión en base al impacto sobre los atributos de calidad.

2.5.3 Impacto de Arquitectura sobre Atributos de Calidad

En [14], el SEI publicó un estudio referente al impacto de aplicar una arquitectura SOA sobre los atributos de calidad[39] de un sistema. El trabajo es relevante, ya que sirvió como motivación y referencia para los estudios de impacto presentados en el Capítulo 3 de esta tesis.

En el artículo del SEI, se cataloga el impacto en base a tres colores: verde, amarillo y rojo. El color verde significa que el atributo posee un nivel de satisfacción razonable, existiendo tecnología y estándares maduros para dar solución a dicho aspecto. El color amarillo significa que existen algunas soluciones para satisfacer dicho atributo, pero requieren mayor madurez

[39] El estudio no aparenta seguir un estándar de calidad claro, comparte algunos atributos con IEEE 1061s.

e investigación. El color rojo, implica que el atributo es problemático en SOA y que requiere un esfuerzo significativo para satisfacer el mismo, debido en parte a la falta de soluciones y estándares maduros.

Notar que el color verde no implica que la implementación del punto no sea compleja, sino que significa que existe tecnologías o enfoques para satisfacer razonablemente el atributo.

Un resumen del impacto discutido en el artículo, se muestra en la Tabla 2.2[40]. Notar que el artículo resulta de interés debido a cierta similitud ya comentada entre SOA y la arquitectura de microservicios.

Tabla 2.2: Impacto de SOA en atributos de calidad [14]

Atributo de Calidad	Impacto
Interoperabilidad	**Verde**
Confiabilidad	**Amarillo**
Disponibilidad	**Amarillo**
Usabilidad	**Amarillo**
Seguridad	**Rojo**
Desempeño	**Rojo**
Escalabilidad	**Amarillo**
Extensibilidad	**Verde**
Adaptabilidad	**Amarillo**
Testabilidad	**Rojo**
Auditabilidad	**Rojo**
Operabilidad y Desplegabilidad	**Amarillo**
Modificabilidad	**Verde**

Si bien SOA posee similitudes con la arquitectura de microservicios, algunos atributos reportados en el artículo difieren entre ambas arquitecturas. Esta diferencia se hace mayor si se consideran aspectos de despliegue, como contenedores y plataformas de orquestación de contenedores sobre clústeres, o también si se consideran estrategias como DevOps.

[40] El análisis completo puede hallarse en [14]. La Tabla 2.2 fue traducida al español por el autor.

2.6 Trabajo Relacionado

En esta sección se presenta trabajo relacionado a la temática principal de la tesis. Se analizan por un lado, trabajos académicos relevantes. Por otro, se estudian productos de PI que posean una arquitectura de microservicios o estén al menos parcialmente basados en los EIP.

2.6.1 Trabajos Académicos

En esta sección se analizan artículos publicados relacionados a PIs que posean arquitectura de microservicios. Los trabajos fueron accedidos a través de la plataforma Timbó[41], a excepción de algunos casos que resultaron de una búsqueda directa en Internet.

Los artículos hallados refieren a casos prácticos de PIs basadas en microservicios, pero donde estas no son de propósito general sino específicas para un escenario de integración. En particular, se observan varias propuestas en el área de "Internet de las Cosas" (i.e. *Internet of Things*, IoT). Tampoco es estudiado en ninguno de los artículos, cuáles escenarios de integración son propicios para usar una PI con arquitectura de microservicios (o si lo son todos).

2.6.1.1 Rediseño de PI Monolítica a Microservicios

En [62] Djogic et. al. muestran un ejemplo de rediseño de una PI basada en SOA hacia una arquitectura de microservicios. La PI rediseñada no es de propósito general, sino que es específica para el área de bienes raíces. El rediseño se hizo primero separando la arquitectura en grupos de microservicios y luego desglosando cada grupo en microservicios concretos. En el artículo, se detalla solamente el desglose de los dos grupos considerados principales, los cuales se muestran en la Figura 2.15.

Como motivo para el rediseño los autores mencionan tres principales: i) la dificultad del mantenimiento, ocasionado por la complejidad de los módulos, ii) el riesgo alto para la creación de nuevas versiones, debido a la complejidad del sistema y a la necesidad de desplegar la plataforma entera, y iii) la necesidad de escalamiento conllevó a un mal uso de los recursos.

Se observa que la arquitectura propuesta corresponde a una PI de dominio específico y no a una PI de propósito general. Por otro lado, se entiende que el nivel de granularidad de los microservicios podría no corresponderse a los parámetros de tamaño vistos en la Sección 2.1, ya que el servicio "procesador de mensajes de datos" aparenta concentrar toda actividad de mediación hecha por la PI.

Por otro lado, se observa que la arquitectura presenta una única base de datos transaccional centralizada mediante el microservicio "*DB Microservice*". Esto aparenta contraponerse a tener una base de datos por servicio y basarse en consistencia eventual (de acuerdo a lo

[41] http://www.timbo.org.uy/

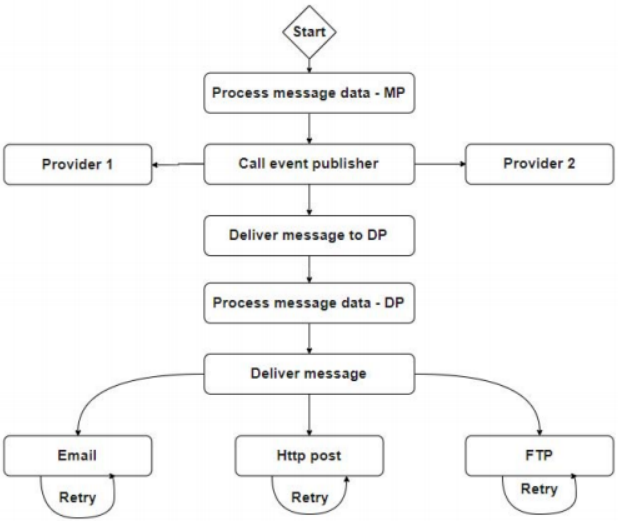

Figura 2.15: Desglose de los grupos de microservicios "núcleo" y "proveedor" [62]

propuesto por Fowler en [9] y Richardson en [18]). El problema que constituye dicha centralización se analiza en detalle en [63].

2.6.1.2 Otros Trabajos

En [64], se propone la arquitectura de un sistema de *Middleware* basado en microservicios para escenarios de IoT (i.e. *Internet of Things*) y *Big Data*. El sistema se plantea como una plataforma para escenarios de fábricas inteligentes[42] que permite integrar dispositivos inteligentes, herramientas de simulación y aplicaciones empresariales. La arquitectura puede verse en la Figura 2.16. Si bien el planteo es interesante, la plataforma presentada no es de propósito general y sus capacidades de mediación son acotadas comparadas con las de plataformas como los ESBs o las iPaaS. Además no analiza si la arquitectura de microservicios es una decisión adecuada de diseño.

En [65] también se propone una arquitectura para PI basada en microservicios para escenarios de IoT. La PI, esta especialmente orientada a proporcionar información respecto a la calidad de datos proveniente de los dispositivos inteligentes. Análogo a [64], la PI no es de propósito general y sus capacidades de mediación son acotadas comparadas con las de plataformas como los ESBs o las iPaaS.

En [66] se presenta una PI muy similar a las anteriores, también para escenarios de IoT. En

[41] Traducido al español por el autor, el original puede encontrarse en [14].
[42] Refiere a fábricas que utilizan dispositivos digitales en el marco de IoT.

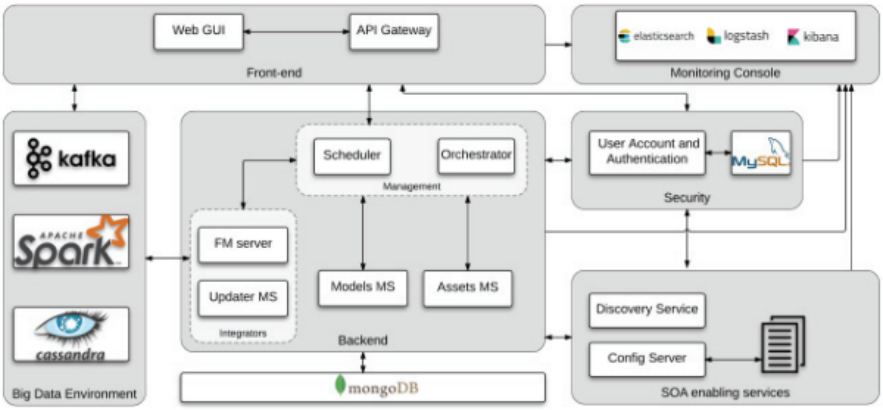

Figura 2.16: Arquitectura de microservicios de la PI en [64]

particular, este trabajo solo plantea como microservicios la interacción con los dispositivos.

Existen también varias propuestas de arquitectura para PI de propósito general, pero que no están basadas en microservicios, como las de los trabajos [1] y [67]. En contraste con las arquitecturas mostradas en estos trabajos, esta tesis (véase Capítulo 4) incorpora, en la arquitectura presentada, nuevos elementos propios de las arquitecturas de microservicios (p. ej. *Circuit Breaker*), así como también propone variantes específicas para ciertos escenarios de integración. Por otro lado, analiza formas de despliegue, no consideradas en dichos trabajos, que incluyen tecnologías recientes como las plataformas de orquestación de contenedores sobre clústeres.

2.6.2 Productos de PI Relacionados

Se describen en esta sección casos de productos de PI que posean una arquitectura de microservicios o que estén basados en los EIP.

En base a la clasificación de PI de Gartner [46] y a la clasificación presentada en [3], el trabajo relacionado se separa en dos categorías principales:

- Plataformas de Integración como Servicio (iPaaS)
- Suites de Integración *On-Premise* (p. ej. ESBs)

Se detalla a continuación un producto representativo por cada categoría, los demás productos estudiados se muestran en el Apéndice C.

2.6.2.1 Plataformas de Integración como Servicio (iPaaS)

En esta sección se describe un ejemplo representativo de un producto del tipo iPaaS.

Elastic.io

Elastic.io[43] es un producto de iPaaS que afirma poseer una arquitectura de microservicios. El producto está basado en el concepto de crear y supervisar flujos de integración. Dicho flujo consta de un conjunto de componentes de integración conectados entre sí mediante una cola FIFO[44] de mensajería persistida (ver Figura 2.17). Cada componente es un microservicio alojado en un contenedor Docker. La plataforma opera bajo Apache Mesos[45] y cada microservicio y contenedor poseen independencia de los demás servicios y sus propios recursos, en coherencia con el paradigma de microservicios [68].

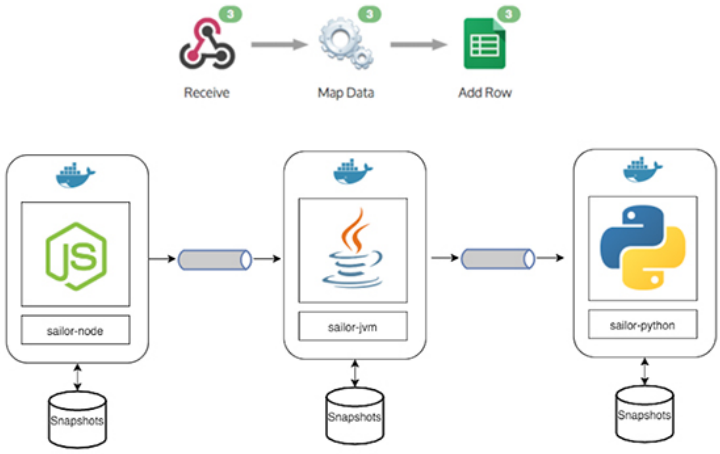

Figura 2.17: Comunicación de ejemplo entre componentes en Elastic.io [69].

Elastic.io brinda una cantidad razonable de componentes de integración predefinidos, los cuales son de código abierto. Además, usuarios técnicos pueden hacer uso de una API para crear componentes personalizados.

El producto permite integraciones híbridas, contando con un conector llamado "*Secure Integration Bridge*" que se despliega en el ambiente interno del cliente. Por tanto, soporta integraciones de sistemas desplegados tanto en la nube como en redes internas.

[43] https://www.elastic.io/
[44] Del inglés "*First in First out*", implica que el primer mensaje retornado es el primero en ingresar a la cola.
[45] http://mesos.apache.org/

En la Figura 2.18, se detalla el registro de una puesta en marcha de un componente de integración. Notar que se descarga y extrae inicialmente un "Slug"[46], el cual es un archivo comprimido con la información del componente. Su ciclo de vida está controlado por la plataforma Elastic.io y tiene una persistencia externa para su estado [71].

Para realizar la supervisión de cada microservicio (i.e. *monitoring*), los registros de ejecución son recogidos desde los canales estándar[47] y almacenados para uso posterior [71]. Además, información de telemetría es recogida mediante APIs de Docker y también almacenada para uso posterior [69].

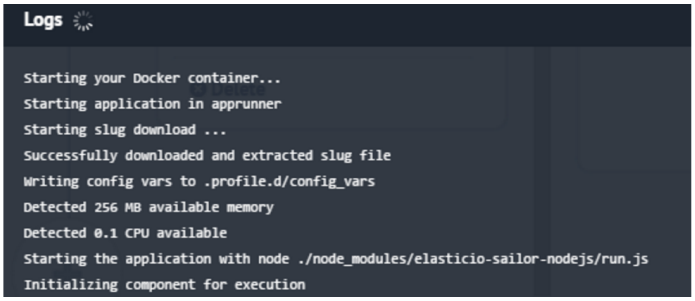

Figura 2.18: Log de inicio de ejecución de un componente de Elastic.io.[48]

Respecto a los EIPs, la plataforma implementa algunos de los más básicos, pero no se refiere a ellos de forma particular en la documentación ni se les da en la plataforma consideración alguna como concepto.

2.6.2.2 Suites de Integración On-Premise

En esta sección se describe un ejemplo representativo de un producto de integración *on-premise*.

WSO2

WSO2[49] posee un producto de integración llamado "*Enterprise Integrator*", el cual combina productos de WSO2 como ESB, *Message Broker*, procesos de negocio, entre otros.

El producto de ESB de WSO2, y por consiguiente también *Enterprise Integrator*, da soporte a los EIP ofreciendo facilidades para implementar los mismos. En la documentación del

[46] Este concepto es tomado de la plataforma Heroku[70].
[47] Refiere a "stdin", "stdout" y "stderr", los flujos de salida estándar. Estos son canales de comunicación predefinidos para los procesos, en algunos sistemas operativos (p. ej. UNIX).
[48] Captura tomada en una prueba de concepto de la plataforma.
[49] https://wso2.com/

producto [72], se explica cómo implementar con WSO2 ESB la mayoría de los sesenta y cinco patrones originales del libro de Hophe y Wolf. A pesar de lo anterior, no tiene en todos los casos un solo conector configurable que implemente todo el patrón de integración. Por ejemplo, el patrón "canal de mensajes fallidos" (i.e. *Dead Letter Channel*) se implementa utilizando tres conectores de WSO2 distintos: "*Store mediator*", "*Message store*" y "*Message processor*".

Si bien WSO2 ofrece herramientas y bibliotecas para la creación de microservicios, ninguno de sus productos de integración aparenta tener una arquitectura de microservicios, de acuerdo a la documentación oficial [72].

2.7 Conclusiones

En este capítulo se presentó un resumen del conocimiento existente considerado relevante para esta tesis.

Por un lado, se observó que no existe una definición consensuada de microservicios, siendo la caracterización de Fowler y Lewis una de las más citadas. El tamaño de los microservicios tampoco ha sido estandarizado, aunque existen varias propuestas entre las que se destaca la de Jon Eaves que afirma que un microservicio debe poder ser reescrito en dos semanas.

Por otro lado, se analizaron los conceptos de coordinación de microservicios, entre los cuales se destaca coreografía y orquestación. Si bien la aplicabilidad de ambas técnicas es motivo de discusión, algunos autores como Richardson consideran que orquestación es apropiado en coordinaciones complejas y extensas, mientras que coreografía es apropiado para flujos simples y con un número de microservicios conciso.

En otro aspecto se analizó que las arquitecturas de microservicios suelen estar acompañadas por el uso de contenedores y plataformas de orquestación de contenedores, como forma de potenciar su escalabilidad y tolerancia a fallas. Además, suele acompañarse por la aplicación de DevOps por parte de los equipos de desarrollo.

En materia de evaluación de arquitecturas de microservicios, se analizaron distintos conjuntos de patrones y buenas prácticas. Se entiende que los patrones propuestos por Richardson constituyen el conjunto más apropiado y completo para evaluar arquitecturas de microservicios. Además se comentaron estándares de calidad y se discutió un trabajo relevante que evalúa una arquitectura de software en base al impacto sobre los atributos de calidad del sistema.

En cuanto a trabajo relacionado, se analizaron productos de la industria y trabajos académicos. Por el lado de productos de la industria se consideraron productos de iPaaS y ESB, entendiéndose que se relacionan más a la caracterización de PI mostrada en el trabajo de González y Ruggia. Se observó que si bien algunos productos afirman estar basados en una arquitectura de microservicios, ninguno describe dicha arquitectura en detalle.

En relación a trabajo académico existente, los artículos hallados refieren a casos prácticos de PIs basadas en microservicios, pero donde estas no son de propósito general sino específicas para un escenario de integración. En particular, se observan varias propuestas en el área de *Internet de las Cosas* (IoT). Por otro lado, se observó que ninguno de los artículos analiza la aplicabilidad de una arquitectura de microservicios para PI, ni considera cuáles escenarios de integración son propicios para su aplicación.

En resumen, las características distintivas de este trabajo son evaluar la aplicabilidad de una arquitectura de microservicios para la construcción de PIs, haciendo foco en los contextos donde es utilizada, y aportar propuestas de arquitectura de microservicios para PIs de propósito general.

3 Aplicabilidad de Microservicios en PI

En este capítulo se analiza la aplicabilidad de una arquitectura de microservicios para la construcción de PIs, tomando como referencia el concepto de PI descrito en la Sección 2.3. Para ello, se plantea determinar en qué contextos o escenarios de integración[1], si los hay, es beneficioso que una PI esté basada en una arquitectura de microservicios. Esto se determina en base al análisis de cómo impacta utilizar dicha arquitectura en PIs, y a la comparación de dicho impacto con los requerimientos y preocupaciones de calidad de los escenarios.

El capítulo se organiza de la siguiente manera. En la Sección 3.1 se describe la metodología utilizada para el análisis. En la Sección 3.2, se describen características de los escenarios de integración y se las agrupa en categorías. En la Sección 3.3 se presenta cómo afectan a la PI las características de los escenarios, y se indican subatributos de calidad preocupantes por cada una de ellas. En la Sección 3.4 se estudia el impacto de aplicar una arquitectura de microservicios en PIs. En la Sección 3.5 se combinan los resultados de las Secciones 3.3 y 3.4 para determinar cómo la arq. de microservicios afecta los subatributos preocupantes de cada característica de escenario. Se obtienen también en dicha sección un conjunto de características de referencia, para determinar si un escenario se beneficia al utilizar una PI basada en microservicios. Por último, en la Sección 3.7 se presentan las conclusiones del capítulo.

3.1 Metodología de Análisis

Para determinar en qué escenarios de integración es propicio utilizar una arquitectura de microservicios en la PI, se estudia cómo los escenarios y la arquitectura afectan a dichas plataformas. Se toma como punto de medida, los atributos de calidad de la PI, en forma si-

[1] Por "escenario de integración" se considera al contexto conformado por los sistemas, las partes interesadas en la integración (i.e. *stakeholders*), la infraestructura y los datos a integrar.

milar al enfoque aplicado en [14] por el SEI[2] (ver Sección 2.5.3). Los atributos y subatributos considerados, son los pertenecientes al estándar ISO/IEC 25010:2011 (ver Apéndice A).

Los principales conceptos referentes al análisis hecho en este capítulo se muestran en la Figura 3.1.

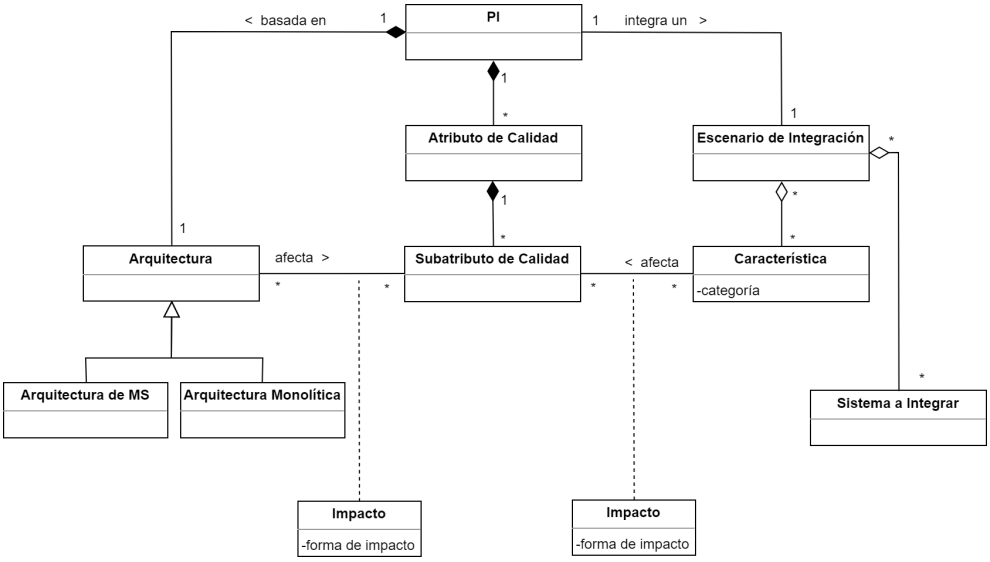

Figura 3.1: Entidades relevantes para el análisis de aplicabilidad.

Se considera "escenario de integración" al contexto conformado por los sistemas, las partes interesadas en la integración (i.e. *stakeholders*), la infraestructura y los datos a integrar. Los escenarios pueden desglosarse en características que lo definen, por ejemplo, la forma de despliegue de la PI.

Por otro lado, una PI está basada en una arquitectura, siendo la arquitectura de microservicios un tipo particular entre otros posibles (p. ej. arquitectura monolítica).

Además, tanto la arquitectura de la PI, como las características del escenario de integración (p. ej. que la PI esté desplegada en la nube), pueden afectar la calidad de la PI. Concretamente, afectan los subatributos de calidad de la misma, y por ende también los atributos de calidad. Específicamente, los subatributos pueden:

- Ser afectados positivamente
- No ser afectados
- Ser una preocupación

[2] *Software Engineering Institute (https://www.sei.cmu.edu/).*

Se considera que un subatributo de calidad constituye una "preocupación" cuando es afectado negativamente, o cuando se requieren consideraciones especiales para satisfacer el subatributo[3].

Por ejemplo, si la PI está desplegada en servidores internos, entonces el subatributo *Disponibilidad* es una preocupación, ya que incendios, cortes de luz u otros siniestros pueden afectar la disponibilidad de la PI. Esto no necesariamente implica que la PI se verá afectada, ya que se podrían desplegar réplicas en lugares geográficos diferentes; sino que implica que consideraciones específicas son requeridas para satisfacer el subatributo. Es decir, si se toman consideraciones la preocupación puede no convertirse en un problema de calidad.

En base a los conceptos definidos, el análisis de este capítulo estudia para cada característica de escenario considerada, cómo una arquitectura de microservicios afecta a los subatributos preocupantes de la PI. Esto permite conocer qué características[4] son beneficiadas por una PI basada en microservicios. Finalmente, se puede obtener un conjunto de características que permitan determinar si un escenario se beneficia al utilizar una PI basada en microservicios.

La metodología sigue el proceso mostrado en la Figura 3.2. El proceso y su orden se describe en forma resumida a continuación:

1. *Caracterización de escenarios*: Se describen características posibles de los escenarios de integración, las cuales se agrupan en categorías. Por ejemplo *PI Desplegada Internamente* es una característica de la categoría *Ubicación de la PI*. Se muestra en la Sección 3.2.

2. *Impacto de características sobre PI*: Se determina como afectan los escenarios a las PIs. Concretamente, se analiza como impactan las características a la PI en base a los atributos de calidad. Se muestra en la Sección 3.3.

3. *Preocupaciones por características*: En base al punto 2, se obtienen los subatributos preocupantes por cada característica. Es decir, las preocupaciones de calidad en la PI según el escenario. Se muestra en la Sección 3.3.

4. *Impacto de arq. de microservicios sobre PI*: Se estudia el impacto de una arquitectura de microservicios sobre los atributos de calidad de la PI. Se muestra en la Sección 3.4.

5. *Impacto de arq. de microservicios sobre preocupaciones*: A partir del análisis del punto 3 y 4, se determina si la arquitectura de microservicios beneficia o no a las preocupaciones de calidad de los escenarios. Se muestra en la Sección 3.5.

6. *Idoneidad de escenarios para PIs basadas en microservicios*: En base al punto 5, se observa si el impacto de una arq. de microservicios es positivo o no sobre las preocupaciones de calidad y se obtienen dos conjuntos de características (se muestra en Secciones 3.5.2 y 3.5.3):

[3] Si se considerase el trabajo del SEI [14], los atributos preocupantes de esta tesis serían etiquetados en color amarillo y rojo (ver Sección 2.5.3).

[4] De aquí en mas "característica" y "característica de escenarios de integración" se usan en forma intercambiable.

a) Características de escenarios propicias para una PI basada en microservicios.

b) Características de escenarios no ideales para una PI basada en microservicios.

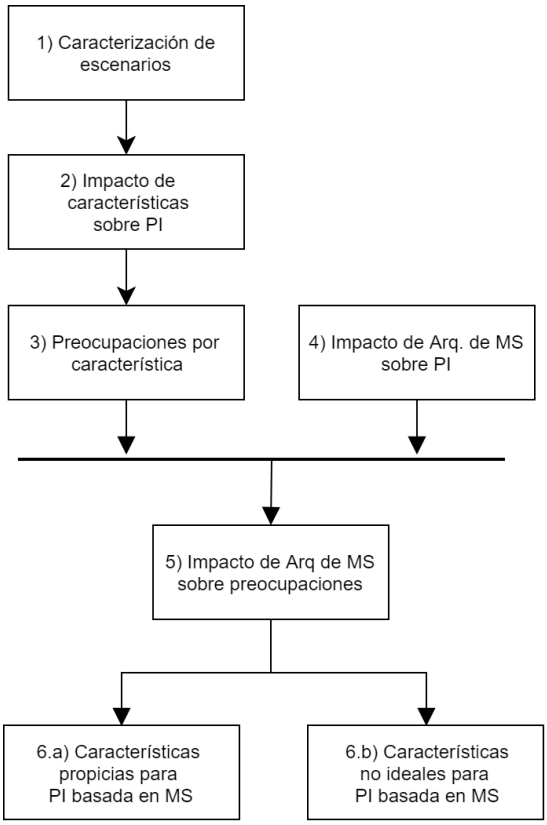

Figura 3.2: Proceso ejecutado para el análisis de aplicabilidad.

3.2 Caracterización de Escenarios de Integración

En esta sección se identifican características de escenarios de integración, con el objetivo de determinar el alcance del análisis[5]. En particular se consideran características que se entendió que afectan la aplicabilidad de microservicios en la PI, lo cual se estudia en secciones posteriores.

[5] Inicialmente se consideró analizar diversos escenarios concretos. Luego se observó que el análisis no se acotaba apropiadamente y que la aplicabilidad era de hecho afectada por características puntuales de los escenarios, a lo cual se decidió analizarlas separadamente.

Las posibles características de los escenarios se agrupan en categorías para mejor visualización. Concretamente, se presentan cinco categorías:

1. Ubicación de la PI
2. Cantidad de organizaciones
3. Ubicación de los sistemas a integrar
4. Particularidades de los sistemas a integrar
5. Particularidades de los datos a integrar

3.2.1 Ubicación de la PI

En relación a la ubicación de despliegue de la PI, se consideran las siguientes características posibles para un escenario de integración:

- Despliegue en una nube pública
- Despliegue interno
  - Despliegue en una nube privada
  - Despliegue en servidores internos
- Despliegue híbrido

El despliegue en una nube pública, significa que la PI es desplegada en una infraestructura de nube de tipo PaaS ofrecida por un proveedor público. En particular, se considera una plataforma de orquestación de contenedores sobre clústeres (ver Sección 2.2.2). Se considera además que la infraestructura posee redundancia de *hardware*.

El despliegue en una nube privada, significa que la PI es desplegada en una plataforma análoga a la de nube pública, pero en un ambiente interno o dedicado, donde la propia empresa se encarga de administrar la plataforma y sus datos, en vez de un proveedor externo. En este trabajo se considerará que la nube privada es complementada con redundancia de *hardware*.

El despliegue en servidores internos, significa que la PI es desplegada en servidores web ordinarios (p. ej. JBoss Wildfly[6]), y no en una plataforma de orquestación de contenedores sobre clústeres. Se considera además que el despliegue es hecho sobre *hardware* de la propia empresa, con servidores en la red interna de la misma.

El despliegue híbrido, asume que la PI es un sistema distribuido, donde una parte del sistema se despliega de forma interna y otra parte en una nube pública. La parte desplegada

[6] www.wildfly.org

internamente puede tener cualquiera de los despliegues internos mencionados.

3.2.2 Cantidad de Organizaciones

En relación a la cantidad de organizaciones, se consideran las siguientes características posibles para un escenario de integración:

- Escenarios intraorganizacionales
- Escenarios interorganizacionales

Este aspecto refiere a si la PI es usada para integrar sistemas de una única empresa, o de un conjunto de empresas. Los casos de integraciones interorganizacionales, son más complejos y tienen requerimientos particulares sobre la PI, como se verá en la Sección 3.3.

3.2.3 Ubicación de los Sistemas a Integrar

En relación a la ubicación de los sistemas a integrar, en contraposición con la ubicación de la PI, se consideran las siguientes características posibles para un escenario de integración, las cuales se dividen en dos según si hay una o más empresas involucradas:

- Escenarios intraorganizacionales
  - Sistemas a integrar y PI ubicados en la misma red
  - Sistemas a integrar y PI ubicados en distinta red
  - Sistemas a integrar mezclados en la nube y en red interna
- Escenarios interorganizacionales
  - Sistemas a integrar en redes internas
  - Sistemas a integrar en la nube y en redes internas
  - Sistemas a integrar en la nube

Notar que en algunos escenarios, no se especifica la ubicación de la PI. En dichos casos la ubicación se discute en el análisis.

3.2.4 Particularidades de los Sistemas a Integrar

En relación a particularidades de los sistemas a integrar, se consideran las siguientes características posibles para un escenario de integración:

- Madurez de los sistemas a integrar
- Cantidad de sistemas a integrar

La primer característica considera la madurez o estabilidad de los sistemas a integrar. Esto se considera como una conjunción entre la cantidad de defectos remanentes, la tendencia a fallas y la frecuencia en que se realizan cambios en el sistema. Se considera debido a que los sistemas inmaduros o volátiles tienen necesidades de integración diferentes que los sistemas poco cambiantes o maduros.

La segunda característica considera la cantidad de sistemas a integrar. En particular, se estudia el caso donde el número de sistemas a integrar es elevado. A efectos prácticos, se propone a criterio de autor más de cinco sistemas a integrar como número alto.

3.2.5 Particularidades de los Datos a Integrar

En relación a particularidades de los datos a integrar, se consideran las siguientes características posibles para un escenario de integración:

- Masividad de datos o de accesos de usuario
- Complejidad de los datos

La primer característica, considera si la PI debe soportar una alta cantidad de datos a integrar, o si un elevado número de usuarios interactúan con la PI de forma concurrente. Se consideran en forma conjunta debido a su similar impacto sobre los atributos de calidad de la PI.

La segunda característica, considera la complejidad de los datos. Esto refiere a si los datos requieren de un procesamiento complejo o computacionalmente caro, en el marco de la integración. Un ejemplo, es un dato codificado cuya transformación de decodificado debe hacerse en la PI. Notar que el procesamiento de los datos, puede también necesitar tecnología o bibliotecas específicas integradas en la PI.

3.3 Impacto de Características de Escenarios sobre PIs

En esta sección se presentan los resultados del análisis que muestra cómo las características afectan los atributos de calidad de la PI. Dicho análisis se muestra en forma detallada en el Apéndice D. El objetivo es obtener qué subatributos de la PI constituyen una preocupación para las características definidas en la Sección 3.2.

En la Tabla 3.1 se muestra una síntesis del análisis. La tabla se subdivide en características agrupadas por categoría. Para cada característica se describe cómo afecta los subatributos de calidad de la PI. Por otro lado, los subatributos que no son afectados por ninguna caracterís-

tica (p. ej. *Estética de Interfaz*) se omiten de la tabla.

Los subatributos de calidad son afectados de tres formas posibles por las características. En particular pueden: i) ser afectados positivamente, ii) no ser afectados, y iii) constituir una preocupación.

A modo de ejemplo en escenarios interorganizacionales la *Modificabilidad* de la PI constituye una preocupación. Esto se debe al menos a dos factores: i) hay varias empresas involucradas, por lo que se requiere comunicación y aprobaciones para lograr modificaciones consensuadas, y ii) integrar sistemas de diferentes organizaciones puede implicar complejidad adicional y que los desarrolladores no entiendan completamente los sistemas de otras empresas para integrarlos.

Otro ejemplo, es que la *Disponibilidad* es afectada positivamente cuando la PI es desplegada en una nube pública. Esto se debe a que usualmente los proveedores de *PaaS* ofrecen un servicio de alta disponibilidad en los sistemas desplegados, lo cual beneficia la PI. Por otro lado, la *Disponibilidad* es una preocupación en despliegues como nube privada y servidores internos, ya que la PI está en la propia empresa y es susceptible a siniestros, a menos que se tengan consideraciones especiales.

En la Tabla 3.1, cabe aclarar que se muestra solo el impacto del escenario interorganizacional, debido a que se entiende que el intraorganizacional no tiene implicancias destacables sobre los subatributos, especialmente en referencia a la conformación de preocupaciones.

Por otro lado, no figuran características de la categoría *Ubicación de los Sistemas a Integrar*. Esto se debe a que durante el análisis se determinó que la ubicación de los sistemas a integrar no incide directamente en la aplicabilidad de una arquitectura de microservicios para la PI. Su impacto esta dado principalmente sobre la latencia y la instalabilidad, como se profundiza en el Apéndice D.

En otro aspecto, se omite de la tabla el despliegue híbrido por dos razones. La primera es que su impacto está mayormente condicionado por cómo son distribuidos los componentes de la PI (i.e. cuáles componentes son desplegados internamente y cuáles en la nube). La segunda es que si se definiese lo anterior, las partes desplegadas interna y públicamente pueden estudiarse en forma separada en base a los despliegues ya analizados. Se profundiza dicho análisis en el Apéndice D, el despliegue híbrido también es considerado y comentado en la Sección 4.2.5.

