Servicios *adaptive-ready* para la reconfiguración dinámica de arquitecturas de microservicios*

Joan Fons[1][0000−0002−3718−3096], Vicente Pelechano[1][0000−0003−1090−230X], Manoli Albert[1][0000−0003−3747−400X], and Miriam Gil[2][0000−0002−2987−1825]

[1] Centro de Investigación PROS, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain {jjfons,pele,malbert}@pros.upv.es
[2] Departament d'Informàtica, Universitat de València, Avenida de la Universidad s/n, 46100 Burjassot, Spain miriam.gil@uv.es

Resumen Las arquitecturas de microservicios ofrecen un enfoque para la ingeniería de sistemas complejos y distribuidos en donde la escalabilidad y disponibilidad de las soluciones son un factor crítico. En la última década han surgido herramientas (orquestadores) orientadas a monitorizar y mantener de manera automática estas infraestructuras para garantizar la calidad del servicio. Sin embargo, estos orquestadores no están diseñados para gestionar automáticamente cambios arquitectónicos sobre la configuración de servicios, necesarios bajo ciertas condiciones operativas. Este trabajo promueve la aplicación de la computación autónoma (a través del uso de bucles de control) como estrategia para la reconfiguración dinámica de arquitecturas de microservicios. Para ello, se introduce el concepto de *servicio adaptive-ready*, como un microservicio que ofrece un contrato de adaptación diseñado para ser usado por los bucles de control. Para ejemplificar la propuesta se utiliza un caso de estudio real en el que se aplican estos principios para desarrollar el sistema informático de una fábrica de producción industrial. Se ha diseñado e implementado un conjunto de servicios *adaptive-ready* y se ha desplegado una arquitectura de microservicios reconfigurable sobre `kubernetes`.

Keywords: Arquitecturas de microservicios · Computación Autónoma · Servicios Web · Reconfiguración arquitectónica

1. Introducción

El desarrollo de sistemas inteligentes y auto-adaptativos es cada vez más complejo debido a la necesidad de interconectar y coordinar múltiples dispositivos y servicios distribuidos. Esta complejidad, junto con la alta disponibilidad que actualmente se demanda, hace que se requieran soluciones ágiles que faciliten la entrega e integración continua [6]. Las arquitecturas de microservicios son un estilo arquitectónico que ha ganado popularidad en este ámbito [2]. En estas arquitecturas, los sistemas software se dividen en componentes más pequeños (microservicios) que son

* Trabajo soportado por el MINECO en el proyecto PROTEUS TIN2017-84094-R

autónomos e independientes, y se comunican a través de mecanismos ligeros como llamadas REST o comunicaciones asíncronas de tipo *publish/subscribe*.

Debido a su naturaleza distribuida y al alto grado de dependencias entre los microservicios, es necesaria la gestión continua de los recursos computacionales requeridos para su despliegue y mantenimiento con garantías. Para simplificar su despligue, configuración y ejecución, los microservicios se empaquetan en contenedores. En este contexto, aparecen los 'orquestadores de contenedores' que monitorizan y controlan la salud, la disponibilidad o la carga de trabajo (uso de CPU, memoria, red, etc.) de estos contenedores. Cuando identifican un problema en alguno de estos parámetros, estos orquestadores toman acciones para parar y/o reiniciar contenedores, escalar (aumentar o disminuir) el número de instancias de contenedores en ejecución, etc. Sin embargo, estos orquestadores no ofrecen mecanismos para reconfigurar la arquitectura de los microservicios, es decir, cambiar los microservicios activos y las relaciones establecidas entre los mismos dinámicamente (en tiempo de ejecución). Esto provoca que en determinadas situaciones (causadas por fallos, por ejemplo) estas limitaciones dificulten tomar las acciones necesarias para solucionar el problema.

