Coordination of Ubiquitous Devices in Pervasive Environments: A Proposal Based on WS-CDL

Oscar A. Testa*, Efraín R. Fonseca C.†, Germán Montejano‡, and Oscar Dieste§
*Facultad de Ciencias Exactas y Naturales, Universidad Nacional de La Pampa. Santa Rosa, Argentina
Email: otesta@exactas.unlpam.edu.ar
†Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE. Quito, Ecuador
Email: erfonseca@espe.edu.ec
‡Facultad de Ciencias Físico Matemáticas y Naturales, Universidad Nacional de San Luis. San Luis, Argentina
Email: gmonte@unsl.edu.ar
§Escuela Técnica Superior de Ing. Informática, Universidad Politécnica de Madrid. Madrid, España
Email: odieste@fi.upm.es

***Resumen*—*Context:* Ubiquitous devices are usual in our daily lives. We are permanently interacting with those devices, and with the services they offer. *Problem:* Ubiquitous devices do not provide services in isolation; they cooperate with other devices. Cooperation mechanisms are primarily proprietary. The few proposals from academy have hardly achieved an impact in practice. This contrasts with web services in the Internet environment, which have standardized mechanisms for the composition of services. *Aim:* We explore the application of Service Oriented Architecture (SOA) specifications and standards for the coordination of ubiquitous devices in pervasive environments. *Methodology:* We apply Design Science. This methodology allows the iterative construction and evaluation of artifacts. *Results:* A coordination framework based on WS-CDL has been built for ubiquitous devices, along with a proof of concept that demonstrates that the framework works properly in a laboratory environment. The transition to real environments looks straightforward.**

***Index Terms*—Software Engineering; SOA; Ubiquitous devices; Services; Choreography Services**

I. Introducción

Dispositivos ubicuos son aquellos 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].

La computación ubicua es un paradigma tecnológico que pretende que las computadoras no se perciban como objetos diferenciados; así como también, que su utilización sea lo más transparente y cómoda posible para las personas, en las diversas situaciones en que existe interacción [1].

Los avances tecnológicos (especialmente de hardware y comunicaciones) han permitido que los dispositivos ubicuos sean al mismo tiempo generadores y consumidores de servicios; es decir, de acuerdo a las capacidades del dispositivo, éste pueda no solo obtener, sino también ofrecer su funcionalidad; lo que se traduce en un ambiente de cooperación entre dispositivos, permitiendo componer funcionalidades más complejas. Por composición, entendemos la forma en que los dispositivos ubicuos se pueden combinar para realizar una tarea determinada. La composición implica que los dispositivos deben comunicarse entre ellos con la finalidad de obtener un servicio con valor agregado, lo que a su vez conlleva desafíos tales como: tolerancia a fallas, escaso nivel de procesamiento, problemas de conectividad, por mencionar algunos ejemplos [2].

Si bien hoy en día podemos decir que distintos dispositivos se pueden comunicar entre ellos, compartiendo de alguna manera sus servicios, generalmente lo realizan a partir de protocolos propietarios y sin seguir definiciones estándar, provocando que otros dispositivos no puedan ser integrados para la comunicación. Esto representa una importante limitación en la composición de dispositivos ubicuos [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. Todas estas dificultades hacen que la composición de dispositivos ubicuos constituya un área de investigación importante, donde los avances no son claros al día de hoy [2].

Esta situación nos motivó a plantear una propuesta que aglutine las especificaciones y estandarizaciones existentes en SOA para la coordinación de dispositivos ubicuos. Si pensamos que cada dispositivo ubicuo es proveedor o consumidor de un servicio, encaja perfectamente en la arquitectura de servicios.

Para poder llevar adelante este tipo de investigación, y de acuerdo a la naturaleza misma de la solución que intentamos encontrar, nos basamos en la metodología de investigación Design Science, ya que la misma se basa en la construcción y evaluación de un artefacto con la intención de dar solución a problemas debidamente identificados [3].

En este caso nos proponemos desarrollar un framework que nos permita ejecutar coreografías con dispositivos ubicuos. Por coreografía entendemos que los dispositivos se comunican de a

978-1-7281-5613-2/19/$31.00 ©2019 IEEE

pares sin un coordinador central para llevar adelante una tarea determinada. Siguiendo los lineamientos de la metodología, el framework será construido de forma evolutiva. Cada nuevo ciclo se basa en la evaluación de los resultados obtenidos en el ciclo previo, aplicando las mejoras necesarias o pertinentes.

La contribución consiste en la construcción de un framework de ejecución de coreografías en ambientes pervasivos a través de la utilización de dispositivos ubicuos. Este framework, por lo tanto, al estar basado en SOA es estandarizado, lo que conlleva a que se evite el problema en la heterogeneidad de dispositivos, además de permitir la utilización de características más avanzadas (dependiendo de la potencia y capacidad que posea el dispositivo) como es el caso de WS-Transaction, WS-Security, etc.

