Coreografía de dispositivos ubicuos basada en SOA: ¿Cuánto de limitados pueden ser los dispositivos?

Oscar A. Testa, Efraín R. Fonseca C., Germán Montejano, and Oscar Dieste

Universidad Nacional de La Pampa
Facultad de Ciencias Exactas y Naturales
La Pampa - Argentina
Universidad de las Fuerzas Armadas ESPE
Departamento de Ciencias de la Computación
Quito - Ecuador
Universidad Nacional de San Luis
Facultad de Ciencias Físico Matemáticas y Naturales
San Luis - Argentina
Universidad Politécnica de Madrid
Escuela Técnica Superior de Ing. Informática
Madrid - España
otesta@exactas.unlpam.edu.ar
erfonseca@espe.edu.ec
gmonte@unsl.edu.ar
odieste@fi.upm.es

Resumen Actualmente nos encontramos inmersos en ambientes donde los dispositivos ubicuos forman parte de nuestras tareas diarias. Los dispositivos ubicuos no proporcionan servicios de forma aislada, sino que deben cooperar con otros dispositivos. Existen desafíos para la composición de dispositivos ubicuos, como el caso de tolerancia a fallas, escaso nivel de procesamiento, etc. Se han desarrollado distintos frameworks para dar solución a la composición de dispositivos y los problemas asociados, sin embargo estos frameworks no están basados sobre mecanismos estándares y abiertos. Ante esto hemos desarrollado nuestro propio framework, el cual cumple con estos requisitos, además de ser interoperable, simple y extensible, basándose sobre la especificación WS-CDL de SOA. El desafío es poder incluir en el framework dispositivos con capacidades menores (memoria y procesamiento). Presentamos un análisis de memoria que nos indique los requerimientos mínimos para la aplicación de los mecanismos de SOA en la coordinación e inclusión de estos dispositivos.

Keywords: Ingeniería de Software; SOA; Dispositivos ubicuos; Servicios; Coreografía de servicios, WS-CDL

1. Introducción

Los avances tecnológicos han permitido que los dispositivos ubicuos [1] sean al mismo tiempo generadores y consumidores de servicios, lo que se traduce en un ambiente de cooperación entre dispositivos, permitiendo aumentar o mejorar las funcionalidades en base a la composición. Por composición, entendemos la forma en que los dispositivos ubicuos se pueden combinar para realizar una tarea determinada. Esto implica que se produzcan desafíos tales como: tolerancia a fallas, escaso nivel de procesamiento, problemas de conectividad, por mencionar algunos ejemplos [2].

La composición de dispositivos ubicuos presenta desafíos adicionales tales como: la heterogeneidad de los mismos, las contingencias de los dispositivos, y la personalización de los mismos (ej: provisión de servicios de acuerdo a las preferencias del usuario). Dado que los dispositivos ubicuos poseen limitaciones de recursos (ej. poca memoria y batería), se deben hacer consideraciones especiales respecto a la eficiencia y rendimiento de la composición de dispositivos.

Las características descritas hacen que la composición de dispositivos ubicuos se constituya un área de investigación muy atractiva, donde los avances aún no son claros al día de hoy [2].

Actualmente existen algunos frameworks de coordinación de dispositivos ubicuos, como **Aura**[3] de la Universidad de Carnegie Mellon; **Gaia** [4] de la Universidad de Illinois; **Oxigen** [5] perteneciente al MIT (Massachusetts Institute of Technology); **Amigo**[6] proveniente de Amigo Consortium; **OSGi**; **Android Studio** y otros menos específicos. Todos estos frameworks tienen como objetivo principal compartir y proveer de servicios a los usuarios.

La academia también ha aportado estudios relacionados a la composición de dispositivos ubicuos, como es el caso de Viroli [7]; Najar & otros [8]; Loke [9]; Palmieri [10]. Estos trabajos van desde ambientes auto organizados hasta descubrimiento de dispositivos de manera automática.

Si bien, se han realizado esfuerzos y avances en la composición de dispositivos ubicuos, éstos esfuerzos no han logrado que los dispositivos se coordinen entre sí sin la necesidad de un coordinador central y sin basarse sobre mecanismos estandarizados y abiertos.