Tabla 3.1: Impacto de características de escenarios sobre la PI

Atributos de Calidad de la PI		**Características de Escenarios**							
		Ubicación de PI			**Particularidades de Sistemas a Integrar**		**Particularidades de Datos a Integrar**		**Otros**
Atributo	**Subatributo**	**Nube pública**	**Nube privada**	**Servidor interno**	**Alta cantidad de sistemas**	**Volatilidad o Inmadurez**	**Masividad de datos/accesos**	**Complejidad de datos**	**Escenario Interorg.**
Eficiencia de Desempeño	Comportamiento del Tiempo	—	—	—	P	—	P	P	—
	Uso de Recursos	A+	A+	P	P	—	P	P	—
	Capacidad	A+	P	P	—	—	P	P	—
Mantenibilidad	Modularidad	—	—	—	P	P	—	—	P
	Reusabilidad	—	—	—	—	—	—	—	—
	Evaluabilidad	—	—	—	P	P	—	—	P
	Modificabilidad	—	—	—	P	P	—	—	P
	Testabilidad	—	—	—	—	—	—	—	P
Portabilidad	Adaptabilidad	A+	A+	P	—	—	P	—	—
	Instalabilidad	—	—	—	P	—	—	—	P
	Reemplazabilidad	—	—	—	P	—	—	P	P
Confiabilidad	Madurez	—	—	—	—	P	—	—	P
	Disponibilidad	A+	P	P	—	—	P	—	—
	Tolerancia a Fallas	A+	A+	P	P	P	P	P	—
	Recuperabilidad	A+	A+	P	P	P	P	P	—
Seguridad	Confidencialidad	P	A+	A+	—	—	—	—	P
	Integridad	P	A+	A+	—	—	—	—	P
	No repudio	P	A+	A+	—	—	—	—	P
	Trazabilidad	P	A+	A+	—	—	—	—	P
	Autenticidad	P	A+	A+	—	—	—	—	P

A+: Afecta positivamente. P: Implica una preocupación —: No afectado

3.4 Impacto de Arquitecturas de Microservicios en PIs

En esta sección se analiza el impacto teórico de aplicar una arquitectura de microservicios a una PI. El análisis se presenta en mayor profundidad y nivel de detalle, ya que se lo considera importante para los objetivos de la tesis.

Para el análisis, se usan los atributos de calidad de la PI para determinar el impacto y se usa como referencia de comparación una arquitectura monolítica. Por ejemplo, si se afirma que el subatributo *Mantenibilidad* es afectado positivamente, será en comparación con una PI análoga de arquitectura monolítica.

La forma de afectar los subatributos se clasifica en tres formas. En particular un subatributo puede:

- No ser afectado
- Ser afectado positivamente
- Ser una preocupación

Además en algunos casos puede existir una condicionante, indicando que el subatributo es afectado solo si cierta condición se cumple, la cual se especifica en el análisis.

La Tabla 3.2, muestra en forma resumida los resultados del análisis, contrastándolos con los resultados obtenidos en la Sección 3.3. De esta forma se visualiza cómo la arquitectura de microservicios afecta la PI, y se contrapone dicho impacto con las preocupaciones de calidad definidas para las características de escenarios.

Tabla 3.2: Impacto de características de escenarios en contraste con microservicios

Atributos de Calidad de la PI		**Características de Escenarios**								
		Ubicación de PI			**Particularidades de Sistemas a Integrar**		**Particularidades de Datos a Integrar**		**Otros**	
Atributo	**Subatributo**	**Nube pública**	**Nube privada**	**Servidor interno**	**Alta cantidad de sistemas**	**Volatilidad o Inmadurez**	**Masividad de datos/accesos**	**Complejidad de datos**	**Escenario Interorg.**	**Arq. de Microservicios**
Eficiencia de Desempeño	Comportamiento del Tiempo	—	—	—	P	—	P	P	—	—
	Uso de Recursos	A+	A+	P	P	—	P	P	—	A+(*)
	Capacidad	A+	P	P	—	—	P	P	—	A+(*)
Mantenibilidad	Modularidad	—	—	—	P	P	—	—	P	A+
	Reusabilidad	—	—	—	—	—	—	—	—	A+
	Evaluabilidad	—	—	—	P	P	—	—	P	P
	Modificabilidad	—	—	—	P	P	—	—	P	A+
	Testabilidad	—	—	—	—	—	—	—	P	P
Portabilidad	Adaptabilidad	A+	A+	P	—	—	P	—	—	A+
	Instalabilidad	—	—	—	P	—	—	—	P	—
	Reemplazabilidad	—	—	—	P	—	—	P	P	A+
Confiabilidad	Madurez	—	—	—	—	P	—	—	P	—
	Disponibilidad	A+	P	P	—	—	P	—	—	A+
	Tolerancia a Fallas	A+	A+	P	P	P	P	P	—	A+
	Recuperabilidad	A+	A+	P	P	P	P	P	—	A+
Seguridad	Confidencialidad	P	A+	A+	—	—	—	—	P	P
	Integridad	P	A+	A+	—	—	—	—	P	P
	No repudio	P	A+	A+	—	—	—	—	P	P
	Trazabilidad	P	A+	A+	—	—	—	—	P	P
	Autenticidad	P	A+	A+	—	—	—	—	P	P

A+: Afecta positivamente. P: Implica una preocupación —: No afectado (*): Solo es afectado si se cumplen ciertas condiciones

Se describe a continuación por cada subatributo, el impacto de aplicar una arquitectura de microservicios a la PI:

- Eficiencia de Desempeño
  - *Comportamiento del tiempo*: En este trabajo no se ejecutaron pruebas comparativas de desempeño. Por tanto, no se posee información concluyente para determinar el impacto sobre el subatributo. Se observa que en teoría podría ser una preocupación [23], debido a las implicancias de los sistemas distribuidos descritas en la Sección 2.1.4, ocasionadas por el aumento de las comunicaciones de red.

    En [11] se presentan pruebas de rendimiento donde se compara una aplicación web desarrollada con un enfoque monolítico, contra una aplicación análoga con arquitectura de microservicios. Los tiempos de respuesta fueron levemente mayores en el caso de microservicios. Aun así, se observan de dicho estudio tres aspectos a considerar: i) la diferencia entre los tiempos es relativamente pequeña[7], ii) solo dos servicios fueron evaluados, y iii) no se consideraron aspectos de escalamiento en las pruebas. Por consiguiente no puede afirmarse, a priori, que el atributo constituya una preocupación en la calidad.

  - *Uso de recursos*: En escenarios donde se requiere escalamiento es afectado positivamente, debido a que se escala a demanda solo los microservicios requeridos, y no necesariamente a toda la PI. Por consiguiente, se hace un uso de recursos más eficiente que en arquitecturas monolíticas.

    El impacto positivo sobre el subatributo es mayor si el uso de microservicios se acompaña con contenedores (reduciendo la redundancia de máquinas virtuales) y si se acompaña de plataformas de orquestación de contenedores sobre clústeres (que proporcionan escalamiento elástico).

    Si el escenario no requiere escalamiento, la arquitectura de microservicios podría utilizar más recursos que la monolítica, debido a que podría poseer la redundancia de desplegar las dependencias compartidas y un servidor web para cada microservicio. Esta redundancia se compensa al escalar, debido a que la arquitectura monolítica debe escalar todos los componentes del sistema.

  - *Capacidad*: Está sujeta a las mismas condicionantes que el subatributo *Uso de Recursos*. El motivo es que si el uso de recursos se ve beneficiado, es más probable que la capacidad cumpla los requerimientos, ya que se utilizan de forma óptima los recursos existentes.

    El impacto positivo sobre el subatributo es mayor si la PI se despliega en una nube pública, ya que los proveedores de plataforma de nube suelen brindar modelos de servicio donde se puede usar recursos a demanda sin una limitante fija.

[7] Para los dos servicios evaluados, las pruebas arrojaron diferencias de 393ms en un servicio y 75ms en otro.

- Mantenibilidad
  - *Modularidad*: Es afectada positivamente, por las características de microservicios explicadas en la Sección 2.1. Al estar dividido el sistema en servicios pequeños y bien definidos, la modularidad de la PI mejora frente a arquitecturas monolíticas.
  - *Reusabilidad*: Es afectada positivamente, por las características de microservicios explicadas en la Sección 2.1. Los microservicios son independientes en su funcionamiento y por consiguiente reutilizables.
  - *Evaluabilidad*: Constituye una preocupación. Si la cantidad de microservicios es alta, puede ser complejo determinar el impacto de una modificación, debido a la dificultad para determinar qué microservicios son afectados por el cambio.
  - *Modificabilidad*: Es afectada positivamente, por las características de microservicios explicadas en la Sección 2.1. Los microservicios tienen tamaño pequeño, funcionamiento bien definido y brindan modularidad al sistema. Por consiguiente, las modificaciones son acotadas y menos complejas, en comparación con la arquitectura monolítica.

    El impacto positivo sobre el subatributo es mayor si un equipo o persona es responsable del ciclo de vida del microservicio, tal como propone Fowler en [9]. En estos casos quien mantiene el microservicio ya conoce su funcionamiento, el cual es autocontenido. Por tanto, no requiere necesariamente de conocimiento global del sistema para aplicar la modificación.
  - *Testabilidad*: Constituye una preocupación. Si bien se simplifican las pruebas unitarias por la modularidad (y usualmente por la automatización), problemas relacionados a los sistemas distribuidos pueden manifestarse si existen muchos microservicios, donde las pruebas integrales son más complejas por haber varias partes involucradas. Además, la distribución dificulta identificar dónde se originan errores no triviales.

    Aun así, algunos autores como Fowler discuten que el subatributo sea realmente una preocupación [23], alegando que existen enfoques para satisfacer el punto.
- Portabilidad
  - *Adaptabilidad*: Es afectada positivamente, ya que en el estándar de calidad considerado (ISO/IEC 25010:2011) el subatributo incluye la capacidad de escalamiento, la cual es beneficiada por la posibilidad de escalar individualmente los microservicios.

    El impacto positivo sobre el subatributo es mayor si es utilizado en conjunto con contenedores, ya que estos aíslan las problemáticas de dependencias y temas de *hardware* y sistema operativo, mejorando la adaptabilidad. También es mayor si la PI se despliega en una plataforma de orquestación de contenedores sobre

clústeres, donde el escalamiento elástico es manejado de forma automatizada.

- *Instalabilidad*: Si bien la separación de un sistema en varias partes agrega complejidad operacional, existen enfoques y herramientas para satisfacer el subatributo, por lo cual no se considera que sea una preocupación. Sin embargo tampoco puede afirmarse que la instalabilidad sea afectada positivamente por la arquitectura de microservicios.

  Por un lado, *DevOps* aporta un enfoque para contemplar este problema, uniendo las responsabilidades de desarrollo y operaciones sobre las mismas personas, y apoyándose en herramientas de entrega continua.

  Un ejemplo de herramienta es *S2I* (i.e. *Source-to-image*) de la plataforma *OpenShift*, que permite a partir de un repositorio, descargar automáticamente un servicio y sus dependencias, generar la "imagen *Docker*"[8] y desplegarla en la plataforma. Este tipo de automatizaciones quitan complejidad a la instalación.

  Por otro lado las plataformas de orquestación de contenedores también mejoran la instalabilidad, al permitir automatizar despliegues y evitar la gestión y configuración de componentes utilitarios, como los balanceadores de carga, descubrimiento de servicios, entre otros.

- *Reemplazabilidad*: Es afectada positivamente. Como se muestra en la Sección 2.1, los microservicios son fáciles de sustituir y permiten reemplazar partes de la PI sin tener que volver a desplegar el sistema entero. Por tanto se afecta positivamente el subatributo.

- Confiabilidad

  - *Tolerancia a Fallas*: Se entiende que es afectada positivamente. Se observa que los sistemas distribuidos agregan problemas de arrastre de fallas y más comunicaciones de red que pueden fallar. Sin embargo, existe una amplia cobertura sobre el tema [17, 18, 26] y existen soluciones varias que permiten satisfacer el subatributo.

    Por un lado el patrón *Circuit Breaker* (ver Sección 2.1.5) contribuye evitando que clientes invoquen un servicio cuando no está disponible o se encuentra en estado de error. El patrón usualmente se complementa con una aplicación general de políticas de expiración de pedidos (i.e. *timeouts*) y de reintentos, que también contribuyen a la tolerancia a fallas. Por otro lado, el patrón *bulkhead* (ver Sección 2.1.5) también brinda tolerancia a fallas, aislándolas para que no afecten el sistema entero.

    En comparación con la arquitectura monolítica, la arquitectura de microservicios no posee un único punto de falla, lo cual permite controlar mejor las fallas en

[8] Refiere a un contenedor ya empaquetado y desplegable, en el marco de *Docker* (https://www.docker.com/).

conjunto con los patrones mencionados. Considerar el siguiente ejemplo: un sistema S1 envía datos a un sistema S2, los cuales deben ser transformados por la PI. Si la transformación es pesada (p. ej. codificar un video) podría consumir recursos hasta provocar la caída del sistema entero. Con una arquitectura de microservicios y el uso del patrón *bulkhead*, el problema se aísla al componente de transformación y falla de forma controlada. También puede recuperarse reiniciando el contenedor.

En el aspecto tecnológico tanto *Netflix Hystrix*[9] como *Polly*[10] brindan una biblioteca específicamente diseñada para tolerancia a fallas y latencia en sistemas distribuidos (ver Sección 2.1.5).

En otro aspecto, el impacto positivo sobre el subatributo es mayor si la PI es desplegada en una nube pública, ya que se posee usualmente redundancia de *hardware* que permite tolerar fallas de *hardware*. Esto también puede emularse en un despliegue interno, si se despliega sobre infraestructura con redundancia de *hardware*.

Por último cabe agregar que debido a las características de las PIs, las cuales realizan mediación en tiempo real, existe poca necesidad de persistencia (usualmente solo para temas de gestión) por lo cual problemas por fallas en transacciones distribuidas y compensaciones no forman parte de los requerimientos habituales de tolerancia en las PIs.

– *Recuperabilidad*: En forma análoga al subatributo *Tolerancia a Fallas*, es afectada positivamente, debido a que existen varios enfoques y tecnología que permiten satisfacer el subatributo.

  En particular, los patrones y tecnología explicitados para la tolerancia a fallas, también aportan a la recuperabilidad de la PI. Por ejemplo, el patrón *Circuit Breaker* cada un cierto tiempo llamado "ventana" vuelve a enviar pedidos al servicio, para comprobar si este está operativo nuevamente. El objetivo, es recuperar la operatividad normal del sistema en vez de simplemente fallar rápido para evitar expiraciones de pedidos. Así mismo, el simple uso de tiempos de expiración (i.e. *timeouts*) y reintentos pueden entenderse como aportes al subatributo, ya que se brinda recuperabilidad frente a una falla de latencia o de comunicación.

  Por otro lado, el patrón *health check* es usado a menudo para detectar si un servicio está disponible y no en estado de error [20]. De esta forma, frente a una falla, se puede determinar si la falla fue puntual y el servicio es capaz de responder pedidos o si se debe abortar la operación. El patrón de *health check* es usado también para supervisar que todos los servicios estén disponibles.

  Por ejemplo, las plataformas de orquestación de contenedores, utilizan el patrón

[9] https://github.com/Netflix/Hystrix/wiki
[10] https://github.com/App-vNext/Polly

health check para determinar si el servicio está operativo y, en caso contrario, reiniciar el contenedor del mismo. Reiniciar el contenedor puede hacerse debido a dos características de los microservicios: i) usualmente no poseen estado (i.e. *stateless*) y ii) el tiempo de inicio es rápido, ya que el microservicio es acotado (y la virtualización de los contenedores es liviana).

– *Disponibilidad*: Por las razones vistas en los subatributos de *Tolerancia a Fallas* y *Recuperabilidad*, la disponibilidad es afectada positivamente.

Además, el patrón de *health check* puede usarse también como método para lograr mantenimiento con alta disponibilidad, ya que durante un despliegue permite determinar cuando la nueva instancia está disponible, antes de remover la versión anterior, lo cual aporta a no tener tiempos de baja del sistema. Esto es usado por algunas plataformas de orquestación de contenedores, como *OpenShift*.

El impacto positivo sobre el subatributo es mayor si la PI es desplegada en una infraestructura con redundancia de *hardware*, lo cual evita que fallas de *hardware* afecten la disponibilidad.

- *Seguridad*:

– *Todos los subatributos*: Implican un preocupación, tal como se detalla en [12], [73] y [74].

Existen múltiples preocupaciones de seguridad en la arquitectura de microservicios; por un lado, los sistemas distribuidos tienen un mayor número de partes expuestas, lo cual implica más servicios y comunicaciones a proteger. Por otro lado, poseer una mayor cantidad de servicios distribuidos dificulta la supervisión y detección de problemas de seguridad [74]. En otra línea, los microservicios están usualmente diseñados para confiar entre sí, a lo cual un microservicio afectado puede comprometer la seguridad del sistema entero[11] [74]. Además, cada microservicio posee usualmente su propia base de datos a proteger, cuyas credenciales deben gestionarse de forma segura. En otro aspecto la tecnología de contenedores es aún cambiante y propensa a defectos de seguridad, lo cual puede afectar el sistema. Por último, según Newman los equipos de desarrollo son mas propensos a introducir defectos de seguridad en arquitecturas de microservicios que en otro tipo de sistemas [75], debido a que suelen considerar menos el enfoque global del sistema.

Si bien la arquitectura plantea numerosas preocupaciones de seguridad, se afirma en varios artículos que existe poca cobertura en el tema [8] [73]. En el análisis de la literatura relacionada a microservicios hecho en [8], la seguridad se posiciona en el penúltimo lugar como tema con menos propuestas académicas, luego de la supervisión. Esto se refleja también en los patrones de Richardson [20], donde se propone solamente un patrón de seguridad.

[11] Ataque conocido coloquialmente como "problema del comisario confundido" (*confused deputy problem*) [73].

Durante la investigación se observaron algunos mecanismos de mitigación. Se observa que usualmente se utiliza un enfoque de protección de las fronteras del sistema. Esto es, se exponen y protegen solamente los microservicios que deben interactuar con los clientes, como forma de evitar problemas de rendimiento y latencias. A menudo se utiliza un intermediario (comúnmente llamados *Edge Service*[12]) que concentra los pedidos externos. Los componentes expuestos, transmiten hacia los microservicios internos un *token*[13] conteniendo información de autenticación y autorización. Otros enfoques destacables son MTLS para autenticación mutua de servicios y el uso de certificados para identificar servicios confiables [73], el cual es utilizado por Netflix.

Las plataformas de orquestación de contenedores usualmente proveen ciertas funcionalidades de seguridad. Por un lado, permiten aplicar seguridad en capa de transporte y proveen el uso de "secretos" para administrar credenciales, como las referentes a bases de datos (p. ej. *Kubernetes Secrets*[14]). Cuando dichas plataformas son provistas en modo PaaS también aplican automáticamente correcciones y actualizaciones de seguridad sobre la tecnología operante. Sin embargo se debe tener consideraciones especiales, ya que si el sistema es desplegado con un proveedor externo, este podría atacar la red de los microservicios internos y comprometer el sistema entero.

Se concluye que existen desafíos de seguridad en microservicios y que es un tema aún poco cubierto. También se comparte lo expresado en [73], donde se afirma que la seguridad en microservicios depende fuertemente de las tecnologías y ambiente en las que se apoya el sistema.

3.5 Impacto de Arq. de Microservicios sobre Preocupaciones de Calidad

En esta sección, se analizan resultados de secciones anteriores para determinar cómo una PI basada en microservicios afecta a los subatributos preocupantes de cada característica de escenario. El análisis es una comparación basada en la Tabla 3.2, la cual resume los resultados de la Secciones 3.3 y 3.4.

3.5.1 Análisis Comparativo de Resultados

En la Tabla 3.3 se presenta un resumen de los resultados del análisis. Se determina por cada característica la aplicabilidad de una PI basada en microservicios. Por ejemplo, se determina

[12] *API Gateway* es un ejemplo de *Edge Server* que posee además otras funcionalidades particulares.
[13] Richardson sugiere el uso de *JSON Web Token* en [20].
[14] https://kubernetes.io/docs/concepts/configuration/secret/

que la ubicación de los sistemas a integrar no afecta la aplicabilidad, ya que una PI basada en microservicios tiene impacto neutro para cualquier característica de dicha categoría, por tanto es indiferente el uso de dicha arquitectura para construir la PI. En otro aspecto, es destacable que algunas de las características de la Sección 3.2 como la *Ubicación de la PI*, fueron desglosadas y reorganizadas como consecuencia del análisis.

Tabla 3.3: Aplicabilidad de PI basada en microservicios por característica de escenario

	Despliegue de PI		**Ubicación de Sistemas a Integrar**	**Particularidades de Sistemas a Integrar**		**Particularidades de Datos a Integrar**		**Cantidad de Organizaciones**	
	Plataforma de orq. de contenedores	**Ubicación de la PI**	**Todas las características**	**Alta cantidad de sistemas**	**Volatilidad o Inmadurez**	**Masividad de datos/accesos**	**Complejidad de datos**	**Escenario Interorg.**	**Escenario Intraorg.**
Aplicabilidad de PI basada en microservicios	✓	—	—	✓	✓	✓	✓	✓	—

✓ Beneficiado por PI de microservicios. — No beneficiado o no afecta la aplicabilidad

El enfoque consiste en observar para cada subatributo preocupante si una arquitectura de microservicios lo afecta positivamente o no. A partir de esto, se contabiliza para cada característica de escenario cuántos de sus subatributos preocupantes son beneficiados por la arquitectura de microservicios. Sin embargo, el análisis no es en todos los casos un conteo lineal, sino que se valora caso a caso la importancia de los subatributos afectados en el contexto.

En la Figura 3.3, se presenta un ejemplo de comparación hecho a partir de la Tabla 3.2, donde se determina el impacto para el caso de alta cantidad de sistemas. Se observan celdas de tres colores: i) en verde los subatributos preocupantes afectados positivamente por la arquitectura de microservicios, ii) en gris se muestran los que no son afectados, y iii) en rojo se presentan los casos donde los subatributos preocupantes son una preocupación también en arquitecturas de microservicios.

En el ejemplo de la Figura 3.3 se puede observar que de los nueve subatributos preocupantes: seis son beneficiados por microservicios, dos no son afectados y solo uno es también una preocupación de arquitecturas de microservicios. Este último, es el subatributo *Evaluabilidad*, que no posee a priori una relevancia tal como para descartar o contrarrestar los seis subatributos preocupantes beneficiados. Por tanto se afirma que escenarios con alta cantidad de sistemas son beneficiados al aplicar una arquitectura de microservicios en la PI[15].

Se aplica la misma metodología del ejemplo para las demás características de escenarios, exceptuando la categoría *Ubicación de la PI*, que se analiza particularmente:

[15] El subatributo *Uso de Recursos* en casos de alta cantidad de sistemas, cumple la condición para que afecte positivamente (marcado con asterisco "*", y detallado en Sección 3.4), ya que es probable que en dicho contexto se requiera escalamiento en la PI.

Atributo	Subatributo	Alta cantidad de sistemas	Arq. de Microservicios
Eficiencia de Desempeño	Comportamiento del Tiempo	P	—
	Uso de Recursos	P	A+(*)
	Capacidad	—	A+(*)
Mantenibilidad	Modularidad	P	A+
	Reusabilidad	—	A+
	Evaluabilidad	P	P
	Modificabilidad	P	A+
	Testabilidad	—	P
Portabilidad	Adaptabilidad	—	A+
	Instalabilidad	P	—
	Reemplazabilidad	P	A+
Confiabilidad	Madurez	—	—
	Disponibilidad	—	A+
	Tolerancia a Fallas	P	A+
	Recuperabilidad	P	A+
Seguridad	Confidencialidad	—	P
	Integridad	—	P
	No repudio	—	P
	Trazabilidad	—	P
	Autenticidad	—	P

Figura 3.3: Ejemplo de impacto de arq. de microservicios sobre subatributos preocupantes

- **Ubicación de la PI:** Durante el análisis se determinó que esta característica puede separarse en dos partes, en base a su incidencia sobre los subatributos: i) si la PI es desplegada en una plataforma de orquestación de contenedores sobre clústeres, y ii) si el despliegue es público o interno. Por tanto se analiza el caso en forma particular.

  Respecto al primer punto, se observa que una arquitectura de microservicios es particularmente beneficiosa cuando es desplegada en una plataforma de orquestación de contenedores sobre clústeres. En este caso, el impacto de microservicios se optimiza en seis subatributos: *Uso de Recursos*, *Adaptabilidad*, *Instalabilidad*, *Recuperabilidad* y *Seguridad*, tal como se explica en la Sección 3.4.

  Respecto al segundo punto, se entiende que la arq. de microservicios puede aplicarse tanto en ambientes internos como públicos, por lo cual no es determinante para el análisis.

En un ambiente público, los proveedores PaaS usualmente ofrecen servicios por cantidad de recursos usados. En dicho caso, la arq. de microservicios puede ser beneficiosa debido a su uso eficiente de recursos en casos de escalamiento. Por otro lado, las preocupaciones de seguridad del despliegue público lo son también en microservicios. Esto no descarta la arquitectura pero requiere especial enfoque y consideraciones sobre el subatributo.

En un ambiente interno, los recursos de *hardware* usualmente son más limitados que con un proveedor público, en estos casos la arq. de microservicios también puede ser beneficiosa para consumir menos recursos, en casos de escalamiento. Por otro lado, las preocupaciones de seguridad podrían tener menor importancia, ya que el ambiente suele ser más controlado al ser interno.

Por consiguiente, se entiende que el segundo punto no afecta la aplicabilidad, ya que podría utilizarse la arquitectura tanto en ambientes públicos como internos.

- **Ubicación de los sistemas a integrar:**

  Como resultado del análisis se determina que la ubicación de los sistemas integrar, en contraposición con la ubicación de la PI, afecta principalmente dos subatributos: la instalabilidad y el comportamiento del tiempo.

  Sin embargo, se entiende que la arquitectura de microservicios tiene un impacto neutro sobre dichos subatributos, por lo cual se considera que no tiene relevancia sobre la aplicabilidad de la arquitectura.

- **Volatilidad o inmadurez de sistemas a integrar:** En escenarios con sistemas a integrar inmaduros y que cambian frecuentemente, se observa a partir de la Tabla 3.2 que dados los seis subatributos de calidad preocupantes que se identificaron, una arquitectura de microservicios afecta positivamente a cuatro, mientras que solo uno es una preocupación también en arquitecturas de microservicios.

  Por consiguiente, se afirma que escenarios con sistemas a integrar volátiles o inmaduros son beneficiados por aplicar una arq. de microservicios en la PI.

- **Masividad de datos o de accesos de usuarios:** En escenarios con cantidades masivas de datos o picos de muchos usuarios concurrentes accediendo a la PI, se observa que de los siete subatributos preocupantes que se identificaron, una arquitectura de microservicios afecta positivamente a seis y ninguno constituye una preocupación también en microservicios.

  Por otro lado, se observa que probablemente en este contexto se requiera escalamiento en la PI. Por consiguiente los subatributos *Uso de Recursos* y *Capacidad*, cumplen la condición para que sean afectados positivamente por una arq. de microservicios (razón por la cual los afectados positivamente son efectivamente seis).

  Por lo tanto, escenarios con masividad de datos o de accesos son beneficiados por una

arquitectura de microservicios en la PI.

- **Complejidad de datos a integrar** En escenarios donde los datos a integrar requieren de un procesamiento complejo y computacionalmente caro, se observa que de los seis subatributos preocupantes que se identificaron, una arquitectura de microservicios afecta positivamente a cinco y ninguno constituye una preocupación también en microservicios.

  Se observa que probablemente en este contexto se requiera escalamiento en la PI, tanto escalamiento vertical (para soportar procesamiento intensivo) como horizontal (para evitar cuellos de botella). Por tanto los subatributos *Uso de Recursos* y *Capacidad*, cumplen la condición para que sean afectados positivamente por una arq. de microservicios.

  Por consiguiente, escenarios con datos complejos a integrar son beneficiados por una arquitectura de microservicios en la PI.

- **Escenario Interorganizacional** En escenarios con sistemas a integrar de varias empresas, se observa que de los doce subatributos preocupantes identificados, tres son beneficiados por una arq. de microservicios en la PI, dos no son afectados, y siete son también preocupaciones de arquitecturas de microservicios. Sin embargo, de los siete mencionados, cinco corresponden a la seguridad. Por tanto se procede a analizar en mayor detalle la aplicabilidad de la arquitectura.

  El impacto positivo de la arquitectura de microservicios en este tipo de escenarios, se da en los atributos de *Mantenibilidad* y *Portabilidad*, lo cual otorga a la PI una evolución organizada y simplificada. En particular, se observa que dicho impacto se hace más notorio y beneficioso (en comparación con una arq. monolítica) a medida que aumentan las empresas y sistemas involucrados.

  Por otro lado, las preocupaciones que también son compartidas por microservicios, se dan en la seguridad y en los subatributos de *Testabilidad* y *Evaluabilidad*. Respecto a los dos últimos, si bien son una preocupación en la arquitectura, existen formas de mitigarlos parcialmente[16] por lo que se entiende que su impacto es menor en comparación con la mejora sobre la mantenibilidad en general. Debe analizarse en cada contexto de proyecto la importancia dada a estos subatributos, y los mecanismos aplicables para mejorar los mismos, de forma de confirmar la idoneidad de la arquitectura de microservicios.

  Respecto a las preocupaciones de seguridad, en la arquitectura de microservicios existen enfoques para satisfacer los requerimientos, aunque tienen la desventaja de ser aún inmaduros. Es de particular importancia el uso de plataformas de orquestación de contenedores ya que resuelven algunas de las preocupaciones de seguridad (ver Sección 3.4).

[16] Especialmente en el caso de *Testabilidad*, donde algunos autores como Fowler comentan expresamente que no lo consideran un problema debido a la existencia de enfoques y tecnología [23].

Se entiende que la arquitectura de microservicios para la PI puede ser beneficiosa en un escenario interorganizacional. Sin embargo, se debe poner foco en mecanismos complementarios para mejorar la seguridad y, según los requerimientos, también la evaluabilidad y testabilidad. En particular, se debería desplegar la PI en una plataforma de orquestación de contenedores, donde se resuelvan parcialmente las preocupaciones de seguridad.

De los resultados anteriores, se puede extraer un conjunto de características que de existir en un escenario, sugieren que es beneficioso utilizar una arquitectura de microservicios en la PI. Análogamente, pueden extraerse un conjunto de características que permitan indicar si un escenario aparenta no ser beneficiado por una PI basada en microservicios.

3.5.2 Características Compatibles con una PI basada en Microservicios

En esta sección se propone un conjunto de características que sugieren que el escenario es propicio para aplicar una PI basada en una arquitectura de microservicios. Las características surgen en su mayoría de los resultados anteriores aunque también se proponen algunas relacionadas a particularidades de microservicios vistas en la Sección 2.1 :

- La PI es desplegada en una plataforma de orquestación de contenedores sobre clústeres.
- Se requiere soportar picos o cantidad elevada de datos a integrar
- Se requiere soportar picos o cantidad elevada de accesos de usuarios.
- Existe un número alto de sistemas a integrar
- Los sistemas a integrar son inmaduros o muy cambiantes.
- Se tienen sistemas a integrar de múltiples empresas (escenario interorganizacional)
- Los datos a integrar son complejos y requieren un procesamiento computacionalmente caro o tecnología especializada.
- Se requiere modificar la PI de forma frecuente sin volver a desplegar la PI entera, en casos de alta disponibilidad[17].

3.5.3 Características No Ideales para una PI basada en Microservicios

Se mencionan algunas características de escenarios que se considera que no son las ideales para utilizar una PI con arquitectura de microservicios. A criterio de autor, no descartan la aplicabilidad de microservicios, pero si disminuyen u opacan algunas de sus principales ventajas.

[17] Esta característica se propone en base a los conceptos de microservicios explicados en la Sección 2.1.

Si el escenario incluye algunas de las características siguientes, el uso de microservicios podría ser contraproducente:

- No contar con una plataforma de orquestación de contenedores sobre clústeres
  - No contar con dicha plataforma reduce los beneficios de la arq. de microservicios y afecta los subatributos *Uso de Recursos*, *Adaptabilidad*, *Instalabilidad*, *Recuperabilidad* y *Seguridad*.
- Los sistemas son pocos y estables
  - Si hay pocos sistemas entonces no es necesario mantener una fuerte modularidad para soportar su mantenimiento. Además, si sufren pocas modificaciones entonces probablemente no se tengan requerimientos considerables de mantenibilidad.
- Escenario intraorganizacional
  - Si se tiene un escenario intraorganizacional, entonces la mantenibilidad es más controlable que si hay varias partes involucradas. La empresa tiene total control sobre la evolución de la PI, por consiguiente la modularidad y mantenibilidad que aporta aplicar microservicios no es tan determinante.

    Esto no implica que no existan escenarios intraorganizacionales donde sea beneficioso una PI basada en microservicios (*Netflix* por ejemplo puede considerarse un caso intraorganizacional), sin embargo otras características de escenarios deben estar presentes para que eso ocurra.
- Pocos usuarios y datos
  - Tener pocos usuarios y pocos datos implica que no sea necesario escalamiento y que las ventajas de microservicios en escalabilidad carezcan de importancia.

3.6 Escenarios Compatibles de Ejemplo

Se muestran a continuación tres ejemplos breves de escenarios de integración que se entiende que son compatibles con una PI basada en microservicios. Estos poseen algunas de las características mencionadas en la Sección 3.5.2.

3.6.1 Escenario 1: Sistemas Cambiantes en la Nube

En la Figura 3.4 se plantea un escenario donde los sistemas a integrar y la PI están desplegados en una nube pública, en un contexto de alta disponibilidad. Se requiere la integración de

una combinación de aplicaciones SaaS y sistemas de la propia empresa, de los cuales algunos son inmaduros.

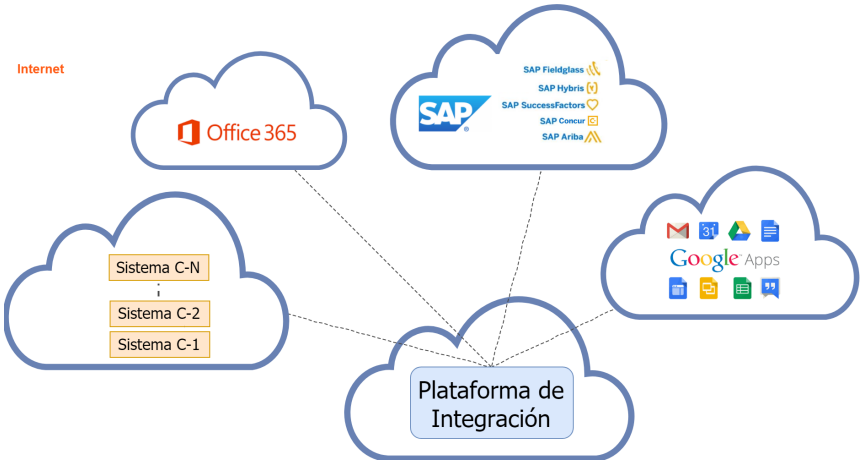

Figura 3.4: Escenario 1: PI desplegada en nube pública.