Lo que resta del artículo se estructura de la siguiente forma: en la sección 2 se describen los antecedentes y el estado de la cuestión de los sistemas ubicuos, con énfasis en la coordinación inherente. En la sección 3 se describe SOA como mecanismo de coordinación de servicios. En la sección 4 se detalla la metodología de investigación seleccionada, de la mano de las preguntas de investigación planteadas. En la sección 5 se describe el framework de coordinación desarrollado. Una prueba de concepto llevada adelante para poder implementar el Framework desarrollado, es descrita en la sección 6. Finalmente, en la Sección 7, se discuten las conclusiones y trabajos futuros.

II. Antecedentes

II-A. Sistemas ubicuos

A finales de la década de los 80 y principios de los 90, Mark Weiser introdujo el término de computación ubicua (también utilizó el acrónimo "ubicomp") [1]. La teoría de Weiser postula que la computación ubicua tiene como propósito mejorar el uso de las computadoras, haciéndolas disponibles en el entorno físico, a través de una multiplicidad de elementos. Por ejemplo, brazaletes que controlan la posición de una persona anciana y activa alarmas en caso de riesgo, paraguas que nos avisan cuándo es necesario utilizarlos, etc. El problema de esta teoría fue que los protocolos de comunicación existentes, no se alineaban con ella y debieron ser mejorados, en especial para permitir la movilidad de los dispositivos [1]. Los equipos disponibles en la actualidad poseen mejores características en lo que respecta por ejemplo al procesamiento, aunque con ciertas limitaciones; por ejemplo, la vida útil de la batería, disponibilidad de conexión a redes, etc. A partir de estos principios de la computación ubicua es que se comenzó a trabajar sobre el desarrollo de entornos que involucren este tipo de dispositivos.

Se han realizado diversos proyectos de entornos ubicuos como **Aura** [4] de la Universidad de Carnegie Mellon, cuyo objetivo principal es el de proveer a cada usuario con un halo invisible de servicios de información, más allá del lugar que se encuentre; **Gaia** [5] de la Universidad de Illinois donde se plantean que los espacios físicos se convierten en espacios activos a partir de la utilización de dispositivos ubicuos y proponen un sistema operativo para manejar todos estos elementos en conjunto; **Oxigen** [6] perteneciente al MIT (Massachusetts Institute of Technology) donde se trabaja para que los dispositivos sean incorporados en la vida humana y cotidiana de manera natural e imperceptible.

Si bien estos proyectos han sido un avance para la integración de los dispositivos, no se ha logrado hacer que los mismos trabajen de forma independiente y colaborativa para la obtención o realización de una tarea específica, sin necesidad de ser coordinados a través de un nodo central.

II-B. Frameworks de desarrollo para la coordinación de dispositivos

Para la coordinación de dispositivos (o sistemas) ubicuos existen algunos mecanismos, como por ejemplo los frameworks de desarrollo. A continuación se describen algunos de estos mecanismos.

Amigo [7] proveniente de Amigo Consortium, el cual se encuentra disponible como software libre en [7], es un framework que provee una capa de middleware que permite la integración de dispositivos y servicios heterogéneos en un ambiente de red o distribuido; **OSGi** es una especificación de un framework de componentes que aporta modularidad a la plataforma Java. Permite además, la creación de módulos altamente cohesivos y poco acoplados que pueden integrarse en aplicaciones más grandes. Además, cada módulo puede desarrollarse, probarse, implementarse, actualizarse y administrarse individualmente con un impacto mínimo o nulo en los otros módulos. **Android Studio** es una plataforma que provee todo lo necesario para desarrollar aplicaciones para Android. Esta plataforma permite el desarrollo de aplicaciones wereables, las cuales pueden ejecutarse directamente en dispositivos wereables, permitiendo el acceso al hardware de bajo nivel como son los sensores, actividades, servicios y más.

Si bien estos frameworks han presentado un avance respecto de la integración de los dispositivos ubicuos con otros elementos de computación, no son suficientes para que los mismos puedan interactuar de manera independiente.

II-C. Propuestas académicas para la coordinación de dispositivos

Existen actualmente también trabajos de investigación relacionados con la coordinación de dispositivos ubicuos en ambientes pervasivos.

Viroli [8] en su trabajo promueve la aplicación de técnicas de auto-organización a la coordinación de servicios de software en el contexto de entornos altamente dinámicos y móviles. Para lograrlo describe un framework donde se plantea que exista una red de dispositivos donde éstos administran los servicios proporcionados, permitiendo de esta manera una composición totalmente distribuida y auto organizada. Esto se logra basándose en un conjunto de leyes de coordinación estructuradas similares a las reacciones químicas. Si bien el trabajo se orienta en la forma de coordinar los servicios, solamente toma en cuenta los dispositivos móviles pero sin mencionar dispositivos con muchas menos prestaciones y

posibilidades de conexionado bajo protocolos ya establecidos como es el caso de sensores, dispositivos biométricos, etc.

Los trabajos de Najar & otros [9] y Loke [10] presentan alternativas para seleccionar, descubrir y/o validar que los servicios o dispositivos sean adecuados para la composición o coordinación que se pretende realizar. No se hace mención a la manera en que pueden colaborar entre ellos estos dispositivos, es decir, el protocolo que utilizan para comunicarse. Además aún está faltando un framework que de soporte a estas nuevas alternativas de descubrimiento, validación y selección de dispositivos.

