PUBLICACIÓN DE TESIS / TRABAJOS FINALES

TESIS DE DOCTORADO

# Coordinación de dispositivos en ambientes ubicuos mediante coreografías

*Oscar Alfredo Testa*

EdUNLPam
Universidad Nacional de La Pampa

REUN
RED DE EDITORIALES DE UNIVERSIDADES NACIONALES

# Coordinación de dispositivos en ambientes ubicuos mediante coreografías

Mg. Ing. Oscar Alfredo Testa

Tesis para optar a la titulación de postgrado
correspondiente al Doctorado en Ingeniería Informática

**UNIVERSIDAD NACIONAL DE SAN LUIS**
**FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS Y NATURALES**

*Director:*

**Dr. Germán Montejano**
**Dr. Oscar Dieste**

**San Luis**
**2020**

Testa, Oscar Alfredo
Coordinación de dispositivos en ambientes ubicuos mediante coreografías / Oscar Alfredo Testa. - 1a ed. - Santa Rosa : Editorial de la Universidad Nacional de La Pampa, 2022.
Libro digital, PDF - (Publicación de tesis o trabajos finales de grado, tesis de maestría, tesis de doctorado y trabajos / Ana María Teresa Rodriguez)

Archivo Digital: descarga y online
ISBN 978-950-863-484-9

1. Computación. 2. Internet. 3. Sistemas de Información. I. Tí
CDD 005.42

Marzo de 2023, Santa Rosa, La Pampa

ISBN 978-950-863-484-9

Cumplido con lo que marca la ley 11723
EdUNLPam - Año 2023
Cnel. Gil 353 PB - CP L6300DUG
SANTA ROSA - La Pampa - Argentina

UNIVERSIDAD NACIONAL DE LA PAMPA
Rector: Oscar Daniel Alpa
Vicerrectora: María Ema Martin

EdUNLPam
Presidente: Ignacio Kotani
Director: Rodolfo David Rodríguez
Consejo Editor:
Gustavo Walter Bertotto
María Marcela Domínguez
Fernando Colli
Edith Alvarellos / Federico Martocci
Carla Etel Suarez / Daniel Omar Maizon
Lucía Carolina Colombato / Jimena Marcos
María Pía Bruno / Laura Noemí Azcona
Alicia María Vignatti / Oscar Alfredo Testa
Mónica Boeris / Natalia Cazaux
María Soledad Mieza / Patricia Bibiana Lázaro

*El presente trabajo de Investigación Doctoral*
*y*
*todo el esfuerzo que representa,*
*esta dedicado a mi familia,*
*Miriam,*
*Luciana y Francisco.*

# Agradecimientos

Este trabajo ha sido el resultado de un objetivo que me propuse hace varios años y que parecía lejano e inalcanzable. Sin embargo, gracias al esfuerzo dedicado y la colaboración de muchas personas ha podido ser llevado adelante.

En primer lugar mi agradecimiento a mis Directores, Germán y Oscar, con los cuales logré forjar una amistad más allá del trabajo y las distancias. A Oscar le agradezco su paciencia permanente.

A mi mujer y mis hijos, quienes me brindaron su tiempo, su acompañamiento, comprensión y aliento para poder llegar a este final.

A mis padres, quienes me dieron la posibilidad inicial de poder estudiar una carrera universitaria, además de darme el ejemplo de esfuerzo y honestidad por sobre todas las cosas.

A Rodrigo Fonseca, otro gran amigo que me dio esta investigación, quien ha brindado muchas horas de su tiempo para dar apoyo en todo lo que se lo solicitó, por sus palabras de aliento y su acompañamiento a la distancia.

A todos, muchas gracias.

*Un especial reconocimiento a las entidades que hicieron posible esta investigación a través de su financiamiento: Universidad Nacional de La Pampa-Facultad de Ciencias Exactas y Naturales. También quiero dar un reconocimiento a la entidad que me dio la posibilidad de incorporarme a su plan de Posgrados: Universidad Nacional de San Luis-Facultad de Ciencias Físico, Matemáticas y Naturales.*

# Resumen

**Introducción:** Actualmente nos encontramos involucrados en ambientes donde los dispositivos ubicuos forman parte de nuestra vida cotidiana y de nuestras tareas diarias. De forma permanente estamos interactuando con dichos dispositivos y más aún, con los servicios que ellos nos brindan.

En casi todos los casos, los dispositivos ubicuos no proporcionan servicios de forma aislada, sino que deben cooperar con otros dispositivos. Actualmente, los mecanismos de cooperación disponibles son fundamentalmente de tipo propietario, y las pocas propuestas provenientes de la academia no han tenido apenas impacto en la práctica.

Distintos autores han planteado la necesidad de la composición de dispositivos ubicuos para afrontar desafíos tales como la tolerancia a fallas, escaso nivel de procesamiento, problemas de conectividad, etc. La necesidad de desarrollar sistemas donde una multiplicidad de dispositivos ubicuos se coordinen entre ellos para lograr un fin no es sólo un problema académico, sino que responde también a necesidades de la industria, como es el caso de la iniciativa Industria 4.0 o de los autos inteligentes, por citar dos ejemplos.

La computación orientada a servicios, y en particular los servicios web en ambiente de internet, proporcionan mecanismos para la composición de servicios. Dichos mecanismos, como por ejemplo las orquestaciones, permiten construir sistemas de negocio complejos y aplicaciones a partir de una gran cantidad de servicios heterogéneos, simples y distribuidos. 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 es proveedor o consumidor de un servicio, la coordinación de dispositivos se ajusta perfectamente con la composición de servicios en ambientes distribuidos. Sin embargo, los mecanismos de composición establecidos para servicios web no son directamente aplicables. Por ejemplo, las orquestaciones no consideran particularidades de los dispositivos ubicuos, como su escasa capacidad de procesamiento o transmisión/recepción de información.

**Objetivos:** El objetivo principal de esta investigación es el de definir un mecanismo de coordinación de dispositivos ubicuos que garantice su interoperabilidad independientemente del modelo y fabricante del mismo; utilizando

los estándares de SOA y de coreografías para la composición de servicios.

**Método:** Se ha utilizado como metodología de investigación design science, ya que es la que mejor se adapta a la naturaleza del problema, planteando como uno de sus lineamientos la construcción de artefactos y su posterior evaluación. En nuestro caso, los artefactos a construir han sido: 1) una especificación de coreografías adaptado a los dispositivos ubicuos y 2) un framework que implementa dicha especificación al nivel de prueba de concepto.

**Resultados:** Se ha obtenido un framework de coordinación de dispositivos a través de la utilización de coreografías, que funciona de manera correcta y dando soporte a las características distintivas de los dispositivos ubicuos. La solución planteada es simple, interoperable y extensible. Hemos logrado trasladar, a un espacio o entorno donde las soluciones realizadas eran ad-hoc, conceptos de una teoría que existía únicamente en SOA, cuyo propósito es el de realizar sistemas interoporables de forma estandarizada y transparente. Esto permite que los dispositivos ubicuos puedan cooperar a la hora de realizar sus tareas con ordenadores, teléfonos móviles, etc. que implementen la interfaz de SOA de forma transparente.

**Conclusión:** La solución planteada ha resuelto satisfactoriamente el problema de investigación, además permite realizar composiciones con dispositivos ubicuos de una manera abierta, basada en estándares y escalable. Esto implica que diversos dispositivos, de diversos fabricantes, y con distintas capacidades, pueden ser incorporados fácilmente al framework y, por consiguiente, participar en coreografías con otros dispositivos de forma sencilla. Además, los dispositivos ubicuos pueden interactuar no sólo con otros dispositivos ubicuos, sino también con aplicaciones basadas en SOA. La utilización de un estándar ya existente en ambientes de internet puede facilitar la adopción de la propuesta en la práctica, ya que: 1) los desarrolladores interesados en la tecnología pueden aplicar sus conocimientos directamente y 2) si se asegura la compatibilidad, todas las herramientas existentes en ambientes de internet podrían ser usadas para el diseño e implementación de sistemas con dispositivos ubicuos.

# Abstract

**Introduction:** We are currently involved in environments where ubiquitous devices are part of our daily lives and tasks. We are permanently interacting with these devices and with the services they offer.

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.

Different authors have raised the need for the composition of ubiquitous devices to face challenges such as fault tolerance, low level of processing, connectivity problems, etc. The need to develop systems where a multiplicity of ubiquitous devices coordinate with each other to achieve an end is not only an academic problem, but also responds to industry needs, such as the Industry 4.0 initiative or smart cars.

Service-oriented computing, and in particular web services, provide mechanisms for the composition of services. These mechanisms, such as orchestrations, allow building complex business systems and applications from a large number of heterogeneous, simple and distributed services. The similarities between the composition of web services and the coordination of ubiquitous devices is striking. If we think that each ubiquitous device in a pervasive environment is a provider or consumer of a service, the coordination of devices fits perfectly with the composition of services in distributed environments. However, the composition mechanisms established for web services are not directly applicable. For example, orchestrations do not consider particularities of ubiquitous devices, such as their low capacity for processing or transmitting / receiving information.

**Aim:** The main aim of this research is to define a coordination mechanism for ubiquitous devices that guarantees their interoperability regardless of its model and manufacturer; using SOA and choreography standards for service composition.

**Research Method:** Design science has been used as a research methodology, since it is the one that best adapts to the nature of the problem, presenting as one of its guidelines the contraction of artifacts and their subsequent evaluation. In our case, the artifacts to be built have been: 1) a choreography specification adapted to ubiquitous devices and 2) a frame-

work that implements said specification at the proof of concept level.

**Results:** A device coordination framework has been obtained through the use of choreography, which works correctly and supports the distinctive characteristics of ubiquitous devices. The proposed solution is simple, interoperable and extensible. We have managed to transfer, to a space or environment where the solutions made were ad-hoc, concepts from a theory that existed only in SOA, whose purpose is to carry out interoporable systems in a standardized and transparent way. This allows ubiquitous devices to cooperate when carrying out their tasks with computers, mobile phones, etc. that implement the SOA interface transparently.

**Conclusions:** The proposed solution has satisfactorily solved the research problem, and also allows compositions to be made with ubiquitous devices in an open, standards-based and scalable way. This implies that different devices, from different manufacturers, and with different capacities, can be easily incorporated into the framework and, therefore, participate in choreographies with other devices in a simple way. Furthermore, ubiquitous devices can interact not only with other ubiquitous devices, but also with SOA-based applications. Using an existing standard in internet environments can facilitate the adoption of the proposal in practice, since: 1) developers interested in technology can apply their knowledge directly and 2) if compatibility is ensured, all tools existing in internet environments could be used for the design and implementation of systems with ubiquitous devices.

# Índice

**Agradecimientos** **v**

**Resumen** **vii**

**1. Introducción** **1**
1.1. Área de trabajo . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2. Importancia del problema . . . . . . . . . . . . . . . . . . . 3
1.3. Problema de investigación . . . . . . . . . . . . . . . . . . . 3
1.4. Objetivo de la tesis . . . . . . . . . . . . . . . . . . . . . . 4
1.5. Metodología . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.6. Contribuciones . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.7. Publicaciones . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.8. Estructura de la tesis . . . . . . . . . . . . . . . . . . . . . 8

**2. Estado de la cuestión** **9**
2.1. Conceptos generales de composición . . . . . . . . . . . . . . 9
2.1.1. Definición de la composición . . . . . . . . . . . . . . . 10
2.1.2. Selección de los servicios de la composición . . . . . . 12
2.1.3. Implementación de la composición . . . . . . . . . . . 14
2.1.4. Ejecución de la composición . . . . . . . . . . . . . . . 15
2.2. Composición de dispositivos ubicuos . . . . . . . . . . . . . . 16
2.2.1. Najar . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.2.2. Loke . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.2.3. Palmieri . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.2.4. Viroli . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.3. Frameworks comerciales . . . . . . . . . . . . . . . . . . . . 20
2.3.1. OSGi (*Open Services Gateway initiative*, Open Services Gateway initiative) . . . . . . . . . . . . . . . . . . 20
2.3.2. Proyecto MOSQUITO . . . . . . . . . . . . . . . . . . 22
2.3.3. Android Studio . . . . . . . . . . . . . . . . . . . . . . 23
2.3.4. OpenHab . . . . . . . . . . . . . . . . . . . . . . . . . 24

2.3.5. KNX . . . . . 25
2.3.6. Z-wave . . . . . 25
2.4. Otros Frameworks académicos . . . . . 26
2.4.1. Aura . . . . . 26
2.4.2. Gaia . . . . . 27
2.4.3. Oxigen . . . . . 27
2.4.4. Amigo . . . . . 28
2.5. Especificaciones asociadas a paradigmas específicos . . . . . 28
2.5.1. SOA (*Service Oriented Architecture*, Arquitectura Orientada a Servicios) . . . . . 29
2.5.2. Web Service Choreography Description Language (WS-CDL (*Web Service Choreography Description Language*, Lenguaje de Descripción de Coreografías con Servicios Web)) . . . . . 29
2.5.3. Web Service Choreography Interface - WSCI (*Web Service Choreography Interface*, Interface de Coreografías de Servicios Web) . . . . . 32
2.5.4. Web Services Business Process Execution Language - WS-BPEL (*Web Service Business Process Execution Language*, Lenguaje de Ejecución de Procesos de Negocio con Servicios Web) . . . . . 33
2.5.5. JADE (*Java Agent DEvelopment Framework*, Framework de Desarrollo de Agentes Java) . . . . . 35
2.6. Carencias del estado de la cuestión . . . . . 36

**3. Planteamiento del problema y Objetivos de Investigación 39**
3.1. Identificación del problema . . . . . 39
3.2. Importancia del problema . . . . . 40
3.3. Objetivos de Investigación . . . . . 41
3.3.1. Objetivo Principal . . . . . 41
3.3.2. Objetivos Específicos . . . . . 42

**4. Metodología 45**
4.1. Selección del método de investigación . . . . . 45
4.2. Design Science . . . . . 46
4.2.1. Descripción general . . . . . 46
4.2.2. Propuestas metodológicas . . . . . 47
4.3. Aplicación en esta investigación . . . . . 51

**5. Aproximación a la solución 55**
5.1. Trabajos que proponen aproximaciones similares . . . . . 56
5.2. Ventajas de la convergencia con SOA . . . . . 57

5.3. Estrategia de solución . . . . . . . . . . . . . . . . . . . . . . . 59
5.3.1. Características SOA . . . . . . . . . . . . . . . . . . . . 59
5.3.2. Características de los dispositivos ubicuos . . . . . . . 60
5.3.3. Convergencia entre SOA y los dispositivos ubicuos . . 62
5.4. Tipos de dispositivos ubicuos . . . . . . . . . . . . . . . . . . . 66
5.5. Solución planteada . . . . . . . . . . . . . . . . . . . . . . . . . 68
5.5.1. Definición del escenario de trabajo . . . . . . . . . . . 68
5.5.2. Pruebas de concepto . . . . . . . . . . . . . . . . . . . . 70

**6. Resolución** **75**
6.1. Primer ciclo de Design-Science . . . . . . . . . . . . . . . . . 75
6.1.1. Características utilizadas de WS-Choreography . . . . 75
6.1.2. Definición de la coreografía del escenario planteado . . 75
6.1.3. Dispositivos utilizados . . . . . . . . . . . . . . . . . . 77
6.1.4. Implementación del framework de coordinación . . . . 78
6.1.5. Evaluación . . . . . . . . . . . . . . . . . . . . . . . . . 83
6.2. Segundo ciclo de Design-Science . . . . . . . . . . . . . . . . . 83
6.2.1. Características utilizadas de WS-Choreography . . . . 84
6.2.2. Definición de la coreografía del escenario planteado . . 84
6.2.3. Dispositivos considerados . . . . . . . . . . . . . . . . 84
6.2.4. Implementación del framework de coordinación . . . . 85
6.2.5. Evaluación . . . . . . . . . . . . . . . . . . . . . . . . . 90
6.3. Tercer ciclo de Design-Science . . . . . . . . . . . . . . . . . . 91
6.3.1. Características utilizadas de WS-Choreography . . . . 92
6.3.2. Definición de la coreografía del escenario planteado . . 92
6.3.3. Dispositivos considerados . . . . . . . . . . . . . . . . 92
6.3.4. Implementación del framework de coordinación . . . . 93
6.3.5. Evaluación . . . . . . . . . . . . . . . . . . . . . . . . . 94
6.4. Cuarto ciclo de Design-Science . . . . . . . . . . . . . . . . . . 97
6.4.1. Características utilizadas de WS-Choreography . . . . 97
6.4.2. Definición de la coreografía del escenario planteado . . 97
6.4.3. Dispositivos considerados . . . . . . . . . . . . . . . . 97
6.4.4. Implementación del framework de coordinación . . . . 97
6.4.5. Evaluación . . . . . . . . . . . . . . . . . . . . . . . . . 99
6.5. Quinto ciclo de Design-Science . . . . . . . . . . . . . . . . . . 100
6.5.1. Características utilizadas de WS-Choreography . . . . 100
6.5.2. Definición de la coreografía del escenario planteado . . 100
6.5.3. Dispositivos considerados . . . . . . . . . . . . . . . . 101
6.5.4. Implementación del framework de coordinación . . . . 101
6.5.5. Evaluación . . . . . . . . . . . . . . . . . . . . . . . . . 106
6.6. Sexto ciclo de Design-Science . . . . . . . . . . . . . . . . . . 107

6.6.1. Características utilizadas de WS-Choreography . . . . 109
6.6.2. Definición de la coreografía del escenario planteado . . 112
6.6.3. Dispositivos considerados . . . . . . . . . . . . . . . . 112
6.6.4. Implementación del framework de coordinación . . . . 112
6.6.5. Evaluación . . . . . . . . . . . . . . . . . . . . . . . . 113

**7. Discusión 117**
7.1. Cumplimiento de los Objetivos de investigación . . . . . . . . 117
7.1.1. Desarrollar un framework que implemente la especificación WS-CDL que pueda ser ejecutado por dispositivos ubicuos . . . . . . . . . . . . . . . . . . . . . . 118
7.1.2. Lograr la coordinación de dispositivos ubicuos teniendo en cuenta las características distintivas de los mismos, tales como escasa capacidad de memoria, de procesamiento, batería, problemas de conectividad, etc. . 119
7.1.3. Asegurar/mantener interoperabilidad entre aplicaciones SOA ejecutadas en servidores arbitrarios y dispositivos ubicuos . . . . . . . . . . . . . . . . . . . . . 119
7.1.4. Conseguir que los dispositivos ubicuos soporten el manejo de distintas especificaciones basadas en SOA (transacciones, seguridad, etc.) . . . . . . . . . . . . . . . . 120
7.2. Limitaciones de la solución propuesta . . . . . . . . . . . . . 120
7.3. Relación con otras tecnologías . . . . . . . . . . . . . . . . . 122
7.3.1. Relación con Frameworks de dispositivos ubicuos . . . 122
7.3.2. Relación con Microservicios . . . . . . . . . . . . . . . 122
7.3.3. Relación con IoT (*Internet of Things*, Internet de las Cosas) . . . . . . . . . . . . . . . . . . . . . . . . . . 123