En el diagrama del escenario los sistemas C-1 a C-N corresponden a sistemas de la propia empresa, de los cuales algunos son aún inmaduros o cambiantes. Por otro lado se asume que en el escenario la PI es desplegada en una plataforma de orquestación de contenedores sobre clústeres.

Se observa que el escenario posee tres características de las presentadas en la Sección 3.5.2: i) Sistemas inmaduros o cambiantes, ii) Alta disponibilidad, y iii) Despliegue de la PI en una plataforma de orquestación de contenedores sobre clústeres.

3.6.2 Escenario 2: Integración Interorganizacional

En la Figura 3.5 se plantea un escenario interorganizacional donde se requiere integrar sistemas ubicados en redes internas diferentes. Se requiere además procesar una gran cantidad de datos heterogéneos provenientes de los distintos sistemas.

La PI está ubicada en la red interna de una de las partes, y se despliega en una nube privada, sobre una plataforma de orquestación de contenedores sobre clústeres.

Se observa que el escenario posee tres características de las presentadas en la Sección 3.5.2: i) Alta cantidad de datos, ii) Escenario interorganizacional, y iii) Despliegue de la PI en una plataforma de orquestación de contenedores sobre clústeres.

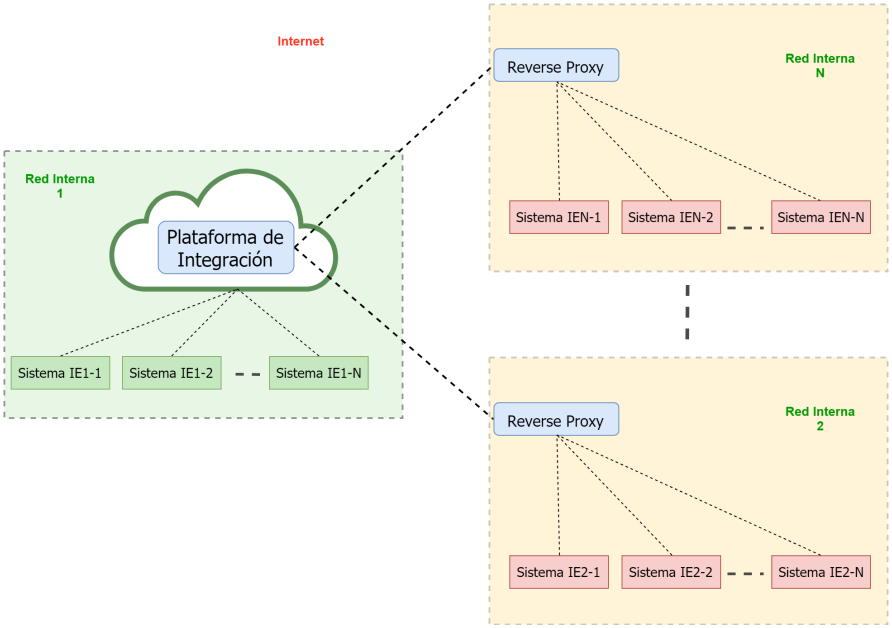

Figura 3.5: Escenario 2: PI interorganizacional.

3.6.3 Escenario 3: Integración con Datos Complejos

En la Figura 3.6 se plantea un escenario con datos complejos a integrar, los cuales necesitan tecnología específica para alguna de las funcionalidades de integración de la PI (p. ej. transformación, ruteo). Se asume además que los sistemas a integrar cambian frecuentemente. Se asume un despliegue interno en una una plataforma de orquestación de contenedores sobre clústeres, en una nube privada.

Se requiere que la PI cuente con componentes de transformación matemática compleja. Para esto, se requiere poder utilizar diferentes bibliotecas especializadas, como por ejemplo GSL[18] o FFTW[19] para procesar transformaciones de Fourier.

Se observa que el escenario posee tres características de las presentadas en la Sección 3.5.2: i) Datos complejos que requieren tecnología específica, ii) Sistemas inmaduros o cambiantes y iii) Despliegue de la PI en una plataforma de orquestación de contenedores sobre clústeres.

[18] https://www.gnu.org/software/gsl/
[19] http://www.fftw.org/

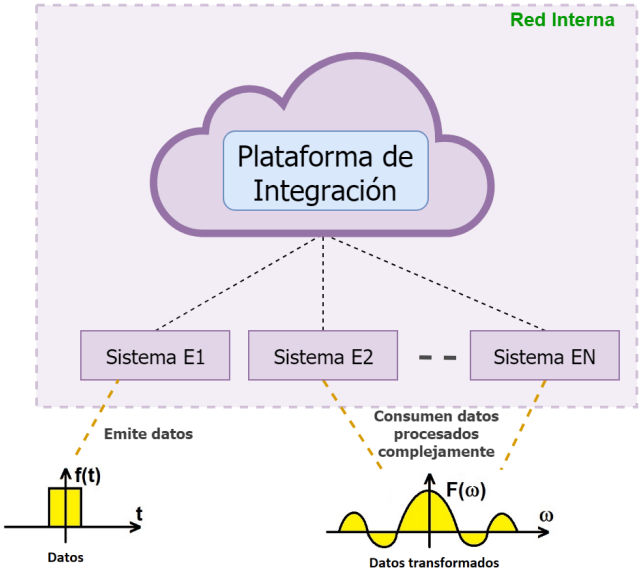

Figura 3.6: Escenario 3: Integración de datos complejos

3.7 Conclusiones

En el capítulo se analizó la aplicabilidad de una arquitectura de microservicios para la construcción de PIs. En particular se planteó determinar cómo afecta dicha arquitectura a la PI y cuáles escenarios de integración son propicios para aplicar el enfoque. Como forma de acotar el alcance, se plantearon primero distintas características que pueden poseer los escenarios; luego, se estudió para las mismas cuáles subatributos de calidad de la PI son preocupantes. Finalmente, se analizó cómo la arquitectura de microservicios incide sobre dichas preocupaciones.

Los resultados principales del capítulo se dividen en dos partes. Por un lado, se presenta un análisis referente al impacto de aplicar una arquitectura de microservicios en PIs. Se observa en dicho análisis que la existencia de determinadas tecnologías y enfoques permiten satisfacer subatributos como la *Tolerancia a Fallas*, que usualmente son preocupantes en sistemas distribuidos. Se observa además que el impacto de microservicios no debe ser analizado en forma aislada sino como un conjunto de enfoques, que incluye el uso de contenedores, la aplicación de *DevOps* y el despliegue en plataformas de orquestación de contenedores. Esto se debe a que dichos enfoques aportan soluciones a algunos desafíos o preocupaciones de la arquitectura, evitando que se conviertan efectivamente en un problema.

Por otro lado, se propone como resultado un conjunto de características de referencia que contribuyen a determinar para un escenario si es propicio usar una PI basada en microser-

vicios. Se observa que algunas características como la ubicación de sistemas a integrar no afectan la aplicabilidad de la arquitectura. En base a los resultados se presentaron algunos escenarios de ejemplo, para los cuales se entiende que aplica una PI basada en microservicios.

A modo de reflexión, se observa que en la práctica los escenarios de integración pueden tener requerimientos empresariales, o del contexto del proyecto, que condicionen la importancia de algunos atributos de calidad. Este condicionamiento, podría afectar la decisión de aplicabilidad de microservicios para la PI. Para estos casos, se propone utilizar los resultados iniciales de impacto de escenarios y microservicios (Secciones 3.3 y 3.4) contrastados en la Tabla 3.2, y determinar a partir de ellos la aplicabilidad según los atributos considerados de importancia en dicho contexto.

Por último, se destaca también el enfoque metodológico basado en el propuesto en [14], para analizar el impacto sobre un sistema a partir de sus atributos de calidad. El propuesto en esta tesis se centra en la obtención de subatributos preocupantes, mientras que el propuesto en [14] clasifica cada subatributo por el nivel de satisfacción logrado. Se observa además, que algunos aspectos parcialmente desacoplados de los atributos de calidad, como el impacto sobre los tiempos de construcción de la PI, escapan al alcance del análisis.

4 Propuestas de Arquitectura

En este capítulo se presentan propuestas de arquitectura para Plataformas de Integración basadas en microservicios. Se define una propuesta principal y variantes alternativas que aplican a distintos contextos, o que analizan decisiones de diseño alternativas.

El capítulo se organiza de la siguiente manera. En la Sección 4.1 se define el marco de trabajo. En la Sección 4.2 se presenta la propuesta principal de arquitectura. En la Sección 4.3 se muestran variantes alternativas y otras consideraciones de diseño analizadas. Por último en la Sección 4.4 se presentan las conclusiones del capítulo.

4.1 Marco de Trabajo

4.1.1 Plataforma de Integración

Para las propuestas de arquitectura, se considera una PI basada en la definición mostrada en la Sección 2.3. Concretamente, la PI brinda soporte al ciclo de vida de Soluciones de Integración (SI) y permite escalarlas de forma manual o automática.

Las SIs son construidas por los usuarios en base a componentes de integración (p. ej. transformación), que ofrecen capacidades de conectividad, mediación y mensajería. Las PIs complementan las funcionalidades de las SIs con servicios de infraestructura, aportando entre otros seguridad, supervisión (i.e. *monitoring*), conectividad y gestión. En la Figura 4.1 se muestra un esquema de los conceptos principales.

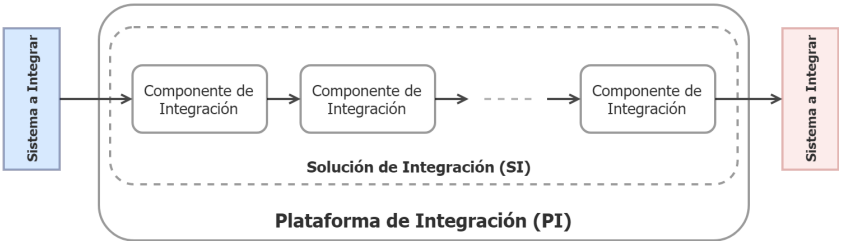

Figura 4.1: Esquema de los conceptos de PI, SI y componentes de integración[1].

4.1.1.1 Soluciones de Integración

La definición de SI considerada en este trabajo, se asemeja a la de "flujo de integración" descrita por Gartner en [6], y se caracteriza por ser una serie de pasos que se ejecutan con el fin de integrar dos o más sistemas. Estas soluciones son desplegadas, ejecutadas y gestionadas en la PI. Cada paso de una SI es un componente de integración configurable que realiza una operación de integración específica (p. ej. transformación de datos).

La ejecución de una SI se inicia a partir de componentes de integración especiales llamados "Disparadores", los cuales reaccionan a eventos, y finaliza cuando el último componente de la SI es ejecutado. Durante la ejecución cada componente de integración ejecuta su acción e invoca al siguiente componente (transmitiéndole información), hasta finalizar el flujo. Cabe agregar que por cada evento que activa a un componente disparador, se ejecuta una nueva instancia independiente de la SI.

4.1.1.2 Componentes de Integración

Para implementar soluciones de integración, la PI provee un conjunto de componentes de integración configurables. Un ejemplo es un componente "transformación", que se configura especificando la transformación concreta a realizar. Los componentes de integración pueden ser de dos tipos: "disparador" o "acción". Los primeros desencadenan la ejecución de la SI, mientras que los segundos efectúan una acción de integración puntual.

Los componentes de integración elegibles son predefinidos en la PI, pero también extensibles, pudiendo adicionar nuevos. En particular, los componentes predefinidos considerados surgen de realizar un desglose de los "componentes base" propuestos en el trabajo de González y Ruggia [1]. Estos son: i) Conector Simple (p. ej. para HTTP), ii) Conector Específico (p. ej. para Salesforce), iii) Virtualización de Servicios, iv) Transformación, v) Ruteo, vi) Supervisión y vii) Mensajería.

[1] Los sistemas a integrar pueden ser varios, se muestran solo dos para mejorar la legibilidad del esquema. Análogo para las SIs, pueden desplegarse múltiples soluciones en la misma PI.

Se excluyen del marco de trabajo, los componentes "avanzados" e "interorganizacionales" del artículo (ver Sección 2.3).

4.1.1.3 Ejemplo Conceptual

En el siguiente ejemplo se muestran los conceptos descritos en las secciones anteriores. Se asume que se requiere integrar tres sistemas desplegados en la nube: *Sistema C1*, *C2* y *C3*; e interactuar con un servicio de correo electrónico externo como *Gmail*[2]. Para abordar esta problemática se utiliza una PI desplegada en la nube y se construye la SI presentada en la Figura 4.2.

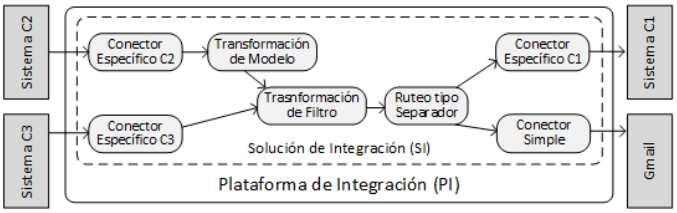

Figura 4.2: Ejemplo de los conceptos de PI, SI y componentes de integración.

Se asume que los sistemas *C2* y *C3* envían una gran cantidad de datos que consisten en órdenes de compra al sistema *C1*. Los tres sistemas soportan protocolos de comunicación diferentes, que a su vez no son estándar (p. ej. *SAP Remote Function Call*[3]). Además, *C2* maneja un modelo de datos diferente a *C1* y *C3*. Se requiere que sólo las órdenes que cumplan determinadas condiciones (p. ej. que refieran a un cierto producto) se envíen a *C1*, notificando por correo electrónico a una casilla de *Gmail*.

La SI se construye con los siguientes componentes de integración: *Conector Específico*, *Conector Simple*, *Transformación* y *Ruteo*. Debido a que los sistemas utilizan protocolos de comunicación que no son estándar, se usan conectores específicos para obtener datos de los sistemas *C2* y *C3*, y para enviar datos a *C1*. Las órdenes de compra enviadas desde *C2* son procesadas con un componente de transformación de modelo y las órdenes que no cumplen las condiciones son descartadas por un componente de transformación de filtro. El componente ruteo de tipo separador envía los datos recibidos a C1 y a una casilla de Gmail, usando los conectores correspondientes.

4.1.2 Forma de Presentación de las Propuestas

Para presentar las propuestas de arquitectura, se hace uso del modelo de vistas 4+1 de Kruchten [59], visto en la Sección 2.5.1. Como notación se utiliza UML, considerando para

[2] https://mail.google.com
[3] https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abenremote_function_call_glosry.htm

las vistas los diagramas mostrados en la Figura 4.3.

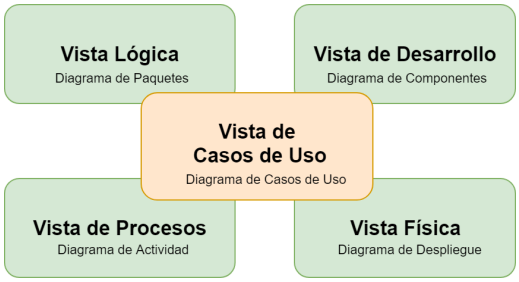

Figura 4.3: Esquema del modelo 4+1 y diagramas de UML utilizados

En este capítulo se presenta primero una propuesta principal de las cinco vistas, y luego se muestran variantes posibles, aplicables a otros contextos o decisiones de diseño. Las variantes presentadas pueden corresponder a una o más vistas del modelo. Finalmente, también se comentan otros aspectos que se consideraron durante el análisis.

4.2 Propuesta Principal de Arquitectura

En esta sección, se presenta la propuesta principal de arquitectura para PI basada en microservicios. La PI propuesta es de propósito general y basada en coreografía. Además, se entiende que es aplicable a diversos escenarios, delimitados por la aplicabilidad de la propia arquitectura de microservicios (véase Capítulo 3). Se organiza el planteo en cinco secciones, donde cada una describe a una de las vistas del modelo 4+1.

4.2.1 Vista de Casos de Uso

En la Figura 4.4 se presentan los principales casos de uso de la PI considerada, así como los actores que los ejecutan. Algunos casos complementarios se omiten en el diagrama para mayor claridad, estos se muestran en el Apéndice E. Se describen a continuación los casos mas destacables.

El caso de Crear e Implementar SIs, es realizado por los administradores, quienes agregan una serie de componentes de integración vinculados entre si. Como se mencionó anteriormente, estos componentes pueden ser de dos tipos: “Disparador” o “Acción”. Cuando se agregan componentes a la SI, es necesario configurarlos. Por ejemplo si se agrega una transformación, la configuración detalla cuál es la transformación concreta a realizar.

El caso de Desplegar una SI, implica la instanciación de los componentes correspondientes a la SI construida, y la puesta en marcha de la misma. Esto incluye transmitir a los compo-

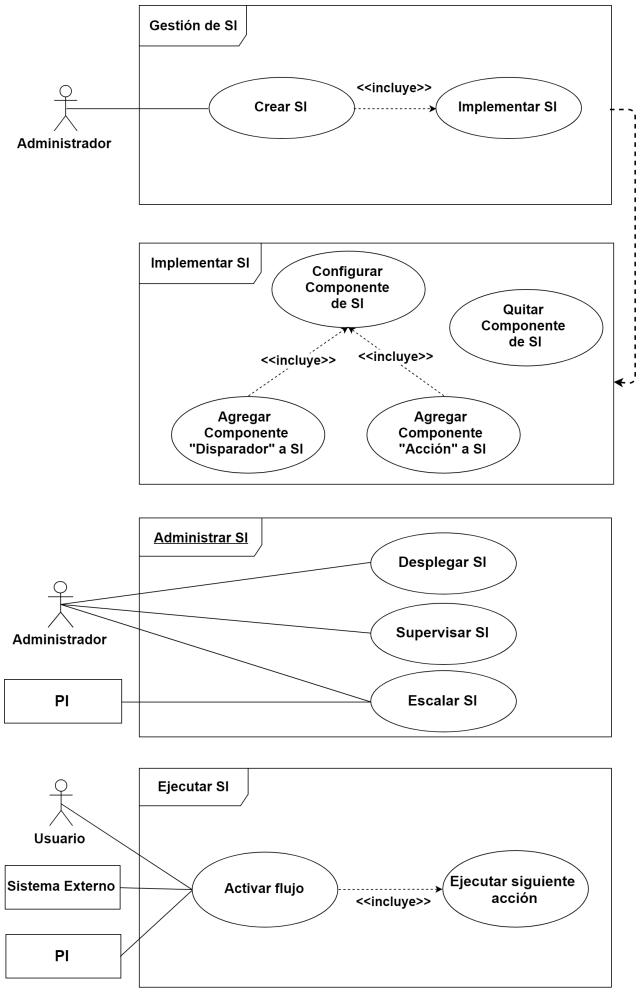

Figura 4.4: Vista de Casos de Uso relevantes

nentes las configuraciones necesarias, definidas durante la construcción. Una vez finalizado el despliegue, los componentes disparadores ya poseen la capacidad de reaccionar a eventos.

El caso de Supervisar SIs, es realizado por el administrador para analizar posibles errores en la ejecución o en la integración de los datos, así como también observar indicadores de ejecución de las SIs (p. ej. cantidad de datos procesados).

El caso de Escalar SIs, puede realizarse de forma manual, es decir, por acción directa de un administrador, o también de forma automática en base a reglas definidas. Así, la PI puede por ejemplo detectar que los recursos están superando un cierto límite, y automáticamente

escalar instanciando nuevos componentes para enfrentar la demanda.

El caso de Ejecución de SIs, inicia cuando un disparador activa la SI, luego de lo cual se invoca al próximo componente de integración para ejecutar la siguiente acción. Los disparadores pueden ser activados por tres actores distintos: un usuario (p. ej. mediante una acción en una interfaz), un sistema externo (p. ej. invocando un servicio web) o la propia PI (p. ej. con un temporizador que active la SI una vez por día).

4.2.2 Vista Lógica

Esta vista parte de los casos de uso identificados en la sección anterior, y se enfoca en una descomposición abstracta en entidades para soportar dichos casos [59]. Estas entidades, son tomadas del dominio general del problema. El desglose realizado, se apoya sobre el principio de responsabilidad única y los conceptos de separación de entidades propuestos por Richardson en [20].

Se presentan tres diagramas, que muestran la descomposición realizada. Se parte de un diagrama de alto nivel, luego un diagrama intermedio muestra cómo los conceptos de alto nivel son desglosados en entidades, y por último un diagrama de bajo nivel muestra cómo las entidades interactúan entre sí.

4.2.2.1 Vista Lógica de Alto Nivel

La Figura 4.5 muestra la vista lógica de alto nivel de abstracción, donde se muestran las dependencias entre los conceptos principales.

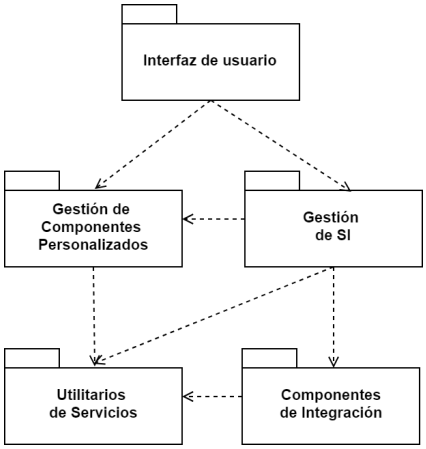

Figura 4.5: Vista Lógica de Alto Nivel

Se observa que el usuario a través de la interfaz, puede crear y desplegar soluciones de integración con el apoyo del subsistema *Gestión de SI*. También puede gestionar componentes personalizados[4], los cuales corresponden a una extensión de los componentes de integración predefinidos.

Las SIs son construidas a partir de componentes de integración y componentes personalizados, por lo cual el subsistema *Gestión de SI* interactúa con los subsistemas *Gestión de Componentes Personalizados* y *Componentes de Integración*.

Por otro lado, las SIs y sus componentes se apoyan en utilitarios de servicios (p. ej. un balanceador de carga), para utilizar otras funcionalidades auxiliares requeridas para el funcionamiento.

4.2.2.2 Desglose de Vista de Alto Nivel

La Figura 4.6 muestra un desglose de los conceptos abstractos mostrados en la Figura 4.5.

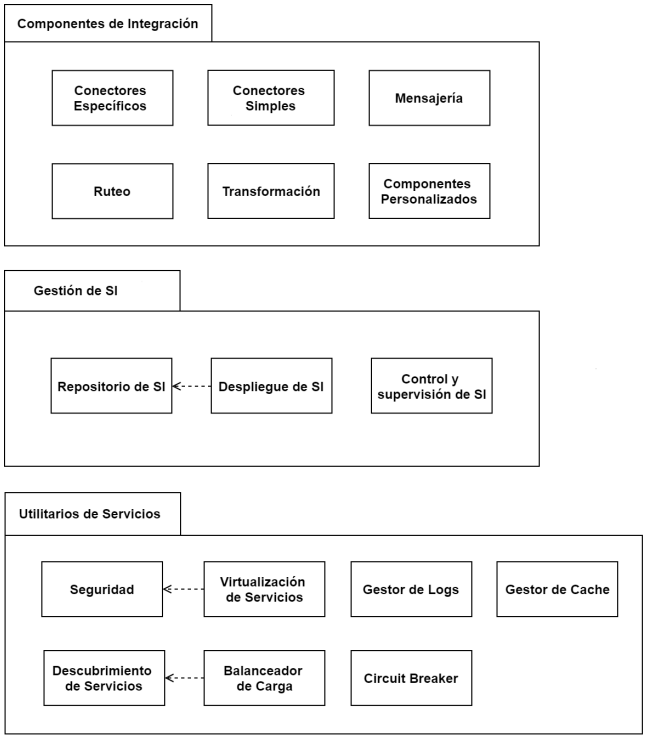

Figura 4.6: Desglose de vista lógica de alto nivel

[4] La gestión de componentes personalizados se muestra en los casos de uso complementarios en el Apéndice E.

Los componentes de integración se desglosan en los componentes mostrados en la Sección 4.1.1.2, incluyendo como forma de extensión a los mismos, los componentes personalizados. Se describen a continuación entidades cuya responsabilidad no es trivial:

La entidad *Conectores Específicos*, resuelve conectividad compleja con sistemas a integrar. Esto se da por ejemplo en protocolos de comunicación no estándares o que requieren una interacción especial. Un ejemplo es un sistema de la empresa SAP[5] que se comunica con el protocolo propietario *SAP Remote Function Call*.

La entidad *Conectores Simples*, refiere a conectores básicos de protocolos estándar (p. ej. HTTP, SMTP). Estos permiten por ejemplo, hacer una invocación simple a un servicio (p. ej. un pedido *GET* de HTTP).

La entidad *Control y Supervisión de SI*, concentra el soporte a los casos de uso de supervisión y escalamiento de las SIs.

La entidad *Repositorio de SI* se encarga de almacenar un versionado con la información de las SIs construidas. Dicha información es usada para realizar el despliegue de una SI.

La entidad *Virtualización de Servicios* permite exponer servicios web. Estos pueden ser usados para obtener o enviar información a los sistemas a integrar vinculados a las SIs. A diferencia del conector simple, la entidad no consume servicios ni realiza interacciones en protocolos habituales, sino que expone servicios web.

La entidad *Gestor de Logs* se encarga de recolectar e indizar la información de ejecución de los componentes. Esto permite conglomerar los registros de forma automática y tener una supervisión en tiempo real de la ejecución de las SIs.

4.2.2.3 Vista Lógica de Bajo Nivel

A partir del desglose de conceptos mostrados, se detalla en la Figura 4.7 el diagrama de bajo nivel, que muestra de forma holística las entidades y su relacionamiento. Se considera que la forma de coordinación es coreográfica.

Se observa que la interfaz de usuario funciona como centralizador de las funcionalidades otorgadas por las principales entidades de gestión de SIs.

La entidad *Despliegue de SI*, obtiene de la entidad *Repositorio de SI* la información de la versión a desplegar. Luego como parte de la puesta en marcha de las SIs, instancia los componentes de integración y componentes personalizados correspondientes, y otorga sus ubicaciones a la entidad *Descubrimiento de Servicios*. Finalmente, proporciona a cada componente la información de cual es el próximo componente a invocar, en la coreografía de la SI.

[5] https://www.sap.com

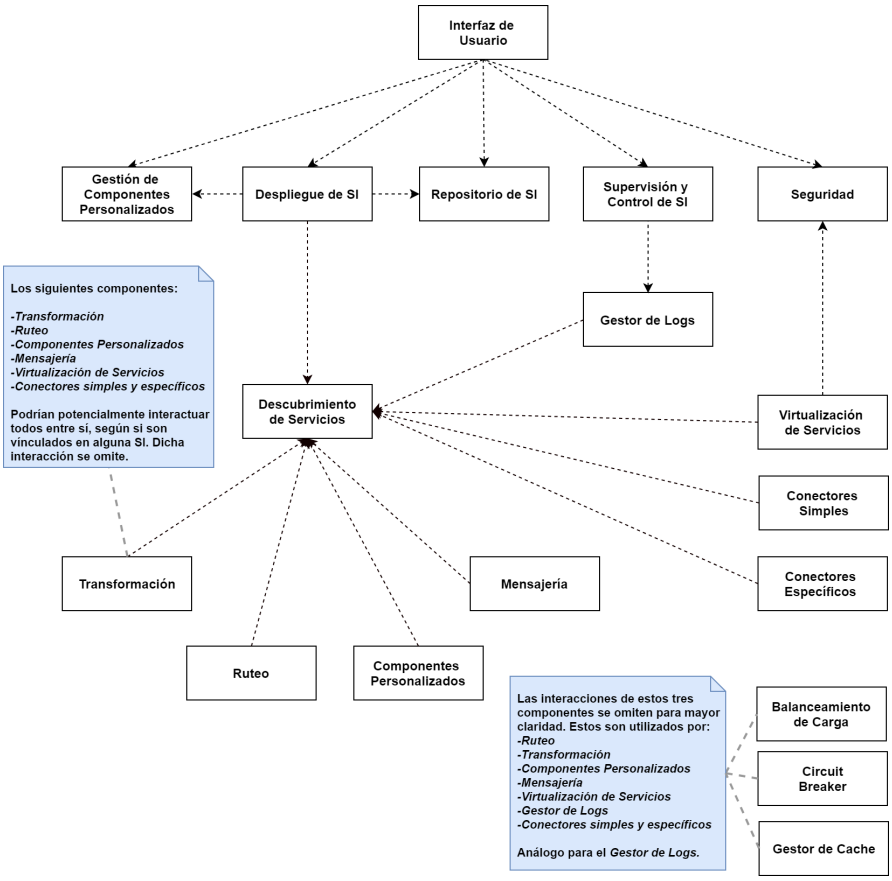

Figura 4.7: Vista Lógica holística para el desglose

Por otro lado, el componente de Seguridad aporta autenticación y autorización a la PI y también a los servicios web expuestos por el componente *Virtualización de Servicios*.

Se observa además, que cada invocación hecha por un componente de integración sobre otro, requiere que se consulte a la entidad *Descubrimiento de Servicios* la ubicación en la red del componente. Incluso si la comunicación ocurre mediante mensajería, se debe primero consultar la ubicación de la entidad *Mensajería*.

Como se muestra en las notas de la Figura 4.7, algunas vinculaciones se omitieron para mayor claridad. Por ejemplo, todos los componentes de integración requieren de las entidades utilitarias *Circuit Breaker*, *Balanceamiento de Carga* y *Gestor de Cache*.

4.2.3 Vista de Desarrollo

En la Figura 4.8 se presenta la vista de desarrollo, la cual describe la modularización en subsistemas, servicios y componentes, y detalla la organización estática en el sistema.

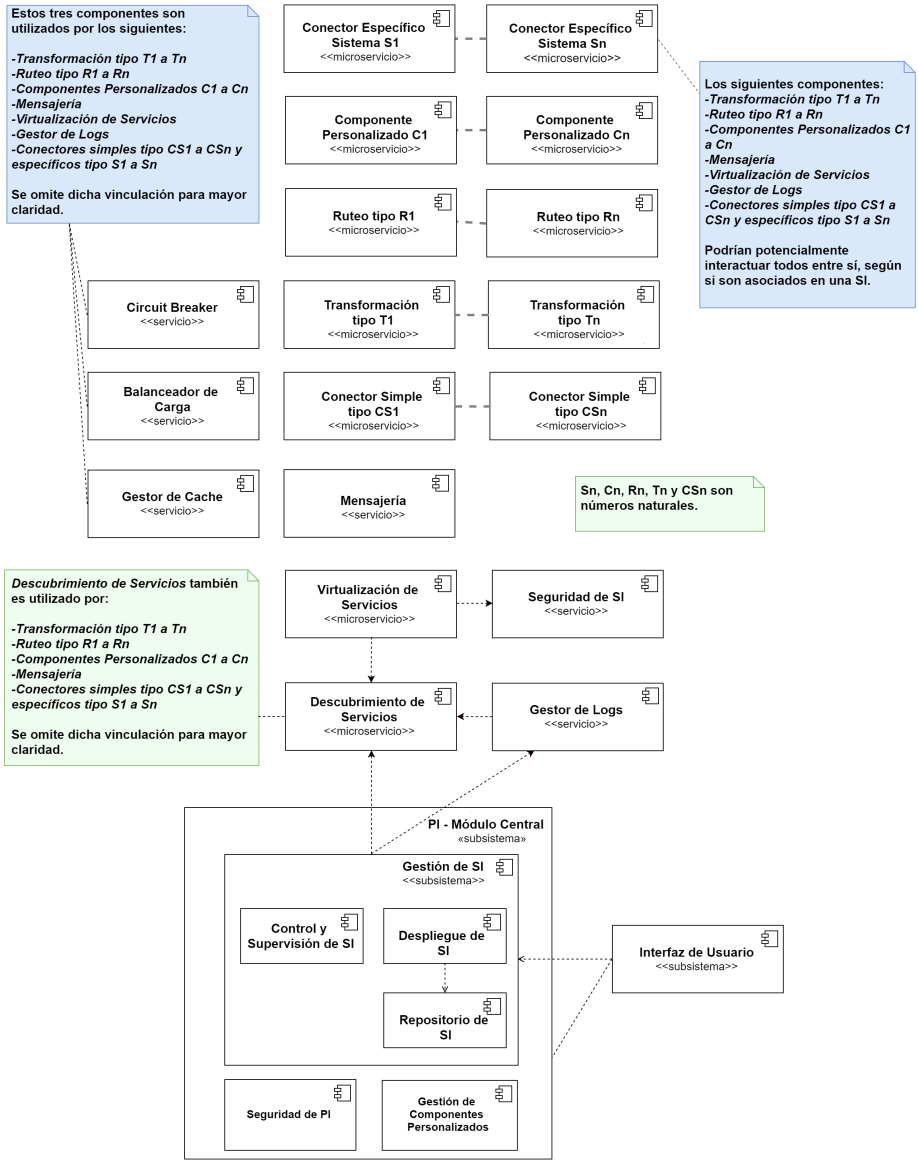

Figura 4.8: Vista de desarrollo

Los conceptos surgen en su mayoría de las entidades definidas en la vista lógica. Al diagrama se introduce la anotación *«microservicio»* para expresar que un componente etiquetado es un microservicio[6].

En el diagrama, puede observarse una separación en dos partes importantes. Por un lado, se tiene el subsistema *PI-Módulo Central*, el cual concentra la parte administrativa de la PI, que incluye principalmente los casos de uso de gestión de SIs como la supervisión y el despliegue. La segunda parte de la vista está compuesta por un conjunto de servicios y microservicios que asisten a la ejecución de SIs, sus actividades de integración y la seguridad de las mismas.

El subsistema *PI-Módulo Central* al encargarse solo de casos de uso administrativos, es usado de forma poco frecuente y demanda pocos recursos, en particular, comparado con componentes que soportan la ejecución de las SIs. Además, sus submódulos no tienen requerimientos elevados de mantenibilidad, ya que son poco cambiantes. Por tanto, se entiende que aplicar microservicios en dicha parte de la PI no tiene un impacto positivo que justifique la separación, por lo cual se la diseña como un subsistema monolítico.

Para la segunda parte de la vista, se analiza cada componente para determinar si se trata de un servicio o un microservicio. Esto se decide en base a parámetros de tamaño y características de microservicios descritas en el Capítulo 2. A modo de ejemplo, el componente *Mensajería* se etiquetó como servicio porque se entiende que su tamaño[7] excede las definiciones de microservicios. El hecho de que no sean microservicios, no descarta que puedan estar desplegados en contenedores.

Es destacable el bajo nivel de abstracción definido para los componentes de integración, en contraste a los mostrados en la Sección 4.1.1.2. Se presentan por ejemplo múltiples variantes de *Ruteo* (*Ruteo tipo R1* a *Rn*), cuyo motivo es disponer de componentes más específicos. Así, se permite definir componentes más puntuales como "Ruteo Basado en Contenido" o "Separador" (i.e. *Splitter*). Esto implica mejoras en mantenibilidad y uso de recursos, que se detallan en el Capítulo 5. Un subcaso o posible instanciación del diseño de abstracción anterior, entre otros posibles, es definir componentes de integración para cada uno de los EIPs.