En el trabajo de Palmieri [11] se abandona la manera de trabajo centralizada (orquestaciones) por métodos más colaborativos y distribuidos, en concreto infraestructuras de pares. Si bien este trabajo se aleja de los enfoques centralizados, no es adecuado para cualquier tipo de dispositivos ya que se basa para la comunicación en las redes de celulares.

Existen otros trabajos sobre composición de servicios en otras áreas como es el caso de sistemas embebidos, actuadores o red de sensores, como es el caso de los trabajos de [12], [13], [14], [15], donde plantean la necesidad de conexión de una manera no jerárquica evidenciando las ventajas de otras formas de conectividad, como es el caso de las coreografías.

II-D. Problemas asociados a la coordinación de dispositivos ubicuos

Si bien se han producido avances en la composición e integración de dispositivos ubicuos, estos esfuerzos no han logrado alcanzar que los dispositivos se coordinen entre ellos sin la necesidad de un coordinador central; más aún, no se basan en mecanismos estandarizados para tal fin.

Además, los estudios revisados trabajan sobre un universo acotado de dispositivos, o bien se basan sobre la idea de la coordinación dinámica (cómo ubicar los servicios en tiempo de ejecución), pero se habla poco acerca del protocolo de composición.

Debido a que los dispositivos ubicuos tienen distintas limitantes como son la cantidad de memoria disponible, la durabilidad de la batería, la disponibilidad de acuerdo a la red del lugar donde se encuentre en un momento determinado, la disponibilidad y confiabilidad de los mismos no puede ser garantizada. Sin embargo, no se han realizado avances donde los dispositivos puedan dialogar entre ellos de una manera abierta, estándar y dando soporte a estas características propias de dichos dispositivos.

Todas estas dificultades hacen que la composición de dispositivos se transforme en un área de investigación muy importante donde los avances no han sido claros al día de hoy [2].

III. Metodología de Investigación

La computación orientada a servicios, y en particular los servicios web, proporcionan mecanismos para la composición de servicios. Las orquestaciones, por ejemplo, son mecanismos bien conocidos que permiten construir sistemas de negocio complejos a partir de una gran cantidad de servicios heterogéneos, simples y distribuidos.

Nuestra propuesta es *aplicar las especificaciones y estandarizaciones existentes en SOA para la coordinación de dispositivos ubicuos en ambientes pervasivos*. Si pensamos que cada dispositivo ubicuo en un ambiente pervasivo es proveedor, o consumidor de un servicio, la coordinación de dispositivos parece encajar perfectamente con la composición de servicios. Sin embargo, en contextos tales como el de Internet de las cosas (IoT de las siglas del inglés Internet of Things), donde los servicios son dinámicos, móviles, menos fiables y dependientes del dispositivo, los mecanismos de composición establecidos para servicios web no son directamente aplicables [16].

Por otra parte, la composición en ambientes pervasivos, como es el caso de las redes de sensores, dispositivos *wearables*, etc., necesita hacer frente a las distintas contingencias que pueden ocurrir con estos elementos, sin descuidar la heterogeneidad de los mismos. No debemos olvidar, que los dispositivos en mención tienen distintas limitantes como son la cantidad de memoria disponible, la durabilidad de la batería, la disponibilidad de acuerdo a la red del lugar donde se encuentre en un momento determinado. Por tanto, en ambientes pervasivos, la disponibilidad y confiabilidad de los dispositivos no puede ser garantizada.

Si bien las semejanzas entre la tecnología SOA y su aplicabilidad a sistemas ubicuos es muy importante, como mencionábamos, la mera traslación de los conceptos no alcanza. Para ello es necesario adaptar los conceptos de SOA para poder dar lugar a soportar las características propias de los dispositivos ubicuos. Es sorprendente que esta similitud no haya sido apenas explorada con anterioridad. Únicamente en el trabajo de Sheng [2] se hace mención a la necesidad de más investigación en esta área. En miras de alcanzar este propósito, nos hemos planteado las siguientes preguntas de investigación:

1. ¿Es SOA capaz de actuar como mecanismo de coordinación de sistemas ubicuos?.
2. ¿Es posible desarrollar un framework que permita incorporar SOA a dispositivos ubicuos?.
3. ¿Es posible demostrar la aplicabilidad de SOA en ambientes ubicuos mediante una prueba de concepto?.

De acuerdo a la naturaleza del problema y en base al objetivo planteado en el presente trabajo de investigación, la metodología seleccionada para llevar adelante el proyecto es "Design Science". Design Science crea y evalúa artefactos de tecnologías de la información con la intención de dar solución a problemas debidamente identificados, tal como es expresado en [3]. Los artefactos que se crean o evalúan a partir de Design Science van desde software, lógica formal y matemática rigurosa, hasta descripciones informales en lenguaje natural.

Otras metodologías podrían ser igualmente aplicables, como es el caso de Action Resarch. No obstante, Action Research no aborda todas las aristas de esta investigación. Más concretamente, Action Research persigue estudiar propuestas de solución a problemáticas planteadas en un contexto específico, de forma evolutiva; no obstante, esta estrategia no considera

ni una evaluación explícita de los artefactos construidos, ni el hecho de que esta investigación no se produce en un contexto específico, sino es general. Otras metodologías, como Case Study, están todavía más alejadas del tipo de investigación planteado en este estudio.