**8. Conclusiones 125**

**9. Futuras líneas 127**
9.1. Líneas futuras de investigación . . . . . . . . . . . . . . . . . 127
9.1.1. Descubrimiento de servicios . . . . . . . . . . . . . . . 127
9.1.2. Introducción de capas de seguridad . . . . . . . . . . . 127
9.1.3. Implementación de transacciones de acuerdo a estándares . . . . . . . . . . . . . . . . . . . . . . . . . . . 128
9.1.4. Convergencia con microservicios . . . . . . . . . . . . . 128
9.1.5. Convergencia con IoT . . . . . . . . . . . . . . . . . . 128
9.2. Líneas futuras de desarrollo . . . . . . . . . . . . . . . . . . . 128
9.2.1. Implementación del framwork de ejecución de coreografías sobre dispositivos ubicuos con calidad industrial 128
9.2.2. Ampliación de las pilas de protocolos . . . . . . . . . . 129

**Bibliografía** **131**

**Lista de acrónimos** **136**

**I Apéndices** **141**

**Análisis de la utilización de memoria** **143**

**Diagramas de clases y código relacionado** **151**

.1. Diagramas de clases en PHP (*Hypertext PreProcessor*, Pre-Procesador de Hypertexto) . . . . . . . . . . . . . . . . . . . . 151

.2. Diagramas de clases en C++ . . . . . . . . . . . . . . . . . . . 162

**Instalación del Framework** **171**

.3. Instalación en PHP . . . . . . . . . . . . . . . . . . . . . . . . . 171

.4. Instalación en Arduino . . . . . . . . . . . . . . . . . . . . . . . 174

**Arquitectura de ejecución** **177**

# Índice de figuras

2.1. Ejemplo de actividades realizando una composición de servicios para la compra de un Artículo . . . . . . . . . . . . . . . 10
2.2. Coreografía vs. Orquestación. . . . . . . . . . . . . . . . . . . 11
2.3. OSGi - Modelo de capas. Fuente - https://www.osgi.org/developer/architecture/layering-osgi/ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.4. Arquitectura MOSQUITO Fuente: (Mühlhäuser y Gurevych, 2008, pag. 576) . . . . . . . . . . . . . . . . . . . . . . . . . 23
2.5. Arquitectura WS-CDL. Fuente: http://cic.puj.edu.co . . . . . 31
2.6. Arquitectura WSCI. Fuente: https://www.w3.org/TR/wsci . 33
2.7. Arquitectura WS-BPEL. Fuente: `http://docs.oasis-open.org/opencsa/sca-bpel` . . . . . . . . . . . . . . . . . . . . 34
2.8. Arquitectura JADE. Fuente: (Bellifemine et al., 2007) . . . . 36

4.1. ciclos-design-science . . . . . . . . . . . . . . . . . . . . . . 50
4.2. actividades-design-science . . . . . . . . . . . . . . . . . . . 51

5.1. Capas que componen SOA. Fuente: (OASIS, 2009) . . . . . . 58
5.2. escenario-trabajo . . . . . . . . . . . . . . . . . . . . . . . . 69
5.3. diagramadeflujo . . . . . . . . . . . . . . . . . . . . . . . . . 74

6.1. diagrama-ejecucion-coreografía . . . . . . . . . . . . . . . . . 77
6.2. diagrama-despliegue-coreografia . . . . . . . . . . . . . . . . 79
6.3. diagrama-clases-php-ppc . . . . . . . . . . . . . . . . . . . . 82
6.4. diagrama-ejecución-coreografía . . . . . . . . . . . . . . . . . 84
6.5. diagrama-despliegue-coreografia . . . . . . . . . . . . . . . . 85
6.6. diagrama-clases-arduino1-spc . . . . . . . . . . . . . . . . . . 88
6.7. diagrama-clases-arduino2-spc . . . . . . . . . . . . . . . . . . 89
6.8. diagrama-ejecución-coreografía . . . . . . . . . . . . . . . . . 92
6.9. figura-comparacion-arduino . . . . . . . . . . . . . . . . . . . 93
6.10. diagrama-despliegue-coreografia . . . . . . . . . . . . . . . . 93
6.11. diagrama-ejecución-coreografía . . . . . . . . . . . . . . . . . 97
6.12. diagrama-transaccion-tcc . . . . . . . . . . . . . . . . . . . . 102

6.13. diagrama-clases-transaccion-tcc . . . . . . . . . . . . . . . . . . 103
6.14. diagrama-clases-transaccion-tcc . . . . . . . . . . . . . . . . . . 105
6.15. diagrama-ejecución-coreografía . . . . . . . . . . . . . . . . . . 115

1. diagrama-clases-transaccion-tcc . . . . . . . . . . . . . . . . . . 151
2. diagrama-clases-transaccion-tcc . . . . . . . . . . . . . . . . . . 163

3. diagrama-ejecucion . . . . . . . . . . . . . . . . . . . . . . . . 177
4. diagrama-ejecucion-siguiente . . . . . . . . . . . . . . . . . . . . . 178

# Índice de Tablas

2.1. Carencias del Estado de la Cuestión . . . . . . . . . . . . . . . 38

5.1. Características de los dispositivos ubicuos . . . . . . . . . . . . 65
5.2. Lista de dispositivos . . . . . . . . . . . . . . . . . . . . . . . . . 67

6.1. Alternativas de solución para desapariciones de los dispositivos 108

# Capítulo 1

# Introducción

La presente tesis doctoral está encuadrada en el área de los dispositivo ubicuos y, o, pervasivos (Sección 1.1). Estos dispositivos deben comunicarse entre sí para realizar tareas mediante la combinación de los servicios que brindan.

En la actualidad existen diversos proyecto para integrar dispositivos ubicuos en la vida de las personas. Estos proyectos van desde la Domótica a la Internet de las cosas, pasando por áreas de aplicación relevantes a nivel de la industria como es el caso de la Industria 4.0 (Sección 1.2). La interacción entre dispositivos se realiza, en general, utilizando protocolos propietarios y sin seguir definiciones estándares, provocando entre otros problemas que dispositivos de distintos proveedores no puedan ser utilizados conjuntamente (Sección 1.3).

Esta investigación tiene como objetivo la definición de un mecanismo de coordinación de dispositivos ubicuos que garantice su interoperabilidad independientemente del modelo y fabricante del mismo (Sección 1.4).

Para poder cumplir con este objetivo hemos seleccionado *Design Science* como el método de investigación que mejor se adapta para la resolución del problema de investigación planteado (Sección 1.5). A través de los distintos ciclos de investigación, hemos creado y validado un framework de ejecución de coreografías con dispositivos ubicuos en ambientes pervasivos utilizando los estándares de SOA (Sección 1.6).

Como resultado de esta investigación, se han obtenido publicaciones relevantes (Sección 1.7) y se espera obtener más a corto y medio plazo.

### 1.1. Área de trabajo

En la actualidad, las actividades cotidianas del hombre se han hecho dependientes de una gran cantidad de dispositivos electrónicos tales como: ordenadores personales, ordenadores portátiles, teléfonos móviles, PDA (*Personal Digital Assistant*, Asistente Digital Personal), tabletas, sensores de mu-

chas y diversas utilidades, entre otros; los cuales logran comunicarse entre sí gracias a diversos protocolos de comunicación inalámbrica, redes de celulares, LAN (*Local Area Network*, Red de Area Local), WAN (*Wide Area Network*, Red de Area Amplia), Bluetooth, etc. Estamos en la presencia de un nuevo escenario social, donde la interacción permanente con estos elementos es ineludible.

Los avances de las comunicaciones entre dispositivos han permitido que estos sean generadores y consumidores de servicios al mismo tiempo; es decir, de acuerdo a sus capacidades un dispositivo puede no solo obtener, sino también ofrecer a otros equipos sus funciones, y así cooperar entre ellos.

La tendencia actual es hacia los ambientes pervasivos, los cuales se caracterizan por estar poblados de numerosos dispositivos que, gracias a la integración extrema de los elementos electrónicos, son invisibles al usuario y están en permanente rastreo de la actividad humana (Weiser, 1993).

La computación ubicua es un desarrollo tecnológico que intenta que las computadoras no se perciban en el entorno como objetos diferenciados, y que la utilización por parte de los seres humanos sea lo más transparente y cómoda posible, facilitando de esta manera la integración en la vida cotidiana. Dispositivos ubicuos son todos aquellos dispositivos que pueden existir en todas partes; es decir, son dispositivos electrónicos que tienen capacidad de procesamiento y comunicación y pueden ser encontrados en cualquier lugar: la oficina, el auto, o la misma ropa con la que vestimos.

Desde hace varios años los dispositivos ubicuos han ganado importancia y presencia en la vida cotidiana de las personas, debido principalmente a que: poseen distintos tipos de sensores (posicionamiento, proximidad, luminosidad, temperatura, etc.), facilitan la conectividad incluso en áreas con poca señal o acceso a las redes, permiten la convergencia tecnológica (computo, medios, telefonía, etc) y brindan acceso a servicios de distinta índole (mapas, ayudas, etc).

Por composición se entiende la forma en que se pueden combinar o enlazar un número variable de servicios para realizar una tarea determinada. La composición en ambientes pervasivos implica que los dispositivos deben dialogar entre ellos para poder compartir los servicios que ofrecen con la finalidad de obtener un servicio con valor agregado, o bien para abordar la solución de una problemática particular, como podría ser la seguridad de un hogar, o la seguridad vial, por mencionar algunos ejemplos.

En ambientes ubicuos, la composición de servicios presenta nuevos desafíos, ya que los mecanismos de composición necesitan hacer frente a distintas contingencias. Los dispositivos ubicuos tienen limitantes como son la cantidad de memoria disponible, la durabilidad de la batería o la conectividad en función de la red del lugar donde se encuentre en un momento determinado. Todas estas características hacen que la composición de dispositivos ubicuos[1]

[1]Si bien los autores se refieren a la composición de servicios, se hace dentro de un

se configure en un área de investigación muy importante donde los avances han sido limitados al día de hoy (Sheng et al., 2014).

## 1.2. Importancia del problema

Existen distintos proyectos en la actualidad donde se intentan integrar sensores y dispositivos ubicuos a la vida cotidiana. Específicamente podemos mencionar la domótica, donde distintos dispositivos deben actuar en coordinación para prevenir un incidente de seguridad (ya sea por robo o por incendio) en nuestros hogares. Otro proyecto que podemos mencionar es el de los wereables o vestibles, que son la inclusión de dispositivos o sensores en la ropa que vestimos, los cuales pueden estar sensando distintas funciones del cuerpo humano, lo que puede permitir asistencia remota a pacientes con problemas; ésta es un área muy importante al día de hoy. Sin embargo, existen áreas de aplicación más relevantes. En la industria, nos encontramos hoy con lo que se denomina Industria 4.0[2], que según define (Wang et al., 2016), describe un sistema de producción ciberfísico que integra instalaciones de producción, sistemas de almacenamiento, logística e incluso requisitos sociales para establecer las redes globales de creación de valor. Cuando se habla de sistemas de producción ciberfísico, se está describiendo un sistema de multiagentes equipándolos con tecnologías emergentes como es el caso de IoT, big data, sistemas embebidos, redes de sensores inalámbricos, computación en la nube, etc.

Como vemos en todos los casos, la tendencia es a que los distintos dispositivos (ubicuos en su gran mayoría) se interconecten entre sí de una manera distribuida, sin un equipo o servicio central que los coordine, para llevar adelante trabajos para los cuales fueron pensados, de manera tal que la automatización sea la más alta posible.

## 1.3. Problema de investigación

Hoy en día los dispositivos ubicuos se pueden comunicar entre ellos, compartiendo de alguna manera sus servicios para realizar una tarea determinada (Krumm, 2010). Sin embargo, dicha interacción se realiza, en general utilizando protocolos propietarios y sin seguir definiciones estándares, provocando que otros dispositivos de otros proveedores (o incluso de los mismos)

contexto de dispositivos ubicuos, lo cual a los fines de este trabajo se puede interpretar como composición de dispositivos, haciendo que la terminología para este caso particular sea más adecuada.

[2]El término de Industria 4.0 fue introducido por el gobierno de Alemania como una iniciativa estratégica, como parte del "High-Tech Strategy 2020 Action Plan", si bien en otros países fue tomando otras denominaciones ("Internet +" en China, "Industrial Internet" en EE.UU., por mencionar los más importantes) pero la estrategia es similar en todos los casos.

no puedan ser utilizados. Esto obviamente representa una importante limitación para la coordinación de dispositivos ubicuos independientemente de su tipo, fabricante o características de procesamiento y memoria.

## 1.4. Objetivo de la tesis

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 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. Solo en (Sheng et al., 2014) encontramos mención a ello, donde se plantea que más investigación en esta área es necesaria.

El presente trabajo de tesis se encuadra en la línea de investigación antes indicada relacionada con la coordinación de dispositivos ubicuos, planteando una estrategia de solución basada en la transposición de los conceptos de SOA. La propuesta consiste en adaptar y aplicar las especificaciones actualmente existentes en SOA para la composición de servicios a los dispositivos ubicuos en ambientes pervasivos. Más concretamente, el objetivo de esta Tesis es:

- Definir un mecanismo de coordinación de dispositivos ubicuos que garantice su interoperabilidad independientemente del modelo y fabricante del mismo, utilizando los estándares de SOA.

Debemos destacar que la aplicación de los conceptos de SOA a dispositivos ubicuos no consiste en una mera traslación de los conceptos de un ambiente a otro, sino que será necesario para ello extender las especificaciones de SOA existentes de modo que se adapten a las circunstancias particulares de los sistemas ubicuos. Asimismo se deberá mantener total compatibilidad con las especificaciones relacionadas a SOA y coreografía de servicios existentes.

Si bien existen dos mecanismos de composición de servicios dentro de una arquitectura orientada a servicios, orquestaciones y coreografías, en esta tesis nos ocuparemos del caso de coordinación de dispositivos ubicuos a través 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. Componer dispositivos a través de coreografías resulta bastante más natural que utilizando orquestaciones, debido a que los dispositivos pueden dialogar entre ellos sin tener un servicio central que los coordina, sobre todo en distintas implementaciones; como las mencionadas de IoT, Industria 4.0, etc, donde los dispositivos carecen de un concentrador que coordine las tareas a llevar adelante.

## 1.5. Metodología

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*.

El paradigma de *design science* tiene sus raíces en la ingeniería. Esta metodología, como bien expresan (March y Smith, 1995), tiene como bloque fundamental la creación o construcción de un artefacto y su posterior evaluación para evaluar tanto su impacto como las características de su diseño. Estos artefactos están representados de forma estructurada y van desde software, lógica formal y matemática rigurosa, hasta descripciones informales en lenguaje natural.

Analizado de forma general el paradigma de *design science* como metodología de investigación, vemos que el mismo se adapta de manera natural al problema y objetivo planteado en este trabajo de tesis. (Hevner, 2007) propone 3 ciclos investigación dentro de *design science*. El ciclo de **relevancia** une el entorno contextual del proyecto de investigación con las actividades científicas de diseño. El ciclo de **rigor** conecta las actividades de la ciencia del diseño con la base de conocimientos de fundamentos científicos, experiencia y conocimientos especializados que informan el proyecto de investigación. El ciclo de **diseño** itera entre las actividades de construcción y evaluación de los artefactos de diseño y los procesos de investigación.

Esta investigación se relaciona principalmente con el ciclo de diseño. No obstante, al finalizar la investigación, el conocimiento científico obtenido aportará mejoras a las especificaciones relacionadas con la coordinación de dispositivos, lo cual está relacionado con el ciclo de rigor de *design science*. El ciclo de relevancia no será abordado durante la realización de esta tesis doctoral.

El trabajo de investigación se ha realizado entre junio de 2015 y marzo de 2019 a tiempo parcial, y ha tenido como resultado la obtención de un framework de ejecución de coreografías de dispositivos ubicuos en ambientes pervasivos. La construcción del framework se ha llevado adelante en seis ciclos de design science. En cada ciclo de investigación se ha obtenido un producto, el cual fue evaluado para determinar el cumplimiento de sus especificaciones. Las mejoras y cambios propuestos se incorporaron en la siguiente etapa o ciclo de investigación, hasta llegar al producto final.

## 1.6. Contribuciones

Los resultados principales de esta investigación se resumen a continuación:

- Se ha construido un **framework de ejecución de coreografías**. Este framework, si bien es ejecutable y puede ser utilizado para desarrollar

cualquier tipo de aplicación con dispositivos ubicuos, no fue realizado con un objetivo comercial; por ello puede contener errores y utilizar con menos eficiencia de la teóricamente posible los recursos disponibles (memoria, recursos, código, etc), etc. La finalidad del framework es la de realizar una prueba de concepto para poder abordar y dar solución a los problemas de investigación que aquí nos planteamos.

- El framework construido es **totalmente compatible con las especificaciones SOA**. El framework puede ser utilizado tanto para servicios de una arquitectura SOA tradicional como una de dispositivos ubicuos, o incluso una mezcla de ambas arquitecturas.

- Se han realizado **aportes y mejoras a la especificación del lenguaje de ejecución de coreografías WS-CDL 1.0**, donde se han planteado en la misma cambios y mejoras para poder dar soporte a las características particulares de los dispositivos ubicuos.

## 1.7. Publicaciones

Los resultados de la investigación han sido importantes y es por ello que se han obtenido publicaciones tanto en Congresos Nacionales como Internacionales. Adicionalmente, varios artículos han sido enviado a diversos congresos, algunos ya aprobados y otros en proceso de aceptación.

Es preciso mencionar que en adición a los artículos detallados, está planificada la escritura un artículo, a corto plazo, para ser publicado en una revista donde se presentará el desarrollo completo del framework con todas sus características.

A continuación se describe brevemente la contribución de los artículos resultado de esta tesis.

- **Artículos Publicados**

  (P1) **O. Testa, E. R. Fonseca, G. Montejano, N. Debnath, O. Dieste. *WS-CDL: Coordinating Ubiquitous Devices in Pervasive Environments Using a Web Standard*, 21th International Conference on Industrial Technology, (ICIT2020), Buenos Aires, Argentina, Febrero 2020.**

  El objetivo de este artículo fue mostrar el framework desarrollado basado en los conceptos de SOA, los cuales fueron trasladados para la coordinación de dispositivos ubicuos en ambientes pervasivos.

  (P2) **O. Testa, E. R. Fonseca, G. Montejano, O. Dieste. *Coordination of Ubiquitous Devices in Pervasive Environments:***

*A Proposal Based on WS-CDL*, **38th International Conference of the Chilean Computer Science Society, (SCCC'2019), Concepción, Chile, Noviembre 2019.**

El objetivo de este artículo fue mostrar a la comunidad científica el framework desarrollado, y a su vez presentar los resultados obtenidos de la utilización del mismo sobre una prueba de concepto realizada con dispositivos ubicuos específicos.

(P3) **O. Testa, E. R. Fonseca, G. Montejano, O. Dieste.** *Cómo mejorar el tráfico de vehículos con dispositivos interconectados de bajo coste: una prueba de concepto*, **7mo Congreso Nacional de Ingeniería Informática - Sistemas de Información , (CoNaIISI 2019), San Justo, Buenos Aires, Argentina, Noviembre 2019.**

En este trabajo se presentó un escenario crítico realista de problemas de tránsito y mostrar cómo un conjunto de dispositivos asequibles interconectados pueden colaborar para solucionarlo.

