Contribuciones a la interoperabilidad continua entre humanos y sistemas automatizados sujetos al paradigma del Internet de las Cosas

Eugenio Rubio Drosdov

en cumplimiento parcial de los requisitos para el grado de
Doctor en
Ingeniería Telemática

Universidad Carlos III de Madrid

Director:
Daniel Díaz Sánchez

Fecha deposito Octubre 2022

Esta tesis se distribuye bajo licencia "Creative Commons
Atribución-NoComercial-CompartirIgual"

Ahora sé cómo de estúpido soy y en qué áreas.

AGRADECIMIENTOS

Esta tesis es la culminación de un viaje. Ha sido largo y pesado. Una montaña rusa de emociones que ha habido que aprender a gestionar y domeñar. Sin embargo, aquí está. El fin de un ciclo. Es posible que solo aquellas personas que han pasado por este proceso sean las únicas capaces de entender cómo es el camino. Simplemente diré que ahora me siento en paz y liberado. La misma sensación que se tiene al llegar a la cumbre de una montaña cuando de camino pensabas "en qué momento me pareció buena idea subir todo esto" o cuando superas una lesión y vuelves a practicar un deporte que hacía meses o años tuviste que dejar en pausa. Simplemente, hay que vivirlo para entender qué es la superación.

Naturalmente, en este tipo de actividades no se está solo. O bien se comparten con alguien, o bien otras personas te animan y te desean lo mejor con cada paso. Quisiera nombrar a algunas de las más importantes:

En primer lugar, posiblemente la persona que más ha atestiguado mis subidas y bajadas emocionales haya sido mi director de tesis, Daniel Díaz Sánchez. Del mismo modo que una semana pasaba cada día a pedirle orientación sobre algunas ideas que tenía, me pasaba otras sin hablar con él. Y aún con todo, ha sido un apoyo constante siendo paciente, motivándome y, por supuesto, compartiendo su inmenso conocimiento. Gracias Dani por acompañarme en la carrera, en el máster y en el doctorado.

Por supuesto, también quisiera dar las gracias a mis compañeros y compañeras del departamento de Ingeniería Telemática que, durante todo este tiempo, me han apoyado y se han preocupado por saber cómo iba el desarrollo de mi trabajo. Especialmente quisiera nombrar a Iria Estévez Ayres, cuyo despacho estaba enfrente del mío y con quién he tenido el placer de tener charlas, frikear y compartir aula, como alumno durante la carrera, y como compañero profe durante los años de doctorado. También nombrar a Mario Muñoz, Luis Sánchez, Carlos Rubio o Celeste Campo entre otros, que han sacado algunos ratos para ayudarme y aconsejarme. Y no quisiera terminar este párrafo sin nombrar a Andrés Marín y a Florina Almenárez, que me han acompañado en este proceso dándome la oportunidad de impartir clases en la universidad, así como ayudándome con la financiación de mi paso por la universidad y mi investigación.

Naturalmente, debo nombrar a mis amistades. Gracias por aguantarme y darme un entorno en el que desahogarme y ser libre. Me habéis cuidado mental y emocionalmente. Sé que os debo tiempo. Ahora, por fin, lo tengo.

Y a mi madre. Nunca se puede mostrar el suficiente agradecimiento a una persona que siempre, pasara lo que pasara, lo ha dado todo por mí. No hay más que decir. No hay florituras ni palabras extra. Solo un "esto ha sido gracias a ti".

CONTENIDOS PUBLICADOS Y PRESENTADOS

Esta tesis doctoral contiene contenidos publicados y presentados en varias contribuciones anteriores. A continuación se muestran las publicaciones del autor las cuales han sido mencionadas durante la tesis.

A. Publicaciones en revistas indexadas en el JCR

En este apartado se presentan los artículos que han sido publicados en alguna revista.

1. E. Rubio-Drosdov, D. Díaz-Sánchez, P. Arias-Cabarcos, F. Almenárez, y A. Marín, *Seamless human-device interaction in the internet of things*, in IEEE Transactions on Consumer Electronics, vol. 63, no. 4, pp. 490-498, noviembre de 2017. DOI: https://doi.org/10.1109/TCE.2017.015076
   Calidad de la revista:
   - Categoría: JCR Q3 (2017).
   - Factor de impacto: 1,802.
   - Categoría: Telecommunications.

   Contribución y rol:
   - Esta contribución está incluida de forma íntegra en los capítulos 5, 6 y 7. El material de esta fuente incluido en la tesis no está señalado por medios tipográficos ni referencias.
   - El rol del autor de esta tesis en esta publicación fue contribuir con la idea principal de la propuesta. Así mismo, llevó a cabo su implementación y evaluación, analizando las diferentes herramientas que se podían usar para su desarrollo, siendo el autor principal del artículo en colaboración y bajo la supervisión de los coautores participantes.

B. Publicaciones en congresos

En este apartado se presentan las publicaciones que han sido publicados en algún congreso o conferencia de ámbito internacional.

1. E. Rubio-Drosdov, D. Díaz-Sánchez, A. Marín y F. Almenárez, *A Framework for Microservice Migration and Performance Assesment*, in 2020 International Conference on Intelligente Environments (IE2020), pp. 291-300, junio de 2020, Madrid, España. DOI: https://doi.org/10.3233/AISE200053

- Esta contribución está incluida de forma íntegra en los capítulos 3, 4 y 7. El material de esta fuente incluido en la tesis no está señalado por medios tipográficos ni referencias.
- El papel del autor de la presente tesis en esta publicación fue contribuir con la idea principal de la arquitectura propuesta, además de llevar a cabo una implementación y evaluación, considerando las diferentes herramientas que se podían usar para su desarrollo. Además, llevó a cabo la escritura del propio artículo en colaboración y bajo la supervisión de los coautores participantes. Así mismo, asistió al congreso donde se expuso dicho trabajo.

2. E. Rubio-Drosdov, D. Díaz-Sánchez, F. Almenárez, y A. Marín, *A Framework for Efficient and Scalable Service Offloading in the Mist,* in 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 460-463, abril de 2019, Limerick, Irlanda. DOI: https://doi.org/10.1109/WF-IoT.2019.8767258
   - Esta contribución está incluida de forma íntegra en los capítulos 3, 4 y 7. El material de esta fuente incluido en la tesis no está señalado por medios tipográficos ni referencias.
   - El rol del autor de esta tesis en esta publicación fue contribuir con la idea principal de la arquitectura propuesta, su implementación y evaluación, analizando las diferentes herramientas que se podían usar para su desarrollo, así como la escritura del propio artículo en colaboración y bajo la supervisión de los coautores participantes. Así mismo, asistió al congreso donde se expuso dicho trabajo.

3. E. Rubio-Drosdov, D. Díaz-Sánchez, P. Arias-Cabarcos, F. Almenárez, y A. Marín, *Towards a seamless human interaction in IoT,* in 2015 International Symposium on Consumer Electronics (ISCE), junio de 2015, Madrid, España. DOI: https://doi.org/10.1109/ISCE.2015.7177781
   - Esta contribución está incluida en los capítulos 5, 6 y 7. El material de esta fuente incluido en la tesis no está señalado por medios tipográficos ni referencias.
   - El rol del autor de esta tesis en esta publicación fue contribuir con la propuesta principal, así como su implementación y evaluación, donde se consideraron diferentes herramientas que se podían usar para su desarrollo. así como la escritura del propio artículo en colaboración y bajo la supervisión de los coautores participantes. Así mismo, asistió al congreso donde se expuso dicho trabajo.

OTROS MÉRITOS DE INVESTIGACIÓN

Parte del contenido se desarrolló como línea de investigación en varios proyectos de I+D, denominados:

- INRISCO: Monitorización de Incidentes en Comunidades Inteligentes: Seguridad y Movilidad (TEC2014-54335-C4-2-R)
- MAGOS: Secure Smart Grid Using Open Source Intelligence (TEC2017-84197-C4-3-R)
- CYNAMON co-financed by European Structural Funds (ESF end FEDER) (P2018/TCS-4566)

RESUMEN

Este trabajo se enmarca en el amplio contexto del Internet de las Cosas, y se centra en la interoperabilidad entre humanos y los sistemas de automatización. En cómo los dispositivos que rodean al usuario interactúan con los elementos computacionales que encuentran en su entorno, pudiendo generar, de forma automática, tareas que sean delegables a otros elementos.

Siendo más concretos, la intención de este trabajo es doble. En primer lugar, proponer una arquitectura, basada en el uso de microservicios, gracias a la cual poder migrar las tareas presentes en los dispositivos cercanos al usuario a otros dentro del mismo entorno. En segundo lugar, desarrollar un sistema que haga posible la interacción del usuario con la arquitectura, permitiendo dar instrucciones al entorno a través del lenguaje natural.

El desarrollo de arquitecturas basadas en microservicios posibilita la migración de un dispositivo a otro de determinadas piezas de software ligeras. Para realizar estas migraciones de la manera más óptima posible, es necesario conocer la compatibilidad entre microservicios y dispositivos. Este trabajo proporciona una manera de determinar esta compatibilidad.

La interacción entre un usuario y su entorno a través del lenguaje natural está muy presente en nuestro día a día, por ejemplo, a través de asistentes virtuales de voz. Se trata de un mecanismo que busca facilitar la interacción entre humano y entorno, pero que, en determinadas situaciones, puede no responder de forma acertada a las peticiones del usuario. Una de las razones por las que se pueden dar estas situaciones es la falta de compresión de la petición dada porque no se han utilizado los términos adecuados. Otra de las razones es la falta de una definición más precisa del entorno donde es posible que coexistan diferentes elementos que, aún siendo diferentes, pueden satisfacer de una forma similar los deseos del usuario dando lugar a posibles confusiones. Que un usuario sea capaz de dar órdenes a un entorno sin necesidad de conocer el lenguaje o los patrones necesarios para que un sistema las entienda, aumenta el número de usuarios que son capaces de interactuar con su entorno. En este trabajo se presenta un sistema que pretende eliminar la necesidad de entrenamiento previo del mismo, así como el requisito, por parte del usuario, de conocer los enunciados o palabras clave necesarias para generar órdenes.

A continuación se presentan las principales contribuciones en este contexto:

- Proposición de un conjunto de directrices que forman el marco teórico para el despliegue de una arquitectura basada en microservicios que ayudan a la descentralización del análisis de los diferentes dispositivos que coexisten en ella, así como al análisis de los diferentes microservicios presentes en la misma.
- Proposición de un conjunto de directrices que forman el marco teórico para el despliegue de infraestructuras efímeras, en arquitecturas basadas en microservicios, que ayudan a simplificar la toma de decisiones tanto en su despliegue como en sus migraciones. En concreto, permitiendo generar entornos donde llevar a cabo diferentes análisis de rendimiento en los dispositivos capaces de alojar un microservicio, añadiendo así un método con el que obtener datos para la toma de decisiones.
- Proposición de un conjunto de directrices que forman el marco teórico para el análisis de órdenes dadas en lenguaje natural por parte de un usuario humano a su entorno cercano.
- Proposición de un conjunto de directrices que forman el marco teórico que facilita el desarrollo de servicios compatibles con la utilización del lenguaje natural para su uso.
- Proposición de un conjunto de directrices que forman el marco teórico que permite, a los desarrolladores, incorporar la información necesaria sobre sus servicios y las acciones que se pueden realizar gracias a ellos, permitiendo una interacción fluida sin añadir complejidad al lenguaje que se debe utilizar.
- Proposición de un conjunto de directrices que forman el marco teórico que facilita la virtualización de las acciones realizables en un entorno a través del análisis de las descripciones provistas por sus desarrolladores.
- Proposición de un conjunto de directrices que forman el marco teórico que pretende eliminar la necesidad de entrenamiento, tanto del entorno, como de los usuarios, para poder relacionar la intención del usuario y la acción que pretende llevar a cabo.
- Implementación, como prueba de concepto, de una arquitectura de microservicios que sigue las directrices expuestas en este trabajo, donde la toma de decisiones sobre sus migraciones responde, tanto a la

posibilidad de despliegue de programas ligeros y de corta duración para el análisis de rendimiento, así como a los deseos expuestos por usuarios potenciales a través del lenguaje natural.

ÍNDICE GENERAL

1 INTRODUCCIÓN 1
1.1 Motivación . . . 2
1.2 Objetivos . . . 4
1.3 Contribuciones . . . 7
1.4 Estructura del documento . . . 8
2 ESTADO DEL ARTE 11
2.1 Internet de las Cosas: contextualización y definiciones . . . 11
2.2 Cloud Computing . . . 13
2.3 Edge/Fog Computing . . . 15
2.3.1 Definición . . . 16
2.3.2 Mist Computing . . . 24
2.4 Computation offloading . . . 25
2.5 Arquitectura de Microservicios . . . 28
2.5.1 Definición . . . 28
2.5.2 Virtualización: contenedores VS máquinas virtuales . . . 29
2.5.3 Herramientas . . . 32
2.5.4 Análisis de rendimiento de microservicios . . . 35
2.6 Humanos y su entorno . . . 36
2.6.1 El entorno: cómo describirlo . . . 36
2.6.2 El humano: procesamiento del lenguaje natural . . . 43
2.7 Notas finales . . . 45
3 ARQUITECTURA DE MICROSERVICIOS: DESCRIPCIÓN 47
3.1 Arquitectura de microservicios . . . 48
3.1.1 Entidades presentes . . . 48
3.1.2 Requisitos y funcionalidades . . . 54
3.2 Análisis de rendimiento . . . 55
3.2.1 Componentes y consideraciones de diseño . . . 57
3.2.2 Plan de pruebas . . . 60
3.3 Conclusiones . . . 61
4 ARQUITECTURA DE MICROSERVICIOS: VALIDACIÓN 63
4.1 Implementación del NS y del μSS . . . 63
4.2 Implementación del método de testeo de rendimiento . . . 66
4.3 Resultados . . . 69
4.4 Conclusiones . . . 71
5 COMUNICACIÓN DEL USUARIO CON SU ENTORNO: DESCRIPCIÓN 75
5.1 El intérprete . . . 75
5.2 Diseño y funcionalidad: descripción de los módulos . . . 78

5.2.1 Filtro de órdenes. 78
5.2.2 Cluster Manager 79
5.2.3 Cluster Comparator 80
5.3 Descripción del entorno y sus acciones 81
5.3.1 Ontologías 81
5.3.2 Etiquetado de acciones y su expansión 82
5.4 Comparación de clusters 85
5.5 Plan de pruebas 87
5.6 Conclusiones 88
6 COMUNICACIÓN DEL USUARIO CON SU ENTORNO: VALIDACIÓN 91
6.1 Acciones y sus descripciones 91
6.2 Virtualización de las acciones 92
6.3 Resultados 95
6.3.1 Sistema sin aprendizaje 97
6.3.2 Sistema con aprendizaje 98
6.4 Conclusiones 99
7 INTEGRACIÓN: DISEÑO Y VALIDACIÓN 103
7.1 Integración: diseño 103
7.1.1 Generación automática de posibles migraciones 105
7.2 Plan de pruebas 105
7.3 Resultados 107
7.4 Conclusiones 107
8 DISCUSIÓN, CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN 111
8.1 Principales contribuciones 112
8.2 Conclusiones 117
8.3 Líneas futuras 119
Bibliography 121

ÍNDICE DE FIGURAS

Figura 1 Diferencias de administración entre usuario y proveedor en una configuración local, IaaS, PaaS y SaaS . 15
Figura 2 Capas de la fog 19
Figura 3 El rol de la Mist en el IoT 25
Figura 4 Arquitectura monolítica VS arquitectura de microservicios 29
Figura 5 Virtualización con hypervisor 30
Figura 6 Virtualización con contenedores 30
Figura 7 Ejemplo básico: 2 hosts, 4 microservicios y 3 servicios 49
Figura 8 Extensión del ejemplo básico con un orquestador . . . 51
Figura 9 Infraestructura de evaluación de rendimiento. 57
Figura 10 Escenario creado para alojar al NS y al µSS. 65
Figura 11 Escenario final para la realización de pruebas de rendimiento. 70
Figura 12 Medidas de tiempos en el despliegue del entorno de testeo. 71
Figura 13 Esquema de despliegue del intérprete 77
Figura 14 Representación de la ontología a la que responden los objetos. 82
Figura 15 Representación de la ontología a la que responden las acciones. 82
Figura 16 Evaluación del sistema sin aprendizaje 99
Figura 17 Evaluación del sistema con aprendizaje 100
Figura 18 Evaluación del sistema integrado con aprendizaje . . 109

ÍNDICE DE TABLAS

Tabla 1 Ejemplo: descripción de las acciones de una televisión 84
Tabla 2 Ejemplo: Etiquetas asociadas a las acciones de una Televisión . . . 84
Tabla 3 Ejemplo: Clústeres asociados a las acciones de una Televisión . . . 85
Tabla 4 Ejemplo: Etiquetas de los clústeres de la tabla 3 85
Tabla 5 Acciones de aplicación. . . . 93
Tabla 6 Acciones de objeto. . . . 94
Tabla 7 Formulario para las acciones de aplicación. . . . 96
Tabla 8 Formulario de acciones de objeto. . . . 97
Tabla 9 Patrón para las acciones creadas de forma automática que hacen referencia a cada una de las posibles migraciones. . . . 105
Tabla 10 Dispositivos definidos para la prueba de integración. 107
Tabla 11 Formulario de acciones y sus correspondientes descripciones . . . 108

1

INTRODUCCIÓN

The most profound technologies are those that disappear.
They weave themselves into the fabric of everyday
life until they are indistinguishable from it.

— Mark Weiser [1]

En 1991 Mark Weiser escribió estas palabras en un artículo [1] que proporcionó una nueva visión sobre cómo el ser humano y las máquinas iban a convivir. En él, describió su visión de un mundo en el que la tecnología está al lado del ser humano, lo complementa. Weiser dice que la tecnología gira alrededor del individuo, llenando su entorno de sistemas de computación conectados entre sí, y capaces de atender, e incluso prever, las necesidades del individuo. En su propuesta, los dispositivos están desplegados en el espacio que rodea al ser humano e incluso forman parte de él, desde pequeños dispositivos llevados por la persona, hasta máquinas embebidas en las paredes de los edificios e incluso en las calles. Weiser concibió una nueva manera de pensar sobre el papel de las computadoras en el mundo. Una manera en la que los ordenadores se desvanecieran en el entorno, haciendo que la interacción, en muchos casos, fuese transparente. En ese momento nace el concepto de *Computación Ubicua*.

30 años después, la tecnología ha evolucionado para darnos una cobertura de red casi total y velocidades de enlace de varios Gbps con la llegada del 5G[1], los dispositivos apenas ocupan unos pocos centímetros cuadrados [2] y la capacidad de computación de los dispositivos móviles que los usuarios llevan encima supera con creces la capacidad del procesador que controlaba el cohete del primer viaje a la Luna[2]. Si nos fijamos en el entorno más cercano al individuo, estamos familiarizados con el término *wearable*: un dispositivo que el usuario lleva consigo, capaz, por ejemplo, de medir el ritmo cardíaco, cuántas calorías gastamos o saber por dónde nos movemos [3]. Otros elementos que rodean al usuario tales como vehículos, edificios, vías públicas o maquinaria, también tienen incorporados dispositivos que sirven, no solo para medir, sino para interactuar con ellos, dar órdenes o incluso tomar decisiones, generando una gran riqueza de contexto [4].

1 https://www.etsi.org/technologies/5G
2 https://www.zmescience.com/science/news-science/smartphone-power-compared-to-apollo-432/

1.1 MOTIVACIÓN

El número de dispositivos que se encuentran conectados a Internet ha hecho que el Internet de las Cosas sea un tema de investigación a destacar tanto dentro del mundo académico como del industrial [5]. El Internet de las Cosas se basa en la comunicación de dispositivos inteligentes a través de una red global y una infraestructura dinámica, facilitando la creación de escenarios que pueden ofrecer computación ubicua [6]. Sin embargo, estos dispositivos se caracterizan por estar limitados tanto en términos computacionales como de almacenamiento, haciéndoles susceptibles a fallo o a no poder ofrecer una calidad de servicio suficiente [7]. Para subsanar estos problemas, se ha propuesto el uso de la computación en la nube, la cual proporciona una infraestructura para la computación masiva, así como un capacidad de almacenamiento casi ilimitada [8]. No obstante, dada la cantidad de datos que el Internet de las Cosas genera, se utiliza un gran ancho de banda implicando, para algunos servicios, una velocidad de respuesta no deseada, generando una bajada en la calidad del servicio ofrecida [9] [10]. Dada esta situación, surge la computación en el borde de la red, Edge Computing, donde los dispositivos finales, así como los elementos que los conectan entre ellos, y junto a la nube, se encargan de parte de la computación, ofreciendo una alta velocidad de respuesta para facilitar las tomas de decisiones en tiempo real [11] [12] [13] [14]. Dado el interés por la distribución computacional en el borde de la red, y que los dispositivos que se encuentran en él se ven limitados en sus capacidades, muchos trabajos científicos se han focalizado en el despliegue de entornos computacionales bajo una arquitectura de microservicios [15] [16] [17] [18] [19].

Uno de los problemas presentes en las arquitecturas de microservicios es la administración de dependencias entre microservicios [20]. Dado que los datos que consume un microservicio pueden depender de los datos generados por otro, y que los microservicios son susceptibles a cambiar de entorno computacional [21], i.e. de dispositivo que lo alberga, es necesario prever cómo va a responder, en términos de rendimiento, cuando cambia de localización. Las soluciones actuales proponen diferentes modelos para este estudio, como la monitorización del dispositivo que alberga al microservicio, donde un elemento central recibe mediciones tomadas en los dispositivos que coexisten en el entorno. El análisis de estos datos proporciona una visión de conjunto al elemento central, el cual puede tomar decisiones de orquestación como, por ejemplo, migrar un microservicio a otro dispositivo [22] [23] [24] [25]. Otros trabajos, en cambio, proponen crear referencias con las que predecir el rendimiento de un microservicio. Estas referencias se crean, típicamente, en entornos de testeo, tomando una

gran cantidad de datos que definen el momento en el que las diferentes pruebas se han llevado a cabo. Es decir, se toma una fotografía del estado del entorno, de forma que, en el futuro, se pueda comparar una situación real con las generadas: si se da una situación parecida, se entiende que el comportamiento se ajustará a estos patrones [26] [27] [28]. Sin embargo, diferentes trabajos argumentan que este tipo de soluciones implican una serie de inconvenientes: el tamaño de los modelos y su procesado requiere de una gran capacidad computacional dado que la cantidad de dispositivos y microservicios desplegados en ellos, potencialmente, puede ser de hasta decenas de miles, generando una latencia no deseada [29] [30]. Con la idea de saber cuál va a ser el comportamiento de un microservicio al instanciarlo en un dispositivo destino, el presente trabajo propone el despliegue de infraestructuras efímeras bajo las cuales poder desplegar pruebas ligeras, i.e., con el mínimo consumo de recursos.

Si prestamos especial atención al borde de la red, a los dispositivos finales, es necesario notar que muchos de ellos son dispositivos personales, es decir, dispositivos que los usuarios llevan consigo. Al fin y al cabo, la tecnología esta hecha por y para los humanos y, dependiendo de su comportamiento, la distribución y la ocupación de los dispositivos se ve afectada [31]. Al hecho de tener en cuenta al usuario, dentro del entorno IoT, atendiendo a que sus acciones afectan a su entorno computacional, se le ha dado el nombre de Human-in-the-Loop [32] (o People-Centric IoT [33]). Yendo más lejos, el concepto de People as a Service establece que el dispositivo del usuario es una puerta de entrada al mundo virtual de forma que, gracias a un perfil de usuario, cuando el individuo (y su dispositivo) pasa a formar parte de un entorno, éste se adapta a las necesidades indicadas en el perfil [34]. Algo similar propone el Internet of People, donde el usuario puede arrancar, de forma proactiva, el establecimiento de una comunicación con otros dispositivos cercanos, ya sean de otros usuarios o que formen parte del entorno, pudiéndolo configurar según sus necesidades [35]. A raíz de este interés en colocar al usuario en el centro, esta tesis busca incluir al ser humano en el despliegue de microservicios. Mientras que los trabajos actuales explotan el análisis de datos para establecer perfiles de usuario, y en función de ellos configurar el entorno, el presente trabajo quiere procurar al usuario una forma de dirigirse a este, haciendo que el entorno se adapte a sus necesidades, las cuales atañen, específicamente, a la localización de los microservicios de los que el usuario puede beneficiarse. Es decir, proporcionar al usuario un entorno en el que, a petición suya, los microservicios migren. En concreto, se ha querido desarrollar una infraestructura capaz de responder a las peticiones realizadas en lenguaje natural por parte del

usuario.

Existen muchas soluciones que permiten el uso del lenguaje natural para permitir la comunicación entre un humano y su entorno, ya sea sanitario [36] [37], un hogar inteligente [38] o incluso un vehículo [39]. Estas soluciones se basan en dos pilares: 1) entrenamiento del sistema y/o 2) conocimiento previo por parte del usuario de la terminología. Este trabajo busca eliminar estos dos requisitos.

1.2 OBJETIVOS

En esta sección, se ponen de manifiesto los objetivos que se han perseguido a lo largo del desarrollo de la tesis. Estos se han definido para dar forma a la propuesta que resulta ser el presente documento: una arquitectura de microservicios que permita una interoperabilidad transparente entre humanos y sistemas automatizados a través de un asistente.

- O1: Contribuir a la orquestación de microservicios en un entorno distribuido:
  - O1.1: Proporcionar un mecanismo para la descripción de microservicios y de los dispositivos que coexisten en un entorno. En particular, se propondrán dos entidades que estarán desplegadas en los dispositivos del entorno con el fin de llevar a cabo dos tareas principales: a) extraer información del dispositivo (entre otros datos, información sobre su rendimiento y su ocupación en memoria y procesado); y b) recopilar información sobre los microservicios desplegados en los dispositivos (en concreto, conocer los requisitos que se deben satisfacer para que dicho microservicio ofrezca cierta calidad de servicio).
  - O1.2: Proporcionar una metodología para el análisis de compatibilidades entre las descripciones de los dispositivos y de los microservicios.
  - 01.3:Diseñar un mecanismo que permita la evaluación del rendimiento de los diferentes microservicios en diferentes dispositivos antes de ser desplegados. Este mecanismo debe:
    - Permitir el despliegue de un escenario de pruebas efímero que permita estudiar el comportamiento de los microservicios en los diferentes dispositivos candidatos para hospedarlos.

    - Consumir, en términos de recursos, una cantidad mínima de ellos para no interferir en el comportamiento esperado del resto de microservicios ya desplegados.
    - Evitar la interrupción de los microservicios ya desplegados.
    - Ser compatible con otros mecanismos de decisión, aportando una fuente extra de datos para tomar dichas decisiones.

- O2: Contribuir, con el diseño de un intérprete de órdenes en lenguaje natural, a la interacción entre usuario y entorno haciéndola más fluida y flexible. En particular, se busca:
    - O2.1: Eliminar el uso de palabras clave para la activación de acciones. En ciertos entornos, es necesario que el usuario utilice unos términos en particular para que el entorno comprenda sus órdenes. Con la eliminación de palabras clave, se busca minimizar el esfuerzo requerido al usuario para interactuar con el entorno.
    - O2.2: Diseñar un mecanismo para la descripción de acciones que puede llevar a cabo un dispositivo.
    - O2.3: Diseñar un mecanismo de intercambio de datos entre dispositivos y el intérprete con la idea de que éste tenga una visión global de las acciones que se pueden llevar a cabo en el entorno. Este intercambio de datos debe:
        - Seguir la filosofía Plug-and-Play. Cuando un dispositivo pasa a formar parte del entorno, debe existir una comunicación entre el intérprete y el dispositivo para conocer las acciones que se pueden activar en el dispositivo. Este intercambio de datos debe poder hacerse de forma automática sin la intervención del usuario.
        - Buscar una compatibilidad alta. Es necesario establecer unas pautas de diálogo entre los dispositivos y el intérprete que permitan, a la mayor variedad de dispositivos posible, formar parte del entorno .
    - O2.4: Evitar el entrenamiento del sistema y de los dispositivos. Se busca minimizar el esfuerzo requerido para que el intérprete sea capaz de conocer los términos que activan las diferentes acciones del entorno.

- O3: Contribuir a la interoperabilidad continua entre humanos y sistemas automatizados integrando el intérprete en un entorno distribuido basado en una arquitectura de microservicios. En particular

se busca proporcionar al usuario la posibilidad de comunicar a su entorno su deseo de migración de microservicios de un dispositivo origen a uno destino. Para cumplir con este objetivo, se va a integrar el intérprete desarrollado para el segundo objetivo dentro de la arquitectura de microservicios que satisface el primer objetivo. Para ello, es necesario:

- O3.1: Diseñar un mecanismo que permita al orquestador conocer las compatibilidades entre dispositivos y microservicios, pudiendo concluir qué dispositivos son capaces de hospedar a qué microservicios para así poder generar una colección de posibles migraciones de microservicios a dispositivos del entorno.
- O3.2: Diseñar un mecanismo que permita al intérprete entender una orden dada en lenguaje natural por parte del usuario que implique la migración de un microservicio a un dispositivo destino.
- O3.3: Diseñar un mecanismo que permita al intérprete establecer cuál de las posibles migraciones es la deseada por el usuario sin necesidad de entrenamiento ni requiriendo un esfuerzo extra por parte del usuario.

1.3 CONTRIBUCIONES

Considerando la motivación y los objetivos planteados en las secciones anteriores, las contribuciones generales de esta tesis son:

- El desarrollo de un marco de trabajo

El trabajo realizado en esta tesis ha sido financiado y apoyado por varios proyectos:

- INRISCO: Monitorización de Incidentes en Comunidades Inteligentes: Seguridad y Movilidad (TEC2014-54335-C4-2-R)
- MAGOS: Secure Smart Grid Using Open Source Intelligence (TEC2017-84197-C4-3-R)
- CYNAMON co-financed by European Structural Funds (ESF end FEDER) (P2018/TCS-4566)

Los resultados obtenidos en la misma, han dado lugar a las siguientes publicaciones:

1. E. Rubio-Drosdov, D. Díaz-Sánchez, A. Marín y F. Almenárez, *A Framework for Microservice Migration and Performance Assesment,* in 2020 International Conference on Intelligente Environments (IE2020), pp. 291-300, junio de 2020, Madrid, España.
   - DOI: https://doi.org/10.3233/AISE200053
2. E. Rubio-Drosdov, D. Díaz-Sánchez, F. Almenárez, y A. Marín, *A Framework for Efficient and Scalable Service Offloading in the Mist*, in 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 460-463, abril de 2019, Limerick, Irlanda.
   - DOI: https://doi.org/10.1109/WF-IoT.2019.8767258
3. E. Rubio-Drosdov, D. Díaz-Sánchez, P. Arias-Cabarcos, F. Almenárez, y A. Marín, *Seamless human-device interaction in the internet of things,* in IEEE Transactions on Consumer Electronics, vol. 63, no. 4, pp. 490-498, noviembre de 2017.
   - DOI: https://doi.org/10.1109/TCE.2017.015076
   - Categoría: JCR Q3 (2017)
   - Factor de impacto: 1,802
   - Posición relativa: 44/87
   - Categoría: Telecommunications

4. E. Rubio-Drosdov, D. Díaz-Sánchez, P. Arias-Cabarcos, F. Almenárez, y A. Marín, *Towards a seamless human interaction in IoT,* in 2015 International Symposium on Consumer Electronics (ISCE), junio de 2015, Madrid, España.
    - DOI: https://doi.org/10.1109/ISCE.2015.7177781

1.4 ESTRUCTURA DEL DOCUMENTO

En las siguiente secciones se presentará la tesis doctoral. En primer lugar, en el capítulo 1 se describen las motivaciones, sección 1.1, y la relevancia de la investigación a través de las publicaciones admitidas en congresos internacionales y revistas, sección 1.3. En dicho capítulo se detallan los objetivos de la tesis, sección 1.2.

Más adelante, en el capítulo 2, se detalla el estado del arte, repasando las soluciones más relevantes y de mayor importancia en la investigación. Para ello, se describen sus puntos fuertes, así como las carencias que han motivado la realización de esta tesis. Este capítulo está dividido en diferentes secciones. En primer lugar, en la sección 2.1 se hace una breve contextualización del presente trabajo introduciendo el concepto de servicio dentro del paradigma del Internet de las Cosas. Seguidamente, en las secciones 2.2 y 2.3 se realiza un repaso de los conceptos de Cloud y Edge computing, comentando las herramientas más habituales para resolver los retos que estos paradigmas computacionales implican. En la sección 2.3.2 se explora el concepto de Mist Computing, paradigma bajo el cual se enmarca la presente tesis. Seguidamente, en las secciones 2.4 y 2.5 se repasan los conceptos principales sobre los cuáles se ha edificado este trabajo, a saber: la descarga computacional y las arquitecturas de microservicios. En su repaso se identificarán los problemas que aparecen en dichos conceptos así como se presentarán las herramientas que actualmente buscan solucionarlos, identificando posibles mejoras que esta tesis busca llevar a cabo cumpliendo con el primero de los objetivos desglosados en la sección 1.2.

La última sección que repasa el estado del arte, sección 2.6, repasa algunos de los mecanismos que se utilizan para permitir al usuario comunicarse con su entorno, ya sea personal, privado o público. En dicha sección se identifican dos de los problemas principales subyacentes. En primer lugar, la necesidad por parte del usuario de, en determinados contextos, conocer los términos que disparan ciertos cambios en el entorno. En segundo lugar, el requisito de los desarrolladores de establecer un código que ayude a describir las capacidades de su producto haciéndolo entendible al usuario no

experto.

Por último, en la sección 2.7 se repasan las necesidades identificadas en el capítulo del estado del arte, capítulo 2, que esta tesis trata de satisfacer.

Un vez finaliza la exposición del estado del arte que afecta a esta investigación, los capítulos siguientes tratan las contribuciones del presente trabajo. En el capitulo 3 se propone una arquitectura de microservicios basada en dos elementos, así como un mecanismo para llevar a cabo análisis de rendimiento de microservicios. Respecto a las dos entidades, en la sección 3.1 se presentan su descripción, requisitos y funcionalidad. Asimismo, en la sección 3.2, se propone una metodología para llevar a cabo pruebas de rendimiento efímeras a microservicios en dispositivos destino. La intención de estas pruebas es la de obtener, en tiempo real, datos extra que un orquestador puede consultar para tomar decisiones de despliegue de dichos microservicios. Adicionalmente, se incluye un plan de pruebas para llevar a cabo la validación de estas propuestas, sección 3.2.2.

En el capítulo 4, se procede a la implementación de dichas propuestas mediante pruebas de concepto simuladas. El objetivo de esta implementación es la validación de las propuestas. En la sección 4.1 se describe la implementación llevada a cabo de las dos entidades presentes en la arquitectura, así como las herramientas utilizadas a tal fin. Seguidamente, en la sección 4.2 se describe la implementación llevada a cabo del mecanismo de pruebas de rendimiento, donde se recogen los detalles de los elementos que la componen. Finalmente, en la sección 4.3 se muestran los resultados obtenidos en las pruebas, cuyas conclusiones se describen en la sección 4.4.

El capítulo siguiente, capítulo 5, se describe el diseño del intérprete de órdenes desarrollado. En concreto, se describe sus componentes, sección 5.1, su diseño y funcionalidad, sección 5.2, así como una metodología a seguir para la descripción del entorno y las acciones que se pueden llevar a cabo en él, sección 5.3. En la sección 5.4 se presenta la formulación de los valores que el intérprete tiene en cuenta para concluir cuánto de acertado ha sido su entendimiento de las órdenes del usuario. Finalmente, en la sección 5.5 se describe el plan de pruebas que se llevará a cabo en el capítulo siguiente a fin de validar la propuesta.

La validación del intérprete se lleva a cabo en el capítulo 6. En él se describen las herramientas utilizadas, la metodología implementada para describir el entorno, sección 6.1, así como la simulación llevada a cabo, sección 6.2. Dicha simulación ha utilizado datos recopilados de usuarios reales.

Éstos no han recibido ningún tipo de indicación previa sobre los términos que deben usar para disparar acciones. Finalmente, la sección 6.3 muestra los resultados obtenidos diferenciando entre dos implementaciones del intérprete: sin y con aprendizaje.

Para terminar con las contribuciones, el capítulo 7 recoge la integración del intérprete dentro de la arquitectura de microservicios presentada en el presente trabajo, sección 7.1, así como el plan de pruebas para validar esta integración, sección 7.2, y los resultados obtenidos tras llevar a cabo dicho plan de pruebas, sección 7.3.

Finalmente, el capítulo 8, repasa las principales contribuciones, sección 8.1, conclusiones, sección 8.2 y las posibles líneas futuras de trabajo 8.3.

2 ESTADO DEL ARTE

Atendiendo a los objetivos descritos, en este capítulo se repasa el estado del arte de aquellos conceptos que sirven de pilar sobre los que se asienta la investigación descrita en esta tesis. En primer lugar, se hace una breve introducción al Internet de las Cosas, paraguas bajo el cual se ha desarrollado todo el trabajo. A continuación, se hace un repaso de los entornos de computación distribuida como son la *cloud*, la *fog* y por último la *mist*. En este repaso, se verá el papel que juegan las arquitecturas basadas en microservicios y se presentarán algunas de las implementaciones y soluciones que existen a día de hoy, haciendo especial hincapié en la *descarga computacional*, en inglés *computation offloading*, y las diferentes técnicas que se aplican para llevarla a cabo. Por último, se hace un repaso sobre la interacción entre humanos y entornos distribuidos, donde se resumirán diferentes técnicas de definición del entorno y se verá qué papel juega el lenguaje natural para interactuar con ellos.

2.1 INTERNET DE LAS COSAS: CONTEXTUALIZACIÓN Y DEFINICIONES

El término Internet de las Cosas fue acuñado por primera vez por Kevin Ashton en una presentación que hizo en Procter and Gamble en 1999 sobre el sistema RFID (Radio Frequency IDentification), en la que dijo *El Internet de las Cosas tiene el potencial de cambiar el mundo, de la misma manera que lo ha hecho Internet. Quizá incluso más* [40]. Dos años después de aquella presentación, en 2001, el MIT presentó su visión del IoT para, en 2005, ser definitivamente definido por la ITU (International Telecommunication Union): *Infraestructura mundial para la sociedad de la información, que proporciona la prestación de servicios avanzados mediante la interconexión de objetos (físicos o virtuales) gracias a la interoperabilidad de tecnologías de la información y la comunicación presentes y futuras*[1].

El Internet de las Cosas ha ganado presencia en el mundo académico como industrial en los últimos años [41]. Esto se debe a las capacidades que promete: un mundo en el que cada *cosa*, física o virtual, esté conectada a Internet y sea capaz de comunicarse con otras, de manera transparente para el usuario e, incluso, para el resto de cosas. Uno de los objetivos más relevantes es proporcionar a estas cosas la posibilidad de tomar medidas,

1 https://www.itu.int/rec/T-REC-Y.2060-201206-I/es

de comunicarse entre ellas y de tomar decisiones en base a los datos generados, adaptando el entorno a las necesidades y gustos del usuario sin la necesidad de dar órdenes explícitas [42] [43].

Los actores principales que constituyen el IoT son:

- **Sensores.** Detectan eventos o cambios en el mundo físico que los rodea generando algún tipo de dato de salida (señal) recolectable [44].
- **Actuadores.** Reciben señales y generan cambios en el estado de los objetos [45].
- **Dispositivos.** Un dispositivo es una pieza de equipo con la capacidad, obligatoria, de comunicación y las capacidades opcionales de detección, accionamiento, adquisición, almacenamiento y procesamiento de datos. Los sensores y los actuadores son dispositivos. Los dispositivos se comunican con otros a través de la red. Los dispositivos son el resultado de combinar software con hardware [46].
- **Identificadores.** Los identificadores dan identidad a las cosas de manera inequívoca: ya sea a través de cierto hardware adjunto al objeto (códigos QR, RFID, etc.) o a través de algún tipo de perfil virtual [47].
- **Software de procesamiento.** Es el software encargado del procesamiento de los datos generados en los entornos IoT. Gracias a este procesado, se pueden tomar decisiones de reconfiguración del entorno o generar avisos, entre otras acciones [48].

Extendiendo un poco más el papel que tiene el software en el IoT, es el momento de introducir el concepto de *servicio*. Un servicio es una pieza de software encargada de una serie de tareas. Su uso se ofrece al exterior, de forma que cualquier otro software pueda realizar peticiones al mismo y comunicarse con él [49]. Es decir, si un software requiere realizar ciertas tareas y ya existe un servicio que las realice, puede, en vez de cambiar su implementación para atajar esas tareas, pedirle al servicio que las haga por él, con el consecuente intercambio de datos. El Internet de las Cosas brinda un escenario en el que diferentes dispositivos pueden colaborar para ofrecer diferentes servicios, tanto a los usuarios humanos como a otros dispositivos [50]. El proceso de selección y descubrimiento de los servicios es una tarea complicada. Descubrir un servicio es el proceso de encontrar aquellos servicios disponibles que satisfagan las necesidades del usuario [51]. Descubrir y usar de forma eficiente los servicios debe tener en cuenta el balanceo de carga, la sobrecarga de la red y latencia, la disponibilidad, la escalabilidad o incluso la movilidad [52].

2.2 CLOUD COMPUTING

En 1969, Leonard Kleinrock dijo *"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of 'computer utilities' which, like present electric and telephone utilities, will service individual homes and offices across the country"* [53]. Nacía entonces la idea de proporcionar capacidades computacionales como un servicio [54].