Nuestro framework [11] propone ventajas alternativas en la composición de dispositivos ubicuos, como es el caso de la estandarización, la extensibilidad, etc. Para ello, construimos un framework que nos permite ejecutar coreografías definidas en el lenguaje en WS-CDL (de las siglas del inglés Web Service Choreography Description Language), con dispositivos ubicuos. Por coreografía entendemos que los dispositivos se comunican de a pares sin un coordinador central para llevar adelante una tarea determinada [12]. El framework desarrollado puede interpretar prácticamente la totalidad de las descripciones realizadas en WS-CDL, además de contar con la posibilidad de trabajar con transacciones distribuidas

[1] Dispositivos electrónicos que tienen capacidad de procesamiento y comunicación, y pueden ser encontrados en cualquier lugar: la oficina, el auto, la casa, o la misma ropa con la que vestimos [1]

al estilo de WS-Transaction [2]. A su vez, también tiene la posibilidad de realizar adecuaciones en la ejecución de la coreografía ante la falla o desaparición de un dispositivo, logrando que la ejecución se recupere y pueda continuar.

Este desarrollo inicial del framework de ejecución de coreografías, ahora tiene como desafío incorporar nuevos dispositivos ubicuos, con menores prestaciones a nivel de capacidad de procesamiento, memoria y otros mecanismos de comunicación. Es decir, la versión actual del framework no puede añadir a dichos dispositivos, debido a las características mencionadas.

Nuestra contribución consiste en lograr la inclusión de dispositivos ubicuos con capacidades cada vez menores, a un ambiente de cooperación y coordinación como es el caso del framework que hemos desarrollado. Para ello realizaremos una análisis de memoria que nos indique cuáles son los requerimientos mínimos para que se puedan aplicar los mecanismos de SOA en la coordinación de los dispositivos.

Lo que resta del artículo se estructura de la siguiente forma: en la Sección 2 se describirá el framework desarrollado junto a determinados conceptos de SOA. En la Sección 3 presentamos las limitaciones que presentan algunos dispositivos en la utilización del framework. El análisis de memoria propiamente dicho, que es en sí la principal contribución del trabajo, es descrita en la Sección 4. Finalmente, en la Sección 5 presentamos las conclusiones y trabajos futuros.

2. Antecedentes

2.1. Computación orientada a servicios

Los mecanismos de composición de servicios, como las orquestaciones y coreografías, son aspectos bien conocidos en SOA (Service Oriented Architecture) que permiten construir sistemas de negocio complejos y aplicaciones a partir de una gran cantidad de servicios heterogéneos, simples y distribuidos. Estos conceptos podrían ser aplicables a ambientes ubicuos, en especial las coreografías, las cuales pueden fácilmente interpretarse en términos de dispositivos y ambientes ubicuos. Sin embargo, en determinados ambientes donde los servicios son dinámicos, móviles, menos fiables y dependientes del dispositivo, los mecanismos de composición establecidos para servicios web no son directamente aplicables [14].

Adicionalmente, la composición de múltiples dispositivos ubicuos presenta nuevos desafíos que no son compatibles con la composición de servicios web. En particular, los mecanismos de composición en ambientes masivos [3] necesitan hacer frente a distintas contingencias que pueden ocurrir con estos elementos, así como también contemplar la heterogeneidad de los mismos. Estos dispositivos tienen distintas limitantes como son la cantidad de memoria disponible, la

[2] El framework iumplementa una capa de transacciones distribuidas con servicios REST, de la forma definida en [13]

[3] Por ejemplo el de dispositivos móviles

durabilidad de la batería, la disponibilidad de acuerdo a la red del lugar donde se encuentren en un momento determinado, etc. En ambientes ubicuos, la disponibilidad y confiabilidad de los dispositivos no puede ser garantizada. No obstante, a pesar de todas las dificultades, existen ventajas que podría brindar la adaptación de los conceptos de SOA para la composición de dispositivos ubicuos, como son la estandarización tanto de los protocolos como de los mecanismos de comunicación entre los dispositivos, y la compatibilidad con otras plataformas de servicios ya existentes.

Las similaridades entre la composición de servicios web y la coordinación de dispositivos ubicuos es sorprendente. Si pensamos que cada dispositivo ubicuo en un ambiente pervasivo [4] es proveedor, o, consumidor de un servicio, la coordinación de dispositivos encaja perfectamente con la composición de servicios. Es también sorprendente que esta similitud no haya sido apenas explorada con anterioridad, salvo en el trabajo de Sheng [2] se hace mención a ello.