Por otro lado, se separa la entidad de seguridad en dos: *Seguridad de PI* y *Seguridad de SI*. La primera se encarga de resolver la autenticación y autorización de la parte de gestión de la PI. La segunda, permite resolver aspectos análogos pero para las SIs, particularmente para servicios expuestos por el componente *Virtualización de Servicios*. El componente *Seguridad de SI*, maneja además distintos usuarios y autorizaciones, y podría tener escenarios de autenticación más exigentes, por ejemplo para proteger datos integrados de múltiples empresas.

Una decisión de diseño que escapa en parte a la Figura 4.8 pero es interesante de comentar, es determinar los componentes elegibles en la interfaz de usuario al construir SIs. Por razones

[6] La anotación *«microservicio»* no necesariamente implica que el componente es desplegado en un contenedor, lo cual es una decisión de despliegue.

[7] Se consultó a modo de referencia, los fuentes del producto RabbitMQ para analizar el tamaño de un componente de mensajería (https://github.com/rabbitmq/rabbitmq-server).

prácticas, la cantidad de componentes de integración disponibles al usuario no puede ser grande, ya que se entiende que afecta la usabilidad de la PI. También deben tomarse en cuenta los diferentes perfiles de usuario[8] que utilizan las PI [46][53].

En esta propuesta, se plantea corresponder directamente (uno a uno) los componentes de integración definidos como microservicios con los componentes elegibles en la interfaz de usuario. Es decir, si existe el microservicio "Ruteo Tipo R1", entonces el usuario en la interfaz puede elegir un componente "Ruteo Tipo R1" para construir SIs. Respecto al número de componentes elegibles, se entiende que la cantidad disponible a usuarios de perfil técnico debería ser similar a la cantidad de EIPs, dada la amplia aceptación de estos. Más concretamente, se entiende que la cantidad de componentes de integración disponibles para el usuario en la interfaz debería estar entre treinta y cincuenta según el propósito técnico de la PI. Además, se sugiere un número menor para usuarios sin perfil técnico, manteniendo la vinculación directa entre microservicio y componente elegible en la interfaz.

4.2.4 Vista de Procesos

La vista de procesos toma en consideración cómo el sistema resuelve la ejecución, concurrencia y manejo de hilos [59]. En este trabajo, el enfoque principal para esta vista es mostrar la forma de ejecución de una SI genérica, y también cómo se resuelven ciertos casos de uso puntuales.

Se agrupa por tanto la sección en tres partes, que discuten la resolución de problemas puntuales de la PI: i) Despliegue de SI, ii) Ejecución de SI y iii) Gestión de Logs. Se eligen estos por ser los casos de ejecución más interesantes de analizar, en materia de decisiones de diseño o complejidad de las interacciones.

4.2.4.1 Despliegue de SI

En la Figura 4.9 se diagrama la puesta en marcha de una solución de integración, la cual es llevada a cabo por el componente *Despliegue de SI*, perteneciente al subsistema *Gestión de SI*.

El subsistema obtiene del repositorio la información de la SI correspondiente a la versión a desplegar, así como la información de los componentes personalizados y componentes de integración pertenecientes a dicha SI. Una vez son obtenidos los fuentes de los componentes y utilitarios correspondientes, se envían a la plataforma de orquestación de contenedores para que sean desplegados.

Luego de efectuado el despliegue, se registran las ubicaciones de los nuevos servicios en el componente *Descubrimiento de Servicios*. Finalmente, a cada uno de los componentes de

[8] Por ejemplo, Gartner caracteriza los perfiles *Citizen Integrator*, *Ad-Hoc Integrator* y *Specialist Integrator* [46].

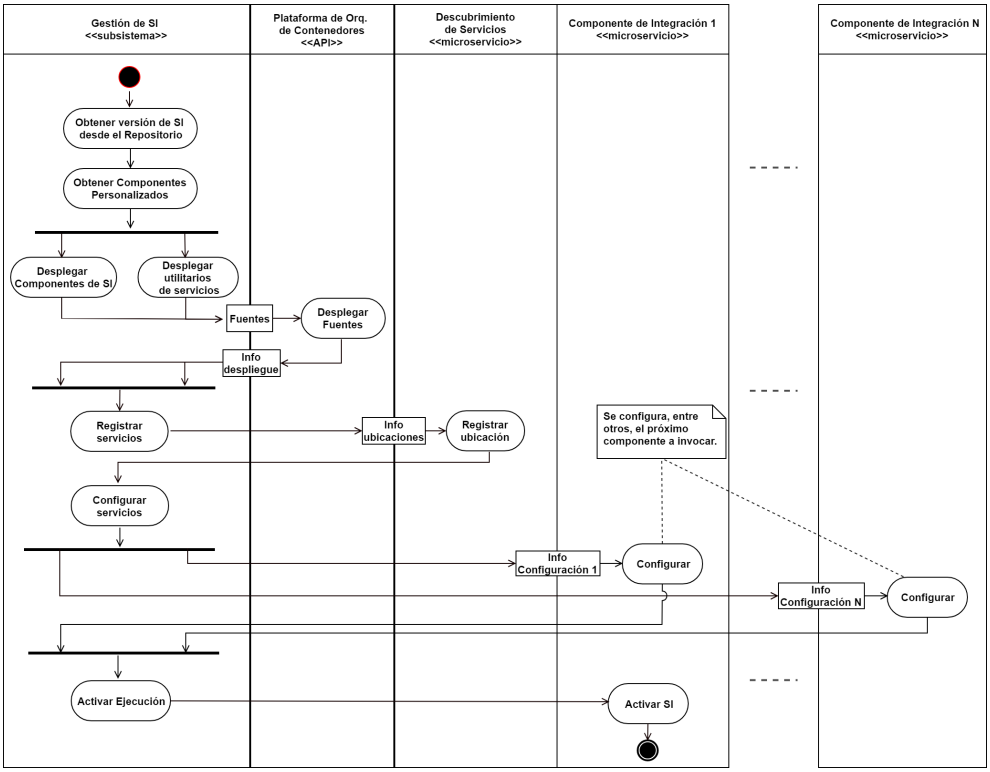

Figura 4.9: Vista de Procesos - Despliegue de SI

integración se les transmite simultáneamente la configuración de ejecución, que incluye a quien ejecutar en el próximo paso, en el marco de coreografía.

Una vez desplegados y configurados los componentes de integración, se activan los componentes disparadores (en el caso de la Figura 4.9 es el *Componente de Integración 1*), a los cual se finaliza el despliegue y la SI queda activa a eventos.

4.2.4.2 Ejecución de SI

Se presentan en esta sección dos ejemplos representativos de ejecución de SI. El primero es un envío de datos a integrar de un sistema a otro (i.e. *one-way*). El segundo es un envío y obtención de datos integrados por parte de un usuario (i.e. *request-response*).

Las ejecuciones muestran la coordinación coreográfica en SIs, donde cada microservicio invoca al próximo. Por otro lado, muestran también dos formas representativas de integración y de activar la SI.

Ejemplo 1: Sistema a Sistema (*one-way*)

En la Figura 4.10 se presenta un diagrama de actividad para la ejecución coreográfica de una SI genérica que integra dos sistemas. En ella, datos de un *Sistema J* son sometidos a una serie de acciones de integración, para luego ser enviados al *Sistema K*, a través del conector específico correspondiente.

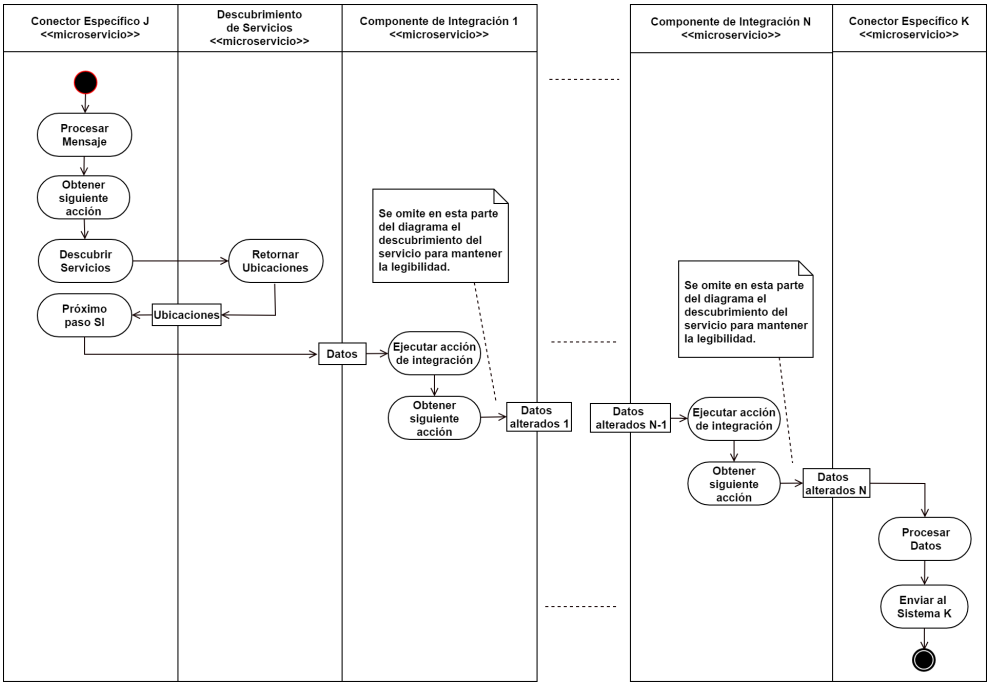

Figura 4.10: Vista de Procesos - Ejecución de SI que integra dos sistemas (*one-way*)

Cada componente, obtiene desde su configuración al próximo componente a invocar y utiliza el servicio de *Descubrimiento de Servicios* para determinar su ubicación. Notar que si bien el ejemplo mencionado finaliza el flujo al invocar al *Sistema K*, este podría continuar. En dicho caso, intercambiar datos con el *Sistema K* sería un paso intermedio y la SI podría incluir más sistemas en la integración.

Se plantea una interacción asincrónica, en el sentido de que un componente no queda esperando la respuesta a la acción de integración del siguiente componente invocado. Esto no quita que la comunicación pueda ser implementada, por ejemplo, con HTTP; ya que se podría esperar solamente confirmación de que la invocación fue recibida, pero no así el resultado de la acción de integración. En la Figura 4.10 se asume una comunicación de este tipo, para omitir interacciones con el componente *Mensajería* y dar mayor claridad al diagrama.

Ejemplo 2: Invocación de Usuario (*request-response*)

En la Figura 4.11 se presenta un diagrama de actividad similar al anterior pero cuya invocación requiere una respuesta por parte de la SI, la cual es activada por un usuario a través de una invocación a un servicio web. Dicho servicio es expuesto con el componente de *Virtualización de Servicios*. Análogo a diagramas anteriores, los datos del pedido son sometidos a una serie de acciones de integración antes de ser enviados al Sistema K.

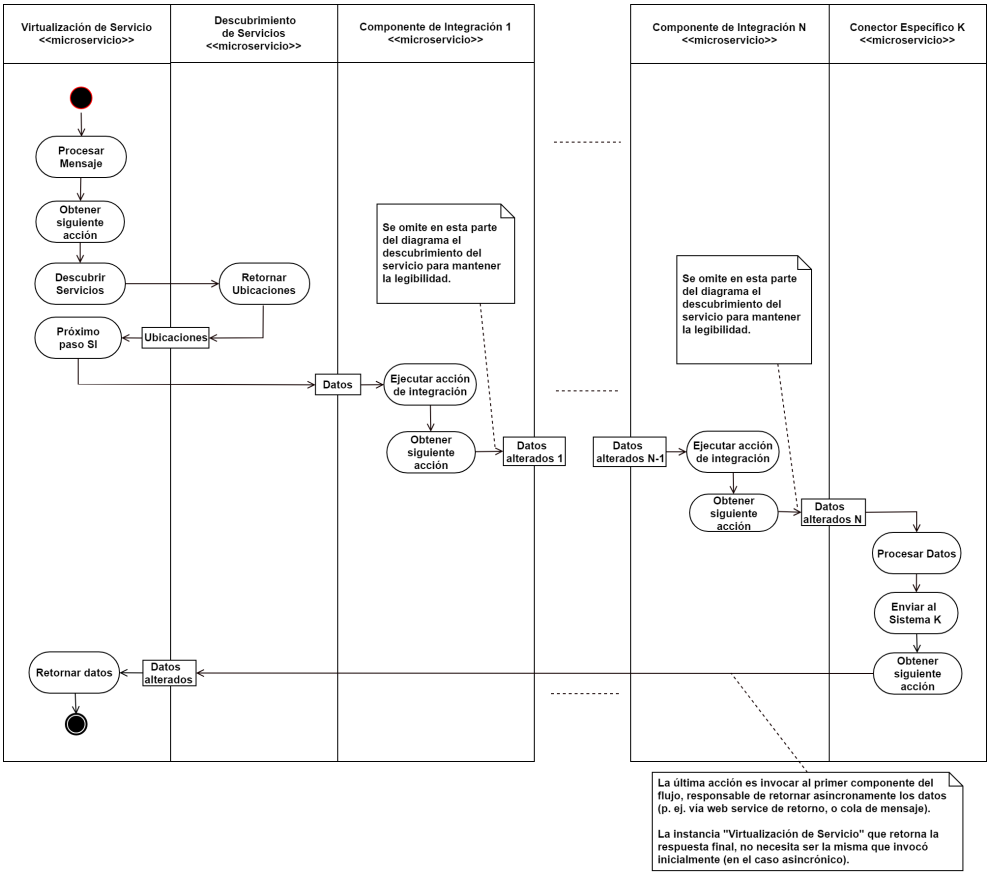

Figura 4.11: Vista de Procesos - SI invocada por servicio web (*request-response*)

El *Conector Específico K* recibe una respuesta del Sistema K que debe ser retornada a través del componente de *Virtualización de Servicio*[9]. Se plantea en la Figura 4.11 un enfoque asincrónico para retornar los datos integrados. Esto puede lograrse usando protocolos asíncronos

[9] El responsable de retornar la respuesta es el propio componente disparador, como forma de desacoplar a los demás componentes de implementar dicha responsabilidad.

como mensajería, o pasando como parámetro una ubicación de retorno, como ser un servicio web, una ruta de archivo o similar.

Se define que si la SI retorna un resultado a través del disparador, entonces la configuración de la SI agrega como acción final que el último componente retorne los datos integrados al primer componente (el disparador). No necesariamente se transmite la respuesta a la misma instancia de "Virtualización de Servicio" que inicio el flujo, ya que al ser asincrónica la respuesta no se mantiene la conexión de invocación inicial abierta (p. ej. de HTTP con el usuario).

Si la PI posee exigencias de retornar datos integrados en tiempo real, y se utilizase un enfoque sincrónico manteniendo la conexión del pedido inicial abierta esperando el resultado, se debe controlar expiraciones del pedido (i.e. *request timeout*), ya que el tiempo total de ejecución de la SI podría superar el tiempo de expiración del llamado inicial.

4.2.4.3 Gestión de Logs

Uno de los problemas a resolver por parte de la PI, es permitir la trazabilidad y supervisión de las soluciones de integración. De la tarea de recolección de logs, es responsable el componente *Gestor de Logs* descrito en la vista de desarrollo. Dicho componente recopila la información de ejecución que asiste a la supervisión de las SIs. En la Figura 4.12 se muestra el proceso.

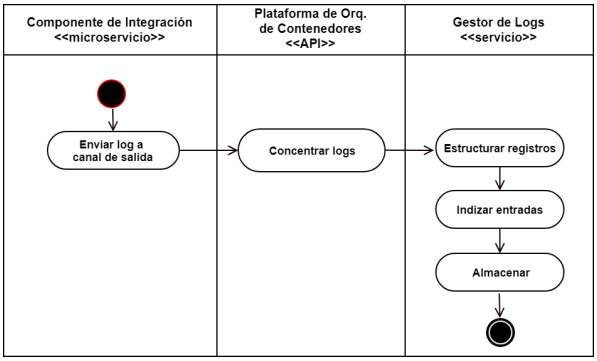

Figura 4.12: Vista de Procesos - Recolección de logs

Cuando la ejecución de la SI no es orquestada, la información de ejecución de la SI está dispersa y debe ser recolectada en cada componente de integración. Esto puede hacerse de varias formas, algunas de las cuales se muestran en la Sección 2.1.7.

En este trabajo, se propone apoyarse en funcionalidades de plataformas de orquestación de contenedores, las cuales colectan la información de los canales estándar del contenedor (i.e. *stderr*, *stdout*) y la concentran en archivos JSON o apoyándose en servicios complementarios

(ver Sección 2.1.7). Una vez concentrados los registros de ejecución el componente *Gestor de Logs* los estructura, indiza y almacena para su uso.

4.2.5 Vista de Despliegue

En la Figura 4.13 se presenta la vista de despliegue, la cual ilustra cómo son desplegados los componentes identificados en vistas anteriores y la correspondencia entre *software* y *hardware*.

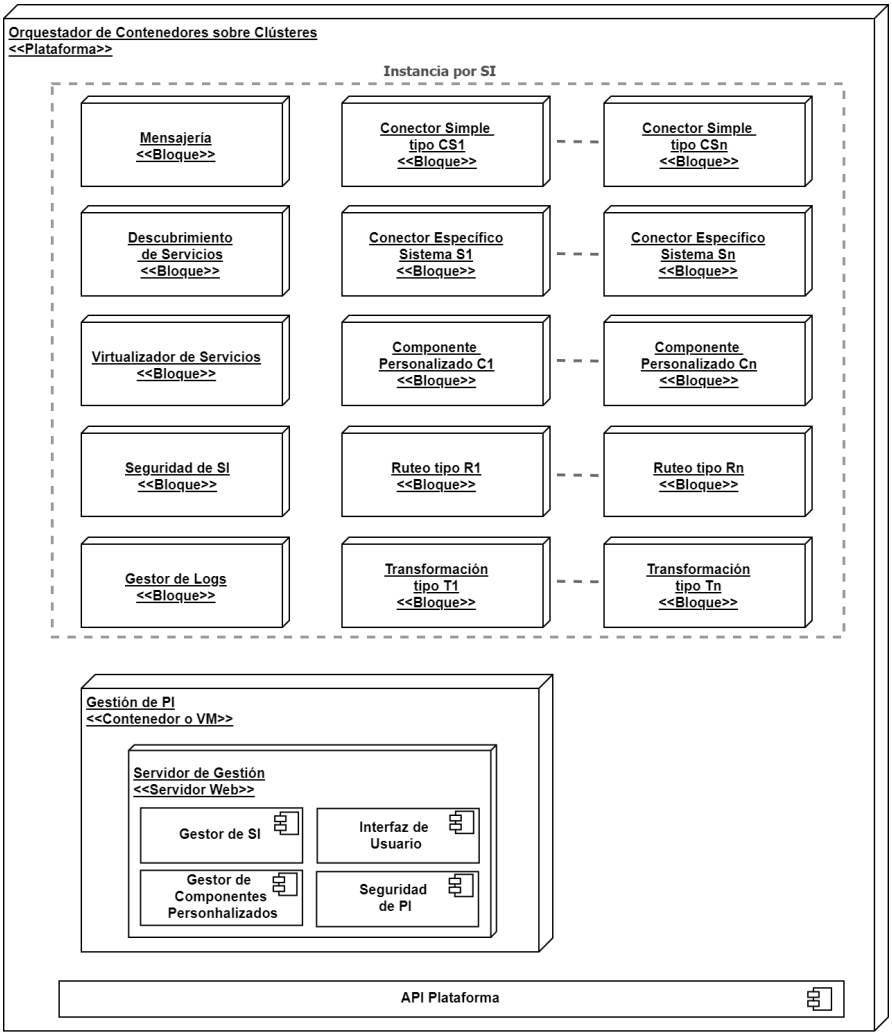

Figura 4.13: Diagrama principal de vista de despliegue

En la vista se propone que todos los componentes de la PI estén desplegados en una plataforma de orquestación de contenedores sobre clústeres, la cual se considera que está ubicada en una nube pública.

La correspondencia entre contenedores y los componentes identificados como servicio o microservicio es de uno a uno, siguiendo el patrón *un servicio por contenedor* de Richardson (ver Sección 2.1.8). Otra alternativa considerada para dicha decisión se discute en forma particular en la Sección 4.3.3.1.

El subsistema monolítico responsable por la administración de la PI, se asume que es desplegado en forma conjunta sobre un contenedor único o una VM. Se lo ubica sobre la misma plataforma de orquestación de contenedores sobre clústeres por practicidad, aunque podría de forma indiferente estar desplegado en otra infraestructura[10].

Por otro lado, en la Figura 4.13, se muestran delimitados por una línea punteada los componentes que se despliegan por cada SI y que operan de forma dedicada en la misma, es decir, que no son compartidos entre SIs.

Los componentes elegidos para operar en forma dedicada por SI, permiten cubrir casos de cantidad de pedidos y datos a integrar elevados. Si la PI se contextualiza en un escenario de baja cantidad de datos a integrar, el enfoque podría modificarse para compartir algunos componentes entre SIs y mejorar el uso de recursos. Los componentes que se propone compartir en dicho caso son: *Mensajería*, *Seguridad de SI* y *Gestor de Logs*. Compartir el componente *Descubrimiento de Servicios* debería ser analizado particularmente para evitar problemas de seguridad, especialmente en escenarios interorganizacionales, donde las SIs pueden ser construidas por distintas partes.

En otro aspecto, se observa que el diagrama tiene componentes desplegados etiquetados como "bloque". Esto refiere a que no consisten simplemente en el servicio o microservicio del componente, sino que son una serie de elementos desplegados como un bloque, lo cual puede observarse en detalle en la Figura 4.14.

Los "bloques", contienen un conjuntos de réplicas del componente original, cuyo número pueden aumentar o disminuir según la demanda sobre el componente. También incluyen un balanceador de carga, para balancear pedidos hacia las réplicas, una persistencia local centralizada, un módulo *Circuit Breaker* que asiste a la resiliencia, y un gestor de cache.

Por otro lado, se destaca que la plataformas de orquestación de contenedores usualmente ya proveen de forma nativa algunos utilitarios utilizados por la PI, como el *Balanceador de Carga*, *Gestor de Cache* y *Descubrimiento de Servicios*. Y proveen soporte a la gestión de los bloques como entidad (p. ej. *Pods* en OpenShift).

En referencia al despliegue en nube pública, si bien se considera para la propuesta principal, podría no ser adecuado para todos los escenarios. En particular es adecuado en casos de sistemas a integrar en la nube o aplicaciones SaaS a integrar (ver Apéndice D).

[10] La API de plataforma usualmente puede usarse de forma externa vía HTTPS y REST.

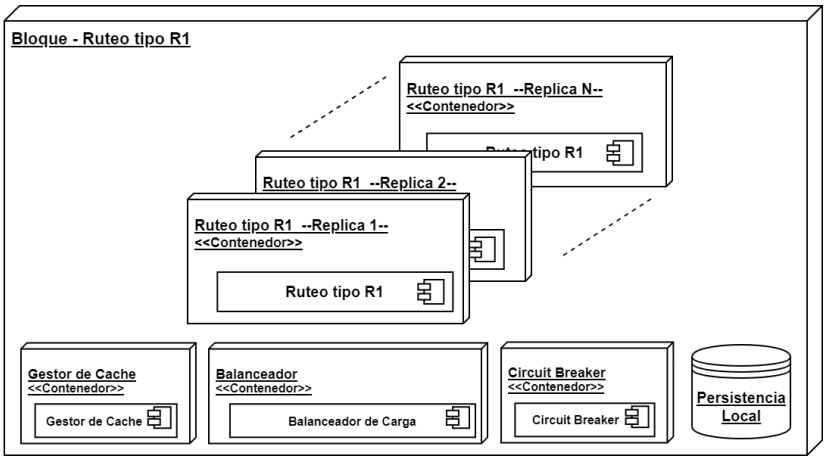

Figura 4.14: Bloque de despliegue para un componente (ejemplo Ruteo tipo R1)

Para determinar en un escenario la mejor ubicación física para los componentes de la PI, se deben considerar dos factores: los protocolos de comunicación soportados por los sistemas a integrar y la ubicación de las cantidades de datos más importantes (i.e. *Data Gravity*). Este tema se discute en el Apéndice D y variantes relacionadas se muestran en la Sección 4.3.2.2.

4.3 Variantes Alternativas

En esta sección se comentan algunas variantes alternativas que se consideraron durante el análisis de la propuesta principal. En la Sección 4.3.1 se muestra una variante que utiliza orquestación en vez de coreografía para la coordinación de microservicios, en la Sección 4.3.2 se muestran variantes que aplican en contextos particulares, mientras que en la Sección 4.3.3 se muestran otras consideraciones destacables analizadas.

4.3.1 Variante de Orquestación

En esta sección se presenta una variante que consiste en cambiar la forma de coordinación de los microservicios, para basarla en orquestación en vez de coreografía. Concretamente la variante introduce un nuevo componente, el orquestador, y modifica la ejecución y despliegue de las SIs. Se muestran en esta sección las vistas más representativas de la diferencia con el enfoque coreográfico. Las demás vistas se muestran en el Apéndice F.

4.3.1.1 Vista Lógica (desglose)

La principal diferencia en el desglose de la Vista Lógica con respecto a la propuesta principal, se produce en el subsistema *Gestión de SI*. En la Figura 4.15 se muestra el desglose de dicho subsistema, donde se observa una nueva entidad, el *Orquestador de SI*. El desglose de los demás subsistemas es análogo a la propuesta principal y no se muestra en la figura.

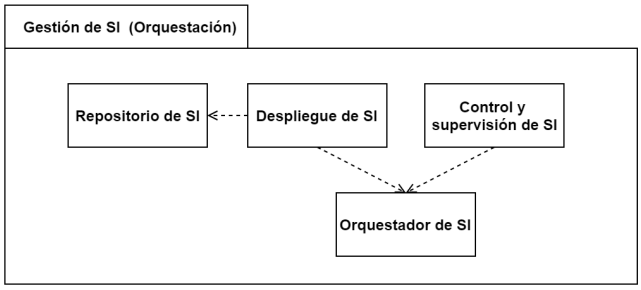

Figura 4.15: Desglose del subsistema *Gestión de SI* en orquestación

Se observa que el componente *Orquestador de SI* es requerido por los componentes *Despliegue de SI* y *Control y supervisión de SI*. El primero, lo requiere debido a que durante la puesta en marcha de SIs se necesita transmitir la información de ejecución al orquestador, de forma que este pueda gestionar la ejecución. El segundo, lo requiere debido a que el orquestador constituye un punto de concentración de los registros de ejecución, otorgando información para asistir la supervisión de SIs.

4.3.1.2 Vista de Procesos

Se muestra en esta vista cómo el enfoque de orquestación modifica la forma de ejecución y despliegue de SIs. El caso de gestión de logs no fue diagramado por considerarse análogo al de coreografía, con la diferencia de ser el *Orquestador de SI* quien provee los registros de ejecución al *Gestor de Logs*.

Ejecución de SI

En la Figura 4.16 se presenta un diagrama de actividad para la ejecución orquestada de una SI representativa. En ella, datos de un *Sistema J* son sometidos a una serie de modificaciones, para luego ser enviados al *Sistema K*, a través del conector correspondiente.

En el diagrama se representaron los componentes de integración en una única columna para mayor claridad, ya que los pasos efectuados por dichos componentes son análogos. Una representación literal mostraría columnas por cada uno de los múltiples componentes, con las mismas acciones.

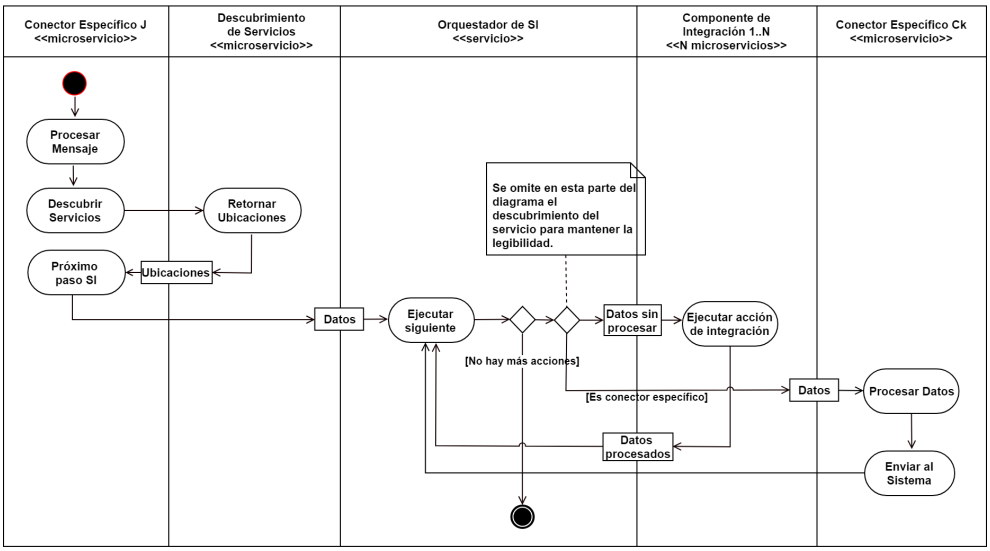

Figura 4.16: Vista de procesos, ejecución de SI orquestada

Se observa que el orquestador invoca a cada uno de los componentes de integración y recolecta sus resultados. Una vez ejecutados todos los pasos, el orquestador transmite al conector final los datos modificados. Se asume una comunicación asincrónica mediante HTTP, para omitir interacciones con el componente de mensajería y dar mayor claridad al diagrama.

Se observa que si la SI debiese retornar un resultado (análogo al caso visto en la Figura 4.11), bastaría con modificar la solución anterior para que el orquestador retorne un resultado al disparador de la SI cuando finalice las acciones a ejecutar.

Despliegue de SI

En la Figura 4.17 se muestra un despliegue de SI en el caso de orquestación. Como principal diferencia con la propuesta coreográfica, se observa que la información de ejecución es trasmitida al *Orquestador de SI*, el cuál luego se encarga de activar la SI mediante la activación de los elementos disparadores. En el ejemplo de la Figura 4.17, el disparador es el *Componente de Integración 1*.

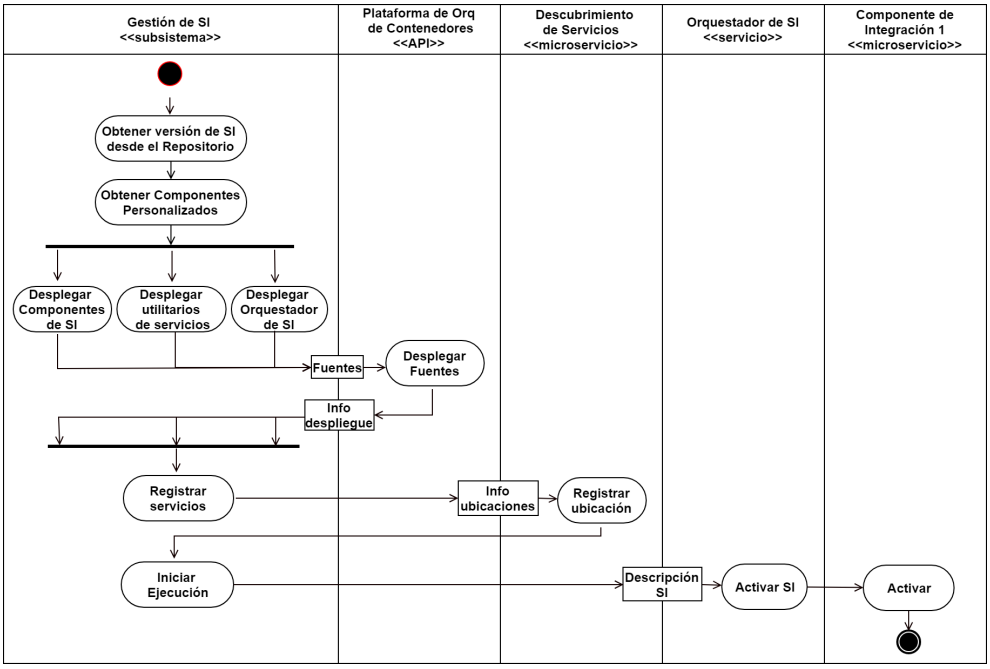

Figura 4.17: Vista de procesos, despliegue de SI en orquestación

Conclusiones de Variante de Orquestación

Se observó que el cambio principal del enfoque está en la introducción del componente orquestador y en algunas interacciones entre los componentes en los casos de ejecución y despliegue de SIs. Esto se debe a que el orquestador concentra la información de ejecución.

Si bien no hay consenso en las principales referencias de cuál es el enfoque de coordinación más indicado (ver Sección 2.1), la variante de orquestación podría ser utilizada en escenarios donde las SIs son extensas y de coordinación compleja, en consideración a lo planteado por Richardson en [19].

4.3.2 Variantes para Contextos Particulares

En esta sección, se presenta una variante que consiste en construir como microservicios solo los componentes de integración que más requieran las ventajas de microservicios, y agrupar en subsistemas monolíticos al resto. El objetivo es minimizar los problemas ocasionados por la distribución (ver Sección 2.1.4). La variante requiere información del escenario concreto para determinar cuáles componentes son definidos como microservicios. Se describen a continuación las vistas modificadas, utilizando como base los escenarios de la Sección 3.6.

4.3.2.1 Vista de Desarrollo

Ejemplo 1: Conectores Específicos como Microservicios

El escenario 1 descrito en la Sección 3.6.1 plantea un contexto de alta disponibilidad donde se requiere integrar múltiples sistemas en la nube, los cuales cambian frecuentemente. En dicho caso, la mantenibilidad sobre los conectores específicos es importante, ya que al ser cambiantes los sistemas aumenta la posibilidad de que cambie también la forma de conectarlos. Por otro lado, la alta disponibilidad planteada en el escenario requiere de mantenimiento sin tiempos de baja.

En el escenario comentado es apropiado construir los conectores específicos como microservicios, además de la virtualización de servicios, tal como se muestra en la Figura 4.18.