En la Figura 1 podemos apreciar las fases que plantea la metodología Design Science asociados a las preguntas de investigación planteadas.

A continuación describimos brevemente cada una de dichas actividades:

- En primer lugar se **identifica la problemática** a ser abordada y la motivación para realizar el trabajo de investigación, para ello se hizo un estudio de literatura exhaustivo sobre la problemática y la relevancia de la solución
- Como segunda actividad, se **definen los objetivos**. Para definir el objetivo nos basamos fundamentalmente en el estado de la cuestión, y también en la falta de investigación en la materia.
- Una vez establecido claramente el problema a investigar, junto a los objetivos y preguntas de investigación, se procedió a trabajar sobre la actividad de **diseño y desarrollo del artefacto**, que en este caso se trató tanto de la adecuación de las especificaciones existentes en SOA y coreografías para el caso particular de dispositivos ubicuos, como también en la producción de un framework de ejecución de coreografías en ambientes ubicuos.
- Dentro de la actividad de **demostración**, se implementó el framework de ejecución de coreografías en ambientes ubicuos a un caso de estudio o escenario de trabajo.
- Finalmente, se trabajó en distintas **evaluaciones** y evoluciones del prototipo planteado inicialmente, que de forma cíclica desde la actividad 3 se determinó el artefacto final.

IV. SOA como mecanismo de coordinación

En esta sección trataremos los mecanismos de coordinación disponibles en SOA, lo cual se asocia a la primer pregunta de investigación que nos planteamos: *¿Es SOA capaz de actuar como mecanismo de coordinación de sistemas ubicuos?.*

IV-A. Conceptos

La arquitectura orientada a servicios (SOA de las siglas del inglés Service Oriented Architecture) permite la creación de sistemas de software escalables que reflejan el negocio de una organización, brindando además una forma bien definida de cómo exponer e invocar los servicios, lo que facilita la integración e interacción de sistemas tanto propios como de terceros.

De acuerdo con Georgakopoulos y Papazoglou [17], para proveer la descomposición de la funcionalidad de una aplicación en servicios, SOA requiere que los mismos sean: Autónomos, independientes de la plataforma y el descubrimiento, y dinámicos en lo concerniente a invocación y composición.

Los bloques de construcción de SOA se basan en tres roles primarios. Estos son: **Proveedor del servicio** (agentes de software que proveen el servicio, responsables de publicar una descripción del servicio en el servicio de registración); **Registro del servicio** y **Cliente del servicio** (agentes de software que hacen solicitudes de ejecución del servicio). Los clientes deben ser capaces de encontrar la descripción de los servicios que ellos necesitan o buscan y deben ser capaces de encontrarlos y poder invocarlos.

IV-B. Servicios Web

El World Wide Web Consortium (W3C) define un servicio web como un sistema de software designado para dar soporte a la interacción de máquina a máquina interoperativa a través de una red [18]. Es esencialmente una abstracción semánticamente bien definida de un conjunto de actividades computacionales o físicas involucrando un número de recursos, con la intención de completar una necesidad o requerimiento de negocio [19].

Un servicio web realiza una tarea específica o un conjunto de tareas, y se describe mediante una especificación de servicio en una notación XML estándar llamada WSDL (de las siglas del inglés Web Services Description Language). La WSDL proporciona los detalles necesarios para interactuar con el servicio, incluidos los formatos de mensaje (que detallan las operaciones), los protocolos de transporte y la ubicación [20].

La interfaz WSDL oculta los detalles de cómo se implementa el servicio, y el servicio se puede utilizar independientemente de la plataforma de hardware o software en la que se implementa e independientemente del lenguaje de programación en el que está escrito.

Los servicios Web se han convertido en la tecnología preferente para la implementación SOA. El éxito de esta tecnología radica en basar su desarrollo sobre infraestructuras ya existentes como lo son HTTP, SOAP y XML [17]. A principios del año 2000 surgió REST (de las siglas del inglés Representational State Transfer), una nueva forma de implementar servicios web. REST se sustenta, al igual que SOA, en infraestructuras ya existentes como XML y HTTP, pero sin utilizar la capa SOAP.

La tecnología de microservicios se fundamenta en los servicios web basados en REST. Un microservicio se define como un proceso independiente y cohesivo interactuando a través de mensajes. La arquitectura de microservicios por su parte se define como una aplicación distribuida donde todos sus módulos son microservicios [21]. Esta arquitectura tecnológica no está ligada a ningún lenguaje de programación en particular, sólo provee una línea base de cómo distribuir los componentes de una aplicación distribuida.