2.2. Framework

El framework de ejecución de coreografías desarrollado[11] se construyó siguiendo los lineamientos de la metodología Design Science, de forma evolutiva. Cada nuevo ciclo se basó en la evaluación de los resultados obtenidos en el ciclo previo, aplicando las mejoras necesarias o pertinentes. Los lenguajes de programación utilizados son PHP y C++ y fueron seleccionados en función de los dispositivos que se utilizaron para la prueba de concepto.

La base de la ejecución de coreografías se centra en una serie de clases que, en principio, leen la descripción en WS-CDL de la coreografía, y en base al dispositivo que lo ejecuta; determina en primer lugar en qué posición de la ejecución de la coreografía se encuentra, esto puede verse en la Figura 1 como el punto inicial.

Luego se determina cuál o cuáles son los pasos siguientes en la ejecución, se puede ver en la Figura 1 en el estado "Determinar siguiente".

Una vez determinados los pasos que se deben ejecutar, se hacen las invocaciones a otros dispositivos, teniendo en cuenta las actividades descriptas en la definición de la coreografía en el lenguaje WS-CDL. Esta ejecución se hace de manera controlada, en el sentido de examinar que no se produzcan contingencias provenientes de las características de los dispositivos (desapariciones, latencia en la respuesta por falta de capacidad de procesamiento, etc.). En base a esta verificación se realizan las tareas correctivas correspondientes, según se han expresado en la definición de la coreografía. La estructura de ejecución del framework la podemos apreciar en la Figura 1.

Durante el desarrollo del framework se trabajó con un escenario de pruebas donde se planteó un ambiente de autopistas inteligentes para dar solución a problemas referidos al tránsito vehicular, donde se simularon vehículos, balizas, Centrales de Emergencias, etc.

[4] Ambientes pervasivos son entornos poblados por varios dispositivos (sensores, actuadores, etc) y aplicaciones de software integrados de forma transparente [1]

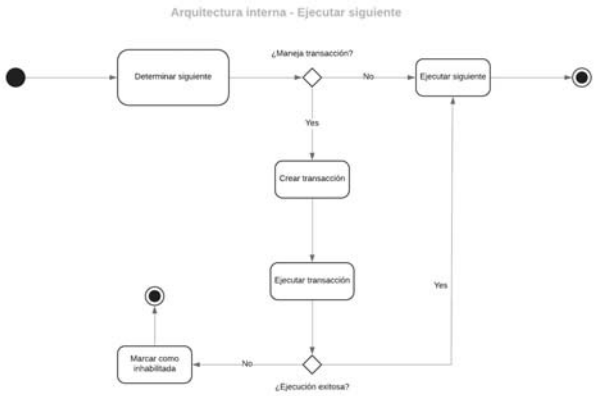

Figura 1. Diagrama de arquitectura de ejecución

El framework como vemos se encuentra desarrollado y funcionando[5], pero aún falta que el mismo permita dispositivos con menores capacidades (memoria, procesamiento, etc.) puedan participar de la ejecución de una coreografía.

3. Limitaciones derivadas de los dispositivos

Algunas características de los dispositivos ubicuos son específicas y perfectamente identificables, como es el caso de la movilidad o volatilidad de su conexión; no obstante, hay otras que no resultan tan claras (capacidad de procesamiento, memoria disponible, batería, etc.), lo que implica que se precisa hacer consideraciones especiales para poder contemplarlas dentro del framework desarrollado.

A continuación mostramos el cuadro 1 con un lista de algunos dispositivos y sus características, especialmente aquellos que tienen menos capacidad tanto de memoria como de procesamiento. Estos dispositivos representan únicamente algunos ejemplos de los que hemos trabajado, sin pretender hacer una lista exhaustiva.

En el caso de prueba seleccionado, el rol VehiculoAccidentado fue representado por una placa Arduino NANO, el rol Baliza fue caracterizado por una placa Arduino MEGA y el de CentralBalizas por un dispositivo Raspberry Pi B+, que se corresponden con los presentados en el Cuadro 1.