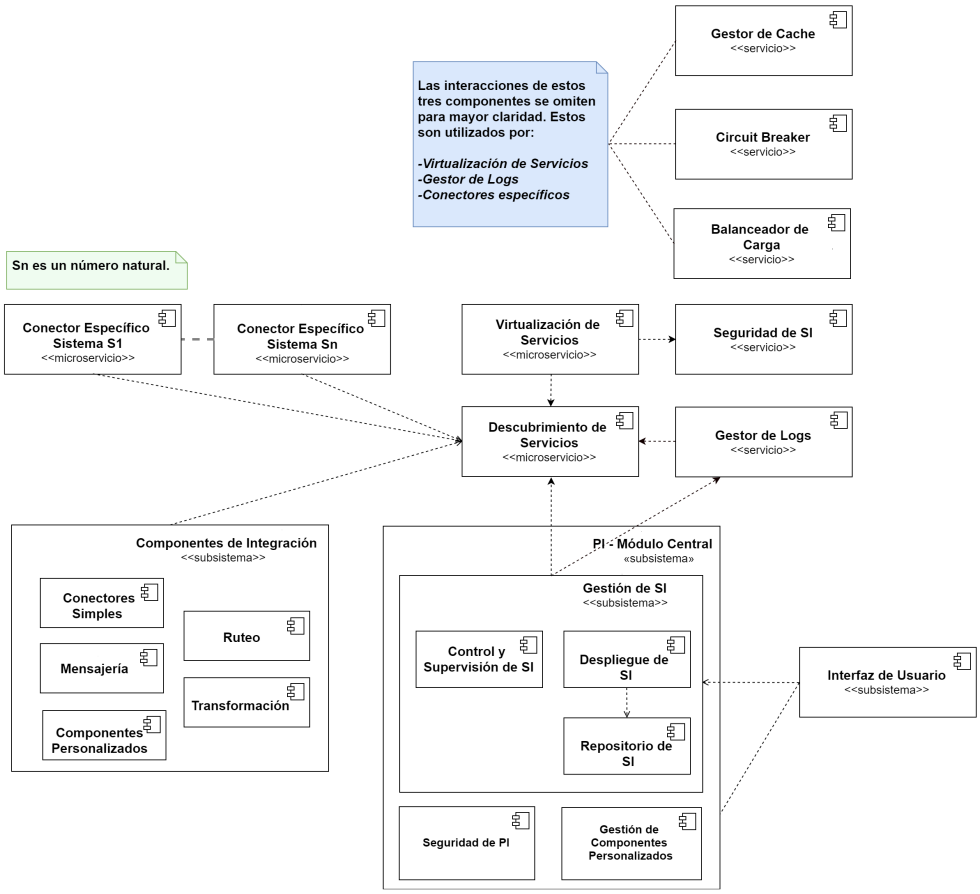

Figura 4.18: Conectores específicos como microservicios

Se destaca del diagrama que los componentes de integración (exceptuando el conector específico) se agruparon en un subsistema monolítico. Este subsistema se separó del módulo central de PI, ya que es de interés que posea recursos de hardware dedicados, acorde a sus tareas de integración.

Esta variante también es aplicable al escenario 2, descrito en la Sección 3.6.2. En dicho caso, cada conector específico podría separarse en dos componentes: uno desplegado en la red interna, conectando directamente con los sistemas a integrar; y otro en la nube, recibiendo los datos del primero y activando la SI.

Ejemplo 2: Transformación y Conectores como Microservicios

En el escenario 3 descrito en la Sección 3.6.3, los componentes de transformación necesitan implementarse utilizando tecnología específica, y los sistemas a integrar cambian frecuentemente.

En dicho contexto es apropiado construir las transformaciones, los conectores específicos y la virtualización de servicio como microservicios. El diagrama de arquitectura es análogo al mostrado en la Figura 4.18 pero definiendo *Transformaciones T1* a *Tn* como microservicios, análogo a lo hecho con transformaciones en la propuesta principal.

Se observa que con esta propuesta, las transformaciones pueden ser implementadas usando tecnología específica. Por ejemplo, pueden utilizar el lenguaje *C++* e incluir bibliotecas como *GSL*, mientras que el resto de la PI puede implementarse usando otro lenguaje.

4.3.2.2 Vista de Despliegue

Si bien en la propuesta principal se despliegan todos los componentes en una nube pública, esto puede no ser lo ideal en todos los escenarios. Para determinar la mejor ubicación de los componentes de integración, se deben considerar dos factores: i) la comunicación con los sistemas a integrar, y ii) la ubicación de las cantidades de datos más importantes (i.e. *Data Gravity*).

Por un lado, el protocolo de comunicación de los sistemas a integrar puede limitar el despliegue de los conectores. Si los sistemas son internos y no disponen de protocolos para comunicarse por Internet, o si existen limitaciones empresariales para exponerlos, entonces los conectores deben ser desplegados en la misma red interna. Esta variante se ilustra en la Figura 4.19.

Lo ideal es que los conectores sean desplegados también en una plataforma de orquestación de contenedores en la red interna. En la Figura 4.19 se asume un servidor web ordinario (p. ej. *Wildfly*[11]) por simplicidad.

Por otro lado, las ubicaciones de las concentraciones principales de datos (i.e. *Data Gravity*)

[11] http://www.wildfly.org/

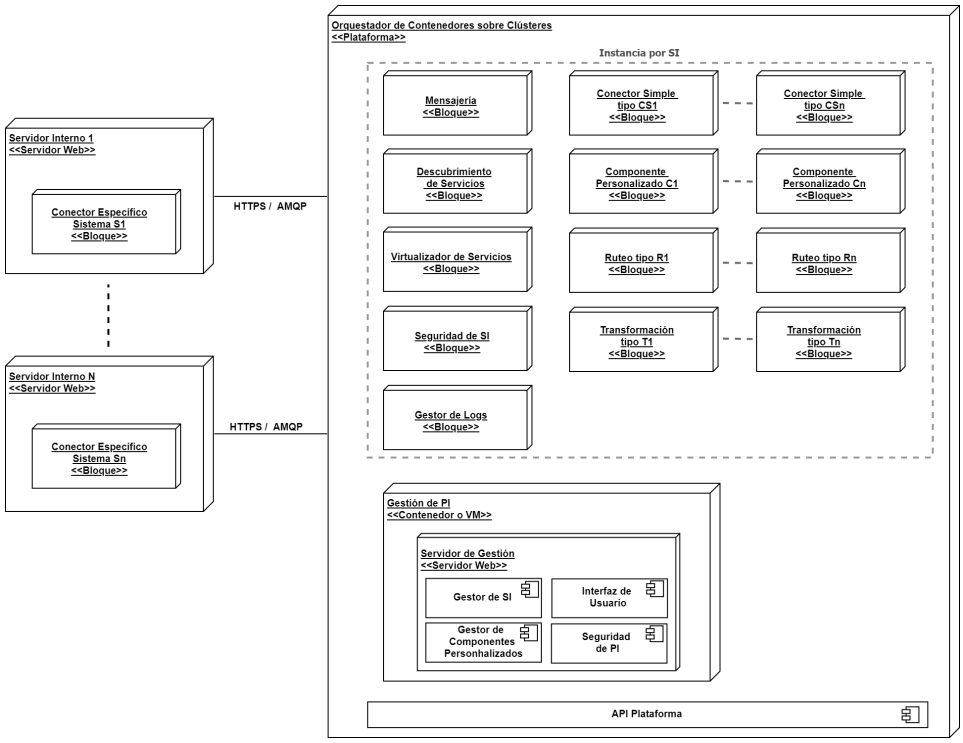

Figura 4.19: Conectores específicos desplegados en forma separada.

deben ser tenidas en cuenta para la ubicación de los componentes de la SI. Esto aplica cuando la PI y los sistemas a integrar no están desplegados en la misma red. En dicho escenario, es estratégico desplegar los componentes de integración cerca de los datos que deben procesar.

Se explica el concepto anterior con el siguiente ejemplo. En la Figura 4.2, dos sistemas envían datos que son transformados, filtrados y separados antes de recibirse en el sistema destino. Si el componente filtro descarta la mayoría de los datos[12], entonces solo unos pocos continúan el flujo, por lo cual las transformación y el filtro concentran la mayoría del procesamiento de la SI. En dicho escenario, el despliegue debería tener los conectores específicos y de transformación desplegados cerca de los sistemas que envían datos (*C2* y *C3*), de forma de reducir la latencia en el punto donde más datos son procesados.

Un diagrama de despliegue para el ejemplo anterior, sería análogo al mostrado en la Figura 4.19 pero adicionando los componentes de transformación de modelos y filtro al servidor interno.

[12] Un ejemplo de este escenario es la detección de fraude, donde muy pocos datos cumplen la condición de filtro.

4.3.2.3 Conclusiones de Variantes de Contexto

Se concluye que si se posee información del escenario puede utilizarse para definir dos aspectos: i) cuáles componentes se construyen como microservicios, y ii) la ubicación de despliegue más apropiada para los componentes. El enfoque puede aplicarse como forma de reducir la latencia o la complejidad de la PI, en escenarios donde existen dichos problemas.

Por otro lado, se observa que el enfoque de esta variante es el seguido por algunos artículos académicos mostrados en la Sección 2.6.1. Por ejemplo, en los artículos con escenarios de *IoT* se construyen solo los conectores de dispositivos como microservicios. Por otro lado, las propuestas de iPaaS mostradas en 2.6.2 se entiende que siguen un enfoque de microservicios más amplio como el mostrado en la propuesta principal.

4.3.3 Otras Consideraciones

4.3.3.1 SI en un Único Contenedor (Vista Despliegue)

Una variante que se consideró durante el análisis es desplegar cada SI en forma completa en un contenedor, es decir, incluir todos los componentes de una SI en el mismo contenedor como se muestra en la Figura 4.20 [13]. De esta forma se gestiona y escala a la SI en forma conjunta. Sin embargo, se observaron a nivel general problemas relacionados a la arquitectura monolítica, debido a la gestión orientada a contenedores que poseen las plataformas planteadas como infraestructura (i.e. plataformas de orquestación de contenedores).

Se observa que la solución no respeta el patrón de Richardson "*Instancia de servicio por contenedor*" [20]. Si bien Richardson también comenta un posible patrón llamado "*múltiples servicios por contenedor*", el cual podría ajustarse a esta variante, menciona varios problemas en él, entre los cuales se destacan los problemas de recursos compartidos, incapacidad de aislamiento, riesgo de conflictos de dependencias y dificultad de la supervisión [20].

Por un lado, es destacable que la variante posee un peor uso de recursos que la propuesta principal en casos de escalamiento automatizado, ya que debe replicar todos los componentes de la SI al escalar horizontalmente. Esto se debe a que en la infraestructura considerada[14] la automatización del escalamiento se realiza en base a la gestión de contenedores, y por tanto, es el contenedor el que se replica.

Por otro lado, en la variante todos los componentes de la SI compiten por los mismos recursos (los asignados al contenedor). De esta forma si algunos componentes de integración efectúan un procesamiento demandante, afectan a la SI en forma completa. Lo anterior también impide lograr un efecto de segmentación (*pipeline*) al procesar los datos, y ocasiona que si alguno de los componentes requiere temporalmente muchos recursos, nuevos datos a

[13] En la Figura 4.20 se muestra solamente el diagrama de bloque y no la PI entera.
[14] En referencia a las plataformas de orquestación de contenedores.

integrar provocarán escalar la SI entera.

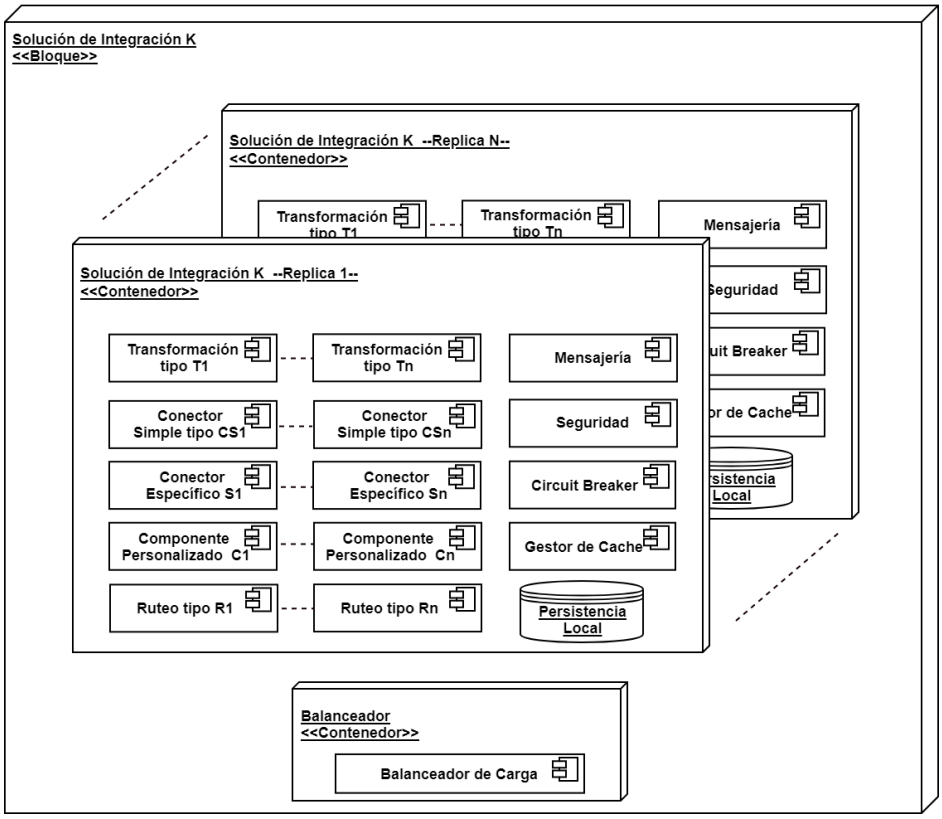

Figura 4.20: Vista de Despliegue - SI desplegada de forma conjunta en un contenedor

Se observa también que si el contenedor falla, provoca la falla de la SI completa. Por otro lado, si falla la SI en un componente puntual, no pueden aprovecharse las ventajas de recuperabilidad de reiniciar el contenedor sin perder el procesamiento hecho por la SI entera hasta el momento de falla.

También es destacable que el cambio y mantenimiento sobre componentes de integración afecta a toda la SI, la cual debe volver a desplegarse.

Como ventaja se observa que la latencia entre la comunicación de los componentes es menor a la propuesta principal. Sin embargo, la diferencia no es significativa dado que en la propuesta principal los componentes ya se encuentran en la misma red. Se observa que si se considera la latencia como la principal ventaja del enfoque, entonces es preferible un enfoque monolítico, que posee desventajas similares a la variante pero mejor impacto en la latencia por ser invocaciones en memoria.

Se observa que problemas análogos surgen al distribuir más de un microservicio por contenedor. En base a lo anterior, se descarta la variante y se determina que la correspondencia ideal entre microservicio o servicio y contenedor debería alinearse a lo definido por el patrón *Instancia de servicio por contenedor*.

4.3.3.2 Granularidad de Microservicios (Vista Desarrollo)

Una decisión que se consideró durante el análisis en la vista de desarrollo, fue la de no desglosar los componentes de integración del trabajo de González y Ruggia. Es decir, proponer por ejemplo un único componente "*Ruteo*" y no múltiples componentes de ruteo según su tipo (i.e. *Ruteo tipo R1* a *Rn*).

Sin embargo, se entendió que de esa forma el tamaño de los componentes de integración no se ajustaba a los parámetros de microservicios (p. ej. que se pueda reescribir en dos semanas) y por tanto dicha arquitectura se asemejaba más a una arquitectura orientada a servicios y no a una de microservicios. Además se observó también un mejor impacto sobre los atributos de calidad de la PI al desglosar los componentes de integración, lo cuál se describe en detalle en la Sección 5.1.

4.4 Conclusiones

En este capítulo se presentaron propuestas de arquitectura de microservicios para Plataformas de Integración. En primer lugar, se describió una propuesta principal correspondiente a una PI de propósito general basada en coreografía, la cual es genérica para los escenarios de integración. Luego se plantearon variantes alternativas. Por un lado se propuso una variante de orquestación, la cual difiere en el enfoque de coordinación de los microservicios. Por otro lado, se presentaron variantes aplicables a escenarios particulares. Por último, se comentaron otras consideraciones que se tuvieron durante el análisis, como el incluir las SIs en forma íntegra en un contenedor.

En las propuestas se introdujeron componentes propios de las arquitecturas de microservicios, como *Circuit Breaker*, *Descubrimiento de Servicios*, *Balanceador de Carga*, *Gestor de Caché* y *Gestor de Logs*. Exceptuando el componente *Circuit Breaker*, los demás componentes usualmente existen también en PIs monolíticas, pero su responsabilidad y forma de uso difieren a las de arquitecturas de microservicios. Por ejemplo, la tarea del *Gestor de Logs* es más compleja en el caso de microservicios, ya que los registros de ejecución están dispersos y deben ser recolectados.

En la propuesta principal, el desglose realizado se basó en patrones y buenas prácticas propuestas por Richardson en [20]. Además, se planteó un enfoque coreográfico y se describió la resolución de algunos problemas generales, como la ejecución y despliegue de SIs, y la gestión de registros de ejecución (*logs*). Por otra parte, se propuso un despliegue basado en

el concepto de "Bloque" por servicio, el cual favorece el escalamiento independiente apoyándose en plataformas de orquestación de contenedores.

En relación a la variante de orquestación, se observó que si bien no hay consenso en las referencias principales respecto al enfoque más indicado, la variante podría aplicar en escenarios donde las SIs son extensas y de coordinación compleja.

Por último, se observó que las variantes de contextos particulares podrían ser más apropiadas que la propuesta principal para algunos escenarios de integración. Sin embargo, las mismas requieren información específica del escenario en forma previa a la construcción de la PI. Algunos ejemplos del enfoque son los artículos académicos mostrados en la Sección 2.6.1.

5 Evaluación y Experimentación

En este capítulo se evalúa la propuesta principal de arquitectura del Capítulo 4 en base a tres factores: i) el impacto de algunas decisiones de diseño sobre los atributos de calidad de la PI, ii) el cumplimiento de patrones y buenas prácticas de microservicios, y iii) la construcción de un prototipo.

El capítulo se organiza de la siguiente manera. En la Sección 5.1 se analiza el impacto de algunas decisiones de diseño sobre atributos de calidad de la PI. En la Sección 5.2 se analiza el cumplimiento de patrones y buenas prácticas de microservicios. Finalmente en la Sección 5.3 se estudia la factibilidad técnica de la propuesta principal, en base a la construcción de un prototipo.

5.1 Evaluación de Decisiones de Diseño Relevantes

Dado que el impacto de una arquitectura de microservicios sobre PIs, ya fue analizado en el Capítulo 3, esta sección, se enfoca particularmente en analizar el impacto de algunas decisiones de diseño tomadas en la propuesta principal que se consideraron de interés. Se listan las mismas a continuación detallando como afectan la calidad de la PI.

- **Bajo nivel de abstracción en componentes**: En la vista de desarrollo se desglosan los componentes de la Sección 4.1.1.2 en múltiples componentes más específicos. Por ejemplo, el componente *Ruteo* se subdivide en diferentes tipos de ruteo: *Ruteo de tipo R1* a *Rn*.

  La alternativa de utilizar el mismo nivel de abstracción de la Sección 4.1.1.2 fue descartada por motivos que se discuten en la Sección 4.3.3 y también por el impacto sobre los atributos analizado en esta sección.

Desglosar los componentes en componentes más específicos, tal como se efectúa en la propuesta principal, posee el siguiente impacto a nivel de subatributos de calidad:

- *Mantenibilidad*: Es afectada positivamente, debido a que cambios en un componente tienen menor impacto y son más fáciles de ejecutar por la modularidad planteada. Por ejemplo, se modifica *Ruteo tipo R2* y no a todo el componente *Ruteo*.
- *Uso de recursos*: Es afectada positivamente, debido a que una mayor granularidad permite que se despliegue, cargue en memoria y ejecute solamente el código necesario. Por ejemplo, se despliega *Ruteo tipo R2* y no a todo el componente *Ruteo*.
- *Idoneidad reconocible*: Es afectada positivamente, ya que debido a la granularidad en la interfaz[1], el usuario puede asociar de forma más directa los componentes disponibles, a la acción de integración que quiere ejecutar.
- *Protección a errores de Usuario*: Es afectada positivamente, ya que si los componentes son más específicos, poseen menos configuración, lo cual disminuye la complejidad al usuario y los errores que pueda cometer. Por ejemplo, se ofrece un componente *Transformación de Filtro* en vez de que el usuario deba configurar un componente *Transformación* para que aplique una transformación de filtro.
- *Operatividad y estética*: Se observa que se deben tener consideraciones para que la cantidad de componentes de integración no sea tan grande que afecte negativamente el subatributo. Esto se realiza definiendo un número acotado de componentes según el perfil técnico del usuario. En la Sección 4.2.3 se presentan propuestas al respecto.

- **Coreografía u Orquestación**: En la propuesta principal, se utiliza coreografía como forma de ejecución de las SIs. Esta decisión, contrasta con la variante de ejecución orquestada (ver Sección 4.3.1) y es discutida en la Sección 2.1, donde se afirma que no existe consenso al respecto en las referencias principales de microservicios.

  De acuerdo a la bibliografía la diferencia de impacto no debería ser considerable, aún así, se entiende que la diferencia del impacto sobre los atributos es la siguiente:

  - *Uso de recursos*: Es afectado positivamente en el enfoque coreográfico, en comparación con orquestación. Por un lado, en el caso de orquestación hay un componente más, el orquestador, que consume recursos. Por otro lado, la respuesta de un componente de integración es retornada al orquestador, antes de que este invoque al siguiente componente. Por tanto el orquestador al actuar de intermediario agrega más comunicaciones que consumen recursos, en comparación con coreografía.

[1] En las propuestas se menciona que la asociación de microservicio a componente elegible por el usuario es 1 a 1.

  - *Comportamiento del tiempo*: Podría afectar positivamente. Si bien no se efectuaron estudios de desempeño se observa que la existencia de un intermediario en orquestación implica una comunicación adicional (frente al caso coreográfico). Por tanto los tiempos de respuesta podrían ser levemente más altos que coreografía.

  - *Tolerancia a fallas*: El enfoque coreográfico posee una mejor tolerancia a fallas que orquestación, ya que el orquestador podría conformar un único punto de falla.

- **Plataforma de orquestación de contenedores**: Otra decisión de diseño fue el despliegue sobre una plataforma de orquestación de contenedores sobre clústeres. Esto optimiza el impacto de la arquitectura de microservicios sobre seis subatributos de calidad: *Uso de Recursos*, *Adaptabilidad*, *Instalabilidad*, *Recuperabilidad* y *Seguridad*, tal como se explica en la Sección 3.4.

- **Variantes para contextos particulares**: Se compara en base a los atributos de calidad, la propuesta principal y la variante de la Sección 4.3.2, la cual propone utilizar información del escenario para definir qué componentes son construidos como microservicios. El impacto sobre los subatributos, posee puntos en común con el análisis del impacto de microservicios visto en el Capítulo 3:

  - *Uso de Recursos*: En contextos donde se requiere escalamiento depende de si los componentes que requieren escalamiento se definieron como microservicios. Si no es así, es afectado negativamente en la variante de contextos particulares, debido a que la mayor parte de la PI es monolítica lo cual implica un peor uso de recursos al escalar horizontalmente, como se detalla en la Sección 3.4. Si el escalamiento no es requerido o recae solamente en los componentes definidos como microservicios, la variante podría afectar positivamente en comparación con la propuesta principal.

  - *Comportamiento del Tiempo*: La variante podría afectar positivamente, ya que al reducirse el número de microservicios, se reducen los posibles efectos negativos vistos en la Sección 3.4.

  - *Instalabilidad*: No hay un impacto considerable sobre el subatributo. El desplegar gran parte de la PI como un único subsistema, podría reducir la complejidad de la instalabilidad; sin embargo, la automatización de despliegue de microservicios en ciertas plataformas ya eliminan buena parte de dicha complejidad (ver Sección 3.4).

  - *Mantenibilidad*: Es afectada negativamente en la variante, debido a que la mayor parte de la PI es monolítica, cuya arquitectura no posee la ventajas de mantenibilidad de microservicios. Solo se posee buena mantenibilidad sobre los componentes definidos como microservicios, lo cual podría ser suficiente según el contexto.

– *Adaptabilidad*: Es afectada negativamente en la variante, debido a que la mayor parte de la PI es monolítica, cuya arquitectura no posee la ventajas de escalabilidad de microservicios. Solo se posee escalabilidad eficiente sobre los componentes definidos como microservicios, lo cual podría ser suficiente según el contexto.

Se observa que el impacto es un balance en base al contexto de las ventajas y desventajas de la arquitectura de microservicios. Análogo a como se mencionó para el componente de gestión de la PI en la Sección 4.2.3, depende de los requerimientos sobre los componentes.

Se concluye que la variante es útil en contextos particulares, cuando unos pocos componentes corresponden a la parte más importante de las soluciones de integración y los demás componentes no requieren las ventajas de microservicios.

5.2 Evaluación en base a Patrones de Microservicios

Esta sección analiza el cumplimiento por parte de la propuesta principal del Capítulo 4, de patrones para microservicios sugeridos por Richardson en [20] y vistos en la Sección 2.1.8.

- *Descomposición por capacidad de negocio*: La propuesta considera las capacidades que deben tener las soluciones de integración (p. ej. transformación, ruteo) y luego define los microservicios en base a esto. Por tanto, se puede afirmar que cumple con el patrón planteado.
- *API Gateway*: El componente *Virtualizador de servicios* actúa como frontera de los microservicios exponiendo y centralizando invocaciones externas, de forma análoga a la descrita por Richardson en el patrón *API Gateway*[2].
- *Una instancia de servicio por contenedor*: Cada microservicio en las propuestas es desplegado en su propio contenedor (ver Sección 4.2.5). Por tanto se cumple con el patrón planteado.
- *Interruptor de circuito*: La arquitectura incluye expresamente el componente de Interruptor de Circuito (*Circuit Breaker*), como puede observarse en las vistas lógica, de desarrollo y de despliegue. Por tanto cumple con dicho patrón.
- *API de chequeo de salud*: El patrón API de chequeo de salud (*Health Check API*), se sugiere expresamente en la arquitectura propuesta y su gestión está dada por el uso de plataformas de orquestación de contenedores sobre clústeres.
- *Configuración exteriorizada*: La configuración de los componentes de integración, no

[2] Los productos de API Gateway en la industria poseen generalmente otras funcionalidades como seguridad, supervisión, entre otros. Sin embargo no están definidas por Richardson en el patrón.

está contenida en los propios microservicios, sino que es traspasada a los mismos por el componente *Despliegue de SI* en tiempo de ejecución. Dicha configuración a su vez puede almacenarse y gestionarse con el manejo de configuraciones que otorgan las plataformas de orquestación de contenedores (p. ej. *Kubernetes Secrets*[3]). Por tanto, se considera que se cumple con el patrón.

- *Una base de datos por servicio*: En la vista de despliegue (Figura 4.14) puede observarse que se utiliza una base de datos para cada microservicio. Por tanto se cumple con el patrón planteado.

5.3 Construcción de un Prototipo

Se implementó un prototipo con el objetivo de validar la factibilidad técnica de la propuesta principal de arquitectura del Capítulo 4. La prueba de concepto se enfoca en la ejecución de una solución de integración, y se basa en el ejemplo conceptual descrito en la Sección 4.1.1.3. El ejemplo se muestra esquematizado en la Figura 5.1. El prototipo fue construido y desplegado sobre OpenShift Origin[4] y se encuentra disponible *online*[5].

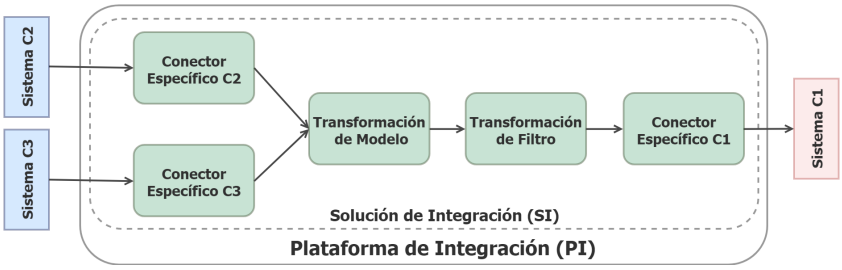

Figura 5.1: Ejemplo conceptual, usado como base para la construcción del prototipo.

5.3.1 Tecnología Utilizada

Se describe brevemente en esta sección las principales tecnologías utilizadas para el prototipo.

- *Docker*: Es una tecnología de contenerización que permite facilitar la creación y gestión de contenedores (ver Sección 2.2). Es utilizado para la operación de los contenedores que empaquetan a los componentes desplegados.

- *OpenShift Origin*: Consiste en una plataforma de orquestación de contenedores (ver Sección 2.2.2). Esta es montada sobre un sistema *unix* y utilizada como plataforma

[3] https://kubernetes.io/docs/concepts/configuration/secret/
[4] https://www.openshift.org/
[5] https://gitlab.fing.edu.uy/open-lins/micro-ip

de base sobre la cual se despliegan los componentes. Permite la gestión, ejecución y supervisión de contenedores, aportando además servicios de infraestructura (p. ej. conectividad entre contenedores, escalamiento), entre otros.

- *Java y Maven*: Java es un lenguaje de propósito general, mientras que Maven es una herramienta para gestionar proyectos de software que aporta gestión de dependencias (entre otros). Es utilizada para construir los componentes y administrar sus dependencias.
- *JBoss Wildfly*: Es un servidor web liviano, se utiliza para exponer los microservicios de los componentes de integración.
- *RabbitMQ*: Es un software que opera como intermediario de mensajería y que aporta operatividad y gestión de colas de mensajes. Se utiliza para implementar el intercambio de mensajes entre componentes.

5.3.2 Diseño de la Solución

El prototipo construido consiste en una SI basada en el ejemplo diagramado en la Figura 5.1. La SI fue prediseñada y desplegada en forma manual como simplificación, aunque se observa que si se requiriese implementar la PI completa, la plataforma Openshift utilizada posee una *API REST* [76] para automatizar el despliegue de componentes.

Cada componente de integración de la SI es un microservicio construido con *Java* y *Maven*, desplegado sobre un servidor *JBoss WildFly* e incluido en un contenedor *Docker*. La imagen de contenedor se genera automáticamente a partir de un vínculo al repositorio y la funcionalidad nativa *S2I* (*Source-to-image*) de OpenShift.

Los componentes de la SI se comunican entre sí utilizando coreografía asíncrona basada en mensajería con *RabbitMQ*. Cada componente determina el próximo paso de la SI en base a su configuración inicial. Esta configuración se apoya en la funcionalidad de *mapas de configuración* de OpenShift. En la arquitectura diseñada, la configuración inicial es realizada por el componente de *Despliegue de SI*, lo cual podría implementarse utilizando la API de OpenShift pero es omitido por simplificación en el prototipo.

En la Figura 5.2 se muestra un ejemplo de la configuración descrita anteriormente. El componente filtro posee un mapa de configuración con dos valores: i) transformación concreta a realizar[6] (descartar ordenes cuyo valor no es mayor a cero) y ii) el siguiente paso a invocar (conector específico 1).

Durante la implementación, se observó que algunos utilitarios definidos en la arquitectura de PI de este trabajo, ya son proporcionados por OpenShift. Entre ellos, el descubrimiento de servicios, balanceo de carga y gestión de logs[7]. En la Figura 5.3 puede observarse a modo de

[6] Se utilizó Jayway jsonpath (*https://github.com/json-path/JsonPath*) para definir el lenguaje de filtro sobre JSON.
[7] La gestión de logs es dada de forma parcial (i.e. recoge los registros pero no unifica los de todos los componentes

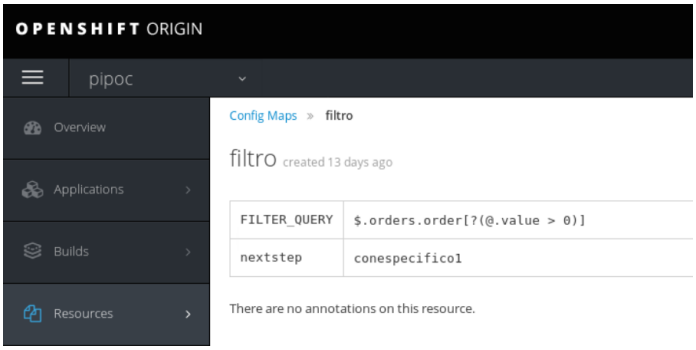

Figura 5.2: Ejemplo de configuración: Transformación de Filtro.

ejemplo, registros de ejecución recolectados por OpenShift desde el canal de salida estándar (i.e. *stdout*).

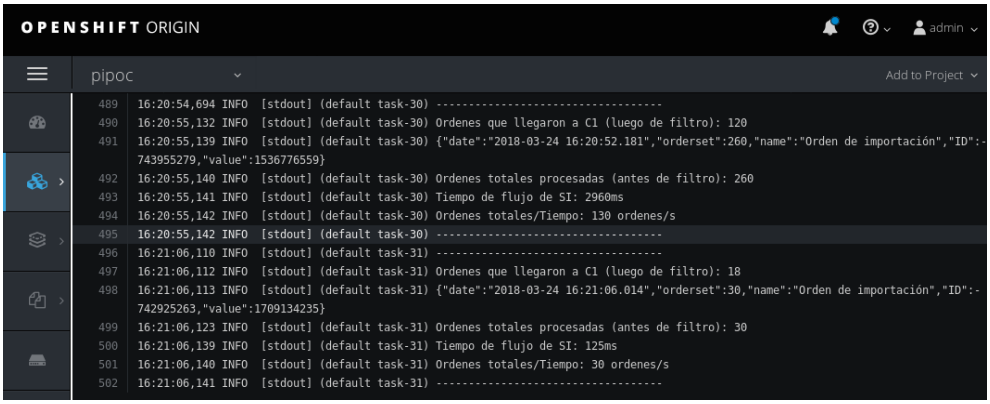

Figura 5.3: Registros de ejecución de la SI.

En la Figura 5.4, se observa un esquema representativo de los microservicios desplegados. Los componentes son empaquetados junto con todas sus dependencias (incluyendo servidor web) en una *Imagen Docker* a partir de la cual se instancia el contenedor. Cada instancia de contenedor es manejada como una unidad de ejecución denominada *Pod*, las cuales son escalables y organizadas en conjuntos de réplicas (*Replica Set*), con un balanceador de carga asignado por OpenShift. Por otro lado, puede observarse que por practicidad los sistemas a integrar también fueron desplegados en OpenShift.

En la Figura 5.5 se observa una pantalla principal mostrando el despliegue en *Openshift* de los conceptos anteriormente mencionados. Cada servicio está desplegado y agrupado como un conjunto de réplicas, que inicialmente posee sólo un *Pod*.

de integración).

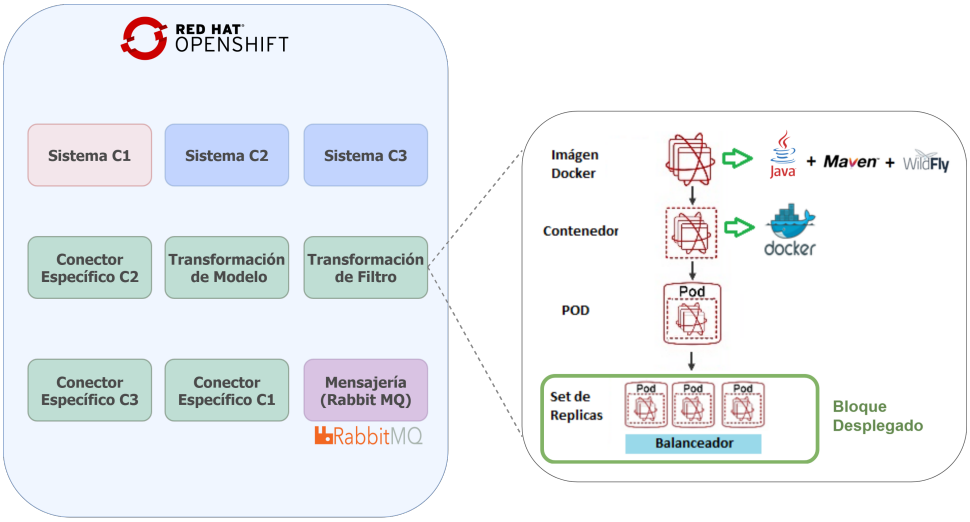

Figura 5.4: Componentes desplegados en OpenShift y tecnología utilizada.