IV-C. Coordinación en SOA

Un desafío clave para SOA y la tecnología de servicios web es la composición de servicios. Para las aplicaciones distribuidas orientadas a servicios, los servicios son la piedra angular. Las aplicaciones distribuidas se desarrollan a partir de la combinación, composición e interacción entre servicios. El verdadero potencial de SOA puede ser explotado únicamente a través de la composición de servicios. Más específicamente, la composición de servicios web es un proceso de agregación de varios servicios en uno solo, con el propósito de que dicho

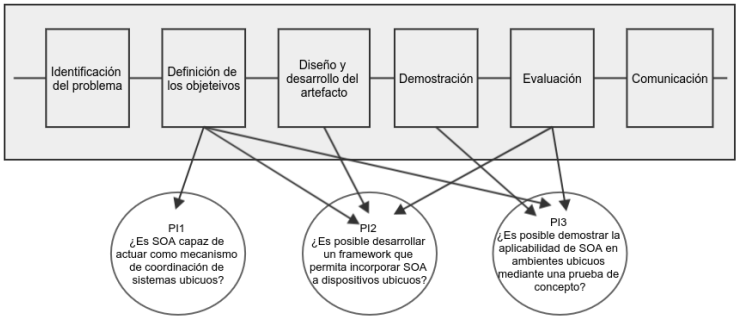

Figura 1. Metodología de investigación

servicio pueda realizar una tarea más compleja, lo que a su vez significa tener aplicaciones mejores [17].

Existen dos formas de describir las actividades que conforman un proceso de negocio: orquestación y coreografía de servicios:

- **Orquestación**: Describe cómo interactúan los servicios en un flujo de trabajo, incluyendo la lógica del negocio y el órden lógico de las interacciones. Un lenguaje de orquestación de servicios es BPEL o WSBPEL para la orquestación de servicios web [19].
- **Coreografía**: Describe la secuencia de mensajes entre servicios, centrándose en el intercambio público de mensajes y el estado de la conversación. A diferencia de la orquestación, que se muestra desde la perspectiva de un coordinador principal, la coreografía se centra en el intercambio de mensajes desde la perspectiva de un observador externo. Cada servicio involucrado en la coreografía debe tener en cuenta los procesos de negocio, decidir cuándo ejecutar sus operaciones y cómo debe interactuar con otros servicios [19]. Dentro de las especificaciones existentes de coreografías nos encontramos con WS-CDL (Web Service Choreography Description Language) y WSCI (Web Service Choreography Interface).

Tanto en OASIS (consorcio sin fines de lucro que impulsa el desarrollo, la convergencia y la adopción de estándares abiertos para la sociedad de la información global) como W3C han trabajado en la especificación de lenguajes de ejecución de orquestaciones y de coreografías. La composición a través de orquestaciones está bien definida y existen muchas implementaciones de la misma, pero no así de las coreografías, de las cuales no se ha avanzado siquiera en el plano de SOA.

En nuestro trabajo nos abocamos a las coreografías, donde hemos trabajado sobre la implementación de un framework de ejecución de coreografías basado en el lenguaje de especificación WS-CDL. Es un lenguaje basado en XML que describe la colaboración entre pares peer-to-peer, mediante la definición, desde un punto de vista global, de los comportamientos comunes y observables de cada participante de un proceso de negocio.

V. Framework de Coordinación

En esta sección nos centraremos en el framework desarrollado, el cual responde a la segunda pregunta de investigación planteada: *¿Es posible desarrollar un framework que permita incorporar SOA a dispositivos ubicuos?*.

El framework de ejecución de coreografías[1] se construyó en base a los lenguajes de programación PHP y C++. Estos lenguajes 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 (del inglés Web Service Choreography Description Language) 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 para luego poder determinar cuál o cuáles son los pasos siguientes en la ejecución. 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 2. Allí se presentan los pasos que se llevan adelante dentro o desde un dispositivo que compone la coreografía, donde en primer lugar se determina cuál o cuáles son los pasos siguientes y los va ejecutando según corresponda. Como vemos la secuencia de ejecución es simple, clara y aplicable a cualquier dispositivo.

Para comprender mejor la estructura del framework desarrollado, mostraremos como ejemplo, un código escrito en el lenguaje de especificación de coreografías WS-CDL donde se aprecia la interacción entre dos dispositivos que forman parte de la coreografía [2]. Desde el primer dispositivo denominado

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

[2]Solamente se muestra un trozo de la totalidad de la especificación XML, a los fines prácticos

VehiculoAccidentadoRole se produce una comunicación con otro dispositivo BalizaRole, éste último a su vez se relaciona con otro dispositivo CentralBalizaRole. Esta coordinación se pueden apreciar en las dos interacciones definidas. Si observamos en la Figura 2, el paso *Determinar siguiente* hace la tarea de fijarse en la definición XML de la coreografía a qué dispositivo debe llamar, en el ejemplo que mostramos debería leer la interacción que corresponde y fijarse cuál es el dispositivo que figura como *toRole*; si esto fuese el caso de la primer interacción del listado, debería invocar o coordinar con el dispositivo que se denomina BalizaRole.

```
<interaction name="reportarAccidente" operation="informarIncidente">
  <participate relationshipType="tns:Vehiculo_Baliza" fromRole="
        tns:VehiculoAccidentadoRole" toRole="tns:BalizaRole" />
  <exchange action="request" name="informarIncidente" informationType="
        tns:avisoIncidenteType">
    <send variable="cdl:getVariable(tns:DatosIncidente,VehiculoAccidentadoRole)"/>
    <receive variable="cdl:getVariable(tns:DatosIncidente,BalizaRole)"/>
  </exchange>
</interaction>

<interaction name="publicarAccidente" operation="publicarIncidente">
  <participate relationshipType="tns:Baliza_CentralBaliza" fromRole="tns:BalizaRole
        " toRole="tns:CentralBalizasRole" />
  <exchange action="request" name="informarIncidente" informationType="
        tns:avisoIncidenteType">
    <send variable="cdl:getVariable(tns:DatosIncidente,BalizaRole)"/>
    <receive variable="cdl:getVariable(tns:DatosIncidente,CentralBalizasRole)"/>
  </exchange>
</interaction>
```