La computación autónoma [3,9] propone un marco de ingeniería para desarrollar sistemas con capacidades de auto-adaptación. Establece una estrategia basada en el uso de bucles de control externos al sistema, donde se implementan las políticas de adaptación. Estos bucles de control son los responsables de monitorizar, analizar, decidir y actuar sobre un sistema al que gestionan. La aplicación de los conceptos de la computación autónoma al dominio de las arquitecturas de microservicios introduce un reto: ¿cómo debería diseñarse e implementarse una arquitectura de microservicios para que facilite ser reconfigurada dinámicamente por las políticas de adaptación de un bucle de control? En este trabajo abordamos esta reconfiguración dinámica de los sistemas desde una perspectiva arquitectónica. Tomando como base el trabajo iniciado [8], en el que se propone una notación y un conjunto de operaciones de adaptación abstractas para modificar arquitecturas de componentes, refinamos esta propuesta para aplicarla al dominio de los microservicios. Por último, proporcionamos unas guías para desarrollar microservicios preparados para ser adaptados (*adaptive ready services* ó $\mathcal{ARS}$) así como infraestructura reutilizable para su desarrollo.

Este trabajo fue aplicado a una fábrica industrial que utiliza una arquitectura de microservicios (en el dominio de la Internet de las Cosas y las Fábricas del Futuro) para controlar y asistir a los procesos de producción. Se reconviertieron algunos microservicios en $\mathcal{ARS}$, y se desplegó un bucle de control que usa las operaciones de adaptación para reconfiguración dinámica de esta arquitectura.

2. Estado del Arte

Uno de los retos actuales de investigación con relación a la gestión de recursos computacionales en la nube para microservicios, es proporcionar soluciones que permitan definir una auto-gestión de estos recursos [12,7]. Una línea prometedora en este sentido es la de habilitar la reconfiguración dinámica de las arquitecturas de microservicios [5]. En [7] se explicita la necesidad de auto-gestionar fallos o incidencias en los servicios, que pueden requerir reparar el sistema a nivel arquitectónico. En [11,10]

argumentan la dificultad de diseñar soluciones para arquitecturas de microservicios auto-adatativas que sean genéricas y que proporcionen componentes reutilizables, y la complejidad para realizar la ingeniería de los bucles de control en estas situaciones. Estos trabajos evidencian la inexistencia de soluciones con estas características en este dominio. Además, constatan las limitaciones tecnológicas de las herramientas de gestión de las infraestructuras de microservicios (orquestadores) en cuanto a sus capacidades de computación autónoma, y promueven el trabajo en este camino.

La computación autónoma es un área de investigación que postula el uso de bucles de control externos como estrategia para introducir estas capacidades de auto-gestión a los sistemas. Desde la propuesta de los bucles MAPE-K [1] como marco de referencia abstracto para diseñar los bucles de control, muchos trabajos han seguido este camino [4,9] proporcionando conceptualizaciones y frameworks abstractos, o soluciones que no son aplicables al ámbito de los microservicios [10].

El objetivo de este trabajo es proponer una estrategia que permita diseñar arquitecturas de microservicios que estén preparados para ser reconfigurados dinámicamente, extendiendo las capacidades de las infraestructuras y herramientas actuales. La propuesta debe introducir y diseñar componentes reutilizables, y debe poder aplicarse de manera amplia sobre este tipo de soluciones. Por último, debe ser concreta para poder implementarse con tecnologías actuales.

3. Ingeniería de Servicios *adaptive-ready*

Una arquitectura de microservicios se compone de un conjunto de microservicios distribuidos e interconectados en una red de computación [6]. Cada microservicio puede proporcionar una o varias funcionalidades (ofrecidas a través de interfaces) y requerir otras interfaces proporcionadas por otros microservicios. La Figura 1 muestra la notación que usaremos, siguiendo el enfoque presentado en [8], así como el concepto de **configuración arquitectónica**, que representa un conjunto de instancias de microservicios y sus relaciones.