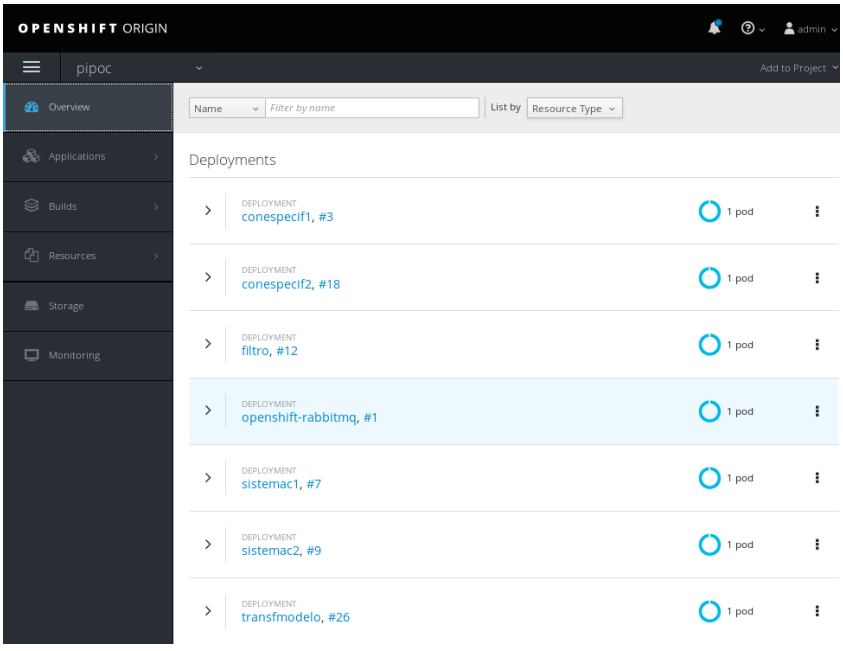

Figura 5.5: Componentes desplegados y en ejecución en OpenShift.

El escalamiento de los microservicios, puede efectuarse de forma manual o automática. Esta

última se realiza mediante la definición de reglas basadas en mediciones sobre el estado del contenedor, como ser memoria disponible, uso de procesador, entre otros. El escalamiento también puede ser efectuado a través de la API REST de OpenShift. Por otro lado, la plataforma resuelve automáticamente el registro de nuevas instancias en el balanceador de carga y en el componente de descubrimiento de servicios.

Es destacable que la plataforma favorece *DevOps* con herramientas que permiten el despliegue automatizado de los microservicios modificados (p. ej. S2I). El mantenimiento de los microservicios se hace sin períodos de indisposición (i.e. *downtime*), ya que Openshift soporta el patrón *API de chequeo de salud* (ver Sección 2.1.8), que permite determinar si un servicio desplegado está operativo. De esta forma es posible quitar la instancia vieja del microservicio, solo cuando la nueva este operativa. Un ejemplo de dicha transición puede observarse en la Figura 5.6.

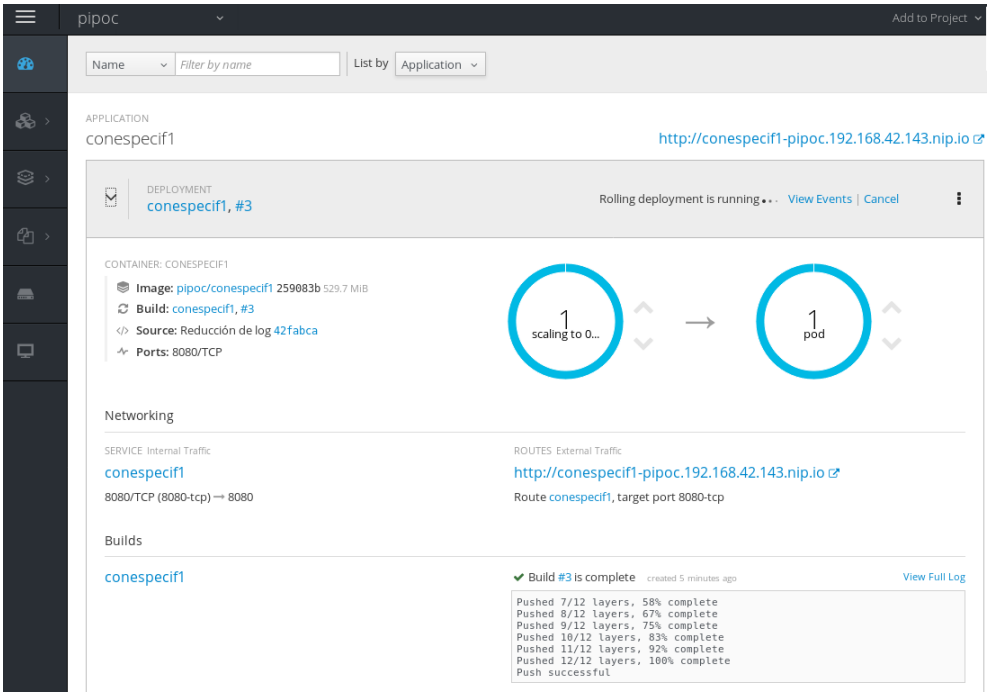

Figura 5.6: Actualización de microservicio sin periodos de indisposición.

Se observa durante la construcción que la plataforma de contenedores simplifica la implementación de la PI, brindando de forma nativa algunas de sus funcionalidades (p. ej. balanceo de carga, descrubrimiento de servicios).

Se observa además que las funcionalidades comentadas en la sección no son exclusivas de OpenShift sino que están presentes en plataformas de orquestación de contenedores. Otros

ejemplos son Heroku[8], CloudFoundry[9] y Kubernetes[10] (sobre el cual se apoya OpenShift). Algunas funcionalidades, como las automatizaciones de S2I, en caso de no existir en otras plataformas, se pueden complementar con herramientas alternativas como Jenkins[11].

5.4 Conclusiones

En este capítulo se hizo una evaluación de la propuesta principal del Capítulo 4 en base a múltiples factores.

Por un lado, se analizaron algunas decisiones de diseño relevantes y se consideró su impacto sobre los atributos de calidad de la PI. Se determinó un impacto positivo en los atributos para las decisiones consideradas y se observó que el impacto de la variante para contextos particulares es un balance en base al contexto de las ventajas y desventajas de arquitecturas de microservicios. Para dicho caso se concluye que la variante es útil en contextos donde pocos componentes concentran la mayor relevancia de la SI y los demás no requieren las ventajas de microservicios. En dichos casos, se afecta positivamente la eficiencia de desempeño y se reduce la complejidad aplicando la variante.

Por otro lado, se observó que la propuesta principal está alineada con varios patrones de microservicios presentados por Richardson. En este aspecto, se observó que los productos de plataforma de orquestación de contenedores apoyan el cumplimiento de algunos de ellos.

Finalmente se pudo comprobar la viabilidad técnica de la propuesta de arquitectura mediante la construcción de un prototipo. El mismo fue implementado sobre la plataforma *OpenShift* y los microservicios se construyeron utilizando *Java* y *Maven*, comunicándose coreográficamente mediante mensajería con *RabbitMQ*. Durante la construcción se observó que algunos componentes y funcionalidades de la PI identificados en el Capítulo 4 (p. ej. balanceamiento de carga) ya son proporcionados por una plataforma de orquestación de contenedores.

[8] https://www.heroku.com/
[9] https://www.cloudfoundry.org/
[10] https://kubernetes.io/
[11] https://jenkins.io/

6 Conclusiones y Trabajo a Futuro

6.1 Resumen y Contribuciones

Los sistemas de software a gran escala, necesitan usualmente de la integración de sistemas heterogéneos, para lo cual se apoyan comúnmente en Plataformas de Integración (PI). Por otro lado, la arquitectura de microservicios ha surgido recientemente impulsada por la industria y está ganando creciente popularidad. Esta posee ventajas como el escalamiento independiente y la mantenibilidad que podrían beneficiar a las PIs en distintos escenarios. Sin embargo su aplicabilidad en PIs no ha sido estudiada, ni tampoco se encontraron propuestas de arquitectura de microservicios aplicables a las PIs en general.

Esta tesis aborda la problemática mencionada. Para esto se analiza, por un lado, cuál es el impacto de utilizar una arquitectura de microservicios para la construcción de PIs, determinando en qué escenarios es propicia su aplicación. Por otro lado, se proponen alternativas de arquitectura y diseño para la construcción de PIs basadas en microservicios y se evalúan en base a tres factores: i) patrones y buenas prácticas de microservicios, ii) atributos de calidad de la PI, y iii) la construcción de un prototipo.

En relación al análisis de impacto, se presentan dos resultados principales. El primer resultado es un conjunto de características de los escenarios de integración, que permite identificar si un escenario se beneficia al utilizar una PI basada en microservicios. Dicho conjunto está divido en dos partes: características que sugieren compatibilidad y características que no son ideales para una PI basada en microservicios. Por ejemplo, que el escenario posea una cantidad elevada de datos a integrar, es una característica que sugiere compatibilidad con una PI basada en microservicios.

El segundo resultado, es el impacto en la calidad de la PI al aplicar una arquitectura de microservicios, el cual se analiza en base a los atributos de calidad de la PI. Se observó por ejemplo que la seguridad es una preocupación en la calidad de la PI si se utiliza una

arquitectura de microservicios. Respecto al enfoque, es destacable el aporte metodológico, el cual está basado en [14] y permite determinar el impacto sobre un sistema en base a sus atributos de calidad.

Por otro lado, para las propuestas de arquitectura se presenta primero una propuesta principal aplicable a diversos escenarios y luego se describe un conjunto de variantes que aplican en escenarios particulares, o que son producto de decisiones de diseño diferentes. También se comentan otras consideraciones hechas durante el análisis. Las propuestas se presentan con el modelo de vistas de 4+1 y son evaluadas de las formas ya descritas.

La propuesta principal es de propósito general, está basada en coreografía y se apoya en contenedores y plataformas de orquestación de contenedores para su despliegue. Las variantes de escenarios particulares, varían en la definición de qué componentes se construyen como microservicios y en su ubicación despliegue. Por otro lado, la variante de orquestación, varía la forma de coordinar la ejecución de los microservicios. Por último, en relación a otras consideraciones analizadas, se observó que desplegar una SI en forma íntegra en un contenedor conlleva a distintos problemas asociados a la arquitectura monolítica.

Se considera que las principales contribuciones de la tesis son los siguientes:

- Análisis y resumen del conocimiento existente en las áreas relacionadas a la problemática de la tesis.
- Análisis del impacto de aplicar una arquitectura de microservicios en Plataformas de Integración.
- Enfoque metodológico basado en el propuesto en [14], para analizar el impacto sobre un sistema a partir de sus atributos de calidad.
- Caracterización de escenarios de integración y principales preocupaciones sobre la calidad de la PI.
- Conjuntos de características que contribuyen a evaluar la idoneidad de un escenario respecto a aplicar una PI basada en microservicios.
- Propuesta principal de arquitectura de microservicios para Plataformas de Integración aplicable a diversos escenarios.
- Propuestas alternativas de arquitectura de microservicios para PI, que aplican en contextos particulares.
- Evaluación de la propuesta principal de arquitectura en base a tres factores: i) patrones y buenas prácticas de microservicios, ii) atributos de calidad de la PI, y iii) prototipado que contribuye a la validación técnica.

6.2 Conclusiones

Se presentan en esta sección, conclusiones y reflexiones del trabajo realizado.

La principal conclusión de esta tesis es que la arquitectura de microservicios es aplicable para la construcción de PIs en escenarios con determinadas características. Para estos se presentaron en este trabajo distintas propuestas de arquitectura y diseño.

En relación a conocimiento existente, se observó que el área de microservicios carece aún de madurez y estandarización. Por un lado, aspectos de base como el tamaño y la definición de microservicios no poseen consenso, existiendo múltiples propuestas. Por otro lado, aspectos como la seguridad, supervisión, impacto en la latencia e impacto de la forma de coordinación, han sido a la fecha poco estudiados, y se entiende que los enfoques y tecnología existentes son aún emergentes y no se han consolidado.

Con respecto a trabajo relacionado en la industria, se destacan los productos del tipo iPaaS, los cuales se asemejan conceptualmente a la PI propuesta. Si bien algunos de estos productos afirman basarse en arquitecturas de microservicios, no se encontró información detallada de dichas arquitecturas. Por el lado de propuestas académicas, se encontraron algunas propuestas de PI basadas en microservicios, aunque estas son de dominio específico, especialmente en el área de *IoT*.

En el proceso de análisis de aplicabilidad, inicialmente se consideró analizar múltiples escenarios concretos. Luego se observó que la aplicabilidad era afectada por características puntuales de los escenarios, por lo que se decidió desglosar los escenarios en características que los conforman y se las analizó en forma separada.

Por otro lado, se observó que el impacto de microservicios no debe ser analizado en forma aislada sino como un conjunto de enfoques, que incluye el uso de contenedores, la aplicación de *DevOps* y el despliegue en plataformas de orquestación de contenedores. Esto se debe a que dichos enfoques aportan soluciones a algunos desafíos o preocupaciones de la arquitectura, evitando que estas se conviertan efectivamente en un problema.

Del análisis de aplicabilidad, también se observa que las partes interesadas (i.e. *stakeholders*) podrían tener requerimientos específicos que afecten la importancia dada a ciertos atributos de calidad, afectando el análisis. Para estos casos, se propone utilizar los resultados de las Secciones 3.3 y 3.4 como punto de partida para determinar la aplicabilidad en base a los requerimientos específicos del contexto.

En otra línea, en el Capítulo 3 se determinó que algunos atributos de calidad podrían constituir una preocupación en arquitecturas de microservicios (p. ej. instalabilidad), sin embargo la tecnología existente aporta facilidades que mitigan el potencial problema de calidad. Se observa que a futuro el avance en tecnología y enfoques relacionados a microservicios puede llegar a resolver otras preocupaciones de calidad que genera actualmente la arquitectura, ampliando así la cantidad de escenarios que son beneficiados por ella.

Respecto a las propuestas de arquitectura, se observa que algunos componentes definidos en la arquitectura de PI, forman parte de las funcionalidades nativas de algunas plataformas de orquestación de contenedores. En forma análoga, existen productos o tecnología que pueden realizar las funciones de algunos componentes de la PI, por ejemplo, *RabbitMQ* para mensajería, o *Netflix Hystrix* para el patrón *Circuit Breaker*.

En relación a las variantes de arquitectura, se observó que algunas pueden ser más apropiadas que la propuesta principal en escenarios particulares. Sin embargo, se requiere poseer información del escenario de integración en forma previa a la construcción de la PI.

Con respecto al prototipo implementado, se observó que la propuesta de arquitectura es viable técnicamente. Además, se observó que las herramientas de entrega continua que poseen algunas plataformas de orquestación de contenedores (p. ej. *OpenShift*), facilitan la instalación de la solución y el reemplazo de los microservicios.

6.3 Trabajo a Futuro

6.3.1 Extender Análisis de Aplicabilidad

Una línea de trabajo a futuro es extender el análisis de aplicabilidad del Capítulo 3.

Lo anterior puede hacerse de varias formas. Por un lado, se puede estudiar nuevas características de escenarios de integración que puedan afectar la aplicabilidad de la arquitectura de microservicios. Por otro lado, los resultados del análisis podrían complementarse con pruebas empíricas (p. ej. pruebas de desempeño) que validen sus resultados. En otro aspecto, puede elegirse una forma diferente de medida que no sea estudiar el impacto sobre los atributos de calidad de la PI, para analizar la aplicabilidad. En particular una línea no analizada es cómo se impactan algunos aspectos del proceso de desarrollo de la PI como ser el tiempo de construcción que no son reflejados de forma directa en los atributos de calidad.

6.3.2 Analizar otras Variantes Específicas

Una posible continuación al trabajo de la tesis es analizar más variantes de arquitectura de microservicios.

En particular es posible analizar otras variantes de contextos específicos, las cuales pueden extender la variante propuesta en la Sección 4.3.2 o definir variantes nuevas. Un ejemplo de lo anterior son las arquitecturas vistas en trabajos académicos, en la Sección 2.6.1. Así, podrían definirse variantes más específicas optimizadas por ejemplo para un escenario de *Internet de las Cosas*.

6.3.3 Extender el Prototipo

Otra línea de trabajo a futuro es extender el prototipo del Capítulo 5 para que contemple todos los aspectos descritos en las propuestas de arquitectura presentadas, así como considerar sus variantes.

El prototipo implementado en el Capítulo 5, si bien aporta a validar técnicamente la arquitectura, también constituye una prueba de concepto acotada, que no posee todos los componentes de integración y elementos descritos en las propuestas de arquitectura. Resultados prácticos de construir la PI entera son de interés, ya que contribuyen a una validación más robusta.

Además, el prototipo construido, se basa en un enfoque de ejecución coreográfico, siendo posible estudiar también el enfoque de orquestación, así como otras variantes mostradas en el Capítulo 4.

6.3.4 Extender Funcionalidades de la PI

Otra forma de continuar el trabajo de esta tesis, es adicionar componentes y funcionalidades a la PI tomada como base para el análisis. Concretamente, se podrían considerar los componentes avanzados e interorganizacionales del trabajo de González y Ruggia (ver Sección 2.3) que agregan funcionalidades como gestión de SLA o mediación avanzada.

Se observa que dichos componentes y funcionalidades podrían ser implementadas utilizando la extensibilidad definida para la PI en el Capítulo 4 (i.e. *Componentes Personalizados*). Sin embargo, se puede considerar soportar las funcionalidades en forma nativa, utilizando enfoques mas adecuados.

6.3.5 Ejecutar Pruebas de Desempeño

Uno de los puntos que requiere investigación, es el posible impacto sobre la latencia de aplicar una arquitectura de microservicios en la PI.

En la tesis no se ejecutaron pruebas de desempeño, aunque en el Capítulo 3 se indica que podría impactar negativamente debido a las características de los sistemas distribuidos. En dicho capítulo también se comentan limitaciones de trabajos actuales sobre el tema [11], los cuales sugieren un impacto negativo leve (en contraste con arq. monolíticas). En el prototipo implementado en el Capítulo 5 no se tuvieron incidencias sobre el comportamiento del tiempo, aunque dichas pruebas son acotadas como para extraer conclusiones.

Se requiere por tanto mayor estudio sobre el punto ya que su impacto podría condicionar la aplicabilidad de la arquitectura.

Referencias

[1] L. González y R. Ruggia. A reference architecture for integration platforms supporting cross-organizational collaboration. En *The 17th International Conference on Information Integration and Web-based Applications & Services*, Bruselas, Bélgica, 2015. ACM.

[2] G. Hohpe y B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 2003.

[3] K. Wähner. Choosing the Right ESB for Your Integration Needs. 2013. [En línea] Disponible en: <http://www.infoq.com/articles/ESB-Integration> [Accedido: 01-06-2017].

[4] P. Mell y T. Grance. The NIST Definition of Cloud Computing. Technical Report 800-145, NIST.

[5] E. Anderson, C. Eschinger, L. Wurster, F. de Silva, R. Contu, V. Liu, F. Biscotti, G. Petri, J. Zhang, M. Yeates, y others. Forecast overview: Public cloud services, market growth 2012-2016. *Worldwide*, 2013.

[6] M. Pezzini y B. J. Lheureux. Integration Platform as a Service: Moving Integration to the Cloud. Technical Report G00210747, Gartner Inc.

[7] D. Merkel, F. Santas, A. Heberle, y T. Ploom. Cloud integration patterns. Con S. Dustdar, F. Leymann, y M. Villari, editores, en *Service Oriented and Cloud Computing*, páginas 199–213. Springer International Publishing, 2015.

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

[9] M. Fowler y J Lewis. Microservices. 2014. [En línea] Disponible en: <http://martinfowler.com/articles/microservices.html> [Accedido: 01-06-2017].

[10] A. Balalaie, A. Heydarnoori, y P. Jamshidi. Migrating to cloud-native architectures

using microservices: An experience report. 2015.

[11] M. Villamizar, O. Garces, H. Castro, M. Verano, L. Salamanca, R. Casallas, y S. Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. En *2015 10th Colombian Computing Conference, 10CCC 2015*, páginas 583–590, Systems and Computing Engineering Department, Universidad de Los Andes, 2015.

[12] C. Esposito, A. Castiglione, y K. Choo. Challenges in delivering software in the cloud as microservices. *IEEE Cloud Computing*, 3(5):10–14, 2016.

[13] F. Montesi y J. Weber. Circuit breakers, discovery, and API gateways in microservices. 2016.

[14] L. O'Brien, L. Bass, y P. Merson. Quality attributes and service-oriented architectures. Technical Report CMU/SEI-2005-TN-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, 2005.

[15] S. Newman. *Building Microservices*. O'Reilly Media, Inc., 1era edición, 2015.

[16] R. Martin. The single responsibility principle. 2014.

[17] S. Daya, N. Van Duy, K. Eati, C.M. Ferreira, D. Glozic, V. Gucer, M. Gupta, S. Joshi, V. Lampkin, M. Martins, y others. *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks, 2016.

[18] C. Richardson. Microservice patterns. 2017.

[19] C. Richardson, D. Bryant, G. Engstrand, J. Long, y A. Silva. Virtual Panel: Microservices Interaction and Governance Model - Orchestration v Choreography. [En línea] Disponible en: <https://www.infoq.com/articles/vp-microservices-orchestration-choreography> [Accedido: 10-11-2017].

[20] C. Richardson. Pattern: Microservice Architecture. [En línea] Disponible en: http://microservices.io/patterns/microservices.html> [Accedido: 01-06-2017].

[21] M. Fowler. Microservices Trade-Offs. 2015. [En línea] Disponible en: <http://martinfowler.com/articles/microservices.html> [Accedido: 01-06-2017].

[22] C. Oliver. Microservices at Netflix: Stop system problems before they start. [En línea] Disponible en: <https://www.datawire.io/netflix-created-microservices-architecture-stop-system- problems-start/> [Accedido: 01-06-2017].

[23] S. Hassan y R. Bahsoon. Microservices and their design trade-offs: A self-adaptive roadmap. *2016 IEEE International Conference on Services Computing (SCC)*, 00:813–818, 2016.

[24] A. Balalaie, A. Heydarnoori, y P. Jamshidi. Microservices architecture enables devops:

Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52, 2016.

[25] B. Fitzgerald y K. Stol. Continuous software engineering: A roadmap and agenda. *The Journal of Systems Software*, 123:176 – 189, 2017.

[26] C. Carneiro y T. Schmelmer. *Microservices From Day One : Build Robust and Scalable Software From the Start.* Apress, 2016.

[27] M. Nygard. *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2007.

[28] T. Cerny, M. Donahoo, y M. Trnka. Contextual understanding of microservice architecture: Current and future directions. *SIGAPP Appl. Comput. Rev.*, 17(4):29–45, Enero 2018.

[29] Elastic.io. Log aggregation for Docker containers in Mesos/Marathon cluster. [En línea] Disponible en: <https://www.elastic.io/log-aggregation-for-docker-containers-in-mesos-marathon-cluster/> [Accedido: 01-06-2017].

[30] T. Mauro. Adopting Microservices at Netflix: Lessons for Architectural Design. [En línea] Disponible en: <https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/> [Accedido: 01-06-2017].

[31] U. Zdun, E. Navarro, y F. Leymann. Ensuring and assessing architecture conformance to microservice decomposition patterns. En *The 15th International Conference on Service-Oriented Computing 2017*, Noviembre 2017.

[32] A. Gupta. Microservice Design Patterns. [En línea] Disponible en: <http://blog.arungupta.me/microservice-design-patterns/> [Accedido: 01-06-2017].

[33] A. Wiggins. The twelve-factor app, 2012.

[34] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, y M. Steinder. Performance evaluation of microservices architectures using containers. *2015 IEEE 14th International Symposium on Network Computing & Applications*, página 27, 2015.

[35] R. Barik, R. Lenka, K. Rao, y D. Ghose. Performance analysis of virtual machines and containers in cloud computing. En *2016 International Conference on Computing, Communication and Automation (ICCCA)*, páginas 1204–1210, Abril 2016.

[36] Zheng Li, M. Kihl, Q. Lu, y J. Andersson. Performance overhead comparison between hypervisor and container based virtualization. 2017.

[37] Docker Inc. What is a Container. [En línea] Disponible en: <https://www.docker.com/what-container> [Accedido: 01-06-2017].

[38] S. Bigelow. Container vs. vm: What's the difference? [En línea] Disponible en: <https://searchservervirtualization.techtarget.com/answer/Containers-

vs-VMs-Whats-the-difference> [Accedido: 01-06-2017].

[39] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), Marzo 2014.

[40] Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing, Cloud Computing, IEEE, IEEE Cloud Comput*, (5):42, 2017.

[41] D. Rensin. *Kubernetes - Scheduling the Future at Cloud Scale*. 2015.

[42] L. González, J. Laborde, M. Galnares, M. Fenoglio, y R. Ruggia. An adaptive enterprise service bus infrastructure for service based systems. En *1st Workshop on Pervasive Analytical Service Clouds for the Enterprise and Beyond*, Berlín, Alemania, 2013.

[43] B. Rienzi, L. González, y R. Ruggia. Towards an ESB-Based enterprise integration platform for geospatial web services. En *GEOProcessing 2013, The Fifth International Conference on Advanced Geographic Information Systems, Applications, and Services*, página 39–45, 2013.

[44] L. González y G. Ortiz. An event-driven integration platform for context-aware web services. *Journal of Universal Computer Science*, 20(8):1071–1088, aug 2014.

[45] G. Llambías, L. González, y R. Ruggia. Towards an integration platform for bioinformatics services. En *1st Workshop on Pervasive Analytical Service Clouds for the Enterprise and Beyond*, Berlín, Alemania, 2013.

[46] Gartner Inc. Market guide for hybrid integration platform-enabling technologies, 2016.

[47] D. Chappell. *Enterprise Service Bus: Theory in Practice*. O'Reilly Media, Junio 2004.

[48] A. Stanford-Clark. Coupled or decoupled, and heavyweight and lightweight delivery considerations in an enterprise service bus. Technical Report G224-9149-00, IBM.

[49] L. González. *Plataforma ESB Adaptativa para Sistemas Basados en Servicios*. Tesis de Maestría en Informática, PEDECIBA Informática | Instituto de Computación – Facultad de Ingeniería – Universidad de la República, 2011.

[50] M. Papazoglou. *Web Services & SOA: Principles & Technology, 2nd ed.* Trans-Atlantic Publications, Inc., Essex, Inglaterra ; New York, 2da edición, Enero 2012.

[51] Elastic.io. When does iPaaS finally get to replace Enterprise Service Bus? [En línea] Disponible en: <https://www.elastic.io/when-ipaas-replace-enterprise-service-bus/> [Accedido: 01-06-2017].

[52] SnapLogic. The rise of the citizen integrator. [En línea] Disponible en: <http://www.snaplogic.com/blog/_citizen-integrator/> [Accedido: 01-06-2017].

[53] Gartner Inc. Market guide for citizen integrator tools, 2015.

[54] O. Zimmermann, C. Pautasso, G. Hohpe, y B. Woolf. A decade of enterprise integration patterns: A conversation with the authors. *IEEE Software*, 1:13–19, Febrero 2016.

[55] WSO2 Inc. Enterprise Integration Patterns with WSO2 ESB, Online Documentation. [En línea] Disponible en: <https://docs.wso2.com/display/IntegrationPatterns/Enterprise+Integration+Patterns+with+WSO2+ESB> [Accedido: 01-06-2017].

[56] MuleSoft Inc. Understanding Enterprise Integration Patterns Using Mule, Online Documentation. [En línea] Disponible en: <https://docs.mulesoft.com/mule-user-guide/v/3.7/understanding-enterprise-integration-patterns-using-mule> [Accedido: 01-06-2017].

[57] Apache Software Foundation. Apache Camel, Online Documentation. [En línea] Disponible en: <http://camel.apache.org/> [Accedido: 01-06-2017].

[58] D. Ritter, N. May, y S. Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. *Information Systems*, 67:36–57, Julio 2017.

[59] P. Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6):42–50, Noviembre 1995.

[60] L. Bass, P. Clements, y R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3ra edición, 2012.

[61] IEEE Computer Society, P. Bourque, y R. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Los Alamitos, USA, 3ra edición, 2014.

[62] E. Djogic, S. Ribic, y D. Donko. Monolithic to microservices redesign of event driven integration platform. En *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, páginas 1411–1414, Mayo 2018.

[63] InfoQ. Managing Data in Microservices. [En línea] Disponible en: <https://www.infoq.com/articles/managing-data-microservices> [Accedido: 01-06-2017].

[64] M. Ciavotta, M. Alge, S. Menato, D. Rovere, y P. Pedrazzoli. A microservice-based middleware for the digital factory. *Procedia Manufacturing*, 11(27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 Junio 2017, Modena, Italy):931 – 938, 2017.

[65] A. Auger, E. Exposito, y E. Lochin. iQAS: An integration platform for QoI assessment as a service for smart cities. En *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, páginas 88–93, Diciembre 2016.

[66] V. George y Q. Mahmoud. Claimsware: A claims-based middleware for securing iot

services. En *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volumen 1, páginas 649–654, Julio 2017.

[67] P. Singh, M. Van Sinderen, y R. Wieringa. Reference architecture for integration platforms. En *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*, Octubre 2017.

[68] Elastic.io. Elastic.io Platform 2.0 democratizes application integration. [En línea] Disponible en: <https://www.elastic.io/elastic-io-democratizes-application-integration/> [Accedido: 01-06-2017].

[69] Elastic.io. Elastic.io new challenges with cloud integration. Presentado en el evento "EA Connect Day". [En línea] Disponible en: <https://info.leanix.net/hubfs/Website_Downloads/EA_Connect_Day_Presentations/EAConnectDay2016-Elasticio-Cloud-Integration.pdf> [Accedido: 01-06-2017].

[70] Heroku. Slugs compiler. [En línea] Disponible en: <https://devcenter.heroku.com/articles/slug-compiler> [Accedido: 01-06-2017].

[71] Elastic.io. Log aggregation for Docker containers in Mesos / Marathon cluster. [En línea] Disponible en: <https://www.elastic.io/log-aggregation-for-docker-containers-in-mesos-marathon-cluster/> [Accedido: 01-06-2017].

[72] WSO2 Inc. WSO2 Enterprise Integrator, Key Concepts, Online Documentation. [En línea] Disponible en: <https://docs.wso2.com/display/EI611/Key+Concepts> [Accedido: 01-06-2017].

[73] T. Yarygina y A. Bagge. Overcoming security challenges in microservice architectures. *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), Service-Oriented System Engineering (SOSE), 2018 IEEE Symposium on, SOSE*, 2018.

[74] Y. Sun, S. Nanda, y T. Jaeger. Security-as-a-service for microservices-based cloud applications. *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2015.

[75] Sam Newman. Security and microservices. NDC Conferences, 2017.

[76] Red Hat Software. Openshift Container Platform v1 REST API Online Documentation. <https://docs.openshift.com/container-latform/3.5/rest_api/openshift_v1.html>.

[77] Y. Duan, F. Guohua, N. Zhou, X. Sun, N. Narendra, y B. Hu. Everything as a service (xaas) on the cloud: Origins, current and future trends. Con Calton Pu y Ajay Mohindra, editores, en *CLOUD*, páginas 621–628. IEEE, 2015.

[78] K. Guttridge, M. Pezzini, E. Golluscio, E. Thoo, K. Iijima, y M. Wilcox. Magic Quadrant for Enterprise Integration Platform as a Service. Technical Report G00277632, Gartner Inc.

[79] SnapLogic. Snaplogic Online Documentation. [En línea] Disponible en: <https://docs-snaplogic.atlassian.net/wiki/spaces/SD/overview> [Accedido: 01-06-2017].

[80] J. Bloomberg. Microservices: Avoiding Dumb Pipes. [En línea] Disponible en: <https://intellyx.com/2015/03/11/microservices-avoiding-dumb-pipes/> [Accedido: 01-06-2017].

[81] J. Bloomberg. Challenges of Microservices Architectures. [En línea] Disponible en: <https://www.snaplogic.com/resources/white-papers/challenges-of-microservices-architectures> [Accedido: 01-06-2017].

[82] Built.io. Built.io Online Documentation. [En línea] Disponible en: <https://flowdocs.built.io/> [Accedido: 01-06-2017].

[83] Built.io. Will Integration Platform As A Service (iPaaS) Replace Enterprise Serial Bus (ESB)? [En línea] Disponible en: <https://www.built.io/blog/will-integration-platform-as-a-service-ipaas-replace-enterprise-serial-bus-esb-> [Accedido: 01-06-2017].

[84] Informatica. Informatica Cloud, Online Help - Secure Agent. [En línea] Disponible en: <https://app3.informaticacloud.com/saas/v389/docs/EN/index.htmpage/cc-cloud-administer/Secure_Agents.html> [Accedido: 01-06-2017].

[85] Informatica. Informatica Reinvents iPaaS with Next-generation Cloud Services Reimagined to Address Data Management Challenges. [En línea] Disponible en: <https://www.informatica.com/about-us/news/news-releases/2017/05/20170516-informatica-reinvents-ipaas-with-next-generation-cloud-services.html> [Accedido: 01-06-2017].

[86] MuleSoft. Mulesoft Cloudhub Architecture, Online Documentation. [En línea] Disponible en: <https://docs.mulesoft.com/runtime-manager/cloudhub-architecture> [Accedido: 01-06-2017].

[87] MuleSoft. Understanding Enterprise Integration Patterns using Mule, Online Documentation. [En línea] Disponible en: <https://docs.mulesoft.com/mule-user-guide/v/3.7/understanding-enterprise-integration-patterns-using-mule> [Accedido: 01-06-2017].

[88] Fiorano. Fiorano Integration Platform Datasheet. [En línea] Disponible en: <http://www.fiorano.com/assets/pdf/datasheets/fiorano_soa_brochure.pdf> [Accedido: 01-06-2017].

[89] Fiorano. Peer-to-Peer Microservice Cloud Pipelines: Powering the Digital Economy. [En línea] Disponible en: <https://www.fiorano.com/whitepapers/Whitepaper-P2P_Microservices_Cloud_Pipelines.pdf> [Accedido: 01-06-2017].

[90] G. Hohpe. Enterprise Integration Patterns Website. [En línea] Disponible en: <http://www.enterpriseintegrationpatterns.com/> [Accedido: 01-06-2017].