En este caso particular, la placa Arduino NANO, tiene tan solo el 12 % de la capacidad de memoria flash (almacenamiento de programa) de la que cuenta Arduino Mega, y el 25 % de la capacidad de almacenamiento de variables en memoria SRAM, esto se puede apreciar en la Figura 2. En un principio el código fuente del framework de ejecución de coreografías pudo ser almacenado en la memoria flash de la placa Arduino NANO, ocupando casi la totalidad de la

[5] El código fuente del framework desarrollado, puede ser accedido a través de esta dirección: `https://github.com/GRISE-UPM/ml_server_rest`.

Dispositivo	**Características distintivas**
Raspberry Pi B+	- Escaso nivel de procesamiento - Escasa capacidad de memoria
Arduino Mega	- Bajo nivel de procesamiento - Baja capacidad de memoria - Escasa conectividad - Dependencia de batería
Arduino NANO	- Escasísimo procesamiento - Escasisima capacidad de memoria - Escasa conectividad - Dependencia de batería
Dispositivos RFID	- Nula capacidad de procesamiento - Capacidad de memoria casi nula - Escasa conectividad - Dependencia de batería

Cuadro 1. Características de los dispositivos ubicuos

misma [6], por lo que no era aconsejable su ejecución con tan poca memoria libre disponible. Hubo que hacer adaptaciones para que finalmente ocupara el 81 % de la memoria flash disponible para programas.

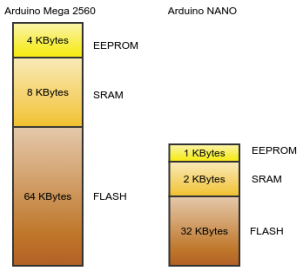

Figura 2. Comparación de memoria de dispositivos Arduino

Luego el otro inconveniente con el que hubo que realizar ajustes fue la memoria SRAM disponible y la necesaria para la ejecución del framework codificado. En una primera instancia el framework ocupaba el 85 % de la memoria disponible, por lo que su ejecución no era factible en absoluto, ya que esta memoria inicial es la que ocupan las variables globales, strings literales, objetos, etc., dejando únicamente un 15 % máximo de memoria libre para el almacenamiento de variables globales y punteros, esto puede verse en el Listado 1.1. Realizando un análisis exhaustivo del código fuente en C++, con el fin de reducir el tamaño de memoria utilizada por variables globales, strings y demás, se logró hacer una op-

[6] En este caso solamente son 30k, ya que 2k son utilizados por el bootloader.

timización del código de tal manera que se llegó a una utilización del 61 % (1260 bytes) de la totalidad de la memoria SRAM. Por lo tanto, quedan disponibles para utilización de variables locales y punteros un 39 %, es decir unos 788 bytes, puede observarse en el Listado 1.2

```
El Sketch usa 24888 bytes (81%) del espacio de almacenamiento de
programa
El maximo es 30720 bytes
Las variables Globales usan 1740 bytes (85%) de la memoria dinamica,
dejando 308 bytes para las variables locales
El maximo es 2048 bytes
```

Listado 1.1. 1ra. Compilación en entorno Arduino

```
El Sketch usa 24888 bytes (81%) del espacio de almacenamiento de
programa
El maximo es 30720 bytes
Las variables Globales usan 1260 bytes (69%) de la memoria dinamica,
dejando 788 bytes para las variables locales
El maximo es 2048 bytes
```

Listado 1.2. 2da. Compilación en entorno Arduino

Esta nueva versión del framework se cargó en la placa Arduino NANO y se intentó una ejecución del mismo, resultando insuficiente la memoria disponible para variables locales y punteros. Para superar esta situación, se realizaron simplificaciones del código para que pueda ser ejecutado en este tipo de placa. Estas simplificaciones implican que se tengan que introducir decisiones específicas para dicha placa de desarrollo.

Luego de realizadas estas pruebas, se pudo determinar ciertos límites para la ejecución de coreografías en dispositivos ubicuos, lo que se detalla a continuación:

- Los dispositivos deben contar con al menos 32 kbytes de memoria flash para poder almacenar de manera adecuada el código de ejecución del framework y con al menos 4 kbytes de memoria SRAM para el almacenamiento de las variables y punteros de programa.
- El motor de ejecución de coreografías puede ser ejecutado en placas Arduino Mega, equipos Raspberry Pi B+, pero no así en la placa Arduino NANO (cuenta solamente con 2 kbytes de memoria SRAM).