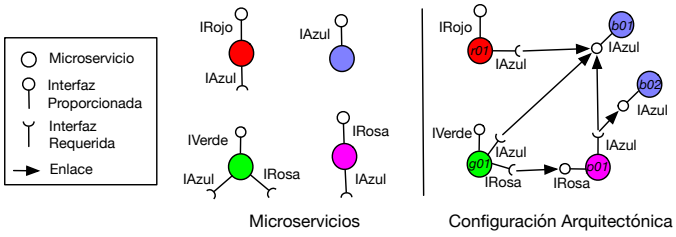

Figura 1. Notación para describir arquitecturas de microservicios

3.1. El Contrato de Adaptación

El contrato de adaptación establece el mecanismo a través del cual los bucles de control auto-gestionarán las arquitecturas de microservicios en base a unas **operaciones de adaptación** que realizarán la reconfiguración arquitectónica en tiempo

de ejecución. Refinamos el trabajo descrito en [8] para aplicarlo a los microservicios, y añadimos dos nuevas operaciones para exponer y ocultar las interfaces de los microservicios. Las operaciones propuestas son las siguientes:

Desplegar M **:** se crea (instancia) el microservicio M
Eliminar M **:** se elimina el microservicio M
Exponer $M :: I$ **:** se pone accesible la interfaz I del microservicio M
Ocultar $M :: I$ **:** se elimina la accesibilidad a la interfaz I del microservicio M
Enlazar $M_1 :: I_r$ **con** $M_2 :: I_p$ **:** se conecta la interfaz requerida I_r del microservicio M_1 con la interfaz I_p del microservicio M_2
Desenlazar $M_1 :: I_r$ **de** $M_2 :: I_p$ **:** se elimina el enlace entre la interfaz requerida I_r de M_1 de I_p de M_2

La Figura 2 muestra un ejemplo de cambio arquitectónico de una configuración inicial a una final, a través de las operaciones de adaptación. Para poder ejecutar estas operaciones sobre los microservicios, éstos deben extenderse para implementar este contrato de adaptación. La siguiente sección detalla cómo realizar esta extensión de los microservicios.

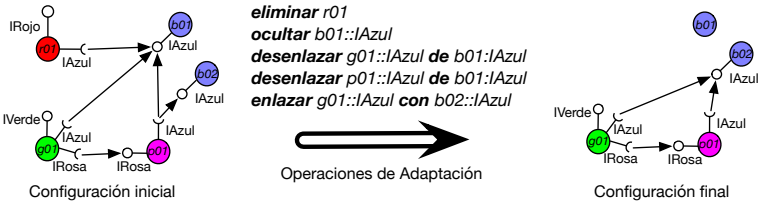

Figura 2. Ejemplo de adaptación

3.2. $\mathcal{ARS}$: Servicios *adaptive-ready*

Definimos un $\mathcal{ARS}$ como un microservicio que se extiende para implementar el contrato de adaptación (ver Figura 3). Este contrato se materializa a través de una **interfaz de adaptación** que expone las operaciones de adaptación. Esta interfaz de adaptación será utilizada por el bucle de control para reconfigurar la arquitectura de microservicios. En la Figura 3 se puede ver un bucle de control MAPE-K [1] usando esta interfaz de adaptación $\mathcal{ARS}$.

La implementación de las operaciones de adaptación debe seguir una serie de convenciones. La operación de `desplegar` un microservicio es la responsable de configurar y arrancar el microservicio. Al iniciar un microservicio $\mathcal{ARS}$, se deberá lanzar la ejecución de la operación `desplegar`, y luego publicar (de manera accesible) la interfaz de adaptación. La operación de `eliminar` un microservicio realiza la operación inversa. Las operaciones de `enlazar` y `desenlazar`, deben realizar cambios sobre las dependencias del microservicio con otros servicios. Las operaciones de `exponer` y

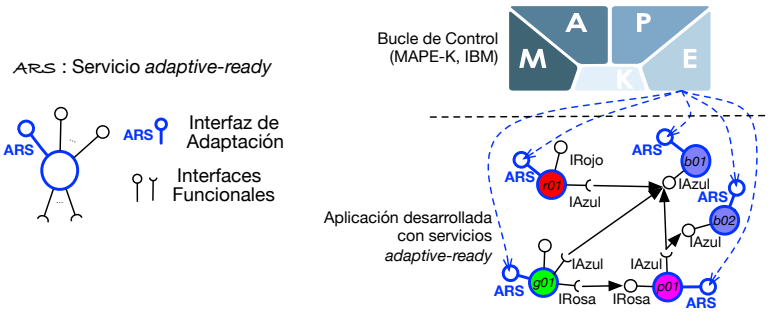

Figura 3. Servicio *adaptive-ready* (izquierda) y Bucle MAPE-K sobre arquitectura de microservicios *adaptive-ready* (derecha)