El paradigma computacional que es la computación en la nube surge para dar solución a esa idea, permitiendo a los usuarios acceder a entornos computacionales que aseguran una calidad de servicio, fiabilidad y pueden ser configurables en función de las necesidades del usuario [55]. Existe una gran cantidad de trabajos que intentan dar una definición al concepto de nube. Buyya et al. definen a la nube como "un sistema de computación distribuida consistente en una colección de computadores físicos o virtuales, interconectados entre sí, que son presentados de forma dinámica y virtual, como uno o varios recursos computacionales unificados, en base a un acuerdo de nivel de servicio, negociado entre el proveedor de servicio y los usuarios" [56]. La definición de Vaquero et al. establece que "(...) la nube es una reserva de recursos cuyo acceso y uso es fácil. Estos recursos pueden ser reconfigurados de forma dinámica para ajustarse a una carga de uso. (...) Estos recursos se puede explotar bajo un modelo de pago-por-uso donde ciertas garantías están aseguradas por el proveedor de servicio a través de acuerdos de nivel de servicio" [57]. La nube proporciona el acceso a aplicaciones como servicios a través de Internet, así como acceso al hardware y al software de los centros de datos que proporcionan ese servicio [58]. La nube provee acceso a un conjunto compartido y configurable de recursos, de forma rápida y con un esfuerzo mínimo [59]. Estos recursos se pueden extender de forma dinámica atendiendo a su uso [60]. Permite el acceso a un entorno de computación de alto nivel a empresas o particulares que no pueden permitirse la creación de una infraestructura privada o que no tienen el conocimiento necesario para desplegarlas [61].

Algunas de las ventajas que proporciona la computación en la nube, entre otras, tanto para las empresas y los usuarios en general, son [62] [63] [64] [65]:

- Da acceso a hardware de forma inmediata.
- Gracias a la redundancia, la nube asegura recuperación de datos.
- Despliegue rápido de servicios y fácil integración de los mismos.

- Facilita el escalado de los servicios: puede incrementar los recursos asignados a un servicio en función de su uso (scale-up), así como decrementarlo (scale-down) de forma extremadamente rápida.
- Abstracción de los detalles de la infraestructura IT que hay por debajo.
- Los usuarios no deben mantener los equipos por lo que no deben invertir en su instalación, mantenimiento ni actualización, ahorrándose estos costes.
- Disminuye los costes de entrada en el mercado para aquellas pequeñas empresas que necesitan desplegar un entorno de computación pero no pueden permitírselo de forma privada.
- Permite que haya mayor competitividad entre empresas ya que tienen acceso, a un precio razonable, a servicios que por sí solas no podrían afrontar.

Existen tres modelos básicos de computación en la nube [66] [67]:

- **Infrastructure as a Service (IaaS).** Un proveedor tiene cierta estructura informática y la ofrece como un servicio en Internet. Esta infraestructura tiene centros de datos donde se aloja el hardware necesario para ello. Los proveedores garantizan el acceso a recursos de computación (procesador, memoria RAM, espacio de disco duro, etc.) y estructuras de red, incluyendo cortafuegos, routers y sistemas de seguridad y backup. Los usuarios escogen los recursos que necesitan y el proveedor se ocupa del mantenimiento y de la actualización del hardware de manera transparente para el usuario, así como de satisfacer las necesidades de escalabilidad que puedan surgir [68].
- **Platform as a Service (PaaS).** En el caso de la PaaS, un proveedor de servicio proporciona una plataforma de software de aplicaciones. El usuario puede desarrollar, ejecutar y gestionar sus propias aplicaciones sin necesidad de preocuparse por la infraestructura [69].
- **Software as a Service (SaaS).** Al ofrecer un software como servicio, el proveedor ofrece ciertos programas, por lo general vía web, que un usuario puede usar. Él es el encargado de actualizar el software y de la corrección de fallos, mientras que el proveedor se encarga de mantener las plataformas y la infraestructura [70].

La figura 1 muestra las diferencias de administración entre usuario, en azul, y proveedor, en rojo.

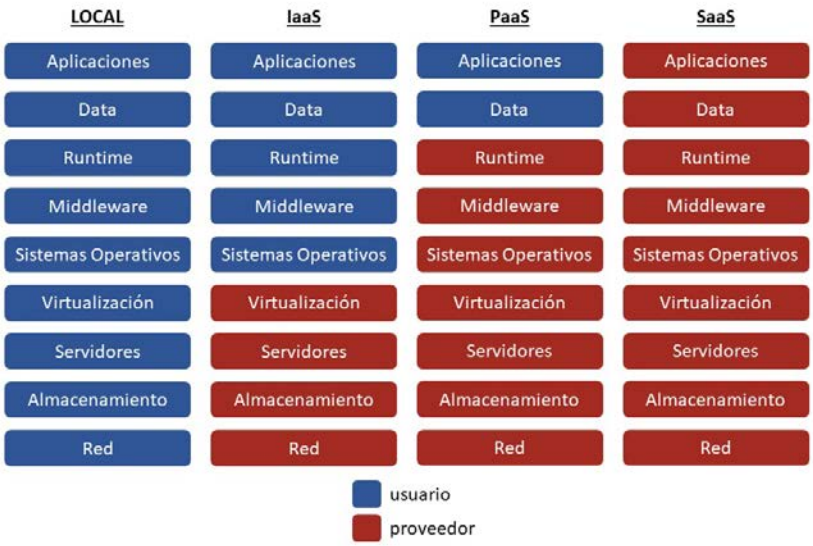


Figura 1: Diferencias de administración entre usuario y proveedor en una configuración local, IaaS, PaaS y SaaS

2.3 EDGE/FOG COMPUTING

Las prestaciones que la computación en la nube puede ofrecer son, en términos generales, mayores a las que se pueden ofrecer de manera local ya que las instalaciones cuentan con un número mayor de dispositivos, los equipos están redundados y éstos tienen mejores prestaciones reuniendo en un mismo entorno elementos de última generación, lo que hace que, a primera vista, realizar tareas de cómputo en estos entornos resulte muy atractivo en términos de velocidad y resistencia a fallo.

Sin embargo, la cantidad de datos que tienen que viajar desde su origen, en el extremo de la red, hasta la nube, cruzando toda la red, puede hacer que partes de la misma se sobrecarguen, sobre todo los puntos de acceso, añadiendo tiempos de espera que no son deseables y generando posibles cuellos de botella en las zonas de la red que están más masificadas, generando retrasos que no son aceptables [71]. Sirva como ejemplo la cantidad de datos que un vehículo conectado a la red puede generar: 2 Gbytes por segundo [72]. Además, los avances en la fabricación de sensores de bajo coste, el desarrollo de protocolos de comunicación entre pequeños dispositivos, sistemas embebidos, actuadores y creación de hardware cada vez más pequeño, ha hecho que el Internet de las Cosas crezca de manera exponencial [73].

Pero no solo se generan datos, también se consumen. Desde electrodomésticos en el hogar hasta las Smart Grids toman estos datos para realizar previsiones o para la toma de decisiones de reconfiguración. Maquinaria industrial, equipamiento médico o las ciudades inteligentes son otros ejemplos de entornos que generan gran cantidad de datos, que necesitan procesarlos y que, además, necesitan consumir datos de su alrededor para llevar a cabo acciones eficientes en un tiempo razonable [74][75][76].

Entonces, la cantidad y heterogeneidad de los datos, así como la necesidad de consumirlos dentro del mismo entorno en el que se generan, aumenta el interés de procesarlos cerca para poder ofrecer las prestaciones suficientes para que los servicios puedan funcionar con la calidad deseada. La realidad aumentada, los sistemas de juegos multijugador, o los vehículos sin conductor, son ejemplos de servicios que necesitan ofrecer un retraso mínimo entre el procesado de los datos y las acciones que realizan para mantener la calidad del servicio y, en casos extremos, hacer que las vidas no estén en peligro [77] [78] [71].

Con la idea de minimizar los tiempos de transmisión de datos, de librar a los centros de computación de cierta carga de trabajo, de minimizar los cuellos de botella generados, así como de poder establecer entornos de computación cercanos al origen de los datos, surge el Edge Computing.

2.3.1 *Definición*

El término edge, en inglés, significa borde. Así, el término Edge Computing (EC) se refiere a la computación realizada en el borde, en este caso, de la red. El borde de la red es la parte de la red que está más cerca del origen de los datos. Este borde está compuesto por los elementos que dan acceso a la red (nodos), que cumplen con el requisito de estar cerca de los datos y de tener la capacidad de llevar a cabo tareas de computación [79].

La computación en el borde de la red intenta solucionar los problemas que ciertos servicios pueden sufrir si delegan su computación en la nube. El Edge Computing aprovecha las capacidades de almacenamiento y procesamiento de una gran cantidad de dispositivos conectados a Internet, con el fin de desplegar un nivel intermedio entre los dispositivos finales y la nube [80]. Estos niveles deben verse como complementarios, teniendo en cuenta las relaciones entre ellos: en el momento en que las tareas no puedan ser llevadas a cabo, o no necesiten satisfacer esas necesidades para las que el Edge Computing está definido, pueden ser delegadas al nivel superior, pudiendo ofrecer los recursos liberados para otras tareas [81]. De esta

manera, se crea un entorno a varios niveles que colabora y trabaja con la idea de ofrecer el procesamiento necesario en el nivel adecuado.

2.3.1.1 *Edge VS Mobile Edge VS Mobile Cloud VS Fog*

Existen varios términos relacionados con este paradigma que, dependiendo de la fuente, pueden o no distinguirse entre sí, de forma que, en muchas ocasiones, se intercambian indistintamente. En algunos textos, el término Edge Computing se refiere al nivel computacional entre los dispositivos finales y la nube, distinguiendo a la Fog Computing de la Mobile Edge Computing, así como de la Mobile Cloud Computing y también de la Cloudlet Computing, siendo todas ellas diferentes implementaciones del Edge Computing [79] [82] [83] [84] [85].

EL término Mobile Cloud Computing propone que los dispositivos móviles deleguen el almacenamiento de datos y su computación a entidades remotas como la Cloud [86] o la Fog [84]. El Cloudlet Computing se basa en el uso de pequeñas infraestructuras de cloud localizadas cerca de los dispositivos finales, habitualmente, ancladas a estaciones base [87]. Del mismo modo que estas implementaciones proporcionan un entorno cercano a los dispositivos finales en donde delegar diferentes tareas computacionales, la Mobile Edge Computing brinda la posibilidad de delegar esas tareas en el entorno RAN (Radio Access Network) de los dispositivos, permitiendo localizar a los dispositivos y conocer su contexto, así como un alto ancho de banda y una baja latencia [88].

Sin embargo, algunos trabajos se refieren al concepto de Edge como Fog y no los distinguen [89] [90] [91], mientras que otros aclaran que el Edge Computing está situado en el borde de la red del proveedor, mientras que la Fog está en el entorno cercano siendo una capa intermedia entre la Cloud y el Edge [92] [93] [94].

En la redacción de este documento, los términos Edge y Fog Computing se consideran lo mismo de forma que, en adelante, con la idea de continuar con la nomenclatura iniciada con el término Cloud (nube), se usará Fog (niebla) como el término general que hace referencia a la computación en el borde de la red.

Aclarado este punto, continuemos con la definición del modelo Fog Computing. Como se ha comentado, se trata de un modelo dividido en capas, o niveles, que permite el acceso a recursos computacionales compartidos. El modelo facilita el despliegue distribuido de servicios y aplicaciones en no-

dos Fog (físicos y virtuales) que se encuentran entre los dispositivos finales y las nubes centralizadas, ofreciendo baja latencia. En la Figura 2 se pueden distinguir las diferentes capas que forman parte de este modelo [95].

En la parte más alta del modelo se encuentra el nivel de la Cloud. En el otro extremo se encuentran los dispositivos finales: como sensores, actuadores, u otros dispositivos inteligentes.

Existe conexión entre los dispositivos finales y los centros de datos. Sin embargo, los dispositivos en sí no dependen de la existencia de la nube. La idea tras la Fog es delegar los servicios de la nube en los nodos intermedios, de manera que la cantidad de comunicación entre los dispositivos finales y la nube sea menor. Los nodos son, entonces, los encargados de recibir datos, procesarlos, analizarlos y almacenarlos.

Los nodos de la Fog están distribuidos geográficamente y son conscientes de su localización. Éstos se encuentran cerca unos de otros y pueden agruparse formando conjuntos, también denominados *clusters*. Dichos nodos, además de encargarse de las tareas computacionales, no dejan de desempeñar su función principal: dar servicio de comunicación a los elementos de la red. Los nodos son switches, gateways, routers y servidores, tanto físicos como virtualizados. Las características que los definen son [96]:

- **Autonomía**. Un nodo es capaz de tomar decisiones de manera local y operar de manera independiente.
- **Heterogeneidad**. Existen diferentes factores que determinan la naturaleza de un nodo, desde el tamaño hasta el tipo de hardware que lo forma.
- **Jerarquía**. Dependiendo de en qué lugar de la fog se encuentren, su comportamiento no será exactamente igual. Los nodos más en el extremo de la red necesitan dar conectividad a muchos dispositivos al mismo tiempo, mientras que los nodos a un salto o más de distancia no están encargados de esta tarea pero, en conjunto, forman un único ente.
- **Administrabilidad**. Cada nodo debe ser accesible y puede administrarse como parte de un todo.
- **Programabilidad**. De manera independiente cada nodo es actualizado y configurado para realizar ciertas tareas que forman parte de un conjunto.

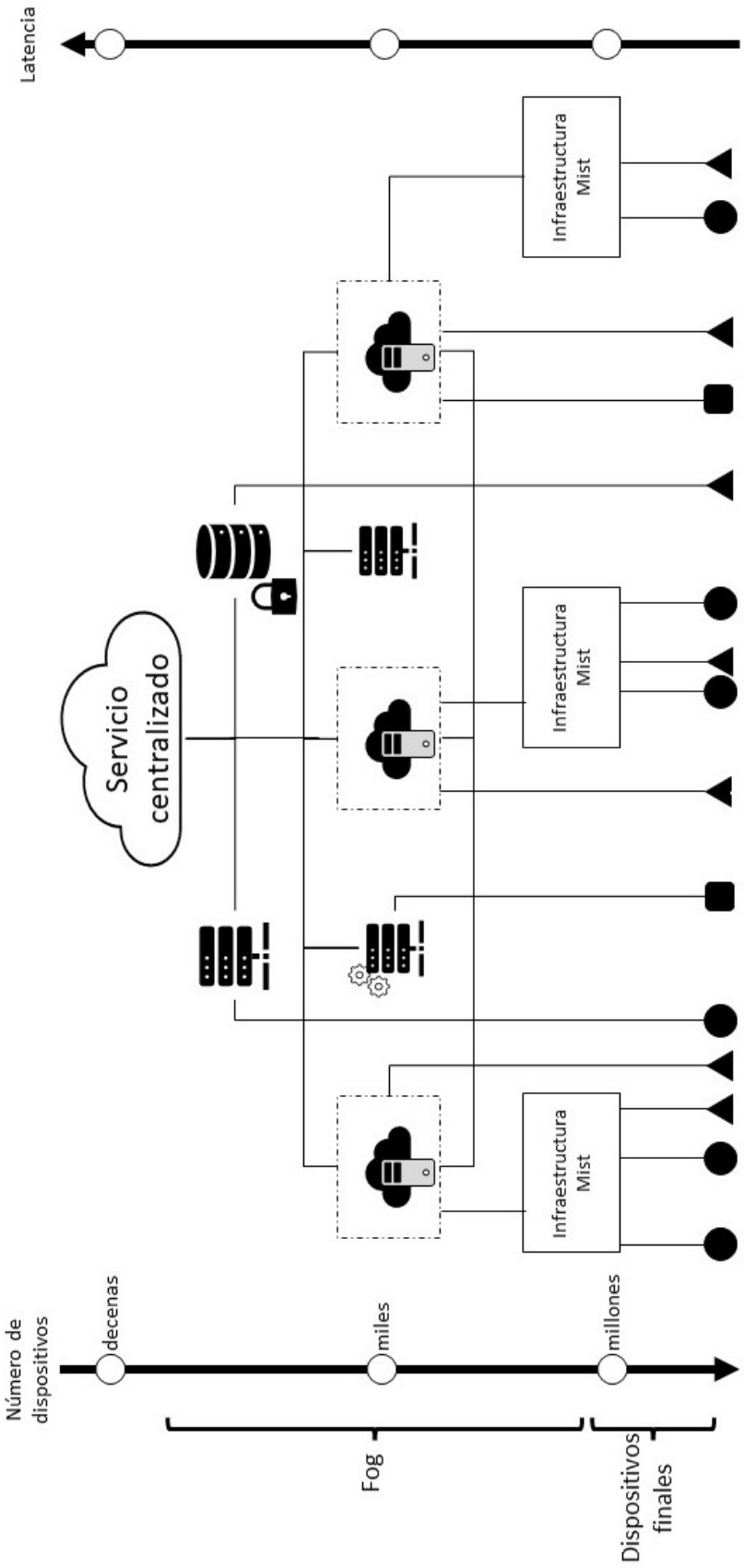


Figura 2: Capas de la fog

- **Conscientes del contexto**. Los nodos saben dónde están y qué otros nodos están a su alrededor. Se pueden organizar en clusters de manera vertical (con nodos de otros niveles) y horizontal (con nodos de similares características).

En contraste con los centros de datos típicos de la nube, donde los servidores de computación están centralizados, en la Fog los recursos están distribuidos en el entorno físico. Con esta distribución, los nodos son capaces de conocer y seguir la localización de los dispositivos conectados ya que se encuentran cerca unos de otros. Este conocimiento sirve para poder ofrecer servicios en tiempo real, dada la baja latencia, y proporcionar movilidad, por ejemplo para redes vehiculares ad-hoc (VANETs) [97] o redes móviles ad-hoc (MANETs) [98]. Además, el conocimiento de la localización de cada nodo permite calcular el camino más corto y eficiente entre nodos.

En cuanto a la heterogeneidad de los nodos, hace falta notar que pueden ser equipos físicos dedicados o virtualizados. Además, las tareas que desempeñan cada uno de ellos no tienen por qué coincidir con los colindantes, ya que sus actividades cambian de manera frecuente: almacenamiento, procesado (de diferente índole y con distintos requisitos) o análisis de datos [99]. Por otro lado, los nodos provienen de distintos fabricantes y no existe un modelo estándar, con lo que, no solo sus características computacionales cambian, sino también su tamaño o el número de interfaces de red y su velocidad [100].

Las infraestructuras Fog deben ser capaces de operar teniendo en cuenta esta heterogeneidad ya que, a fin de cuentas, los nodos deben ser capaces de cooperar incluso entre diferentes proveedores [101]. En términos generales, la interoperabilidad en la Fog implica que cada nodo es capaz de proveer y usar servicios de otros actores de la Fog: nodos, operadores de red o dueños de las infraestructuras. Sabiendo que las aplicaciones y los servicios pueden ser ejecutados en cualquier nodo, es necesario añadir un control en el desarrollo de interfaces de control y de datos, de manera que, los diferentes dominios a los que puedan pertenecer los nodos no resulten un problema [100].

Con la proliferación del Internet de las Cosas, existe una gran cantidad de dispositivos finales que demandan un acceso wireless a los servicios disponibles en la fog. Las tecnologías wireless con las que tienen que ser compatibles deben incluir, entre otras, 3G, WiFi, LTE, ZigBee, Bluetooth o 5G [77].

La Fog, por definición, es escalable porque se pueden crear clusters de nodos, o incluso clusters de clusters, y el número es, a priori, indefinido. Estos clusters son, en definitiva, agrupaciones de recursos, que para el usuario debe ser el producto de un proceso transparente: el usuario ve que tiene acceso a una serie de recursos, pero no sabe dónde se hallan [102]. Además, se debe poder hacer un balanceo de carga entre nodos/clusters: cuando un nodo/cluster alcanza cierto límite, el resto de nodos cercanos puede ayudar a realizar sus tareas. El hecho de que estén físicamente cerca ofrece la posibilidad de ejecutar más tareas en paralelo.

2.3.1.2 *Despliegue y clasificación de los nodos Fog*

En cuanto a los modelos de despliegue de la Fog, estos se pueden clasificar en 4 tipos: privada, comunitaria, pública o híbrida (combinación de las anteriores) [103][104][105].

- **Fog Privada**. Los proveedores de servicios Fog configuran su Fog de manera que sea accesible solo por un determinada organización. Los nodos que formen parte de ella pueden ser propiedad, ser administrados y/o usados, por la propia organización y/o por terceros.
- **Fog Comunitaria**. El modelo de despliegue comunitario implica que la infraestructura se comparte por varias organizaciones. Esto implica la conformidad con las regulaciones y requisitos industriales sobre privacidad, como la norma ISO 27001 [106].
- **Fog Pública**. En este caso, el acceso a los servicios de la Fog se ofrece a cualquier entidad que pague por el servicio, sin tener en cuenta la vinculación entre ellas o con el proveedor de servicio, aunque también puede tener un acceso gratuito. Modelos basados en suscripción también son admisibles en este tipo de despliegue, donde los nodos son propiedad de empresas, entornos académicos o de investigación e incluso del gobierno.

Atendiendo a las características de los nodos que forman parte de la fog, podemos clasificarlos en cuatro tipos:

- **Servidores.** Los nodos de la fog clasificados como servidores son aquellos nodos dedicados principalmente a tareas computacionales, aunque pueden dar conectividad. Su tarea principal es la de proporcionar, específicamente, almacenamiento y computación [107]. Su emplazamiento es muy diverso: paradas de autobús, centros comerciales, parques o la vía pública (p.e.: microservidores [108] [109], nano servidores [110]).

- **Dispositivos de red.** Los dispositivos de la red, como switches, routers o gateways, cuya tarea es la de enrutamiento, forwarding, conversiones analógico/digital, pueden usarse potencialmente como nodos de computación. Para ello, es necesario que los nodos que se quieran usar a este fin, tengan añadidos los recursos hardware para poder ampliarse como procesadores más potentes y programables, o ranuras de memoria extensible [111] [112] [113] [114]

- **Cloudlets.** Las cloudlets han sido diseñadas para extender el servicio en la nube a los usuarios móviles ayudando a la Mobile Edge Computing [115]. Sin embargo, en algunos trabajos se han propuesto directamente como nodos de la Fog que, por lo general, están virtualizados [116] [117].

- **Estaciones Base.** Las estaciones base son los puntos de acceso de los dispositivos finales a la red siendo, al mismo tiempo, los nodos de la red más cercanos. La posibilidad de usarlos como nodos computacionales se ha explotado en gran cantidad de trabajos [118] [119] [120] [121] [122] [123] [124]. Microceldas, Road Site Units, BS, eNB, gNB, son usados para dar servicio Fog.

2.3.1.3 *Herramientas*

Con la proliferación del paradigma del Cloud Computing y los derivados de éste, como la Fog Computing, se han desarrollado diferentes herramientas tanto privadas como de código abierto con la idea de ofrecer infraestructuras y plataformas de computación como un servicio. En este apartado se comentan algunas de las más extendidas.

OpenStack[2] es un proyecto de código abierto que ofrece una solución software para el despliegue de computación en la nube ofreciendo infraestructura como servicio a tres niveles: computación, red y almacenamiento [125]. Las aplicaciones se colocan sobre OpenStack y pueden acceder a la infraestructura a través de su API. Los usuarios lo manejan a través de un panel web, por línea de comandos o a través de una API RESTful. Los elementos que lo componen son: Nova (computación), Cinder (almacenamiento en bloque), Glance (biblioteca de imágenes), Swift (almacenamiento de objetos), Neutron (red), Keystone (identidad) y Heat (orquestación).

Azure Stack Edge (ASE)[3] es una solución desarrollada por Microsoft para ofrecer el uso de la nube híbrida como servicio. El concepto nube híbrida

2 http://www.openstack.org
3 https://azure.microsoft.com/es-es/products/azure-stack/edge/

hace alusión a la capacidad de correr aplicaciones de manera local en una nube propiedad de la entidad que despliega los servicios y, cuando sea necesario, hacerlo en una nube alejada de esta entidad. ASE se encarga entonces de administrar la nube local y de poder fácilmente migrar, si es necesario, los servicios a la nube Azure. ASE ofrece los siguientes servicios a las nubes locales:

1. Infraestructura como Servicio (IaaS): Máquinas Virtuales, Almacenamiento y Red.
2. Plataforma como Servicio (PaaS): Azure APP Service, Azure Functions, Orquestación de contenedores (Azure Service Fabric, Azure Kubernetes Services), SQL Server y MySQL.
3. Software as a Service (SaaS): Orquestación, Contenedores, API Gateway, Databases, Security y Monitoring.

El término Network Function Virtualization, NFV, hace referencia a la virtualización de las funcionalidades de la red, ofreciéndolas como recursos a través de una infraestructura llamada NFV Infrastructure, constituida por elementos de propósito general, donde se alojan estas funcionalidades como firewalls, DNS o caching, y en los que se pueden llevar a cabo tareas computacionales. Uno o varios elementos pueden conectarse para ofrecer un entorno en el que los servicios se pueden desplegar [126] [127] [128]. Un ejemplo de plataformas disponibles es OPNFV, una plataforma de fuente abierta para el desarrollo y despliegue de funciones de red virtualizadas, introducida por la Linux Foundation en 2014.

La manera en la que una red se administra se basa en el acceso a los nodos de la red de manera separada, configurando cada uno de ellos de forma aislada. La idea tras Software-Defined Networking, SDN [129] es la de centralizar el plano de control, sin afectar al plano de datos, haciendo que los nodos deleguen sus tareas de enrutamiento al controlador SDN. Esta manera de administrar la red provee la capacidad de desplegar políticas de control a todos los nodos haciendo a la red, en su conjunto, programable.

Otro ejemplo viene de la mano de Siemens. Siemens Mindsphere[4] es el sistema operativo abierto basado en la nube de Siemens. Ha sido desarrollado para la monitorización de procesos en entornos industriales [130].

4 https://siemens.mindsphere.io/en

Dado que Cisco fue la compañía que introdujo el concepto Fog Computing, es la misma compañía la que lleva desarrollando una gran variedad de productos en este campo. Cisco ha juntado Cisco IOS con Linux[5] en los dispositivos de red, habitualmente routers creando Cisco IOx [131]. Cisco IOx permite servicios Machine-to-machine (M2M) en entornos Fog gracias a sus APIs. De esta manera, las aplicaciones alojadas en los dispositivos de la Fog pueden comunicarse con dispositivos IoT vía protocolos M2M. La filosofía *IOx: bring your own interface* permite integrar sin esfuerzo nuevas tecnologías de comunicación que tengan como común denominador una arquitectura IP. Otra herramienta desarrollada en Cisco es Cisco Data in Motion (DMo)[6], una tecnología de administración y análisis de datos en el borde de la red [132]. DMo proporciona una API RESTful simple basada en reglas para la construcción de aplicaciones. Estas reglas se pueden añadir o eliminar sin necesidad de parar los procesos. Está destinada a ser desplegada en dispositivos de manera distribuida con la idea de controlar los flujos de datos de dispositivos IoT.

Como último ejemplo podemos mencionar Vortex[7], una plataforma para aplicaciones IoT que ofrece compartir datos de una manera segura y eficiente para dispositivos IoT, gateways y la nube [133]. En ella se puede integrar protocolos de paso de mensajes como MQTT o CoAP. Además, ofrece un control de acceso con autenticación simétrica y asimétrica. Cada dispositivo IoT está conectado con un dispositivo Vortex, formando una entidad DDS 2.0 (Data Distribution Service) y es en el dispositivo Vortex donde está instalado el software Vortex.

2.3.2 *Mist Computing*

El desarrollo de la Fog ha hecho que se preste atención a los nodos más cercanos a los dispositivos finales favoreciendo su desarrollo, de manera que estos nodos están más especializados y dedicados, a no solo ofrecer conectividad sino, además, a permitir desplegar tareas computacionales en ellos. Estos nodos periféricos configuran, junto a los dispositivos finales, la Mist [95]. Este paradigma computacional lleva al extremo último de la red, es decir, a los propios actuadores, sensores y dispositivos, la posibilidad de ofrecer y pedir recursos computacionales a dispositivos cercanos [134]. La Mist está diseñada para satisfacer las necesidades de aquellas aplicaciones que requieren una muy baja latencia y muy alto rendimiento [135]. Con

5 https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html

6 https://www.programmableweb.com/api/cisco-data-motion

7 https://www.adlinktech.com/cn/PrismTech-Vortex-DDS.aspx

la ayuda de la Mist, los datos se pueden preprocesar, de forma que solo aquellos datos relevantes serán enviados al siguiente nivel, lo que ayuda a ahorrar tiempo de procesado [136] [137]. Una propuesta interesante es el concepto de "Follow Me Cloud", que ofrece servicios de cloud en los nodos más cercanos al usuario. Basa su funcionamiento en la localización de los dispositivos finales: sigue al usuario a través de la red y desplaza los servicios Cloud haciéndolos permanecer tan cerca como sea posible al usuario [138]. La figura 3 ofrece una visión sucinta sobre los diferentes niveles computacionales, distinguiendo la Cloud, la Fog y la Mist.

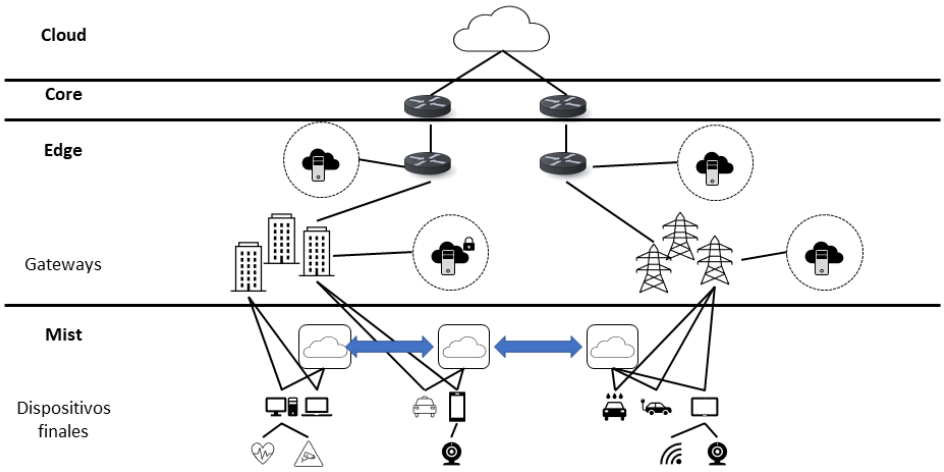


Figura 3: El rol de la Mist en el IoT

2.4 COMPUTATION OFFLOADING

A la hora de decidir qué datos y qué tareas se realizan con ellos en el borde de la red, es necesario comentar que existen diferentes técnicas de división y partición de tareas, así como distintas estrategias para decidir cómo se derivan/delegan las cargas de trabajo de un entorno/dispositivo a otro. Esencialmente, son tres [139]:

1. **Ejecución local total.** La totalidad de las tareas computacionales se realiza en la fuente de los datos [140].

2. **Ejecución externa total.** La totalidad de las tareas computacionales se delega [13].

3. **Ejecución parcial.** Parte de las tareas se delega y otra parte se ejecuta de manera local [141].

Decidir cómo y qué tareas se delegan puede depender de aspectos como decisiones de usuario y sus preferencias, calidad de la conectividad,

capacidades del dispositivo, o cuánto de disponibles están las plataformas cercanas. Las razones por las que se puede decidir migrar tareas computacionales a los nodos de la Fog se pueden clasificar como sigue:

1. **Falta de recursos.** Cuando una aplicación requiere más recursos que los ofrecidos por el dispositivo origen, algunas de sus tareas deben ser migradas a un dispositivo que ayude a su ejecución [142].

2. **Bajar la latencia.** La distancia entre dispositivos puede afectar a la latencia de las aplicaciones [143]. De esta manera es preferible usar dispositivos cercanos.

3. **Balanceo de carga.** Cuando un dispositivo ha alcanzado su máximo nivel de ocupación, las nuevas tareas que le lleguen pueden delegarse a otros dispositivos (preferiblemente cercanos) [144].

4. **Almacenamiento de larga duración y de gran cantidad de datos.** Dada la naturaleza de los dispositivos finales, y su conectividad intermitente (p.e., smartphones), es posible delegar el almacenamiento de los datos generados en dispositivos de la Fog (o de la Cloud) para su posterior consulta o, en el caso de datos públicos, para mantener su disponibilidad ya que los nodos de la Fog tienen una conectividad más estable [145].

5. **Procesado de datos de diferentes fuentes.** Las tareas que requieren una gran cantidad de datos que provienen de una gran cantidad de dispositivos, se pueden delegar en nodos con mayor conectividad para no saturar a los dispositivos finales, ya que, al final, todos los datos deben enviarse al dispositivo que los vaya a procesar [146].

6. **Ahorro de batería.** Los dispositivos finales tienden a depender de una batería y a no estar conectados a una fuente de alimentación continua, con lo que el despliegue de aplicaciones y servicios en ellos implica una alta necesidad energética. Realizar delegación de tareas a dispositivos de la Fog ayuda a salvarla [147].

Existen diferentes trabajos que analizan este problema en el escenario de la delegación a la nube. CloneCloud [148], MAUI [149] y ThinkAir [150], son soluciones donde cada dispositivo móvil tiene un clon en la nube en el que se se instancian las aplicaciones, en vez de hacerlo en el dispositivo móvil. Existen otros trabajos [151] [152] que proponen dividir las tareas computacionales en unidades computacionales y son éstas las que se derivan y migran a la nube con la idea reducir el consumo de energía.

En el caso de la MEC, existen trabajos que analizan este problema, donde las tareas se derivan a las estaciones base [153]. El trabajo presentado por Maofei Deng et al. [154] propone una manera de desgranar tareas de dispositivos móviles con la idea de derivarlas a nodos SCceNB. El trabajo presentado por F. Lobillo et al. [155] está centrado en las celdas LTE donde se despliegan pequeños servidores donde realizar las tareas. Los trabajos de [156] [157] [158] también presentan soluciones para realizar computation offloading en la MEC, donde las tareas se dividen y paralelizan describiendo además las dependencias entre ellas, modelando éstas con la idea de minimizar el tiempo de ejecución y maximizar el ahorro de energía. Desde el punto de vista del desarrollador, existen diferentes trabajos que proponen soluciones en las que el propio código de la aplicación está etiquetado para que, en caso de necesidad, la división se pueda realizar usando estos puntos de corte [159] [160].

El trabajo presentado por Liang et al. propone una arquitectura para realizar offloading basada en SDN [161]. Hacen uso de la posibilidad de crear políticas para diferentes tipos de flujos de tráfico para diferente tipo de aplicaciones. Del mismo modo se pueden generar políticas para realizar el offloading.

La propuesta de Fricker et al. [162] se basa en el balanceo de carga entre datacenters de la Fog. Cuando un datacenter está saturado puede bloquear las peticiones que se le hacen para realizar offloading, de esta manera, esas peticiones se reenvían a otros datacenters colindantes. Este trabajo analiza cómo variar la tasa de bloqueo en función de la ocupación de los datacenters.

Aura [163] es un mecanismo con el que se pueden establecer pequeñas fog ad-hoc para dispositivos IoT usando los dispositivos colindantes. Estos dispositivos pueden ser vehículos, teléfonos móviles, relojes inteligentes y así seguidamente, que eventualmente tienen disponibles sus recursos. Para fomentar el ofrecimiento de estos recursos, los usuarios reciben moneda criptográfica. El trabajo presentado por Pu et al. [164] genera fogs efímeras donde los dispositivos colaboran para llevar a cabo las tareas. Las decisiones de delegación las toman los nodos de la red. Con la idea de evitar la sobreexplotación de la plataforma generada, queda constancia de qué dispositivos han requerido realizar offloading de manera que, en el futuro, estos se elijan prioritariamente para saldar su deuda. Liu et al. insisten en la importancia de proponer incentivos a los dispositivos para poder realizar delegación de tareas [165], ya que consideran complicado motivar a los dispositivos para que realicen tareas de otros. La creación de femtoclouds

se propone en el trabajo Habak et al. [166]. Estas plataformas se basan en el uso de recursos sin usar de los dispositivos cercanos. Del mismo modo que en los trabajos anteriores, los dispositivos reciben crédito dependiendo del contexto en el que estén: cafeterías, parkings, campus universitarios, etc.

Realizar offloading dividiendo las tareas en microtareas es parte del trabajo presentado [167] por Meurisch et al. En este trabajo se analiza el problema de realizar offloading en contextos desconocidos o en los que no se sabe si se puede confiar, una situación habitual para los dispositivos móviles. En su metodología, las microtareas se usan como prueba para los dispositivos cercanos disponibles. Si el rendimiento de la red es aceptable, se considera que se pueden migrar tareas de mayor tamaño.

2.5 ARQUITECTURA DE MICROSERVICIOS

Hasta este momento se ha contextualizado el presente trabajo, haciendo un repaso del concepto *computación distribuida*, queriendo aclarar los diferentes niveles de ejecución que se diferencian en la bibliografía y los términos que frecuentemente aparecen junto a ellos. Sin embargo, en esta sección se va a mover el foco al concepto de *microservicio*. En las siguientes líneas se busca presentar el papel que los microservicios pueden jugar en los entornos de computación distribuida, así como presentar algunas de las herramientas que este trabajo ha utilizado.

2.5.1 *Definición*

El desarrollo de sistemas más complejos y más rápidos ha hecho que las implementaciones monolíticas puedan no satisfacer sus requisitos: baja latencia, alta velocidad de procesado, entre otros. Para resolver este problema, surgen las arquitecturas basadas en microservicios [168].

En las arquitecturas monolíticas, cada módulo que compone un servicio o una aplicación comparte recursos con los demás módulos del propio servicio o de la aplicación. Así mismo, cada uno de ellos no se puede ejecutar de forma aislada [169]. Toda la aplicación está alojada en el mismo lugar y cualquier cambio en cualquier nivel afecta a su funcionamiento general pudiendo, en caso de fallo, dejarla en estado no disponible, lo que hace de las arquitecturas monolíticas complicadas de desplegar, mantener y de actualizar [170]. Con la idea de simplificar estos procesos, surge la arquitectura de microservicios [171]. Las arquitecturas basadas en microservicios se basan en la descomposición de los servicios o aplicaciones en bloques más pequeños, los microservicios, cada uno de ellos encargado de una tarea.

Estos microservicios cooperan, componiendo los diferentes servicios [172]. La figura 4 muestra una comparación esquemática de ambos conceptos.

Su naturaleza distribuida hace de ella una arquitectura atractiva para la computación en la cloud y en la fog [173] [174]. Un microservicio aporta, o lleva a cabo, una tarea concreta. Cada microservicio se puede desplegar de forma independiente, de forma que las actualizaciones se realizan de forma más eficiente. Del mismo modo, son escalables de forma independiente: se puede asignar recursos al microservicio más demandado en vez de a todo el conjunto. Además, tiene mayor tolerancia a fallos ya que se puede aislar el microservicio que falla, previendo fallos en cascada que puedan dejar sin servicio al conjunto.

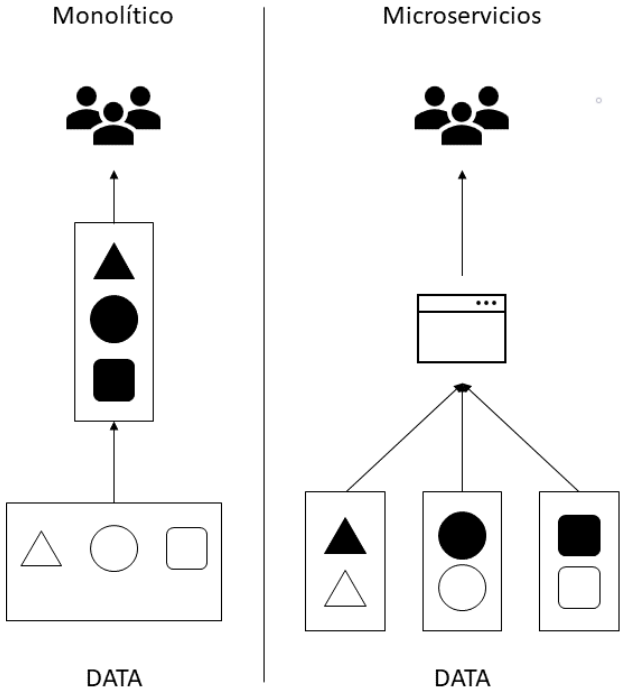


Figura 4: Arquitectura monolítica VS arquitectura de microservicios

2.5.2 *Virtualización: contenedores VS máquinas virtuales*

Los entornos de computación como la nube se basan en el uso de la virtualización para ofrecer sus servicios, de forma que diferentes aplicaciones se pueden alojar en la misma máquina física, de forma aislada, aunque cada una de ellas requiera diferentes recursos para funcionar correctamente [175]. Existen dos tipos de tecnologías para virtualizar estos entornos de

ejecución: los *hypervisors* y los *contenedores* [176].

La virtualización basada en hypervisors es capaz de alojar diferentes máquinas virtuales independientes aisladas entre sí y de la máquina que las hospeda, el host [177]. Al estar aisladas, cada máquina virtual puede tener un sistema operativo diferente, incluso distinto al del host. Dentro de cada sistema operativo se despliegan los diferentes servicios o aplicaciones. En la figura 5 se esquematiza cada una de estas partes.

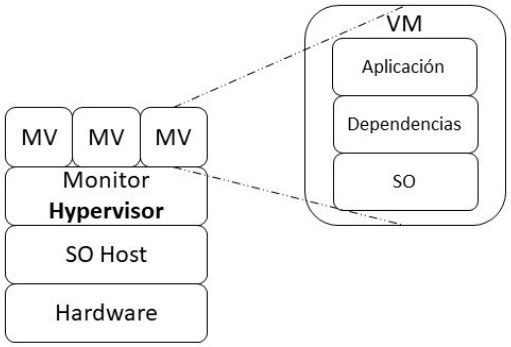


Figura 5: Virtualización con hypervisor

Los contenedores son pequeños entornos de ejecución aislados que contienen las especificaciones básicas para que el software contenido en él funcione: desde librerías a bases de datos. De esta manera, cada contenedor es independiente y cualquier problema en su interior no afecta al dispositivo que lo aloja, ya que el contenedor es un entorno restringido con unos límites definidos [178]. En la figura 6 se muestra cada uno de los componentes de una virtualización basada en contenedores.