(P4) **O. Testa, E. R. Fonseca, G. Montejano, O. Dieste.** *Coreografía de dispositivos ubicuos basada en SOA: ¿Cuánto de limitados pueden ser los dispositivos?*, **XXV Congreso Argentino de Ciencias de la Computación, (CACIC'2019), Río Cuarto, Argentina, Octubre 2019.**

En este trabajo se presentó un análisis de memoria para la utilización del framework de ejecución de coreografías en dispositivos ubicuos. Su finalidad es determinar cuál es el límite de memoria y procesamiento que deben poseer los dispositivos para que puedan ser incorporados en la coreografía.

(P5) **Oscar A. Testa.** *Coordinación de dispositivos en ambientes ubicuos mediante coreografías.* **XXI Congreso Iberoamericano en Ingeniería de Software (CibSE2018), Bogotá, Colombia, Abril 2018.**

El objetivo de este artículo fue dar a conocer a la comunidad de Ingeniería de Software, una aproximación a la investigación que se describe es este manuscrito, cuando aún estaba en sus etapas iniciales. La contribución del artículo para la tesis consistió en recibir comentarios y sugerencias de experimentados investigadores en el área, que nos dieron sus perspectivas de la investigación en curso.

(P6) **O. Dieste, E. R. Fonseca, G. Montejano, O. Testa.** *Desafíos en el desarrollo de sistemas ubicuos: Una propuesta de solución basada en SOA*, **3er. Congreso Nacional de Ingeniería Informática / Sistemas de Información, (CONAIISI'2015), Buenos Aires, Argentina, Noviembre 2015.**

> El objetivo de este artículo fue dar los primeros pasos en el estudio de la realización de coordinaciones de dispositivos ubicuos en ambientes pervasivos a través de la utilización de una arquitectura orientada a servicios, para que la comunidad de Ingeniería de Software brindara su aval a la investigación que comenzaba a dar sus primeros pasos.

## 1.8. Estructura de la tesis

La tesis se estructura de la siguiente manera:

- En el capítulo 2 se presentan los antecedentes sobre los cuales se basó la investigación, donde se estudia tanto la base de conocimiento científico acerca de la materia, así como los distintos frameworks y entornos existentes.
- En el capítulo 3 se hace el planteamiento del problema de investigación que se lleva adelante, donde se identifica el problema por un lado y por otro su importancia. Además en este capítulo se plantean los objetivos de investigación.
- En el capítulo 4 se detalla la metodología de investigación utilizada así como la fundamentación de la elección del método.
- Una aproximación a la solución al problema planteado se presenta en el capítulo 5.
- En el capítulo 6 se desarrolla la resolución del problema de investigación planteado, a través de la presentación de los distintos ciclos de *design science*.
- Una vez que se ha desarrollado la solución se presenta una discusión de los resultados obtenidos, capítulo 7.
- Finalmente en el capítulo 8 se abordan las conclusiones finales
- Por último en el capítulo 9 se desarrollan los trabajos futuros.

Adicionalmente se pueden consultar los Anexos a esta tesis, los cuales son:

- Anexo A donde se hace un análisis de utilización tanto de memoria como de capacidad de procesamiento requerido para la ejecución del framework presentado.
- El Anexo B presenta los diagramas de clases y el código relacionado relativo al desarrollo realizado.

- En el Anexo C se detalla los pasos que se necesitan llevar adelante para realizar una correcta instalación del framework para su utilización.
- En el Anexo D se muestra la arquitectura de ejecución del framework desarrollado.

# Capítulo 2

# Estado de la cuestión

En este capítulo abordaremos el estudio del Estado de la cuestión, en el cual describiremos las distintas aristas relacionadas con el tema de investigación de la presente Tesis. En lo que sigue discutiremos desde aspectos generales acerca de la composición de distintos elementos (servicios, dispositivos, etc.) en ambientes pervasivos, pasando por distintos frameworks comerciales y académicos que trabajan con dispositivos ubicuos, para finalizar haciendo un análisis de las especificaciones emanadas de OASIS (*Organization for Advanced Structured Information Systems*) y de W3C (*World Wide Web Consortium*, Consorcio Mundial Web) relacionadas con este trabajo.

## 2.1. Conceptos generales de composición

La composición de distintos elementos (servicios, dispositivos, etc.) en una entidad abstracta de mayor nivel tiene como objetivo principal que dicha entidad realice una tarea compleja mediante la utilización de diversos elementos más simples (Sheng et al., 2014). La teoría de composición más elaborada se ha desarrollado en el área de los servicios web, por lo que en esta sección utilizaremos la terminología asociada a SOA. Nótese, no obstante, que los conceptos que describiremos son universalmente aplicables.

Normalmente se hace una distinción entre servicios atómicos y compuestos. Un servicio atómico es un punto de acceso a una aplicación que no se basa en otro servicio para poder llevar adelante su trabajo. En cambio, un servicio compuesto es una estructura que interacciona con uno o más servicios atómicos o compuestos, los cuales colaboran entre ellos para poder llevar adelante su trabajo.

Para poder llevar adelante la composición de servicios (o dispositivos, en nuestro caso) es necesario realizar ciertas actividades (definición, selección, implementación, ejecución), las cuales deben respetar una secuencia más o menos estricta, y que detallaremos a continuación. Cada una de estas fases necesita que se cumplan una serie de requisitos. Estas fases y requisitos han

sido definidos en (Sheng et al., 2014), los cuales se describen a continuación.

### 2.1.1. Definición de la composición

Durante esta fase el diseñador establece los requerimientos que debe cumplir la composición. Estos requerimientos representan las características del problema a solucionar y las necesidades de los distintos *stakeholders*. También se deben tener en cuenta todos aquellos requerimientos no funcionales tales como la fiabilidad, disponibilidad, etc.

Como parte de esta definición se debe confeccionar un modelo donde se especifiquen las distintas tareas o actividades que se deben realizar, las cuales pueden llevarse adelante a través de procesos automáticos o semi-automáticos. Las actividades son los pasos o acciones que se llevan adelante en una composición de servicios, donde estas actividades pueden involucrar distintas organizaciones o procesos independientes. En la figura 2.1 podemos apreciar una composición de servicios a través de la utilización de actividades. En dicha figura se presenta el proceso llevado adelante en la compra de artículos de un cliente, en donde las distintas actividades se comunican entre sí para poder cumplimentar con el pedido que realiza el cliente. En particular podemos apreciar que la composición comienza con una tarea de *Seleccionar Artículo* para luego pasar a la actividad *Solicitar Artículo* y sucesivamente ir cumpliendo los distintos pasos hasta recibir el pedido en su domicilio, registrando el pago del mismo.

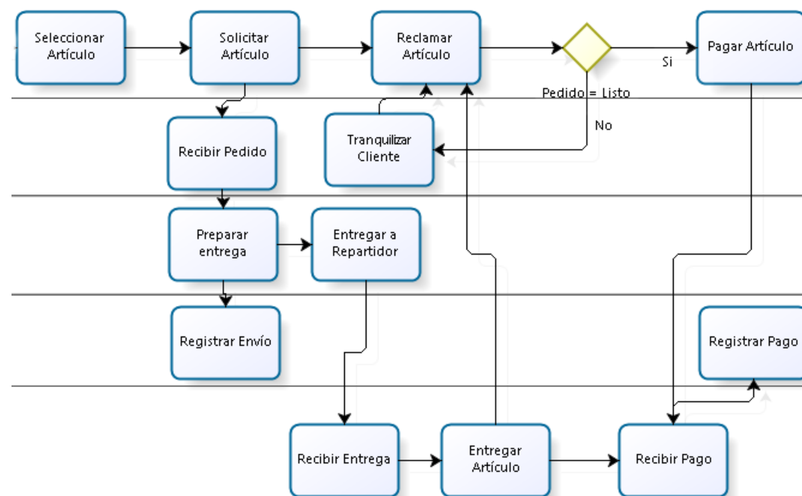


Figura 2.1: Ejemplo de actividades realizando una composición de servicios para la compra de un Artículo

Estas actividades se pueden describir de dos maneras diferentes: *Orquestación* y *coreografía.*

- **Orquestación:** Describe cómo interactúan los servicios (o dispositivos) en un flujo de trabajo, incluyendo la lógica del negocio y el orden de las interacciones. El aspecto clave de las orquestaciones es que existe un único controlador de toda la composición. Esto se puede apreciar en la parte derecha de la figura 2.2, donde el servicio denominado **Servicio Orquestador** es el encargado de coordinar la orquestación. Un lenguaje de orquestación de servicios es BPEL (*Business Process Execution Language*, Lenguaje de Ejecución de Procesos de Negocio), o WS-BPEL para la orquestación de servicios web (Rosen et al., 2008). En el ámbito de los sistemas ubicuos podemos nombrar como entornos a Amigo (Ami, 2019) , Aura (Harkes et al., 2002), Gaia (Campbell et al., 2002) OpenHab (ope, 2019).

- **Coreografía:** Describe la secuencia de mensajes entre servicios (o dispositivos), 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 el proceso general de ejecución, cuándo ejecutar sus operaciones y cómo interactuar con otros servicios (Rosen et al., 2008). En la figura 2.2 esto puede observarse en la parte izquierda de la misma, donde todos los servicios interactúan entre ellos sin un coordinador central. Un lenguaje que describe este tipo de composición es WS-CDL, el cual está orientado a Servicios Web. Los trabajos de (Palmieri, 2013) y de (Viroli, 2013) en el ámbito de los dispositivos ubicuos se asemejan a esta forma de coordinación. También existen entornos orientados a coreografías, como es el framework KNX (knx, 2019).

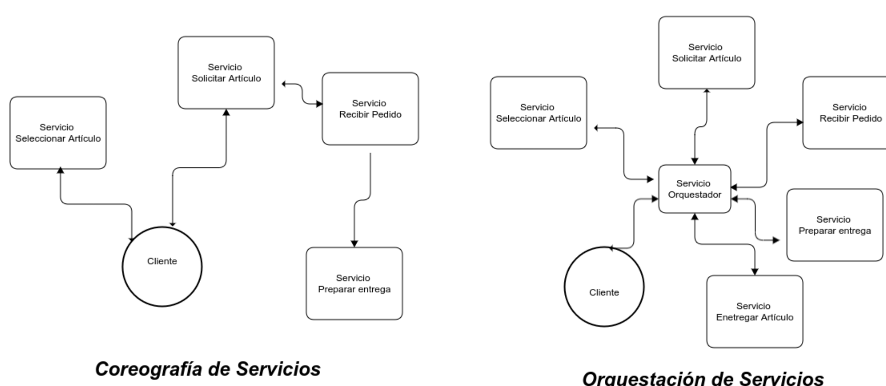


Figura 2.2: Coreografía vs. Orquestación.

Los mecanismos de definición de la composición (tanto orquestaciones co-

mo coreografías) deben poseer dos características principales: *Expresabilidad* y *corrección.*

- *Expresabilidad:* Se refiere al poder expresivo que posee un lenguaje de modelado. Este lenguaje debe permitir al desarrollador modelar las interacciones entre los servicios de la composición. Esto implica poder expresar estructuras complejas como secuencias, selecciones, iteraciones, concurrencia, asignación de actividades a roles, representación de datos, restricciones temporales, manejo de excepciones y transacciones.
- *Corrección:* La corrección se refiere a la habilidad de asegurar que la composición responde a los requisitos y requerimientos funcionales relevados.

Al cumplir con estas características, el lenguaje de modelado facilita el trabajo del desarrollador. En caso de que el lenguaje no cumpla con estas características, la tarea del desarrollador tendrá mayor complejidad y la composición será más propensa a sufrir errores. Tanto el lenguaje de especificación de orquestaciones WS-BPEL como el de coreografías WS-CDL cumplen con estas características.

### 2.1.2. Selección de los servicios de la composición

Por cada actividad que se ha modelado en el paso anterior, y en base a las necesidades de dicha actividad, se seleccionan los servicios (o dispositivos) candidatos a cumplir dicha tarea. Por ejemplo, para la actividad **Recibir Pedido** mostrada en la figura 2.1, es necesario seleccionar los servicios que sean candidatos a cumplir la actividad, como puede ser: recepcionar orden de pedido, comprobar correctitud orden, comprobar disponibilidad, etc.

Esta selección se hace a través de una búsqueda en un registro de servicios o, si no existieran servicios del tipo adecuado, se procedería a la confección de servicios específicos. En caso de que haya más de un servicio candidato, se seleccionará aquel que se adapte mejor a los necesidades de la tarea. Esta selección de servicios se puede llevar adelante de dos maneras posibles: *estática* o *dinámica.*

- **Estática:** La selección de servicios es en tiempo de diseño. Los servicios que componen la solución son elegidos y puestos en funcionamiento. Este tipo de selección es válida cuando los servicios, y las operaciones que proporcionan, rara vez son modificados o cambiados. Este tipo de selección no permite cambios ni adaptaciones en tiempo real, en caso de que éstas fueran necesarias. Las orquestaciones definidas en WS-BPEL son del tipo estático, así como las coreografías que se realizan en OpenHab (ope, 2019). Mosquito (moq, 2019) es otro ejemplo de

un proyecto que realiza composiciones estáticas con un coordinador central.

- **Dinámica:** Por el contrario con la anterior, este tipo de selección sí permite determinar y reemplazar los componentes en tiempo de ejecución. En esta forma el descubrimiento, la identificación, la selección y la composición de los servicios se hace en tiempo de ejecución, de manera tal que solo al momento de la ejecución se realizan estas tareas.

  La composición dinámica es ideal para los entornos de servicios web debido a su naturaleza dinámica, esto es, los servicios al estar en la nube, pueden sufrir desapariciones o contingencias sin previo aviso. Sin embargo la selección dinámica de servicios es una actividad desafiante, ya que se deben considerar una serie de cuestiones importantes, como la corrección de la composición, los límites de tiempo, el soporte transaccional, etc. Es decir que, pueden producirse fallas al momento de realizar la composición.

  En este punto podemos mencionar como ejemplo, por un lado el trabajo de (Najar et al., 2014), donde se propone un mecanismo de descubrimiento dinámico, el cual puede ser utilizado tanto para coreografías como para orquestaciones. Aura (Harkes et al., 2002) o Gaia (Campbell et al., 2002) son ejemplos de frameworks que tienen descubrimiento dinámico pero de una forma centralizada como si fuese una orquestación.

También debemos mencionar la manera en que estos servicios pueden ser seleccionados para poder realizar la composición. Esto puede llevarse a cabo de forma: *manual* o *automática* o *semi-automática*.

- **Manual:** En este caso, el diseñador debe crear la composición manualmente utilizando algún lenguaje estándar de servicios web, como puede ser BPEL, OWL-S, o bien otra herramienta inespecífica como puede ser el caso de un diagrama de flujo de datos [1]. Más tarde, durante la etapa de implementación de la composición (ver sección 2.1.3), el programador deberá unir los servicios web de acuerdo a lo indicado en el diseño también de forma manual, atendiendo a los detalles particulares del API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) de cada servicio. Este tipo de selección consume mucho tiempo y es propenso a generar errores. Adicionalmente, este tipo de selección es de naturaleza inherentemente estática.

[1] En este caso el diseñador no dispondría de las características de expresabilidad o de correctitud proporcionadas por un lenguaje específico, pero ello no implica que la composición de servicios final no vaya a cumplir los requerimientos planteados durante la definición de la composición (ver sección 2.1.1)

- **Automática:** La selección automática consiste en encontrar los servicios adecuados para la actividad seleccionada en tiempo de ejecución. Para ello se debe hacer una comparación entre los requisitos requeridos del servicio y las descripciones disponibles de los mismos. El soporte de automatización se considera la piedra angular para proporcionar un descubrimiento de servicios efectivo en entornos grandes y dinámicos. Hay dos formas de implementar el descubrimiento de servicios: coincidencia sintáctica y coincidencia semántica. Los métodos de coincidencia sintáctica solo ofrecen funciones de búsqueda basadas en nombres e identificadores, en cuyo caso la precisión de encontrar un servicio adecuado es baja. Por otro lado, en los métodos basados en la semántica, los servicios proporcionan descripciones semánticas de muchos aspectos (por ejemplo, operaciones, entradas, salidas e incluso capacidades y comportamiento), para aumentar el nivel de automatización en el descubrimiento de servicios web.

  En caso de que la búsqueda de los servicios adecuados, ya sea a través de métodos sintácticos o semánticos, no se haga de manera correcta, se verá afectada la composición en su conjunto, ya que no se podrá disponer de servicios acordes o adecuados para las tareas que se necesitan llevar adelante(Sheng et al., 2014).

  El resultado del descubrimiento del servicio, realizada a través de una automatización por ejemplo, podría ser un conjunto de servicios con una funcionalidad similar (o idéntica) pero con requisitos no funcionales diferentes. Los sistemas de composición de servicios deben tener la capacidad de seleccionar el servicio con la mejor calidad de servicio disponible, entre los servicios candidatos. Dentro de los requisitos de calidad de servicio encontramos rendimiento, confiabilidad, escalabilidad, capacidad, etc.

  En caso de que los servicios seleccionados no sean los correctos, fundamentalmente por no cumplir con algunas de los requisitos no funcionales, puede hacer que la composición falle en su conjunto, debido a problemas insalvables a la hora de la ejecución de la misma(Sheng et al., 2014).

- **Semi-automática:** En este caso, solamente algunos pasos del proceso de composición es automático, y otros se realizan de forma manual. Es decir, algunos servicios se seleccionan de forma manual y otros de manera automática, dependiendo de la complejidad que requiera identificar un servicio que cumpla determinados requisitos (Sheng et al., 2014).

### 2.1.3. Implementación de la composición

En esta fase el servicio compuesto es implementado mediante la composición de servicios específicos para permitir su instanciación e invocación por los usuarios finales. El resultado de esta fase es un servicio compuesto ejecutable que, desde el punto de vista de los usuarios finales, es indistinguible de cualquier otro servicio.

En esta fase es donde se deben hacer los ajustes necesarios para que los servicios implementados puedan ejecutarse de manera correcta sobre la plataforma de producción. Por ejemplo, la actividad **Recibir Pedido** mostrada en la figura 2.1, necesita de los servicios **recepcionar orden**, **comprobar correctitud orden**, **comprobar disponibilidad**, los cuales deben ser compuestos dentro de la actividad.

### 2.1.4. Ejecución de la composición

En esta fase el servicio compuesto es instanciado y ejecutado por un motor de ejecución, el cual puede ser un controlador centralizado (orquestación) o distribuido (coreografía), el cual es también responsable de invocar cada uno de los elementos que lo componen.

Una vez que se ejecuta el servicio compuesto, su progreso debe ser administrado y monitoreado para obtener una visión clara de cómo funcionan el servicio compuesto y sus servicios componentes dentro del entorno de ejecución. Además, las propiedades y requisitos que se verifican en el momento del diseño pueden violarse en tiempo de ejecución. El monitoreo de la composición de los servicios implica varias actividades distintas que incluyen: registrar y ver detalles de ejecución del servicio compuesto y las instancias del servicio componente; obtener estadísticas de calidad de servicio mediante el análisis de los datos de ejecución del servicio compuesto y los servicios componentes (por ejemplo, tiempo de respuesta, rendimiento y disponibilidad); y verificar los requisitos funcionales, así como evaluar las propiedades no funcionales del servicio compuesto.