`ocultar` interfaces, deben activar y parar, respectivamente, las implementaciones de dichas interfaces proporcionadas. A continuación se presenta la aplicación de esta propuesta sobre un escenario real.

4. Caso de Estudio: La fábrica de cantoneras de cartón

Embalpack Levante S.L. es una fábrica, situada en Favara (Valencia), que se dedica a la producción de cantoneras de cartón a partir de un proceso de conformado de papel. Esta fábrica usa diferente maquinaria para la producción: 4 líneas de trabajo, 2 rebobinadoras, 1 cortadora, 2 troqueladoras, y múltiples dispositivos físicos para proporcionar feedback a los operarios (entre otros). En estos últimos años el centro PROS de la UPV ha realizado una asesoría tecnológica para diseñar e implantar una infraestructura de computación alineada con el concepto de Internet de las Cosas y las Fábricas del Futuro.

Se ha diseñado una arquitectura de microservicios que cooperan y extienden las capacidades de producción físicas de esta maquinaria. Existen servicios íntimamente ligados a dispositivos físicos, que o bien se ejecutan embebidos dentro estos, o bien se ubican en módulos de cómputo externos (de tipo Raspberry Pi) conectados directamente a ellos (a través de sus interfaces de comunicación físicas). El funcionamiento de este tipo de servicios depende fuertemente de que los dispositivos físicos estén activos y operativos. Estos microservicios conviven y cooperan con otras aplicaciones monolíticas, como el sistema de planificación y gestión de recursos (`ERP`), un servicio o middleware de comunicaciones (`COMM`), o un módulo de gestión del stock (`SSTOCK`), entre otras. La comunicación entre ellos se realiza a través de llamadas síncronas y asíncronas. La implementación y despliegue de los servicios ha funcionado (en el pasado) en contenedores `docker` ejecutándose en máquinas virtuales `Virtual Box`.

La Figura 4 muestra los microservicios principales del sistema. Como se puede observar en color morado, existen servicios ofrecidos directamente por dispositivos físicos (máquinas de producción `DevPL`, rebobinadoras `DevREW` o semáforos de producción `DevTL`). En verde, los que extienden procesos productivos, recolectando y procesando datos, o proporcionando interfaces de acceso a los dispositivos físicos.