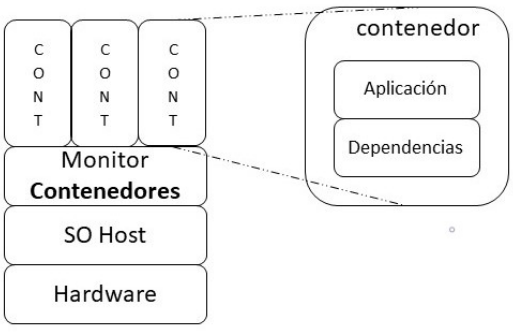


Figura 6: Virtualización con contenedores

Por un lado, en la imagen 5 podemos observar que cada máquina virtual requiere de un sistema operativo entero para funcionar, mientras que en la

6 no es necesario contar con un sistema operativo entero. Un sistema operativo contiene las dependencias necesarias que dan soporte para que un gran número de aplicaciones puedan funcionar en él (algunos ejemplos de dependencias son las bases de datos, librerías, APIs, etc). No obstante, las aplicaciones solo requieren un conjunto de ellas de manera que, si solo se necesitase desplegar una aplicación concreta, muchas de las dependencias que ofrece el sistema operativo no llegarían a necesitarse. Por otro lado, en el caso de los contenedores, las dependencias de los contenedores son las justas para que la aplicación funcionen. Si se necesitase desplegar una aplicación concreta, las dependencias del contenedor serán las necesarias para dar soporte a la aplicación.

La diferencia en el funcionamiento de ambos enfoques radica en lo siguiente: los contenedores no emulan el hardware físico, a diferencia de la máquina virtual, aunque los contenedores comparten el kernel con el host. Esto hace que los contenedores consuman menos recursos [179]. El funcionamiento de los contenedores se apoya en el uso de los "namespaces" y de los "cGroups" de Linux. Los namespaces aislan el sistema de archivos al que tiene acceso un grupo de procesos. Los cGroups organizan a los procesos en un árbol jerárquico, limitando el uso de los recursos: al desplegar una aplicación en un contenedor sus namespaces quedan aislados de los de otros procesos del sistema. Debido a esto, generalmente, las máquinas virtuales son más pesadas que los contenedores y requieren de más tiempo para su arranque, minutos frente a segundos [180]. Los contenedores proporcionan un entorno más flexible para el despliegue de aplicaciones que deben responder en tiempo real [178]. Además, es habitual que los contenedores alojen un único servicio o aplicación, lo que los hace más ligeros, minimizando el tiempo de despliegue [181] haciendo de los contenedores una herramienta muy atractiva para el despliegue de arquitecturas basadas en microservicios: cada microservicio está instanciado en un contenedor diferente [182] [183].

En el siguiente apartado se describen algunas de las herramientas más extendidas para el despliegue de arquitecturas de microservicios en contenedores. Algunas de ellas se utilizan en entornos de computación distribuida como la Mist, la Fog y la Cloud.

2.5.3 *Herramientas*

2.5.3.1 *Virtualización de contenedores*

Algunas de las herramientas para la virtualización de contenedores más extendidas son las siguientes:

OPENVZ OpenVZ es una tecnología de contenedores para Linux. Cada contenedor creado se considera un servidor independiente, conocidos como Virtual Private Servers. OpenVZ es de código abierto, accesible bajo GNU GPL. Cada contenedor cuenta con sus direcciones IP, espacio de memoria, procesos propios, aplicaciones y librerías [184].

LXC LXC es el sucesor de OpenVZ. Permite la virtualización al nivel del sistema operativo, permitiendo crear diferentes entornos Linux, virtuales y aislados, dentro de una única máquina. Los contenedores no emulan hardware. La única emulación que ocurre es la creación del sistema operativo Linux contenido. De esta manera, la creación de los contenedores es mucho más rápida que la creación de máquinas virtuales [185].

DOCKER Docker[8] es una infraestructura muy popular para construir, empaquetar y correr aplicaciones dentro de contenedores. Las aplicaciones están empaquetadas en forma de imágenes que contienen el sistema de ficheros necesario para que la aplicación funcione: bibliotecas, ejecutables, archivos de configuración, etc. Las imágenes se encuentran en repositorios que pueden ser accedidos desde cualquier dispositivo. Para desplegar un contenedor, en primer lugar se descarga la imagen del repositorio y, a continuación, se despliega el contenedor con la imagen en su interior. El encargado de administrar los contenedores, desde el punto de vista de aislamiento de red y espacio de recursos, es la instancia Docker que está corriendo en el host [186].

BALENA Balena está diseñado para escenarios IoT. Está basado en el proyecto Moby, creado por Docker, con la idea de acelerar el despliegue de contenedores. Se trata de una sistema de bloques que permiten al usuario configurar un entorno basado en contenedores [187]. Se puede considerar a Balena como una versión ligera de Docker ya que elimina las características menos usadas en Docker [188]

8 https://www.docker.com/

2.5.3.2 *Orquestación de contenedores*

En los entornos en los que varias máquinas tienen desplegadas tecnologías de virtualización de contenedores, es necesario contar con una herramienta que permita coordinarlas, esto es, entre otras tareas, arrancar los contenedores, eliminarlos, actualizarlos o migrarlos de una máquina a otra [189]. Estas tareas se engloban en el concepto de orquestación. La orquestación permite, además, establecer políticas de acceso, así como reglas de adaptación (del tipo "si sucede algo, hacer esto"). La orquestación posibilita un despliegue ágil y rápido, así como una administración eficiente del entorno. Algunas de las herramientas de orquestación de contenedores más usadas son las siguientes:

DOCKER SWARM Un enjambre, swarm, es un conjunto de máquinas con una instancia de Docker corriendo en cada una de ellas. Docker Swarm[9] es la parte de Docker encargada de realizar la orquestación entre máquinas físicas y virtuales. Cuando un nuevo contenedor se quiere añadir al enjambre, Docker Swarm aplica alguna política definida para decidir en qué máquina alojarla. Del mismo modo, las máquinas que forman parte del enjambre sirven como respaldo al resto de máquinas, ofreciendo redundancia: si un nodo cae, los servicios dados por los contenedores alojados se migran a otra máquina. Es muy habitual su uso a la hora de desplegar aplicaciones con balanceo de carga que deben responder de manera automática a las caídas [190].

KUBERNETES Kubernetes[10] es el sistema de administración de contenedores desarrollado por Google. A diferencia de otros orquestadores desarrollados por Google (como Borg, posteriormente Omega), Kubernetes es de código abierto. Es una infraestructura para sistemas distribuidos que se puede usar en clouds compuestas por pequeñas Raspberry Pi o en sistemas que cuenten con ordenadores de última generación [191].

APACHE MESOS-MARATHON. Es necesario mencionar que la combinación de Kubernetes y Docker se ha vuelto el estándar de facto. Aunque existen otros como Apache Marathon[11]. Apache Marathon está diseñada para trabajar sobre Apache Mesos[12] y DC/OS. Apache Mesos hace posible ver un datacenter como un único lugar donde existen recursos, actuando como sistema operativo del centro de datos [192].

9 docs.docker.com/engine/swarm/
10 https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/
11 https://mesosphere.github.io/marathon/>
12 mesos.apache.org

2.5.3.3 *Contenedores, Fog y Cloud*

En la sección 2.5 se ha visto por qué el uso de contenedores resulta ser un método atractivo para el despliegue de microservicios, del mismo modo que los microservicios resultan atractivos para la computación en la fog y en la cloud. Existen numerosos trabajos que ahondan en el despliegue de contenedores en entornos Fog, de los cuales se comentan algunos de los más relevantes.

AGILE El trabajo desarrollado por Dolui et al. [193] presenta AGILE, un framework de código abierto para gateways IoT, que ofrece administración de dispositivos, protocolos, almacenamiento de datos, seguridad y control de acceso. Está desarrollada bajo una arquitectura de microservicios donde cada servicio está desplegado dentro de un contenedor. Los contenedores que forman parte del núcleo de los gateways tienen implementados los protocolos ZigBee y BLE. Los dispositivos se administran a través de un manager que está encargado de ofrecer las diferentes funcionalidades de cada dispositivo a los demás. Se pueden desplegar protocolos a través de una interfaz, así como administrar implementaciones de los protocolos ya desplegados. También es posible administrar las funcionalidades del gateway a través de una interfaz de usuario, que permite controlar los servicios desplegados.

KURA GATEWAY Kura[13] es un framework de código abierto para aplicaciones IoT basado en Java/OSGi. Ofrece acceso a través de un API a las interfaces hardware de los gateway IoT (puertos, GPS, watchdog, GPIOs, I2C, etc). Kura usa MQTT como su protocolo base. Los componentes de Kura permiten mecanismos de control en los gateways con creación de firewalls, operaciones NAT, VPN, etc. El trabajo realizado por Bellavista et al. propone una solución basada en Kura y Docker Swarm [194].

FOGERNETES Fogernetes [195] (una extensión de Kubernetes) es una plataforma que permite la administración y el despliegue de aplicaciones en la Fog con requisitos específicos en dispositivos heterogéneos con diferentes capacidades: mapea los recursos disponibles con las necesidades de los servicios desplegados en los nodos. Lo hace a través de etiquetas que tienen tanto los recursos como los dispositivos.

FOGGY. Foggy [196] es una plataforma para la orquestación y administración de recursos IoT en infraestructuras Fog. Se usan contenedores Docker para implementar sus funcionalidades: negociación de recursos y orques-

13 http://www.eclipse.org/kura

tación de tareas, pudiendo crear reglas basadas en medidas sacadas de la monitorización del sistema o basadas en otro tipo de criterios. Está pensado para los requisitos que las redes 5G demandan.

2.5.4 *Análisis de rendimiento de microservicios*

Las arquitecturas basadas en microservicios buscan crear microservicios pequeños y ligeros, dedicados a una única tarea. De esta manera, para crear un servicio más complejo, los microservicios que lo forman deben comunicarse entre sí, lo cual afecta a su rendimiento. Nos encontramos entonces ante un problema: a más microservicios más fácil resulta la escalabilidad, pero mayor es el intercambio de datos [197]. La tarea de dividir un servicio complejo en microservicios conlleva un problema particular: es necesario poder evaluar cada uno de ellos por separado al mismo tiempo que es necesario poder evaluar el funcionamiento en conjunto. Existen muchas estrategias para la evaluación de arquitecturas basadas en microservicios, de las cuales tres son las más utilizadas: evaluación de integración, evaluación End-2-End y evaluación de unidades [198].

La evaluación de integración consiste en comprobar que el sistema, en conjunto, satisface las especificaciones requeridas [199]. En el caso de los microservicios, realizar este tipo de evaluación favorece el verificar que i) los microservicios consiguen llevar a cabo las tareas colaborativas para las que estén diseñados; ii) que la comunicación entre ellos funciona correctamente, sin pérdida de datos; iii) que terceras partes pueden hacer uso de los servicios prestados [200].

En el caso del End-2-End, se evalúa el sistema entero realizando tests a cada una de las funcionalidades por separado [172]. Este método busca simular situaciones reales bajo las cuales evaluar al sistema.

Por último, la evaluación de unidades consiste en evaluar partes pequeñas del sistema. Existen dos herramientas muy extendidas para el testeo de unidades: JUnit y NUnit [201]. JUnit está pensado para evaluar software en código Java [202], mientras que NUnit sirve para evaluar .NET [203]. A través de anotaciones en el código, se puede especificar qué partes se ejecutan antes del test, durante o después. Además, se puede establecer qué respuestas se esperan y se comparan con las respuestas reales, mostrando esta comparativa a través de una interfaz gráfica. En el caso de los microservicios, se evalúa un microservicio concreto, proporcionándole los datos de entrada que necesite, sin considerar al resto del sistema [204]. No obstante, dado que los microservicios pueden depender de datos suministrados

por otros para realizar sus tareas, es necesario idear alguna estrategia para proporcionarle esos datos sin la necesidad de desplegar el resto de microservicios, es decir, es necesario generar datos falsos (en el sentido de que no han sido recopilados o calculados) de entrada [205]. Existe una metodología que responde a esta idea: "mocking". Mediante mocks, se crean elementos que simulan el comportamiento de los microservicios de los que depende el microservicio bajo evaluación [206]. Cada microservicio que se quiere simular es un mock. El uso de mocks está bastante extendido ya que su uso puede hacer más cómodo el proceso de evaluación de unidades [207] [208]. Cada mock se configura de forma aislada para responder a una peticiones concretas, devolviendo unos datos preconfigurados que no necesita generar. Existen varias herramientas para ello como Mockito[14], EasyMock [209], JMock (Java) [210] o Mock[15] (Python), que proporcionan diferentes APIs para crear los mocks y para establecer los valores de respuesta.

2.6 HUMANOS Y SU ENTORNO

Hasta ahora, este capítulo ha servido para contextualizar la presente tesis respecto a los trabajos relacionados, introduciendo diferentes conceptos en los que este trabajo se basa que, principalmente, son: 1) la computación distribuida, y 2) la descarga y migración computacional entre dispositivos. Situándonos ahora en el extremo de la red, en los dispositivos finales con los que los seres humanos interactúan, continuamos con el análisis del estado del arte atendiendo no solo a la interacción entre humanos y dispositivos finales, sino también entre humanos y los entornos que los rodean.

2.6.1 *El entorno: cómo describirlo*

Los humanos vivimos rodeados de dispositivos inteligentes. Éstos muestran sus capacidades de forma que los humanos puedan decidir si usarlos como herramientas para llevar a cabo alguna tarea. Que los humanos interactúen con su entorno es uno de los pilares de la computación ubicua [211]. Hoy en día, un individuo cuenta con acceso a más de un dispositivo personal [212], no solo móviles, PCs de mesa, portátiles o tablets, sino relojes, sensores que toman datos en tiempo real o e-books, sin olvidarnos de aquellos que están embebidos en los lugares en los que una persona está, como cámaras de vigilancia o displays de información, así como electrodomésticos o equipos de oficina, por mencionar algunos ejemplos. Y el

14 https://site.mockito.org
15 https://docs.python.org/3/library/unittest.mock.html

número sigue creciendo, añadiendo dispositivos con propósitos específicos, como ropa inteligente, o gadgets que ayudan a mejorar la vida de personas con pérdida de visión o auditiva, e incluso exoesqueletos que permiten a una persona coger pesos que, sin ellos, sería imposible portar. Del mismo modo que crece este número, crece la necesidad de permitir el intercambio de información entre ellos, y mediar en la interacción entre humanos y dispositivos.

Hoy en día, estos dispositivos son controlados de una gran variedad de formas [213], como por ejemplo, interfaces gráficas y pantallas táctiles, teclados, dispositivos manuales equipados con giroscopios, u otros que capturan la voz o el movimiento ocular. De entre estos, este trabajo pone el foco en aquellos que permiten al usuario usar el lenguaje natural, por ejemplo a través de la voz o texto, donde una importante cantidad de recursos en investigación se han dedicado ya que, para un ser humano, resulta algo innato, natural y, si se hace correctamente, eficiente [214].

Sin embargo, las soluciones en estos campos, como la captura de la voz (que no el reconocimiento de la voz, cuya intención es saber quién habla, mientras que la de la captura es reconocer qué se dice), deben lidiar con ciertos malentendidos entre los fabricantes de los dispositivos y los usuarios [215]. Por un lado, los fabricantes deben incluir una definición de los términos que el usuario debe conocer para dirigirse a los dispositivos [216]. Por otro lado, los usuarios deben saber un conjunto limitado de términos o sentencias que disparen ciertas acciones [217]. Estas circunstancias resultan una tarea particularmente difícil para el fabricante: debe considerar y seleccionar los términos adecuados considerando uno o varios idiomas y sus aspectos culturales (dependientes de la localización del usuario, e.g.: inglés entre EEUU e Inglaterra o español entre España y Latino América). Es posible, entonces, que el comportamiento de los dispositivos no sea el deseado dadas las diferencias entre los significados de las palabras, o incluso dado cierto orden de las mismas en una frase: es posible que el orden no encaje con la definición dada cuando, desde un punto de vista humano resulta fácil comprobar que son iguales: dos órdenes como “apaga la televisión” o “televisión fuera” significan lo mismo. Es posible que palabras, o frases, que están cerca desde un punto de vista sintáctico o semántico no funcionen o incluso funcionen mal, disparando alguna acción no deseada, lo que decrementa la experiencia de usuario.

A ello debe sumarse el hecho de que los usuarios cambian de localización y, por tanto, de entorno muy a menudo: hogar, oficinas, centros de estudio, edificios públicos o privados, calles, parques o transporte, entre otros. Bajo

el paradigma del Internet de las Cosas, estos entornos y sus contextos se pueden agrupar en los siguientes tres tipos [218]:

1. Personal. Donde los dispositivos son usados por una única persona, como wearables para monitorización de la salud.

2. Privado. Donde el número de usuarios y de dispositivos están delimitados y son bien conocidos, como en un hogar o un vehículo privado.

3. Público. Donde el número de usuarios puede o no ser precalculado y están en constante cambio (i.e.: el flujo de dispositivos es alto). Algunos ejemplos son las oficinas, centros de estudio, edificios públicos, tiendas o la ciudad inteligente.

En los entornos personales y privados, los dispositivos no cambian con mucha frecuencia. Podría asumirse que los usuarios están familiarizados con el entorno. Sin embargo, en un entorno público, el número potencial de usuarios y de dispositivos es muy elevado, del mismo modo que la cantidad de entornos públicos a los que un usuario se desplaza es alto. Como resultado de esto, resulta complicado para los usuarios interactuar con los dispositivos, del mismo modo que para los fabricantes resulta complejo incluir conjuntos de términos que incluyan tantos como sean necesarios para compatibilizar sus productos con los diferentes (en términos culturales) usuarios potenciales.

2.6.1.1 *El problema principal*

Es un hecho que los usuarios quieren sentirse cómodos con el uso de los dispositivos. Este es un requisito básico que, de no ser atajado, puede resultar en un experiencia molesta o frustrante para aquellos usuarios que no estén familiarizados con los sistemas. Muchos usuarios se quejan por la curva de aprendizaje que sus nuevos dispositivos requieren.

Los términos elegidos por los desarrolladores pueden no estar alineados en la misma dirección que aquellos que los usuarios elegirían. Las personas encargadas del diseño deben ser conscientes de que las imágenes, palabras clave o términos que identifican un dispositivos o un servicio deben ser fáciles de entender por cualquier usuario. Por ejemplo, en un editor de texto, los iconos de guardar o abrir un documento vienen acompañados por etiquetas o descripciones breves, de forma que resultan más inteligibles.

Existen diferentes trabajos que sirven para ilustrar este problema. Un ejemplo en esta línea de investigación es el trabajo desarrollado por Matsuda et al. [219] en el que se propone una interfaz de usuario para un sistema

audiovisual controlado por lenguaje natural. El usuario realiza peticiones al sistema y éste le hace ciertas preguntas con la idea de concluir qué es lo que precisa el usuario. En este caso, existe un problema de base, y es el hecho de que las preguntas hacen referencia a partes del sistema, de forma que el usuario debe conocerlo o estar familiarizado con él. Otro ejemplo es el trabajo desarrollado por Park et al. [220], donde se propone una interfaz de usuario para dispositivos móviles basado en el uso del lenguaje natural, donde las peticiones se traducen a comandos entendibles por el dispositivo basándose en patrones, lo que implica que órdenes que no encajen con dichos patrones no serán entendidas.

2.6.1.2 *Métodos de descripción*

En el Internet de las Cosas es necesario saber identificar de forma inequívoca a cada uno de los elementos que están conectados. Con esta idea, los elementos, que no cuenten ya con uno, son frecuentemente aumentados con metadatos que ayudan a describirlos. Es decir, identificar a un elemento no se basa exclusivamente en asignarle un código identificativo, sino en, además, añadir un conjunto de datos que profundicen en su descripción. Estos datos responden a diferentes métodos organizativos. Los más habituales son las *ontologías* y las *etiquetas*.

ONTOLOGÍAS Una de las formas más habituales de organizar la información relacionada con un elemento es hacer uso de ontologías. Una ontología es una especificación formal de un concepto compartido [221]. Las ontologías permiten representar un área de conocimiento, permitiendo comparar dos conceptos y cuánto de cerca están el uno del otro [222]. Esto implica, irremediablemente, que, para que una ontología sea útil y usable por varias entidades, los conceptos representados sean un estándar. Desde el punto de vista del IoT, esto puede resultar en un problema dada la gran cantidad de dispositivos que coexisten, las diversas capacidades que tienen, así como el número de fabricantes, ya que las diferentes ontologías deberían recoger todas las posibles variantes de forma precisa y concienzuda. Las ontologías son herramientas muy potentes y útiles en entornos reducidos, pero su procesamiento resulta caro en términos de recursos, y esto es problemático ya que a más conceptos a describir, mayor gasto. No obstante, existen un gran cantidad de trabajos que usan ontologías para describir desde diferentes enfoques un entorno IoT, algunos de los cuales se describen a continuación:

Es sabido que los deseos y necesidades de los usuarios son los desencadenantes de diferentes tipos de operaciones. Para poder desarrollar una

ontología que recoja estas actividades, es necesario tener en cuenta aspectos como los perfiles de usuario o la privacidad. El trabajo desarrollado por Preuveneers et al. [223] propone una ontología para describir entornos inteligentes llamada CoDAMoS, centrada en la descripción de cuatro componentes: usuario, entorno, servicio y plataforma; dividiendo el nivel de abstracción en dos: tareas y actividades. SOUPA, la ontología estándar para aplicaciones ubicuas [224], representa diferentes agentes que tienen asociadas necesidades e intenciones, así como una ubicación en tiempo y espacio, acciones, y políticas de seguridad y privacidad, todo recogido dentro de entornos de computación ubicua. CoBrA-Ont [225] es una extensión del trabajo anterior, enfocado en la privacidad del usuario, restringiendo cómo se comparte la información entre sensores y dispositivos. El trabajo desarrollado por Riboni et al. [226] propone una arquitectura para el reconocimiento de actividades, COSAR, basado en el uso de una ontología para las actividades humanas, ActivO.

Además de ontologías relacionadas con las actividades, existen otras que ayudan a describir un contexto y una situación: si se conoce el contexto, se puede llegar a deducir la situación en la que se encuentra un usuario. Chen et al. [227] propone una ontología para modelar entornos físicos, sensores, dispositivos y servicios. El trabajo desarrollado por Daniele et al. [228] propone SAREF, una ontología centrada en el concepto de dispositivo "un objeto tangible diseñado para realizar una tarea particular en hogares, edificios públicos u oficinas. Para llevar a cabo esa tarea, el dispositivo lleva a cabo una o más funciones". Es necesario comentar, llegado este punto, una de las ontologías principales para describir sensores: Semantic Sensor Network (SSN) Ontology [229]. Ésta proporciona descripciones de conceptos como despliegue, dispositivos y datos. Como fruto de este trabajo, se han desarrollado otras ontologías como SOSA[16] que, a su vez, a dado a luz a otras como el trabajo presentado por Kotis et al. [230], donde se describe una ontología específica para IoT, expansión de SSN. Otra ontología que ha partido de SSN es IoT-Lite Ontology[17], presentada por Bermudez et al. [231], la cual está focalizada en la interoperabilidad entre diferentes plataformas IoT, y extendida por Kikuchi et al. [232], donde los sensores se abstraen como componentes. Ahvar et al. presenta FUSE-IT [233], una ontología que combina diferentes modelos, incluyendo SAREF y SSN, para proporcionar un visión unificada de los hogares inteligentes.

Dado que las ontologías se han convertido en una solución adoptada por muchos trabajos, cada vez más investigaciones han procurado desarrollar

16 https://www.w3.org/TR/vocab-ssn
17 http://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/

herramientas para ayudar en el modelado de ontologías, pudiéndose destacar las siguientes como las más relevantes. Protege [234], desarrollada en Stanford, se ha convertido en una de las más populares ya que es compatible con OKBC (Open Knowledge Base Connectivity [235]) y con los estándares W3C - OWL 2 Web Ontology Language y con las especificaciones RFD del consorcio WWW. WebOnto [236], Ontosaurus [237], WebODE [238] u Ontolingua Server [239], son otros ejemplos (una comparación interesante se realiza por Su et al. [240] y por Kapoor et al. [241]).

Una de las utilidades de las ontologías es la anotación semántica, que consiste en identificar conceptos de un documento de forma que, para entender su significado, se relacionan con ontologías ya existentes [242]. De esta manera, se consigue saber a qué concepto, que ya ha sido etiquetado y analizado, hace referencia el texto. En términos generales, estos sistemas extraen nombres/pronombres (entidades con nombre/términos relevantes) de los textos, consultando diferentes bases de datos con la idea de conocer otros términos que estén relacionados con los extraídos, pudiendo unir conceptos del texto original [243]. De esta forma, un documento se puede representar como un espacio finito de conceptos unidos con cierta ponderación dependiendo de su posible significado dentro del texto.

En los últimos años, los trabajos relacionados con la anotación semántica se han focalizado en dos campos: web e IoT, siendo la primera la que que se lleve un gran porcentaje de los mismos. Maio et al. presentan FBASAM [244], un método para crear anotaciones semánticas automáticas de documentos web, de forma que se obtiene una representación abstracta de los conceptos y sus relaciones. Está diseñado para obtener anotaciones relacionadas con las ontologías que se proporcionen en ese momento. Charton et al. proponen otro método automático para identificar entidades llamado ASAM4NE [245]. Este método se basa en comparar las palabras que aparecen antes y después de determinadas entidades con el contenido de artículos de la Wikipedia, identificando aquellas entidades que son más relevantes atendiendo a la similitud entre éstas y su aparición en los artículos. Diallio et al. proponen un sistema basado en ontologías (OBSAA [246]) para automatizar la anotación usando tecnologías de procesamiento de lenguaje natural. En este caso, los conceptos están relacionados con ontologías biomédicas. Una revisión de anotaciones web se puede ver en el trabajo de Rong y sus colegas, donde además se proponen varios métodos para la creación de anotaciones [247].

En el caso de la anotación relacionada con el Internet de las Cosas, mucho del trabajo está focalizado en la anotación de datos generados por redes

de sensores (SSN). Barnaghi et al. proponen un modelo semántico (SM2SS) para describir flujos de datos cuyo origen son los sensores, publicarlos, descubrirlos y organizarlos [248]. Otro trabajo, el desarrollado por Kolozali et al., presenta KBA4IoTDS, un sistema de anotación en tiempo real de flujos de datos, para poder realizar entregas de grandes conjuntos de datos usando AMPQ (Message Queueing Protocol) [249]. La tesis de fin de máster de Chenyi [250] propone SOESAF, una solución orientada a servicios que, de forma manual, anota la función, el estado e información básica. En su tesis, Bing propone SAM4IoTD, donde las anotaciones se hacen sobre documentos IoT, añadiendo información a los archivos compartidos [251]. En el trabajo Mayer et al. [252] se desarrolla Open Semantic Framework para aplicaciones IoT industriales. Este trabajo propone una solución para poder desplegar aplicaciones semánticas en un entorno industrial, permitiendo la comunicación entre la maquinaria y el IoT.

TAGGING Añadir palabras clave a un contenido es lo que se denomina "tagging", o etiquetado. Cada tag está compuesto por una o varias palabras que ayudan a comprender el concepto que se está etiquetando: son datos sobre los propios datos (metadatos). En el desarrollo de software es muy frecuente encontrarnos con contenidos etiquetados que permiten una comunicación entre las personas que desarrollan el producto y los usuarios de este que, por lo general, no son expertos [253]. Estos tags resultan útiles para organizar información y clasificar contenidos, por ejemplo, en páginas web. Los usuarios etiquetan sus posts eligiendo los términos que consideren. Sin embargo, ante el poco o bajo control que se realiza sobre esta etiquetación, podemos encontrar dos fenómenos: sinonimia de tags [254] y la explosión de tags [255]. El primer fenómeno describe cómo podemos encontrar diferentes tags que, semánticamente, significan los mismo, pero que clasifican los contenidos en categorías diferentes, ya que no se realiza un filtrado de los mismos (e.g.: el uso de mayúsculas y minúsculas) ni se establecen relaciones entre conceptos. Esto lleva a que los contenidos que se etiquetan manualmente generan una gran cantidad de tags innecesarios que, además, se pueden agrupar.

Con la idea de aplacar estos problemas, se han desarrollado diferentes técnicas de automatización de etiquetado que se pueden, a grandes rasgos, clasificar en dos categorías: centradas en el contenido y centradas en el filtrado. Las primeras (como FastTagRec [256] o TagCNN [257]) emplean aprendizaje máquina para construir clasificadores (utilizando como datos de entrada los contenidos que ya existen), los cuales utilizan para etiquetar los contenidos nuevos. En cambio, los segundos (por ejemplo, TagMulRec [258]), se basan en analizar semánticamente los contenidos que ya existen con los que se quieren etiquetar, de forma que, en función de su similitud,

los nuevos acaban etiquetados. Un trabajo interesante que combina estas dos soluciones es el desarrollado por Li et al. [259], donde se presenta un clasificador (TagDC) para recomendar etiquetas de contenido, siendo estas recomendaciones extraídas por el análisis del contexto de contenido ya existente. En el trabajo presentado por Zhou et al. [257] se hace una comparativa en profundidad entre diferentes métodos, presentando cuatro posibles alternativas, basadas en deep learning: TagCNN, TagRNN, TagHAN y TagRCNN, relacionadas, respectivamente, con Redes Neuronales Convolucionales (CNN), Redes Neuronales Recurrentes (RNN), Redes de Atención Jerarquizada (HAN), y una combinación de las dos primeras, RCNN.

2.6.2 *El humano: procesamiento del lenguaje natural*

El término *Procesamiento del Lenguaje Natural*, en inglés Natural Language Processing (NLP), hace alusión a la tarea de manipular el lenguaje natural humano a través de un sofware. Esto significa que, dado un texto, una máquina sea capaz de entender su contenido, extrayendo las relaciones entre las palabras, la importancia de éstas basándose en reglas gramaticales, siendo capaces de interpretarlo, y responder del mismo modo que lo haría otro humano. Es decir, dar la capacidad a una máquina de entender un conjunto de palabras del mismo modo que el ser humano. Algunos casos de uso, entre otros, son:

- **Detección de Spam.** Una aplicación interesante es la detección de emails fraudulentos, incluso de phishing, a través del análisis de texto. Es habitual encontrar erratas en el nombre de supuestas compañías, una gramática pobre, lenguaje amenazante o urgencia, en correos fraudulentos. Con el NLP se puede automatizar la detección de estos mensajes.

- **Traducción máquina.** Realizar una traducción eficiente implica capturar de la forma más exacta el significado y el tono de un idioma para poder transmitir ambos sentimientos en el idioma destino.

- **Asistentes virtuales y chatbots.** Estamos bastante habituados a hablar con Siri, Alexa, Cortana u otros asistentes, así como de realizar preguntas a través de una ventana de chat a una compañía. Al otro lado existe una máquina que, en primera instancia, procura responder a nuestras peticiones, pero que, en caso de ser necesario, puede redirigirnos a un humano o hacernos preguntas para poder responder de una forma más precisa.

- **Análisis de sentimientos en redes sociales.** Una aplicación del NLP es percibir qué sensaciones se relacionan con según qué producto

con la idea de dar respuestas a posible inquietudes o, incluso, crear campañas de publicidad enfocadas a mitigar, por ejemplo, una mala apreciación o acogida de dicho producto.

- **Resumen de contenido.** Crear resúmenes de conjuntos grandes de datos, o de información, ayuda a su clasificación, así como a crear bases de datos.

Existen varias tareas que, juntas, permiten a una máquina entender un mensaje expresado de forma humana. Éstas son:

- **Reconocimiento de discurso, Speech Recognition.** Convierte datos expresados, en general, a través de la voz, en texto. Sin embargo, es necesario aclarar que no solo se puede considerar la voz como fuente de información, sino también cualquier otro tipo de origen como puede ser el movimiento de ojos o la lengua de signos. En cualquier caso, la información acaba convertida en texto.
- **Etiquetado de términos, Speech Tagging.** Identifica y diferencia, de cada término, de qué tipo es: nombres, verbos, adjetivos, etc.
- **Desambiguación.** Selecciona el significado de cada término dependiendo del contexto.
- **Reconocimiento de entidades, Named Entity Recognition.** Identifica las entidades propias, es decir, los nombres propios: ciudades, nombres de personas, etc.
- **Resolución de referencias cruzadas.** Identificar qué términos hacen referencia a qué parte del texto. Un ejemplo es el uso de pronombres.
- **Análisis de sentimientos.** Extrae características subjetivas del texto: emociones, sarcasmo, confusión, etc.
- **Generación de lenguaje natural.** Es la tarea de colocar cierta información en lenguaje humano. Suele describirse como la tarea contraria al reconocimiento del discurso.

Existen diversos trabajos que analizan cada una de estas tareas. Entre ellos, cabe destacar aquellos que hacen un estudio en conjunto de estas tareas y, dentro de ellas, de los distintos mecanismos que actualmente están en desarrollo. Por ejemplo, el trabajo presentado por Qiu et al. hace un análisis de diferentes modelos preentrenados para NLP incluyendo arquitecturas, tareas, extensiones, y soluciones que, gracias a esos modelos, se pueden llevar a cabo [260]. Otter et al. presenta un estudio sobre trabajos relevantes en el campo del Deep Learning aplicado a NLP, añadiendo recomendaciones para investigaciones futuras [261].

2.7 NOTAS FINALES

El análisis que se ha llevado a cabo del estado del arte se ha dividido en dos partes bien diferenciables: 1) la computación en el borde de la red, haciendo especial hincapié en la descarga computacional y en el papel de los microservicios y los contenedores en el despliegue de entornos distribuidos; y 2) el problema que surge a la hora de orientar en la misma dirección el lenguaje natural utilizado por los usuarios para comunicarse con un entorno, y la forma en la que este se describe. El trabajo desarrollado a lo largo de esta tesis parte de ambos conceptos, e intenta unirlos proponiendo una método para la descripción de entornos distribuidos, y las posibles migraciones en el mismo, de forma que un usuario pueda, de forma activa, originarlas con la menor necesidad de aprendizaje posible sobre los términos que las disparan.

Concretamente, y como se presentará en los siguientes capítulos, se busca mejorar la migración de microservicios (corriendo en contenedores) en el borde de la red donde viven los dispositivos finales (en la Mist), proponiendo un marco de trabajo basado en 1.1) el análisis de los diferentes dispositivos envueltos en ella, así como sus capacidades; y 1.2) el análisis del rendimiento que, presuntamente, otro dispositivo/entorno ofrece. El desarrollo de este primer punto, tras evaluar las diferentes soluciones existentes, propone una manera de realizar el análisis del estado de los dispositivos presentes, mientras que el segundo, propone extender las soluciones actuales con nuevas métricas, realizando tests de rendimiento en aquellos dispositivos/entornos disponibles donde se podría llegar a migrar un microservicio, siendo extensibles, adaptables y complementables las soluciones ya presentes en el estado del arte.

A esto hay que añadir la búsqueda de la mejora en la comunicación entre entorno y humano a través del procesamiento del lenguaje natural, proponiendo 2.1) una manera de describir el entorno y las posibles acciones realizables (concretamente migraciones); y 2.2) una forma de análisis del contenido de las órdenes dadas por el usuario, buscando la mayor compatibilidad e inclusión posible para cualquier tipo de usuario y dispositivo de comunicación. Pensando en ambas tareas a la vez, se propone el uso de varias herramientas NLP para exponer la naturaleza de un entorno y, además, extraer el contenido relevante de una orden, de forma que lo extraído corresponda con alguna de las posibles acciones descritas en ese entorno.

3

ARQUITECTURA DE MICROSERVICIOS: DESCRIPCIÓN

En la sección 2.3 se han presentado las diferencias entre la Fog y la Cloud, identificando las razones por las que surge la Fog, así como su definición. Gracias a la Fog, es posible distribuir servicios en un entorno cercano al dispositivo final (estaciones base, o pequeñas clouds cercanas físicamente), ofreciendo mejores tiempos de respuesta que en un despliegue en una Cloud centralizada. Así mismo, en la sección 2.3.2 se ha presentado el concepto de Mist Computing, un paradigma computacional que propone acercar los servicios todavía más a los dispositivos finales, desplegándolos en los puntos de acceso a la red o en los propios dispositivos finales.

El primero de los objetivos presentados en el capítulo 1, sección 1.2, es el de contribuir a la orquestación de microservicios en un entorno distribuido. Esta tesis focaliza su atención en los entornos Mist, descritos en la sección 2.3.2. Este objetivo se ha desglosado en tres subobjetivos:

- O1.1: Proporcionar un mecanismo para la descripción de microservicios y de los dispositivos que coexisten en un entorno. En particular, el entorno será un entorno Mist. En el presente capítulo, sección 3.1.1, se describen la metodología a seguir para poder generar estas descripciones, tanto de microservicios como de dispositivos, así como las entidades diseñadas que están presentes en el entorno, las cuales son las encargadas de recopilar estas descripciones.

- O1.2: Proporcionar una metodología para el análisis de compatibilidades entre las descripciones de los dispositivos y las de los microservicios. En la sección 3.1.2 del presente capítulo se describen los requisitos que se deben cumplir, así como las funcionalidades que se deben ofrecer, por parte de los fabricantes como de los desarrolladores.

- O1.3: Diseñar un mecanismo que permita la evaluación del rendimiento de los diferentes microservicios en diferentes dispositivos antes de ser desplegados. En la seccion 3.2 del presente capítulo se describe el diseño de este mecanismo, así como los de todos los elementos que lo constituyen.

3.1 ARQUITECTURA DE MICROSERVICIOS

Típicamente, un servicio desplegado bajo una arquitectura de microservicios es el resultado de coordinar el uso de un conjunto concreto de microservicios desplegados en un entorno. Cada uno de estos microservicios, puede ser instanciado en un dispositivo diferente. Para que un microservicio pueda ser alojado, sus necesidades deben ser compatibles con las capacidades del dispositivo en el que se quiere instanciar. Por ejemplo, un microservicio que, para funcionar correctamente, necesite el uso de cierto periférico, necesitará poder disponer de él en el dispositivo que lo hospede.

Para saber cuáles son estas necesidades, un microservicio debe ser capaz de proporcionar una descripción de las mismas que pueda ser entendida por cualquier dispositivo. Los dispositivos, a su vez, deben poder facilitar una descripción de sus características o capacidades, de manera que, al comparar una descripción de un microservicio con la de un dispositivo se pueda concluir si el segundo puede albergar al primero con una calidad dada.

A la hora de hacer estas descripciones, existe una gran cantidad de notaciones relacionadas con expresiones de taxonomías en formatos como XML, JSON, RDF, RDFS, OWL, SPARQL, Turtle o N-Triples, sin embargo, esta tesis no está centrada en las descripciones en sí, sino que plantea cómo se intercambian éstas y en su comparación.

El tipo de entorno en el que está focalizado el presente trabajo está compuesto por un número indeterminado de dispositivos finales, los cuáles hospedan diferentes microservicios. Combinando estos microservicios se crean diferentes servicios complejos. Para que se puedan conocer cuáles son las necesidades de los microservicios, así como las capacidades que ofrece cada uno de los dispositivos finales, es necesario que cada uno de los microservicios y cada uno de los dispositivos tengan asociado una descripción de estas necesidades y capacidades (ver figura 7).

3.1.1 *Entidades presentes*

Con la idea de recopilar la información que describe a los dispositivos y los microservicios, se presentan dos aplicaciones que serán instanciadas como microservicios. Éstas serán las encargadas de recoger las descripciones tanto de dispositivos como de microservicios: el *Node Summarizer* y el *MicroService Summarizer*.

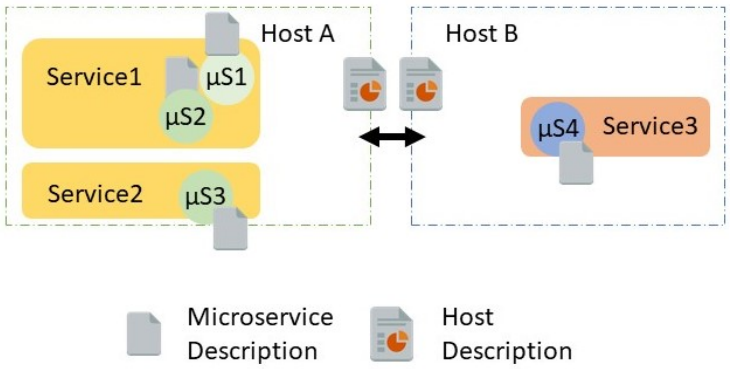


Figura 7: Ejemplo básico: 2 hosts, 4 microservicios y 3 servicios

El Node Summarizer (NS) tendrá acceso a la información que describe el estado del dispositivo. Estos datos hacen referencia a los periféricos disponibles y sus especificaciones, así como al uso de los recursos del dispositivo, e.g., uso de CPU, memoria RAM, velocidad o número de núcleos. Las descripciones de los microservicios que se alojan en un dispositivo son recopiladas por el MicroService Summarizer (µSS). Dicha descripción recoge qué recursos necesita un microservicio para funcionar: desde periféricos, hasta los recursos mínimos para dar cierta calidad de servicio. Ambos microservicios conviven en los dispositivos.

La manera en la que llevan a cabo su tarea sigue el siguiente comportamiento:

- El NS realiza consultas periódicas al sistema operativo actualizando cada cierto tiempo la información guardada.
- El µSS puede comportarse de dos maneras diferentes:
    1. Modo pasivo. El µSS espera actualizaciones de los microservicios con los que convive: cada cierto tiempo los microservicios envían su descripción, actualizándola si es necesario. Aunque la descripción del microservicio puede no cambiar con frecuencia, estas actualizaciones periódicas servirán también para verificar que los microservicios siguen disponibles, haciendo las veces de heartbeat.
    2. Modo activo. Es el µSS, solicita periódicamente al microservicio información actualizada sobre el servicio prestado que, de nuevo, aunque esta información no vaya a cambiar con frecuencia, es necesario comprobar si el microservicio sigue disponible.