Los requisitos que típicamente se exigen a las servicios compuestos durante esta fase son los siguientes: adaptabilidad, escalabilidad y confiabilidad.

- *Adaptabilidad:* Esto implica que los servicios pueden no estar disponibles en un momento determinado, otros pueden aparecer para reemplazarlos, etc. La ejecución de servicios compuestos debe tener la capacidad de adaptarse o ajustar dinámicamente sus comportamientos para cumplir con los cambios de los requisitos al momento de su ejecución. Esta característica se basa fundamentalmente en la selección de los servicios o elementos que pueden utilizarse como reemplazo[2]

[2] En composiciones estáticas habría que tomar precauciones para que esto sea posible, ej.: seleccionar previamente los dispositivos alternativos.

- *Escalabilidad:* Cuando el uso del servicio compuesto aumenta, por ejemplo, porque se incrementa su base de usuarios, los servicios que forman parte de la composición serán invocados con mayor frecuencia. En consecuencia, la provisión de los servicios componentes debe realizarse a partir de plataformas que permitan realizar las invocaciones de manera eficiente y con buenos tiempos de respuesta.

- *Confiabilidad:* Hace referencia al al grado de robustez del sistema de composición de servicios frente a comportamientos excepcionales durante su ejecución. La fiabilidad implica el manejo y la recuperación de las excepciones generadas dentro del servicio compuesto y fallas generadas por las interacciones con otros servicios proporcionados por terceras partes.

Estos requisitos pueden no cumplirse en su totalidad y, sin embargo, la composición no verse afectada. No obstante, es importante que la composición de servicios se apegue lo que más se pueda a estas características, ya que la ausencia de alguna de ellas puede hacer que la ejecución sufra fallas o no se pueda realizar en su totalidad.

También en esta etapa se deberán comprobar los requisitos de calidad de servicio, monitoreo, escalabilidad, etc. que se especificaron en la fase de definición.

## 2.2. Composición de dispositivos ubicuos

En esta sección abordaremos distintos trabajos académicos relacionados a la coordinación de dispositivos ubicuos, donde se discuten tanto la forma como la manera en que se descubren servicios y dispositivos de manera tal que permitan la ejecución de composiciones, tal como hemos viste en las secciones anteriores.

### 2.2.1. Najar

(Najar et al., 2014) plantea que los sistemas de información deben adaptarse a las nuevas tecnologías que aportan los dispositivos ubicuos al ser integrados dentro del entorno de trabajo, lo que denomina la nueva generación de los sistemas de información pervasivos. Estos nuevos sistemas de información intentan incrementar la productividad del usuario haciendo que los servicios de los sistemas de información se encuentren disponibles en todo momento y lugar. Los sistemas de información tradicionales proporcionan su funcionalidad en entornos muy controlados, por ejemplo, un centro de cálculo. En contraste, los sistemas de información pervasivos deben soportar múltiples dispositivos y servicios heterogéneos, con todos los problemas que esto implica.

La transparencia y la proactividad son factores claves en el diseño de los sistemas de información pervasivos. Estos sistemas deben poder ofrecer servicios a los usuarios teniendo en cuenta sus objetivos y el contexto donde se encuentran inmersos, predecir los nuevos objetivos que se puedan plantear a los usuarios en dicho contexto. Por ejemplo, podríamos pensar que de acuerdo a la historia con que una persona se ha desplazado dentro de un aeropuerto en viajes previos, se le podrían ofrecer servicios de cafetería, visitar áreas de descanso hasta tomar su vuelo.

Para proporcionar una solución al problema planteado, los autores proponen un modelo de predicción y descubrimiento de dispositivos centrado en el usuario, basado en el contexto en que el usuario se encuentra y en sus intenciones. Estas intenciones son las que permiten identificar el objetivo que el usuario quiere satisfacer cuando tiene una necesidad a cubrir. A su vez, estas intenciones se producen en un contexto determinado que puede condicionar la implementación de los servicios en los dispositivos disponibles. Por lo tanto, el modelo propuesto intenta predecir las necesidades del usuario, basándose en su historia, para ofrecerle los servicios que más se adecúan a sus necesidades reales. Para poder seleccionar el servicio más adecuado a las necesidades del usuario se utilizan ontologías y emparejamientos semánticos.

Como ejemplo podemos mencionar una persona que realiza viajes en avión de manera constante; por lo tanto su estadía en aeropuertos es asidua. De acuerdo al contexto (es decir al lugar en donde se encuentra) y al comportamiento agendado de esta persona en el pasado, el sistema puede proponerle servicios a medida que se desplaza dentro del aeropuerto, como puede ser un bar que sirve determinada comida del gusto del cliente, salas de descanso en base al horario del vuelo, etc.

En este trabajo **no se hace mención a la forma en que los dispositivos se comunican o se relacionan entre sí luego de que fueron descubiertos, de forma tal que puedan colaborar para satisfacer las necesidades del usuario. Los autores se centran exclusivamente en los algoritmos, pero no hacen mención a ningún mecanismo que permita manejar a estos dispositivos y que brinde un protocolo de composición.**

### 2.2.2. Loke

(Loke, 2012) enfoca la problemática de coordinación de sensores, móviles y otros dispositivos en ambientes sensibles al contexto, combinados para brindar información en la nube referida tanto a los estados de los sensores como de la información del contexto de los mismos. Esto es, aborda la problemática de la selección de dispositivos, para luego coordinarlos, tal y como hemos indicado en la sección2.1.

El fundamento del trabajo de (Loke, 2012) es la definición de *cloudlets*. Un cloudlet es un conjunto o cluster confiable de computadoras con variedad

de recursos disponibles para ser utilizados por dispositivos móviles cercanos a ellas. También define los conceptos de *sensor-cloudlet* y *context-cloudet*. El primero se refiere a sensores como recursos. El segundo se refiere a recursos con información del contexto que lo rodea. A su vez los servicios provistos por el cloudlet tienen ciertas características que lo diferencian de los servicios cloud, a saber: son transitorios, focalizados únicamente al ambiente actual que los rodea, son descentralizados, trabajan principalmente en LAN y, por último, son pocos los usuarios conectados al mismo tiempo.

A partir de este nuevo modelo de aplicaciones, se pueden utilizar dispositivos sensibles al contexto disponibles como aplicaciones en la nube, en aplicaciones específicas, a demanda y con un pago por la utilización de esa información. Inicialmente plantean un estudio del estado actual de la problemática, para luego plantear la aplicación de un modelo abstracto de sistemas sensibles al contexto para la composición de dispositivos en dichos ambientes basados en servicios[3] en la nube.

A manera de ejemplo podemos mencionar las tiendas minoristas de un centro comercial. Una serie de lectores RFID (*Radio Frequency Identification*, Identificación por radiofrecuencia) almacenan información de productos de las tiendas. A partir de allí, un cliente del centro comercial dispone de una aplicación en su teléfono móvil que le permite leer el contenido de estos dispositivos RFID donde puede consultar sobre productos de su lista de compras, para de esa manera poder saber qué locales comerciales cercanos a su posición poseen dichos productos. Estas aplicaciones, podrían ser libres o de pago, en el caso de las aplicaciones que se instalan en el cliente podrían ser libres y en cambio la de las tiendas podrían ser pagas.

El modelo se plantea como una extensión del lenguaje de Prolog. Como ventajas de la utilización del lenguaje prolog, los autores mencionan que la composición de recursos puede realizarse de forma declarativa utilizando expresiones de alto nivel de abstracción, muy útiles para programar aplicaciones. Otra ventaja es que la declaración descriptiva de la composición de recursos facilita el análisis de los costos y confiabilidad de las composiciones. Por último, los operadores de meta nivel son útiles para administrar la ejecución de las expresiones.

Desde la perspectiva de esta tesis, el trabajo de Loke **no aborda adecuadamente la composición de dispositivos, al igual que en el trabajo de Najar, no se hace mención acerca de la forma en que estos dispositivos puedan coordinarse entre ellos para en conjunto llevar adelante una tarea específica, sino que por el contrario, existe un centralizador que concentra la conectividad de estos dispositivos para brindar información al cliente. Además, trabaja sobre redes de tipo LAN principalmente**.

[3]El concepto de servicio aquí utilizado es el de servicios brindados por cloudcomputing, no como servicios de SOA

### 2.2.3. Palmieri

Existen otros trabajos como el de (Palmieri, 2013) donde se toma la problemática de la composición de dispositivos desde un punto de vista del descubrimiento de los mismos en forma dinámica.

El autor plantea la necesidad de encontrar nuevas formas de descubrir servicios ya que la arquitectura centralizada tradicional está cambiando a una arquitectura de a pares (peer to peer), donde la disponibilidad computacional (computadoras, móviles, dispositivos ubicuos, etc.) puede estar en redes totalmente heterogéneas accesibles independientemente del lugar donde se encuentre. Para ello proponen una nueva manera de descubrimiento basada en un modelo de búsqueda totalmente distribuida y paralela, el cual no requiere de ninguna inteligencia centralizada, de roles establecidos, ni de mecanismos de comunicación estables.

En este modelo o framework de búsqueda y descubrimiento de servicios, cada dispositivo participa activamente en el proceso de descubrimiento a través de dos maneras: por un lado en la publicación de los servicios, y por otro, en las actividades de búsqueda de los mismos. Para ello, adaptaron el algoritmo de búsqueda de camino aleatorio basado en la idea de paso lento a través de los distintos nodos(Sarshar et al., 2004; Sarshar et al., 2006), donde introdujeron algunas mejoras como evitar que se produzcan bucles en la búsqueda y la optimización de la selección del camino a utilizar en la búsqueda, de manera que no sea absolutamente aleatoria. De esta forma se obtiene un algoritmo que puede encontrar un servicio ofrecido en una red de baja potencia de una organización con un esfuerzo de búsqueda relativamente bajo.

Si bien este trabajo se aleja de los enfoques centralizados, se basa únicamente en el descubrimiento y búsqueda de nuevos servicios en ambientes pervasivos, sin ser parte del trabajo un mecanismo o framework de comunicación posterior al descubrimiento de los mismos.

### 2.2.4. Viroli

(Viroli, 2013) estudia la aplicación de técnicas de auto-organización para la composición de servicios de software en el contexto de entornos altamente dinámicos y móviles, como es el caso de los sistemas pervasivos que intentan predecir las necesidades de las personas en un ambiente determinado. Este enfoque tiene por objeto solucionar los problemas que se producen en ambientes pervasivos cuando se intenta utilizar mecanismos de coordinación centralizados y controlados por el hombre.

Para este tipo de técnicas introduce la noción de espacio bioquímico de tuplas. En este modelo, las tuplas se asocian con un nivel de actividad, que se asemeja a la concentración química y mide el grado en que la tupla puede influir en el estado de coordinación del sistema; por ejemplo, una tupla con

bajo nivel de actividad sería bastante inerte, por lo tanto, participa en la coordinación con muy baja frecuencia.

Las reacciones de tipo químico, instaladas adecuadamente en el espacio de las tuplas, evolucionan el nivel de actividad de las tuplas a lo largo del tiempo de la misma manera que la concentración química se desarrolla en los sistemas químicos, bajo los supuestos de químicos simples en soluciones bien mezcladas. Estas reacciones están destinadas a promover la explotación de patrones químicos, ya sea inspirados por la química natural o diseñados desde cero, que pueden hacer surgir propiedades de auto-organización como mencionamos inicialmente.

Un beneficio principal de este enfoque es la capacidad de implementar una especie de comportamiento "ecológico" en sistemas coordinados: esto se encuentra particularmente útil en el contexto de la computación generalizada(Viroli y Zambonelli, 2010; Quitadamo et al., 2007), donde es típico administrar aplicaciones complejas como ecosistemas de servicios generalizados y dispositivos, promoviendo patrones de competencia, extinción y auto-organización espacial.

Si bien el enfoque es novedoso por su inspiración en las reacciones químicas, **es un desarrollo teórico, donde no se vislumbra cómo sería el framework de composición ya que el trabajo no menciona cómo se produce la coordinación de los distintos dispositivos. A su vez, en este trabajo no se menciona la utilización de dispositivos con menores prestaciones como es el caso de sensores o actuadores que, si bien se nombran, no son tenidos en cuenta durante el desarrollo del trabajo**.

## 2.3. Frameworks comerciales

En esta sección abordaremos el estudio de distintos frameworks existentes que trabajan sobre dispositivos ubicuos. Nos hemos abocado al estudio de OSGi, MOSQUITO, Android Studio, OpenHab y Z-wave.

### 2.3.1. OSGi

En primer lugar hablaremos de OSGi. Para ello organizaremos su presentación en distintas subsecciones con el fin de presentar de una manera clara sus características: Qué es, Arquitectura, Popularidad y Composiciones.

#### 2.3.1.1. Qué es OSGi

El OSGi es la respuesta en la plataforma Java a la programación modular. Un módulo es un componente autocontenido dentro de un sistema mayor que no necesita de referencias externas para su existencia y funcionamiento. Detrás de esta definición de módulo tan abstracta se esconden dos

de los principios fundamentales de la programación: Alta Cohesión y Bajo Acoplamiento.

Todo módulo debe tener una responsabilidad y funcionalidad perfectamente delimitada dentro de un sistema mayor en el que se encuentre integrado (alta cohesión). Un módulo debe también evitar cualquier dependencia con el resto de los módulos para realizar la tarea que tiene asignada (bajo acoplamiento). Inevitablemente, siempre existirá un cierto grado de acoplamiento, que debe estar, en la medida de lo posible, articulado mediante conceptos abstractos, lo que en Java vienen a ser las interfaces.

La tecnología OSGi permite 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 otros módulos. La componentización de módulos de software y aplicaciones garantiza una interoperabilidad y manejo remoto de aplicaciones y servicios sobre una gran variedad de dispositivos.

#### 2.3.1.2. Arquitectura de OSGi

La especificación OSGi propone una arquitectura implementada sobre una JVM (*Java Virtual Machine*, Máquina Virtual Java) que permite la instalación, actualización y eliminación de módulos, tal y como muestra el gráfico 2.3. Cada módulo podrá publicar las clases que serán visibles por otros módulos de la máquina virtual y podrá consumir cualquier clase que haya sido publicada por otros módulos de la VM (*Virtual Machine*, Máquina Virtual). Entre los módulos cliente y proveedor existe un registro central, en el que los proveedores registran los servicios que desean hacer públicos al resto de módulos y los clientes solicitan los servicios que desean consumir. Evidentemente un módulo puede ser, a la vez, cliente y proveedor.

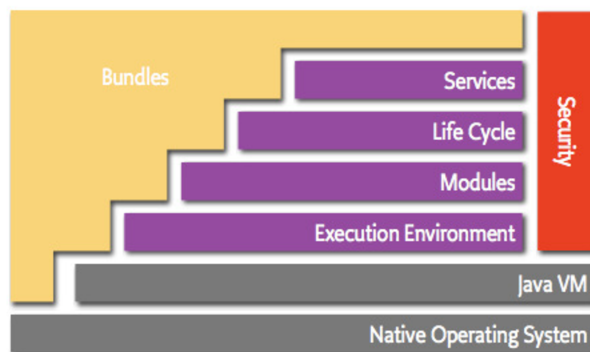


Figura 2.3: OSGi - Modelo de capas. Fuente - https://www.osgi.org/developer/architecture/layering-osgi/

#### 2.3.1.3. Popularidad

Esta plataforma de desarrollo no fue muy popular en sus comienzos (año 1999). Esto ha cambiado en los últimos años debido a que muchos fabricantes se han dado cuenta de la idoneidad de OSGi para constituir la base de las próximas versiones de servidores de aplicaciones Java. Prueba de este interés es que frameworks Java como Eclipse y Spring ya están basadas en él, mientras que Apache e IBM ya han preparado también versiones de sus servidores de aplicaciones Java EE (Felix y WebSphere, respectivamente) siguiendo la filosofía OSGi.

#### 2.3.1.4. Composición en OSGi

La composición de servicios en esta plataforma se hace de una manera dinámica ofreciendo primitivas estándares para que la misma se pueda llevar adelante de forma sencilla, por ejemplo a través de la utilización de BPEL. La composición puede ser realizada a través de la relación entre aplicaciones o bien componentes de software (por ej. librerías) o bien servicios. Esta composición puede ser realizada de dos maneras posibles: de forma dinámica a través de la utilización de "service binding", tal como denominan a la forma en que en tiempo de ejecución los servicios pueden ser descubiertos (los servicios se publican de manera dinámica en una carpeta virtual del servidor para que puedan ser descubiertos y utilizados en caso de ser necesario); por otro lado también existe la composición estática, que se realiza en tiempo de desarrollo.

**El problema principal de esta plataforma radica en la necesidad de contar con una máquina virtual Java para su ejecución, lo cual hace que los dispositivos que puedan formar parte de una composición es limitada, ya que debe contar con determinadas características de hardware y software como por ejemplo un sistema operativo. Este requisito de contar con un sistema operativo y una máquina virtual Java hace inviable su incorporación en dispositivos poco potentes como es el caso de la mayoría de los dispositivos ubicuos**.

### 2.3.2. Proyecto MOSQUITO

Este framework surge como un proyecto de investigación de la Unión Europea. Los autores (Hayat et al., 2007) tienen como objetivo proveer un acceso seguro, confiable y ubicuo a aplicaciones de negocio. Para ello desarrollaron una infraestructura técnica, donde los usuarios puedan realizar procesos de negocio de una manera colaborativa, a través de la utilización de sus dispositivos móviles. La arquitectura del mismo puede verse en la figura 2.4. Allí podemos apreciar el modelo en capas del proyecto, donde por encima se

encuentra la capa de aplicaciones junto a las políticas de seguridad aplicadas, luego una capa de middleware que permite conectar las aplicaciones con la plataforma en si, que es la capa inferior.

El corazón de la arquitectura de MOSQUITO es un middleware[4] que incluye un conjunto de servicios de seguridad. Las aplicaciones son desarrolladas en la parte más alta del middleware lo que asegura la seguridad de las mismas. A su vez el framework permite la incorporación de información del contexto a través de la utilización de sensores.