4. Análisis de las limitaciones de procesamiento y memoria

Las limitaciones derivadas de los dispositivos nos permitieron encontrar que la coreografía descripta en WS-CDL no puede ser utilizada en ciertos dispositivos, ya que para su interpretación se requiere de un cierto tamaño de memoria y una capacidad de procesamiento determinada. Ahora bien, los factores que producen que la descripción en WS-CDL necesite determinadas capacidades de los dispositivos ubicuos radica principalmente en el formato en el cual se encuentra especificada; esto es, XML. La especificación XML consume un espacio adicional de memoria, el cual justo es escaso en este tipo de dispositivos; además, el

procesamiento e interpretación de la estructura XML necesita un rendimiento de procesador considerable. A modo de ejemplo, la especificación completa de la coreografía utilizada en el escenario de pruebas tiene una longitud de 8 kbytes, lo que representa cuatro veces más de memoria de lo que cuenta una placa Arduino Nano y el doble de los 4 kbytes necesarios para la ejecución del framework. Es preciso recordar que el escenario propuesto es reducido respecto de otras especificaciones que se pueden dar en ambientes de producción, o en el mismo caso del escenario planteado pero en un ambiente real.

Para poder implementar en el framework la ejecución de la coreografía definida en el lenguaje WS-CDL, es necesario que se hagan simplificaciones de la especificación, ya que la misma es muy extensa. Una alternativa es que éstas simplificaciones sean realizadas con anterioridad y pre-cargadas en el dispositivo (de forma estática); y por otro lado, que las mismas se hagan a demanda en tiempo real. Hemos implementado en el framework la simplificación pre-cargada, pero para una mayor independencia tanto de los dispositivos como de la coreografía en su conjunto, se debe implementar la segunda alternativa, que es la lectura a demanda de la especificación. En el primero de los casos, la simplificación pre-cargada, el ahorro de memoria ha llegado a más de un 50 %, ya que de los 8 kbytes necesarios solo por la especificación en XML, se necesitan 4 kbytes para la totalidad de le ejecución (esto incluye variables y punteros que hacen al procesamiento propiamente dicho), lo cual puede ser visualizado en el Listado 1.3 donde podemos apreciar el resultado de una compilación en el entorno Arduino. En el segundo caso, de solicitar la especificación o parte de la misma a demanda, los valores se reducirían aún más, pero con la necesidad de hacer mayor uso de las comunicaciones para poder solicitar a un nodo centralizador o a otros dispositivos con mayor capacidad los trozos de especificación necesarios para la ejecución.

Además de las limitaciones de memoria planteadas hasta este punto, nos encontramos con aquellos dispositivos que no tienen capacidad de procesamiento, como es el caso de tarjetas RFID. Éstos dispositivos también deben ser considerados para ser parte de la ejecución de una coreografía, dentro del framework desarrollado, pero para ello se deben hacer consideraciones especiales. Estos dispositivos al contar únicamente con la posibilidad de almacenar información y no poseer capacidad de procesamiento no pueden ejecutar de manera convencional la coreografía, pero sí pueden ser el vínculo para que otros dispositivos puedan encontrar información que no pueden almacenar en su memoria debido a la escasa cantidad como planteamos anteriormente. Para ser más específicos, estos dispositivos que solamente tienen la capacidad de almacenar información podrían contener la definición de la coreografía (o parte de ella) en formato XML para que otros dispositivos con escasa capacidad de memoria puedan solicitar dicha información para poder continuar con la ejecución, tal como se plantea en el párrafo anterior.

```
El Sketch usa 25368 bytes (9%) del espacio de almacenamiento de
programa
El maximo es 253952 bytes
Las variables Globales usan 2055 bytes (25%) de la memoria dinamica,
dejando 6137 bytes para las variables locales
```

```
El maximo es 8192 bytes
```

Listado 1.3. Compilación en entorno Arduino

5. Conclusiones y Futuras Líneas

El framework para dispositivos ubicuos en ambientes pervasivos obtenido cumple con la propuesta inicial de trabajo, el cual está basado en los conceptos de SOA, que implementa la especificación WS-CDL. Esta solución a su vez cuenta con ciertas ventajas por sobre el resto de los frameworks existentes, siendo éste abierto, estándar, extensible e interoperable. En contrapartida tiene ciertas desventajas como es el caso de las limitaciones de memoria y de requerimientos de capacidad de procesamiento que hacen que el framework deba ser mejorado y ampliado para dar soporte a estas falencias.