El origen de ambos microservicios puede ser privado o público. Es decir, su codificación proviene del mismo desarrollador del dispositivo y no

se puede redefinir por terceras partes (privado), o bien puede ser fruto del desarrollo cooperativo de diferentes individuos (público) y se puede consultar e instanciar a través de, por ejemplo, un repositorio de dominio público. En el caso del NS, resulta más cómodo que el proveedor del dispositivo añada este microservicio por defecto (o indique el modo en que se puede instanciar), ya que se trata de la recopilación de datos que pueden ser más o menos sensibles. Además, el proveedor conoce perfectamente el dispositivo, de forma que sabe qué información puede revelar y qué registros consultar para obtenerla.

Sin embargo, en el caso del μSS, contar con un repositorio público de donde sacar las necesidades de los servicios a desplegar, debería ser la estructura más conveniente. De esta manera, si las entidades que hacen uso de un microservicio lo desean, pueden añadir informes sobre el rendimiento de los microservicios que ya han sido probados, de forma que las descripciones originales pueden ser modificadas por los desarrolladores. Así, cada microservicio tendrá asociado un perfil con datos que indiquen cómo rinden en según qué circunstancias, ya sea porque ha sido proporcionado por los desarrolladores o bien porque ha sido obtenido de la experiencia.

En entornos de computación distribuida, es habitual que esté presente alguna entidad que monitorice, orqueste y planifique, tanto el estado de los dispositivos involucrados, como las tareas que se llevan a cabo, atendiendo, entre otros aspectos, al balanceo de carga o a proporcionar una calidad de servicio. Con la orquestación, los microservicios se despliegan, reconfiguran, actualizan y se administran con una alteración mínima de los servicios prestados. Por ello, en entornos donde suceden una gran cantidad de cambios, ya sea en la configuración o en el número de dispositivos que coexisten, como en el IoT, resulta útil contar con una entidad que se encargue de esas tareas de coordinación. Esta entidad sería un orquestador como los descritos en la sección 2.5.3.2 del estado del arte.

La propuesta de esta sección del presente capítulo es que las tareas del orquestador se puedan apoyar en el NS y μSS, haciendo posible que el orquestador se subscriba a estos dos microservicios. De esta manera, usando la información que le suministran, el orquestador puede tomar decisiones de coordinación. De esta forma, la complejidad del orquestador se simplifica dado que ya no es él el elemento que debe calcular estas medidas sino que, simplemente, las consulta.

En la figura 8 se muestra la integración de estos tres elementos, el NS, el μSS y el orquestador, al entorno mostrado en la figura 7 donde se po-

día ver cómo estaban constituidos los diferentes servicios. En la figura 8 se puede observar como los NS y µSS de los diferentes dispositivos están unidos al orquestador. Estas uniones ilustran el hecho de que el orquestador está subscrito a los NS y µSS de los dispositivos: el orquestador tiene acceso a las descripciones de los diferentes dispositivos y microservicios gracias a haber desplegado los microservicios NS y µSS. Para poder subscribirse a estos dos microservicios, éstos deben estar escuchando peticiones en sendos puertos bien conocidos. De esta manera, cuando los dispositivos pasan a formar parte del entorno, el orquestador sabe a dónde dirigir sus consultas.

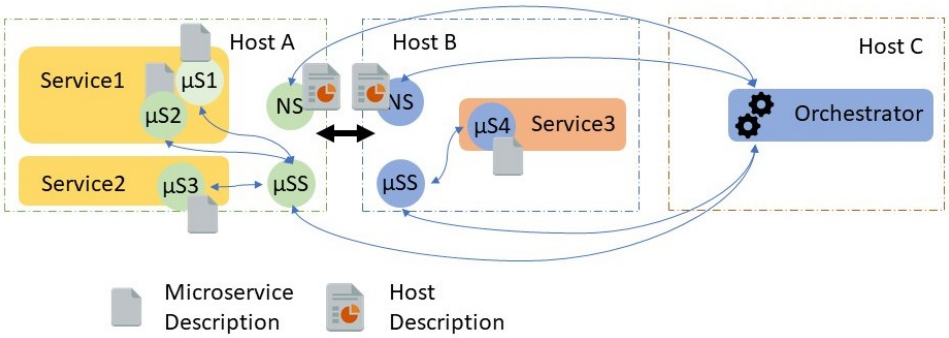


Figura 8: Extensión del ejemplo básico con un orquestador

Los dos microservicios, el NS y el µSS presentados pretenden facilitar las tareas de orquestación en un entorno basado en microservicios. A continuación, se describen tres de estas tareas y cómo se pueden ver beneficiadas por la presencia de estas dos entidades: el descubrimiento de microservicios, el conocimiento del estado de los dispositivos y la migración de microservicios.

3.1.1.1 *Descubrimiento de microservicios*

En el proceso de descubrimiento de un microservicio, un cliente puede realizar una búsqueda especificando el nombre del microservicio o alguno de los atributos del microservicio que busca. Existen diferentes protocolos de descubrimiento que se basan en plantillas predefinidas que definen el formato de los nombres de los microservicios, por ejemplo, usando multicast DNS [262] [263]. Además, algunos protocolos ofrecen la posibilidad de usar un conjunto predefinido de atributos, como en el caso de Bluetooth DSP [264].

Los clientes, los microservicios y los orquestadores tienen dos opciones a la hora de intercambiar datos sobre su presencia y el registro a los microservicios: anuncio o consulta [265]. El anuncio consiste en darse a co-

nocer, mientras que la consulta consiste en preguntar qué microservicios están presentes. En ambas situaciones, la localización de los microservicios responsables debe ser conocida mediante el uso de puertos conocidos, mecanismos de descubrimiento de servicios tradicionales, registros DNS específicos o Zeroconf.

A la hora de descubrir, anunciar o consultar la existencia de un microservicio, es necesario poder limitar el alcance de estas acciones, de forma que se minimice la computación innecesaria en clientes y microservicios que no deban atender a estas peticiones y anuncios. Las limitaciones basadas en la topología de la red, como las llamadas de un salto, single-hop, se basan en la suposición de que todos los elementos que forman parte del entorno están emplazados en el mismo dominio. Esta asunción puede ser correcta para entornos pequeños, pero no necesariamente se pueda aplicar a entornos más generales donde coexistan diferentes dominios y redes. Otro tipo de limitación es la basada en roles, de forma que los clientes se autentican en un dominio y tienen acceso a todos los servicios del mismo, independientemente de la distancia a la que se encuentren. Un tercer tipo de limitación es la basada en contexto, de forma que solo se comuniquen aquellos clientes y servicios que se encuentren dentro del mismo contexto, como por ejemplo una localización física concreta.

Para solucionar estos inconvenientes, el µSS puede resultar útil dado que su papel es el de comunicarse con el orquestador y los microservicios. Puede ser configurado para que responda en nombre del microservicio a las peticiones de descubrimiento o consulta. Así, si se quiere limitar el descubrimiento de un microservicio, se puede añadir una regla al µSS que indique que no debe contestar a según qué peticiones, ya sea filtrándolas por perfil, dominio o número de saltos en la red.

3.1.1.2 *Estado de los dispositivos*

En el Estado del Arte se han presentado algunos orquestadores de contenedores (sección 2.5.3.2). Un orquestador se basa en las métricas tomadas de los dispositivos para saber cuál tiene más o menos carga de trabajo y, en función de ello, decidir qué dispositivo aloja qué contenedor. Estas métricas suelen ser tres: el uso de la CPU, la carga de la red y el consumo de memoria [266]. Existen dos enfoques desde los que se toman estas medidas dentro del despliegue de contenedores: 1) monitorizar los dispositivos y 2) monitorizar las plataformas de despliegue de contenedores.

En el primer caso, el orquestador se encarga de desplegar un agente en el dispositivo para que tome esas medidas. Para poder hacerlo, se debe conocer con exactitud el dispositivo en el que se despliega. Es decir, debe saberse qué registros hace falta consultar para obtener la métrica deseada. Sin embargo, dada la heterogeneidad de dispositivos, es posible que la misma métrica no se consulte en el mismo registro para diferentes dispositivos. Para solventar esta situación, los entornos suelen ser bien conocidos, por ejemplo, teniendo dispositivos del mismo fabricante o siendo, los dispositivos y el orquestador, desarrollados por al misma entidad.

En el segundo caso, del mismo modo que en el primero, el orquestador despliega un agente para que tome esas medidas pero, en este caso, con la intención de tomar las medidas de las plataformas de despliegue de contenedores. Para que esto sea posible, es necesario contar con la misma plataforma en todos los dispositivos, no permitiendo el despliegue de varias plataformas simultáneamente incurriendo en la necesidad de añadir una entidad que las coordine.

La razón de ser del Node Summarizer es la de servir de punto de acceso a la información de cualquier dispositivo. Al comienzo de la presente sección (3.1.1), se ha descrito el comportamiento del Node Summarizer así como el origen del mismo. En particular, se ha comentado que su origen debe ser el propio fabricante del dispositivo. Es decir, el fabricante del dispositivo debe añadir este microservicio, el Node Summarizer, a sus dispositivos. Este microservicio monitorizará al dispositivo pudiéndose consultar sus métricas en un puerto bien conocido. De esta manera, el orquestador presente en el entorno puede subscribirse al Node Summarizer, siendo ventajoso para el desarrollo de los orquestadores: éstos no necesitan desplegar un sistema de monitorización en cada dispositivo ya que cada uno tiene su propio microservicio encargado de esta tarea, el Node Summarizer.

3.1.1.3 *Migración de microservicios*

Cuando se requiere migrar un microservicio de un dispositivo a otro, es necesario conocer las necesidades del microservicio y las características de los dispositivos disponibles para poder encontrar aquel que mejor satisfaga dichas necesidades. Con los dos microservicios presentados, un orquestador no tiene la necesidad de desplegar ningún tipo de software para el análisis de los dispositivos ni necesita conocer de antemano los servicios que coordina. Cuando un desarrollador implementa un orquestador puede centrarse en las políticas y reglas que va a seguir, así como las acciones que va a llevar a cabo sin preocuparse por el método en el que obtiene las

métricas que necesita para establecerlas porque puede subscribirse tanto al NS como al μSS para obtenerlas.

3.1.2 *Requisitos y funcionalidades*

A lo largo de la presente sección, 3.1, se ha querido establecer un marco de trabajo bajo el cual los fabricantes de dispositivos, así como los desarrolladores de microservicios, lleven a cabo su labor. A continuación, se recogen los requisitos principales que deben cumplir tanto fabricantes como desarrolladores, así como una serie de funcionalidades que deben ofrecer los microservicios descritos:

- Los fabricantes de los dispositivos deben proporcionar tanto al Node Summarizer como al microService Summarizer.
- Ambos microservicios deben dar servicio en puertos bien conocidos. En particular:
  - En el caso del Node Summarizer habrá un único puerto a través del cual un orquestador puede realizar consultas sobre el estado del dispositivo.
  - En el caso del microService Summarizer habrá dos puertos. Uno se utilizará para que los microservicies puedan anunciar su existencia. El otro se usará para que el orquestador pueda realizar consultas sobre el estado de los microservicios que se han anunciado.
- La información que ambos microservicios recopilan se debe poder consultar a través de una API.
- En el caso del Node Summarizer, dicha información debe incluir información relativa a los componentes Hardware (periféricos, CPU, memoria y red), así como su estado (ocupación y disponibilidad).
- En el caso del microService Summarizer, dicha información debe incluir información relativa a las necesidades de los microservicios (e.g., la necesidad de contar con ciertos periféricos).
- Ambos microservicios se deben poder instanciar en diferentes plataformas de despliegue de contenedores (ver la sección 2.5.3.2 del estado del arte).
- Los fabricantes deben encargarse de la actualización y mantenimiento del microservicio.

- Los desarrolladores de los microservicios deben desarrollar sus microservicios haciendo que éstos anuncien sus necesidades al microService Summarizer.
- La información relativa a estas necesidades debe especificar el tipo de periféricos que necesita el microservicio para funcionar correctamente.

3.2 ANÁLISIS DE RENDIMIENTO

El primero de los objetivos presentados en la sección 1.2, contribuir a la orquestación de microservicios en un entorno distribuido, se desglosa en tres subobjetivos. Las sección anterior (sección 3.1) alude a los dos primeros, mientras que la presente sección (3.2) busca satisfacer el tercero: diseñar un mecanismo que permita la evaluación del rendimiento de los diferentes microservicios en diferentes dispositivos antes de ser desplegados.

Como se ha visto en el estado del arte, sección 2.5.4, existen muchas estrategias a la hora de analizar y evaluar el rendimiento de un arquitectura de microservicios. Entre ellas, este trabajo se ha centrado en la idea tras el concepto de "mocking". En esta sección se presenta un marco de trabajo para posibilitar el análisis de rendimiento a través de una metodología basada en mocking, gracias al cual se pueden obtener más datos en los que basar una toma de decisiones a la hora de instanciar un microservicio en un dispositivo.

Intentar predecir el rendimiento de una pieza de software en un entorno antes de desplegarlo, y durante su ciclo de vida, es una tarea complicada y que ha sido atajada por numerosos trabajos que proponen diferentes modelos de comportamiento en los que basarse para poder realizar balanceo de carga (sección 2.4 del estado del arte). Sin embargo, diversos trabajos defienden que el procesado de estos modelos no se deba aplicar en entornos en los que los dispositivos sean heterogéneos, limitados en capacidades como memoria o procesado y, además, variando su número con frecuencia [29] [30]. Estas soluciones no son óptimas si tomamos en consideración el tiempo que lleva hacer estos análisis: si el entorno cambia de forma frecuente es posible que las estimaciones dejen de valer antes de ser procesadas.

Con la idea de seleccionar posibles destinos para los microservicios que satisfagan sus requisitos, se puede considerar realizar tests de rendimiento en ellos. De esta manera, antes de instanciar un microservicio en un dispositivo de forma definitiva, se podría realizar un despliegue momentáneo

en él durante el cual llevar a cabo los tests y, en función de los resultados durante ese periodo, decidir si desplegarlo o no en ese dispositivo. Si esto se realizara en paralelo en varios dispositivos, se podría llevar a cabo una comparativa de los resultados eligiendo como destino final aquél que satisfaga mejor las pruebas realizadas. Dado que las personas que desarrollan los microservicios conocen de su propia mano cuáles son los requisitos que necesitan dichos microservicios, los propios desarrolladores pueden crear los tests específicos que evalúen la respuesta de estos microservicios a diferentes condiciones de contorno.

La propuesta de esta sección consiste en realizar pruebas de corta duración del microservicio que se quiera instanciar en varios dispositivos candidatos a la vez. De esta manera, ayudándose de los datos obtenidos por las pruebas, el orquestador del entorno puede tomar sus decisiones de despliegue considerando esta información extra.

No obstante, es necesario tener en cuenta que los microservicios pueden depender de otros para su funcionamiento. Esto implica que, cuando se quiera desplegar el entorno de análisis, para llevar a cabo las pruebas, sea necesario que el microservicio disponga de unos datos de entrada. Existen dos posibilidades para facilitar estos datos: obtenerlos de otro microservicio o proporcionar datos no reales. La primera de las opciones podría implicar interrumpir el funcionamiento normal del microservicio del que depende durante el periodo de pruebas, perturbando el funcionamiento normal del dispositivo candidato. Sin embargo, la segunda opción satisface la necesidad de crear un entorno controlado de pruebas, aislándolo del resto de microservicios que ya conviven en el dispositivo. Los datos son proporcionados por el entorno de pruebas, con la idea de que en todos los dispositivos donde se esté llevando a cabo la misma prueba se utilicen los mismos datos.

El entorno de pruebas propuesto se basa en la existencia de dos componentes extra para cada microservicio: 1) una versión reducida del microservicio, y 2) un conjunto de tests que se puedan aplicar a los microservicios. Estos componentes los debe proporcionar la entidad que desarrolla el microservicio, preferiblemente a través de un repositorio, siendo este accesible para el orquestador.

Por un lado, las versiones reducidas de un microservicio simulan su comportamiento. Una versión reducida de un microservicio es lo que se denomina “mock”. En el estado del arte se han mencionado algunas herramientas habituales en la evaluación de unidades (sección 2.5.4). Su misión

es la de proporcionar un conjunto limitado de datos del mismo modo que lo haría la versión completa del microservicio: usando el mismo formato, siendo del mismo tamaño, etc. Sin embargo, estos datos no son obtenidos en el momento sino que han sido seleccionados por los desarrolladores. Tomando como ejemplo el caso de un microservicio que tome imágenes a través de una webcam, su versión mock consistirá en devolver siempre la misma imagen o conjunto de imágenes. De esta manera, aquellos microservicios que requieran imágenes tomadas por una webcam recibirán estas imágenes predefinidas.

Por otro lado, los tests serán pequeños scripts que se ejecutarán como un microservicio. Los tests podrán a prueba el microservicio desplegado para su análisis, recopilando datos y realizando cálculos que se enviarán al orquestador para los tome en consideración a la hora de decidir dónde instanciar el microservicio.

3.2.1 *Componentes y consideraciones de diseño*

La figura 9 muestra las relaciones entre los diferentes componentes del entorno de pruebas propuesto. A continuación se identifican cada uno de estos componentes:

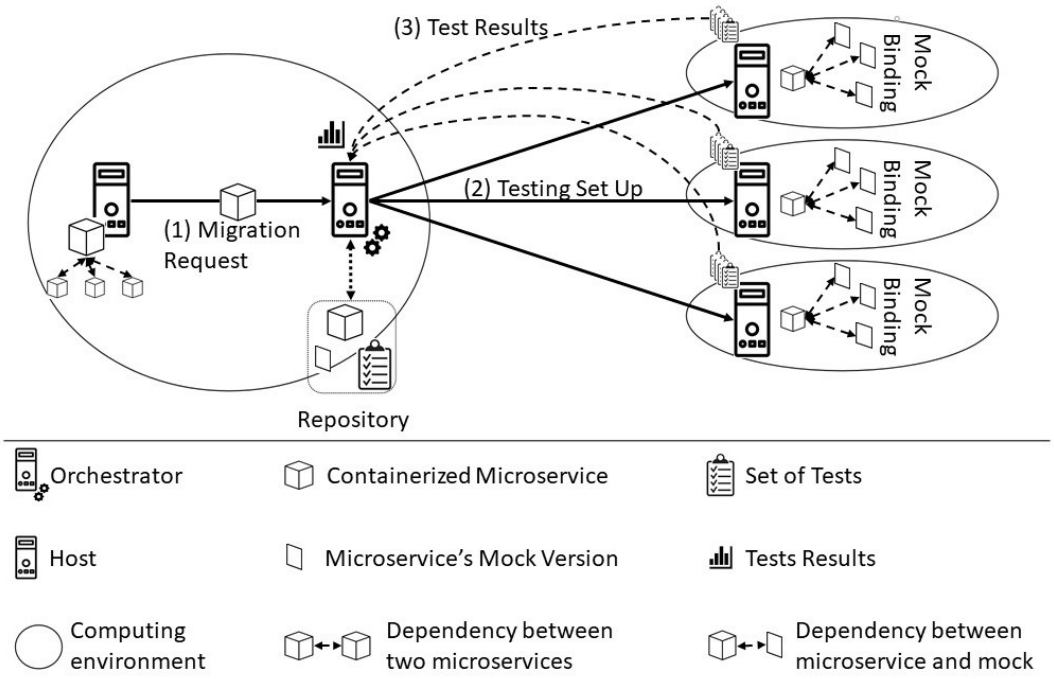


Figura 9: Infraestructura de evaluación de rendimiento.

- **Orquestador.** Como se ha visto en la sección anterior, sección 3.1, en entornos distribuidos donde coexisten diferentes microservicios es habitual la existencia de un orquestador, encargado de tomar decisiones de distribución y migración de los microservicios entre dispositivos. El orquestador será el encargado de establecer si un microservicio debe migrar de un dispositivo origen a otro destino y, además, de establecer cuáles son los dispositivos destino candidatos.

- **Repositorio software.** En estos repositorios se podrán consultar las versiones mock de los microservicios, así como el conjunto de pruebas que se les puede practicar.

- **Versión mock de cada microservicio.** La versión mock de un microservicio simula su comportamiento. Estas versiones se instanciarán en los dispositivos candidatos para hospedar al microservicio que migra. Las versiones reducidas que se despliegan son las de los microservicios de los que depende el microservicio que migra.

- **Dispositivos destino candidatos.** Los dispositivos destino candidatos son aquellos dispositivos que, a priori, son compatibles con el microservicio que migra. Es decir, son capaces de albergarlo, pero no necesariamente ofreciendo el mejor rendimiento.

- **Conjunto de pruebas.** Cada microservicio debe tener asociado, además de su versión mock, un conjunto de pruebas a las que se puede someter el microservicio en cuestión.

Seguidamente, se identifican diferentes consideraciones de diseño para poder desplegar el entorno de pruebas propuesto:

- La manera en la que se establecen los dispositivos candidatos en este trabajo responde al uso de las entidades presentadas en la sección 3.1.1.

- Las versiones mock de los microservicios deben interferir de forma mínima en el comportamiento normal del dispositivo destino. De esta manera, si el microservicio A, que se quiere reubicar, depende de B y C, se instanciarán, en cada dispositivo destino candidato, el microservicio A, en su versión completa, y las versiones reducidas de B y C.

- La forma en la que se establece la relación entre el microservicio que migra y las versiones mock debe ser ligera, i.e., no debe sobrecargar al dispositivo candidato en términos de uso de red, memoria y uso de la CPU.

- Los conjuntos de pruebas de los microservicios deben llevarse a cabo en un tiempo mínimo, con la idea de obtener los datos de rendimiento. Cada conjunto de pruebas debe tener definido:
  - Qué características se quieren medir.
  - Qué valores pueden tomar estas características.
  - Qué rango de valores es aceptable y cuál no.

Teniendo en cuenta estas consideraciones, el diseño del trabajo desarrollado en esta tesis se ha basado en el uso de dos componentes: MockServer[1], y el protocolo MQTT[2].

MockServer es un servidor que responde a peticiones HTTP preconfiguradas llamadas *expectations* (en adelante expectativas). Si una petición encaja con una expectativa, el servidor responde con la respuesta que se haya configurado para esa expectativa en particular. Si no encaja con ninguna expectativa se devuelve un código de estado 404. Las versiones mock de los microservicios serán expectativa en el MockServer. Así, cuando un microservicio requiera unos datos de otro del que depende pedirá estos datos a la expectativa.

MQTT (MQ Telemetry Transport) es un protocolo de paso de mensajes. Existen dos tipos de entidades: un broker y un número indeterminado de clientes. El broker es el servidor que recibe todos los mensajes de los clientes, enrutando diferentes mensajes a diferentes clientes. La información que se maneja está organizada jerárquicamente por temas. Cuando un cliente quiere publicar cierta información de forma que se distribuya, éste envía un mensaje al broker con la información y, a continuación, el broker lo reenvía a aquellos clientes que estén subscritos a ese tema. El cliente que publica esa información no tiene en cuenta cuántos otros clientes están subscritos ni su localización, del mismo modo que aquellos clientes que se subscriben no tienen que saber nada sobre el cliente que publica. Se trata de un protocolo utilizado para comunicaciones machine-to-machine (M2M) y una recomendación ISO (ISO/IEC 20922:2016[3]). El protocolo MQTT se utilizará para poder transmitir los resultados de los tests. De esta manera, el orquestador albergará al broker mientras que el intercambio de datos sobre los resultados se realizará a través de publicaciones.

Este trabajo se ha decantado por el uso de estas dos tecnologías para cumplir con las consideraciones previas debido a que:

1 https://www.mock-server.com/
2 https://mqtt.org/
3 https://www.iso.org/obp/ui/#iso:std:iso-iec:20922:ed-1:v1:en

- las expectativas de MockServer están delimitadas a su espacio de memoria de forma que no se solapa con la memoria utilizada por otras aplicaciones.
- las expectativas no utilizan los periféricos de los dispositivos candidatos ya que responden con datos precargados.
- el tiempo para la configuración de las expectativas es mínimo.
- las dependencias entre el microservicio que migra y las versiones mock de los demás microservicios, las expectativas, se basa en peticiones HTTP.
- el protocolo MQTT pueden ser utilizado por pequeños dispositivos hasta grandes servidores dedicados, satisfaciendo las diferentes necesidades de los dispositivos del entorno que, en general, no serán homogéneos.

3.2.2 *Plan de pruebas*

Con la idea de validar la presente propuesta, se ha definido el siguiente plan de pruebas compuesto por los siguientes elementos y configuraciones:

- Un entorno distribuido. El escenario de pruebas debe estar compuesto por dispositivos finales cercanos. En el capítulo siguiente, 4, se describen estos dispositivos y sus características.
- Cada dispositivo debe contar con un motor de despliegue de contenedores. Como se ha visto en el estado del arte, sección 2.5, los contenedores proporcionan un entorno flexible, siendo habitual que alojen un único servicio, resultando ser una herramienta muy atractiva para el despliegue de arquitecturas de microservicios. En concreto se hará uso de la plataforma Docker que, como se ha visto en el estado del arte, es una herramienta muy popular en los entornos distribuidos.
- Los microservicios que cohabiten en el entorno, así como los microservicios presentados en 3.1.1, estarán instanciados en contenedores. En concreto, cada dispositivo del entorno contará con un microservicio Node Summarizer y un microservicio microService Summarizer, además de los microservicios que coexistan en el entorno. En el capítulo siguiente, capítulo 4, se describe en detalle qué microservicios conviven en el entorno.
- Debe existir un orquestador. Como se ha descrito en las secciones previas, el papel del orquestador es el coordinar, entre otros aspectos, la migración de microservicios, tarea en la que se focaliza este trabajo.

- Se hará uso de MockServer para crear las versiones reducidas de los microservicios, esto es, generar las expectativas que simulen el comportamiento de los microservicios. Cada dispositivo tendrá un instancia MockServer.
- Se hará uso de pruebas de unidad. Las pruebas que se definen para cada uno de los microservicios serán de tipo "unittest", de forma que focalice en el comportamiento correcto del microservicio. En concreto, las pruebas se focalizarán en el tiempo de respuesta de los microservicios, es decir, cuánto tiempo tardan en realizar la tarea para la que han sido diseñados estos microservicios. Estas métricas serán enviadas al orquestador a través del protocolo MQTT.
- Además del tiempo de respuesta de los microservicios, se quiere tomar medidas sobre el tiempo invertido en la configuración de las diferentes instancias MockServer.

3.3 CONCLUSIONES

A lo largo de este capítulo se han realizado diferentes propuestas. En primer lugar, se han presentado dos entidades, el Node Summarizer y el microService Summarizer (sección 3.1.1). Éstas surgen de la necesidad de simplificar el diseño de orquestadores dentro de entornos distribuidos. Por un lado, el Node Summarizer evita añadir al diseño de los orquestadores la necesidad de incluir un mecanismo para la monitorización del entorno. Dado que cada dispositivo del entorno cuenta con un Node Summarizer, será el propio dispositivo, a través de este microservicio, el que facilite sus métricas de estado (rendimiento, memoria, conectividad y estado de los periféricos, entre otros datos). Por otro lado, el microService Summarizer simplifica el diseño de los orquestadores haciendo que estos no requieran implementar un protocolo complejo de descubrimiento de microservicios ya que, gracias al uso de puertos bien conocidos, saben a qué aplicación preguntar sobre el estado de los microservicios que corren en cada uno de los dispositivos.

Basada en la presencia de estas dos entidades, en la sección 3.2 se ha propuesto un mecanismo que permite obtener más métricas para la toma de decisiones en entornos distribuidos. En concreto, este mecanismo se basa en el despliegue y configuración de entornos de prueba efímeros. Este mecanismo permite a los desarrolladores diseñar y añadir pruebas a los microservicios que desarrollan que permitan obtener métricas menos generalistas: son los propios desarrolladores de microservicios los que diseñan

las pruebas a las que van a someter a los microservicios.

En el siguiente capítulo, capítulo 4 se presenta una prueba de concepto con el fin de validar estas propuestas. En concreto, se establecerá un escenario de pruebas que satisface los requisitos y funcionalidades valorados en las secciones 3.1.2 y 3.2.2. En primer lugar, en la sección 4.1 se describirá la implementación de las entidades presentadas en 3.1.1. Seguidamente, en la sección 4.2, se describe la configuración de los diferentes componentes software nombrados: MockServer, servidor MQTT, la plataforma Docker, las diferentes pruebas de unidad y los diferentes microservicios que se utilizan para validar la propuesta. Por último, en la sección 4.3, se analizan e interpretan las diferentes métricas tomadas.

4

ARQUITECTURA DE MICROSERVICIOS: VALIDACIÓN

En el capítulo 3 se ha propuesto una arquitectura de microservicios, presentando dos entidades que deben estar presentes en los dispositivos que formen parte del entorno en el que se despliegue la arquitectura de microservicios. Así mismo, se ha descrito una metodología para poder desplegar entornos de pruebas efímeros en los que llevar a cabo diferentes pruebas que permitan evaluar el rendimiento de los microservicios. Estas pruebas generan datos que pueden ser usados para la toma decisiones en el despliegue de microservicios.

En el presente capítulo, se desgrana la implementación llevada a cabo basada en el diseño expuesto en el capítulo 3. En concreto, en la sección 4.1, se expone la implementación del diseño de las entidades presentadas en el capítulo 3, sección 3.1.1. En la sección 4.2, se plantea la implementación del diseño del método de testeo de rendimiento introducido en el capítulo 3, sección 3.2. Respecto al plan de de pruebas descrito en la sección 3.2.2 del capítulo 3, en la sección 4.3 del presente capítulo, se recoge su configuración, así como los resultados obtenidos.

4.1 IMPLEMENTACIÓN DEL NS Y DEL μSS

En los sistemas operativos existentes hoy en día, como Windows 10 y 11 o los diferentes sabores de Linux, existen herramientas de serie que proporcionan información sobre los dispositivos que están conectados (o montados) en la máquina, del mismo modo que se puede consultar cuál es el consumo de los diferentes discos duros o de la memoria RAM. En esta prueba de concepto del presente capítulo, nos centraremos en el sistema operativo Windows 10 y en las herramientas que tiene incorporadas.

En el caso de Windows, es el Administrador de Dispositivos el que proporciona esta información permitiendo, como su nombre indica, administrarlos, manipulando los drivers que controlan a esos dispositivos pudiendo pausarlos, desinstalarlos o resetearlos, entre otras operaciones. En el caso de Linux, su manipulación se realiza a través de diferentes comandos propios del sistema como “mount” y existe una gran variedad de herramientas, como “inxi”, que proporcionan la misma información que su ho-

mólogo en Windows.

Es habitual que este tipo de herramientas muestren la información de manera visual, de forma que un usuario humano pueda, de forma amigable, comprobar el comportamiento y el estado de los dispositivos que tiene conectados a su equipo. Sin embargo, también es posible consultar esta información a través de líneas de comandos (en el caso de Windows, a través del cmd o del PowerShell) haciendo uso, entre otras, de una herramienta, que también viene de serie, llamada PnPutil[1], usada, generalmente, por usuarios más expertos. A continuación se muestra un ejemplo de uso de esta herramienta. En este caso, se muestra la existencia de un ratón conectado.

```
Id. de instancia:               HID\SYNA2393&Col01\5&842333f&0&0000
Descripción del dispositivo:    Mouse compatible con HID
Nombre de clase:                Mouse
GUID de clase:                  {4d36e96f-e325-11ce-bfc1-08002be10318}
Nombre del fabricante:          Microsoft
Estado:                         Iniciado
Nombre del controlador:         msmouse.inf
```

Para implementar el Node Summarizer, se ha creado un pequeño programa en Python que hace uso de la herramienta PnPutil a través de una librería que permite ejecutar acciones de línea de comandos y obtener la salida de las mismas.

En concreto la librería "subprocess"[2], y el comando `pnputil /enum-devices /connected` que permite listar los dispositivos que se pueden ver a través del Administrador de Tareas. Una vez listados, el programa los contabiliza separándolos en clases[3] definidas por Microsoft: Volume (para almacenaje), Bluetooth (dispositivos BT), Mouse (ratones), Display (pantallas), Net (elementos de red), Camera (webcams), etc.

En el caso del μSS, se ha creado un segundo programa que proporciona un servicio RESTful en un puerto conocido de forma que los microservicios sepan a dónde deben enviar su información. En concreto, los datos que se envían responden a las diferentes clases definidas por Microsoft (en formato JSON). A continuación, se muestra un ejemplo genérico, bajo este formato, del contenido de la información relativa a posibles microservicios desplegados en un entorno:

```
1:  {''microservices'':[
```

1 https://docs.microsoft.com/es-es/windows-hardware/drivers/devtest/pnputil
2 https://docs.python.org/3/library/subprocess.html
3 https://docs.microsoft.com/en-us/windows-hardware/drivers/install/system-defined-device-setup-classes-available-to-vendors

```
2:    {''name'':''microservice name'',
3:      ''required HW class'':[
4:        {
5:          ''class_name'':''Generic Class'',
6:          ''description'':''additional info if needed''
7:        }],
8:      ''current host'':{
9:        ''ipv4'':''1.2.3.4'',
10:        ''mac'':''00:00:00:00:00:00'',
11:     }
12:   }
13:   ]
14:  }
```

La figura 10 se puede ver cómo el NS y el µSS están corriendo en contenedores aislados. En este ejemplo, conviven con un único microservicio que también está instanciado en un contenedor.

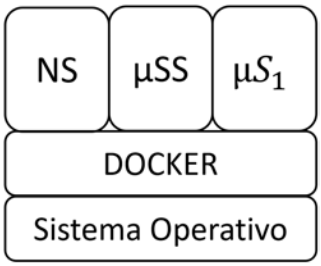


Figura 10: Escenario creado para alojar al NS y al µSS.

Los tres contenedores están aislados entre sí ya que, en Docker, el espacio de nombres es individual para cada contenedor, de forma que cada uno se puede considerar una máquina diferente. Esto implica que desde un contenedor no se puede ejecutar ninguna operación en otros directamente. La única forma de comunicarse es a través de sus puertos y direcciones, como dos máquinas separadas. Esto supone un problema para el NS ya que es el encargado de ejecutar un script que consulta al Sistema Operativo, una capa que para él no existe dado que Docker no permite a los contenedores tener acceso a propio host. Para solucionar esto, al crear el contenedor deben añadirse los siguientes flags que permiten al contenedor tener acceso al propio host:

```
1:  docker run
2:      --privileged
3:      --net=host --pid=host --ipc=host
4:      --volumne /:/host
5:      <nombre imagen>
6:      chroot /host
```

La primera línea arranca un contenedor con la imagen indicada en la línea 5. La segunda línea proporciona privilegios de `root` en el host. Es decir, permite al contenedor tener acceso al host i.e., acceso a archivos, periféricos, etc. La combinación de este flag con los siguientes permite un acceso absoluto. La línea 3 está compuesta por varios flags: dar acceso total a las interfaces de red, dar acceso a la tabla de procesos del host, dar acceso a las IPC del host. El flag `volumne` de la cuarta línea monta el sistema de archivos root del host en el directorio `host` del contenedor. Para poder desplegar al Node Summarizer y hacer correr su script, la imagen que se ha elegido ha sido "busybox", una versión hiperreducida de utilidades UNIX. Por último, el comando `chroot` de la línea 6 da acceso total al host del contenedor. De esta manera, tenemos un contenedor Docker que está corriendo como root con acceso total al host.

4.2 IMPLEMENTACIÓN DEL MÉTODO DE TESTEO DE RENDIMIENTO

En el capítulo anterior, sección 3.2, se propuso un marco de trabajo bajo el cual llevar a cabo tests efímeros de rendimiento a aquellos microservicios que necesiten instanciarse en otros dispositivos. En la sección 3.2.1, se listaron los componentes y consideraciones de diseño para poder realizar estos tests. A continuación, se describe el entorno establecido para llevar a cabo una validación del marco de trabajo propuesto.

Dentro de este entorno se distinguen dos tipos de dispositivos: el orquestador y los hosts. Cada uno de los dispositivos que forman parte del entorno tendrá instanciados los siguientes elementos:

- Orquestador y Hosts:
  - El gestor de contenedores Docker.
- Hosts:
  - Un contenedor ejecutando al Node Summarizer.
  - Un contenedor ejecutando al microService Summarizer.
  - Un contenedor ejecutando una instancia MockServer.
- Orquestador:
  - Un contenedor ejecutando al Broker MQTT.

El papel de cada uno de los dispositivos es:

1. Orquestador. El orquestador será el encargado de:

- Consultar en el repositorio las versiones mock de los microservicios así como los tests que se pueden llevar a cabo. En el caso de las versiones mock, consulta la configuración que se debe llevar a cabo en los MockServers de los diferentes hosts. En el caso de los tests, éstos se han implementado como diferentes scripts que llevan a cabo pruebas de unidad, pequeñas porciones de código que comprueban si un componente concreto funciona correctamente o no. Concretamente, se ha usado la librería *unittest*[4] de Python. Dichos scripts se despliegan dentro de un contenedor, teniendo configurado el destino de los resultados: concretamente al servidor MQTT del orquestador.
- Deplegar el entorno de pruebas. El orquestador se encargará de configurar los MockServers así como de instanciar los contenedores que continen los scripts de los tests.
- Recibir el resultado de los tests. El servidor MQTT que tiene instanciado será el encargado de recibir los resultados de los tests. EL servidor está desplegado dentro de un contenedor Docker[5]

2. Host. Los hosts serán los dispositivos que albergan a los diferentes microservicios presentes en el entorno.

Como prueba de concepto para la validación de la propuesta, se ha establecido un escenario con los siguientes dos microservicios instanciados en diferentes contenedores dentro del mismo dispositivo:

- Un controlador webcam. Se ha utilizado un contenedor de acceso libre desarrollado por la comunidad Docker[6].
- Un detector de caras. En concreto, se ha utilizado la herramienta OpenCV de Python[7] que detecta caras en imágenes.

Las razones por la que se han elegido estos dos microservicios son las siguientes:

- Dependencia entre microservicios: el microservicio de detección de caras necesita como datos de entrada los recogidos por el microservicio que controla la webcam.
- Uso de periféricos: se ha considerado necesario el uso de algún periférico familiar para el usuario, la webcam, dado que la propuesta del presente trabajo se desarrolla cerca del usuario final.

4 https://docs.python.org/3/library/unittest.html
5 https://hub.docker.com/_/eclipse-mosquitto
6 https://registry.hub.docker.com/r/romankspb/webcam
7 https://pypi.org/project/opencv-python/

- Complejidad en el desarrollo de una tarea: se ha estimado oportuno contar con una tarea que implique una complejidad de cálculo y sea compatible con el microservicio que controla la webcam, como resulta ser la tarea de detección de caras.

En este escenario, el dispositivo que alberga estos dos microservicios hace una petición al orquestador para migrar el servicio de reconocimiento de caras. Cuando el orquestador recibe esta petición lleva a cabo las siguientes tareas:

- Establecimiento de los dispositivos candidatos. Dado que el orquestador está suscrito a los microservicios NS y μSS, puede establecer cuáles de los dispositivos presentes es compatible con el microservicio comparando la descripción de dicho microservicio con las de los diferentes dispositivos.

- Consulta al repositorio. En este escenario, el repositorio se ha simulado de forma local: el orquestador consulta una base de datos propia. En ella están guardados 1) los ficheros de configuración de los MockServer, es decir, las expectativas que se deben configurar; y 2) los comandos para instanciar en los dispositivos candidatos los diferentes contenedores de testing. En el caso particular de la migración del microservicio de reconocimiento de caras, el orquestador consultará la configuración relativa al microservicio del que depende, la webcam. Además, consultará los comandos para instanciar las diferentes pruebas que se le puede hacer al microservicio de reconocimiento de caras.

- Configuración de los MockServer. Con la información recopilada tras consultar al repositorio, el orquestador se encarga de realizar la configuración de las expectativas de los MockServer. En este caso, la versión reducida del servicio webcam que se usa como mock para las pruebas consiste en devolver una imagen de entre un set precargado, es decir, no hace uso de una webcam, simplemente imita su comportamiento devolviendo imágenes predefinidas. De esta manera se consigue:
  - que el equipo que va a realizar los tests no tenga microservicios totalmente desplegados, minimizando el uso de memoria y sin interrumpir el funcionamiento normal de los demás servicios ya que no hace uso de otros elementos.
  - el microservicio del que depende en origen no necesita enviar datos a través de la red porque ya se consiguen al consultar las versiones mock.

- Despliegue del microservicio migrante y de los contenedores de testing. En el momento en el que las diferentes instancias MockServer quedan configuradas, el orquestador despliega diferentes instancias del microservicio migrante en los distintos dispositivos candidatos, añadiendo en su configuración la localización de las versiones mock de los microservicios de los que depende. Es decir, se debe indicar adónde debe dirigir sus peticiones, simulando una situación real que permita evaluar su rendimiento. En este caso, al levantar el microservicio OpenCV, se debe indicar la expectativa del MockServer que le va a surtir de imágenes. Además de lo anterior, el orquestador instancia los contenedores de testing. Los contenedores de testing deben conocer la localización del Broker MQTT para poder publicar los resultados de las pruebas.
- Recuperación de los resultados de testing. Cuando los contenedores de testing han ejecutado sus tests, devuelven los resultados al orquestador haciendo una publicación en el Broker MQTT. En el escenario implementado se ha configurado al orquestador para que espere la respuesta de todos los dispositivos candidatos.
- Toma de decisiones. Con los datos recogidos a través del Broker MQTT, el orquestador está en disposición de tomar decisiones incluyendo los resultados obtenidos.

En la figura 11 se puede observar la configuración del escenario para la validación de la propuesta.

4.3 RESULTADOS

Como se ha comentado en el plan de pruebas descrito en el capítulo 3, sección 3.2.2, las pruebas llevadas a cabo son test de unidad con las que se medirá, por un lado, el tiempo de respuesta de los microservicios y, por otro, el tiempo de configuración del entorno de pruebas desde que se inicia hasta que se recuperan los resultados de los tests.