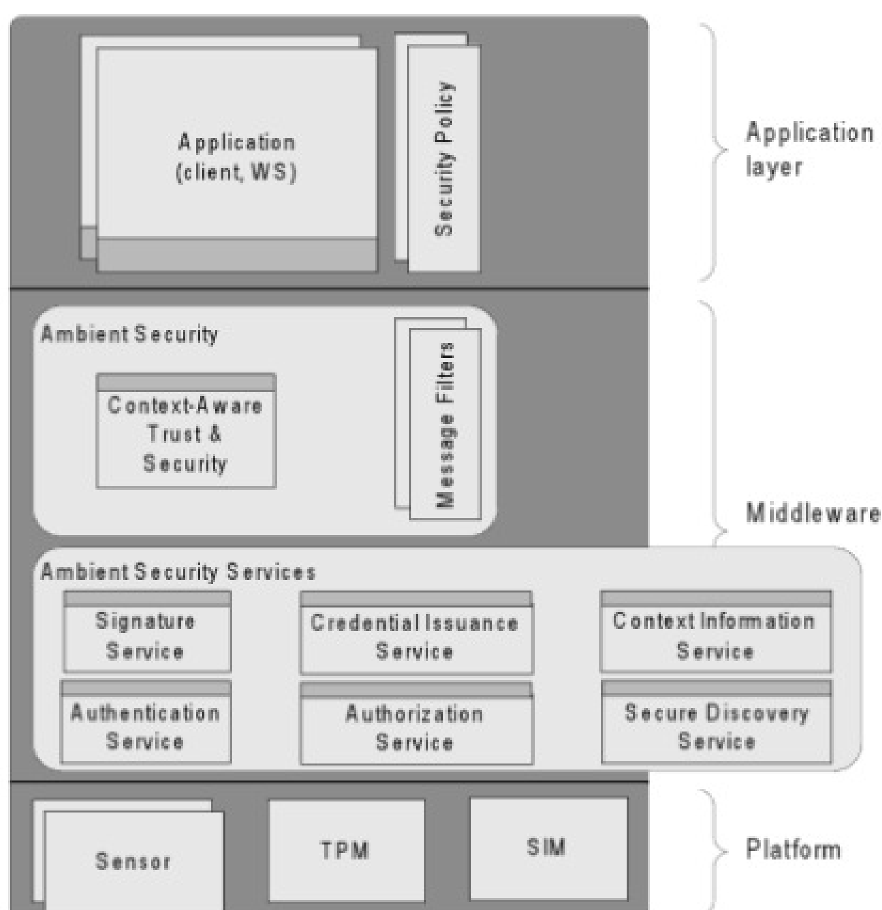


Figura 2.4: Arquitectura MOSQUITO Fuente: (Mühlhäuser y Gurevych, 2008, pag. 576)

Este framework está enfocado principalmente en brindar seguridad a los accesos a los sistemas de negocio desde distintos dispositivos ubicuos como

[4]Por middleware entendemos que es el software que permite conectar con otro software, permitiendo el flujo de datos de una aplicación a otra.

es el caso de móviles o sensores que brindan información del contexto. Si bien es un punto importante la seguridad en ambientes de este estilo, los autores **no brindan un panorama de cómo estos dispositivos deben coordinarse entre ellos**.

### 2.3.3. Android Studio

Otro entorno de desarrollo es el que se plantea en Android Studio, 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 es el caso de los sensores. Estos dispositivos necesitan de la conexión a otro dispositivo de mano (un teléfono móvil, por ejemplo), con mayor capacidad de procesamiento para poder llevar adelante las tareas de comunicación y acceso a la red.

Esta plataforma permite a su vez emular dispositivos ubicuos para hacer distintos testeos de los desarrollos realizados.

Si bien Android Studio es bastante popular, no deja de ser una plataforma de desarrollo. El desarrollador es el responsable de definir el comportamiento y coordinación de los dispositivos. **La plataforma no brinda un protocolo o mecanismo de comunicación entre los dispositivos más o menos estandarizado que puede ser reaprovechado por el desarrollador. Finalmente, los dispositivos wereables necesitan de otros dispositivos con características más avanzadas para poder comunicarse, esto es, Android Studio considera los dispositivos wearables como satélites de un dispositivo principal, y no como dispositivos ubicuos que puedan actuar independientemente en una composición**.

### 2.3.4. OpenHab

OpenHab es un framework para administrar dispositivos presentes en un hogar, esto es, dispositivos típicamente usados en Domótica. Esta plataforma está desarrollada en Java y es una implementación del framework OSGi, por lo que es de código abierto. Esta plataforma necesita estar instalada sobre una computadora que actúa como integrador de todos los dispositivos, permitiendo administrarlos y tener información del estado de cada uno de ellos. Cada dispositivo perteneciente a un fabricante determinado puede exponer una serie de funcionalidades extras (no solamente mostrarse como activos o inactivos a un momento determinado) que permiten obtener información específica del mismo, y de esta manera poder tener un mayor control del mismo desde el framework.

Si bien un punto importante de esta plataforma es su carácter abierto, **posee el inconveniente de que se exige que los dispositivos tienen que implementar el mismo estándar (por ej. KNX, HTTP (*Hyper-***

***text Transfer Protocol*, Protocolo de Transferencia de Hypertexto), iCloud, etc.) para poder estar interconectados. Esto perjudica la utilización de dispositivos de los distintos fabricantes. Finalmente, es necesario que los dispositivos dialoguen con este nodo central y no entre ellos para poder realizar una acción determinada, esto es, la única modalidad de composición permitido en esta plataforma es la orquestación, pero no la coreografía**.

### 2.3.5. KNX

KNX es un sistema de instalación de domótica, tanto para casas particulares como para edificios. Este sistema está basado sobre la norma estándar internacional ISO/IEC 14543-3, lo que permite que dispositivos de distintos fabricantes puedan adaptarse a dicha norma para poder ser integrados dentro del sistema.

La forma de comunicación por excelencia de KNX es el par trenzado (TP1), pero además permite conexión a través de PL110 (red eléctrica) y Ethernet (IP). Es decir: todas las formas de interconexión de dispositivos es a través de un sistema cableado; no tiene la posibilidad de interconectarse a través de protocolos inalámbricos.

Una de las características relevantes de KNX es su arquitectura distribuida; esto implica que no es necesario un controlador central (por ejemplo un ordenador o un autómata) para controlar la instalación, ya que cada elemento del sistema dispone de su propia inteligencia y puede comunicarse libremente con otros dispositivos.

Otra de las características importantes de KNX es que se pueden configurar mediante un software único todos los elementos (independientemente del modelo y fabricante). Con dicho software se realiza tanto el diseño y programación del proyecto como la puesta en marcha, mantenimiento y diagnóstico de la instalación.

En contrapartida, **KNX posee diversos inconvenientes para representar una solución al problema de investigación planteado. En primer lugar, KNX es un sistema propietario, que a su vez tiene costos de licenciamiento a partir de la administración de más de 5 dispositivos. Además no existe la posibilidad de que los dispositivos compatibles con KNX puedan dialogar con otros equipos o computadoras no basadas en este sistema para llevar adelante un objetivo mayor como puede ser un sistema integrado. Finalmente, KNX se focaliza únicamente en la domótica, lo que a su vez está relacionado con comunicaciones de cierto alcance, fiables y seguras, y probablemente también con un suministro ininterrumpido de electricidad. Estas características no están aseguradas en los ambientes pervasivos**.

### 2.3.6. Z-wave

Z-Wave es una tecnología de comunicación inalámbrica sin estandarizar, aunque existe la Alianza Z-Wave para garantizar la interoperabilidad de los dispositivos empleados. El estándar es cerrado, y por tanto es necesario ser miembro para acceder a él. Aún así, es fácil encontrar documentación de este protocolo.

La topología de red es tipo malla con un controlador central, y cada elemento se comporta como un nodo que puede ser receptor, emisor o relay. Permite realizar agrupaciones para asociar la misma funcionalidad a todos los elementos del grupo. Igualmente permite hacer escenas, esto es, encadenar una serie o conjunto de acciones en función de los eventos que se producen en un hogar, ya sea como disparador de una acción programada o no programada (por ej. encender la luz de una habitación cuando se abre la puerta).

Si bien esta tecnología es bastante utilizada, tiene como contrapartida que **es un protocolo de comunicación privado, lo que deriva en costos de instalación, los cuales pueden ser elevados. A su vez, al ser un protocolo privativo, se corre con el riesgo riesgo de no poder integrar otros dispositivos de otros fabricantes que no se adapten a dicho protocolo**.

## 2.4. Otros Frameworks académicos

En esta sección abordaremos el estudio de distintos frameworks académicos existentes que trabajan sobre dispositivos ubicuos.

### 2.4.1. Aura

**Aura**(Harkes et al., 2002), de la Universidad de Carnegie Mellon, está específicamente pensado para ambientes pervasivos basándose en comunicaciones inalámbricas, wearables, computadoras pequeñas y espacios inteligentes. El foco principal de este proyecto se basa en que la atención del usuario no está enfocada únicamente en estos dispositivos, sino que por el contrario, el usuario debe estar atento a otros estímulos (como es el caso de conducir un vehículo, caminar, etc).

Por lo tanto, el objetivo principal de Aura es el de proveer a cada usuario con un halo invisible de servicios de información más allá del lugar que se encuentre. Para llevar adelante este objetivo principal se centra en el concepto de proactividad, que es la capacidad del sistema en anticipar las actividades que va a realizar el usuario, para lo cual adapta y ajusta los dispositivos cercanos para que brinden las prestaciones que necesita la persona. Esto lo consigue a través de servicios de información contextual que almacenan información recolectada en tiempo real y en base a demandas, en bases de

datos virtuales, para luego ser consultadas en la medida que se necesitan.

Para poder brindar este halo de servicios invisibles al usuario, Aura se enfoca en todos los aspectos de hardware de los dispositivos, para lo cual se deben realizar adaptaciones específicas en los servicios para que estos dispositivos se puedan integrar. A su vez, debe existir un middleware para que estos dispositivos puedan interconectarse y entenderse para poder desarrollar sus actividades. Es decir, cuenta con una capa intermedia, el middleware, que trabaja como concentrador de los dispositivos conectados a cada momento, para poder luego saber qué servicios pueden brindar en un momento determinado.

**Este framework se encuentra alineado con nuestros objetivos de investigación, aunque con diferencias notables con nuestra propuesta, las cuales se describirán en el capítulo de Discusión 7**.

### 2.4.2. Gaia

**Gaia**(Campbell et al., 2002), de la Universidad de Illinois, plantea que los espacios físicos se convierten en espacios activos a partir de la utilización de dispositivos ubicuos y proponen un sistema operativo para manejar dichos dispositivos. El sistema operativo propuesto proporciona las funciones más comunes de cualquier sistema operativo, como es el caso de señales, eventos, sistemas de archivos, seguridad, procesos, etc. Adicionalmente, este sistema operativo extiende los conceptos comunes para incluir información del contexto, reconocimiento de ubicación, dispositivos móviles y actuadores. Todas estos conceptos se encuentran disponibles en el middleware que permita manejar o administrar los recursos contenidos en los espacios activos. El sistema operativo debe ser capaz localizar el dispositivo más apropiado para una determinada tarea o acción, detectar nuevos dispositivos que se incorporan al espacio y adaptar los datos que se intercambian a distintos formatos en caso de ser necesario para que sean interpretados por otros dispositivos.

El objetivo de este proyecto es el de extender los conceptos de los sistemas operativos tradicionales para que puedan incluir información del contexto, conocimiento de la ubicación, dispositivos móviles, actuadores, etc.

**Si bien este proyecto intenta integrar los dispositivos presentes en los espacios activos donde se encuentran, tiene como desventaja el hecho de tener que instalar un middleware al cual todos deben conectarse y desde donde se realiza la conectividad entre los dispositivos**.

### 2.4.3. Oxigen

**Oxigen**(Science y Laboratory, 2002), perteneciente al MIT (*(Massachusetts Institute of Technology*, Instituto de Tecnología de Massachusetts), persigue que los dispositivos sean incorporados en la vida cotidiana de manera

natural e imperceptible.

Los creadores del proyecto consideran que las computadoras o dispositivos deben estar al servicio de las personas y no a la inversa como es habitual inclusive al día de hoy, donde debemos adaptarnos al lenguaje de estos dispositivos para poder interactuar con ellos. El objetivo de este proyecto se basa en que las personas no tengan que llevar los dispositivos, sino por el contrario, estos deben ir adaptándose a los distintos ambientes donde las personas se van desplazando, haciendo que éstos se comporten como dispositivos genéricos altamente configurables. Además, la forma de comunicación entre los dispositivos y las personas es a través del lenguaje natural. Oxigen permite que los dispositivos puedan ser integrados a la vida cotidiana de las personas, colaborando en sus tareas. De esta forma los dispositivos permiten conocer ubicaciones de las personas, comunicarlas a otros dispositivos para que éstos puedan percibir las necesidades de las personas y brindarle servicios acordes a sus necesidades. Más aún, la comunicación sería a través de gestos o en el lenguaje humano, por lo que se podría pedir: imprimir gráfico en la impresora color más cercana.

**Si bien este proyecto se asemeja bastante a nuestra investigación, ya que los dispositivos deben coordinarse entre ellos para poder brindar los servicios a las personas, el objetivo principal del mismo es justamente brindar servicios a las personas. No se encuentra claro de qué forma o bajo qué protocolos se realiza la coordinación de los dispositivos**.

### 2.4.4. Amigo

**Amigo**(Ami, 2019), desarrollado por Amigo Consortium, 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.

Amigo proporciona un middleware interoperable y estandarizado de servicios destinados al usuario o persona, de manera inteligente para el entorno hogareño o domótica. Ofrece a los usuarios una interacción intuitiva y personalizada al proporcionarle una interoperabilidad entre servicios y los dispositivos que forman parte del hogar, de manera que éstos pueden ser descubiertos de una manera automática a través de servicios de descubrimiento.

**En este proyecto, al igual que en los anteriores, se basa principalmente en brindar servicios a las personas que se integran en un determinado espacio inteligente, a través de la utilización de un middleware. Si bien existen coincidencias con este trabajo de investigación, existen diferencias importantes que se discutirán en el capítulo de Discusión 7**.

## 2.5. Especificaciones asociadas a paradigmas específicos

En lo que sigue se realizará un estudio de las distintas especificaciones estándar del W3C sobre coreografía y orquestaciones de servicios, ellas son: WS-CDL (conocida también como WS-Choreography), WSCI, y la especificación de OASIS respecto de orquestación de servicios que es WS-BPEL. También existe JADE que permite la implementación de sistemas multiagentes en lenguaje JAVA, implementando la especificación FIPA (*Foundation for Intelligent, Physical Agents*, Fundación para Agentes Físicos Inteligentes).

Algunas de estas especificaciones mencionadas poseen frameworks de desarrollos específicos como puede ser el caso de JADE o de WS-BPEL.

### 2.5.1. SOA

Desde finales de la década de 1990 la computación orientada a servicios SOC (*Software Oriented Computing*, Computación Orientada a Servicios) ha emergido como un paradigma importante de computación que cambió en gran medida la manera en que hoy en día las aplicaciones de software son diseñadas, entregadas y consumidas.

La orientación a servicios remonta sus comienzos a mediados de la década de 1980, cuando llegó al mercado la computación distribuida y las llamadas a procedimientos remotos (Rosen et al., 2008). A mediados de 1996, se define por primera vez la arquitectura orientada a servicios, la cual tomó mucha importancia a partir de dos grandes causas: la falta de aceptación de los modelos de computación distribuida de los años 90 (DCE (*Distributed Computing Environment*, Ambiente Distribuído de Computación) y CORBA (*Common Object Request Broker Architecture*)) y por la aparición en el mercado de los servicios web, los cuales tuvieron una gran aceptación. A partir del año 2003 SOA toma fuerza y popularidad dentro de la computación distribuida.

En la SOC, los servicios son utilizados como piezas fundamentales para dar soporte al desarrollo de aplicaciones de manera rápida y a bajo costo, principalmente en ambientes heterogéneos (Yu et al., 2008; Papazoglou y van den Heuvel, 2007; Bouguettaya et al., 2013). SOA es una arquitectura flexible y adaptable que permite integrar fácilmente sistemas, datos y aplicaciones mediante la utilización de servicios (Pant y Juric, 2008; Yu y Ong, 2009). Esta arquitectura permite la creación de sistemas de software altamente escalables que reflejan el negocio de la 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. SOA minimiza la brecha semántica entre los modelos de proceso de negocio y las aplicaciones de software actuales (Pant y Juric, 2008), permitiendo un mejor alineamiento de las TI (*Information Technology*, Tecnologías de la Información) con las necesidades de negocio (Rosen et al., 2008; Zimmermann O, 2013; Inaganti y Behara, 2007). SOC se basa en SOA

para organizar las aplicaciones de software y la infraestructura en una serie de servicios interactuando.

### 2.5.2. Web Service Choreography Description Language (WS-CDL)

#### 2.5.2.1. Qué es WS-CDL

WS-CDL es una especificación surgida del W3C Web Services Choreography Working Group(W3C, 2006). Los trabajos del grupo finalizaron en julio de 2005, proponiendo WS-CDL como candidato a Recomendación (”Candidate Recommendation"). El W3C no ha promocionado esta especificación en los últimos años, a diferencia de la especificación WS-BPEL para orquestaciones, la cual es utilizada en la definición de procesos de negocio.

WS-CDL define modelos de procesos de negocio basados en XML (*eXtensible Markup Language*, Lenguaje de Marcas Extensible), los cuales se basan en protocolos de colaboración entre participantes que exponen su funcionalidad mediante servicios Web. En WS-CDL, los servicios actúan como pares, y las interacciones pueden tener un ciclo de vida y estados de larga duración.

La especificación de la coreografía de servicios Web está pensada para la composición de cualquier tipo de participantes sin importar la plataforma de soporte o el modelo de programación utilizado para la implementación del servicio.

Desde el punto de vista del lenguaje, WS-CDL proporciona una descripción de la interacción entre servicios, así como una definición de los formatos de información que intercambian los participantes. La perspectiva del intercambio de mensajes entre los participantes es global, esto es, no depende de un punto de vista específico como en el caso de las orquestaciones. WS-CDL ofrece reglas que utiliza cada participante para analizar el estado de la coreografía y deducir cuál podría ser el siguiente intercambio de mensajes.

#### 2.5.2.2. Arquitectura WS-CDL

WS-CDL es una especificación organizada por capas, permitiendo diferentes niveles de expresión de las coreografías de una composición. En el nivel más alto, existe un paquete que contiene todas las definiciones realizadas por WS-CDL. Estas coreografías deben incluir como mínimo un conjunto de roles definidos por ciertos comportamientos, una serie de relaciones entre dichos roles, canales utilizados por los roles para interactuar y un espacio o bloque donde se especifican las interacciones que definen la coreografía propiamente dicha. En este espacio o bloque, se describe un conjunto básico de conexiones de servicios que permiten la colaboración entre roles para lograr un objetivo; sin embargo es posible adicionar una composición estructurada, permitiendo la combinación en secuencias o actividades paralelas de las interacciones y otras coreografías. Todo esto puede observarse en la Figura 2.5.

Para comprender mejor la estructura de una coreografía descripta en WS-CDL, mostraremos como ejemplo un código escrito utilizando esta especificación, donde se aprecia la interacción entre dos dispositivos que forman parte de la coreografía [5]. Desde el primer dispositivo denominado 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: *"reportarAccidente"* y *"publicarAccidente"*.

```
<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>
```

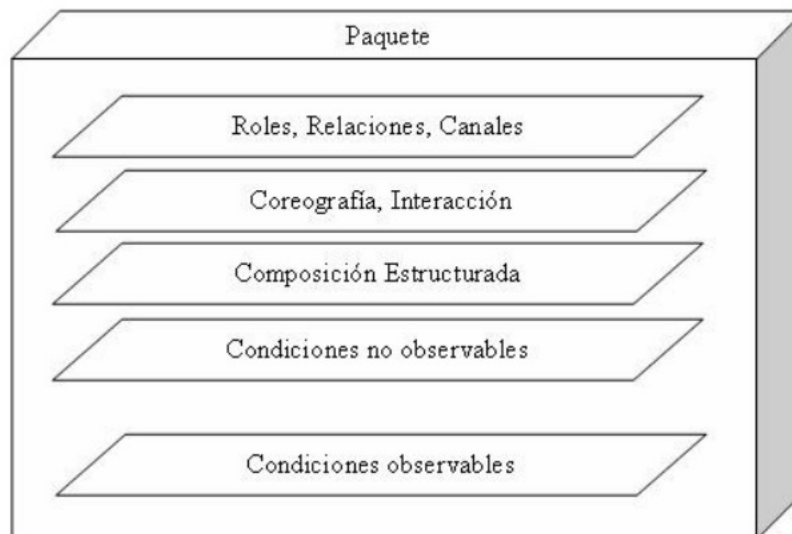


Figura 2.5: Arquitectura WS-CDL. Fuente: http://cic.puj.edu.co

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

#### 2.5.2.3. Uso académico e industrial

Esta especificación es poco utilizada dentro de la programación distribuida y los servicios web. Solamente en la especificación de procesos de negocio BPMN (*Business Process Model and Notation*, Modelo y Notación de Procesos de Negocio) 2.0 se comienza a dar soporte para el manejo de coreografías, pero no basándose en esta especificación. Al momento de escribir este trabajo, no existen implementaciones industriales de esta especificación que puedan ser utilizadas para realizar coreografía de servicios en la práctica.