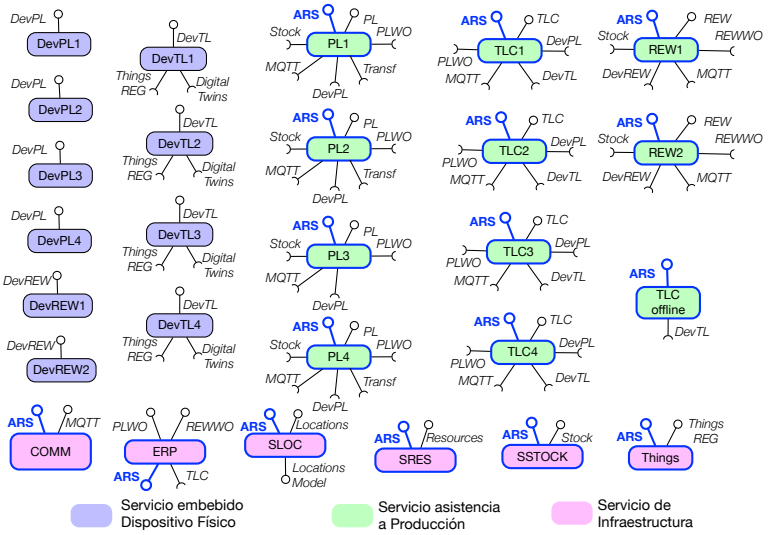

Figura 4. Microservicios principales en Embalpack

Los servicios `PL` están vinculados a las máquinas de producción, los `TLC` son controladores de semáforos, y los `REW` se conectan con las rebobinadoras. Por último, en rosa, servicios de infraestructura (`ERP`, `COMM`, `SSTOCK`, etc.). La figura muestra las interfaces proporcionadas y requeridas para cada microservicio. Se puede observar cómo algunos microservicios se definen como *adaptive-ready* (contorno en azul), y exponen la interfaz de adaptación $\mathcal{ARS}$.

4.1. Microservicios *adatpive-ready* en `kubernetes`

Se eligió `kubernetes` como la tecnología para orquestar la disponibilidad y realizar el despliegue de los diferentes microservicios $\mathcal{ARS}$ debido a su madurez y soporte, además de que permite configurar soluciones *'on-premise'*. Una solución en `kubernetes` se define a través de un `clúster` de `nodos` (máquinas físicas o virtuales) sobre las que `kubernetes` desplegará los microservicios. Uno de estos nodos actúa como *maestro* y se encarga de mantener la información de la infraestructura actualizada y ofrecer una API (`api-server`) para consultar y configurar el `clúster`. Los microservicios deben empaquetarse (para su despliegue) como contenedores (por ejemplo, usando `docker`), y se ejecutan dentro de `pods`[3] en los `nodos`. Cuando `kubernetes` decide iniciar un microservicio, se descarga la imagen de su contenedor de un repositorio, y lo ejecuta dentro de un `pod` (que alojará en algún `nodo` del `clúster`). Los `pods` se comunican continua y automáticamente con la infraestructura para confirmar su estado operativo.

`Kubernetes` usa el concepto de `Deployment` como mecanismo para auto-instalar y garantizar la disponibilidad de un `pod`. Si un `pod` deja de responder, `kubernetes`

[3] Un `pod` es un contenedor específico de `kubernetes` que ejecuta contenedores

buscará su configuración de `Deployment` para determinar si debe o no desplegar uno nuevo (y eliminar el que falla). A través del concepto de `Service` se hace accesible el puerto de un `pod` en el `clúster` y hacia el exterior. Este es el mecanismo utilizado para publicar las interfaces que un microservicio expone (debe declararse un `Service` por interfaz). Siguiendo esta aproximación, hemos empaquetado los microservicios de la solución para Embalpack en contenedores `docker` y los hemos subido a un `Docker Registry`. Configuramos un `clúster kubernetes` formado por 7 nodos (incluyendo el *maestro*), usando los PCs asociados a las líneas de producción y las rebobinadoras. El resto de servicios físicos se ejecutan embebidos en sus dispositivos (máquinas) físicos, fuera del `clúster`.

Sobre esta solución desarrollamos un bucle de control de tipo MAPE-K [1] (el diseño e implementación de este bucle queda fuera de los objetivos de este artículo) para extender las capacidades de computación autónoma que `kubernetes` no ofrece. La Figura 5 muestra una configuración arquitectónica de un subconjunto de los microservicios de la solución ejecutándose sobre esta infraestructura. A continuación se presenta cómo reconvertimos los microservicios en microservicios $\mathcal{ARS}$, y cómo introdujimos una capa de `Efectores del Sistema` [1] como intermediaria entre el bucle de control y los microservicios $\mathcal{ARS}$.