Para validar la presente propuesta, se ha considerado contar con tres dispositivos candidatos, es decir, tres dispositivos que son capaces de albergar al microservicio migrante, en este caso, el microservicio de reconocimiento de caras. A continuación, se desglosan sus características más relevantes, atendiendo a aquellas que afectan a su capacidad de cómputo como son la velocidad de procesado, el número de procesadores, la memoria RAM y su Sistema Operativo:

- Dispositivo 1: 2GHz, 1 Core, 1.9 GB (RAM), Windows 10.

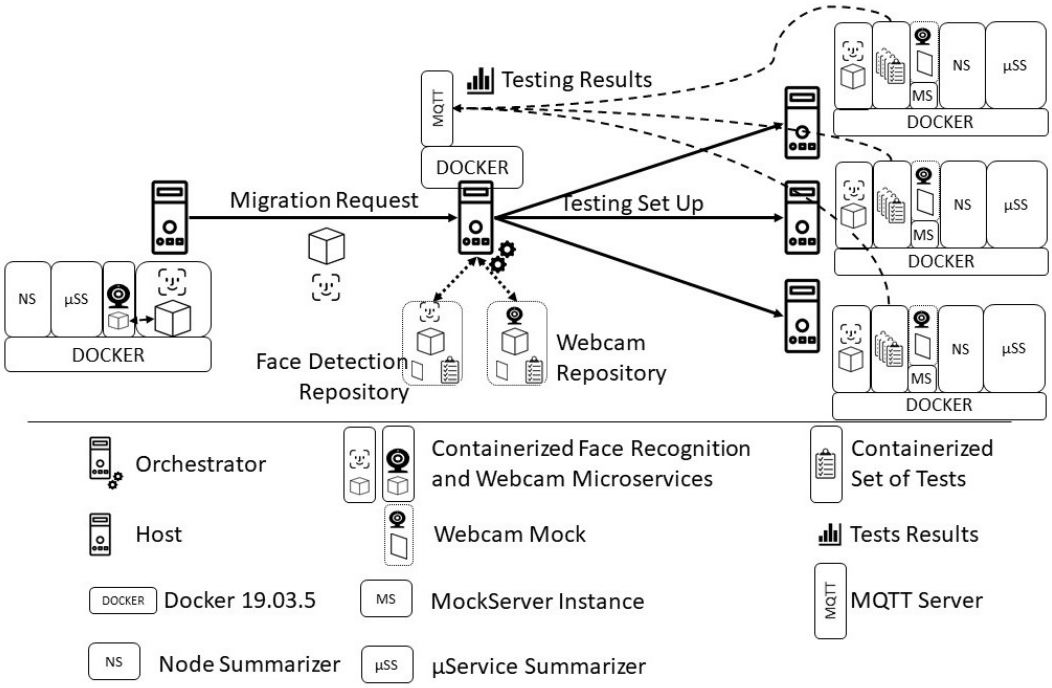


Figura 11: Escenario final para la realización de pruebas de rendimiento.

- Dispositivo 2: 2GHz, 2 Core, 1 GB (RAM), Windows 10.
- Dispositivo 3: 2.66GHz, 4 Core, 2 GB (RAM), Windows 10.

Para llevar a cabo la validación, todos estos dispositivos están dedicados en exclusiva a este ejercicio, con lo que no existen otros procesos corriendo en paralelo más que los descritos en la sección anterior (sección 4.2) como son el Broker MQTT, los servidores MockServer, los NS y los µSS.

Una vez presentada la configuración del entorno, se describe a continuación el tipo de pruebas y mediciones llevadas a cabo. En total, se han realizado 30 repeticiones. Cada repetición consiste en la configuración del espacio de pruebas una vez recibida la petición de migración. El proceso de configuración implica tres pasos 1) configuración de los MockServer; 2) despliegue del microservicio de detección; y 3) llevar a cabo la detección de caras.

Las medidas tomadas han sido dos. La primera, el tiempo que se tarda en configurar los MockServer. Estas medidas se toman dentro de cada uno de los dispositivos. La segunda medida ha sido el tiempo que requiere el proceso de detección de caras. Teniendo en consideración las características de los tres dispositivos candidatos, cabría esperar que el dispositivo 3

tuviera una mejor respuesta ya que cuenta con mayor velocidad de cómputo, mayor cantidad de procesadores, así como mayor cantidad de memoria RAM.

En la figura 12 se muestran los tiempos obtenidos. Estos tiempos muestran diferencias notorias entre los tres tipos de dispositivos evaluados. Por un lado, la distribución de los tiempos del primer dispositivo tienen mayor desviación típica que el resto. Por otro lado, los dispositivos D2 y D3 tienen una relación casi de uno a dos en sus características: D3 es solo un tercio más rápido que D2 pero tiene el doble de núcleos y el doble de memoria RAM. Con estos números, podemos entender por qué sus tiempos presentan una desviación típica menor y medias más bajas que D1, presentando D3 mejores tiempos que D2. Si consideramos las características de los dispositivos, de forma intuitiva cabe pensar que el tercer dispositivo, D3, sería el mejor candidato para albergar un nuevo microservicio ya que es más rápido y tiene más memoria RAM que los demás. Esto encaja con los resultados mostrados en la figura 12, donde se ve cómo los tiempos medios de configuración del MS y la detección de caras son menores que en el resto de dispositivos.

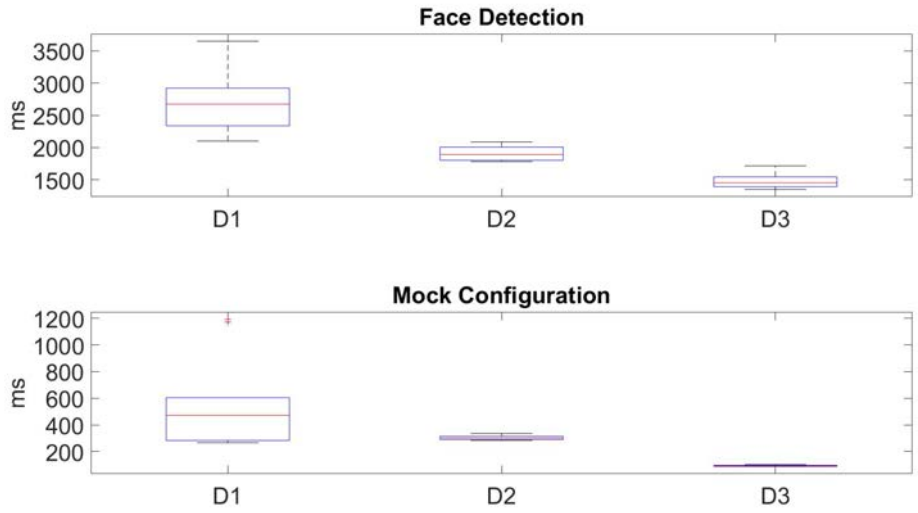


Figura 12: Medidas de tiempos en el despliegue del entorno de testeo.

4.4 CONCLUSIONES

En el capítulo anterior, capítulo 3, se han descrito dos entidades, el Node Summarizer y el microService Summarizer, que, desde el punto de vista de este trabajo, deberían estar presentes en entornos de computación distribui-

da basados en arquitecturas de microservicios. Su papel es el de facilitar tareas al orquestador del entorno. Así mismo, se propuso un mecanismo para la toma de métricas que permitan a este orquestador tener más datos sobre los que basar la toma de decisiones a la hora de determinar si un microservicio puede o no migrar de un dispositivo a otro. El capítulo anterior, capítulo 3, junto con el presente, capítulo 4 han servido para satisfacer el primero de los objetivos descritos en el capítulo 1, sección 1.2. En particular,

- Se han establecido los requisitos y funcionalidades, sección 3.1.2, que deben cumplir tanto fabricantes como desarrolladores para proporcionar los dos microservicios presentados en la sección 3.1.1. En el capítulo 4, sección 4.1, se ha llevado a cabo una posible implementación de estos dos microservicios siguiendo estas pautas consiguiendo, de esta manera, proporcionar un mecanismo para describir a los microservicios como a los dispositivos presentes, objetivo O1.1, como proporcionar un marco de trabajo para analizar la compatibilidad que existe entre ellos, objetivo O1.2.

- En el capítulo 3, sección 3.2, se ha propuesto el marco de trabajo bajo el cual llevar a cabo pruebas de corta duración a los microservicios que se quieran instanciar en otro dispositivo de forma que se puedan obtener más datos con los que un orquestador pueda tomar decisiones de despliegue. En el capítulo 4, sección 4.2, se ha descrito la implementación realizada de los diferentes componentes necesarios para poder crear estos entornos de pruebas cortas. De esta manera, se ha conseguido satisfacer el objetivo O1.3, consiguiendo:
  - crear escenarios de pruebas efímeros;
  - que el consumo de recursos sea mínimo;
  - evitar la interrupción del comportamiento normal de los microservicios presentes;
  - añadir una fuente de datos extra para la toma de decisiones.

- Por último, se ha llevado a cabo un plan de pruebas, 3.2.2 y una implementación del mismo que ha servido como prueba de concepto para comprobar que la propuesta teórica de la arquitectura es viable. Así mismo, consideramos que las medidas obtenidas, sección 4.3, sobre los tiempos necesarios para desplegar el entorno y llevar a cabo los test es asequible y razonable.

En los siguientes dos capítulos se expone el trabajo llevado a cabo para alcanzar el segundo de los objetivos (O2) descrito en el capítulo 1, sección 1.2: diseñar un intérprete de órdenes en lenguaje natural gracias al cual el

usuario pueda interactuar con su entorno.

En primer lugar, en el capítulo 5 se hará una descripción del diseño del intérprete. Así mismo, se planteará el plan de pruebas llevado a cabo para la validación del diseño.

En segundo lugar, en el capítulo 6, se detalla tanto la implementación del diseño descrito como el resultado de la evaluación llevada a cabo.

Como el trabajo llevado a cabo en esta tesis busca, como fin último, satisfacer el tercero de los objetivos, O3, contribuir a la interoperabilidad continua entre humanos y sistemas automatizados, en el capítulo 7 se recoge la integración de la arquitectura de microservicios, descrita en los capítulos 3 y 4, con el intérprete de órdenes, expuesto en los capítulos 5 y 6.

5

COMUNICACIÓN DEL USUARIO CON SU ENTORNO: DESCRIPCIÓN

El presente capítulo describe cómo se ha alcanzado el segundo objetivo expuesto en la introducción (sección 1.2), en el que se indicaba la idoneidad del lenguaje natural para el control de los dispositivos, así como los retos a los que era necesario hacer frente. Como contribución a este objetivo, se razonaba la necesidad de desarrollar un interprete de lenguaje natural que permitiese al usuario interactuar con el entorno usando su lenguaje natural, sin necesidad de recordar palabras clave y sin necesidad de entrenar al sistema previamente. En este capítulo, se describe el diseño del intérprete de órdenes desarrollado, así como un plan de pruebas para su validación. En el siguiente capítulo, capítulo 6, se describe la implementación llevada tanto del intérprete como del plan de pruebas.

En base al cumplimiento del segundo de los objetivos descritos en la sección 1.2, O2, este capítulo está organizado de la siguiente manera. En primer lugar, se detallan los componentes del intérprete, identificando las tareas que realizan, sección 5.1. A continuación, se presenta el concepto de *orden primitiva*, resultado del análisis de las peticiones dadas por el usuario, sección 5.2, cumpliendo el objetivo O2.1. Seguidamente, se verá la metodología a seguir para que el intérprete conozca el entorno en el que está y las diferentes acciones que se pueden llevar a cabo en dicho entorno, sección 5.3, satisfaciendo el segundo subobjetivo O2.2. En la siguiente sección, sección 5.4, se describe cómo se establecen las relaciones entre las peticiones efectuadas por el usuario y las acciones que se pueden llevar a cabo en el entorno, objetivos O2.3 y O2.4. Finalmente, en la sección 5.5 se estable el plan de pruebas llevado a cabo para validar esta propuesta.

5.1 EL INTÉRPRETE

Antes de pasar a describir el diseño del intérprete, es necesario mencionar que este diseño está centrado en cómo se establece la relación entre los términos usados por el usuario y las diferentes acciones que se pueden llevar a cabo dentro del entorno que rodea al usuario. Este trabajo no ahonda en la forma en la que las órdenes son captadas ni su fuente (e.g., oral, gestual o por introducción de texto). Su funcionamiento se basa en la existencia de una transcripción de lo que originalmente podría no ser tex-

to. Además, se debe comentar que el trabajo desarrollado se ha llevado a cabo haciendo uso exclusivo del inglés, con lo que solo se han considerado transcripciones hechas a ese idioma.

A la hora de diseñar el intérprete se han considerado los siguientes requisitos que se deben cumplir:

- El intérprete debe ser capaz de interpretar la intención del usuario analizando sus órdenes dadas en lenguaje natural. Es necesario establecer una serie de normas que permitan al intérprete reconocer los términos que son clave en una orden.
- Así mismo, debe conocer las diferentes acciones que se pueden llevar a cabo en el entorno. Hace falta implantar un mecanismo que permita al intérprete reconocer qué acciones existen.
- Su finalidad es la deducir, tras el análisis de la intención del usuario, qué acción se debe llevar a cabo de entre todas las posibles. Por lo tanto, es necesario establecer un mecanismo con el que se puede llevar a cabo una comparación entre lo deseado por el usuario y las posibilidades del entorno.
- Además, el intérprete es una aplicación desplegada en algún dispositivo del entorno que se debe comunicar con un orquestador para que éste inicie las acciones que el usuario desea llevar a cabo. De esta manera, es necesario crear un método de comunicación entre el orquestador y el intérprete.

Basándonos en estos requisitos, el intérprete quedará desplegado en un entorno compuesto por los siguientes elementos: el usuario, el propio intérprete, un orquestador y los diferentes dispositivos que se quieren controlar. El intérprete media entre el usuario y los dispositivos, los cuales están administrados y controlados por el orquestador. El orquestador es el encargado de gestionar y de descubrir a los diferentes dispositivos y servicios que forman parte del entorno. El despliegue del intérprete se lleva a cabo en un entorno controlado por un orquestador, de manera que entre ambos se establece un diálogo en el que se comunica, en sentido orquestador-intérprete, las diferentes acciones que se pueden llevar a cabo y, en sentido intérprete-orquestador, la acción que el usuario quiere llevar a cabo.

Los módulos que forman parte del intérprete son:

- Filtro de órdenes. Es la parte del intérprete encargada de analizar la orden en lenguaje natural para reducirla, como se verá en la sección

siguiente, a un conjunto de datos que la representan y que el intérprete manejará.

- Comparador y administrador de clústeres. El término "clúster" en este contexto se presenta en la siguiente sección. Estos dos módulos forman el núcleo de la inteligencia del intérprete. Son ellos los encargados de tomar, por un lado, los datos filtrados de la orden original y, por otro, los datos del entorno, concluyendo qué acciones el usuario quiere llevar a cabo.
- Buzón de entrada y salida. Es la parte del intérprete encargada de comunicarse con el orquestador del entorno.

En la figura 13 se muestra el esquema de despliegue del intérprete, así como se identifican cada uno de los módulos que lo componen. En la siguiente sección se desarrolla la funcionalidad de cada uno de estos módulos.

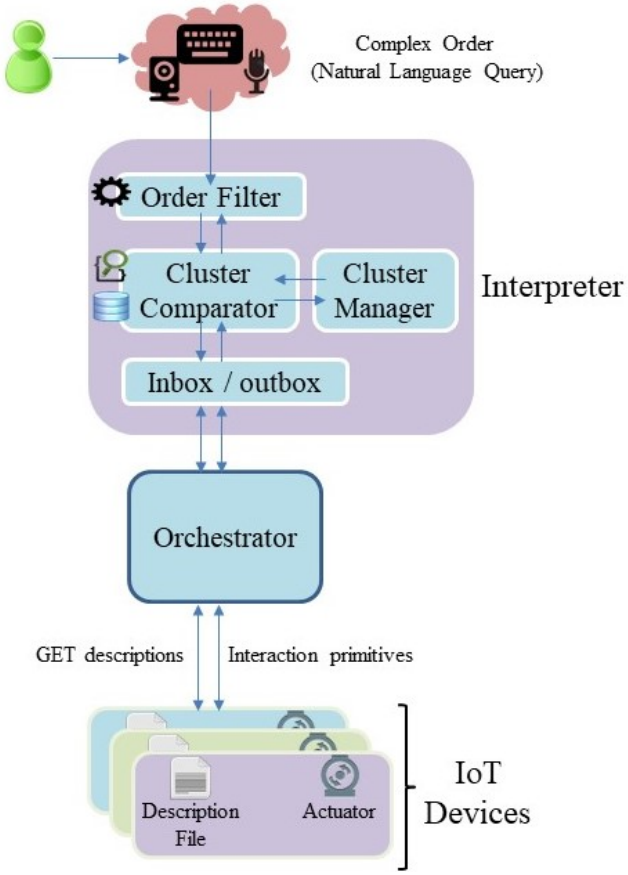


Figura 13: Esquema de despliegue del intérprete

5.2 DISEÑO Y FUNCIONALIDAD: DESCRIPCIÓN DE LOS MÓDULOS

En esta sección se presentan cada uno de los módulos que conforman al intérprete. En concreto, se describe su diseño y funcionalidad. En la siguiente sección, sección 5.3, se profundiza en la descripción del entorno, mientras que en la sección 5.4 se describe cómo se lleva a cabo el emparejamiento entre deseos del usuario y las acciones presentes.

5.2.1 *Filtro de órdenes.*

Es la parte encargada de analizar sintácticamente la transcripción de la orden original. Concretamente, este análisis etiqueta cada uno de los términos, estableciendo su papel sintáctico dentro del texto. Los términos se pueden etiquetar como pronombres, nombre, adjetivos, verbos, y así seguidamente. Una vez establecidas las relaciones, se extraen aquellos términos más relevantes, formando lo que se ha llamado en este trabajo como *orden primitiva*. Una orden primitiva es un vector de términos relevantes y no una cadena de texto. Cada elemento del vector puede estar compuesto por más de una palabra. Por ello, en adelante, cada término de la orden primitiva se va a denominar "tag", o etiqueta. Como se ha descrito en el estado del arte, un tag está constituido por una o varias palabras que etiquetan un concepto, de forma que ayudan a describir y a comprender dicho concepto (ver 2.6.1.2).

Cuando un usuario da una orden, de ella se extraen los términos más relevantes, aquellos que aportan mayor información creando la orden primitiva. Para saber qué términos son los más relevantes, se analizan las peticiones y de ellas se seleccionan los que sean principales usando una herramienta de procesamiento de lenguaje natural, Natural Language Processing (NLP) en inglés. En el caso del presente trabajo, se ha decidido usar la herramienta desarrollada por la universidad de Stanford que recibe el nombre de Stanza[1] [267], cuya predecesora fue CoreNLP [268]. Ésta sirve, entre otros usos, para hacer un marcado de texto en el que se anotan las relaciones entre las palabras, es decir, su dependencia sintáctica: verbos, nombres, adjetivos, etc.

De entre todas las relaciones posibles entre los términos, se han elegido las siguientes relaciones sintácticas. La razón por la que se ha concluido que dichas relaciones son las que permiten obtener, de una orden, la mayor información posible eliminando los términos menos relevantes, es que los sustantivos ("nouns" en inglés), los verbos, los adjetivos y los adverbios,

1 https://stanfordnlp.github.io/stanza/

están presentes en la gran mayoría de los idiomas. Los sustantivos hacen referencias a elementos concretos. Los verbos se refieren a las acciones, cuya identificación es, precisamente, la labor principal del intérprete. En cambio, los adjetivos modifican a los sustantivos y los adverbios a los verbos, completando su intención y añadiendo una mayor concreción. En concreto, las etiquetas que representan estas relaciones dentro de la herramienta Stanza, son las siguientes:

- dobjt. Direct Object. El objeto del verbo.
- nmod. Noun Modifier. Un término que modifica o especifica el significado de un término.
- compound. Modifica o especifica el significado de una término formando uno compuesto.
- nummod. Numeric Modifier. Especifica el significado en un término con una cantidad.
- case. Añade información sobre elementos nominales.
- nsubj. Nominal Subject. El sujeto sintáctico de una cláusula.
- advmod. Adverb Modifier. Adverbio que modifica el significado de una palabra.
- root. El verbo.

5.2.2 *Cluster Manager*

Gracias al Cluster Manager, el intérprete es capaz de determinar cuánto de parecidas son dos órdenes primitivas compuestas por diferentes tags. Para llevar a cabo esta tarea, el Cluster Mánager se apoya en la consulta de diccionarios de sinónimos y términos relacionados. En este trabajo, los términos se extraen de una base de datos desarrollada por la Universidad de Princeton llamada WordNet [269]. Se trata de una de las base de datos léxica más utilizada por la academia, donde nombres, verbos, adjetivos y adverbios están agrupados en conjuntos de sinónimos y palabras relacionadas.

Un cluster de tags es una agrupación de términos. Cada cluster contiene dos tipos de tags: sinónimos y términos relacionados. Los tags que se utilizan para generar estos clústeres provienen de dos fuentes: las órdenes primitivas y las descripciones de las acciones. Las descripciones de las acciones, como se describe con mayor precisión en la sección 5.3, contienen

un conjunto de tags. Por cada uno de esos tags se realiza una consulta a la base de datos nombrada anteriormente, WordNet, de forma que se extraen un conjunto de sinónimos y términos relacionados. Todos estos términos, el tag que originó la consulta, así como los términos obtenidos en la misma, forman un cluster. Dado que existen términos que pueden ser, dependiendo del contexto, un verbo, un adjetivo, un adverbio o un sustantivo, cada uno de los clústeres, además de los términos en sí, tiene indicado de qué tipo es. Gracias a esta agrupación de los términos, cuando el intérprete analiza una orden primitiva, que está compuesta por tags, puede consultar a qué clústeres pertenecen dichos tags, de forma que puede establecer cuál de las acciones encaja mejor con la orden dada. En la sección 5.4 se describe este proceso.

Sirva el siguiente ejemplo para describir la creación de clústeres. Si una descripción de una acción incluyera el tag "light", como verbo, el cluster manager haría que este tag formará un cluster con otros tags como "switch on" o "illuminate". Así, el uso de cualquiera de estos tres tags se considerará equivalente. Este mismo término, "light", utilizado como nombre (noun en inglés), estará relacionado, entre otros, con "lightness" y "brightness", formando un segundo clúster.

5.2.3 *Cluster Comparator*

Este módulo es el encargado de llevar a cabo la comparación entre los tags que componen una orden primitiva y los tags relacionados con las diferentes acciones del entorno. En concreto, sus tareas son:

1. Conocer las descripciones de las acciones que se pueden realizar en un entorno.
2. Extraer tags relevantes de las descripciones de las acciones.
3. Consultar y crear junto al cluster manager los clusters en los que están los tags asociados a las acciones.
4. Determinar qué acción quiere realizar el usuario. Para ello, consulta al cluster manager en qué clusters están los tags de la orden primitiva y los compara con los clusters a los que pertenecen los tags de las descripciones asociadas a las acciones.

Las tres primeras tareas descritas se engloban en la siguiente sección, 5.3, mientras que la descripción de la cuarta tarea se encuentra en la sección 5.4.

5.3 DESCRIPCIÓN DEL ENTORNO Y SUS ACCIONES

En este trabajo, el término "acción" hace referencia a una tarea que puede llevar a cabo algún elemento del entorno. Esas acciones implican cambios en el entorno, es decir, un cambio en el estado de algún elemento del mismo, ya sean estos cambios físicos o virtuales. El término "elemento" se refiere a las diferentes aplicaciones que están desplegadas en los dispositivos que forman parte del entorno. Es decir, las acciones que se pueden ejecutar en un entorno las llevan a cabo elementos software desplegados en los dispositivos, por ejemplo, una aplicación, un servicio o un microservicio.

Cuando el intérprete recibe las descripciones de las diferentes acciones del entorno, éstas son analizadas por el Cluster Comparator. Las descripciones son proporcionadas por el orquestador, el cual las recopila de los dispositivos del entorno. En el presente capítulo, así como en el siguiente, capítulo 6, se presupone que esta recopilación de datos, así como su intercambio entre orquestador e intérprete, es posible. En el capítulo 7 se propone un mecanismo para llevar a cabo este intercambio, utilizando el trabajo descrito en los capítulos anteriores, capítulos 3 y 4.

Continuando con la descripción, cada elemento debe tener asociado qué acciones puede realizar. Estas descripciones son analizadas por el Cluster Comparator, junto con el Cluster Manager, y de ellas se extraen los términos más relevantes, generando un conjunto de tags. Las descripciones las suministra el equipo que desarrolle ese elemento. Asimismo, existe la posibilidad de añadir, además de la descripción, un conjunto de tags inicial. De esta manera, cuando una descripción se analiza para obtener unos tags relacionados con la acción, se pueden añadir los proporcionados por el fabricante.

Con la idea de establecer una método estándar para proporcionar las descripciones, se va a hacer uso de ontologías. A continuación, se describe la ontología utilizada para describir tanto a los elementos que forman parte del entorno como a las acciones que pueden llevar a cabo.

5.3.1 *Ontologías*

En este trabajo, se ha utilizado el lenguaje OWL, en inglés, Web Ontology Language. Se trata de un lenguaje semántico diseñado para representar información rica en contexto sobre elementos abstractos, también llamadas *clases*. Esta información define las relaciones entre clases. OWL formaliza

los dominios en los que se encuentran las clases, define a cada clase y declara las propiedades que tienen.

Por un lado, la ontología propuesta que describe a los elementos se muestra en la figura 14. El término *domain* indica a qué elemento corresponde. El rango, *range*, señala qué tipo de valor es. Por último, el tipo, *type*, apunta qué tipo de clase es. El campo "anyURI" al que hace referencia la característica "canDo" identifica de forma unívoca a una acción. Un elemento puede tener varias ramificaciones "canDo", indicando que puede realizar varias acciones.

Por otro lado, la ontología que describe a las acciones que un elemento puede realizar se muestra en la figura 15. Las acciones tienen asociados dos campos: su descripción y una conjunto de tags. En la siguiente subsección se describe el origen de estos términos y su manejo dentro del Cluster Manager.

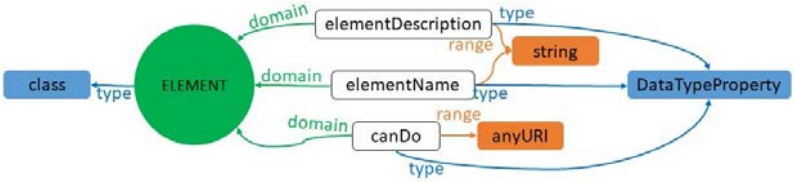


Figura 14: Representación de la ontología a la que responden los objetos.

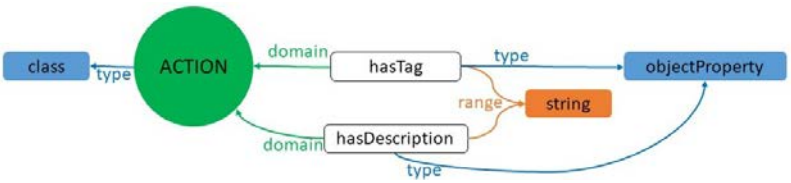


Figura 15: Representación de la ontología a la que responden las acciones.

5.3.2 *Etiquetado de acciones y su expansión*

Como ya se ha comentado, cada una de las acciones que se pueden disparar dentro de un entorno debe tener asociada una descripción de la misma. Además, de forma opcional, puede tener asociada un conjunto de tags. Esta idea está presente en nuestro día a día, ya sea con un apoyo visual (como iconos en cualquier aplicación software) o a través de descripciones breves (como la información que surge al dejar el cursor del ratón encima de algún icono).

En el Estado del Arte se hizo especial hincapié en el problema principal a la hora de seleccionar los términos adecuados que describan un recurso: idealmente, deben encajar con las términos que cualquier persona usaría (ver 2.6.1.1). Este es un problema doble. Por un lado, es cierto que relacionar un recurso con un número elevado de palabras haría que un gran número de usuarios diera con alguno de los términos asociados a ese recurso. Los hábitos, el nivel cultural, así como la segmentación por edades, marcan grandes diferencias en el uso del lenguaje, con lo que cabe pensar que más palabras implican más acierto. No obstante, el número de falsos positivos crecería ya que habría muchos recursos que encajasen relativamente bien con los términos usados por el usuario. Por otro lado, buscar una definición sucinta, y bastante precisa de un recurso, bajaría el número de falsos positivos pero no sería compatible con una gran parte de los usuarios potenciales ya que no tendrían por qué dar con los términos adecuados (ya que hay menos), dependiendo de su nivel cultural, circunstancias socioeconómicas y el lenguaje del día a día.

Precisamente, para dar solución a este problema, este trabajo propone hacer uso de sinónimos y palabras relacionadas de manera controlada a través de diccionarios. Para poder hacerlo, las diferentes acciones tienen asociadas unas descripciones básicas dadas por el desarrollador (más sucintas que las que podríamos encontrar en su manual de uso). El orquestador con el que se comunica el intérprete se las envía siguiendo las ontologías descritas anteriormente (5.3.1). Esta información se puede enviar utilizando el formato de algún lenguaje de notación, como JSON, OWL, u otros. No es fin de este trabajo hacer un análisis comparativo entre las diferente opciones.

Una vez el intérprete haya recibido una descripción, la procesa utilizando la herramienta de análisis de lenguaje natural descrita en la sección 5.2.1. Como resultado del procesamiento, se identifica el papel léxico que tiene cada término (verbo, adjetivo, etc.), generando un conjunto inicial de tags asociados. Este conjunto contendrá un subconjunto de los términos utilizados en la descripción, resultado de considerar las relaciones descritas en esa misma sección.

Una vez hecho el filtrado de las descripciones y extraídos los tags más relevantes, cada uno de esos tags pasa a formar parte de un cluster, el cual se expande con sinónimos y términos relacionados. Los términos relacionados son aquellos que, basándose en características culturales, son usados frecuentemente en combinación con el término original en la descripción. El origen de estos términos es la herramienta WordNet descrita en la sección 5.2.2. Tras este proceso, el Cluster Manager establece y guarda una

relación entre las acciones del entorno y los clústeres de tags, de forma que las acciones pasan a tener asociados los clústeres de los que forman parte los tags extraídos de las descripción de la acción.

A continuación, se ilustra este proceso con un ejemplo sencillo. En la tabla 1 se pueden ver las diferentes acciones que una televisión puede llevar a cabo. En esta tabla se puede ver tanto el nombre de la acción como su descripción.

Action Name	Action Description
switch off	switches off the television
switch on	switches on the television
change channel	changes the television channel

Tabla 1: Ejemplo: descripción de las acciones de una televisión

Cuando las descripciones de las acciones son analizadas por el Cluster Manager, se obtienen los tags más relevantes de éstas. En la tabla 2 se han añadido los tags que se extraen de las descripciones de la tabla 1.

Action Name	Action Description	Tags
switch off	switches off the television	television, switch off
switch on	switches on the television	television, switch on
change channel	changes the television channel	television, change, channel

Tabla 2: Ejemplo: Etiquetas asociadas a las acciones de una Televisión

El siguiente paso es agrupar en clústeres los términos, añadiendo sinónimos y otros términos relacionados. Por ejemplo, el tag "television" forma parte del mismo cluster que los términos "telly" o "tv" (dos términos informales). Continuando con el ejemplo de la tabla anterior, las tablas 3 y 4 ilustran esta actualización. En la tabla 3 se pueden ver los identificadores de los clústeres en los que se encuentran los diferentes tags de la tabla 2. Cada uno de estos clústeres contiene una serie de tags relacionados dependiendo de su etiqueta semántica (verbo, adjetivo, etc).

En la tabla 4 se pueden ver los diferentes tags que están relacionados entre sí.

Action Name	Action Description	Cluster IDs
switch off	switches off the television	1234, 5678
switch on	switches on the television	1234, 9012
change channel	changes the television channel	1234, 3456, 7890

Tabla 3: Ejemplo: Clústeres asociados a las acciones de una Televisión

Cluster IDs	Type	Related words
1234	Noun	television, TV, telly, television set, idiot box, boob tube, tv set
5678	Verb	switch off, cut, turn off, turn out
9012	Verb	switch on, turn on
3456	Verb	switch, shift, change, vary, alter, modify
7890	Noun	channel, television channel, TV channel, communication channel, line, groove, transmission channel, duct, epithelial duct

Tabla 4: Ejemplo: Etiquetas de los clústeres de la tabla 3

5.4 COMPARACIÓN DE CLUSTERS

Una vez procesadas todas las descripciones de las acciones y establecidos los clústers de tags que están relacionados con ellas, ya es posible atender a las peticiones del usuario, utilizando la misma herramienta de análisis de language natural Stanza. Cuando se analiza la orden se obtiene la orden primitiva de dicha orden. A continuación, el Cluster Comparator consulta al Clúster Manager los clústeres a los que pertenecen los tags de la orden primitiva. Este conjunto de clústeres se compara con los clústeres que están relacionados con cada una de las acciones. El conjunto al cuál se parezca más indicará qué acción debe ejecutarse.

Para comparar dos sets de datos se ha utilizado el índice de similitud de Jaccard, o coeficiente de Jaccard, que indica cuál es el ratio de elementos comunes entre dos conjuntos. Entre los conjuntos A y B, el coeficiente de Jaccard, CJ, se determina como:

$$CJ(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

El valor CJ está comprendido entre 0 y 1, incluidos, indicando el valor 0 que no existe ninguna similitud entre ambos conjuntos, mientras que el valor 1 indica que ambos contienen a los mismos elementos, es decir, que son iguales.

De esta manera, cada vez que se procesa una orden primitiva, se calcula su similitud con todas las acciones posibles. Dado que los verbos se refieren a acciones y que este trabajo está focalizado en la identificación de las acciones, se ha establecido una ponderación extra a los verbos. En concreto, se ha incrementado el valor CJ atendiendo a si el verbo utilizado en la orden primitiva está incluido en algún cluster de una acción. A este valor extra se le ha dado el nombre de *incremento de verbo*, IV. Si esto sucede, el coeficiente de Jaccard se incrementa de la siguiente manera:

$$CJ(A,B)_{final} = CJ(A,B)(1 + IV) \tag{2}$$

con $0 \leqslant IV \leqslant 1$. Asimismo, se ha establecido que el valor CJ_{final} sature a 1. Este incremento se ha establecido para asegurar una distancia mayor entre aquellas acciones cuyos conjuntos de clústeres resulten muy parecidos. Por ejemplo, si observamos la tabla 3 podemos ver que los clústeres asociados a las acciones "switch off" y "switch on" se solapan, siendo diferente el clúster asociado al verbo.

Una vez calculados los coeficientes de Jaccard, éstos se utilizan para ordenar las acciones de mayor a menor probabilidad de acierto. Además, se ha indicado un valor mínimo del coeficiente de Jaccard, CJ_{min}, para que el sistema considere cuánto de seguro está sobre si una acción es la que debe realizarse. Esto significa que, si el CJ es mayor que CJ_{min}, el sistema puede dar por seguro que esa acción es la que se desea activar.

En la siguiente sección, sección 5.5, se establece el plan de pruebas llevado a cabo para validar la propuesta del presente capítulo y, en ella, se establecerá el estudio que se llevará a cabo en función de los valores posibles que pueden tomar el CJ_{final} y el IV.

5.5 PLAN DE PRUEBAS

Con la idea de validar la presente propuesta, se ha definido el siguiente plan de pruebas compuesto por los siguientes elementos y configuraciones:

- Simulación de un entorno real. En este plan de pruebas, se va a configurar un entorno virtual en el que poder crear acciones asociadas a elementos que representan reales. Es decir, estos elementos virtuales emulan el comportamiento de elementos que, en un escenario real, cambian su estado a través de diferentes aplicaciones software. En particular, se simulará un hogar inteligente.

- Uso de un modelo de actores. Con la idea de hacer una simulación concurrente, se utilizará este modelo en el que cada elemento será un actor [270]. Un actor puede realizar tres tareas: crear otro actor, enviar un mensaje y actuar cambiando su estado al recibir un mensaje. En esta simulación, los actores serán, a saber: los elementos que realizan acciones, así como el orquestador y el intérprete.

- Utilización por parte de usuarios reales. Dado que se quiere emular un entorno real y que el intérprete interactúa con el usuario, la presente propuesta se va a poner a prueba con usuarios reales. En concreto, se recogerán órdenes dadas en lenguaje natural por usuarios reales. A los usuarios se les presentará una situación con un objetivo. El usuario debe cumplir este objetivo dirigiéndose al entorno en lenguaje natural.

- Mediciones. En al apartado 5.4 se han definido tres valores, el coeficiente de Jaccard final (CJ_{final}), el coeficiente de Jaccard mínimo (CJ_{min}) y el incremento de verbo (IV), en los que el intérprete se basa para tomar decisiones. En este plan de pruebas se quiere hacer una comparación entre las combinaciones posibles de los valores CJ_{min} e IV para determinar si existe alguna diferencia en el rendimiento del intérprete.

- Aprendizaje. Además de hacer comparaciones con diferentes valores CJ_{min} e IV, se quiere hacer una comparación entre un sistema con retroalimentación y el mismo sistema sin ella. Siendo más exactos, el sistema con retroalimentación, cuando el intérprete no pueda concluir con certeza qué acción se quiere llevar a cabo, pedirá al usuario que seleccione la correcta de entre aquellas de las que duda el sistema. Así, el sistema anotará la orden primitiva en esa interacción asociada a esa acción.

5.6 CONCLUSIONES

A lo largo de este capítulo se ha realizado una propuesta principal: el diseño de un intérprete de órdenes naturales dadas por usuarios reales. La funcionalidad de este intérprete se apoya en dos pilares: el análisis del lenguaje natural y el uso de diccionarios de sinónimos y términos relacionados. Como ya se ha visto, cuando un usuario realiza una petición al entorno en lenguaje natural, el intérprete resume dicha petición creando lo que se ha denominado orden primitiva. El análisis de esta orden primitiva, junto a la gestión de términos relacionados y sinónimos, permite que los usuarios no tengan la necesidad de emplear los términos específicos que activen una orden en particular, minimizando el esfuerzo requerido al usuario para interactuar con el entorno. Este diseño busca satisfacer el primer subobjetivo, O2.1, del segundo objetivo principal presentado en la introducción de este documento, sección 1.2.

En la sección 5.3 se ha propuesto un mecanismo que permite al intérprete conocer el entorno en el que se encuentra de forma que sea capaz de concluir qué acción quiere llevar a cabo un usuario. Este mecanismo se basa en la participación de los desarrolladores de las aplicaciones, siendo éstos los que describan, de forma sucinta, las acciones que un elemento puede realizar. Dado que los desarrolladores se ven en la necesidad de incluir en sus productos manuales de usuario, ya sea para un usuario medio o documentación más específica para usuarios más técnicos, pretender que los desarrolladores incluyan una breve descripción de las acciones no debería considerarse un esfuerzo extra por su parte. De esta forma, el mecanismo basado en ontologías presentado en dicha sección que sirve para llevar a cabo la tarea de que el intérprete conozca el entorno, satisface el segundo y el tercero de los subobjetivos, 02.2 y O2.3, del segundo objetivo principal presentado en la introducción de este documento.

Por último, con la idea de cumplir el cuarto de los subobjetivos, O2.4, del segundo objetivo principal, el diseño presentado no incluye un sistema de entrenamiento previo. Dado que el análisis de las órdenes, así como el uso de diccionarios de sinónimos y términos relacionados, son herramientas dadas que funcionan por sí solas, el intérprete no requiere entrenarlas sino que, simplemente, se apoya en ellas.

En el siguiente capitulo, capítulo 6, se presenta una prueba de concepto con el fin de validar esta propuesta. En concreto, se establecerá un escenario de pruebas con las características descritas en el plan de pruebas, sección 5.5. Como se ha indicado en dicha sección, este escenario recoge elementos

y acciones que se pueden encontrar en un hogar inteligente y con los que un usuario medio suele interactuar. En particular, la sección 6.1, recoge los elementos virtuales presentes en el entorno y las acciones que llevan a cabo. En dicha sección, los elementos son aplicaciones software. En cambio, los elementos recogidos en la sección 6.2, son elementos físicos cuyo estado está controlado por algún tipo de software.

Con la idea de demostrar la viabilidad de este diseño, como se ha indicado en el plan de pruebas, el sistema se pondrá a estudio con usuarios reales. Cada usuario responderá a un formulario de situaciones en el que se plantearán diferentes objetivos de entre todas las acciones posibles, procurando no dar ninguna indicación sobre los términos que se pueden utilizar. La parámetro que se quiere medir es la precisión del sistema a la hora de elegir la acción correcta. Para ello, utilizando el mismo conjunto de respuestas de los usuarios, se pondrá a examen el sistema con diferentes valores de los parámetros definidos en la sección 5.4. Además, se hará una comparación entre un sistema donde exista retroalimentación por parte del usuario y uno en el que no. Los resultados obtenidos de esta comparación se muestran en la sección 6.3.

6

COMUNICACIÓN DEL USUARIO CON SU ENTORNO: VALIDACIÓN

El capítulo anterior, capítulo 5, sección 5.1, se han presentado una serie de requisitos que el intérprete debe satisfacer. Bajo estos requisitos, ha quedado establecido el entorno en el que se despliega, así como los módulos del mismo, sección 5.2. En esta descripción de los módulos se han nombrado dos herramientas sobre las que se apoya este trabajo, la herramienta Stanza para el filtrado de las peticiones en lenguaje natural, creando la orden primitiva; y WordNet, utilizada para la administración de los clústeres de términos relacionados y sinónimos. Así mismo, se ha propuesto una ontología para el intercambio de información entre el entorno y el intérprete con la finalidad de que éste conozca las diferentes acciones que se pueden ejecutar, sección 5.3. Además, se ha presentado el método definido para la comparación de clústeres, cuya finalidad es concluir qué acción de entre las posibles el usuario desea llevar a cabo, sección 5.4. Finalmente, en la sección 5.5 se ha descrito el plan de pruebas diseñado para la validación de la propuesta. En el presente capítulo se lleva a cabo dicha validación.