A día de hoy, la utilización y estudio por parte de la academia sigue siendo importante, ya que se siguen realizando investigaciones relacionadas a la coordinación de servicios web a través de la utilización de coreografías como es el caso del estudio de ecosistemas de big data, la seguridad en la comunicación de los servicios web coreografiados, etc.

Si bien el tema de la coordinación de servicios web a través de coreografías se sigue estudiando, no se han realizado avances en la utilización de la especificación WS-CDL para la coordinación de dispositivos ubicuos.

### 2.5.3. Web Service Choreography Interface - WSCI

#### 2.5.3.1. Qué es WSCI

Dentro de las necesidades planteadas al trabajar con Servicios Web, y más aún en la composición de servicios, se plantean las siguientes preguntas:

- ¿Pueden los mensajes ser enviados y recibidos en cualquier orden?.
- ¿Cuáles son las reglas que gobiernan la secuenciación de mensajes?
- ¿Existe alguna relación entre los mensajes entrantes y salientes?
- ¿Se puede hacer una vista o gráfico del proceso completo de mensajes intervinientes en una composición de servicios?
- ¿Existe un comienzo y fin en la secuencia de mensajes? ¿Se puede dejar una secuencia de mensajes trunca o sin acabar?

Todas estas preguntas intentan ser resueltas con la especificación WSCI. Esta especificación describe cómo las operaciones definidas en la descripción del servicio WSDL (*Web Service Description Language*, Lenguaje de Descripción de Servicios Web) pueden ser coordinadas en un contexto de intercambio de mensajes en donde estos servicios participan.

WSCI describe fundamentalmente cómo estas operaciones deben exponer la información relevante, es decir, dar respuesta a las preguntas antes planteadas como: la correlación que debe existir entre los mensajes, el manejo de excepciones, la descripción de las transacciones y las capacidades de

participación dinámica. WSCI también describe la interdependencia entre las operaciones de los Servicios Web, por lo que cada servicio que quiere participar de la coreografía puede:

- Entender cómo interactuar con un servicio en el contexto de un determinado proceso.
- Comprender anticipadamente el comportamiento de un determinado servicio en cualquier punto del ciclo de vida del proceso.

#### 2.5.3.2. Arquitectura WSCI

Esta especificación consiste de una capa por encima de las distintas capas que componen los Servicios Web. Cada acción definida a través de WSCI representa una unidad de trabajo, la cual es mapeada a una operación especificada en WSDL. Como puede observarse en la Figura 2.6, la coreografía escrita en WSCI incluye un conjunto de interfaces, una por cada participante de la coreografía, definiendo el comportamiento de los mensajes que se deben intercambiar entre ellos.

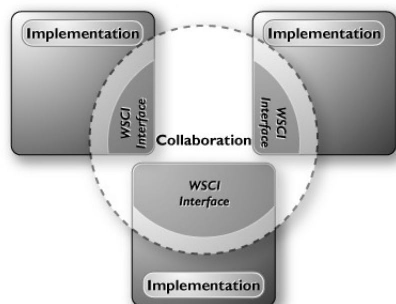


Figura 2.6: Arquitectura WSCI. Fuente: https://www.w3.org/TR/wsci

#### 2.5.3.3. Uso académico e industrial

Esta especificación, si bien fue inicialmente promovida por empresas como Sun, IBM, Intalio, etc., no ha tenido implementaciones reales para su utilización, ya que las coreografías fueron relegadas por las orquestaciones y por ende los lenguajes que las soportan, tales como WS-CDL.

En el ámbito académico no se han realizado avances en los últimos 10 años sobre esta especificación, por lo que su utilización ha sido casi nula.

Esta especificación es un complemento a la especificación WS-CDL analizada previamente, por lo que al no existir implementaciones industriales de aquella, tampoco se pueden encontrar de esta especificación.

### 2.5.4. Web Services Business Process Execution Language - WS-BPEL

#### 2.5.4.1. Qué es WS-BPEL

Es un lenguaje estándar para la integración y automatización de procesos dentro de lo que es la denominada orquestación de servicios web. El objetivo principal de este lenguaje es la de estandarizar la orquestación de servicios. Este lenguaje permite describir las composiciones de dos maneras diferentes: se puede especificar los detalles exactos de los procesos de negocio o bien especificar solamente los intercambios de mensajes públicos entre las partes involucradas (Pant y Juric, 2008).

Un proceso BPEL especifica el orden exacto en que los servicios web participantes deben ser invocados, ya sea de manera secuencial o en paralelo, además de permitir expresar condiciones de comportamiento al momento de ser invocados.

#### 2.5.4.2. Arquitectura WS-BPEL

La arquitectura que subyace al lenguaje consiste en un elemento central que es el encargado de llevar adelante la orquestación de los servicios que en conjunto resolverán el proceso de negocio que se intenta modelar y ejecutar en BPEL. Este elemento central es, en definitiva, el motor de ejecución de BPEL.

La figura 2.7 muestra un proceso de negocio modelado a través de WS-BPEL. Básicamente la arquitectura invoca la ejecución de este proceso el cual se conecta con los servicios correspondientes para llevar adelante su tarea. WS-BPEL es el encargado de orquestar la ejecución del proceso en su totalidad y de los servicios involucrados.

#### 2.5.4.3. Uso académico e industrial

Este lenguaje, y en especial la orquestación de servicios, es lo que se ha hecho más popular desde el comienzo de la utilización de composiciones con servicios web, siendo la especificación que tiene el mayor número de implementaciones. El éxito se debe en gran medida a su integración con la especificación BPMN, donde se definen los procesos de negocio que luego se ejecutan a través de BPEL.

Para la academia es un área de investigación muy importante a día de hoy. Se pueden ver trabajos actualizados sobre la materia, además de estar intrínsecamente relacionado con la notación de procesos de negocios (BPMN), donde se realizan avances constantes. La composición de servicios web sigue siendo un área de interés ya que, como hemos visto, es una tarea compleja y que requiere de niveles de confiabilidad y disponibilidad de los servicios que necesita ser estudiada en todos sus aspectos.

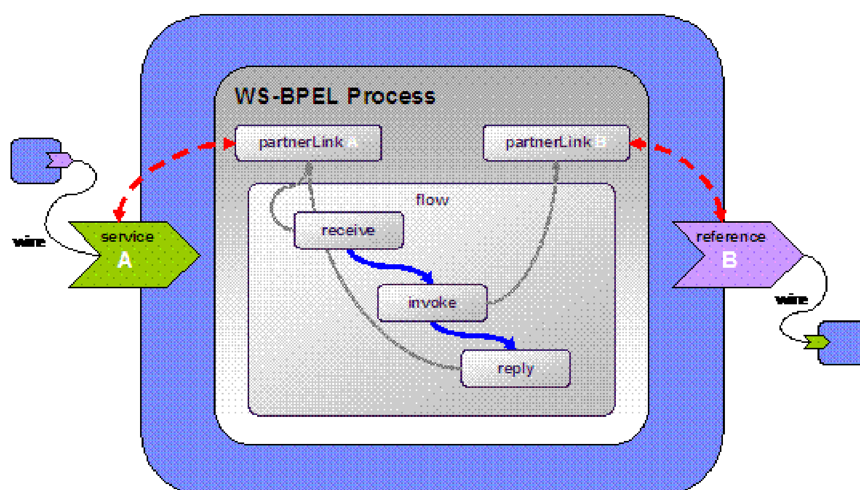


Figura 2.7: Arquitectura WS-BPEL. Fuente: `http://docs.oasis-open.org/opencsa/sca-bpel`

Su vinculación con los procesos de negocio, los cuales son ejecutados a través de WS-BPEL, ha permitido que su utilización se extienda en el ámbito comercial e industrial, ya que permite no solo modelar y ejecutar los procesos de negocio especificados a través de BPMN, sino que además beneficia la reconfiguración de los mismos. Son muchos los sectores industriales que han sacado provecho de esta tecnología, como la automotriz, la de e-commerce, etc.

Si bien WS-BPEL es un lenguaje de amplia utilización, necesita para su ejecución de un nodo central que haga las veces de coordinador del resto de los servicios o dispositivos que se quieran relacionar. Esto implica que se necesite para su ejecución un equipo con capacidades de procesamiento elevadas, por lo cual los costos para su implementación son elevados.

### 2.5.5. JADE

#### 2.5.5.1. ¿Qué es JADE?

JADE es un framework de desarrollo de sistemas orientados a agentes totalmente desarrollado en el lenguaje de programación Java (Bellifemine et al., 2007). Su objetivo es simplificar la implementación de sistemas multi-agentes a través de la utilización de un middleware que es compatible con la especificación FIPA y que, a través de la utilización de una serie de herramientas gráficas, da soporte a las fases de debugging y deployment. Un sistema basado en JADE puede estar distribuido a lo largo de varias computadoras cuya configuración puede ser controlada vía una interface gráfica remota. Esta configuración puede ser cambiada incluso en ejecución, cambiando agentes desde una computadora a otra en la forma que se necesite

(Bellifemine et al., 2007).

#### 2.5.5.2. Arquitectura JADE

La Figura 2.8 muestra los elementos principales que conforman la arquitectura de JADE. Ésta se compone por contenedores de agentes distribuidos a lo largo de la red. Los contenedores que son los procesos Java que provee la plataforma JADE junto a todos los servicios necesarios para hospedar y ejecutar agentes. Existe un contenedor especial denominado Main-container, que representa el punto concentrador de la plataforma. Este contenedor es el primero en ser levantado por la plataforma y donde el resto de los contenedores deben registrarse.

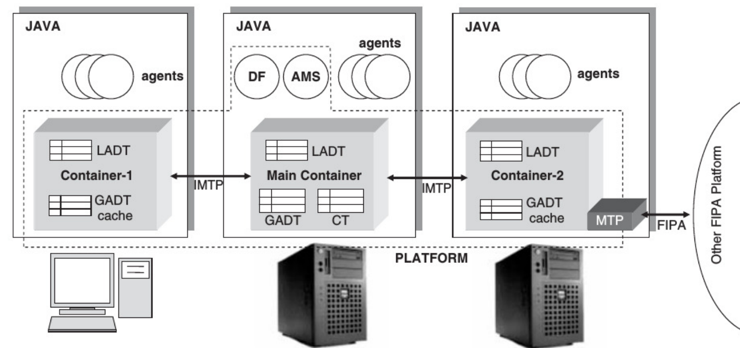


Figura 2.8: Arquitectura JADE. Fuente: (Bellifemine et al., 2007)

#### 2.5.5.3. Uso académico e industrial

La tecnología de agentes es considerada uno de los más importantes paradigmas de computación, siendo uno de sus puntos más importantes la integración de los sistemas. El agente autónomo actúa sin intervención humana y tiene control sobre sus acciones y su estado interno. A su vez, el agente es social porque coopera con otros agentes para llevar adelante su tarea. Al ser una tecnología que incorpora inteligencia artificial, ha sido utilizada ampliamente para múltiples aplicaciones, tales como el prototipo de simulación de tráfico aéreo OASIS (Optimal Aircraft Sequencing using Intelligent Scheduling); o en el área de telecomunicaciones; sistemas de robótica; etc. (Bellifemine et al., 2007).

JADE es a día de hoy una tecnología que sigue siendo desarrollada. Siguen haciendo estudios para su implementación en distintos ámbitos computacionales, tales como la robótica.

Si bien JADE es un framework para el desarrollo de aplicaciones multiagente que facilita el desarrollo de aplicaciones distribuidas, este se basa sobre

el concepto de un contenedor central que es quien organiza las conexiones entre el resto de los agentes. Además, otro de los inconvenientes importantes a la hora de la utilización es la necesidad de contar con una máquina virtual Java para la ejecución del mismo, lo que limita la cantidad de equipos que pueden ser parte de la ejecución.

## 2.6. Carencias del estado de la cuestión

Los trabajos antes presentados muestran un panorama del estado de la cuestión relacionado con temas como dispositivos ubicuos, arquitecturas existentes para la composición de servicios o dispositivos en distintos ambientes, descubrimiento de nuevos servicios, tendencias actuales hacia dónde se está enfocando la tecnología de servicios en general, y la de composición y descubrimiento de los mismos en particular.

La tabla 2.1 muestra por un lado un resumen del estudio del Estado de la Cuestión y por otro las falencias en la coordinación de dispositivos ubicuos, los cuales no han sido abordados ni en procesos de investigación académica ni desde la industria. En la misma se han enumerado las distintas carencias que se han detectado, las cuales fueron clasificadas de acuerdo a su importancia en relación a este trabajo de investigación. Como se observa, tanto los frameworks como los trabajos de autores de la academia presentan algunas de las carencias aquí enumeradas.

| Problema \ Autor o Framework | Najar | Loke | Palmieri | Viroli | Mosquito | Android Studio | KNX | OSGi | OpenHab | Z-wave | Aura | Amigo | Gaia | Oxigen |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Falta de un protocolo estándar para comunicación | X | X | | | X | X | | | | | | | | |
| No contempla la totalidad de los dispositivos ubicuos (orientados a celulares, domótica, etc.) | | | X | X | | X | X | | | | X | X | X | X |
| Necesidad de contar con una máquina virtual Java o con algún software instalado para su ejecución | | | | | | X | | X | X | | | | | |
| Necesidad de contar con un middleware | | | | | X | | | | X | | X | X | X | X |
| Necesidad de contar con cableado para la comunicación | | | | | | | X | | | | | | | |
| Necesidad de que los dispositivos se adapten a la norma | | | | | | | X | | X | X | | | | |

Tabla 2.1: Carencias del Estado de la Cuestión

# Capítulo 3

# Planteamiento del problema y Objetivos de Investigación

En este capítulo haremos un planteamiento del problema, identificándolo en primer lugar. Luego indicaremos la importancia del mismo y, por último, haremos la presentación de los objetivos de investigación que nos hemos planteado.

## 3.1. Identificación del problema

Como hemos presentado en el Estado de la Cuestión, existen diversos frameworks de sistemas ubicuos (**Aura**(Harkes et al., 2002), **Gaia**(Campbell et al., 2002), **Oxigen**(Science y Laboratory, 2002), **Amigo**(Ami, 2019), OSGi, Android Studio, etc.) los cuales abordan la conectividad de los dispositivos ubicuos, en algunos casos incluso de manera sensible al contexto. No obstante, dichos frameworks poseen alguna de estas carencias: 1) no permiten que los dispositivos pueden conversar entre ellos de una manera abierta y estándar, o bien 2) los dispositivos no poseen la capacidad de comunicarse con otros dispositivos de la misma índole, ya que la comunicación debe ser canalizada por una conexión a un dispositivo con mayor capacidad de trabajo (como son los teléfonos móviles, por citar un ejemplo).

Se puede afirmar que, hoy en día, distintos sensores o dispositivos se pueden comunicar entre ellos, compartiendo de alguna manera sus servicios. **Sin embargo, la composición de dispositivos se realiza a partir de protocolos propietarios y sin seguir definiciones estándares, provocando que dispositivos de otros proveedores (o incluso de los mismos) no puedan ser utilizados**. Esto obviamente representa una importante limitación en la composición de dispositivos ubicuos.

Existen algunas investigaciones que abordan aspectos particulares de la composición de dispositivos. Trabajos como los de (Najar et al., 2014), (Loke, 2012) o (Palmieri, 2013) indagan sobre la manera de poder seleccionar,

descubrir y/o validar que los dispositivos sean adecuados para la composición que se pretende realizar. Sin embargo, ninguno de estos trabajos se encuentra mínimamente operativo en entornos reales. Probablemente, el trabajo más avanzado es el de (Palmieri, 2013), donde se abandona la manera de trabajo centralizada (orquestaciones) por métodos más colaborativos y distribuidos, como es el caso de las coreografías. Aún así se basa únicamente en el descubrimiento y búsqueda de nuevos servicios y dispositivos en ambientes pervasivos, sin ser parte del trabajo un mecanismo o framework de comunicación posterior al descubrimiento de los mismos. En el campo de los frameworks académicos Aura(Harkes et al., 2002), Gaia(Campbell et al., 2002), Oxigen(Science y Laboratory, 2002) o Amigo(Ami, 2019) se necesita la instalación o implementación de un middleware para que los dispositivos se conecten y puedan interconectarse. Además en el caso de de Amigo(Ami, 2019) y Oxigen(Science y Laboratory, 2002) se centran básicamente en brindar servicios a las personas que integran un espacio determinado y no en la manera en que estos dispositivos pueden coordinarse para brindar dichos servicios.

Adicionalmente la composición de múltiples dispositivos ubicuos presenta desafíos específicos. Hoy en día nos encontramos con dispositivos de la vida cotidiana que se encuentran conectados a Internet y a partir de allí pueden interactuar con servicios web ya definidos y en funcionamiento. En particular, los mecanismos de composición en ambientes masivos **necesita hacer frente las distintas contingencias que pueden ocurrir con los dispositivos**, además de contemplar la heterogeneidad de los mismos. Los dispositivos ubicuos acostumbran a tener distintas limitantes, como son la cantidad de memoria disponible, la durabilidad de la batería, y la conectividad de acuerdo a la red del lugar donde se encuentre en un momento determinado. En ambientes ubicuos, la disponibilidad y confiabilidad de los dispositivos no puede ser garantizada. 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 (Sheng et al., 2014)[1].

## 3.2. Importancia del problema

Existen distintos proyectos en la actualidad donde se intenta integrar sensores y dispositivos ubicuos a la vida cotidiana. Específicamente podemos mencionar la domótica, donde varios dispositivos y sensores deben actuar en coordinación para prevenir, por ejemplo, un incidente de seguridad en nuestros hogares como podría ser un robo o un incendio. Sin embargo, existen

[1] Si bien los autores se refieren a la composición de servicios, se hace dentro de un contexto de dispositivos ubicuos, lo cual a los fines de este trabajo se puede interpretar como composición de dispositivos

áreas de aplicación más relevantes. En la industria, se están llevando a cabo iniciativas denominadas como Industria 4.0 (Tapia, 2017)[2], donde se intenta lograr dentro de una planta fabril la intercomunicación de todos los dispositivos que componen la cadena de producción con el fin de que coordinen entre ellos las tareas a realizar en base a los tiempos a cumplir, stocks disponibles, demanda en línea de los productos, etc.