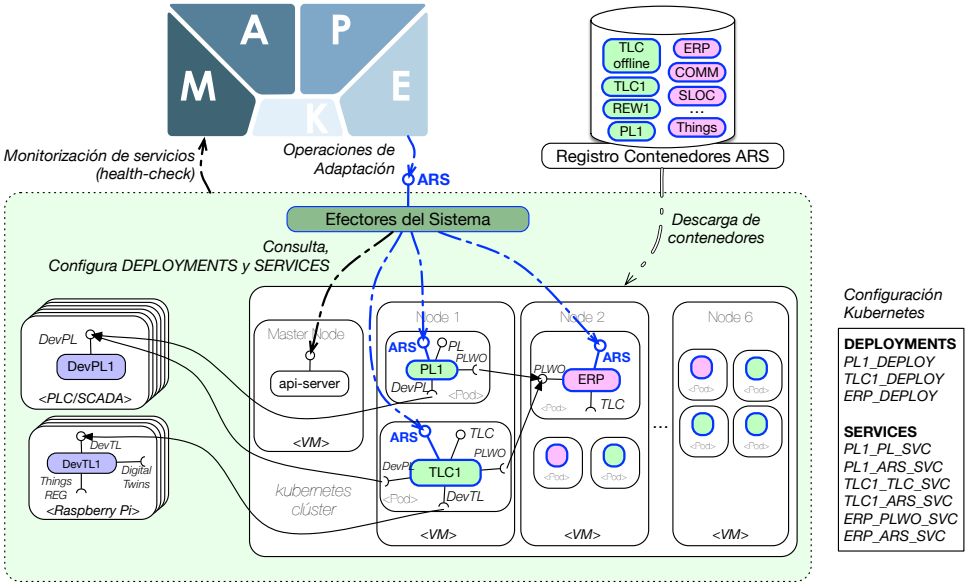

Figura 5. Configuración arquitectónica basada $\mathcal{ARS}$ sobre `kubernetes`

4.2. Implementación de los Microservicios *adaptive-ready*

Los microservicios que inicialmente existían en la fábrica se extendieron para convertirlos en *adaptive-ready*, implementando la interfaz de adaptación $\mathcal{ARS}$. Para facilitar la extensión de estos microservicios, desarrollamos un componente reutili-

zable que expone la interfaz de adaptación como una API REST, y proporciona una implementación básica de las operaciones de adaptación (que hay que ajustar para cada microservicio concreto). Después, re-empaquetamos cada microservicio en un contenedor `docker` y lo configuramos para que al iniciarse, publique el API de adaptación $\mathcal{ARS}$ y lance la operación de `desplegar microservicio`. Por último, subimos estos contenedores al `Registro de Contenedores` $\mathcal{ARS}$ (ver Figura 5).

4.3. Implementación de la capa de Efectores

Los `Efectores del Sistema` [1] son una capa responsable de canalizar las peticiones de adaptación. Esta capa actúa de intermediaria entre el bucle de control y los microservicios $\mathcal{ARS}$, y se ofrece como servicio a través de la interfaz de adaptación ($\mathcal{ARS}$). De esta manera, el bucle de control sólo debe interactuar con estos `efectores`, independientemente de la tecnología usada para desplegar la arquitectura de microservicios. Esta capa es reutilizable para desarrollar otras soluciones de microservicios $\mathcal{ARS}$ con `kubernetes`.

El principal objetivo de estos `efectores` es gestionar, de manera transparente a los microservicios y al bucle de control, el `clúster de kubernetes` y redirigir las peticiones de adaptación a los microservicios $\mathcal{ARS}$. Estos `efectores` utilizan el API que ofrece el servicio `api-server` disponible en el `nodo maestro` de `kubernetes` para: 1) consultar y modificar dinámicamente los `Deployment`s y `Service`s existentes, y para 2) encontrar los `Service`s asociados a las interfaces de adaptación de los microservicios. La implementación de las operaciones realizan las siguientes acciones:

`desplegar` M`:` crea un `Deployment M_DEPLOYMENT` en `kubernetes` para el microservicio M [4]. Esto provocará que `kubernetes` descargue la imagen del contenedor del microservicio M del registro de contenedores, y lo ejecute dentro de un `pod`. Además, configura un `Service M_ARS_SVC` para exponer la interfaz de adaptación del servicio M

`eliminar` M`:` elimina el `Deployment M_DEPLOYMENT`, causando que `kubernetes` detenga los `pod`s asociados al microservicio M. Después, invoca la operación `eliminar` sobre la interfaz de adaptación del microservicio M (buscando el `Service` de nombre `M_ARS_SVC`). Finalmente, elimina todos los `Services` definidos para las interfaces funcionales de M, incluído el `Service M_ARS_SVC`