En concreto, en la sección 6.1, se describe el entorno en el que se encuentra el intérprete, así como cuáles son las acciones presentes en el mismo. Como se ha indicado en el plan de pruebas, se trata de una simulación de un hogar inteligente. En la sección 6.2, se presentan las herramientas utilizadas para implementar el modelo de actores descrito en el plan de pruebas, así como se describen los pasos seguidos para establecer el entorno virtual, además de la metodología seguida para recopilar órdenes de usuarios reales. Por último, en la sección 6.3, y siguiendo con el plan de pruebas, se muestran los resultados obtenidos utilizando diferentes valores de los parámetros descritos en la sección 5.4, poniendo a prueba al intérprete con retroalimentación del usuario como sin ella. En la misma sección se detalla el método de ponderación de la tasa de acierto del sistema, en el que se desglosan los diferentes niveles de acierto del sistema.

6.1 ACCIONES Y SUS DESCRIPCIONES

El entorno que se ha tomado en consideración ha sido un hogar inteligente. En esta sección se describen cuáles han sido las acciones y los elementos que las realizan. Estos elementos han sido extraídos considerando

diferentes trabajos en los que se describen diferentes escenarios de hogar inteligente [271] [272] [273] [274].

En el entorno implementado, se puede hacer una clasificación de los elementos presentes en dos categorías: aplicaciones y objetos. Esta distinción se hace debido a que, en un entorno como un hogar inteligente, los cambios de estado se realizan sobre objetos físicos (e.g., lámparas o televisiones) y sobre aplicaciones (e.g., redes sociales), que pueden estar instaladas en más de un dispositivo. Por un lado, las acciones de aplicación son acciones realizadas por aplicaciones y, por otro lado, las acciones de objeto son acciones que solo puede llevar a cabo ese objeto. En la tabla 5 se recogen las acciones de aplicación y en la tabla 6 las de objeto. En la primera columna se puede ver el nombre de las diferentes aplicaciones y de los objetos. La segunda columna muestra el nombre de las acciones que pueden llevar a cabo, mientras que la tercera recoge sus descripciones.

6.2 VIRTUALIZACIÓN DE LAS ACCIONES

Como se ha indicado en el plan de evaluación, se ha implementado un modelo, de actores en el que cada entidad presente se conoce como un *actor*. Se trata de un modelo para computación concurrente [270]. Un actor es una unidad fundamental de computación. Las únicas tareas que puede llevar a cabo son: 1) crear otro actor, 2) enviar un mensaje o 3) indicar cómo reaccionar a un mensaje entrante. Los actores guardan información privada sobre su estado, que cambiará en función de los mensajes que reciba, siendo ésta la única forma de cambiarla ya que, además, están aislados en memoria, de forma que no comparten información, resultando este modelo ideal para la implementación de la prueba de concepto descrita en el presente capítulo. Los mensajes que reciben se procesan como una cola FIFO y son asíncronos. Este trabajo se ha decantado por el uso de Pykka, una implementación en Python de este modelo.

Los actores presentes en la simulación son:

- **Los objetos y aplicaciones.** Con la idea de simular las acciones y los objetos/aplicaciones que las ejecutan de la sección anterior, sección 6.1, se ha establecido que cada objeto y cada aplicación sea un actor diferente. Su estado cambia en función de los mensajes recibidos por parte del orquestador, el cual le envía un mensaje indicando la acción que se tiene que ejecutar.
- **El intérprete.** Para simular el comportamiento del intérprete, este también se ha incluido en el modelo como un actor. El intérprete

App name	Action name	Action description
Calendar	Consult meeting	Consults if there is any appointment
	Cancel meeting	Cancels a specific appointment
	Add meeting	Adds an appointment to the calendar
WhatsApp	Read whatsapp	Reads the last unread whatsapp message
	Send whatsapp	Sends a whatsapp message
Facebook app	Post comment	adds a post to the user wall
	Send message to a friend	Sends a Messenger message to a friend
	Log in	Logs in in facebook app
	Share publication	Shares a specific publication
	Delete comment	Deletes a specific comment
	Log out	Logs out from facebook
Spotify	Restart song	Restarts the current song
	Search song	Searchs a song by title
	Pause song	Pauses the current song
	Play song	Plays a song by title
Twitter	Read tweet notification	Reads the last notification
	New tweet	Writes a new tweet
Youtube App	Search video	searchs a video by title
	Comment video	comments the current video
	Volume down	Turns volumne down
	Volume up	Turns volume up
	Mute video	Mutes the current video
	Pause video	Pauses the current video

Tabla 5: Acciones de aplicación.

también se ha implementado en lenguaje Python. Como ya se ha descrito, es el encargado de analizar las órdenes dadas por los usuarios, así como de analizar las descripciones de las acciones recibidas por el orquestador. Con la idea de automatizar la puesta a prueba del sistema, las órdenes recogidas y recopiladas en un archivo de texto al cual tendrá acceso el intérprete. Además, también con la finalidad de que dicha puesta a prueba sea eficiente, las medidas tomadas sobre el acierto del sistema las llevará a cabo el propio intérprete. La herramienta utilizada, para analizar las órdenes originales, a fin de obtener las diferentes órdenes primitivas, se ha sido la librería "stanza" de Python, mientras que, para llevar a cabo las consultas a la herramienta WordNet a la hora de administrar los diferentes clústeres, se ha utilizado la librería "nltk" de Python.

Resource	Action name	Action description
Cell phone	Make call	Makes a call to a specific contact
	Send SMS	Sends a SMS to a specific contact
	Answer call	Answers the incoming call
Clock	Switch off the alarm	Switches off the ringing alarm
	Set alarm	Sets an alarm at a specified time
	Read hour	Returns the current time
Computer	Send email	Sends an email to a specific contact
	Search in google	Does a Google search
Crystal Lamp	Switch on	Switches on the crystal lamp
	Switch off	Switches off the crystal lamp
Ikea Lamp	Switch on	Switches on the Ikea lamp
	Switch off	Switches off the Ikea lamp
Radio	Change dial	Changes Radio dial to a specific frecuency
	Switch on	Switches on the radio
	Switch off	Switches off the radio
Television LG 405	Change channel	Changes the current tv channel
	Switch on	Switches on the television
	Switch off	Switches off the television

Tabla 6: Acciones de objeto.

- **El orquestador**. Se ha creado un actor que representa al orquestador, el cual se comunica, por un lado, con los diferentes actores que representan a los objetos y a las aplicaciones y, por otro, con el intérprete. Será el encargado de dos tareas. La primera consiste en recoger las descripciones de las acciones del entorno para enviárselas al intérprete. La segunda es recibir la acción que el intérprete deduce que el usuario quiere llevar a cabo para comunicar al objeto que realiza esa acción que tiene que efectuarla.

Para crear el escenario, se ha codificado un programa en Python que realiza los siguientes pasos:

1. Crear al intérprete.
2. Crear al orquestador.
3. Crear a los diferentes elementos, aplicaciones y objetos, presentes. Cada uno de ellos tiene acceso a su descripción, un archivo en notación OWL que sigue las ontologías descritas en el capítulo anterior, sección 5.3. Una vez creado, envía estas descripciones al orquestador quien, a su vez, las envía al intérprete.

En el plan de pruebas desglosado en la el capítulo anterior, sección 5.5, se considera que las órdenes que se suministran al intérprete de órdenes deben provenir de aportaciones de usuarios reales. Para recopilarlas, se pidió a un grupo de personas que rellenaran un formulario. En éste, se exponían diferentes situaciones en las que el usuario deseaba realizar cierta tarea. El número total de usuarios que ayudaron a recolectar estos términos fue 51. Los usuarios no responden a un perfil concreto: entre ellos hay personas de diferentes áreas, niveles socioeconómicos y edades. En total, se recogieron 2162 frases, habiendo introducido algún usuario más de una por situación. La tabla 7 recoge el formulario planteado para las acciones de aplicación descritas en la tabla 5. La tabla 8 muestra el formulario planteado para las acciones de objeto descritas en la tabla 6.

Con la idea de automatizar el análisis de éxito del sistema, las diferentes órdenes recogidas son administradas al intérprete por bloques, es decir, por cada una de las acciones posibles, el intérprete analiza las frases de los usuarios que se sabe deben activar esa acción. Este análisis se lleva a cabo con diferentes valores de los parámetros coeficiente de Jaccard e incremento de verbo descritos en la sección 5.4 del capítulo anterior. Cuando el intérprete concluye qué acción debe ser ejecutada por cada una de esas órdenes de usuario, se realiza un análisis de su precisión. Los resultados se presentan y se comentan en la siguiente sección, sección 6.3.

6.3 RESULTADOS

Para poner a prueba el sistema, se han considerado diferentes valores de los parámetros definidos en el capítulo anterior, sección 5.4: incremento de verbo, IV, y coeficiente de Jaccard mínimo, CJ_{min}. En particular, se han utilizado los siguientes, con pasos del 5 %:

$$50\,\% \leqslant IV \leqslant 100\,\% \tag{3}$$

$$0\,\% \leqslant CJ_{min} \leqslant 100\,\% \tag{4}$$

Para cada pareja de estos valores, el intérprete establece cuáles son las 5 acciones, ordenadas por coeficiente de Jaccard, que podrían responder a la orden del usuario. Para evaluar su eficiencia, se ha aplicado el criterio que se describe a continuación:

1. Si la acción que se sabe que es correcta es devuelta por el intérprete en primera posición, y su coeficiente de Jaccard es mayor o igual que el mínimo, se valora con un 5 sobre 5, considerándose un ajuste perfecto.

App name	Action name	Action description
Calendar	Consult meeting	Quieres saber qué citas tienes para hoy
	Cancel meeting	Quieres cancelar la cita del sábado a la 1 de la tarde
	Add meeting	Quieres añadir una cita para el jueves a las 10 de la mañana
WhatsApp	Read whatsapp	Tienes una notificación de WhatsApp y quieres leer el mensaje
	Send whatsapp	Has quedado con tus amigos y quieres escribirles un whatsapp para decirles que llegas tarde
Facebook app	Post comment	Quieres publicar tu sesión de gimnasio
	Send message to a friend	Quieres escribir a Ana un mensaje a través de FB
	Log in	Quieres entrar en FB
	Share publication	Quieres compartir el vídeo que estás viendo en Youtube
	Delete comment	Quieres borrar tu último comentario
	Log out	Quieres salir de FB
Spotify	Restart song	Quieres reiniciar la canción que estás escuchando
	Search song	Quieres buscar una canción de Navidad
	Pause song	Quieres pausar la música
	Play song	Quieres reproducir el tema "mi canción"
Twitter	Read tweet notification	Te acaba de sonar una notificación de Twitter, ¿cómo la lees?
	New tweet	Quieres twittear el texto "esto es un texto"
Youtube App	Search video	Quieres buscar un vídeo de gatitos
	Comment video	Quieres comentar el vídeo que estás viendo
	Volume down	Quieres bajar el volumen del vídeo que estás viendo
	Volume up	Quieres subir el volumen del vídeo que estás viendo
	Mute video	Quieres quitar el sonido al vídeo que estás viendo
	Pause video	Quieres pausar el vídeo que estás viendo

Tabla 7: Formulario para las acciones de aplicación.

2. Si la acción que se sabe que es correcta es devuelta por el intérprete en primera posición, y su coeficiente de Jaccard es menor que el mínimo, se valora con un 4 sobre 5, considerándose un ajuste casi perfecto.

3. Si la acción que se sabe que es correcta es devuelta por el intérprete en una posición diferente a la primera, se valora en orden decreciente, esto es, una acción en segunda posición se valora con un 4, una en tercer lugar con un 3, y así seguidamente.

4. Si la acción correcta no se encuentra en lo devuelto por el intérprete se evalúa con 0.

5. Además de lo anterior, si la acción correcta empata con otras candidatas, se le resta una unidad a este valor, no pudiendo valer menos de 0, independientemente de la posición en la que esté.

Además, como se ha descrito en el plan de pruebas, el intérprete ha sido implementado y evaluado de dos maneras diferentes: sin aprendizaje y con aprendizaje. Este aprendizaje consiste en realimentar al sistema en función de su seguridad. El intérprete se considera seguro de haber deducido la

Resource	Action name	Action description
Cell phone	Make call	Quieres llamar a tu madre
	Send SMS	Quieres escribir un SMS a tu abuela
	Answer call	Te llaman al móvil y quieres responder
Clock	Switch off the alarm	Está sonando la alarma y quieres quitarla
	Set alarm	Quieres poner la alarma a las 7 de la mañana
	Read hour	Quieres saber qué hora es
Computer	Send email	Quieres enviar un email a Daniel
	Search in google	Quieres buscar información sobre Nepal
Crystal Lamp	Switch on	Quieres encender la lámpara de cristal de tu salón
	Switch off	Quieres apagar la lámpara de cristal de tu salón
Ikea Lamp	Switch on	Quieres encender la lámpara de Ikea que acabas de comprar
	Switch off	Quieres apagar la lámpara de Ikea que acabas de comprar
Radio	Change dial	Quieres poner Rock-FM
	Switch on	Quieres encender la radio
	Switch off	Quieres apagar la radio
Television LG 405	Change channel	Quieres poner el canal 2
	Switch on	Quieres encender la TV
	Switch off	Quieres apagar la TV

Tabla 8: Formulario de acciones de objeto.

acción correcta cuando el coeficiente de Jaccard de la acción en primera posición supera el coeficiente de Jaccard mínimo y no hay otra acción candidata el mismo nivel. En el caso de que la acción que se debe disparar no reúna estas condiciones para considerarla como la acción correcta, el intérprete pide al usuario que, de entre el conjunto de 5 acciones que ha concluido que encajan mejor con la orden del usuario, éste elija la correcta. Hace falta notar aquí que, en esta prueba de concepto, ese aprendizaje se ha automatizado: dado que al evaluarse a sí mismo el intérprete sabe qué acción debería ejecutarse, él mismo se retroalimenta, simulando la interacción con un usuario real.

6.3.1 *Sistema sin aprendizaje*

En la figura 16 se disponen los resultados obtenidos para el sistema sin aprendizaje. En la figura se pueden diferenciar dos tipos de imágenes: un diagrama de cajas (primera fila, primera columna) y una serie de diagramas de barras.

Por una lado, en el diagrama de cajas se ha querido mostrar las evaluaciones de las decisiones tomadas por el intérprete que, como se ha indicado en esta sección, estos valores se encuentran entre 5, ajuste perfecto, y 0, ningún ajuste. Los valores distintos de 0 se han calculado sobre el porcentaje

de acierto. En este caso en particular, un 47.74 % de las decisiones han sido evaluadas con un 0, mientras que el resto, 52.26 %, han sido evaluadas con 1 o más, siendo este valor al que hace referencia el resto de porcentajes: el 76.80 % de las respuestas diferentes de 0 ha sido evaluada con un 5, el 8.81 % con un 4, un 9.09 % con un 3, un 3.31 % con un 2 y un 1.99 % con un 1.

Por otro lado, los diagramas de barras muestran los valores de acierto obtenidos para diferentes valores de CJ_{min} con pasos del 5 %. Como se puede observar, no existe una gran variación para distintos valores de CJ_{min}, aunque se puede apreciar cómo el porcentaje de valores máximos (5 sobre 5) disminuye con valores de CJ_{min} mayores, del mismo modo que es máximo cuando este valor CJ_{min} es mínimo. Esto es esperable ya que a mayor valor del CJ_{min} menor cantidad de ajustes perfectos debe haber, ya que es un límite inferior. Además, cabe comentar que entre las diferentes gráficas, correspondientes a distintos valores de IV, no hay gran diferencia: el perfil es prácticamente idéntico.

6.3.2 *Sistema con aprendizaje*

En la figura 17 se disponen los resultados obtenidos para el sistema con aprendizaje. En la figura se pueden diferenciar dos tipos de imágenes: un diagrama de cajas (primera fila, primera columna) y una serie de diagramas de barras.

Por un lado, en el diagrama de cajas, del mismo modo que en la imagen 16, se muestran los valores distintos de 0 que se han calculado sobre el porcentaje de acierto. En el caso del sistema con aprendizaje, analizando el mismo conjunto de órdenes dadas por los usuarios, la cantidad de decisiones tomadas por el intérprete evaluadas con 0 es del 3.61 %, un porcentaje mucho menor que en el caso sin aprendizaje (47.74 %). Como se puede observar en la imagen, el 71.77 % de las evaluaciones distintas de 0, ha sido evaluada con un 5, es decir, el 71.77 % del 96.39 % de frases evaluadas con una puntuación distinta a 0, mientras que el 16.12 % lo ha sido con un 4, con un 3 el 6.89 %, con un 2 el 3.4 % y con un 1 el 1.81 %.

Por otro lado, los diagramas de barras muestran los valores de acierto obtenidos para diferentes valores de CJ_{min} con pasos del 5 %. En este caso, el perfil para diferentes valores de IV es similar: existe una variación para distintos valores de CJ_{min}, pero ésta se conserva entre los diferentes valores de IV. Como se puede observar, para valores bajos y altos de CJ_{min} el número de aciertos perfectos es más pequeño comparado con los valores intermedios. La explicación que se da a la forma de este perfil es que, por un lado, los valores bajos implican una cantidad de empates entre acciones

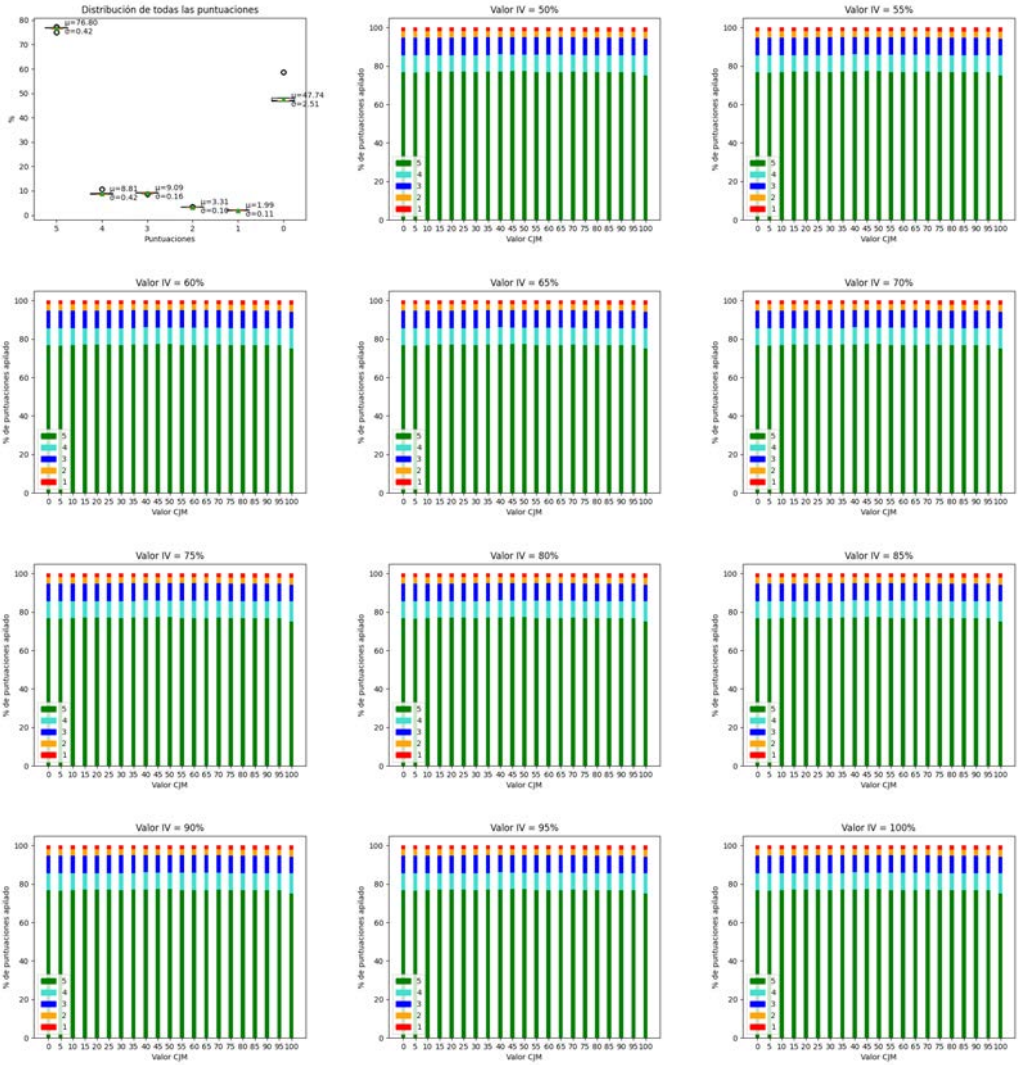


Figura 16: Evaluación del sistema sin aprendizaje

mayor, lo que revierte, según el criterio de evaluación, en un decremento en la cantidad de aciertos perfectos (para que lo sean no puede haber empates entre acciones) y, por otro lado, como es lógico según este criterio, los valores más altos implican menos aciertos porque el sistema estará menos seguro.

6.4 CONCLUSIONES

A lo largo de este capítulo se ha presentado la implementación del intérprete diseñado en el capítulo 5 con la finalidad de hacer una prueba de concepto y demostrar que este diseño es viable. En primer lugar, se ha presentado el entorno, un hogar inteligente, junto con las acciones conside-

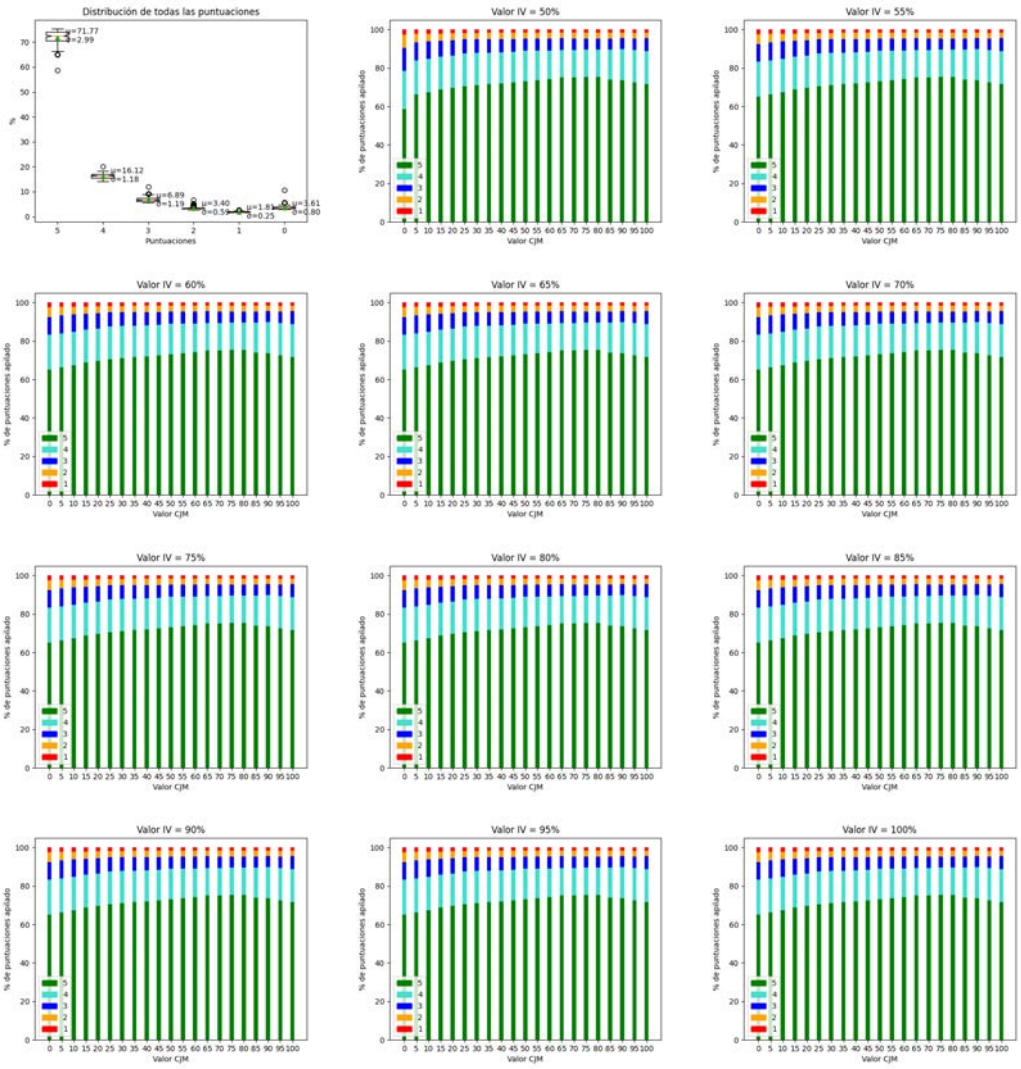


Figura 17: Evaluación del sistema con aprendizaje

radas en él, sección 6.1. A continuación, se han presentado las herramientas utilizadas para crear el entorno donde, como se ha explicado, se ha implementado un modelo de actores, sección 6.2. Como se ha comentado, el sistema se ha puesto a prueba con usuarios reales, habiendo sido recopiladas más de 2000 órdenes dadas en lenguaje natural. Para evaluar el sistema, en la sección 6.3 se ha presentado el criterio empleado, mostrando los resultados de aplicar esta evaluación al sistema sin aprendizaje, figura 16, y con aprendizaje, figura 17.

A día de hoy, el uso del Lenguaje Natural como interfaz de usuario se ha convertido en tendencia como tema de investigación y está presente en muchos de los productos de mercado inteligentes que nos rodean. En este capítulo, se ha mostrado un sistema que, precisamente, usa el proce-

samiento del lenguaje natural para disminuir la complejidad en la interacción humano-dispositivo, y minimiza la distancia que puede haber entre desarrolladores/fabricantes y los usuarios finales. La implementación del intérprete de órdenes descrita en este capítulo ha servido para poder demostrar la viabilidad del diseño. Así mismo, los resultados derivados de su evaluación permiten considerar esta solución como válida a la hora de minimizar dos problemas: el primero, para los desarrolladores, elegir los términos más apropiados y compatibles con el usuario final, y el segundo, para el usuario, el aprendizaje de los términos exactos para dirigirse a un entorno que vaya a recibir sus órdenes. Esto satisface el segundo de los objetivos principales, O2, así como sus cuatro subobjetivos, presentados en la sección 1.2:

- Objetivo O2.1, eliminar el uso de palabras clave para la activación de acciones. El intérprete presentado se basa en el uso de diccionarios y en el análisis de las órdenes dadas en lenguaje natural. Gracias a ello, el usuario no necesita recordar ninguna palabra clave, ni término concreto, que dispare una acción en particular. Como se ha visto en los resultados, los aciertos se concentran en la mayor de las puntuaciones posibles: 5 sobre 5.

- Objetivos O2.2 y O2.3, diseñar un mecanismo para la descripción de acciones que puede llevar a cabo un dispositivo y diseñar el mecanismo de intercambio de datos entre dispositivos e intérprete. El mecanismo diseñado se basa en el uso de una ontología, gracias a la cual el intérprete puede reconocer qué acciones existen en el entorno y cuáles son sus descripciones, tras el análisis de estas descripciones y de los términos que las acciones pueden tener asociados. Cuando se quiere añadir al entorno una acción que se puede ejecutar a través del intérprete, basta con seguir este mecanismo.

- Objetivo O2.4, evitar el entrenamiento del sistema y de los dispositivos. El intérprete implementado no ha requerido de un entrenamiento previo, aunque sí se ha considerado añadir al intérprete la capacidad de aprender de sus decisiones a través de retroalimentación por parte del usuario. Los resultados obtenidos muestran una gran tasa de acierto, sobre todo en el sistema con aprendizaje, con lo que cabe concluir que no hace falta entrenar al sistema, pero sí es necesario que éste incluya un método de aprendizaje.

En el siguiente capítulo se presenta la integración final de todos los elementos presentados en esta tesis. Hasta ahora, esta tesis ha mostrado el trabajo desarrollado divido en dos partes bien diferenciadas: la arquitectura de microservicios, capítulos 3 y 4, así como el intérprete de órdenes

en lenguaje natural, capítulos 5 y 6. En el capítulo 7 se quiere satisfacer el tercer objetivo principal de este trabajo, O3 (sección 1.2), consistente en contribuir a la interoperabilidad continua entre humanos y sistemas automatizados a través del uso del lenguaje natural. Para ello, se llevará a cabo la integración del intérprete dentro de la arquitectura de microservicios. Este trabajo busca proporcionar al usuario la posibilidad de comunicarse con el entorno y pedir que los microservicios migren de un dispositivo origen a otro destino a través de órdenes en lenguaje natural. En la sección 7.1 se presenta el diseño de esta integración. En esta sección, se describe el mecanismo propuesto para generar de forma automática las acciones que permitan al intérprete conocer qué migraciones se pueden hacer y, por tanto, responder a las peticiones del usuario. En la sección 7.2 se describe el plan de pruebas que se quiere llevar a cabo para validar la propuesta. Posteriormente, en la sección 7.3 se exponen los resultados obtenido al implementar el plan de pruebas. Por último, en la sección 7.4 se analizan las conclusiones a las que se ha llegado tras observar los resultados.

7

INTEGRACIÓN: DISEÑO Y VALIDACIÓN

En los capítulos anteriores se han descrito dos propuestas. En primer lugar, capítulos 3 y 4, una arquitectura de microservicios que incluye, como se ha visto, varios elementos en los que un orquestador se puede apoyar con el fin de conocer el estado de los dispositivos que coordina, así como los microservicios que en ellos están desplegados. En segundo lugar, capítulos 5 y 6, se ha descrito el diseño e implementación de un intérprete de órdenes capaz de minimizar la necesidad de aprendizaje por parte del usuario, de forma que pueda dirigirse a un entorno sin el requisito de conocer los términos clave para ello.

Con la idea de integrar ambas propuestas, y con el fin de presentar la idea principal de esta tesis, el presente capítulo describe la metodología seguida para componer el sistema final: una arquitectura de microservicios que permita la interoperabilidad transparente entre humanos y sistemas automatizados a través de un asistente. Este capítulo está dividido en varias partes. En primer lugar, en la sección 7.1, se presenta el diseño de la integración, presentando los elementos que la componen. Esta sección, se busca satisfacer el tercer objetivo principal, O3, describiendo el diseño de un mecanismo que permita a) al orquestador conocer las compatibilidades entre dispositivos y microservicios; b) al intérprete entender una orden en lenguaje natural cuya finalidad sea la migración de un microservicio; y c) establecer la relación entre la orden dada y las migraciones posibles. Seguidamente, en la sección 7.2, se establece el plan de pruebas de dicho diseño, basado en los planes de pruebas de los capítulos anteriores. Por último, en la sección 7.3, se muestran los resultados de evaluar la implementación de esta integración, aplicando el mismo criterio que en el capítulo anterior, capítulo 6. Para llevar a cabo la validación, del mismo modo que la que se ha realizado en el capítulo anterior, el sistema se ha puesto a prueba con usuarios reales.

7.1 INTEGRACIÓN: DISEÑO

La integración del intérprete dentro de la arquitectura de microservicios descrita en el presente trabajo, tiene como finalidad crear un mecanismo que permita al usuario realizar peticiones a su entorno cuya intención sea la de mover un microservicio de un dispositivo a otro. Para implementar

este mecanismo, se han identificado tres tareas principales. En primer lugar, el intérprete debe ser capaz de entender la orden del usuario, identificando qué microservicio tiene intención de mover a qué destino. En segundo lugar, el intérprete debe conocer qué microservicios y dispositivos coexisten en el entorno. Por último, el orquestador debe disparar la migración del microservicio. En esta sección se describe el diseño que permite llevar a cabo estas tareas.

Las entidades descritas en el capítulo 3, sección 3.1.1, permiten conocer, por un lado, las necesidades de los microservicios y, por otro, las capacidades que ofrecen los dispositivos. Como se ha descrito en dicho capítulo, el orquestador se apoya en ambos microservicios para conocer el entorno. Con las descripciones de microservicios y dispositivos, el orquestador es capaz de saber de antemano qué dispositivo tiene la posibilidad de albergar qué microservicio, simplemente, comparando los requisitos exigidos con las capacidades ofrecidas. El mecanismo creado para que el intérprete conozca qué migraciones se pueden llevar a cabo, se basa en esta información. En el diseño e implementación del intérprete llevada a cabo en los capítulos anteriores, se ha visto cómo el orquestador alimenta al intérprete con las descripciones de las acciones que se pueden llevar a cabo (utilizando una ontología). En esta ocasión, el orquestador será el encargado de crear acciones que representen todas las migraciones posibles utilizando la información adquirida gracias a las entidades NS y μSS. Es decir, dados unos microservicios y unos dispositivos, el orquestador establece acciones virtuales que él lleva a cabo en las que se indica qué microservicio se quiere mover a qué dispositivo destino: por cada compatibilidad microservicio-dispositivo existirá una acción diferente. En la sección 7.1.1 se describe este proceso de generación automática de posibles migraciones.

En la integración llevada a cabo se ha tenido en consideración la posibilidad de que el usuario quiera realizar una migración, pero que ésta no se pueda llevar a cabo por el estado de los dispositivos o por la compatibilidad entre microservicio y dispositivo destino. Con la idea de procurar al sistema la posibilidad de entablar un diálogo con el usuario, el sistema debe ser capaz de proponer al usuario alternativas en el momento en el que éste quiera llevar a cabo una migración que no sea posible. Para que el sistema pueda sugerir al usuario otra opción diferente a la petición efectuada, éste hace uso del mecanismo de análisis de rendimiento descrito en el capítulo 3, sección 3.2.

7.1.1 *Generación automática de posibles migraciones*

Para generar las acciones que representan toda las posibles migraciones dentro del entorno, se ha creado un patrón que el orquestador debe seguir para crearlas. Este patrón responde al etiquetado de las acciones que el intérprete analiza para conocer estas acciones que se ha introducido en la sección 5.3.2. En concreto, las acciones generadas tendrán asociados tres tipos de datos: un nombre, una descripción y una serie de tags. Estas tripletas responden al patrón que se muestra en la tabla 9.

Action Name	Action Description	Tags
move “microservice id” from “device id” to “device id”	moves “microservice id” from “deviceA id” to “deviceB id”	“move”, “microservice id”, “deviceA id”, “deviceB id”

Tabla 9: Patrón para las acciones creadas de forma automática que hacen referencia a cada una de las posibles migraciones.

Los componentes y consideraciones de diseño a considerar para la integración del intérprete en la arquitectura de microservicios son los mismos que los presentados en el capítulo 3, sección 3.2.1.

7.2 PLAN DE PRUEBAS

El plan de pruebas llevado a cabo para validar la presente propuesta, se ha basado en el plan de pruebas diseñado en las secciones 3.2.2 y 5.5 de los capítulos anteriores. Sin embargo, existen las siguientes consideraciones:

- El entorno de pruebas simulará un hogar inteligente. De igual manera, y con las mismas herramientas, que en el plan de pruebas del intérprete (sección 5.5), se ha implementado un modelo de actores que simula un hogar inteligente. El número de dispositivos, su naturaleza, y su definición HW, se han determinado a partir de diferentes trabajos en los que se evalúan hogares inteligentes ([275] [276] [277] [278] [279]). En términos generales, los dispositivos tienen como denominador común el poseer alguna o varias, de las siguientes capacidades: mostrar información por pantalla, reproducir sonido, así como tener entrada de voz (micrófono) y de imagen (cámara). En concreto, se han definido 10 dispositivos cuyas capacidades se puede ver en la tabla 10.

- El intérprete, el orquestador, así como el NS y el µSS también serán actores. Cada uno de los dispositivos tendrá asociado un NS y un

μSS, los cuáles intercambiarán datos con el orquestador. Éste será el encargado de crear las acciones referentes a las posibles migraciones, las cuales dará a conocer al intérprete.

- En cuanto a los microservicios, se ha estipulado que cada elemento HW tiene asociado un microservicio que lo utiliza. De esta forma, cuando en la tabla 10 se indica que la Smart TV tiene una pantalla, de forma implícita también se indica que existe un microservicio cuya función es mostrar imágenes a través de la pantalla del dispositivo en el que está instanciado. Es necesario aclarar que solo se han considerado a los microservicios de forma individual. En otras palabras, en este tipo de escenarios, atender a peticiones complejas que impliquen la migración de más de un microservicio a la vez no se ha valorado. Peticiones como "quiero ver la TV en la tablet" requeriría migrar todo el conjunto de microservicios que ofrecen el servicio de TV, es decir, audio e imagen, como mínimo. En la prueba llevada a cabo, las peticiones se hacen para microservicios concretos. La composición de migraciones complejas se deja para líneas futuras de investigación.

- En esta simulación, para que el orquestador sepa si un microservicio puede ser alojado en un dispositivo, solo se ha considerado la información HW sin añadir requisitos de rendimiento, aspecto que también se deja para lineas de investigación futuras.

- Las mediciones que se tomarán, del mismo modo que en el plan de pruebas del intérprete (sección 5.5), considerarán diferentes valores del coeficiente de Jaccard mínimo (CJ_{min}) y del incremento de verbo (IV), aunque considerando exclusivamente una implementación del intérprete con aprendizaje. Como se vio en los resultados del capítulo anterior (seccion 6.3), el rendimiento del intérprete es muy superior cuando es capaz de aprender que cuando no, por lo que sólo se ha tenido en consideración dicha implementación.

- El sistema se probará con usuarios reales. Igual que en el plan de pruebas del intérprete (sección 5.5), el sistema se pondrá a prueba por usuario reales. En concreto, se recogerán órdenes de migración dadas en lenguaje natural. A los usuarios se les presentará un entorno y un objetivo, teniendo que dirigirse al sistema para cumplir ese objetivo. Gracias a la participación de 50 usuarios de diferentes perfiles, se han recopilado 450 órdenes a través de un formulario en el que se planteaban diferentes situaciones. Las situaciones propuestas se recopilan en la tabla 11. Entre los usuarios, existen personas de diferentes edades, nivel educativo y situación económica.

Device Description	Attached HW
My Smart phone	Display, Microphone
My Friend Smart phone	Display, Microphone, Speakers
Smart TV	Display, Speakers
Smart Speakers	Speakers
Smart display	Display, Speaker
Outdoor home security camera	Camera
Display connected to the outdoor home security camera	Display
Laptop	Display, Speakers, Microphone, Camera, Keyboard
Smart fridge	Display, Speakers
Smart projector	Display

Tabla 10: Dispositivos definidos para la prueba de integración.

7.3 RESULTADOS

Para poner a prueba el sistema, se han considerado diferentes valores de los parámetros definidos en el capítulo anterior, sección 5.4, del mismo modo que en el capítulo 6, sección 6.3. Es decir, utilizando los valores expresados en 3 y 4. Además, para evaluar el rendimiento del intérprete se ha aplicado el mismo criterio que el definido en la sección 6.3. Como se ha dicho anteriormente, el sistema se ha evaluado teniendo en cuenta solamente la configuración con aprendizaje. En la figura 18 se muestra el resultado de la evaluación del sistema.

En la figura 18, se puede observar en la primera imagen la distribución de la evaluación del sistema. En este caso, el 19.04 % de las órdenes recogidas no se han entendido, distribuyendo el 80.96 % restante, prácticamente en su totalidad en el valor 5 sobre 5, es decir, entendimiento perfecto, con un 99.51 % . El total de órdenes dadas evaluadas con valores diferentes a 5 no es relevante en comparación. Es decir, se han entendido a la perfección 403 órdenes sobre 500 dadas, una eficiencia del 80.56 % del total. Debido a estos valores, el patrón que se observa en resto de imágenes es muy similar.

7.4 CONCLUSIONES

En este capítulo se ha visto la integración del intérprete presentado en el capítulo 5 dentro de la arquitectura de sistemas introducida en el capítulo

Descripción de la situación
Acabas de llegar a casa y enciendes la TV. Una vez encendida, te pones a cocinar. Sin embargo, quieres que la imagen se muestre en la pantalla inteligente que tienes en la cocina.
Estás teniendo una conversación por teléfono y quieres que la conversación, en vez de oírse por el altavoz del móvil, se oiga por los altavoces del salón.
Estás viendo una vídeo de youtube en tu móvil. En un momento dado decides que quieres ver la imagen directamente a través del proyector (el audio se queda en el móvil).
Estás viendo una vídeo de youtube en tu móvil. En un momento dado decides que quieres oír el audio del vídeo a través de los altavoces inteligentes. (el vídeo se queda en el móvil)
Quieres jugar a un videojuego en tu portátil. Sin embargo, tu equipo no cuenta con un micrófono para que tus colegas te puedan oír, con lo que quieres utilizar el de tu móvil.
En tu casa tienes instalada una webcam en la puerta de la entrada, por fuera. Además, por dentro, tienes una pantalla donde se muestra lo que se ve a través de la webcam. Llaman a la puerta pero, en vez de ir hasta la pantalla, quieres ver lo que muestra la pantalla en tu móvil.
Tu frigorífico es capaz de mostrarte en una pantalla que tiene incorporada los artículos que contiene y sugerirte si quieres hacer la compra online. Sin embargo, se ha estropeado y no responde cuando pulsas sobre la pantalla con tus dedos, con lo que quieres que se muestre la información en tu móvil, ya que su funcionalidad táctil sí está operativa.
Estás escuchando música por los altavoces inteligentes. Sin embargo, estás a punto de salir de casa, con lo que quieres que el sonido ya no salga por los altavoces inteligentes sino que lo reproduzca tu móvil.
Estás trabajando en tu portátil pero quieres que tu pantalla se muestre en el televisor en vez de en tu ordenador para poder trabajar con mayor comodidad.