Más específicamente, determinamos que nuestra propuesta requiere un mínimo de 4 Kbytes de memoria SRAM y 32 Kbytes de memoria Flash, para ser ejecutada en dispositivos ubicuos y una capacidad de procesamiento de al menos 8 bits a 16Mhz. Estos requerimientos necesarios para la ejecución del framework permiten que el mismo pueda ser utilizado en placas del estilo Arduino Mega o equipos como Raspberry Pi B+. No obstante, estas mismas características mínimas limitan el uso de por ejemplo placas de desarrollo Arduino Nano (y todas aquellas que posean igual o menor capacidad) y dispositivos RFID.

Sin embargo, en base a las adaptaciones propuestas en la Sección 4, donde hablamos de reducciones de la especificación podríamos incluir los dispositivos que han quedado excluídos por escasa capacidad de memoria o procesamiento como puede ser el caso de placas Arduino Nano, dispositivos RFID, wereables, etc.

Como futuras líneas de trabajo avizoramos la necesidad de ahondar en la búsqueda de alternativas de mejora para las soluciones propuestas (reducciones ad-hoc y peticiones a demanda) sobre la interpretación de la especificación en XML de la coreografía. De este modo, nuestro framework puede ser utilizado por dispositivos de menor capacidad antes mencionados y similares.

Agradecimientos

Esta investigación fue financiada en parte por el rubro de la Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia (CEDIA) para el Grupo de Trabajo: "Internet de las Cosas y Ciudades Inteligentes" y en parte por el rubro de la Universidad de las Fuerzas Armadas ESPE del Ecuador para estancias de investigación.

Referencias

1. M. Weiser, "Hot topics-ubiquitous computing," *Computer*, vol. 26, pp. 71–72, Oct 1993.

2. Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Information Sciences*, vol. 280, no. 0, pp. 218–238, 2014.
3. J. Harkes, T. Farbacher, and N. Miller, "Project Aura Distraction-free Ubiquitous Computing." Available at `http://www.cs.cmu.edu/~./aura/people.html`, Last visited: Aug 15th, 2002.
4. R. H. Campbell, D. M. Mickunas, D. Reed, and K. Nahrstedt, "Active Spaces for Ubiquitous Computing." Available at `http://gaia.cs.illinois.edu/`, Last visited: Aug 15th, 2002.
5. M. L. f. C. Science and M. A. I. Laboratory, "Pervasive Human-Centered Computing." Available at `http://oxygen.csail.mit.edu/`, Last visited: Aug 15th, 2002.
6. "Amigo Project." Available at `http://gforge.inria.fr/projects/amigo/`.
7. M. Viroli, "On competitive self-composition in pervasive services," *Science of Computer Programming*, vol. 78, no. 5, pp. 556–568, 2013. Special section: Principles and Practice of Programming in Java 2009/2010 & Special section: Self-Organizing Coordination.
8. S. Najar, M. K. Pinheiro, and C. Souveyet, "A New Approach for Service Discovery and Prediction on Pervasive Information System," *Procedia Computer Science*, vol. 32, pp. 421–428, 2014. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).
9. S. W. Loke, "Supporting ubiquitous sensor-cloudlets and context-cloudlets: Programming compositions of context-aware systems for mobile users," *Future Generation Computer Systems*, vol. 28, no. 4, pp. 619–632, 2012.
10. F. Palmieri, "Scalable service discovery in ubiquitous and pervasive computing architectures: A percolation-driven approach," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 693–703, 2013. Special Section: Recent Developments in High Performance Computing and Security.
11. G. M. Oscar A. Testa, Efraín R. Fonseca C. and O. Dieste, "Coordination of ubiquitous devices in pervasive environments: A proposal based on ws-cdl," in *XXXVIII International Conference of the Chilean Computer Science Society (SCCC 2019)*, p. en proceso de publicación, November 2019.
12. M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley Indianapolis, 2008.
13. G. Pardon and C. Pautasso, "Atomic distributed transactions: a restful design," (Seoul, Korea), ACM, April 2014.
14. G. Cassar, P. Barnaghi, W. Wang, S. De, and K. Moessner, "Composition of services in pervasive environments: A Divide and Conquer approach," in *Computers and Communications (ISCC), 2013 IEEE Symposium on*, pp. 000226–000232, July 2013.