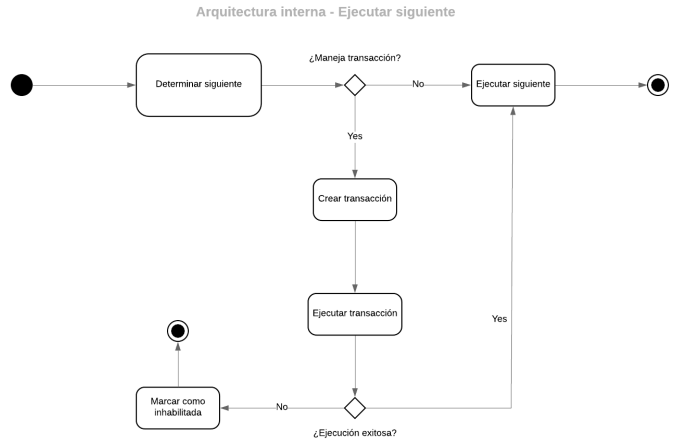

Figura 2. Diagrama de arquitectura de ejecución

A continuación mostramos cómo se implementan los pasos de la ejecución de la coreografía, descriptos en la figura 2, pero ya en el código propiamente dicho. En la figura 3 se pueden apreciar los diagramas de clases en C++; éstas clases son las encargadas de llevar adelante la ejecución de la coreografía. En este diagrama podemos apreciar también que se encuentran ya implementadas las clases que dan manejo a una transacción dentro de la ejecución.

Para poder hacer uso del framework para un caso específico se deben incluir las librerías que implementan las clases dentro del código específico de la coreografía que se desea llevar adelante. Además, se deben generar clases que heredan de las clases preexistentes en el framework. A continuación se muestra un template de una clase que implemente y haga uso del framework. Más específicamente, Se deben incluir las librerías connection y RESTWebServer, y se deben im-

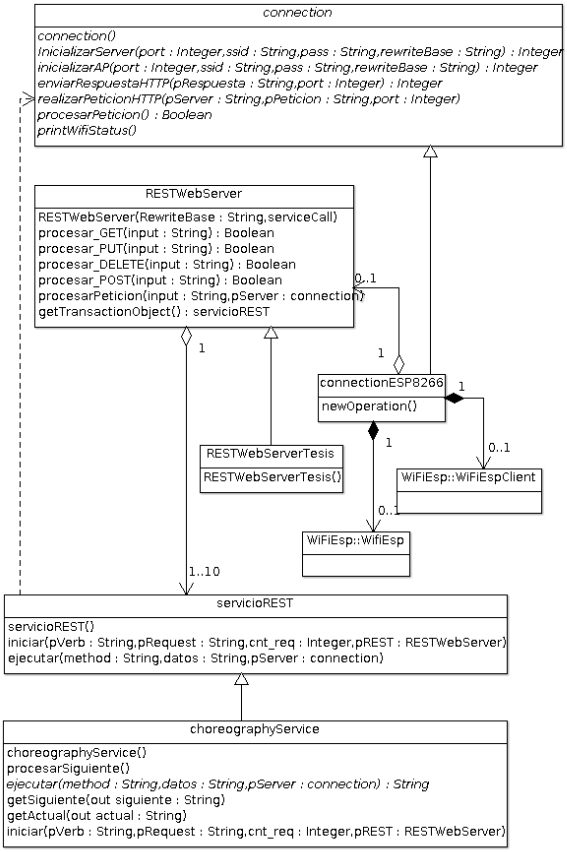

Figura 3. Diagrama de clases para ejecución de coreografías

plementar las clases constructoras, iniciarlas y ejecutarlas. En este caso mostramos el código que implementa el dispositivo BalizaRole, el cual es parte de las interacciones mostradas en la definición previa.

```
#include "ClaseX.h"
#include <connection.h>
#include <RESTWebServer.h>

BalizaRole::BalizaRole() : choreographyService(){
    chor_act_role = "BalizaRole";

    return;
}

int BalizaRole::iniciar(){
  return choreographyService::iniciar();
}

void BalizaRole::ejecutar(){
  if (strcmp(method,"informarIncidente") == 0){
    if (this->_verb != 'O'){
      pServer->enviarRespuestaHTTP("405_Method_not_allowed","");
      return;
    }
    String respuesta = this->informarIncidente(method,datosIncidente);
    strcpy(sRespuesta,"{\"resultado\":\"");
    strcat(sRespuesta,respuesta.c_str());
    strcat(sRespuesta,"\",\"token\":");
    strcat(sRespuesta,_token);
    strcat(sRespuesta,"}");
    pServer->enviarRespuestaHTTP("200_OK",sRespuesta);

    // LLamo a que se ejecute lo que tenga que seguir de la coreografia
    choreographyService::ejecutar();
}

}
```

El template mostrado corresponde a la programación en lenguaje C++; no obstante, la programación en PHP es muy similar a la presentada, incluso al ser un lenguaje de más alto

nivel, su implementación es mucho más simple.