Tabla 11: Formulario de acciones y sus correspondientes descripciones

3, en este caso, simulado con un modelo de actores. En concreto, en la sección 7.1 se ha propuesto un patrón bajo el cual se puede generar, de forma automática, una serie de acciones que representan todas las posibles

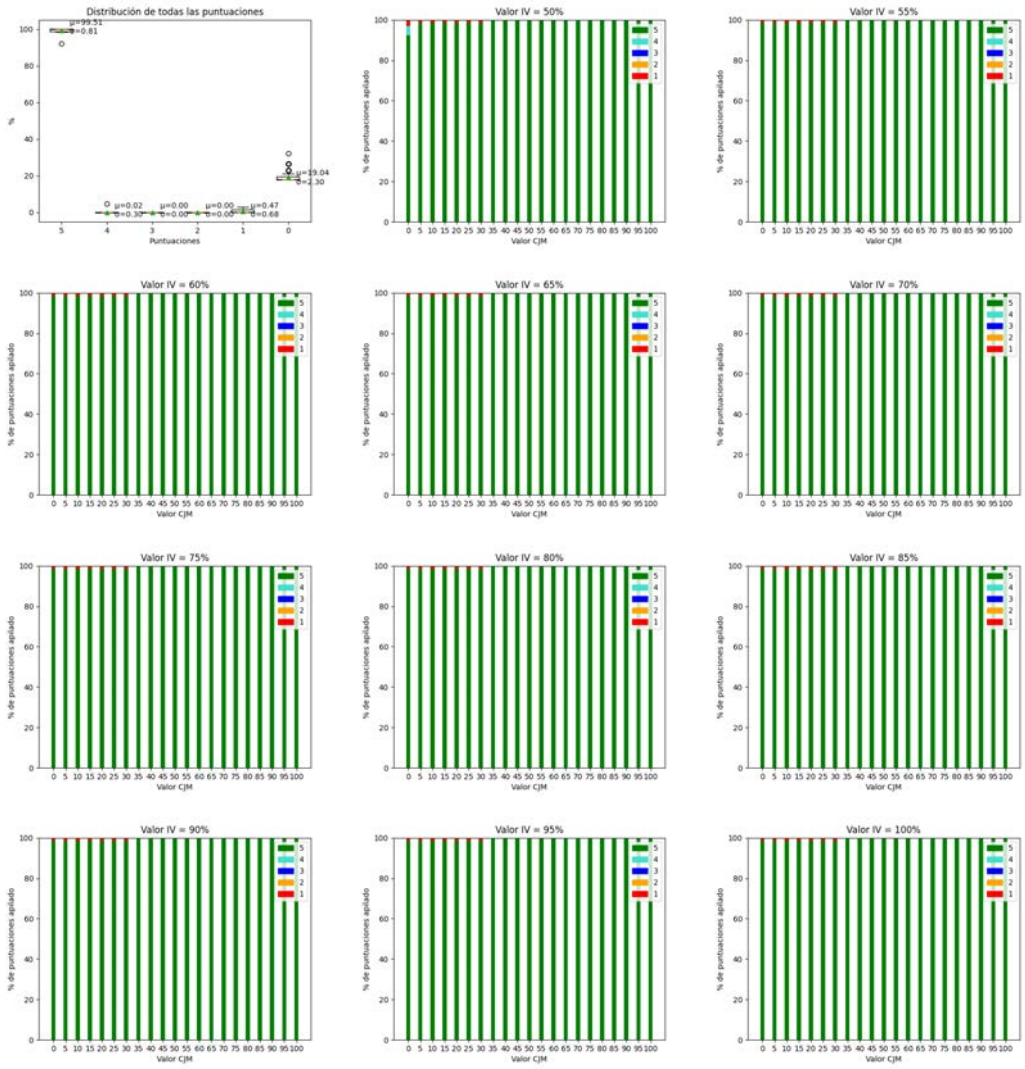


Figura 18: Evaluación del sistema integrado con aprendizaje

migraciones que se pueden llevar a cabo en el entorno. De esta manera, se satisfacen los subobjetivos del tercer objetivo principal desglosado en la sección 1.2. A saber:

- Objetivo O3.1. Gracias al uso de las entidades descritas en el capítulo 3, el orquestador es capaz de conocer las compatibilidades entre microservicios y dispositivos, pudiendo llevar a cabo dicha generación, creando acciones que permitan llevar a cabo las migraciones.
- Objetivos O3.2 y O3.3. Gracias al diseño del intérprete descrito en el capítulo 5, y considerando el patrón presentado en el presente capítulo, el intérprete es capaz de extraer la información necesaria para identificar cuál es la migración que el usuario quiere llevar a cabo.

Dado que las acciones que el orquestador le ha suministrado representan las posibles migraciones, el intérprete puede hacer las comparaciones necesarias entre clústeres de tags para concluir qué acción, es decir, qué migración, se debe realizar. Del mismo modo que en la implementación del intérprete, tampoco ha sido necesario entrenar al sistema ya que, a través de consultas a diferentes diccionarios, es capaz de entender una gran cantidad de órdenes, como se ha visto en los resultado de la prueba de concepto llevada a cabo, sección 7.3.

En el siguiente, y último, capítulo se hace una análisis detallado del cumplimiento de los objetivos, yendo objetivo a objetivo, identificando y discutiendo qué contribuciones se han hecho a lo largo de esta tesis, sección 8.1. Seguidamente, en la sección 8.2, se desglosan las diferentes conclusiones a las que se ha llegado a lo largo del trabajo descrito en esta tesis. Así mismo, al final de dicho capítulo, se hacen diferentes propuestas sobre futuras líneas de investigación que pueden continuar este trabajo, sección 8.3.

8

DISCUSIÓN, CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN

La computación ubicua es una realidad: cada vez existe una mayor cantidad de dispositivos con capacidad de cómputo que nos rodean y con los que el ser humano interactúa. Las economías de escala han permitido abaratar los costes de producción del hardware programable, permitiendo una gran variedad de tipo de dispositivos tanto desde el punto de vista de capacidad computacional, como de la cantidad de memoria, tamaño o finalidad.

El paradigma que propone el Internet de las Cosas se basa en que una gran cantidad de dispositivos inteligentes están conectados a través de una red global y una infraestructura dinámica, permitiendo la aparición de escenarios para la computación ubicua. Sin embargo, estos dispositivos se caracterizan por estar limitados en términos computacionales, siendo susceptibles al fallo o a no poder dar una calidad de servicio suficiente.

Con la idea de subsanar este problema, surge el concepto de computación en la nube, la cual proporciona una infraestructura para la computación masiva, así como una enorme capacidad de almacenamiento. No obstante, dada la cantidad de datos que se generan dentro del Internet de las Cosas y, en consecuencia, la enorme dedicación de recursos necesaria para procesar estos datos, aparecen otros paradigmas computacionales que permiten llevar a cabo parte del procesado fuera de la nube. Estos paradigmas son principalmente dos: el Fog Computing, donde los dispositivos intermedios entre la nube y los finales llevan a cabo tareas computacionales, y el Mist Computing, paradigma en el que son los dispositivos finales los que forman una infraestructura con los elementos en el borde de la red donde, también, poder llevar a cabo parte de la computación.

Esta solución de llevar tareas computacionales al borde de la red cuenta con el inconveniente de que los dispositivos que lo forman están limitados. De esta manera, aparece como solución para su administración un mecanismo para poder llevar a cabo una computación distribuida, consistente en dividir servicios complejos en microservicios dedicados a tareas pequeñas. Estos microservicios se instancian en dispositivos más limitados que los que se pueden encontrar en la nube.

Dado que los dispositivos finales son, en su mayoría, dispositivos utilizados por usuarios reales, seres humanos que interactúan con un entorno compuesto por una gran cantidad de estos dispositivos, este trabajo a querido contribuir en la mejora de esta interacción. Esta tesis propone una serie de pautas y mecanismos gracias a los cuáles un usuario pueda pedir al entorno la migración de microservicios de un dispositivo a otro a través del lenguaje natural.

8.1 PRINCIPALES CONTRIBUCIONES

Contrastando el resultado de la investigación con los objetivos planteados a su inicio, se puede afirmar que se han conseguido alcanzar dichos objetivos de forma satisfactoria. Las contribuciones de esta tesis consisten en propuestas y validaciones: se ha planteado una arquitectura de microservicios y los elementos y entidades que permiten el reconocimiento del entorno; se ha propuesto un mecanismo bajo esta arquitectura que posibilita la creación de pruebas efímeras de rendimiento para los microservicios; validación de la arquitectura y del mecanismo de creación de pruebas efímeras; se ha planteado un mecanismo con el que representar el entorno de forma que un usuario sea capaz de interactuar con él a través del lenguaje natural; validación del mecanismo de descripción haciendo uso de un entorno simulado el cuál es probado por usuarios reales; se ha llevado a cabo una integración del mecanismo de descripción dentro de la arquitectura de microservicios descrita, permitiendo al usuario llevar a cabo migraciones de microservicios a través del uso del lenguaje natural; validación de dicha integración a través de una simulación con usuarios reales.

A continuación, se describen estas contribuciones enumeradas en el párrafo anterior:

1. **Propuesta de una arquitectura de microservicios y los elementos y entidades que permiten el reconocimiento del entorno, así como de un mecanismo bajo dicha arquitectura que posibilita la creación de pruebas efímeras de rendimiento para los microservicios.**

   En el capítulo 2, secciones 2.3 a 2.5 se mostró el papel de los dispositivos finales dentro de los paradigmas de computación distribuida conocidos como Fog y Mist Computing. En el caso particular del Mist Computing, los dispositivos finales llevan a cabo tareas computacionales pequeñas. Como se comenta en dichas secciones, una arquitectura de microservicios resulta muy atractiva en este tipo de escenarios. Con la idea de poder coordinar a los microservicios, aparece la necesi-

dad de tener, dentro del entorno computacional, algún elemento que orqueste el despliegue de los mismos, entre otras tareas.

Para facilitar la tarea principal de orquestación a dicho orquestador, en el capítulo 3 se ha propuesto, descrito y formalizado, una serie de directrices que permiten crear un mecanismo basado en el diseño de dos entidades. Para ello, en dicho capítulo:

- Se describe el problema en detalle, identificando las necesidades específicas a cubrir del orquestador en una arquitectura de microservicios.
- Se proponen dos entidades identificando las tareas que un orquestador lleva a cabo que se pueden ver beneficiadas por dichas entidades.
- Se establece la funcionalidad de ambas entidades, identificando los requisitos que deben cumplir.
- Se propone un mecanismo de análisis de rendimiento de microservicios, capaz de desplegar entornos efímeros de pruebas en los diferentes dispositivos candidatos a ser el destino final de un microservicio que necesita ser migrado.
- Se describen los componentes que forman parte de la arquitectura de microservicios donde se puede implementar este mecanismo de análisis de rendimiento.
- Se propone un plan de pruebas para la evaluación de dicho mecanismo de análisis de rendimiento.

A continuación, se valoran los resultados obtenidos frente a los objetivos planteados en la sección 1.2:

- Objetivo 01.1, proporcionar un mecanismo para la descripción de microservicios y de los dispositivos que coexisten en un entorno. Las tareas principales de las dos entidades presentadas en el capítulo 3 son, por un lado, extraer información del dispositivo en el que se encuentran, tarea que lleva a cabo el Node Summarizer y, por otro lado, recopilar información sobre los microservicios desplegados en el propio dispositivo.
- Objetivo 01.2, proporcionar una metodología para el análisis de compatibilidades entre las descripciones de los dispositivos y de los microservicios. Ambas entidades recogen los datos en forma de descripción que, posteriormente, sirve para comparar las capacidades de los dispositivos con los requerimientos de los microservicios.

- Objetivo 01.3, diseñar un mecanismo que permita la evaluación del rendimiento de los diferentes microservicios en diferentes dispositivos antes de ser desplegados. Este mecanismo:
  - se despliega de forma efímera, llevando a cabo la evaluación de los microservicios en poco tiempo;
  - interfiere de forma mínima en el comportamiento normal de los dispositivos en los que se llevan a cabo dichas evaluaciones, evitando su interrupción;
  - es compatible con otros mecanismos de decisión: este mecanismo proporciona al orquestador una información extra que puede utilizar para tomar decisiones.

2. **Validación de la arquitectura y del mecanismo de creación de pruebas efímeras.**

   Las entidades presentadas, así como el mecanismo de análisis de rendimiento de los microservicios, se han validado mediante simulaciones en Python. El capítulo 4 recoge las pruebas llevadas a cabo que demuestran la viabilidad del mecanismo.

   La prueba de concepto llevada a cabo se ha basado en el uso de contenedores Docker, así como la herramienta MockServer y el protocolo MQTT. Los resultados obtenidos en las métricas efectuadas prueban que este diseño es asequible y su implementación razonable, ya que tiene un consumo de requisitos mínimo, evita la interrupción de los microservicios ya presentes y siendo una fuente de datos extra para la toma de decisiones.

3. **Propuesta de un mecanismo con el que representar el entorno de forma que un usuario sea capaz de interactuar con él a través del lenguaje natural.**

   De acuerdo con los objetivos planteados en la sección 1.2, en el capítulo 5, se propone un sistema que permite al usuario dirigirse a su entorno con la intención de dar órdenes al mismo en lenguaje natural. Los problemas de este tipo de sistemas se desglosan en la sección 2.6 y el diseño llevado a cabo para solventarlos se define en la sección 5.2. Estos problemas son, a grandes rasgos:

   - Conocimiento de palabras clave. En soluciones actuales, cuando un usuario quiere interactuar con su entorno, es necesario que éste conozca los términos que activan cada una de las diferentes acciones que se pueden llevar a cabo. Eliminar esta necesidad es el objetivo principal del sistema diseñado.

- Describir el entorno. Las acciones que se pueden llevar a cabo dentro de un entorno deben ser entendibles por el sistema para que, dada una orden de usuario, comprenda a cuál de las diferentes acciones se refiere. Esto implica, necesariamente, proponer una metodología que permita describir las diferentes acciones por parte de los desarrolladores de los dispositivos que las llevan a cabo.

La solución propuesta en el capítulo 5 cubre estas necesidades. En él se propone un sistema basado en la consulta de diccionarios que elimina la necesidad por parte del usuario de estudiar los términos clave asociados a las diferentes acciones. Además, para llevar a cabo la descripción del entorno se propone un método de descripción de los elementos presentes en el entorno, así como de las acciones que llevan a cabo. Basándose en estas descripciones, así como en la administración de las respuestas obtenidas tras las consultas a los diferentes diccionario, el sistema permite establecer la intención del usuario dentro del entorno, concluyendo qué acción desea disparar. Así mismo, se propone un plan de pruebas para la validación de dicho sistema llevada a cabo en el capítulo 6.

4. **Validación del mecanismo de descripción haciendo uso de un entorno simulado el cuál es probado por usuarios reales.**

   La validación del intérprete se ha llevado a cabo mediante una simulación en Python. El capítulo 6 recoge esta simulación que demuestra la funcionalidad del mecanismo. En dicha simulación se muestra un entorno hipotético donde diferentes elementos capaces de llevar a cabo distintas acciones coexisten. Este entorno simulado se ha puesto a prueba con usuarios reales, los cuales han aportado más de 2000 órdenes que han sido procesadas por el intérprete.

Atendiendo a los resultados obtenidos, frente a los objetivos planteados en la sección 1.2, se demuestra la viabilidad del intérprete:

- Objetivo O2.1, eliminar el uso de palabras clave. Los resultados expuestos en la sección 6.3 demuestran cómo las órdenes dadas por los diferentes usuarios, que responden a perfiles diferentes a nivel socio-cultural, son interpretadas por el intérprete con éxito, validando el diseño del intérprete del capítulo 5.
- Objetivo O2.2, diseño de un mecanismo para la descripción de acciones y objetivo O2.3, diseño de un mecanismo para el intercambio de datos entre dispositivos y el intérprete. El método propuesto para la descripción de acciones se basa en el uso de

ontologías, permitiendo al intérprete su análisis con el fin de llevar a cabo su tarea principal: asociar los términos de las órdenes dadas con los términos que corresponden a cada una de dichas acciones.

- Objetivo O2.4, eliminar la necesidad de entrenamiento del sistema y de los dispositivos. Dado que el sistema realiza consultas a diccionarios externos, el sistema no es necesario un entrenamiento previo eliminando la necesidad de dedicar un tiempo de preparación extra para un correcto funcionamiento.

5. **Planteamiento de la integración del mecanismo de descripción dentro de la arquitectura de microservicios descrita y validación de la misma a través de una simulación con usuarios reales.**

   En el capítulo 7 se presenta la integración del intérprete dentro de la arquitectura de microservicios. Esta integración se ha simulado con Python y puesto a prueba con usuarios reales.

   En dicho capítulo se propone un mecanismo para la creación automática de acciones relacionadas con la migración de microservicios, el cual permite al intérprete conocer la compatibilidad entre dispositivos y microservicios en forma de acciones virtuales. En dicho capítulo, se establece un plan de pruebas, sección 7.2, cuyos resultados, sección 7.3, muestran cómo el intérprete de órdenes es capaz de establecer qué migración se quiere llevar a cabo por parte del usuario. El sistema se ha puesto a prueba con 500 órdenes dadas por usuarios con diferentes perfiles socio-culturales. Los resultados muestran cómo esta integración evita la necesidad, por parte del usuario, de conocer palabras clave asociadas a ninguna migración, del mismo modo que el sistema tampoco necesita un entrenamiento previo.

   De esta manera, el sistema propuesto cubre el tercero de los objetivos principales desglosados en la sección 1.2:

   - Objetivo O3.1: gracias a las entidades descritas en el presente trabajo que forman parte de la arquitectura de microservicios, el orquestador que forme parte ella es capaz de conocer las diferentes compatibilidades que hay entre microservicios y dispositivos, permitiendo la generación automática de acciones virtuales que representan dichas migraciones.
   - Objetivo O3.2: dado que las acciones virtuales asociadas a las migraciones siguen el diseño descrito en la ontología presentada

en el capítulo 5, el intérprete es capaz de manejar los términos asociados a las mismas.

- Objetivo 03.3: atendiendo a estas descripciones y a la extracción de términos clave de las órdenes de los usuarios, el intérprete, del mismo modo que en el diseño presentado en el capítulo 4, consigue establecer, sin necesidad de entrenamiento, qué acción (en este caso qué migración) quiere llevar a cabo el usuario.

8.2 CONCLUSIONES

Una vez identificadas las contribuciones principales de esta tesis, se puede llevar a las siguientes conclusiones:

- En los entornos de computación distribuida donde los dispositivos finales llevan a cabo tareas computacionales conjuntas, en forma de microservicios, como en la Mist Computing, es necesario tener un elemento que orqueste la asignación de dichos microservicios entre los diferentes dispositivos.

- Actualmente, uno de las formatos más extendido en el despliegue de microservicios es el contenedor: un contenedor alberga a un microservicio y cada dispositivo cuenta con un gestor de contenedores encargado de instanciarlos.

- Los orquestadores presentes en los entornos computacionales basados en contenedores de microservicios cuentan con un elemento orquestador encargado de conocer el estado tanto de los microservicios como de los dispositivos que los albergan. En esta investigación se proponen dos entidades que benefician al orquestador en estas tareas, simplificando su codificación y desarrollo.

- El orquestador de este tipo de entornos asigna e instancia a los diferentes microservicios en distintos dispositivos. Para ello, debe conocer el estado de los dispositivos, así como las necesidades de los microservicios. No obstante, el uso de modelos que predicen el comportamiento de los mismos ayudan al orquestador a tomar decisiones sobre en qué dispositivo instanciar qué microservicio. Sin embargo, éstos requieren unas capacidades de cómputo que entornos como la Mist no pueden ofrecer, al mismo tiempo que este tipo de entornos son especialmente dinámicos, cambiando su estado con frecuencia suficiente para que los cálculos hechos en un instante de tiempo dejen de ser representativos del estado del entorno. Esto lleva a la necesidad de diseñar mecanismos que ofrezcan al orquestador más datos

en los que basarse para llevar a cabo sus tareas que no impliquen un alto consumo de recursos y cuya obtención se realice en un periodo de tiempo asumible.

- Llevar a cabo pruebas efímeras de rendimiento a los microservicios en los diferentes dispositivos donde puede ser albergado ayuda a obtener estos datos extra. En los capítulos 3 y 4 se propone un mecanismo para la obtención de dichos datos que tiene las siguientes características:
  - El tiempo de despliegue es mínimo.
  - Permite obtener información de contexto.
  - No interfiere en el comportamiento normal de los microservicios ya desplegados y que no están siendo evaluados.
- Las simulaciones llevadas a cabo muestran la capacidad del mecanismo de alimentar a un orquestador con información en tiempo real del posible rendimiento de los microservicios antes de ser instanciados en un dispositivo final.
- En cuanto a los sistemas que permiten a los usuarios hacer peticiones a su entorno en lenguaje natural, actualmente mucha soluciones se basan en el uso de palabras y términos clave para disparar acciones. Esto implica dos problemas: por un lado, el usuario debe conocer estos términos y, por otro lado, el equipo de desarrollo debe seleccionar un conjunto de términos para cada acción que sea compatible con la mayor cantidad de usuarios potenciales posible.

  Con la idea de solventar estos dos problemas, se propone un mecanismo basado en el uso de diccionarios para poder ampliar el conjunto de términos relacionados con las acciones del entorno. El intérprete que lleva a cabo esta tarea, presentado en los capítulos 5 y 6, cumple con este cometido dado que posee las siguientes características:
  - Permite analizar la intención de las órdenes dadas en lenguaje natural, extrayendo los términos más relevantes.
  - Es capaz de establecer un administrar conjuntos de términos relacionados en función de su papel sintáctico (nombre, verbo, adverbio o adjetivo).
  - Conoce la información de contexto gracias a que se comunica con el orquestador del entorno y a la definición de las acciones.
  - Es independiente del lenguaje de programación. En el presente trabajo se han establecido una serie de funcionalidades que debe

cumplir siendo implementables en cualquier lenguaje de programación.

8.3 LÍNEAS FUTURAS

Esta tesis propone nuevos mecanismos para la mejora de la interoperabilidad continua entre humanos y sistema automatizados bajo el paradigma del Internet de las Cosas. No obstante, quedan muchas mejoras y precisiones que hacer a las propuestas planteadas en este trabajo, a la vez que se abren líneas de investigación futuras que permitan que estos mecanismos sean una realidad. Durante la realización de esta tesis se han identificado algunas mejoras y líneas de trabajo futuro que se resumen a continuación:

1. **Estudio y definición de un mecanismo de extracción de información de contexto:** En el capítulo 3 se proponen dos entidades que permiten identificar y caracterizar el entorno que nos rodea. Para que esta caracterización sea lo más completa posible resultaría favorable recurrir a los datos estadísticos que las plataformas de despliegue de contenedores ya tienen desarrollados.

2. **Estudio, simulación y validación de implementaciones alternativas:** En el capítulo 4 se lleva a cabo una implementación, a modo de prueba de concepto, de las dos entidades presentadas en el capítulo 3. Resultaría interesante implementar distintas versiones de éstas para diferentes sistemas operativos o para sistemas más limitados, como es el caso de los wearables.

3. **Estudio y definición de un mecanismo para evaluación de rendimiento de los microservicios:** En la sección 3.2 se ha propuesto un mecanismo capaz de desplegar entornos de prueba efímeros gracias a los cuales se puede poner a prueba un microservicio en un dispositivos destino antes de desplegarlo de forma definitiva. Sería interesante estudiar diferentes implementaciones de las versiones reducidas de los microservicios y realizar comparativas entre éstas, como cuál es el impacto sobre el funcionamiento normal de los microservicios, o cuál es el efecto sobre los recursos de los dispositivos.

4. **Estudio y comparación con modelos:** También sería positivo realizar una comparación con diferentes modelos actuales, como los vistos en el estado del arte, sección 2.5.4 comprobando en qué tipo de situaciones conviene o no implementar la solución presentada en este trabajo.

5. **Análisis del mecanismo de pruebas de rendimiento en diferentes escenarios:** Respecto a la definición de los tests, resultaría provechoso evaluar cómo de precisos son. En concreto, concluir en qué tipo de situaciones tomar decisiones basadas en ese tipo de tests resulta una ventaja o si no merece la pena llevar a cabo estos tests ya que, por ejemplo, el entorno pudiera ser extremadamente cambiante.

6. **Extensión de los requisitos de los microservicios:** Además, sería útil incluir definiciones sobre requisitos de rendimiento en los requerimientos de los microservicios. De esta manera, si las entidades que hacen uso de un microservicio lo desean, pueden añadir reportes sobre el rendimiento de lo mismos, de forma que las descripciones originales pueden ser modificadas por los desarrolladores. Así, cada microservicio tendrá asociado un perfil con datos que indiquen cómo rinden en según qué circunstancias.

7. **Estudio y validación de implementaciones reales del intérprete:** Desde el punto de vista de la filosofía DIY, si se incluyese esta capa de funcionalidad a soluciones que ya existen en el mercado (como los asistentes personales), se podrían añadir elementos creados por los usuarios, virtuales o tangibles que, a priori, no podrían ser compatibles con esas soluciones. No haría falta adquirir elementos compatibles con estos asistentes o un SW específico, bastaría con crear los elementos en base a lo propuesto en el capítulo 3.

8. **Extensión de los conjuntos de términos relacionados atendiendo al contexto:** Sería positivo que existiera un mecanismo que permitiera conocer la localización de los dispositivos, así como del usuario. De esa manera, cuando un usuario desee llevar a cabo una acción, su orden puede verse completada con esta información minimizando la cantidad de falsos positivos, es decir, de acciones que pueden encajar con la petición efectuada.

BIBLIOGRAFÍA

[1] Mark Weiser. "The Computer for the 21st Century". En: *Scientific American* 265.3 (ene. de 1991), págs. 66-75.

[2] Tuan Nguyen Gia, Victor Kathan Sarker, Igor Tcarenko, Amir M. Rahmani, Tomi Westerlund, Pasi Liljeberg y Hannu Tenhunen. "Energy efficient wearable sensor node for IoT-based fall detection systems". En: *Microprocessors and Microsystems* 56 (2018), págs. 34-46. ISSN: 0141-9331.

[3] Rex Liu, Albara Ah Ramli, Huanle Zhang, Esha Datta, Erik Henricson y Xin Liu. "An Overview of Human Activity Recognition Using Wearable Sensors: Healthcare and Artificial Intelligence". En: *arXiv preprint arXiv:2103.15990* (2021).

[4] Joel Poncha Lemayian y Fadi Al-Turjman. "Intelligent IoT communication in smart environments: an overview". En: *Artificial intelligence in IoT* (2019), págs. 207-221.

[5] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer y Shahid Khan. "Future Internet: the Internet of Things Architecture, possible applications and key challenges". En: *2012 10th International Conference on Frontiers of Information Technology*. IEEE. 2012, págs. 257-260.

[6] Hany F Atlam, Ahmed Alenezi, Abdulrahman Alharthi, Robert J Walters y Gary B Wills. "Integration of cloud computing with internet of things: challenges and open issues". En: *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2017, págs. 670-675.

[7] Alessio Botta, Walter De Donato, Valerio Persico y Antonio Pescapé. "Integration of cloud computing and internet of things: a survey". En: *Future generation computer systems* 56 (2016), págs. 684-700.

[8] Shaik Masthan Babu, A Jaya Lakshmi y B Thirumala Rao. "A study on cloud based Internet of Things: CloudIoT". En: *2015 global conference on communication technologies (GCCT)*. IEEE. 2015, págs. 60-65.

[9] Yuan Ai, Mugen Peng y Kecheng Zhang. "Edge computing technologies for Internet of Things: a primer". En: *Digital Communications and Networks* 4.2 (2018), págs. 77-86.

[10] Amir Vahid Dastjerdi y Rajkumar Buyya. "Fog Computing: Helping the Internet of Things Realize Its Potential". En: *Computer* 49.8 (2016), págs. 112-116.

[11] Keyan Cao, Yefan Liu, Gongjie Meng y Qimeng Sun. "An overview on edge computing research". En: *IEEE access* 8 (2020), págs. 85714-85728.

[12] Tie Qiu, Jiancheng Chi, Xiaobo Zhou, Zhaolong Ning, Mohammed Atiquzzaman y Dapeng Oliver Wu. "Edge computing in industrial internet of things: Architecture, advances and challenges". En: *IEEE Communications Surveys & Tutorials* 22.4 (2020), págs. 2462-2488.

[13] Hai Lin, Sherali Zeadally, Zhihong Chen, Houda Labiod y Lusheng Wang. "A survey on computation offloading modeling for edge computing". En: *Journal of Network and Computer Applications* (2020), pág. 102781.

[14] Chao Gong, Fuhong Lin, Xiaowen Gong y Yueming Lu. "Intelligent cooperative edge computing in internet of things". En: *IEEE Internet of Things Journal* 7.10 (2020), págs. 9372-9382.

[15] Badr El Khalyly, Abdessamad Belangour, Mouad Banane y Allae Erraissi. "A comparative study of Microservices-Based IoT platforms". En: *International Journal of Advanced Computer Science and Applications (IJACSA)* 11.7 (2020), págs. 389-398.

[16] Qian Qu, Ronghua Xu, Seyed Yahya Nikouei y Yu Chen. "An experimental study on microservices based edge computing platforms". En: *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2020, págs. 836-841.

[17] Long Sun, Yan Li y Raheel Ahmed Memon. "An open IoT framework based on microservices architecture". En: *China Communications* 14.2 (2017), págs. 154-162.

[18] Muhammad Alam, Joao Rufino, Joaquim Ferreira, Syed Hassan Ahmed, Nadir Shah y Yuanfang Chen. "Orchestration of microservices for iot using docker and edge computing". En: *IEEE Communications Magazine* 56.9 (2018), págs. 118-123.

[19] Guisheng Fan, Liang Chen, Huiqun Yu y Wei Qi. "Multi-objective optimization of container-based microservice scheduling in edge computing". En: *Computer Science and Information Systems* 00 (2020), págs. 41-41.

[20] Ruozhou Yu, Vishnu Teja Kilari, Guoliang Xue y Dejun Yang. "Load balancing for interdependent IoT microservices". En: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE. 2019, págs. 298-306.

[21] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, Xin Peng, Wenli Zheng y Minyi Guo. "QoS-Aware and Resource Efficient Microservice Deployment in Cloud-Edge Continuum". En: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2021, págs. 932-941.

[22] Vittorio Cortellessa, Daniele Di Pompeo, Romina Eramo y Michele Tucci. "A model-driven approach for continuous performance engineering in microservice-based systems". En: *Journal of Systems and Software* 183 (2022), pág. 111084.

[23] Klayton Castro, Polyane Wercelens, Rafael Padilha, Italo Gervasio, Flávio EG de Deus, William F Giozza, Rafael T de Sousa y col. "Performance Evaluation of the Virtualization Environment of a Microservices-Based Payroll System". En: *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE. 2020, págs. 1-6.

[24] Nathan Cruz Coulson, Stelios Sotiriadis y Nik Bessis. "Adaptive microservice scaling for elastic applications". En: *IEEE Internet of Things Journal* 7.5 (2020), págs. 4195-4202.

[25] Malith Jayasinghe, Jayathma Chathurangani, Gayal Kuruppu, Pasindu Tennage y Srinath Perera. "An Analysis of Throughput and Latency Behaviours Under Microservice Decomposition". En: *International Conference on Web Engineering*. Springer. 2020, págs. 53-69.

[26] Martin Grambow, Lukas Meusel, Erik Wittern y David Bermbach. "Benchmarking microservice performance: a pattern-based approach". En: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 2020, págs. 232-241.

[27] Alberto Avritzer, Vincenzo Ferme, Andrea Janes, Barbara Russo, André van Hoorn, Henning Schulz, Daniel Menasché y Vilc Rufino. "Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests". En: *Journal of Systems and Software* 165 (2020), pág. 110564.

[28] David Lennick, Akramul Azim y Ramiro Liscano. "A Microservice-Based Architecture for Performance and Energy Benchmarking of Docker-Host Linux Distributions on Internet-of-Things Devices". En: *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*. Vol. 1. IEEE. 2021, págs. 705-711.

[29] Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte y Johannes Wettinger. "Performance engineering for microservices: research challenges and

directions". En: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. 2017, págs. 223-226.

[30] Shmuel Tyszberowicz, Robert Heinrich, Bo Liu y Zhiming Liu. "Identifying microservices using functional decomposition". En: *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer. 2018, págs. 50-65.

[31] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka y Thierry Turletti. "A survey of software-defined networking: Past, present, and future of programmable networks". En: *IEEE Communications surveys & tutorials* 16.3 (2014), págs. 1617-1634.

[32] Lingjie Duan, Longbo Huang, Cedric Langbort, Alexey Pozdnukhov, Jean Walrand y Lin Zhang. "Human-in-the-loop mobile networks: A survey of recent advancements". En: *Ieee journal on selected areas in communications* 35.4 (2017), págs. 813-831.

[33] Lin Yang, Wenfeng Li, Masoud Ghandehari y Giancarlo Fortino. "People-centric cognitive internet of things for the quantitative analysis of environmental exposure". En: *IEEE Internet of Things Journal* 5.4 (2017), págs. 2353-2366.

[34] David Bandera, Alejandro Pérez-Vereda, Jose Carlos Canal-Velasco, Ernesto Pimentel y col. "A programming framework for people as a service". En: (2019).

[35] Marco Conti, Andrea Passarella y Sajal K Das. "The Internet of People (IoP): A new wave in pervasive mobile computing". En: *Pervasive and Mobile Computing* 41 (2017), págs. 1-27.

[36] Robert R Morris, Kareem Kouddous, Rohan Kshirsagar y Stephen M Schueller. "Towards an artificially empathic conversational agent for mental health applications: system design and user perceptions". En: *Journal of medical Internet research* 20.6 (2018), e10148.

[37] Alisha Pradhan, Kanika Mehta y Leah Findlater. "'Accessibility Came by Accident' Use of Voice-Controlled Intelligent Personal Assistants by People with Disabilities". En: *Proceedings of the 2018 CHI Conference on human factors in computing systems*. 2018, págs. 1-13.

[38] Michal Luria, Guy Hoffman y Oren Zuckerman. "Comparing social robot, screen and voice interfaces for smart-home control". En: *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017, págs. 580-628.

[39] Shih-Chieh Lin, Chang-Hong Hsu, Walter Talamonti, Yunqi Zhang, Steve Oney, Jason Mars y Lingjia Tang. "Adasa: A conversational in-vehicle digital assistant for advanced driver assistance features". En: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 2018, págs. 531-542.

[40] Kevin Ashton y col. "That 'internet of things' thing". En: *RFID journal* 22.7 (2009), págs. 97-114.

[41] Avish Karmakar, Naiwrita Dey, Tapadyuti Baral, Manojeet Chowdhury y Md Rehan. "Industrial internet of things: a review". En: *2019 international conference on opto-electronics and applied optics (optronix)*. IEEE. 2019, págs. 1-6.

[42] Claude Castelluccia, Mathieu Cunche, Daniel Le Métayer y Victor Morel. "Enhancing transparency and consent in the IoT". En: *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2018, págs. 116-119.

[43] Alexia Dini Kounoudes y Georgia M Kapitsaki. "A mapping of IoT user-centric privacy preserving approaches to the GDPR". En: *Internet of Things* 11 (2020), pág. 100179.

[44] Ammar Rayes y Samer Salam. "The things in iot: Sensors and actuators". En: *Internet of Things From Hype to Reality*. Springer, 2017, págs. 57-77.

[45] Shadab Alam, Shams Tabrez Siddiqui, Ausaf Ahmad, Riaz Ahmad y Mohammed Shuaib. "Internet of Things (IoT) enabling technologies, requirements, and security challenges". En: *Advances in data and information sciences*. Springer, 2020, págs. 119-126.

[46] Suada Hadzovic, Sasa Mrdovic y Milutin Radonjic. "Identification of IoT Actors". En: *Sensors* 21.6 (2021), pág. 2093.

[47] Haris Aftab, Komal Gilani, JiEun Lee, Lewis Nkenyereye, SeungMyeong Jeong y JaeSeung Song. "Analysis of identifiers in IoT platforms". En: *Digital Communications and Networks* 6.3 (2020), págs. 333-340.

[48] Farshad Firouzi, Bahar Farahani, Markus Weinberger, Gabriel DePace y Fereidoon Shams Aliee. "Iot fundamentals: Definitions, architectures, challenges, and promises". En: *Intelligent Internet of Things*. Springer, 2020, págs. 3-50.

[49] Kathryn B Laskey y Kenneth Laskey. "Service oriented architecture". En: *Wiley Interdisciplinary Reviews: Computational Statistics* 1.1 (2009), págs. 101-105.

[50] Behrouz Pourghebleh, Vahideh Hayyolalam y Amir Aghaei Anvigh. "Service discovery in the Internet of Things: review of current trends and research challenges". En: *Wireless Networks* 26.7 (2020), págs. 5371-5391.

[51] Rahatara Ferdousi y Parichay Kumar Mandal. "LOAMY: A cloud-based middleware for CoAP-based IoT service discovery". En: *2019 Second international conference on advanced computational and communication paradigms (ICACCP)*. IEEE. 2019, págs. 1-6.

[52] Ahmad AlZubi, Abdulaziz Alarifi, Mohammed Al-Maitah y Omar A Albasheer. "Location assisted delay-less service discovery method for IoT environments". En: *Computer Communications* 150 (2020), págs. 405-412.

[53] Leonard Kleinrock. "An Internet vision: the invisible global infrastructure". En: *Ad Hoc Networks* 1.1 (2003), págs. 3-11.

[54] Ali Sunyaev. "Cloud computing". En: *Internet computing*. Springer, 2020, págs. 195-236.

[55] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer y Wolfgang Karl. "Scientific cloud computing: Early definition and experience". En: *2008 10th ieee international conference on high performance computing and communications*. Ieee. 2008, págs. 825-830.

[56] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg e Ivona Brandic. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility". En: *Future Generation computer systems* 25.6 (2009), págs. 599-616.

[57] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres y Maik Lindner. *A break in the clouds: towards a cloud definition*. 2008.

[58] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica y col. *Above the clouds: A berkeley view of cloud computing*. Inf. téc. Technical Report UCB/EECS-2009-28, EECS Department, University of California, 2009.

[59] Peter Mell, Tim Grance y col. "The NIST definition of cloud computing". En: (2011).

[60] Shufen Zhang, Hongcan Yan y Xuebin Chen. "Research on key technologies of cloud computing". En: *Physics Procedia* 33 (2012), págs. 1791-1797.

[61] Ben Halpert. *Auditing cloud computing: a security and privacy guide*. Vol. 21. John Wiley & Sons, 2011.

[62] Maricela-Georgiana Avram. "Advantages and challenges of adopting cloud computing from an enterprise perspective". En: *Procedia Technology* 12 (2014), págs. 529-534.

[63] Dothang Truong. "How cloud computing enhances competitive advantages: A research model for small businesses". En: *The Business Review, Cambridge* 15.1 (2010), págs. 59-65.

[64] Timothy Wood, Emmanuel Cecchet, Kadangode K Ramakrishnan, Prashant J Shenoy, Jacobus E van der Merwe y Arun Venkataramani. "Disaster recovery as a cloud service: economic benefits & deployment challenges." En: *HotCloud* 10 (2010), págs. 8-15.

[65] Pankesh Patel, Ajith H Ranabahu y Amit P Sheth. "Service level agreement in cloud computing". En: (2009).

[66] Chnar Mustafa Mohammed, Subhi RM Zebaree y col. "Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: A review". En: *International Journal of Science and Business* 5.2 (2021), págs. 17-30.

[67] Manish Saraswat y RC Tripathi. "Cloud Computing: Analysis of Top 5 CSPs in SaaS, PaaS and IaaS Platforms". En: *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*. IEEE. 2020, págs. 300-305.

[68] Sonia Shahzadi, Muddesar Iqbal, Zia Ul Qayyum y Tasos Dagiuklas. "Infrastructure as a service (IaaS): A comparative performance analysis of open-source cloud platforms". En: *2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE. 2017, págs. 1-6.

[69] Deepali Bajaj, Urmil Bharti, Anita Goel y SC Gupta. "Paas Providers and Their Offerings". En: *International Journal of Scientific & Technology Research* 9.2 (2020), págs. 4009-4015.

[70] Ritu Maheshwari, Aayushi Toshniwal y Avnish Dubey. "Software As A Service Architecture and its Security Issues: A Review". En: *2020 Fourth International Conference on Inventive Systems and Control (ICISC)*. IEEE. 2020, págs. 766-770.

[71] Raafat O Aburukba, Mazin AliKarrar, Taha Landolsi y Khaled El-Fakih. "Scheduling Internet of Things requests to minimize latency in hybrid Fog-Cloud computing". En: *Future Generation Computer Systems* 111 (2020), págs. 539-551.

[72] Shaoshan Liu, Jie Tang, Chao Wang, Quan Wang y Jean-Luc Gaudiot. "A unified cloud platform for autonomous driving". En: *Computer* 50.12 (2017), págs. 42-49.

[73] Santosh Pattar, Rajkumar Buyya, Kuppanna Rajuk Venugopal, SS Iyengar y LM Patnaik. "Searching for the IoT resources: fundamentals, requirements, comprehensive review, and future directions". En: *IEEE Communications Surveys & Tutorials* 20.3 (2018), págs. 2101-2132.

[74] Shancang Li, Li Da Xu y Shanshan Zhao. "The internet of things: a survey". En: *Information Systems Frontiers* 17.2 (2015), págs. 243-259.

[75] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic y Marimuthu Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions". En: *Future generation computer systems* 29.7 (2013), págs. 1645-1660.

[76] Charith Perera, Arkady Zaslavsky, Peter Christen y Dimitrios Georgakopoulos. "Context aware computing for the internet of things: A survey". En: *IEEE communications surveys & tutorials* 16.1 (2013), págs. 414-454.

[77] Flavio Bonomi, Rodolfo Milito, Jiang Zhu y Sateesh Addepalli. "Fog computing and its role in the internet of things". En: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, págs. 13-16.