Otra área donde los dispositivos ubicuos están ganando importancia es la automotriz, donde los esfuerzos se enfocan en que distintos sensores monitoreen funciones vitales del conductor (como es el caso de presión arterial, pulsaciones, etc), del vehículo (combustible, presión de neumáticos), de la carretera, etc., y en caso de que detecten anomalías, actúen en conjunto con otros dispositivos del vehículo para evitar accidentes. Un ejemplo de esto puede observarse en http://lidesvergara2.weebly.com/vehiculos-inteligentes.html, donde se muestran algunos de los avances en esta materia. Este es un área particularmente importante respecto de este trabajo de Tesis, ya que es un ámbito al que se puede aplicar el prototipo de framework de coordinación resultado de esta tesis.

Otra área importante de trabajo e investigación es el de Internet de las Cosas, que se refiere a la interconexión digital de objetos cotidianos con Internet, como es el caso de los smartwatch, wearables, electrodomésticos, etc. Esto significa que una gran cantidad de dispositivos ubicuos incorporarán una dirección IP, situándolos en un ambiente web. A su vez, va a ser necesario en el corto plazo que estos dispositivos ubicuos no solo cooperen entre sí, sino que además, deberán hacerlo con servicios web (o incluso microservicios) que permitirán acceder a aplicaciones tradicionales con el objetivo de poder realizar, por ejemplo, el pedido de productos a un supermercado a partir de los consumos que se produjeron en una heladera inteligente. Un ejemplo de este tipo de dispositivos podemos encontrarlo en el siguiente informe https://www.infobae.com/2013/09/25/1511462-la-heladera-inteligente-desembarco-la-argentina/.

En resumen, se puede afirmar que el problema de investigación planteado está relacionado íntimamente con diversas áreas de una gran importancia tecnológica, industrial o social. En estas áreas, como es el caso de la Industria 4.0, se están realizando fuertes inversiones económicas, a la espera de los consiguientes beneficios (Tapia, 2017). Solucionar en todo o en parte los desafíos que se plantean con los dispositivos ubicuos, haciendo foco en la composición de una manera abierta y estándar y considerando las características específicas de estos dispositivos puede suponer un gran paso adelante en las áreas de aplicación antes citadas.

[2] El término Industria 4.0 fue acuñado en Alemania en el año 2011, pero también se la conoce como Internet Industrial de las cosas.

## 3.3. Objetivos de Investigación

### 3.3.1. Objetivo Principal

El objetivo principal de esta investigación se ha planteado como **Definir un mecanismo de coordinación de dispositivos ubicuos que garantice su interoperabilidad independientemente del modelo y fabricante del mismo; utilizando los estándares de SOA y de coreografías para la composición de servicios**. Alcanzar este objetivo permitirá que la interoperabilidad y coordinación de los dispositivos ubicuos se realice de manera abierta, transparente y basada en estándares de la tecnología actual, proporcionando además integración de tecnologías como IoT y microservicios bajo un mismo mecanismo de coordinación.

A partir de este objetivo principal hemos planteado una serie de objetivos más específicos, los cuales suponen, por un lado, una descomposición más detallada del objetivo principal y, por otro, permiten definir las distintas actividades que se desarrollarán a lo largo del proceso de investigación, tal y como se describe en mayor detalle en el Capítulo 4.

### 3.3.2. Objetivos Específicos

- **Desarrollar un framework que implemente la especificación WS-CDL que pueda ser ejecutado por dispositivos ubicuos:** Este objetivo nos permitirá obtener un framework de ejecución de coreografías lo suficientemente versátil como para que el mismo pueda ser implementado en dispositivos con distintas capacidades tanto de memoria como de procesamiento. Este objetivo se encuentra directamente relacionado con el objetivo principal de investigación ya que, por un lado permitirá realizar la coordinación de los dispositivos de una manera estándar y por otro, permitir la interoperabilidad de dispositivos de distintos fabricantes.

- **Lograr la coordinación de dispositivos ubicuos independientemente de las características distintivas de los mismos, tales como escasa capacidad de memoria, de procesamiento, batería, etc.:** Aquí se deberá hacer hincapié en las distintas capacidades de los dispositivos ubicuos para que el framework pueda ser ejecutado aún en los casos que se disponga de escasa memoria o baja capacidad de procesamiento por un lado y por otro que optimice la utilización de la batería que disponen estos dispositivos. Se preve hacer extensiones del lenguaje de especificación de coreografías WS-CDL para permitir este tipo de características diferenciadas.

- **Asegurar/mantener interoperabilidad entre aplicaciones SOA ejecutadas en servidores arbitrarios y dispositivos ubicuos:** Se

deberá dotar al framework de capacidades de tiempos de latencia y de manejo de desapariciones de los dispositivos ubicuos respecto de los servidores o computadoras que ejecutan servicios de una manera mucho más eficiente y veloz que los dispositivos ubicuos. Por ejemplo, acceder a un sistema de compras on-line de un supermercado a través de servicios web desde un dispositivo ubicuo representa un desafío debido a la diferencia de capacidades de procesamiento y de latencia en las respuestas que deben ser tenidos en cuenta.

- **Conseguir que los dispositivos ubicuos soporten el manejo de distintas especificaciones basadas en SOA (transacciones, seguridad, etc.):** Es necesario que los dispositivos ubicuos participen de transacciones, ya que de lo contrario ello supondría un obstáculo para mantener la interoperabilidad con aplicaciones SOA. Para ello se deberán hacer esfuerzos en el desarrollo del framework para que permita este tipo de características en dispositivos con tan pocas capacidades, ya que, implementar características de manejo de transacciones tiene un costo de procesamiento y de consumo de memoria muy alto.

# Capítulo 4

# Metodología

En este capítulo justificaremos la utilización de "design science" como metodología de investigación. Realizaremos una descripción de dicha metodología así como de la manera en que fue aplicada en el presente trabajo de Tesis.

## 4.1. Selección del método de investigación

De acuerdo a la naturaleza del problema, y en base al objetivo planteado en el presente trabajo de investigación, el método de investigación seleccionada para llevar adelante la tesis 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 lo expresan (March y Smith, 1995). Estos artefactos están representados en una forma estructurada y van desde software, lógica formal y matemática rigurosa hasta descripciones informales en lenguaje natural.

El problema que aborda la presente tesis es la composición de múltiples dispositivos ubicuos. De hecho, diversos autores, tales como (Najar et al., 2014), (Loke, 2012), (Palmieri, 2013) o (Viroli, 2013) han propuesto soluciones parciales, tal y como se ha indicado en el capítulo de Planteamiento del Problema.

La solución pasa necesariamente por la definición de un mecanismo de coordinación y su implementación en un artefacto (esto es, un framework de coordinación) que permita la composición de dispositivos ubicuos. Por todo ello, "design science" parece el método más alineada con la presente tesis. Otros métodos podrían ser igualmente aplicables, como es el caso de Action Research. 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. Esta estrategia no considera ni (1) una evaluación explícita de los artefactos construidos ni (2) el hecho de que esta investigación no se produce en

un contexto específico, sino general. Otros métodos como Case Study están todavía menos alineadas con el tipo de investigación planteado en esta Tesis.

Finalmente, al estar una Tesis orientada a una temática totalmente aplicada, se debe seleccionar un caso de prueba en el que demostrar la viabilidad del framework desarrollado. En otras palabras: la validación de la tesis consistirá en la creación de una prueba de concepto. La prueba de concepto se alinea estrechamente con la actividad de demostración como plantea (Peffers et al., 2007), lo cual aumenta nuestra confianza en la idoneidad del método de investigación.

## 4.2. Design Science

### 4.2.1. Descripción general

Los Sistemas de Información son una disciplina de investigación donde se aplican teorías de otras ciencias como, por ejemplo, Computación, Economía y hasta las Ciencias Sociales, para resolver problemas que se relacionan con tecnologías de la información y las organizaciones. En consecuencia, los paradigmas de investigación predominantes son los heredados de dichas ciencias. En contrapartida, el Diseño, esto es, el Acto de Crear una solución aplicable a un determinado problema, es un paradigma de investigación aceptado en otras disciplinas como la Ingeniería. El Diseño como metodología sólo se ha aplicado en una pequeña cantidad de trabajos de Sistemas de Información (Peffers et al., 2007).

Los investigadores en Sistemas de Información comenzaron a precisar el concepto de “design science” en los comienzos de la década de 1990 (Peffers et al., 2007). La naturaleza de “design science” es la generación de un artefacto, el cual debe ser diseñado y evaluado durante un proceso iterativo hasta alcanzar el grado de madurez que se necesita de acuerdo a la problemática planteada en cada caso. Esta aproximación diferencia a “design science” de otros paradigmas de investigación, como puede ser la teoría de construir y testear. También se diferencia de las ciencias naturales y las sociales, ya que mientras éstas intentan comprender la realidad, “design science” intenta crear cosas que sirven para propósitos humanos (Peffers et al., 2007).

(Hevner et al., 2004) plantea 7 directrices para realizar trabajos de investigación basados en la metodología de “design science”.

1. **Diseño de artefactos:** Toda investigación de “design science” debe producir un artefacto real en la forma de una construcción, un modelo, un método o una instancia, en el dominio del problema.

2. **Importancia del problema:** El objetivo de la investigación debe ser el desarrollo de soluciones basadas en tecnología para un problema importante y de relevancia.

3. **Evaluación del diseño:** La utilidad, calidad y eficacia de un artefacto diseñado debe ser rigurosamente demostrado a través de métodos de evaluación bien ejecutados.

4. **Contribución a la investigación:** La investigación utilizando "design science" debe proveer contribuciones claras y verificables en el área del diseño del artefacto, los fundamentos del diseño y/o en las metodologías de diseño.

5. **Rigor de la investigación:** "Design science" se basa en la aplicación de métodos rigurosos tanto en la construcción como en la evaluación del artefacto diseñado o construido.

6. **El diseño como un proceso:** La búsqueda para la construcción de un artefacto efectivo requiere la utilización de medios disponibles para alcanzar los fines deseados al mismo tiempo que se satisfacen las leyes que rigen la naturaleza del problema.

7. **Comunicación de los resultados:** Las investigaciones bajo esta metodología deben ser presentadas de una manera efectiva tanto a audiencias de base tecnológica como de gestión.

La esencia de estas directrices es que la investigación, debe producir un artefacto creado para resolver un problema. Además, este debe ser relevante para la solución de un problema aún no resuelto hasta el momento. Su utilidad, calidad y eficacia debe ser evaluada rigurosamente (Hevner et al., 2004). A su vez la investigación debe representar una contribución al conocimiento existente sobre el problema en cuestión. El rigor que se debe utilizar para su evaluación debe aplicarse también en la etapa de desarrollo del artefacto.

### 4.2.2. Propuestas metodológicas

Desde sus comienzos se ha evidenciado la necesidad de establecer una metodología para la aplicación de "design science". Una metodología es un sistema de principios, prácticas y procedimientos aplicados a una rama de conocimiento específica. A continuación exponemos algunas propuestas metodológicas destacadas.

#### 4.2.2.1. March et al.

"Design science" es inherentemente un método de solución de problemas, al igual que el objetivo que se plantea en este trabajo. El principio fundamental de una investigación dirigida por "design science" es que el conocimiento y comprensión de un problema, el diseño y su solución asociada se adquieren a través de la construcción y aplicación de un artefacto.

La metodología propuesta por (March y Smith, 1995) está dirigida por la distinción entre las actividades de investigación y los resultados de investigación. Dentro de los resultados de investigación identifica cuatro artefactos que son producidos por "design science". Los artefactos son:

- Construcciones: Las construcciones forman el vocabulario de un dominio. Constituyen una conceptualización utilizada para describir problemas dentro del dominio y para especificar las soluciones a los mismos. Las construcciones o artefactos son construidos para dar solución a problemas no resueltos hasta la actualidad.

- Modelos: Los modelos usan las construcciones para representar una situación del mundo real. Los modelos ayudan al entendimiento del problema y la solución, y frecuentemente representan la conexión entre los componentes del problema y la solución permitiendo la exploración de los efectos de las decisiones y cambios en el mundo real.

- Métodos: Los métodos definen procesos y proveen una guía en cómo solucionar los problemas, es decir, en cómo buscar en el espacio de solución. Estos modelos pueden variar desde algoritmos hasta descripciones textuales e informales de cómo se encaró la solución, o alguna combinación de ambos.

- Instanciaciones: Las instanciaciones son la realización de un artefacto en su ambiente. Las instanciaciones demuestran la viabilidad y efectividad de los modelos y métodos definidos.

Las cuatro actividades de investigación son:

- Construir: se refiere a la construcción del artefacto, demostrando que el mismo puede ser construido. Un artefacto se construye para realizar una tarea específica y para demostrar su factibilidad, volviéndose luego en un objeto de estudio. Estas construcciones deben ser evaluadas científicamente.

- Evaluar: se refiere al desarrollo de criterios de evaluación del desempeño del artefacto construido. Se realiza la evaluación para determinar si se ha logrado un progreso. Aquí es donde se debe responder a la pregunta ¿Qué tan bien está construido el artefacto?. Para poder evaluar se necesita el desarrollo de métricas y de la medición del artefacto respecto de estas métricas.

- Teorizar: se refiere a la realización de teorías acerca de los artefactos construidos. Las teorías explican las características del artefacto y su interacción con el entorno, que dan como resultado el rendimiento observado.

- Justificar: se refiere a la justificación de las teorías obtenidas o realizadas, es decir, se debe reunir evidencia para probar la teoría.

#### 4.2.2.2. Hevner

En los procesos de investigación basados en "design science" existen tres ciclos de relevancia, propuestos por (Hevner et al., 2004), los cuales se pueden apreciar en la Figura 4.1. Estos ciclos se basan en cómo se entiende, ejecuta y evalúa una investigación en Sistemas de Información combinado los paradigmas de Ciencias del Comportamiento y de "design science". Los ciclos son los siguientes:

- **Ciclo de relevancia.** Se encarga de unir el entorno contextual del proyecto de investigación con las actividades de "design science". Se logra conectar los requerimientos pertenecientes al dominio del problema con el ciclo de diseño, donde se construye y evalúa el artefacto construido.
- **Ciclo de rigor.** Conecta las actividades de "design science" con la base de conocimiento científico, experiencia y resultados que brinda el proyecto de investigación. Permite de esta manera cerrar el ciclo de investigación, donde los resultados pueden ser expuestos a la comunidad científica en general.
- **Ciclo de diseño.** Es el ciclo central es el que itera entre las actividades de construcción y evaluación del artefacto construido. En este ciclo se logra transformar los requisitos existentes en un producto o artefacto finalizado, el cual a través de distintos ciclos va evolucionando desde el prototipo inicial hasta el producto final. A partir de este momento es cuando se pueden hacer los aportes científicos requeridos por el ciclo de rigor.

#### 4.2.2.3. Peffers

(Peffers et al., 2007) asume una dimensión más operativa. Según este autor, la metodología "design science" debe tener un proceso de aplicación bien definido. Para ello, propone un modelo de seis actividades secuenciadas. La Figura 4.2 muestra de manera gráfica estas actividades planteadas por el autor, así como también las relaciones que se producen entre las mismas. Dichas actividades se describen a continuación:

1. **Identificación del problema y motivación:** Se debe definir el problema específico de investigación y se tiene que justificar también el valor de una solución. Los recursos necesarios para realizar esta actividad incluyen el conocimiento del estado del problema y la importancia de su solución.

2. **Definir los objetivos para una solución:** Consiste en inferir los objetivos de una solución a partir de la definición del problema y el conocimiento de lo que es posible y factible. Los recursos necesarios para esta actividad incluyen el conocimiento del estado de los problemas y las soluciones actuales, si las hay, y su eficacia.

3. **Diseño y Desarrollo:** Crear el artefacto propiamente dicho. Los recursos que se requieren para pasar de los objetivos al diseño y desarrollo incluyen el conocimiento de la teoría que se puede aplicar en una solución.

4. **Demostración:** Se debe demostrar el uso del artefacto para resolver uno o más casos del problema planteado. Los recursos requeridos para la demostración incluyen un conocimiento efectivo de cómo usar el artefacto para resolver cosas específicas del problema planteado.

5. **Evaluación:** Se debe observar y medir qué tan bien proporciona el artefacto diseñado y construido una solución al problema planteado. Esta actividad implica comparar los objetivos con los resultados reales observados del uso del artefacto en la demostración. Al final de esta actividad, los investigadores pueden decidir si repetir el paso tres para tratar de mejorar la efectividad del artefacto o continuar con la actividad de comunicación y postergar mejoras adicionales para proyectos posteriores.

6. **Comunicación:** Se debe comunicar el problema y su importancia, el artefacto, su utilidad y novedad, el rigor de su diseño y su efectividad a los investigadores y otras audiencias relevantes, como los profesionales en ejercicio, cuando sea apropiado.

La figura 4.1 representa esquemáticamente la relación entre los 3 ciclos de “design science” según (Hevner, 2007) y la presente investigación.

### 4.3. Aplicación en esta investigación

En las secciones anteriores hemos descrito las características generales del método “design science”, junto a las metodologías planteadas por distintos autores, en orden creciente de complejidad y detalle, para llevar adelante una investigación de estas características. Plantearemos a continuación la secuencia concreta de pasos y actividades que, de acuerdo a los lineamientos de “design science”, para alcanzar los objetivos de investigación.

En primer lugar, como parte del ciclo de relevancia:

- Se comenzó con la identificación del problema, el estudio exhaustivo del estado de la cuestión y la definición de la importancia de la solución, tal y como se presentó en los capítulos 2 y 3.

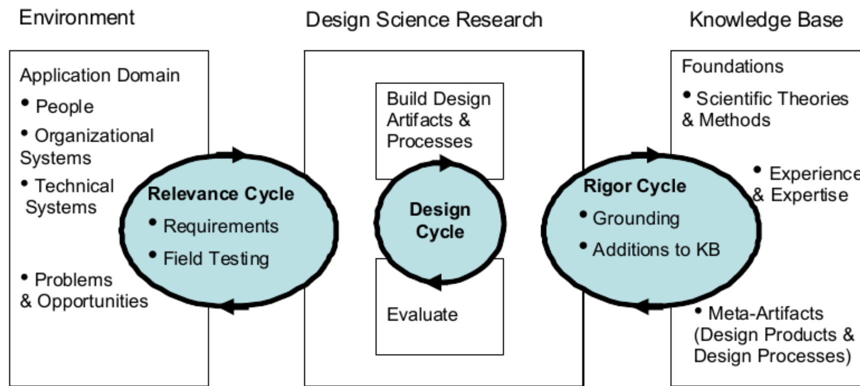


Figura 4.1: Ciclos en el proceso de Investigación de “design science” según Hevner. Fuente: (Hevner, 2007)

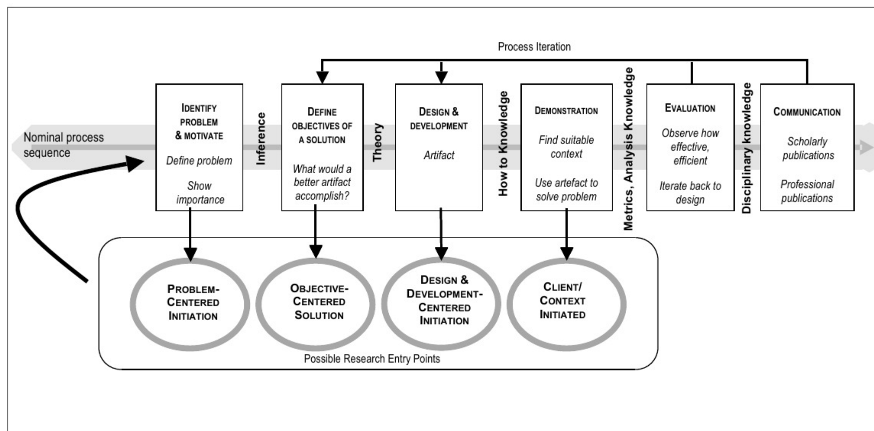


Figura 4.2: Actividades en el proceso de “design science” según Peffers. Fuente: (Peffers et al., 2007)

Para continuar con la investigación y los siguientes pasos, se trabajó sobre el ciclo de diseño, en donde:

- Como segunda actividad, se definieron los objetivos de investigación: **Definir un mecanismo de coordinación de dispositivos ubicuos que garantice su interoperabilidad independientemente del modelo y fabricante del mismo; utilizando los estándares de SOA y de coreografías para la composición de servicios**. Para definir el objetivo nos basamos fundamentalmente en el estado de la cuestión, y también de la falta de investigación en la materia como bien se ha expresado en el Capítulo 2.