VI. Prueba de concepto

En esta sección nos centraremos en la prueba de concepto que realizamos para poder implementar el framework presentado, que a su vez responde a la tercera pregunta de investigación que nos planteamos: *¿Es posible demostrar la aplicabilidad de SOA en ambientes ubicuos mediante una prueba de concepto?*.

Para validar la funcionalidad del framework de coordinación, se ha seleccionado una prueba de concepto basada en un escenario de trabajo específico; no obstante, creemos que esta solución podría ser extrapolada hacia prácticamente cualquier ambiente de trabajo.

Se planteó un ambiente inteligente relacionado directamente con los problemas referentes al tráfico vehicular a lo largo de las autopistas, rutas o carreteras. Simulamos vehículos que transitan por una carretera, los cuales se comunican con balizas distribuidas a lo largo de la vía. Esta comunicación puede ser bi direccional, cuando el vehículo informa de un problema encontrado o bien que la baliza da aviso de posibles problemas a lo largo de la carretera. A su vez las balizas pueden comunicarse con centrales de coordinación o de emergencia según corresponda. Una aplicación real de este escenario podría ser que un ómnibus con pasajeros se traslada desde una ciudad a otra a través de una carretera normal (con dos carriles), el conductor del vehículo cuenta con un sensor que detecta la posibilidad de que esté por sufrir un ataque cardíaco, o más aún, que ya lo esté sufriendo en ese momento (este dispositivo puede ser una placa de desarrollo Arduino). A partir de ese instante, el vehículo le avisa al conductor (ya sea visualmente como auditivamente), a su vez intenta comunicarse con una baliza (la cual puede ser representada por una placa Arduino) de la carretera para que la ayuda llegue lo antes posible al lugar del evento. También podría disparar alertas a los vehículos cercanos, para que los mismos puedan tomar acciones preventivas, además de avisar al pasaje del ómnibus y detener la marcha en caso de ser necesario.

Los dispositivos utilizado para la construcción de la prueba de concepto son los siguientes:

- Equipo servidor: PC con 16GB de memoria RAM, con procesador de 4 núcleos conectado a una dirección IP de acceso público, sistema operativo Linux, Apache 2.4.18, PHP versión 7.0.15 y una base de Datos PostgreSQL 9.3.
- Equipo Laptop: notebook de escritorio con 12GB de memoria RAM, procesador de doble núcleo conectado a una VPN con el servidor principal mencionado en el punto anterior, sistema operativo Linux, Apache 2.4.18, PHP versión 7.0.15.
- Equipo RaspberryPi B+: equipo con un procesador de 32 bits, con un único núcleo, 1GB de memoria RAM, sistema operativo RaspBian, servidor web Lighttpd versión 1.4.x, PHP versión 5.5.x trabajando en modo CGI. Este equipo está conectado a la VPN que tiene como servidor al equipo server mencionado previamente.
- Placa de desarrollo Arduino Mega 2560, que posee 8k de memoria para datos y un procesador de 16 bits; módulo wifi (ESP8266-01).
- Placa de desarrollo Arduino Nano V3 donde la memoria disponible para datos es de apenas 2K y un procesador de 16 bits; módulo wifi (ESP8266-01).

Las limitaciones de los dispositivos se centran en la limitada capacidad de procesamiento, memoria, conectividad y batería; de hecho, las placas Arduino dependen de una batería para poder operar. Los dispositivos ubicuos utilizados y el caso de prueba son representativos tanto de los problemas reales a los que éstos dispositivos se enfrentan, donde la ubicuidad es el común denominador, así como el tipo de problema que se esquematiza.

La arquitectura de ejecución se puede observar en la Figura 4, donde se aprecia cada dispositivo con el rol que cumple dentro de la coreografía. La placa Arduino representa el rol de un dispositivo Baliza, la RaspberryPi implementa a los dispositivos con los roles de VehiculoAccidentado y VehiculoTransito, y los equipos con más capacidad de procesamiento a los roles de CentralBaliza y CentralEmergencia.

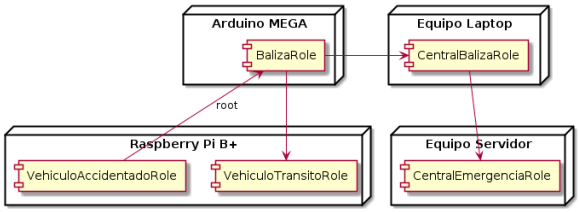

Figura 4. Diagrama de despliegue de dispositivos y roles

El diagrama de secuencia de la Fig. 5 representa la ejecución de la coreografía, en base a la prueba de concepto definida. Esta figura se genera automáticamente desde el mismo framework ya que por cada ejecución que se hace de un caso de prueba, se genera un código de identificación único que es informado en la ejecución para luego poder obtener el gráfico correspondiente. En el diagrama se pueden observar, como etiquetas en la secuencia de mensajes, el formato de información que viaja desde un dispositivo hacia otro, el cual está definido dentro de la definición de la coreografía realizada en WS-CDL. De acuerdo al extracto en XML que mostramos previamente, las dos interacciones que allí presentamos se ven representadas en este diagrama como las interacciones marcadas con las numeraciones 1, 2, 3, 7 y 10 respectivamente.