`exponer` $M :: I$`:` configura un nuevo `Service` en con el nombre `M_I_SVC`, y redirige la petición al microservicio M a través de su interfaz de adaptación `M_ARS_SVC`

`ocultar` $M :: I$`:` elimina el `Service M_I_SVC` declarado, y redirige la petición al microservicio M a través de su interfaz de adaptación `M_ARS_SVC`

`enlazar` $M_1 :: Ir$ `con` $M_2 :: Ip$ `y desenlazar` $M_1 :: Ir$ `de` $M_2 :: Ip$`:` no cambian la configuración de `kubernetes` y redirigen la petición al `Service` M_1`_ARS_SVC`

A través de estas operaciones se consigue que la infraestructura de `kubernetes` se actualice dinámicamente en base a las operaciones de adaptación solicitadas.

[4] En la solución utilizada, se configuran los `Deployments` con un número de réplicas igual a 1 para garantizar disponibilidad. En otros escenarios, esta configuración podría ser diferente, e incluso específica para cada tipo de servicio

4.4. Escenarios de Adaptación

Atendiendo a los microservicios diseñados para la fábrica y sus dependencias, existen situaciones en las que se requiere reconfigurar dinámicamente la arquitectura de la solución. Cada necesidad de reconfiguración la desarrollamos como un escenario de adaptación que implementamos a través de una política de adaptación dentro del bucle de control. Estos escenarios se disparan cuando se detecta una condición sobre la infraestructura de microservicios (típicamente, debido a monitorizaciones de tipo *health-check*), y para cada escenario, la política de adaptación propone un conjunto de operaciones de adaptación.

En la actualidad se han implementado 32 escenarios de adaptación (su número crece a medida que se identifican nuevos contextos o problemas con las configuraciones). La Figura 6 muestra 4 de estos escenarios, relacionados con la disponibilidad (o no) de los servicios que se ejecutan embebidos en las máquinas (`DevPL`) y en los semáforos (`DevTL`), aplicados sobre la línea 1. En cada situación, el bucle de adaptación solicita la ejecución de las operaciones de adaptación asociadas, provocando la reconfiguración arquitectónica y estableciendo la configuración mostrada la figura.

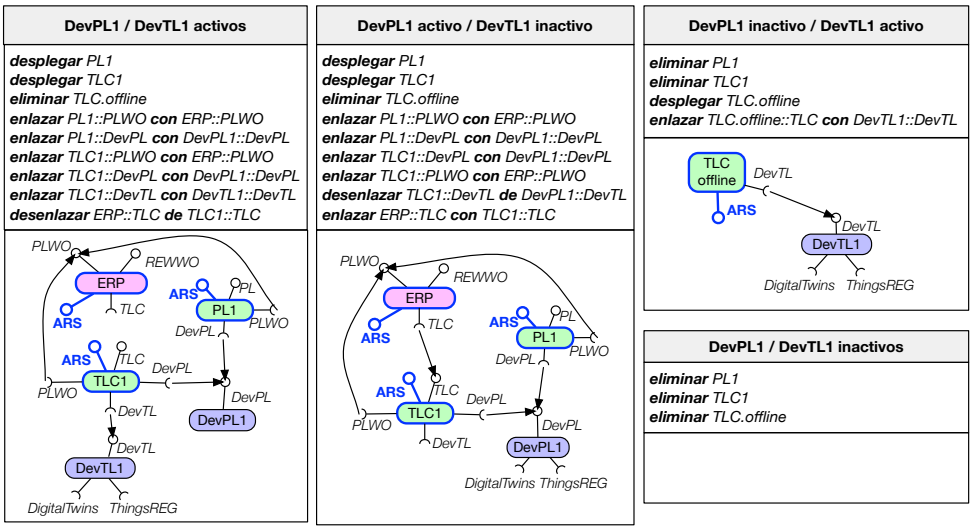

Figura 6. Escenarios de Computación Autónoma en Embalpack

5. Conclusiones

Las herramientas e infraestructuras actuales para la gestión de arquitecturas de microservicios poseen limitaciones con respecto a su capacidad de realizar reconfiguraciones arquitectónicas de manera automática. En este trabajo se ha propuesto una estrategia para diseñar servicios *adaptive-ready* que faciliten el soluciones reconfigurables, basadas en el uso de bucles de control (siguiendo la perspectiva de la computación autónoma). Para conseguirlo, se ha definido una interfaz de adaptación