[91] Fiorano. Fiorano - Adapters. [En línea] Disponible en: <http://www.fiorano.com/products/components/adapters.php> [Accedido: 01-06-2017].

Apéndices

A Estándar ISO/IEC 25010:2011 de Calidad de Productos de Software

Este apéndice es una traducción del estándar ISO/IEC 25010:2011 que no adiciona información ni comentarios por parte del autor de esta tesis, a excepción de la organización de la sección. En la Tabla A.1 pueden observarse todos los atributos y subatributos de calidad definidos por el estándar mencionado, el cual es usado en este trabajo.

A continuación se presenta la descripción y notas de cada atributo y subatributo, tal como se muestran en el estándar[1]. La organización es la siguiente: los atributos de calidad se presentan numerados y en negrita, detallando para cada uno su descripción y subatributos. Las descripciones pueden incluir ejemplos o notas extendiendo la definición, de acuerdo a como está presentado en el estándar.

1. **Idoneidad funcional**

   Grado en el cual un producto o sistema provee funcionalidades que permitan cumplir con las necesidades explicitas e implícitas, al usarse bajo ciertas condiciones.

   Nota 1: La adecuación funcional no contempla la especificación funcional.

   Posee los siguientes subatributos:

   a) Completitud funcional: Grado en el cual un conjunto de funcionalidades cubre todas las tareas especificadas y los objetivos del usuario.

   b) Correctitud funcional: Grado en el cual un producto o sistema provee los resultados correctos, con el nivel de precisión requerido.

   c) Adecuación funcional: Grado en el cual las funcionalidades facilitan el cumplimiento de las tareas y objetivos de usuario especificados.

[1] Traducido al español por el autor. El original puede encontrarse en la especificación del estándar.

Tabla A.1: Atributos y subatributos del estándar ISO/IEC 25010:2011

Atributo	**Subatributo**
Idoneidad Funcional	Completitud Funcional
	Correctitud Funcional
	Adecuación Funcional
Eficiencia de Desempeño	Comportamiento del Tiempo
	Uso de Recursos
	Capacidad
Compatibilidad	Coexistencia
	Interoperabilidad
Usabilidad	Idoneidad Reconocible
	Autoaprendizaje
	Operabilidad
	Protección a Errores de Usuario
	Estética de Interfaz de Usuario
	Accesibilidad
Mantenibilidad	Modularidad
	Reusabilidad
	Evaluabilidad
	Modificabilidad
	Testabilidad
Portabilidad	Adaptabilidad
	Instalabilidad
	Reemplazabilidad
Confiabilidad	Madurez
	Disponibilidad
	Tolerancia a Fallas
	Recuperabilidad
Seguridad	Confidencialidad
	Integridad
	No repudio
	Trazabilidad
	Autenticidad

Traducido al español por el autor.

Ejemplo 1: Se le presenta a un usuario solamente los pasos necesarios para completar una tarea, excluyendo pasos innecesarios.

2. **Eficiencia de desempeño**

Desempeño relacionado a la cantidad de recursos utilizados bajo condiciones especifi-

cadas.

Nota 1: Los recursos pueden incluir el uso de otros productos de software, materiales (p. ej. persistencia) y configuración de software y hardware.

Posee los siguientes subatributos:

a) Comportamiento del tiempo: Grado en el cual la respuesta, tiempos de procesamiento y ratios de salida (i.e. *throughput*) de un producto o sistema al ejecutar sus funcionalidades, cumple los requisitos.

b) Uso de Recursos: Grado en el cual la cantidad de recursos usados por un producto o sistema al ejecutar sus funcionalidades, cumple los requisitos.

   Nota 1: Los recursos humanos no son incluidos en el subatributo.

c) Capacidad: Grado en que los límites máximos de un producto, sistema o parámetro de sistema cumplen con los requisitos.

   Nota 1: Los parámetros de sistema pueden incluir por ejemplo el número de items que pueden ser persistidos, el número de usuarios concurrentes, el ancho de banda de comunicación, la cantidad de transacciones y el tamaño de la base de datos.

3. **Compatibilidad**

   Grado en el cual un producto, sistema o componente puede intercambiar información con otros productos, sistemas o componentes, para cumplir las funcionalidades requeridas, mientras comparte el ambiente de hardware o software.

   Posee los siguientes subatributos:

   a) Coexistencia: Grado en el cual un producto puede ejecutar sus funcionalidades eficientemente mientras comparte ambiente y recursos con otros productos, sin perjudicar o impactar en los mismos.

   b) Interoperabilidad: Grado en el cual dos o más sistemas, productos o componentes pueden intercambiar información y usar la información intercambiada.

4. **Usabilidad**

   Grado en el cual un producto o sistema puede ser usado por los usuarios para alcanzar las metas especificadas con efectividad, eficiencia y satisfacción en un contexto de uso específico.

   Posee los siguientes subatributos:

   a) Idoneidad Reconocible: Grado en el que los usuarios pueden reconocer si un producto o sistema es apropiado para sus necesidades.

Nota 1: La idoneidad reconocible depende de la habilidad para reconocer la idoneidad de las funcionalidades de un producto o sistema a partir de la documentación asociada o de la primera impresión causada.

Nota 2: La información provista para el producto o sistema puede incluir demostraciones, tutoriales, documentaciones, o la información del sitio inicial (en caso de sitios web).

b) Autoaprendizaje: Grado en el que un sistema o producto permite a un usuario aprender del mismo sin riesgo y con efectividad, eficiencia y satisfacción.

c) Operabilidad: Grado en el que un producto o sistema posee atributos que facilitan su operación y control.

d) Protección a Errores de Usuario: Grado en el que un sistema protege a un usuario de cometer errores.

e) Estética de Interfaz de Usuario: Grado en que la interfaz de usuario agrada y satisface a un usuario en base a la interacción con la misma.

Nota 1: Refiere a propiedades del sistema o producto que incrementen la satisfacción del usuario, como el uso de colores y la naturaleza del diseño gráfico.

f) Accesibilidad: Grado en que un producto o sistema puede ser usado por personas con un un amplio rango de características y capacidades, en un contexto determinado.

Nota 1: El rango de capacidades incluye la edad del usuario.

5. **Mantenibilidad**

Grado de efectividad y eficiencia con el que un producto o sistema puede ser modificado por los responsables especificados.

Nota 1: Las modificaciones pueden incluir correcciones, mejoras o adaptaciones del software, así como cambios en el ambiente, en los requisitos o en la especificación funcional. Se consideran las modificaciones hechas por equipos especializados, de negocio, operacionales o usuarios del sistema.

Nota 2: Mantenibilidad incluye la instalación de actualizaciones o mejoras.

Nota 3: Puede ser interpretada como la capacidad inherente a un producto o sistema de facilitar actividades de mantenimiento o también como la calidad percibida por los responsables del mantenimiento al efectuar dicha tarea.

Posee los siguientes subatributos:

a) Modularidad: Grado en que un producto o sistema es compuesto por componentes discretos, tal que el cambio en un componente posee mínimo impacto en

los demás.

b) Reusabilidad: Grado en que un activo del sistema puede ser usado para construir nuevos activos o en otros sistemas.

c) Evaluabilidad: Grado de efectividad y eficiencia con el cual es posible determinar el impacto sobre un producto o sistema, de un cambio pretendido para una o más de sus partes. También incluye el grado de efectividad y eficiencia con el cual es posible diagnosticar causantes de falla, o identificar partes a ser modificadas para un producto o sistema.

Nota 1: Su implementación puede incluir crear mecanismos para el producto o sistema que le permitan analizar sus propias fallas y proveer reportes en casos de errores u otros eventos.

d) Modificabilidad: Grado de efectividad y eficiencia con el cual un producto o sistema puede ser modificado sin introducir defectos y sin degradar la calidad existente del mismo.

Nota 1: Su implementación incluye la codificación, diseño, documentación y verificación de las modificaciones.

Nota 2: Los subatributos modularidad y evaluabilidad pueden influenciar la modificabilidad.

Nota 3: Modificabilidad es una combinación de la capacidad de cambio y la estabilidad de un producto o sistema.

e) Testabilidad: Grado de efectividad y eficiencia con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

6. **Portabilidad**

Grado de efectividad y eficiencia con el cual un producto, sistema o componente puede ser transferido desde un ambiente de hardware, software, operación o uso; a otro distinto.

Nota 1: Portabilidad puede ser interpretada como la capacidad inherente a un producto o sistema de facilitar las actividades de migración, o la calidad experimentada durante las actividades de migración de un producto o sistema.

Posee los siguientes subatributos:

a) Adaptabilidad: Grado de efectividad y eficiencia con el cual un producto o sistema puede adaptarse a diferentes ambientes de uso, operación o de hardware y software.

Nota 1: La adaptabilidad incluye la escalabilidad de la capacidad interna (p. ej. volúmenes de transacciones y tablas).

Nota 2: Incluye adaptaciones hechas por equipos especializados, de negocio, operacionales o usuarios del sistema.

Nota 3: Si las adaptaciones son hechas por un usuario, la adaptabilidad corresponde a la adecuación individual a dicho usuario.

b) Instalabilidad: Grado de efectividad y eficiencia con el cual un producto o sistema puede instalarse o desinstalarse de forma exitosa en un ambiente especificado.

Nota 1: Si el producto o sistema debe ser instalado por un usuario, la instalabilidad puede afectar los subatributos adecuación funcional y operabilidad.

c) Reemplazabilidad: Grado en el que un producto o sistema puede reemplazar otro producto de software especificado para el mismo propósito y en el mismo ambiente.

Nota 1: La reemplazabilidad de una nueva versión de un producto de software es importante para el usuario al actualizar.

Nota 2: La reemplzabilidad podría incluirse dentro de los subatributos instalabilidad y adaptabilidad. Se separa en un nuevo subatributo debido a su importancia.

Nota 3: La reemplzabilidad reduce el riesgo de dependencia (i.e. *lock-in*), de forma que un producto de software puede ser usado en lugar del actual, mediante el uso de formatos estandarizados.

7. **Confiabilidad**

Grado en que un sistema, producto o componente ejecuta funcionalidades específicas bajo condiciones determinadas durante un periodo de tiempo definido.

Nota 1: El desgaste no ocurre en software. Limitaciones de confiabilidad ocurren por fallas en los requerimientos, diseño e implementación o por cambios contextuales.

Posee los siguientes subatributos:

a) Madurez: Grado en que un sistema, producto o componente cumple los requisitos de confiabilidad bajo operación normal.

Nota 1: El concepto de madurez puede también ser aplicado a otros atributos de calidad para indicar el grado con el que cumplen sus requisitos bajo operación normal.

b) Disponibilidad: Grado en que un sistema, producto o componente está operacional y accesible cuando se requiere su uso.

Nota 1: Externamente, la disponibilidad puede ser mostrada como el tiempo total durante el que el sistema, producto o componente está operativo. El subatributo es por tanto una combinación de madurez (que define frecuencia de falla) y tolerancia a falla con recuperabilidad (que define tiempo de indisposición cuando ocurre una falla),

c) Tolerancia a Fallas: Grado en que un sistema, producto o componente opera como es esperado a pesar de la presencia de errores de hardware o software.

d) Recuperabilidad: En el caso de una falla o interrupción, grado en que un producto o sistema puede recuperar los datos y reestablecer la operatividad normal.

Nota 1: Luego de una falla, un sistema puede no estar disponible durante un tiempo. Dicho tiempo está asociado a la recuperabilidad.

8. **Seguridad**

Grado en que un producto o sistema protege la información y los datos, de forma de que personas u otros sistemas tengan el acceso a datos apropiado a su nivel de autorización.

Nota 1: Incluye datos almacenados en, o por, el producto o sistema. También aplica a datos en trasmisión.

Nota 2: El grado en que un producto o sistema continúa proveyendo servicios a pesar de ataques, es cubierto por el subatributo recuperabilidad.

Nota 3: El grado en que un producto o sistema es resistente a un ataque esta cubierto por el subatributo integridad.

Posee los siguientes subatributos:

a) Confidencialidad: Grado en que un producto o sistema asegura que los datos solo pueden ser accedidos por quienes poseen autorización.

b) Integridad: Grado en que un sistema, producto o componente previene acceso no autorizado o modificación de programas o datos.

c) No repudio: Grado en que puede probarse la ocurrencia de acciones o eventos en un sistema, de forma de que no puedan ser repudiados luego de ocurridos.

d) Trazabilidad: Grado en que las acciones de una entidad pueden ser vinculadas en forma única a dicha entidad.

e) Autenticidad: Grado en que puede probarse que la identidad de un sujeto o recurso es la que este plantea ser.

B

Tipos de servicios en la nube

Los tipos de servicios clásicos que ofrece la nube, son tres [4]:

1. Infraestructura como servicio (IaaS)
2. Plataforma como servicio (PaaS)
3. Software como servicio (SaaS)

En la Figura B.1 se muestra, a grandes rasgos, las diferencias entre los distintos servicios clásicos ofrecidos en la nube.

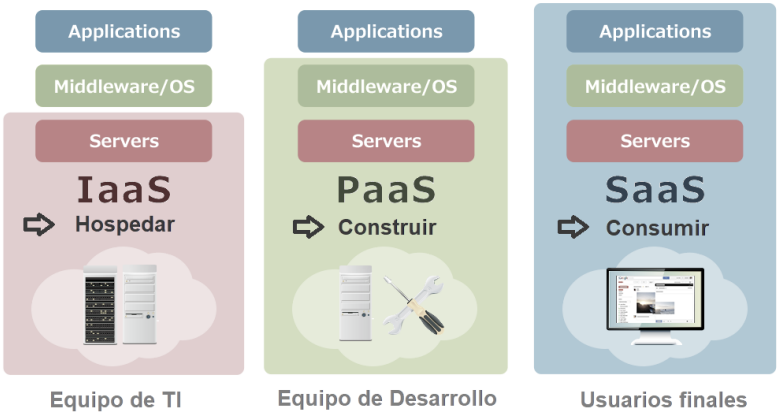

Figura B.1: Servicios clásicos de Cloud Computing[1]

[1] Imágen modificada de https://medium.com/@Albihany/true-cloud-story-about-iaas-paas-saas-47cfea883271

La infraestructura como servicio ofrece alojamiento en la nube. Los proveedores de IaaS suelen proporcionar al cliente máquinas virtuales que tienen un sistema operativo montado y acceso a Internet. Corre por cuenta del usuario instalar y configurar software junto a sus dependencias, tal como se haría en un servidor nuevo. Si se quisiese alojar un sitio web por ejemplo, el usuario debe instalar y configurar el servidor web y todas las dependencias de ejecución del sitio antes del software en sí. Por tanto, dado que requiere de tareas propias de operaciones, apunta a ser utilizada (al menos en primer instancia) por un equipo de operaciones o de TI.

Por otro lado, PaaS ofrece un servicio de más alto nivel que IaaS. Otorga las funcionalidades necesarias para construir y desplegar un sistema. Esto es, ambiente de desarrollo, integración de repositorios, automatización de despliegues, supervisión, etc. Por tanto, se puede decir que sus servicios apuntan a equipos de desarrollo.

Por último el software como servicio (SaaS) ofrece una aplicación o sistema listo para usar en un enfoque *multitenant* con un espacio de datos y operación aislado. Brinda un producto de software ya construido y por tanto, no es necesario involucrar para su puesta en marcha a personal de TI ni a Ingenieros de Software. Por tanto, sus servicios apuntan a usuarios de negocio que pueden utilizar el software de inmediato.

Además de los servicios clásicos en la nube [4] (i.e. IaaS, PaaS, SaaS), existen otras variantes de servicio en la nube. Dada la variedad de propuestas, esto a conllevado al término "*everything as a service*" o "XaaS" [77]. Ejemplos son: persistencia como servicio (abreviado SaaS o STaaS), mensajería como servicio (abreviado MaaS o MQaaS), monitoreo como servicio, entre otros. En la Sección 2.3.2, se hace foco en el concepto plataforma de integración como servicio (iPaaS), el cual es una variante de la plataforma como servicio que ofrece herramientas y funcionalidades para la integración de sistemas.

C Otros Productos de PI Relacionados

Se muestran en este apéndice otros casos de productos de PI estudiados que poseen una arquitectura de microservicios o están basados en los EIP.

Análogo a la Sección 2.6.2 del trabajo relacionado, se separa en dos categorías principales:

- Plataformas de Integración como Servicio (iPaaS)
- Suites de Integración On-Premise (p. ej. ESBs)

C.1 Productos iPaaS

En [78] Gartner realiza un análisis, clasificación y evaluación de los distintos productos de iPaaS de la industria. La elección de cuáles iPaaS se analizan en esta sección, se basa en dicho articulo. El foco del análisis, es determinar si los productos poseen una arquitectura de microservicios y observar sus capacidades de mediación, analizando si implementan los EIP.

Se muestran a continuación otros productos de PI del tipo iPaaS estudiados.

C.1.1 SnapLogic

La empresa SnapLogic ofrece una iPaaS llamada "*SnapLogic Enterprise Integration Cloud*", que se basa en la construcción de flujos de integración. Se apoya en dos conceptos principales: "*Snaps*" y "*Snaplex*".

Snaps es el nombre dado a componentes de integración utilizables en un flujo de integración. *Snaplex* es el motor de procesamiento de datos y plataforma de ejecución de la iPaaS. Este

escala de forma automática y es elástico, encargándose de manejar y supervisar los *Snaps*. El escalamiento, puede ser configurado a partir de un SLA definido (p. ej. uso de memoria) o en base a volumen de datos a procesar. Como puede observarse en la Figura C.1, *Snaplex* posee tres variantes, según la ubicación y tipo de los datos integrar:

- *CloudPlex* - Para escenarios de integración en la nube.
- *HadooPlex* - Para escenarios de integración con *Big Data*.
- *GroundPlex* - Para escenarios de integración de sistemas internos.

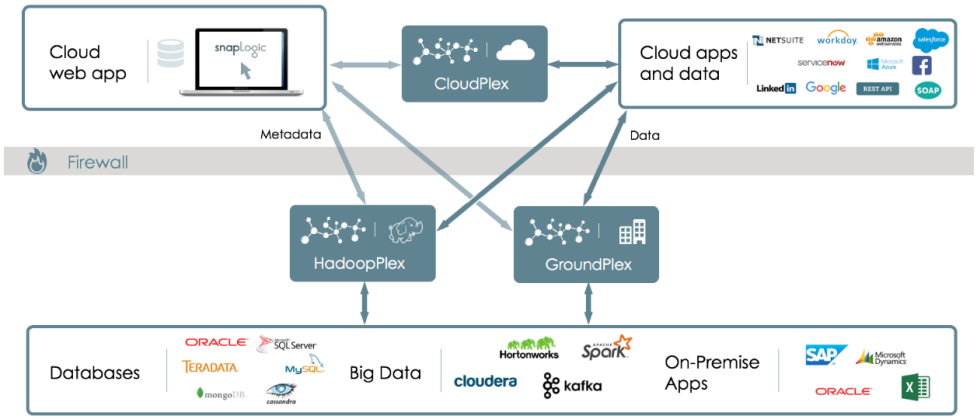

Figura C.1: Opciones de despliegue de *Snaplex* [79]

Si bien Gartner no detalla que SnapLogic utilice microservicios en su arquitectura, el equipo de Snaplogic y la empresa analista Intellyx lo mencionan explícitamente en algunos artículos [80, 81]. En estos, se afirma que los *Snaps* están construidos como microservicios hospedados en contenedores, los cuales son, a su vez, administrados por *Snaplex*. No se brinda en las documentaciones detalles de la arquitectura. Respecto a la implementación de los EIPs, tampoco se especifica en la documentación que estos sean implementados [79].

C.1.2 Built.io

La empresa Built.io[1] propone un producto de iPaaS llamado "*Built.io Flow*", el cual tiene dos variantes: i) *Built.io Flow Express*, y ii) *Built.io Flow Enterprise*.

Si bien ambas variantes están basadas en flujos de integración, difieren en el tipo de usuario al cual apuntan, ofreciendo herramientas para perfiles con habilidades técnicas distintas (véase 2.3.3).

[1] https://www.built.io/

Built.io Flow Express apunta a usuarios de negocio con un conocimiento técnico bajo. La versión posee exclusivamente componentes prediseñados, que el usuario simplemente conecta para crear un flujo de integración simple. Los componentes aportan funcionalidades de alto nivel de abstracción que permiten efectuar integraciones puntuales de herramientas de negocio. En la Figura C.2 pueden verse algunos ejemplos.

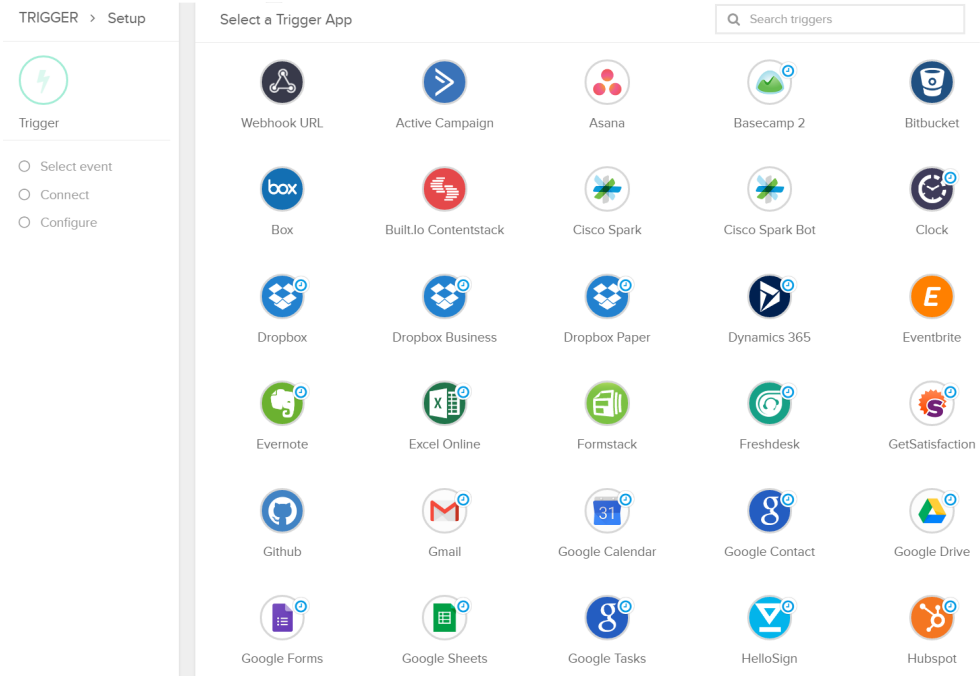

Figura C.2: Algunos disparadores de flujo de *Built.io Flow Express* [2]

Built.io Flow Enterprise está pensado para usuarios con perfil técnico y conocimiento de programación. Ofrece las mismas características que la versión *Express*, pero permite agregar conectores propios mediante una interfaz predefinida. Así mismo ofrece componentes de integración avanzados para efectuar operaciones de bajo nivel de abstracción.

Por otro lado, el producto ofrece un conector llamado "*Enterprise Gateway*" para integrar escenarios internos y en la nube. En la Figura C.3 puede observarse un diagrama de la arquitectura.

Gartner afirma que la plataforma está construida usando microservicios y que está alineada a las prácticas de DevOps y entrega continua [78]. Sin embargo dicha información no pudo ser corroborada en la documentación de Built.io [82]. La arquitectura de los productos iPaaS no está documentada ni explicitada. Built.io tiene algunos artículos en su Blog[3] que reflexionan

[2] Captura tomada en una prueba de concepto de la plataforma.
[3] https://www.built.io/blog

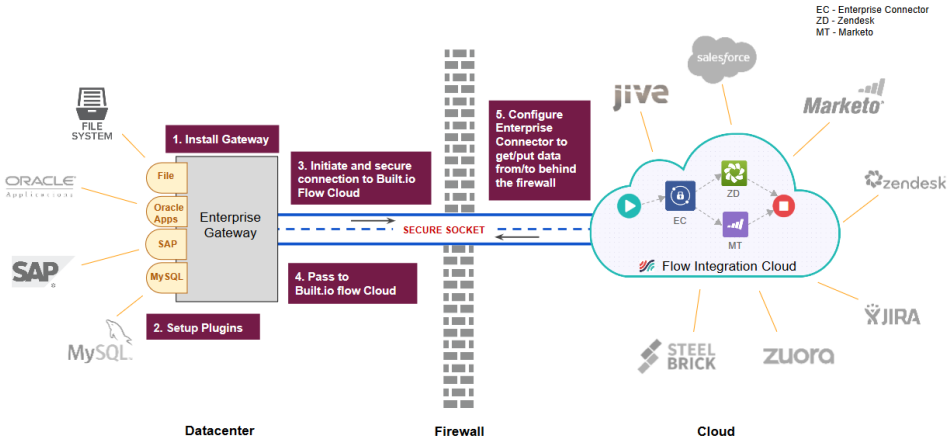

Figura C.3: Esquema del producto *Built.io Flow* [82]

acerca de microservicios y la evolución de los ESB [83] pero no detallan la estructura interna de los productos *Built.io Flow*. En otro aspecto, la plataforma soporta algunos patrones básicos pero no se enfoca en los EIP ni los referencia directamente.

C.1.3 Informatica

La empresa Informatica[4] propone un producto de iPaaS llamado "*Informatica Cloud Integration*". Su metodología de integración difiere un poco de sus competidores. No está tan fuertemente basada en flujos, aunque si ofrece la funcionalidad.

Para dar soporte a integración de sistemas internos, la plataforma utiliza un concentrador de fuentes de datos llamado "*Secure Agent*", el cual se despliega en la red interna de la empresa. Puede verse un esquema en la Figura C.4.

Se observa en su versión de prueba, una interfaz gráfica más compleja que otras iPaaS. Los usuarios configuran primeramente conexiones a fuentes de datos, con la opción de usar conectores prediseñados y conexiones a través del componente *Secure Agent*. Luego, se puede crear tareas de integración, o bien crear un flujo con varias de ellas.

Tanto Gartner como el propio Informatica afirman que la arquitectura de la plataforma está basada en microservicios [78, 85]. Informatica lo destaca como característica que le permite entregar mejoras y nuevos desarrollos más rápidamente, así como también acelerar las integraciones con arquitecturas de clientes [85].

Por otro lado, en la documentación del conector "Secure Agent" se afirma que este usa mi-

[4] https://www.informatica.com/

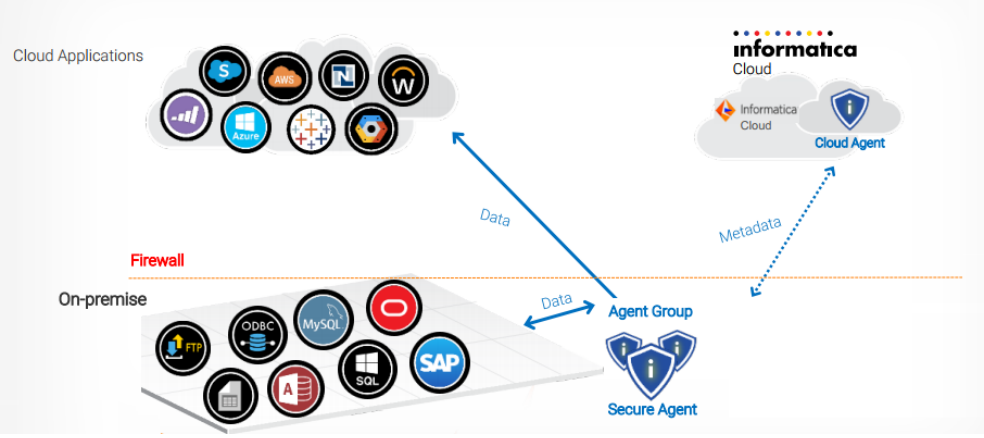

Figura C.4: Escenario de integración híbrido con *Informática Cloud Integration* [84].

croservicios conectables para procesamiento de datos. Segun la empresa esto le permite publicar actualizaciones manteniendo la disponibilidad [84].

Respecto al uso de los EIP, no se hace referencia a ellos directamente en la documentación ni tampoco se observó su implementación en pruebas de concepto hechas. Aún así, en las tareas de integración se poseen de forma prediseñada algunos EIP. Esta observación fue también hecha en el ya citado trabajo de Ritter et. al. [58] donde se identifica que la plataforma implementa parcialmente múltiples EIP.

C.1.4 MuleSoft

La empresa MuleSoft[5] propone un producto de iPaaS llamado "*Cloudhub*". Los usuarios pueden crear flujos de integración en la plataforma, a través de un entorno de desarrollo llamado "*AnyPoint Studio*".

Si bien MuleSoft está visto por Gartner como uno de los líderes en materia de iPaaS, no se encontró evidencia en las documentaciones de MuleSoft que indiquen que su arquitectura está basada en microservicios [86].

Por otro lado, *Cloudhub* da un mayor soporte a los EIPs que sus competidores. En su documentación, incluye un paralelismo entre componentes de integración propios y los EIP [87].

[5] https://www.mulesoft.com/

C.2 Productos de Integración On-Premise

En esta sección se muestra otro producto de integración interna considerado llamado "*Fiorano Integration Platform*".

La empresa Fiorano[6] propone una suite de integración interna llamada "*Fiorano Integration Platform*" la cual está basada en un ESB llamado "*Fiorano ESB*". Un esquema puede verse en la Figura C.5.

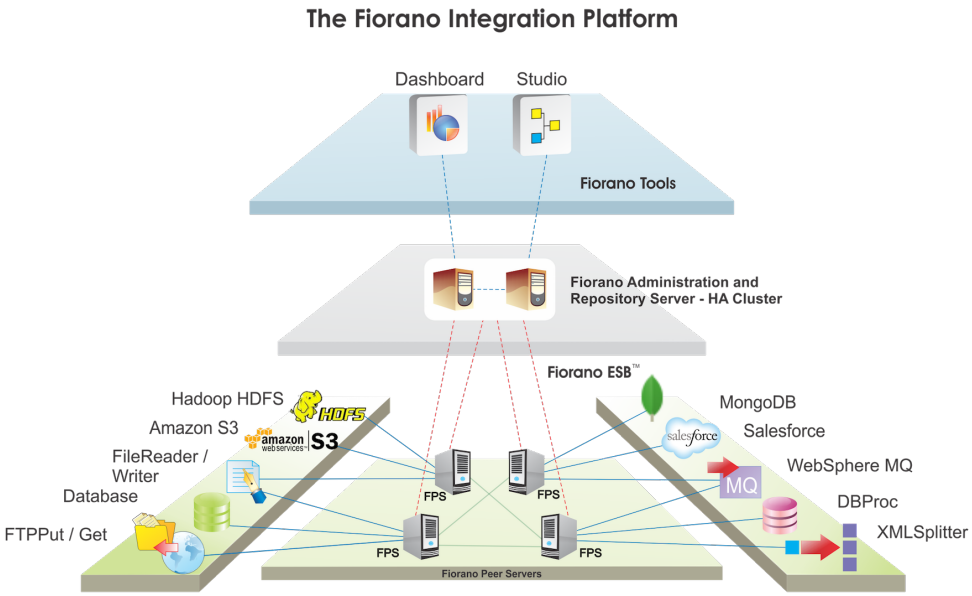

Figura C.5: Esquema general de la PI de Fiorano [88].

La PI de acuerdo a su documentación [89], está basada en una arquitectura punto-a-punto (i.e. *peer-to-peer*) en donde servidores de Fiorano hospedan microservicios que se comunican entre sí.

Los componentes de integración en Fiorano, están construidos como microservicios que se comunican por JMS. Estos siguen los lineamientos descritos por Fowler respecto a simplicidad en las conexiones. Además, se basan en mensajería que puede ser administrada por productos externos como *Apache ActiveMQ*[7].

En el sitio oficial de *Enterprise Integration Patterns* [90], mantenido por Gregor Hophe, se afirma que los EIP son aplicables al producto Fiorano (probablemente refiera a Fiorano ESB,

[6] http://www.fiorano.com/
[7] https://activemq.apache.org/

pero no especifica). Aun así, no se menciona si Fiorano implementa o no en la práctica los EIP. En lo que puede observarse en su documentación [91], Fiorano implementa algunos de los EIP básicos como separador (i.e. *Splitter*), agregador (i.e. *Aggregator*), transferencia de archivos (i.e. *File transfer*) y ruteo basado en contenido (i.e. *Content-based Router*).

D

Impacto de Características de Escenario sobre la PI

Este apéndice describe el análisis hecho para determinar el impacto de las características de los escenarios de integración sobre las PIs. A partir de este, se determina que subatributos de la PI constituyen una preocupación, para cada característica de escenario. Es decir, se determina las preocupaciones de calidad por característica de escenario. El resumen del análisis, se muestra en la Sección 3.3.

El apéndice se organiza en base a las categorías presentadas en la Sección 3.2, mostrando para las características de cada categoría su impacto en la calidad de la PI.

D.1 Impacto de la ubicación de la PI

Se considera que la plataforma de integración puede ser desplegada en una nube pública, en una nube privada, en un servidor interno o de forma híbrida. El impacto de esta decisión sobre los atributos de calidad de la PI, se analiza en las siguientes secciones (D.1.1 a D.1.3).

D.1.1 Despliegue en Nube Pública

Si la plataforma de integración es desplegada en una nube pública, sobre una plataforma de orquestación de contenedores sobre clústeres (se asume un servicio de PaaS, ver Sección 3.2.1), los siguientes subatributos de calidad son afectados:

- *Uso de Recursos*: Es afectado positivamente por las funcionalidades de escalamiento elástico automatizado que poseen las plataformas de orquestación de contenedores. Estas permiten aumentar o disminuir las réplicas de los servicios de la PI a demanda y hacer un uso eficiente de los recursos.

- *Capacidad*: Es afectada positivamente, debido a que los proveedores de infraestructura de nube usualmente facilitan la disponibilidad de mayores recursos de *hardware* a demanda. También es usual los servicios de pago por recursos usados.
- *Adaptabilidad*: En el estándar considerado, la adaptabilidad incluye la escalabilidad del sistema, las cuales se ven afectadas positivamente por razones análogas a las presentadas en el subatributo *Uso de Recursos*.
- *Disponibilidad*: Es afectada positivamente, ya que los proveedores de infraestructura en la nube ofrecen alta disponibilidad para los sistemas desplegados.
- *Tolerancia a Fallas*: Es afectada positivamente por la tolerancia a fallas de *hardware* (por redundancia) que posee usualmente la infraestructura de nube.
- *Recuperabilidad*: Es afectada positivamente por dos aspectos: i) por motivos análogos al subatributo *Tolerancia a Fallas*, y ii) por la capacidad de las plataformas de orquestación de supervisar y mantener la operatividad de los servicios. La plataforma controla que cierta cantidad de réplicas del servicio están activas, en caso de falla, reinicia el contenedor mientras otras réplicas mantienen operatividad.
- *Seguridad (Todos los subatributos)*: Constituye una preocupación, ya que la PI es más vulnerable en la nube pública que en la red interna. Esto se debe a que está más expuesta, tanto al público como al proveedor de infraestructura. Por otro lado, se posee menos control sobre el acceso de red, ya que *firewalls* y *proxys* corren por cuenta de terceros y no están bajo el control empresarial.