Este escenario que se plantea puede ser fácilmente abstraído a otras situaciones como puede ser un sistema de alarma hogareño, coordinación de una flota de distribución de mercadería, provisión de servicios a pasajeros en tránsito dentro de un aeropuerto, etc.

VII. Conclusiones y Trabajo Futuro

El framework construido ha permitido alcanzar el objetivo principal de trabajo: *aplicar las especificaciones y estandarizaciones existentes en SOA para la coordinación de dispositivos*

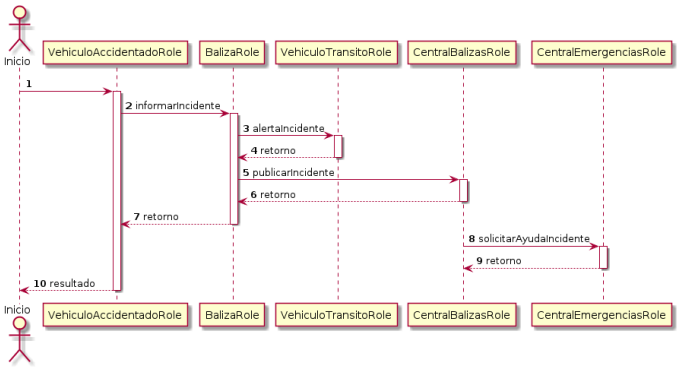

Figura 5. Diagrama de secuencia de ejecución de la coreografía

ubicuos en ambientes pervasivos. El framework propuesto funciona de manera correcta, cumpliendo y respetando las características propias de los dispositivos ubicuos. La solución encontrada al problema de investigación es simple, interoperable y extensible, ya que está basada en una extensión de los conceptos de una teoría que existía únicamente en SOA a un espacio donde no existían, esto es, los ambientes pervasivos.

Actualmente nos encontramos trabajando en el refinamiento y ajuste de un mecanismo de manejo de transacciones dentro de la ejecución de la coreografía, con un commit en dos fases, a través de la utilización de la especificación WS-Transaction [22]. Otra de las líneas de investigación actuales es la implementación de una capa de seguridad, similar a la de WS-Security [23] y de confiabilidad tanto en la entrega de los mensajes como de la ejecución en su conjunto.

Quedan además otros trabajos como es el caso de una mayor parametrización del framework, permitiendo de esta manera que nuevos dispositivos puedan ser incorporados a la ejecución sin necesidad de tener que modificar el código fuente. Otro de los puntos en los que debemos trabajar es en la posibilidad de incorporar otros protocolos de comunicación como puede ser el caso de Bluetooth, NFC, etc. Otro aspecto importante es el de establecer mecanismos para que se incorporen a la ejecución dispositivos sin capacidad de procesamiento, como puede ser el caso de tarjetas RFID.

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] S. T. March and G. F. Smith, "Design and Natural Science Research on Information Technology," *Decis. Support Syst.*, vol. 15, pp. 251–266, Dec. 1995.

[4] 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.

[5] 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.

[6] 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.

[7] "Amigo Project." Available at http://gforge.inria.fr/projects/amigo/.

[8] 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.

[9] 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).

[10] 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.

[11] 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.

[12] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel, "Services collaboration in wireless sensor and actuator networks: Orchestration versus choreography," in *2012 IEEE Symposium on Computers and Communications (ISCC)*, pp. 000411–000418, July 2012.

[13] C. Duhart, P. Sauvage, and C. Bertelle, "Emma: A resource oriented framework for service choreography over wireless sensor and actor networks," *International Journal of Wireless Information Networks*, 06 2015.

[14] L. Mostarda, S. Marinovic, and N. Dulay, "Distributed orchestration of pervasive services," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 166–173, April 2010.

[15] Z. Zhou, D. Zhao, L. Liu, and P. C. Hung, "Energy-aware composition for wireless sensor networks as a service," *Future Generation Computer Systems*, vol. 80, pp. 299 – 310, 2018.

[16] 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.

[17] D. Georgakopoulos and M. P. Papazoglou, *Service-Oriented Computing*. The MIT Press, 2008.

[18] W3C, "Web Service Architecture." https://www.w3.org/TR/ws-arch/, 2004.

[19] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley Indianapolis, 2008.

[20] IBM, "Web Service Definition." https://www.ibm.com/support/knowledgecenter/en/SSGMCP_4.2.0/com.ibm.cics.ts.webservices.doc/concepts/dfhws_definition.html, 2018.

[21] N. Dragoni, S. Giallorenzo, A. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering* (M. Mazzara and B. Meyer, eds.), Springer, Sept. 2017.

[22] OASIS, "Web Services Atomic Transaction (WS-AtomicTransaction)." http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html, 07 2007.

[23] OASIS, "Web Services Security (WS-Security)." https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 11 2006.