[78] Subhadeep Sarkar, Subarna Chatterjee y Sudip Misra. "Assessment of the Suitability of Fog Computing in the Context of Internet of Things". En: *IEEE Transactions on Cloud Computing* 6.1 (2015), págs. 46-59.

[79] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li y Lanyu Xu. "Edge computing: Vision and challenges". En: *IEEE internet of things journal* 3.5 (2016), págs. 637-646.

[80] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob y Arif Ahmed. "Edge computing: A survey". En: *Future Generation Computer Systems* 97 (2019), págs. 219-235.

[81] Ahmet Cihat Baktir, Atay Ozgovde y Cem Ersoy. "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions". En: *IEEE Communications Surveys & Tutorials* 19.4 (2017), págs. 2359-2391.

[82] Marjan Gusev y Schahram Dustdar. "Going back to the roots — the evolution of edge computing, an iot perspective". En: *IEEE Internet Computing* 22.2 (2018), págs. 5-15.

[83] Koustabh Dolui y Soumya Kanti Datta. "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing". En: *2017 Global Internet of Things Summit (GIoTS)*. IEEE. 2017, págs. 1-6.

[84] Rodrigo Roman, Javier Lopez y Masahiro Mambo. "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges". En: *Future Generation Computer Systems* 78 (2018), págs. 680-698.

[85] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang y Tao Li. "Edge-oriented computing paradigms: A survey on architecture design and system management". En: *ACM Computing Surveys (CSUR)* 51.2 (2018), págs. 1-34.

[86] Hoang T Dinh, Chonho Lee, Dusit Niyato y Ping Wang. "A survey of mobile cloud computing: architecture, applications, and approaches". En: *Wireless communications and mobile computing* 13.18 (2013), págs. 1587-1611.

[87] Rashid G Alakbarov. "Method for Effective Use of Cloudlet Network Resources." En: *International Journal of Computer Network & Information Security* 12.5 (2020).

[88] Pavel Mach y Zdenek Becvar. "Mobile edge computing: A survey on architecture and computation offloading". En: *IEEE Communications Surveys & Tutorials* 19.3 (2017), págs. 1628-1656.

[89] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang y Tao Li. "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet". En: *ACM Computing Surveys (CSUR)* 52.6 (2019), págs. 1-36.

[90] Simar Preet Singh, Anand Nayyar, Rajesh Kumar y Anju Sharma. "Fog computing: from architecture to edge computing and big data processing". En: *The Journal of Supercomputing* 75.4 (2019), págs. 2070-2105.

[91] Tina Francis y Muthiya Madhiajagan. "A Comparison of Cloud Execution Mechanisms: Fog, Edge and Clone Cloud Computing". En: *Proceeding of the Electrical Engineering Computer Science and Informatics* 4.1 (2017), págs. 446-450.

[92] Baotong Chen, Jiafu Wan, Antonio Celesti, Di Li, Haider Abbas y Qin Zhang. "Edge computing in IoT-based manufacturing". En: *IEEE Communications Magazine* 56.9 (2018), págs. 103-109.

[93] G Sai Sesha Chalapathi, Vinay Chamola, Aabhaas Vaish y Rajkumar Buyya. "Industrial internet of things (iiot) applications of edge and fog computing: A review and future directions". En: *Fog/Edge Computing For Security, Privacy, and Applications* (2021), págs. 293-325.

[94] Redowan Mahmud, Ramamohanarao Kotagiri y Rajkumar Buyya. "Fog computing: A taxonomy, survey and future directions". En: *Internet of everything*. Springer, 2018, págs. 103-130.

[95] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Nedim S Goren, Charif Mahmoudi y col. "Fog computing conceptual model". En: (2018).

[96] Cosmin Avasalcai, Ilir Murturi y Schahram Dustdar. "Edge and fog: A survey, use cases, and future challenges". En: *Fog Computing: Theory and Practice* (2020), págs. 43-65.

[97] Kang Kai, Wang Cong y Luo Tao. "Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues". En: *the journal of China Universities of Posts and Telecommunications* 23.2 (2016), págs. 56-96.

[98] Xintong Song, Yaodong Huang, Qian Zhou, Fan Ye, Yuanyuan Yang y Xiaoming Li. "Pervasive edge data sharing in MANET". En: *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2017, págs. 133-138.

[99] Zaigham Mahmood y Muthu Ramachandran. "Fog computing: concepts, principles and related paradigms". En: *Fog Computing*. Springer, 2018, págs. 3-21.

[100] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow y Paul A Polakos. "A comprehensive survey on fog computing: State-of-the-art and research challenges". En: *IEEE communications surveys & tutorials* 20.1 (2017), págs. 416-464.

[101] Korosh Vatanparvar y Mohammad Abdullah Al Faruque. "Control-as-a-service in cyber-physical energy systems over fog computing". En: *Fog Computing in the Internet of Things*. Springer, 2018, págs. 123-144.

[102] David Bermbach, Frank Pallas, David Garcia Perez, Pierluigi Plebani, Maya Anderson, Ronen Kat y Stefan Tai. "A research perspective on fog computing". En: *International Conference on Service-Oriented Computing*. Springer. 2017, págs. 198-210.

[103] Khalid A Fakeeh. "Privacy and security problems in fog computing". En: *Commun Appl Electron* 4.7 (2016).

[104] Chii Chang, Satish Narayana Srirama y Rajkumar Buyya. "Indie fog: An efficient fog-computing infrastructure for the internet of things". En: *Computer* 50.9 (2017), págs. 92-98.

[105] Shanhe Yi, Zhengrui Qin y Qun Li. "Security and privacy issues of fog computing: A survey". En: *International conference on wireless algorithms, systems, and applications*. Springer. 2015, págs. 685-695.

[106] Manar Abu Talib, Adel Khelifi y Tahsin Ugurlu. "Using ISO 27001 in teaching information security". En: *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE. 2012, págs. 3149-3153.

[107] Deze Zeng, Lin Gu, Song Guo, Zixue Cheng y Shui Yu. "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system". En: *IEEE Transactions on Computers* 65.12 (2016), págs. 3702-3712.

[108] Wangbong Lee, Kidong Nam, Hak-Gyun Roh y Sang-Ha Kim. "A gateway based fog computing architecture for wireless sensors and actuator networks". En: *2016 18th International Conference on Advanced Communication Technology (ICACT)*. IEEE. 2016, págs. 210-213.

[109] Mohammad Aazam y Eui-Nam Huh. "Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT". En: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. IEEE. 2015, págs. 687-694.

[110] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan y Rodney S Tucker. "Fog computing may help to save energy in cloud computing". En: *IEEE Journal on Selected Areas in Communications* 34.5 (2016), págs. 1728-1739.

[111] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder y Boris Koldehofe. "Mobile fog: A programming model for large-scale applications on the internet of things". En: *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. 2013, págs. 15-20.

[112] Mohamed Saleem Haja Nazmudeen, Au Thien Wan y Seyed M Buhari. "Improved throughput for power line communication (plc) for smart meters using fog computing based data aggregation approach". En: *2016 IEEE international smart cities conference (ISC2)*. IEEE. 2016, págs. 1-4.

[113] Mohammad Aazam y Eui-Nam Huh. "Fog computing and smart gateway based communication for cloud of things". En: *2014 International Conference on Future Internet of Things and Cloud*. IEEE. 2014, págs. 464-470.

[114] Simone Cirani, Gianluigi Ferrari, Nicola Iotti y Marco Picone. "The IoT hub: A fog node for seamless management of heterogeneous connected smart objects". En: *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking-Workshops (SECON Workshops)*. IEEE. 2015, págs. 1-6.

[115] Md Redowan Mahmud, Mahbuba Afrin, Md Abdur Razzaque, Mohammad Mehedi Hassan, Abdulhameed Alelaiwi y Majed Alrubaian. "Maximizing quality of experience through context-aware mobile application scheduling in Cloudlet infrastructure". En: *Software: Practice and Experience* 46.11 (2016), págs. 1525-1545.

[116] Clinton Dsouza, Gail-Joon Ahn y Marthony Taguinod. "Policy-driven security management for fog computing: Preliminary framework and a case study". En: *Proceedings of the 2014 IEEE 15th international conference on information reuse and integration (IEEE IRI 2014)*. IEEE. 2014, págs. 16-23.

[117] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti y Matteo Nardelli. "On QoS-aware scheduling of data stream applications over fog computing infrastructures". En: *2015 IEEE Symposium on Computers and Communication (ISCC)*. IEEE. 2015, págs. 271-276.

[118] Shi Yan, Mugen Peng y Wenbo Wang. "User access mode selection in fog computing based radio access networks". En: *2016 IEEE International Conference on communications (ICC)*. IEEE. 2016, págs. 1-6.

[119] Nguyen B Truong, Gyu Myoung Lee y Yacine Ghamri-Doudane. "Software defined networking-based vehicular adhoc network with fog computing". En: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. Ieee. 2015, págs. 1202-1207.

[120] Jessica Oueis, Emilio Calvanese Strinati, Stefania Sardellitti y Sergio Barbarossa. "Small cell clustering for efficient distributed fog computing: A multi-user case". En: *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*. IEEE. 2015, págs. 1-5.

[121] Xueshi Hou, Yong Li, Min Chen, Di Wu, Depeng Jin y Sheng Chen. "Vehicular fog computing: A viewpoint of vehicles as the infrastructures". En: *IEEE Transactions on Vehicular Technology* 65.6 (2016), págs. 3860-3873.

[122] Dongdong Ye, Maoqiang Wu, Shensheng Tang y Rong Yu. "Scalable fog computing with service offloading in bus networks". En: *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE. 2016, págs. 247-251.

[123] Jessica Oueis, Emilio Calvanese Strinati y Sergio Barbarossa. "The fog balancing: Load distribution for small cell cloud computing". En: *2015 IEEE 81st vehicular technology conference (VTC spring)*. IEEE. 2015, págs. 1-6.

[124] César Augusto García-Pérez y Pedro Merino. "Experimental evaluation of fog computing techniques to reduce latency in LTE networks". En: *Transactions on Emerging Telecommunications Technologies* 29.4 (2018), e3201.

[125] Omar Sefraoui, Mohammed Aissaoui y Mohsine Eleuldj. "OpenStack: toward an open-source solution for cloud computing". En: *International Journal of Computer Applications* 55.3 (2012), págs. 38-42.

[126] Nicolas Huin, Brigitte Jaumard y Frédéric Giroire. "Optimal network service chain provisioning". En: *IEEE/ACM Transactions on Networking* 26.3 (2018), págs. 1320-1333.

[127] Wanfu Ding, Wen Qi, Jianping Wang y Biao Chen. "OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV". En: *IEEE Network* 29.3 (2015), págs. 30-35.

[128] Yuan-Yao Shih, Hsin-Peng Lin, Ai-Chun Pang, Ching-Chih Chuang y Chun-Ting Chou. "An NFV-based service framework for IoT applications in edge computing environments". En: *IEEE Transactions on Network and Service Management* 16.4 (2019), págs. 1419-1434.

[129] Manar Jammal, Taranpreet Singh, Abdallah Shami, Rasool Asal y Yiming Li. "Software defined networking: State of the art and research challenges". En: *Computer Networks* 72 (2014), págs. 74-98.

[130] Dimitri Petrik y Georg Herzwurm. "iIoT ecosystem development through boundary resources: a Siemens MindSphere case study". En: *Proceedings of the 2nd ACM SIGSOFT International Workshop on Software-Intensive Business: Start-Ups, Platforms, and Ecosystems*. 2019, págs. 1-6.

[131] Ivan Stojmenovic y Sheng Wen. "The fog computing paradigm: Scenarios and security issues". En: *2014 federated conference on computer science and information systems*. IEEE. 2014, págs. 1-8.

[132] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh y Rajkumar Buyya. "Fog computing: Principles, architectures, and applications". En: *Internet of things*. Elsevier, 2016, págs. 61-75.

[133] Peng Zhang, Joseph K Liu, F Richard Yu, Mehdi Sookhak, Man Ho Au y Xiapu Luo. "A survey on access control in fog computing". En: *IEEE Communications Magazine* 56.2 (2018), págs. 144-149.

[134] Jürgo S Preden, Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid y Emine Calis. "The benefits of self-awareness and attention in fog and mist computing". En: *Computer* 48.7 (2015), págs. 37-45.

[135] Shwet Ketu y Pramod Kumar Mishra. "Cloud, Fog and Mist Computing in IoT: An Indication of Emerging Opportunities". En: *IETE Technical Review* (2021), págs. 1-12.

[136] Minoru Uehara. "Mist computing: Linking cloudlet to fogs". En: *International Conference on Computational Science/Intelligence & Applied Informatics*. Springer. 2017, págs. 201-213.

[137] Manas Kumar Yogi, K Chandrasekhar y G Vijay Kumar. "Mist computing: Principles, trends and future direction". En: *arXiv preprint arXiv:1709.06927* (2017).

[138] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres y Nigel Davies. "The case for vm-based cloudlets in mobile computing". En: *IEEE pervasive Computing* 8.4 (2009), págs. 14-23.

[139] Constantinos Psomas y Ioannis Krikidis. "Wireless powered mobile edge computing: Offloading or local computation?" En: *IEEE Communications Letters* 24.11 (2020), págs. 2642-2646.

[140] Arash Bozorgchenani, Daniele Tarchi y Giovanni Emanuele Corazza. "An energy and delay-efficient partial offloading technique for fog computing architectures". En: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, págs. 1-6.

[141] Zhufang Kuang, Linfeng Li, Jie Gao, Lian Zhao y Anfeng Liu. "Partial offloading scheduling and power allocation for mobile edge computing systems". En: *IEEE Internet of Things Journal* 6.4 (2019), págs. 6774-6785.

[142] Xavi Masip-Bruin, Eva Marín-Tordera, Ghazal Tashakor, Admela Jukan y Guang-Jie Ren. "Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems". En: *IEEE Wireless Communications* 23.5 (2016), págs. 120-128.

[143] Cheng Huang, Rongxing Lu y Kim-Kwang Raymond Choo. "Vehicular fog computing: architecture, use case, and security and forensic challenges". En: *IEEE Communications Magazine* 55.11 (2017), págs. 105-111.

[144] Opeyemi Osanaiye, Shuo Chen, Zheng Yan, Rongxing Lu, Kim-Kwang Raymond Choo y Mqhele Dlodlo. "From cloud to fog computing: A review and a conceptual live VM migration framework". En: *IEEE Access* 5 (2017), págs. 8284-8300.

[145] Gonçalo Carvalho, Bruno Cabral, Vasco Pereira y Jorge Bernardino. "Computation offloading in Edge Computing environments using Artificial Intelligence techniques". En: *Engineering Applications of Artificial Intelligence* 95 (2020), pág. 103840.

[146] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu y Bharat Bhargava. "A survey of computation offloading for mobile systems". En: *Mobile networks and Applications* 18.1 (2013), págs. 129-140.

[147] Xiaohui Zhao, Liqiang Zhao y Kai Liang. "An energy consumption oriented offloading algorithm for fog computing". En: *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. Springer. 2016, págs. 293-301.

[148] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik y Ashwin Patti. "Clonecloud: elastic execution between mobile device and cloud". En: *Proceedings of the sixth conference on Computer systems*. 2011, págs. 301-314.

[149] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra y Paramvir Bahl. "Maui: making smartphones last longer with code offload". En: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 2010, págs. 49-62.

[150] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier y Xinwen Zhang. "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading". En: *2012 Proceedings IEEE Infocom*. IEEE. 2012, págs. 945-953.

[151] Yonggang Wen, Weiwen Zhang y Haiyun Luo. "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones". En: *2012 Proceedings Ieee Infocom*. IEEE. 2012, págs. 2716-2720.

[152] Weiwen Zhang, Yonggang Wen y Dapeng Oliver Wu. "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing". En: *2013 Proceedings Ieee Infocom*. IEEE. 2013, págs. 190-194.

[153] Thinh Quang Dinh, Jianhua Tang, Quang Duy La y Tony QS Quek. "Offloading in mobile edge computing: Task allocation and computational frequency scaling". En: *IEEE Transactions on Communications* 65.8 (2017), págs. 3571-3584.

[154] Maofei Deng, Hui Tian y Bo Fan. "Fine-granularity based application offloading policy in cloud-enhanced small cell networks". En: *2016 IEEE International Conference on Communications Workshops (ICC)*. IEEE. 2016, págs. 638-643.

[155] Felicia Lobillo, Zdenek Becvar, Miguel Angel Puente, Pavel Mach, Francesco Lo Presti, Fabrizio Gambetti, Mariana Goldhamer, Josep Vidal, Anggoro K Widiawan y Emilio Calvanesse. "An architecture for mobile computation offloading on cloud-enabled LTE small cells". En: *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE. 2014, págs. 1-6.

[156] Weiwen Zhang, Yonggang Wen y Dapeng Oliver Wu. "Collaborative task execution in mobile cloud computing under a stochastic wireless channel". En: *IEEE Transactions on Wireless Communications* 14.1 (2014), págs. 81-93.

[157] Yuan Zhang, Hao Liu, Lei Jiao y Xiaoming Fu. "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing". En: *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*. IEEE. 2012, págs. 80-86.

[158] S Eman Mahmoodi, RN Uma y KP Subbalakshmi. "Optimal joint scheduling and cloud offloading for mobile applications". En: *IEEE Transactions on Cloud Computing* 7.2 (2016), págs. 301-313.

[159] Jianhui Liu y Qi Zhang. "Code-partitioning offloading schemes in mobile edge computing for augmented reality". En: *IEEE Access* 7 (2019), págs. 11222-11236.

[160] Quang-Huy Nguyen y Falko Dressler. "A smartphone perspective on computation offloading—a survey". En: *Computer Communications* 159 (2020), págs. 133-154.

[161] Kai Liang, Liqiang Zhao, Xiaoli Chu y Hsiao-Hwa Chen. "An integrated architecture for software defined and virtualized radio access networks with fog computing". En: *IEEE Network* 31.1 (2017), págs. 80-87.

[162] Christine Fricker, Fabrice Guillemin, Philippe Robert y Guilherme Thompson. "Analysis of an offloading scheme for data centers in the framework of fog computing". En: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 1.4 (2016), págs. 1-18.

[163] Ragib Hasan, Mahmud Hossain y Rasib Khan. "Aura: An incentive-driven ad-hoc IoT cloud framework for proximal mobile computation offloading". En: *Future Generation Computer Systems* 86 (2018), págs. 821-835.

[164] Lingjun Pu, Xu Chen, Jingdong Xu y Xiaoming Fu. "D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration". En: *IEEE Journal on Selected Areas in Communications* 34.12 (2016), págs. 3887-3901.

[165] Yang Liu, Changqiao Xu, Yufeng Zhan, Zhixin Liu, Jianfeng Guan y Hongke Zhang. "Incentive mechanism for computation offloading using edge computing: A Stackelberg game approach". En: *Computer Networks* 129 (2017), págs. 399-409.

[166] Karim Habak, Mostafa Ammar, Khaled A Harras y Ellen Zegura. "Femto clouds: Leveraging mobile devices to provide cloud service at the edge". En: *2015 IEEE 8th international conference on cloud computing*. IEEE. 2015, págs. 9-16.

[167] Christian Meurisch, Julien Gedeon, The An Binh Nguyen, Fabian Kaup y Max Muhlhauser. "Decision support for computational offloading by probing unknown services". En: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, págs. 1-9.

[168] Petar Krivic, Pavle Skocir, Mario Kusek y Gordan Jezic. "Microservices as agents in IoT systems". En: *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*. Springer. 2017, págs. 22-31.

[169] Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Stephan T Larsen y Manuel Mazzara. "From monolithic to microservices: An experience report from the banking domain". En: *IEEE Software* 35.3 (2018), págs. 50-55.

[170] Daniel Escobar, Diana Cárdenas, Rolando Amarillo, Eddie Castro, Kelly Garcés, Carlos Parra y Rubby Casallas. "Towards the understanding and evolution of monolithic applications as microservices". En: *2016 XLII Latin American Computing Conference (CLEI)*. IEEE. 2016, págs. 1-11.

[171] Rui Chen, Shanshan Li y Zheng Li. "From monolith to microservices: A dataflow-driven approach". En: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2017, págs. 466-475.

[172] Sam Newman. *Building microservices*. O'Reilly Media, Inc., 2021.

[173] Samodha Pallewatta, Vassilis Kostakos y Rajkumar Buyya. "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments". En: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 2019, págs. 71-81.

[174] Cleber Jorge Lira de Santana, Brenno de Mello Alencar y Cássio V Serafim Prazeres. "Reactive microservices for the internet of things: A case study in fog computing". En: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2019, págs. 1243-1251.

[175] Hanan Shukur, Subhi Zeebaree, Rizgar Zebari, Diyar Zeebaree, Omar Ahmed y Azar Salih. "Cloud computing virtualization of resources allocation for distributed systems". En: *Journal of Applied Science and Technology Trends* 1.3 (2020), págs. 98-105.

[176] Vitor Goncalves da Silva, Marite Kirikova y Gundars Alksnis. "Containers for Virtualization: An Overview." En: *Appl. Comput. Syst.* 23.1 (2018), págs. 21-27.

[177] Roberto Morabito, Jimmy Kjällman y Miika Komu. "Hypervisors vs. lightweight virtualization: a performance comparison". En: *2015 IEEE International Conference on Cloud Engineering*. IEEE. 2015, págs. 386-393.

[178] Václav Struhár, Moris Behnam, Mohammad Ashjaei y Alessandro V Papadopoulos. "Real-time containers: A survey". En: *2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2020.

[179] Ryan Wu, Bingwei Liu, Yu Chen, Erik Blasch, Haibin Ling y Genshe Chen. "A container-based elastic cloud architecture for pseudo real-time exploitation of wide area motion imagery (wami) stream". En: *Journal of Signal Processing Systems* 88.2 (2017), págs. 219-231.

[180] Kinnary Jangla. "Containers". En: *Accelerating Development Velocity Using Docker*. Springer, 2018, págs. 1-8.

[181] Satish Narayana Srirama, Mainak Adhikari y Souvik Paul. "Application deployment using containers with auto-scaling for microservices in cloud environment". En: *Journal of Network and Computer Applications* 160 (2020), pág. 102629.

[182] Marcelo Amaral, Jorda Polo, David Carrera, Iqbal Mohomed, Merve Unuvar y Malgorzata Steinder. "Performance evaluation of microservices architectures using containers". En: *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE. 2015, págs. 27-34.

[183] Claus Pahl, Pooyan Jamshidi y Olaf Zimmermann. "Microservices and containers". En: *Software Engineering 2020* (2020).

[184] Andrey Mirkin, Alexey Kuznetsov y Kir Kolyshkin. "Containers checkpointing and live migration". En: *Proceedings of the Linux Symposium*. Vol. 2. 2008, págs. 85-90.

[185] Marek Moravcik, Pavel Segec, Martin Kontsek, Jana Uramova y Jozef Papan. "Comparison of LXC and Docker Technologies". En: *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE. 2020, págs. 481-486.

[186] Dirk Merkel y col. "Docker: lightweight linux containers for consistent development and deployment". En: *Linux journal* 2014.239 (2014), pág. 2.

[187] Dani Adhipta, Selo Sulistyo y Widyawan Widyawan. "A Process Checkpoint Evaluation at User Space of Docker Framework on Distributed Computing Infrastructure". En: *2020 12th International Conference on Information Technology and Electrical Engineering (ICITEE)*. IEEE. 2020, págs. 141-145.

[188] Harri Manninen, Vesa Jääskeläinen y Jan Olaf Blech. "Performance evaluation of containerization platforms for control and monitoring devices". En: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE. 2020, págs. 1061-1064.

[189] Joao Rufino, Muhammad Alam, Joaquim Ferreira, Abdur Rehman y Kim Fung Tsang. "Orchestration of containerized microservices for IIoT using Docker". En: *2017 IEEE International Conference on Industrial Technology (ICIT)*. IEEE. 2017, págs. 1532-1536.

[190] Nikhil Marathe, Ankita Gandhi y Jaimeel M Shah. "Docker swarm and kubernetes in cloud computing environment". En: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE. 2019, págs. 179-184.

[191] Paridhika Kayal. "Kubernetes in Fog Computing: Feasibility Demonstration, Limitations and Improvement Scope". En: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE. 2020, págs. 1-6.

[192] Dharmesh Kakadia. *Apache Mesos Essentials*. Packt Publishing Birmingham, UK, 2015.

[193] Koustabh Dolui y Csaba Kiraly. "Towards multi-container deployment on IoT gateways". En: *2018 ieee global communications conference (globecom)*. IEEE. 2018, págs. 1-7.

[194] Paolo Bellavista y Alessandro Zanni. "Feasibility of fog computing deployment based on docker containerization over raspberrypi". En: *Proceedings of the 18th international conference on distributed computing and networking*. 2017, págs. 1-10.

[195] Cecil Wöbker, Andreas Seitz, Harald Mueller y Bernd Bruegge. "Fogernetes: Deployment and management of fog computing applications". En: *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2018, págs. 1-7.

[196] Daniele Santoro, Daniel Zozin, Daniele Pizzolli, Francesco De Pellegrini y Silvio Cretti. "Foggy: A platform for workload orchestration in a fog computing environment". En: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2017, págs. 231-234.

[197] Shanshan Li, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen y Muhammad Ali Babar. "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review". En: *Information and Software Technology* 131 (2021), pág. 106449.

[198] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle y Gastón Márquez. "Design, monitoring, and testing of microservices systems: The practitioners' perspective". En: *Journal of Systems and Software* 182 (2021), pág. 111061.

[199] André De Camargo, Ivan Salvadori, Ronaldo dos Santos Mello y Frank Siqueira. "An architecture to automate performance tests on microservices". En: *Proceedings of the 18th international conference on information integration and web-based applications and services*. 2016, págs. 422-429.

[200] Cesare Pautasso, Olaf Zimmermann, Mike Amundsen, James Lewis y Nicolai Josuttis. "Microservices in practice, part 1: Reality check and service design". En: *IEEE software* 34.01 (2017), págs. 91-98.

[201] Maura Cerioli, Maurizio Leotta y Filippo Ricca. "What 5 million job advertisements tell us about testing: a preliminary empirical investigation". En: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 2020, págs. 1586-1594.

[202] Praveen Kumar Venkatesan, Rikhil Gade Rozario y Jinan Fiaidhi. "Junit framework for unit testing." En: (2020).

[203] Anna Derezińska y Sofia Krutko. "Migration of Unit Tests of C# Programs". En: *International Conference on Dependability and Complex Systems*. Springer. 2021, págs. 53-62.

[204] Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, Wen-Tin Lee, Shin-Jie Lee y Nien-Lin Hsueh. "Using service dependency graph to analyze and test microservices". En: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 2. IEEE. 2018, págs. 81-86.

[205] Luca Gazzola, Maayan Goldstein, Leonardo Mariani, Itai Segall y Luca Ussi. "Automatic ex-vivo regression testing of microservices". En: *Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test*. 2020, págs. 11-20.

[206] Davide Spadini, Mauricio Aniche, Magiel Bruntink y Alberto Bacchelli. "To mock or not to mock? an empirical study on mocking practices". En: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE. 2017, págs. 402-412.

[207] Shaikh Mostafa y Xiaoyin Wang. "An empirical study on the usage of mocking frameworks in software testing". En: *2014 14th international conference on quality software*. IEEE. 2014, págs. 127-132.

[208] Madhuri R Marri, Tao Xie, Nikolai Tillmann, Jonathan De Halleux y Wolfram Schulte. "An empirical study of testing file-system-dependent software with mock objects". En: *2009 ICSE Workshop on Automation of Software Test*. IEEE. 2009, págs. 149-153.

[209] Tammo Freese. "EasyMock: dynamic mock objects for JUnit". En: *Proceedings of the 3rd International Conference on Extreme Programming and Agile Processes in Software Engineering*. 2002, págs. 1-5.

[210] Steve Freeman, Tim Mackinnon, Nat Pryce y Joe Walnes. "jMock: supporting responsibility-based design with mock objects". En: *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. 2004, págs. 4-5.

[211] Florian Scharf, Christian Wolters, Michael Herczeg y Jörg Cassens. "Cross-Device Interaction Definition, Taxonomy and Applications". En: (2013).

[212] Pai Chet Ng, Petros Spachos, Stefano Gregori y Konstantinos N Plataniotis. "Personal Devices for Contact Tracing: Smartphones and Wearables to Fight Covid-19". En: *IEEE Communications Magazine* 59.9 (2021), págs. 24-29.

[213] Mohammad S Hasan y Hongnian Yu. "Innovative developments in HCI and future trends". En: *International Journal of Automation and Computing* 14.1 (2017), págs. 10-20.

[214] Gülin Dede y Murat Hüsnü Sazlı. "Speech recognition with artificial neural networks". En: *Digital Signal Processing* 20.3 (2010), págs. 763-768.

[215] George W. Furnas, Thomas K. Landauer, Louis M. Gomez y Susan T. Dumais. "The vocabulary problem in human-system communication". En: *Communications of the ACM* 30.11 (1987), págs. 964-971.

[216] Thomas K. Landauer, Kathleen M. Galotti y STEVE Hartwell. "Natural command names and initial learning: A study of text-editing terms". En: *Communications of the ACM* 26.7 (1983), págs. 495-503.

[217] P Barnard, N Hammond, A MacLean y J Morton. "Learning and remembering interactive commands". En: *Proceedings of the 1982 Conference on Human Factors in Computing Systems*. 1982, págs. 2-7.

[218] Luigi Atzori, Antonio Iera y Giacomo Morabito. "The internet of things: A survey". En: *Computer networks* 54.15 (2010), págs. 2787-2805.

[219] Morimasa Matsuda, Takako Nonaka y Tomohiro Hase. "An av control method using natural language understanding". En: *IEEE Transactions on Consumer Electronics* 52.3 (2006), págs. 990-997.

[220] So-Young Park, Jeunghyun Byun, Hae-Chang Rim, Do-Gil Lee y Heuiseok Lim. "Natural language-based user interface for mobile devices with limited resources". En: *IEEE Transactions on Consumer Electronics* 56.4 (2010), págs. 2086-2092.

[221] Thomas R Gruber. "A translation approach to portable ontology specifications". En: *Knowledge acquisition* 5.2 (1993), págs. 199-220.

[222] Bessam Abdulrazak, Belkacem Chikhaoui, Charles Gouin-Vallerand y Benoit Fraikin. "A standard ontology for smart spaces". En: *International Journal of Web and Grid Services* 6.3 (2010), págs. 244-268.

[223] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers y Koen De Bosschere. "Towards an extensible context ontology for ambient intelligence". En: *European Symposium on Ambient Intelligence*. Springer. 2004, págs. 148-159.

[224] Harry Chen, Filip Perich, Tim Finin y Anupam Joshi. "Soupa: Standard ontology for ubiquitous and pervasive applications". En: *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.* IEEE. 2004, págs. 258-267.

[225] Harry Chen, Tim Finin y Anupam Joshi. "An ontology for context-aware pervasive computing environments". En: *The knowledge engineering review* 18.3 (2003), págs. 197-207.

[226] Daniele Riboni y Claudio Bettini. "COSAR: hybrid reasoning for context-aware activity recognition". En: *Personal and Ubiquitous Computing* 15.3 (2011), págs. 271-289.

[227] Liming Chen, Chris Nugent, Maurice Mulvenna, Dewar Finlay y Xin Hong. "Semantic smart homes: towards knowledge rich assisted living environments". En: *Intelligent Patient Management*. Springer, 2009, págs. 279-296.

[228] Laura Daniele, Frank den Hartog y Jasper Roes. "Created in close interaction with the industry: the smart appliances reference (SAREF) ontology". En: *International Workshop Formal Ontologies Meet Industries*. Springer. 2015, págs. 100-112.

[229] Michael Compton, Payam Barnaghi, Luis Bermudez, Raul Garcia-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog y col. "The SSN ontology of the W3C semantic sensor network incubator group". En: *Journal of Web Semantics* 17 (2012), págs. 25-32.

[230] Konstantinos Kotis y Artem Katasonov. "An iot-ontology for the representation of interconnected, clustered and aligned smart entities". En: *Technical report, VTT Technical Research Center, Finland VTT Technical Research Center, Finland* (2012).

[231] Maria Bermudez-Edo, Tarek Elsaleh, Payam Barnaghi y Kerry Taylor. "IoT-Lite: a lightweight semantic model for the Internet of Things". En: *2016 INTL IEEE conferences on ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress (uic/atc/scalcom/cbdcom/iop/smartworld)*. IEEE. 2016, págs. 90-97.

[232] Shinji Kikuchi, Akihito Nakamura, Daishi Yoshino y col. "Evaluation on information model about sensors featured by relationships to measured structural objects". En: *Advances in Internet of Things* 6.03 (2016), pág. 31.

[233] Shohreh Ahvar, Gabriel Santos, Nouredine Tamani, Bernard Istasse, Isabel Praça, Paul-Emmanuel Brun, Yacine Ghamri y Noël Crespi. "Ontology-based model for trusted critical site supervision in FUSE-IT". En: *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. IEEE. 2017, págs. 313-315.

[234] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens y Chris Wroe. "A practical guide to building owl ontologies using protégé 4 and coode tools edition 1.2". En: *The university of Manchester* 107 (2009).

[235] Vinay K Chaudhri, Adam Farquhar, Richard Fikes, Peter D Karp y James P Rice. "Open Knowledge Base Connectivity 2.0.3 (Proposed)". En: *Artificial Intelligence Center of SRI International and Knowledge Systems Laboratory of Stanford University* (1998).

[236] John Domingue. "Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the web". En: (1998).

[237] Bill Swartout, Ramesh Patil, Kevin Knight y Tom Russ. "Ontosaurus: a tool for browsing and editing ontologies". En: *9th Banff Knowledge Aquisition for KNowledge-based systems Workshop*. 1996.

[238] Julio C Arpirez, Oscar Corcho, Mariano Fernandez-Lopez y Asunción Gómez-Pérez. "WebODE: a scalable workbench for ontological engineering". En: *Proceedings of the 1st international conference on Knowledge capture*. 2001, págs. 6-13.

[239] Adam Farquhar, Richard Fikes y James Rice. "The ontolingua server: A tool for collaborative ontology construction". En: *International journal of human-computer studies* 46.6 (1997), págs. 707-727.

[240] Xiaomeng Su y Lars Ilebrekke. "A comparative study of ontology languages and tools". En: *International Conference on Advanced Information Systems Engineering*. Springer. 2002, págs. 761-765.

[241] Bhaskar Kapoor y Savita Sharma. "A comparative study ontology building tools for semantic web applications". En: *International Journal of Web & Semantic Technology* 1.3 (2010), págs. 1-13.

[242] Fagui Liu, Ping Li y Dacheng Deng. "Device-oriented automatic semantic annotation in IoT". En: *Journal of Sensors* 2017 (2017).

[243] Felix Kuhr, Bjarne Witten y Ralf Möller. "Corpus-driven annotation enrichment". En: *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*. IEEE. 2019, págs. 138-141.

[244] Carmen De Maio, Giuseppe Fenza, Mariacristina Gallo, Vincenzo Loia y Sabrina Senatore. "Formal and relational concept analysis for fuzzy-based automatic semantic annotation". En: *Applied intelligence* 40.1 (2014), págs. 154-177.

[245] Eric Charton, Michel Gagnon y Benoit Ozell. "Automatic semantic web annotation of named entities". En: *Canadian Conference on Artificial Intelligence*. Springer. 2011, págs. 74-85.

[246] Gayo Diallo, Michel Simonet y Ana Simonet. "An approach to automatic ontology-based annotation of biomedical texts". En: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2006, págs. 1024-1033.

[247] Ding Rong. "The Research on Automatic Semantic Annotation Methods". En: *Lanzhou University of Technology, Lanzhou, China* (2012).

[248] Payam Barnaghi, Wei Wang, Lijun Dong y Chonggang Wang. "A linked-data model for semantic sensor streams". En: *2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE Cyber, physical and social computing*. IEEE. 2013, págs. 468-475.

[249] Sefki Kolozali, Maria Bermudez-Edo, Daniel Puschmann, Frieder Ganz y Payam Barnaghi. "A knowledge-based approach for real-time iot data stream annotation and processing". En: *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*. IEEE. 2014, págs. 215-222.

[250] P Chenyi. "Service-oriented entity semantic annotation in internet of things [MS thesis]". En: *South China University of Technology, Guangzhou, China* (2015).

[251] B Jia. "Research on Semantic-based Service Architecture and Key Algorithms for the Internet of Things". En: *Jilin University* (2013).

[252] Simon Mayer, Jack Hodges, Dan Yu, Mareike Kritzler y Florian Michahelles. "An open semantic framework for the industrial Internet of Things". En: *IEEE Intelligent Systems* 32.1 (2017), págs. 96-101.

[253] Christoph Treude y Margaret-Anne Storey. "How tagging helps bridge the gap between social and technical aspects in software development". En: *2009 IEEE 31st International Conference on Software Engineering*. IEEE. 2009, págs. 12-22.

[254] Arash Joorabchi, Michael English y Abdulhussain E Mahdi. "Automatic mapping of user tags to wikipedia concepts: The case of a q&a website–stackoverflow". En: *Journal of Information Science* 41.5 (2015), págs. 570-583.

[255] Anton Barua, Stephen W Thomas y Ahmed E Hassan. "What are developers talking about? an analysis of topics and trends in stack overflow". En: *Empirical Software Engineering* 19.3 (2014), págs. 619-654.

[256] Jin Liu, Pingyi Zhou, Zijiang Yang, Xiao Liu y John Grundy. "FastTagRec: fast tag recommendation for software information sites". En: *Automated Software Engineering* 25.4 (2018), págs. 675-701.

[257] Pingyi Zhou, Jin Liu, Xiao Liu, Zijiang Yang y John Grundy. "Is deep learning better than traditional approaches in tag recommendation for software information sites?" En: *Information and software technology* 109 (2019), págs. 1-13.

[258] Pingyi Zhou, Jin Liu, Zijiang Yang y Guangyou Zhou. "Scalable tag recommendation for software information sites". En: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2017, págs. 272-282.

[259] Can Li, Ling Xu, Meng Yan y Yan Lei. "TagDC: A tag recommendation method for software information sites with a combination of deep learning and collaborative filtering". En: *Journal of Systems and Software* 170 (2020), pág. 110783.

[260] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai y Xuanjing Huang. "Pre-trained models for natural language processing: A survey". En: *Science China Technological Sciences* (2020), págs. 1-26.

[261] Daniel W Otter, Julian R Medina y Jugal K Kalita. "A survey of the usages of deep learning for natural language processing". En: *IEEE Transactions on Neural Networks and Learning Systems* 32.2 (2020), págs. 604-624.

[262] Stuart Cheshire y Marc Krochmal. *DNS-based service discovery*. Inf. téc. RFC 6763, February, 2013.

[263] Antonio J Jara, Pedro Martinez-Julia y Antonio Skarmeta. "Light-weight multicast DNS and DNS-SD (lmDNS-SD): IPv6-based resource and service discovery for the web of things". En: *2012 Sixth international conference on innovative mobile and internet services in ubiquitous computing*. IEEE. 2012, págs. 731-738.

[264] Eugene A Gryazin. "Service discovery in bluetooth". En: *Group for Robotics and Virtual Reality. Department of Computer Science. Helsinki University of Technology, Helsinki, Finland. Published at NEC CiteSeer, Scientific Literature Digital Library* (2006).

[265] Feng Zhu, Matt W Mutka y Lionel M Ni. "Service discovery in pervasive computing environments". En: *IEEE Pervasive computing* 4.4 (2005), págs. 81-90.

[266] Ahmed A Ismail, Haitham S Hamza y Amira M Kotb. "Performance evaluation of open source iot platforms". En: *2018 IEEE global conference on internet of things (GCIoT)*. IEEE. 2018, págs. 1-5.

[267] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton y Christopher D. Manning. "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages". En: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2020.

[268] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard y David McClosky. "The Stanford CoreNLP natural language processing toolkit". En: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, págs. 55-60.

[269] George A Miller. "WordNet: a lexical database for English". En: *Communications of the ACM* 38.11 (1995), págs. 39-41.

[270] Carl Hewitt. "Actor model of computation: scalable robust information systems". En: *arXiv preprint arXiv:1008.1459* (2010).

[271] Alexandra Voit, Tonja Machulla, Dominik Weber, Valentin Schwind, Stefan Schneegass y Niels Henze. "Exploring notifications in smart home environments". En: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*. 2016, págs. 942-947.

[272] Shuo Feng, Peyman Setoodeh y Simon Haykin. "Smart home: Cognitive interactive people-centric Internet of Things". En: *IEEE Communications Magazine* 55.2 (2017), págs. 34-39.

[273] Lana Jalal y Maurizio Murroni. "The impact of multi-sensorial media in smart home scenario on user experience and emotions". En: *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE. 2019, págs. 1-6.

[274] Fariba Sadri. "Ambient intelligence: A survey". En: *ACM Computing Surveys (CSUR)* 43.4 (2011), págs. 1-66.

[275] Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan y Shelley XQ Zhang. "The intelligent home testbed". En: *environment* 2 (1999), pág. 15.

[276] Jorge Parra, M Anwar Hossain, Aitor Uribarren, Eduardo Jacob y Abdulmotaleb El Saddik. "Flexible smart home architecture using device profile for web services: A peer-to-peer approach". En: *International Journal of Smart Home* 3.2 (2009), págs. 39-56.

[277] K Yang y Sung-Bae Cho. "Towards sustainable smart homes by a hierarchical hybrid architecture of an intelligent agent". En: *Sustainability* 8.10 (2016), pág. 1020.

[278] Trung Nguyen, Barth Lakshmanan, Chengjie Lin, Weihua Sheng, Ye Gu, Meiqin Liu y Senlin Zhang. "A miniature smart home testbed for research and education". En: *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE. 2017, págs. 1637-1642.

[279] Jie Jiang, Riccardo Pozza, Nigel Gilbert y Klaus Moessner. "Makesense: An iot testbed for social research of indoor activities". En: *ACM Transactions on Internet of Things* 1.3 (2020), págs. 1-25.