D.1.2 Despliegue en Nube Privada

Si la plataforma de integración es desplegada en una nube privada, sobre una plataforma de orquestación de contenedores sobre clústeres, los siguientes subatributos son afectados:

- *Uso de Recursos*: Análogo al caso de nube pública, por ser desplegado sobre una plataforma de orquestación de contenedores sobre clústeres.
- *Capacidad*: A diferencia del despliegue en nube pública, depende de los recursos de *hardware* disponibles de la empresa. Se considera que los recursos internos suelen ser menores que los de proveedores de nube pública, y por ende, el despliegue implica una preocupación sobre el subatributo.
- *Adaptabilidad*: Análogo al caso de nube pública, por ser desplegado sobre una plataforma de orquestación de contenedores sobre clústeres.
- *Disponibilidad*: Asumiendo que la PI no es distribuida geográficamente, está sujeta a fallas o siniestros que puedan ocurrir dentro de la propia empresa (p. ej. cortes de luz sostenidos, incendios, entre otros). Por tanto constituye una preocupación.

- *Tolerancia a Fallas*: Análogo al caso de nube pública.
- *Recuperabilidad*: Análogo al caso de nube pública, por ser una plataforma de orquestación de contenedores sobre clústeres y porque se asume redundancia de *hardware*.
- *Seguridad (Todos los subatributos)*: Es afectada positivamente en comparación con el despliegue en nube pública. Al ser un despliegue interno, se tiene control directo sobre la plataforma, su hardware, los accesos de red y sus intermediarios (p. ej. *proxys* y *firewalls*).

D.1.3 Despliegue en Servidores Internos

Si la plataforma de integración es desplegada en servidores internos, es decir sin una plataforma de orquestación de contenedores sobre clústeres, los siguientes subatributos son afectados:

- *Uso de Recursos*: Constituye una preocupación, ya que si no se posee escalamiento elástico, se debe disponer todo el tiempo de la cantidad de recursos suficiente para soportar el peor caso de demanda.
- *Capacidad*: Constituye una preocupación, por motivos análogos al caso de despliegue en nube privada.
- *Adaptabilidad*: Constituye una preocupación ya que no posee capacidad de escalamiento elástico.
- *Disponibilidad*: Constituye una preocupación, por motivos análogos al caso de despliegue en nube privada.
- *Tolerancia a Fallas*: Constituye una preocupación, ya que no posee redundancia de *hardware*.
- *Recuperabilidad*: Constituye una preocupación, ya que no posee redundancia de hardware ni las ventajas de recuperabilidad de las plataformas de orquestación de contenedores sobre clústeres.
- *Seguridad (Todos los subatributos)*: Es afectada positivamente, por razones análogas al despliegue en nube privada.

D.1.4 Despliegue Híbrido

En este caso, se asume que la PI es un sistema distribuido, donde una parte del sistema se despliega de forma interna y otra parte en una nube pública.

El impacto analizado, será únicamente relacionado al despliegue y no a características de los

sistemas distribuidos ya que no son características del escenario sino de la PI misma[1]. Por ejemplo, la evaluabilidad en los sistemas distribuidos constituye una preocupación, pero se omite por no estar relacionado al despliegue.

Los siguientes subatributos son afectados:

- *Comportamiento del tiempo*: Depende de la distribución de los componentes. El comportamiento podría ser afectado positivamente si se tiene en cuenta la ubicación de los datos importantes (i.e. *data gravity*) para la distribución de los componentes de la PI que los procesan. En dicho caso, podría reducirse las comunicaciones al procesar los datos mas cerca de su ubicación, lo cual mejoraría el desempeño del tiempo. Esto se profundiza en la Sección 4.2.5.
- *Uso de Recursos*: En base a secciones anteriores, se puede analizar el subatributo para los componentes internos y públicos de forma separada. Se observa que los componentes internos podrían ser una preocupación, pero su criticidad está en parte asociada a cómo se distribuyan los componentes de la PI.
- *Capacidad*: Análogo al subatributo *Uso de Recursos*.
- *Adaptabilidad*: Análogo al subatributo *Uso de Recursos*.
- *Instalabilidad*: Es una preocupación, ya que se debe instalar la PI en más de un ambiente y además configurar la conexión entre los mismos.
- *Disponibilidad*: Análogo al subatributo *Uso de Recursos*.
- *Tolerancia a Fallas*: Análogo al subatributo *Uso de Recursos*.
- *Recuperabilidad*: Análogo al subatributo *Uso de Recursos*.
- *Seguridad (Todos los subatributos)*: En base a secciones anteriores, se puede analizar el subatributo para los componentes internos y públicos de forma separada. Se observa que los componentes públicos podrían ser una preocupación.

D.2 Impacto por Ubicación de los Sistemas a Integrar

La ubicación de los sistemas a integrar, en contraposición con la ubicación de la PI, afecta dos subatributos de calidad: la instalabilidad y el comportamiento del tiempo.

El impacto en el comportamiento del tiempo es complejo de determinar. El caso de una única empresa (i.e. intraorganizacional) se puede precisar bajo ciertas condiciones, como se muestra en la Sección D.2.1. En el caso de varias empresas (i.e. interorganizacional), es más complejo por las múltiples redes internas, y depende de la cantidad de sistemas y su

[1] Este impacto se estudia con el de arquitectura de microservicios en la Sección 3.4.

distribución. Se profundiza en las siguientes secciones.

D.2.1 Ubicación en Escenarios Intraorganizacionales

En escenarios intraorganizacionales es mas fácil predecir el impacto de la ubicación de los sistemas a integrar sobre la PI. Las variantes posibles se dividen en tres según si la PI y sistemas a integrar están o no en la misma red. Se analiza a continuación cada caso.

D.2.1.1 Sistemas a Integrar y PI en misma Red

Este caso se da si tanto la PI como los sistemas a integrar están desplegados en la red interna, o si por el contrario, todos están desplegados en la nube. Se afectan los siguientes subatributos:

- *Comportamiento del Tiempo*: Se ve afectado positivamente, ya que al estar todos los sistemas desplegados en la misma red, los tiempos de respuesta deberían ser menores. Notar que si el despliegue es interno, podría considerarse incluso disminuir la seguridad para mejorar la latencia.

  En el caso de despliegue en la nube, se asumirá que es el mismo proveedor de infraestructura y que todos los sistemas están en la misma región.

- *Instalabilidad*: Se ve afectada positivamente, ya que no hay que hacer configuraciones para exponer los sistemas a integrar (p. ej. configurar un *reverse proxy*).

D.2.1.2 Sistemas a Integrar y PI en distinta Red

Corresponde a escenarios donde los sistemas a integrar no se encuentran en la misma red que la PI. Tal es el caso si la PI está desplegada en la nube y los sistemas a integrar en una red interna o viceversa[2]. Se afectan los siguientes subatributos:

- *Comportamiento del Tiempo*: Constituye una preocupación, ya que se asume una mayor latencia entre la PI y los sistemas a integrar. Esta latencia puede empeorar si la nube se encuentra en otra región geográfica que los sistemas a integrar, y si estos últimos son expuestos mediante un intermediario (i.e. *reverse proxy*).

- *Instalabilidad*: Constituye una preocupación, debido a que se debe realizar un mayor esfuerzo de configuración para exponer los sistemas que estén en la red interna (tanto si se trata de la PI en la red interna como si se trata de los sistemas a integrar) y también para conectar la PI con los sistemas a integrar.

[2] Este caso podría ocurrir si, por ejemplo, la empresa tiene políticas de control estricto que exijan que la PI esté desplegada internamente.

D.2.1.3 Sistemas a Integrar en la Nube y en Red Interna

Si los sistemas a integrar están distribuidos entre la nube y la red interna se afectan los siguientes subatributos:

- *Comportamiento del Tiempo*: Es complejo estimar el impacto sobre la latencia y determinar cual es la ubicación más efectiva para la PI. Notar que la ubicación de la PI debe minimizar la latencia a los sistemas a integrar, sin embargo esto no implica necesariamente desplegar la PI donde estén la mayoría de los sistemas, ya que hacer eso podría aumentar demasiado la latencia a otro sistema crítico. El impacto por tanto es indeterminado.
- *Instalabilidad*: Constituye una preocupación, por razones análogas al caso de sistemas a integrar y PI ubicados en distinta red.

D.2.2 Ubicación en Escenarios Interorganizacionales

Los escenarios interorganizacionales son más complejos debido a que se tienen múltiples redes internas. En este caso, la PI puede estar desplegada en alguna de las redes internas o bien en una nube pública. El impacto sobre el *Comportamiento del Tiempo*, dependerá de varios factores que se explican a continuación. La instalabilidad, se analiza de forma separada en la Sección D.5

A diferencia del caso intraorganizacional, se analiza las posibles opciones de despliegue para la PI en forma independiente, en base a la ubicación de los sistemas a integrar.

D.2.2.1 Sistemas a Integrar en Redes Internas

En contraste al caso intraorganizacional, se necesita determinar no solo si es preferible un despliegue de la PI en la nube o en la red interna, sino además en cuál de las posibles redes internas.

A priori, no está claro que un despliegue de la PI en la nube tenga peor latencia que un despliegue en red interna en este escenario. Por ejemplo, si las redes internas están en distintos continentes, podría ser mas eficiente desplegar la PI en una nube con menor latencia hacia cada red interna por separado. Depende en gran parte de la solución de integración definida.

En definitiva, el impacto y conveniencia del despliegue de la PI no queda determinado. Este depende de la cantidad de sistemas, su ubicación, su rol en la integración y su latencia individual.

D.2.2.2 Sistemas a Integrar en la Nube y en Redes Internas

Si se tienen sistemas a integrar dispersos entre la nube y redes internas, depende de factores similares a los vistos en el caso intraorganizacional. No se puede estimar de forma abstracta el impacto ni el despliegue mas conveniente para la PI, sin determinar primero la cantidad, ubicación, rol en la integración y latencia individual de los sistemas a integrar.

D.2.2.3 Sistemas a Integrar en la Nube

Si los sistemas a integrar están en la nube, entonces es conveniente que el despliegue de la PI sea realizado también en la nube, en cuyo caso se afecta positivamente el *Comportamiento del Tiempo*, en comparación con desplegar la PI en redes internas.

D.3 Impacto por Particularidades de los Sistemas a Integrar

Se estudia en esta sección, dos particularidades que pueden afectar los atributos de calidad de la PI: i) la madurez de los sistemas a integrar, y ii) la cantidad de sistemas a integrar.

D.3.1 Madurez de los Sistemas a Integrar

Si los sistemas a integrar son inmaduros o cambian con frecuencia, existen implicancias sobre los subatributos de calidad de la PI. Principalmente porque cambia la forma de integración, y porque es probable que la PI deba acompasar los cambios y problemas que surjan del mantenimiento de los sistemas a integrar.

Ejemplos de inmadurez o volatilidad son sistemas a integrar que se encuentran aún en versiones iniciales. El software como servicio (*SaaS*) también puede considerarse un ejemplo, ya que suele ser ofrecido bajo un marco de despliegue continuo, lo cual suele implicar que el sistema tiene una evolución activa.

Se describen a continuación los subatributos afectados:

- *Modularidad*: Constituye una preocupación, ya que se necesita que la PI tenga una correcta modularidad para acotar el impacto de los cambios y facilitar el mantenimiento.
- *Evaluabilidad*: Constituye una preocupación, ya que es más complejo evaluar el impacto de una modificación puntual en la PI, ya que los cambios podrían provocar comportamientos inesperados en la integración, por ser inestables los sistemas a integrar.
- *Modificabilidad*: Constituye una preocupación. De forma análoga al caso de la evaluabilidad, es dificultoso hacer modificaciones en la PI, ya que los cambios pueden pro-

ducir errores o comportamientos inesperados en la integración, por la inmadurez de los sistemas a integrar.

- *Madurez*: Constituye una preocupación. Si los sistemas a integrar son volátiles y cambian frecuentemente, es probable que la PI también deba ser modificada regularmente para acompasar dichos cambios. Por tanto si la PI es cambiante, es posible que se afecte su madurez.

  Un ejemplo es que un sistema a integrar cambie en parte la forma de comunicación, lo cual implique cambiar los conectores específicos. Otro ejemplo, es que se realice una transformación compleja (por ejemplo con un componente personalizado) y que luego cambios en la estructura de datos impliquen cambios en el módulo de transformación.

- *Tolerancia a Fallas*: Constituye una preocupación, ya que la inmadurez de los sistemas aumenta la diversidad y frecuencia de las posibles fallas. Lo cual implica una necesidad de tolerancia mayor por parte de la PI.

- *Recuperabilidad*: De forma análoga a la tolerancia a fallas, constituye una preocupación debido a la probabilidad mayor de fallas que pueden afectar la operatividad de la PI.

D.3.2 Cantidad alta de sistemas a integrar

Si la cantidad de sistemas a integrar es elevada, existen implicancias sobre los atributos de calidad de la PI. Además, el impacto es más acentuado a medida que el número de sistemas a integrar aumenta. A efectos prácticos, se considera en este trabajo más de cinco sistemas a integrar como número alto, lo cual es propuesto a criterio del autor.

Los subatributos afectados son los siguientes:

- *Comportamiento de Tiempo*: Constituye una preocupación, debido a que la latencia entre la PI y cada sistema a integrar podría acumularse. Esto implica que a medida que aumenta la cantidad de sistemas el tiempo total de la integración se ve afectado. Por otro lado, la cantidad de recursos necesaria para integrar todos los sistemas también aumenta, lo cual puede conllevar a una degradación del desempeño.

- *Uso de Recursos*: Constituye una preocupación, ya que más cantidad de sistemas a integrar demandan más recursos de la PI.

- *Modularidad*: Constituye una preocupación, ya que dado el número elevado de sistemas potencialmente heterogéneos, es necesaria una buena modularidad en la PI, que permita mantener eficazmente las integraciones a dichos sistemas.

- *Evaluabilidad*: Constituye una preocupación, debido a que cuantos más sistemas deben integrarse, más complejo es determinar el impacto que un cambio en la PI tendrá sobre las integraciones implementadas.

- *Modificabilidad*: Constituye una preocupación, ya que la alta cantidad de sistemas dificulta modificar la PI, porque los cambios pueden introducir errores no esperados en alguno de los sistemas a integrar.
- *Instalabilidad*: Constituye una preocupación, ya que para las soluciones de integración, se deben configurar la conexión de red a todos los sistemas a integrar, de forma de conectarlos con la PI. Al crecer la cantidad de sistemas, dicha tarea se hace más extensa.
- *Reemplazabilidad*: Constituye una preocupación, ya que si el número de sistemas a integrar es elevado dificulta reemplazar la PI, especialmente si las integraciones a los sistemas son personalizadas y no implican protocolos simples. Dada la complejidad de configuración ya mencionada, es importante poder reemplazar solo las partes necesarias de la PI.
- *Tolerancia a Fallas*: Constituye una preocupación, ya que se posee una cantidad alta de sistemas heterogéneos, que aumentan la probabilidad y diversidad de errores en la PI.
- *Recuperabilidad*: Constituye una preocupación, ya que se diversifican los errores posibles sobre los cuales recuperarse, por motivos análogos al caso de tolerancia a fallas.

D.4 Impacto por particularidades de los datos a integrar

Algunas particularidades de los datos a integrar pueden afectar los atributos de calidad de la PI. En este trabajo, se estudian dos factores: i) masividad de datos o de accesos de usuario, y ii) complejidad de los datos.

D.4.1 Masividad de datos o de accesos de usuario

Esta característica refiere a escenarios donde se requiere procesar una cantidad masiva de datos en forma concurrente, o donde haya un número elevado de usuarios usando la PI y sus servicios al mismo tiempo.

Los subatributos afectados son los siguientes:

- *Comportamiento del Tiempo*: Constituye una preocupación, ya que la masividad de datos y de accesos genera una alta exigencia sobre los recursos que puede degradar el desempeño. Por otro lado, si los recursos son escalables, se debe considerar las demoras para efectuar el escalamiento horizontal a demanda (p. ej. el tiempo de iniciar nuevas réplicas de contenedores).
- *Uso de Recursos*: Constituye una preocupación, ya que la cantidad de datos y accesos consume de forma importante los recursos. Además, si la cantidad es muy fluctuante,

puede tener como consecuencia que la elasticidad del escalamiento provoque un mal uso de recursos en períodos breves, mientras el escalamiento automático acompasa el cambio en la cantidad de datos o de accesos.

- *Capacidad*: Constituye una preocupación por motivos análogos a los presentados en *Uso de Recursos*.
- *Adaptabilidad*: Constituye una preocupación, ya que se requiere que la PI pueda escalar a demanda y adaptarse a los picos de acceso y de datos concurrentes que puedan ocurrir.
- *Disponibilidad*: Constituye una preocupación, debido a la visibilidad[3] que tiene la PI en el caso de masividad de accesos; y a la cantidad de integraciones que fracasarían si la PI no está operativa en el caso de masividad de datos.
- *Tolerancia a Fallas*: Constituye una preocupación, debido a que la alta cantidad de datos y accesos implica una mayor probabilidad de errores, por lo cual se requiere mayor tolerancia a fallas por parte de la PI para mantener la operatividad del sistema.
- *Recuperabilidad*: Constituye una preocupación por motivos análogos al subatributo *Tolerancia a Fallas*.

D.4.2 Complejidad de los datos

Se estudia el impacto de la complejidad de los datos. Esto refiere en este trabajo a si los datos requieren de un procesamiento complejo y computacionalmente caro o si necesitan de bibliotecas y tecnología especializada para su integración.

Los subatributos afectados son los siguientes:

- *Comportamiento del Tiempo*: Si es computacionalmente caro procesar los datos, es muy probable que se incurra en cierta latencia para procesar los pedidos, por lo cual el subatributo constituye una preocupación.
- *Uso de Recursos*: Constituye una preocupación, ya que por ser computacionalmente caro procesar los datos, se usarán y necesitarán más recursos.
- *Capacidad*: Constituye una preocupación por motivos análogos al caso de uso de recursos.
- *Reemplazabilidad*: Constituye una preocupación, ya que el hecho de que los datos requieran tecnología específica o procesamiento complejo, implica que la PI es más difícil de reemplazar.
- *Tolerancia a Fallas*: Constituye una preocupación, ya que por la latencia, es posible

[3] Refiere a la exposición a un gran número de personas que se verían afectadas en caso de falla.

que las soluciones de integración incurran en errores de caducidad de pedidos (i.e. *request timeout*), por tanto se requiere de un correcto manejo de errores y tolerancia a fallas. Además, la complejidad de los datos puede causar errores no esperados, especialmente si se involucran tecnologías especializadas.

- *Recuperabilidad*: Análogo al caso de tolerancia a fallas, constituye una preocupación, ya que la PI debe poder recuperarse a fallas de caducidad de pedidos y de procesamiento de los datos.

D.5 Impacto en escenarios interorganizacionales

Los escenarios de integración donde participan sistemas de varias empresas, tienen un grado mayor de complejidad que afecta los atributos de calidad de la PI.

En particular, la complejidad proviene de la alta interacción y necesidad de formalidades y aprobaciones de las múltiples partes, además de la heterogeneidad de los sistemas y del conocimiento general sobre ellos.

Los subatributos afectados son los siguientes:

- *Modularidad*: Constituye una preocupación, debido a que hay varios sistemas involucrados de diferentes partes interesadas. Esto requiere la correcta separación de los componentes, especialmente los conectores específicos de los distintos sistemas a integrar.
- *Evaluabilidad*: Constituye una preocupación, debido a que hay más empresas y sistemas heterogéneos, cuyo comportamiento no siempre es entendido en su completitud por todas las partes, haciendo más dificultoso evaluar el impacto de los cambios.
- *Modificabilidad*: Constituye una preocupación, debido a dos factores: i) se requiere integrar sistemas heterogéneos de diferentes empresas, y ii) muchas partes involucradas implica la necesidad de comunicación constante, y comúnmente de distintas aprobaciones para lograr modificaciones.
- *Testabilidad*: Constituye una preocupación por motivos análogos a los subatributos anteriores.
- *Instalabilidad*: Constituye una preocupación, debido a que se debe configurar conexiones y componentes en múltiples redes internas (para los distintos sistemas a integrar), con políticas empresariales diferentes. Si la PI es compleja de instalar y conectar, el proyecto puede fracasar o irse de costos.
- *Reemplazabilidad*: Constituye una preocupación, debido a que al haber más partes involucradas posiblemente haya más requerimientos implementados o diversidad en estos, mayor necesidad de evolución, y mayor configuración específica a los distintos

sistemas. Esto implica que la PI sea difícil de reemplazar.

- *Madurez*: Constituye una preocupación por motivos análogos a la reemplazabilidad. Al estar sometida a implementaciones y cambios heterogéneos, es más difícil alcanzar la madurez de la PI.
- *Seguridad (Todos los subatributos)*: Constituye una preocupación, debido a que la información es compartida y modificada por múltiples empresas. Su importancia es mayor si las empresas poseen información sensible o si no tienen un historial de colaboración entre ellas.

E

Casos de Uso Complementarios de PI

Se muestra en este apéndice una vista completa de los casos de uso considerados para PIs. En esta se incluyen algunos casos que fueron omitidos en el Capítulo 4 por poseer una relevancia menor. El diagrama completo puede verse en la Figura E.1.

Los casos omitidos pertenecen a tres módulos concretos: *Gestión de SI*, *Gestión de Componentes Personalizados* y *Administrar SI*. El primero, adiciona los casos Modificar SI y Eliminar SI, que forman parte del ciclo de vida de las soluciones de integración. El segundo, es un módulo que no se muestra en el Capítulo 4 y que considera la gestión de los componentes personalizados, que constituyen una forma de extensión de los componentes de integración predefinidos. El tercero, adiciona el caso de Retirar SI, para quitar una SI que está desplegada y en ejecución.

En relación al caso de Eliminar SI, involucra la eliminación del código y configuraciones construidas para la SI. Se destaca que si la SI está desplegada no necesariamente involucra retirar la misma, la cual podría mantenerse en ejecución.

Respecto a los componentes personalizados, los administradores pueden crear sus propios componentes de integración (i.e. Componentes Personalizados) para extender las funcionalidades de la PI. Estos son almacenados de forma diferenciada y una vez agregados a la PI, pueden usarse en la construcción de SIs de igual forma que los componentes nativos.

En relación al caso de Retirar SI, corresponde a detener la ejecución de una SI y remover la misma de la plataforma de ejecución, de forma de que no consuma recursos. Al finalizar el caso la SI ya no responderá a eventos disparadores ni estará en ejecución, hasta que no sea desplegada nuevamente.

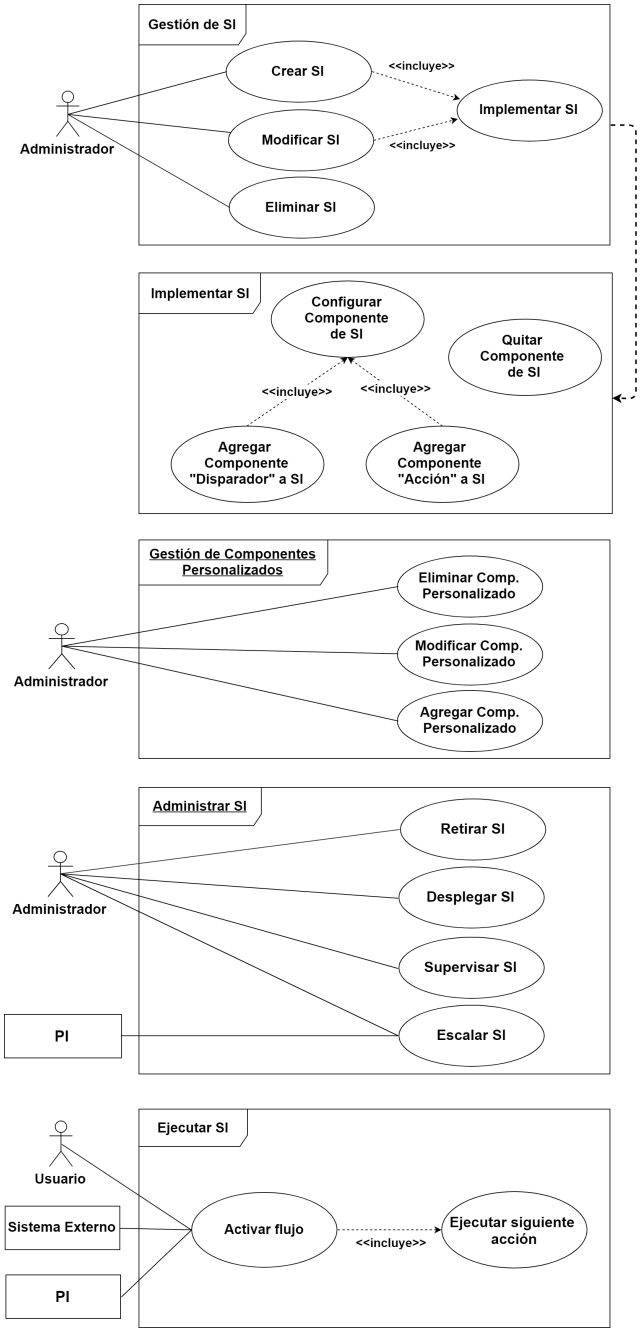

Figura E.1: Vista de todos los Casos de Uso de la PI

F

Variante de Orquestación (Vistas Complementarias)

En esta sección, se presentan las vistas de la variante de orquestación que no fueron mostradas en la Sección 4.3.1 y que difieren con la propuesta principal.

Como se menciona en el Capítulo 4, la variante modifica la forma de coordinación de los microservicios, introduciendo un nuevo componente (el orquestador) y modificando aspectos como la ejecución y despliegue de las SIs.

F.1 Vista Lógica de Bajo Nivel

En la Figura F.1 se presenta el diagrama de bajo nivel que muestra de forma holística la vista lógica. Se observa que la vista lógica de alto nivel no sufre cambios, y que el desglose de vista lógica se muestra en la Sección 4.3.1.1.

Respecto al diagrama, si bien algunas vinculaciones se mantienen igual a la vista coreográfica, se observa que el orquestador requiere de los componentes de integración para ejecutar la SI, a los cuáles invoca en forma ordenada.

Por otro lado se observa que los servicios web de la entidad *Virtualización de Servicios* y los conectores específicos, activan la SI al reaccionar a un evento, y requieren del orquestador para continuar la ejecución de la SI.

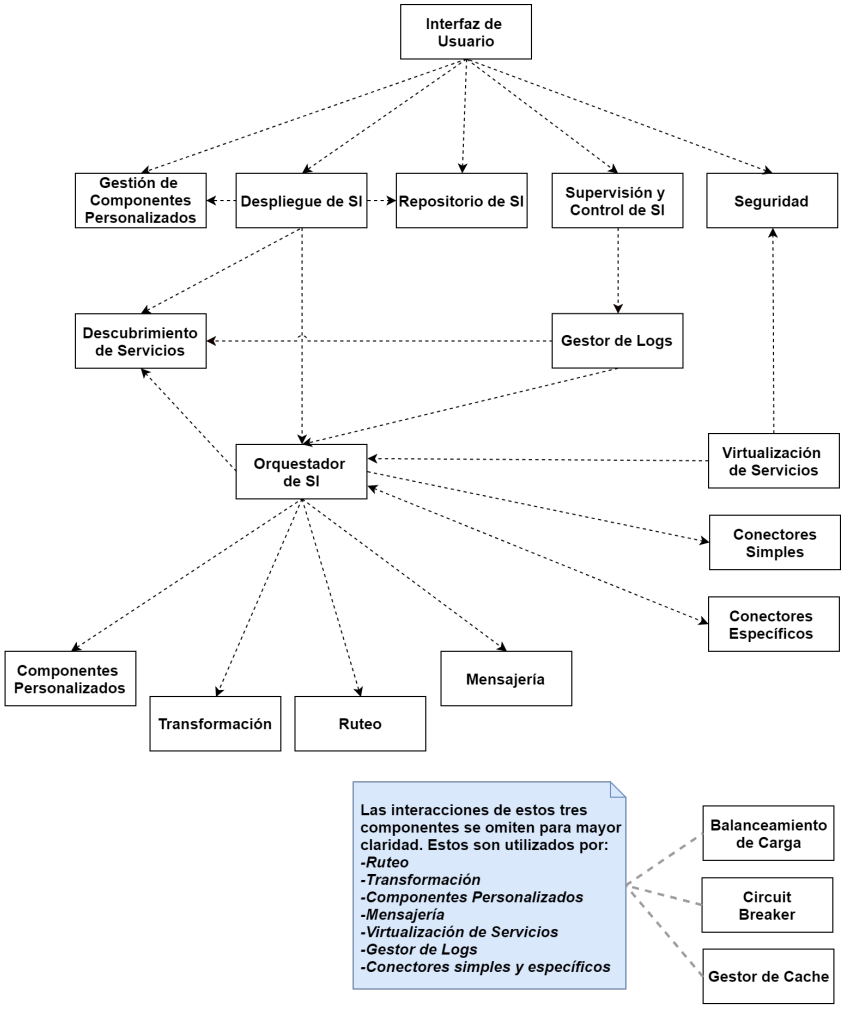

Figura F.1: Vista Lógica holística para el desglose (Orquestación)

F.2 Vista de Desarrollo

En la Figura F.2 se muestra la vista de desarrollo, la cual es similar a la coreográfica con la diferencia del orquestador y sus referencias. Se observa que el modulo central está vinculado al *Orquestador de SI*, debido a las vinculaciones de la entidad *Despliegue de SI*, mostradas en la vista lógica.

Por otro lado el orquestador requiere potencialmente de todos los componentes de integración para ejecutar SIs. Sin embargo dicha vinculación se omite en el diagrama.

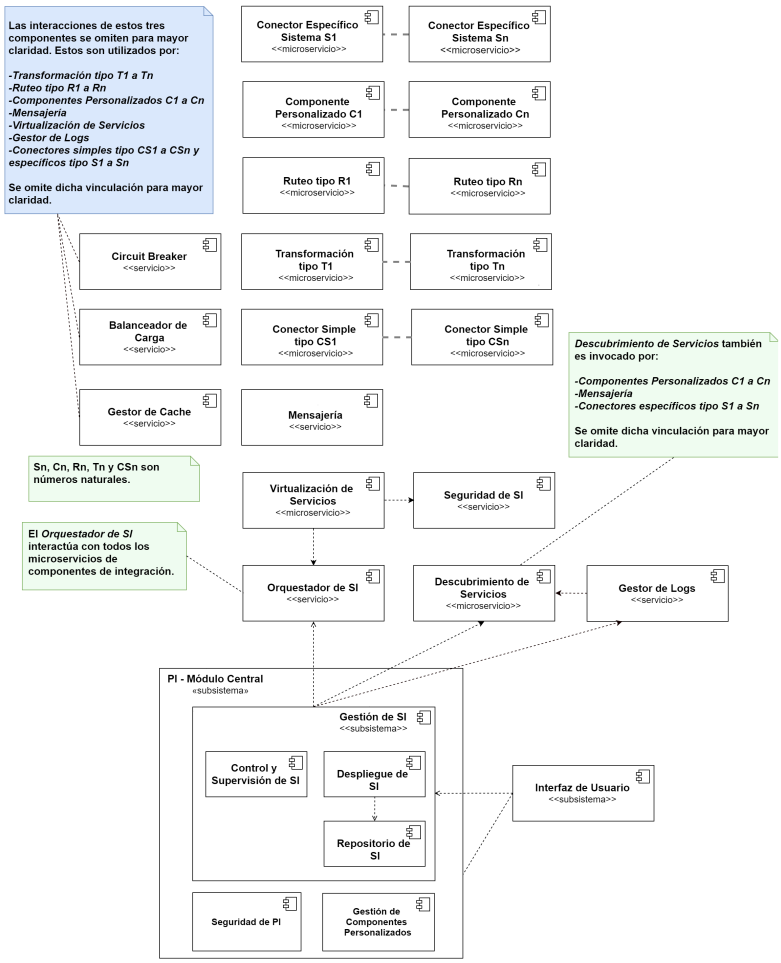

Figura F.2: Vista de desarrollo, ejecución de SI orquestada

F.3 Vista de Despliegue

En la Figura F.3 se muestra la vista de despliegue en el caso de orquestación, la cual es análoga al caso coreográfico pero desplegando el componente de *Orquestación de SI*, el cual es instanciado y opera de forma dedicada para cada SI.

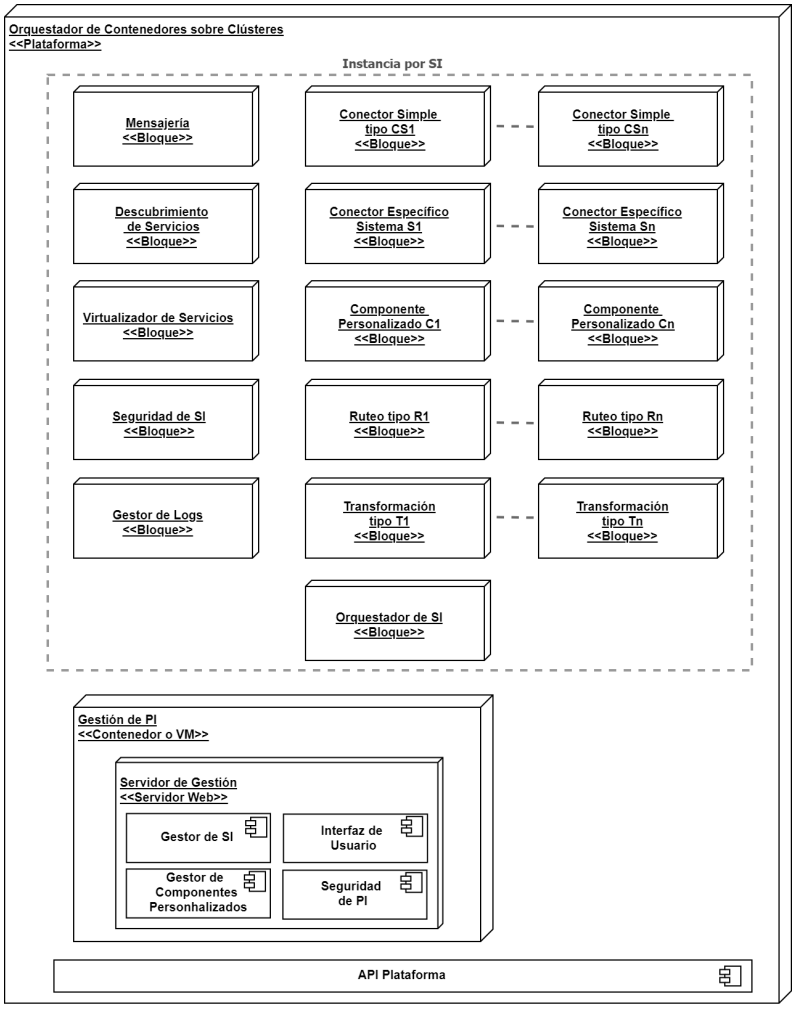

Figura F.3: Vista de despliegue (Orquestación)