que deben implementar estos servicios que sirve de contrato de uso para los bucles de control. Usando esta estrategia se consigue, además, separar los aspectos de adaptación de los aspectos funcionales a la hora de desarrollar los servicios *adaptive-ready*. Esto ha reportado diferentes beneficios, como el desacople de las políticas de adaptación de la implementación de los servicios, la posibilidad de reemplazar el bucle de control sin que afecte a la implementación de los microservicios, o abstraer a los microservicios de la tecnología subyacente en la que se van a ejecutar y auto-adaptar.

Se ha validado la propuesta en una aplicación industrial real. Dada la naturaleza cambiante del sistema informático de la empresa (a causa de los cambios tecnológicos que está acometiendo con la incorporación de nuevos servicios y dispositivos), se ha constatado esta separación de aspectos de adaptación de los aspectos funcionales de los servicios. Esto está permitiendo incorporar estos nuevos servicios y dispositivos, y refinar las reglas de adaptación del bucle de control. De hecho, de los 16 escenarios de reconfiguración inicialmente detectados, en estos momentos existen 32 (y se prevee que este número siga creciendo), sin haber requerido la reimplementación de los servicios participantes. Como resultado, hemos obtenido la implementación de componentes reutilizables que ayudarán a desarrollar otras soluciones, como son los `efectores del sistema` para `kubernetes`, y la implementación de un componente para extender los microservicios y convertirlo en $\mathcal{ARS}$.

Referencias

1. An Architectural Blueprint for Autonomic Computing. Tech. rep., IBM (Jun 2005)
2. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices Architecture Enables DevOps: Migration Cloud-Native Architecture. IEEE Software **33**(3), 42–52 (2016)
3. Brun, Y., et al: Engineering Self-Adaptive Systems through Feedback Loops. In: Software Engineering for Self-Adaptive Systems (2009)
4. Cheng, B.H., et al.: Software Engineering for Self-Adaptive Systems. pp. 1–26. Springer-Verlag, Berlin, Heidelberg (2009)
5. Dragoni, N., et al: Microservices: yesterday, today, and tomorrow. CoRR **abs/1606.04036** (2016)
6. Fowler, M., Foemmel, M.: Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html (2006)
7. Jamshidi, P., Pahl, C., Mendonça, N., Lewis, J., Tilkov, S.: Microservices: The Journey So Far and Challenges Ahead. IEEE Software **35**, 24–35 (05 2018)
8. Kramer, J., Magee, J.: Self-Managed Systems: an Architectural Challenge. In: Future of Software Engineering (FOSE '07). pp. 259–268 (May 2007)
9. de Lemos, R., et al.: Software Engineering for Self-Adaptive Systems: A second Research Roadmap. In: Software Engineering for Self-Adaptive Systems II, Lecture Notes in Computer Science (LNCS), vol. 7475, pp. 1–32. Springer (January 2013)
10. Mendonça, N., Jamshidi, P., Garlan, D., Pahl, C.: Developing Self-Adaptive Microservice Systems: Challenges and Directions. IEEE Software (11 2019)
11. Mendonça, N.C., Garlan, D., Schmerl, B., Cámara, J.: Generality vs. Reusability in Architecture-Based Self-Adaptation: The Case for Self-Adaptive Microservices. In: The 12th European Conference on Software Architecture: Companion Proceedings. Madrid, Spain (24-28 September 2018)
12. Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., Bohnert, T.M.: Self-managing Cloud-Native Applications: Design, implementation, and experience. Future Generation Computer Systems **72**, 165–179 (2017)