- 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. Gran parte del trabajo estuvo dedicado a la traslación de las especificaciones existentes en SOA para el caso particular de los dispositivos ubicuos, como así también en la producción de un framework de ejecución.

- Dentro de la actividad de demostración, se aplicó el framework de ejecución de coreografías en ambientes ubicuos a un caso de estudio o escenario de trabajo. Este escenario consiste en simular una carretera inteligente, donde los vehículos que transitan junto a balizas y distintos dispositivos a lo largo de la carretera se comunican con el fin de prevenir accidentes, mejorar las condiciones de manejo, etc.

- Finalmente, se realizaron distintas evaluaciones al final de cada ciclo de trabajo, con el fin de determinar si el framework ha logrado trasladar adecuadamente las especificaciones SOA existentes al contexto de los dispositivos ubicuos. Se comenzó inicialmente con un prototipo básico para luego ir evolucionando en cada ciclo hasta llegar al artefacto final. La cantidad de iteraciones o ciclos de desarrollo se fijó en 6, ya que es la cantidad necesaria para poder implementar las características de los dispositivos ubicuos en ambientes SOA, tal y como se describe en el capítulo de la Aproximación a la Solución 5.

Como parte del ciclo de rigor, hemos planteado las siguientes actividades:

- Por último, la actividad de comunicación fue llevada a cabo a través de la presentación de los resultados en distintos congresos, journals y workshops. La lista de publicaciones está disponible en la Introducción de este informe de Tesis. A su vez se encuentra en confección un trabajo donde se presente el resultado final de la investigación, el cual nos planteamos enviarlo a IEEE Computer ó el Journal of Systems and Software. A través de las mismas se expuso ante la comunidad científica

de los resultados alcanzados, así como de las contribuciones realizadas a las especificaciones existentes en SOA para soportar las distintas características de los dispositivos ubicuos.

Desde una perspectiva general, esta investigación se relaciona principalmente con el ciclo de diseño según (Hevner, 2007). Nuestro objetivo general es desarrollar un mecanismo de coordinación de dispositivos ubicuos. No obstante, no buscamos *cualquier* mecanismo de coordinación; en el Estado de la Cuestión ya hemos expuesto diversas alternativas. Por el contrario, perseguimos que el mecanismo de coordinación posea ciertas características como sencillez, eficacia y facilidad de estandarización.

Al finalizar la investigación, el conocimiento científico obtenido aportará mejoras a nuestra comprensión acerca de la coordinación de dispositivos, lo cual está relacionado con el ciclo de rigor de "design science".

Tal y como muestra la figura 4.1 el ciclo de relevancia ha sido expuesto en la importancia del problema.

Para la implementación del ciclo de diseño nos hemos basado fundamentalmente en el proceso presentado por (Peffers et al., 2007).

# Capítulo 5

# Aproximación a la solución

La computación orientada a servicios, y en particular los servicios web en ambiente de Internet, proporcionan mecanismos para la composición de servicios. Dichos mecanismos, como las orquestaciones y las coreografías, son aspectos bien conocidos de la computación orientada a servicios que permiten construir aplicaciones y sistemas de negocio complejos a partir de una gran cantidad de servicios heterogéneos, simples y distribuidos. Podría pensarse que son aplicables a ambientes pervasivos.

**Nuestra propuesta es utilizar las estandarizaciones existentes en SOA para la coordinación de dispositivos ubicuos en ambientes pervasivos.** Si pensamos cada dispositivo ubicuo como un proveedor o consumidor de un servicio, la arquitectura de una aplicación basada en dispositivos ubicuos encaja perfectamente en la arquitectura de servicios.

En cualquier caso, aunque existan ciertos paralelismos, SOA y los ambientes pervasivos poseen marcadas diferencias. En contextos pervasivos, donde los servicios son dinámicos, móviles, menos fiables y dependientes del dispositivo, los mecanismos de composición establecidos para servicios web no son directamente aplicables (Cassar et al., 2013). A modo de ejemplo, pensemos en un sistema de siembra inteligente, donde de acuerdo a distintos sensores que toman la temperatura y humedad de la tierra en distintos puntos de un campo deben determinar por un lado, si es necesario activar el sistema de riego automático para brindarle mayor humedad al terreno y por otro si hay que disparar alarmas en alguna central para indicar que el sistema de siembra debe comenzar. Cualquier falla o problema que pueda existir en alguno de los componentes puede hacer que falle el sistema en general.

Las aplicaciones asumen implícitamente la existencia de equipos con ciertas características, tanto de procesamiento (en muchos casos, los equipos son mini-ordenadores o sistemas de altas prestaciones), memoria (en el orden de decenas o cientos de GB), almacenamiento secundario (prácticamente ilimitado hoy en día) y conectividad (protocolos de TCP/IP sobre redes de alta velocidad). Los dispositivos ubicuos representan todo lo contrario: bajas

capacidades de procesamiento, memoria, almacenamiento y múltiples mecanismos de conectividad. Lograr una convergencia entre ambos mundos no es, por lo tanto, ni sencilla ni inmediata.

## 5.1. Trabajos que proponen aproximaciones similares

No existe ningún trabajo que haya planteado trasladar los conceptos de SOA al dominio de los dispositivos ubicuos. No obstante algunos autores han sugerido distintos enfoques y escenarios similares para resolver la problemática de la coordinación de dispositivos.

(Dragoni et al., 2017b) define a los microservicios como un proceso independiente, cohesivo interactuando con otros a través de mensajes. Desde un punto de vista técnico, los microservicios deben ser componentes independientes e implementados conceptualmente de forma aislada.

El mismo autor define la arquitectura de microservicios como un nuevo paradigma de la programación basado en la composición de pequeños servicios, cada uno ejecutando sus propios procesos y comunicándose a través de mecanismos ligeros (mensajes). Este enfoque se ha basado en los conceptos de SOA, hasta el punto de que Netflix usó una arquitectura muy similar bajo el nombre de: Fine grained SOA (Wang y Tonse, 2013).

Sin entrar en mucho más detalle sobre los microservicios y la arquitectura asociada, (Dragoni et al., 2017b), señala que la forma de componer microservicios se basa principalmente en las orquestaciones y las coreografías. Siendo más adecuadas las coreografías, ya que en estas todos los componentes tienen un mismo nivel de relevancia e importancia en la ejecución de la misma. Estos conceptos se basan sobre los estándares de WS-BPEL y WS-CDL respectivamente, los cuales derivan de SOA.

Existen trabajos e investigaciones realizadas 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 (Cherrier et al., 2012), (Duhart et al., 2015), (Mostarda et al., 2010), (Zhou et al., 2018), 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.

La literatura respecto de la arquitectura orientada a servicios así como la de computación orientada a servicios se encuentra muy estudiada al momento y, además, existen muchos desarrollos basados en los estándares de OASIS. Es decir se evidencia una madurez importante sobre esta tecnología, por lo que su traspolación a otros ambientes, como el caso de dispositivos ubicuos, resulta casi natural.

Mirko Viroli(Viroli, 2013), nos plantea el típico escenario de colaboración entre distintos actores para encontrar alojamiento entre los horarios de un vuelo, o de otros servicios disponibles en el aeropuerto (es decir, no solamente servicios de hotel). Esta coordinación o coreografía de servicios se

produce entre dispositivos ubicuos, donde la conectividad entre ellos no se basa en redes del tipo TCP/IP. Ahora bien, los principios de coordinación entre los dispositivos están por encima de los protocolos de comunicación que puedan existir entre los mismos. Sin muchas complicaciones utilizamos una declaración de interfaces con WSDL y una orquestación de los servicios a través de una implementación de coreografías como WS-CDL o bien con WSCI. Sólo realizamos algunos ajustes claro, en los end-points de las declaraciones WSDL además de otros ajustes a la tecnología existente. Además estos dispositivos se encargan de coordinar una transacción (a través de las especificaciones WS-Atomic Transaction o Web Services Coordination) que permite la realización de una reserva en el hotel y una cena en un determinado restaurant de las cercanías.

Es importante destacar con este simple escenario que la coreografía que pueden llevar adelante estos servicios implementados en los distintos dispositivos móviles es fácilmente (desde un punto de vista de modelado) adaptable a la tecnología de servicios SOA, proporcionando diversas ventajas respecto a los frameworks o sistemas activos existentes hoy día de sistemas ubicuos, tal y como se describe en la Sección 5.2. No obstante, la traslación directa de los conceptos de SOA a los dispositivos ubicuos no es posible, ya que la composición de servicios tiene muchas aristas que la componen y que hacen que sea compleja no solo la composición sino la ejecución de la misma. Además debe sumarse a ello los problemas o inconvenientes que pueden producir los dispositivos ubicuos (desapariciones, disponibilidad, etc.). Resulta importante conocer y establecer los mecanismos para la realización de coreografías en este tipo de ambientes, donde la disponibilidad y fiabilidad de los participantes es altamente volátil.

## 5.2. Ventajas de la convergencia con SOA

Los ejemplos descritos anteriormente muestran cómo los conceptos de SOA podrían ser aplicables a dispositivos ubicuos en ambientes pervasivos. Esta convergencia de los ambientes pervasivos con SOA, nos permite utilizar los beneficios de las especificaciones existentes, por lo que de esta manera se puede desarrollar coordinaciones de dispositivos ubicuos que tengan acceso a sistemas tradicionales ya existentes en las organizaciones. En este caso se puede hacer uso de transacciones utilizando dispositivos ubicuos coordinados a través de una coreografía. Es decir, no solamente se puede integrar la coordinación de dispositivos ubicuos en ambientes pervasivos a sistemas que actualmente utilizan SOA, sino que además nos permitirá interoperar con otras tecnologías que en el futuro adopten SOA como puede ser el caso de IoT o microservicios.

Adicionalmente, la convergencia con SOA proporciona numerosos beneficios adicionales, debido a la inclusión de los dispositivos ubicuos en la pila de

protocolos de SOA, tal y como puede apreciarse en la Figura 5.1. Esto daría la posibilidad de que los dispositivos ubicuos dejen de ser meros recolectores de datos del contexto donde se encuentran para ser parte ser actores que forman parte de la organización.

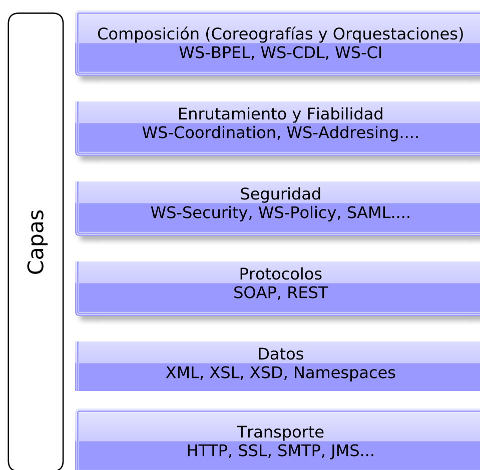


Figura 5.1: Capas que componen SOA. Fuente: (OASIS, 2009)

En relación a esta pila de protocolos destacamos las WS-Coordination (OASIS, 2009) y WS-AtomicTransaction (OASIS, 2007), las cuales definen cómo deben tratarse la confiabilidad y la tolerancia a fallos en ambientes distribuidos para el manejo de transacciones.

WS-Coordination define un framework extensible que permite a diferentes protocolos de coordinación ser insertados en un trabajo coordinado entre clientes y servicios. Este framework, a su vez, permite a los participantes obtener un acuerdo consistente en la salida de actividades distribuidas. Los protocolos de coordinación que pueden ser definidos para este framework pueden abarcar una gran variedad de actividades, incluyendo protocolos simples para una operación corta y protocolos para actividades de negocio de gran tamaño y que conllevan mucho tiempo en ejecución.

En cualquier lugar donde se utilice WS-Coordination, los mismos requerimientos genéricos se presentan:

- Instanciación (o activación) de un nuevo coordinador para un protocolo de coordinación específico, para una instancia de una aplicación particular.

- Registración de los participantes con el coordinador.

- Propagación del contexto.

Los componentes principales involucrados en la utilización y definición del WS-Coordination son los siguientes:

- Servicio de activación: En nombre de un protocolo de coordinación específico, este servicio crea un nuevo coordinador y el contexto asociado. Al mismo tiempo y de forma implícita crea un servicio de registración.
- Servicio de registración: Nuevamente actuando en nombre de un protocolo de coordinación específico, una instancia de este servicio es utilizada por los distintos participantes para inscribirse con el coordinador.
- El contexto: este contiene información necesaria para que el WS-Coordination realice la coordinación, así como también información específica del protocolo implementado.

La especificación WS-AtomicTransaction provee la definición de un tipo de coordinación de Transacción Atómica utilizada para coordinar actividades del tipo "todo o nada". Transacciones atómicas normalmente requieren de un alto nivel de confianza entre los participantes y son de corta duración. Esta especificación define protocolos que permiten la existencia de sistemas que procesen transacciones por sobre aplicaciones particulares o propietarias e interoperen entre los distintos proveedores de software y hardware. Dentro del ámbito del WS-AtomicTransaction, los servicios inscriben recursos transaccionales, como pueden ser las bases de datos o las colas de mensajes, como participantes. Cuando la transacción termina, la decisión del WS-AtomicTransaction es propagada a cada uno de los recursos que figuran anotados o enlistados y las acciones de commit o rollbak son realizadas por cada uno de estos recursos.

## 5.3. Estrategia de solución

La metodología seleccionada para la realización de esta tesis es "Design Science". Además de los méritos indicados en el Capítulo 4, dicha metodología permite que la convergencia entre SOA y los dispositivos ubicuos se realice de forma paulatina, mediante la construcción de una secuencia de prototipos que incluyen de forma incremental características de SOA o de dispositivos ubicuos, según converge en cada estadío.

### 5.3.1. Características SOA

El objetivo principal de SOA es habilitar la interoperabilidad de propósito general a través de tecnologías existentes y la extensibilidad hacia futuros propósitos y arquitecturas. Por ello los principios de diseño de SOA son independientes de cualquier tecnología específica, como los servicios web o los J2EE Enterprise Java Beans. En particular SOA prescribe que todas

las funciones de una aplicación basada en esta arquitectura son proveedoras de servicios. Es decir, los servicios SOA incluyen todas las funciones de negocio y los procesos de negocio relacionados que conforman la aplicación, así como toda función relacionada con el sistema la cual es necesaria para llevar adelante la aplicación basada en SOA.

Adicionalmente, para proveer la descomposición de la funcionalidad de la aplicación en servicios, SOA requiere que los mismos cumplan con determinadas características como que los mismos deben ser autónomos, independientes de la plataforma y su descubrimiento/invocación y coordinación debe realizarse de manera dinámica Georgakopoulos y Papazoglou (2008).

A su vez los bloques de construcción de SOA son tres: proveedores de los servicios, registros de los servicios y clientes de los servicios. Para poder llevar adelante la interacción de estos bloques de construcción se han definido diversas especificaciones que han permitido que los distintos bloques puedan interactuar de manera ordenada y estándar.

Los servicios luego se comunican a partir de estas definiciones estándares, desde los protocolos de comunicación como es el caso de HTTP basados en la capa de transporte cuyo modelo es TCP/IP. Esta base común y estándar hace que SOA sea independiente del sistema operativo o lenguaje de programación que se elija para el desarrollo de cada servicio. Por encima de estos protocolos estándares, se basan las definiciones de los servicios, a través del lenguaje de especificación de servicios WSDL, el cual permite especificar la interacción de datos que se producirá con el servicio y a su vez el punto de acceso al mismo, el cual puede realizarse a través de protocolos estándares como es el caso de SOAP (*Simple Object Access Protocol*, Protocolo Simple de Acceso a Objetos) o REST (*Representational State Transfer*, Transferencia de Estado Representacional). Esto puede apreciarse en la Figura 5.1.

Una vez que los servicios se encuentran definidos, existen especificaciones que permiten llevar adelante la forma en que los mismos trabajarán en conjunto para llevar adelante una tarea o incluso un sistema completo de forma distribuida. Estas formas de coordinación de los distintos servicios se describen como orquestaciones o coreografías, cuyas especificaciones estándares se denominan BPEL y WS-CDL.

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 es proveedor, o, consumidor de un servicio, la coordinación de dispositivos encaja perfectamente con la composición de servicios.

### 5.3.2. Características de los dispositivos ubicuos

La computación ubicua se esfuerza por crear un paradigma completamente nuevo del entorno informático habitual en casi todos estos aspectos. Los

sistemas ubicuos utilizan un conjunto heterogéneo de computadoras, que incluyen dispositivos invisibles incrustados en objetos cotidianos como automóviles, electrodomésticos, máquinas y herramientas, etc., dispositivos móviles como PDAs y teléfonos inteligentes; ordenadores normales como computadoras portátiles, y dispositivos muy grandes como los smart TVs.

Todos estas computadoras tienen diferentes sistemas operativos, interfaces de red y capacidades de entrada / salida, entre otras. Sin embargo, dichas diferencias palidecen en comparación a las peculiaridades de los dispositivos ubicuos. Con mucha frecuencia, estos dispositivos no cuentan con un sistema operativo completo, o con una gran capacidad de memoria o procesamiento, ya que estos dispositivos están diseñados para llevar adelante una tarea específica, lo cual a su vez permite reducir el tamaño y el costo de producción de los mismos.

Las características contempladas para esta investigación son las indicadas a continuación. Si bien los dispositivos ubicuos pueden clasificarse usando distintas características o propiedades que las que aquí se enumeran, las seleccionadas son las más distintivas y/o las que pueden impactar en la convergencia con SOA.

- **Recursos limitados**: Algunos tipos de dispositivos ubicuos, tienen limitaciones de: memoria para el almacenamiento, batería, visualización, de ingreso de información y escasa capacidad computacional (Poslad, 2009, pag. 19,76).

- **Movilidad y volatilidad en el acceso a servicios**: Los dispositivos ubicuos tienen alta movilidad, lo cual conlleva a que pueda acceder a zonas con poca conectividad o incluso nula y estar situados en ambientes con mucha interferencia, lo que hace que tengan microcortes en su conectividad. (Poslad, 2009, pag. 76), (Krumm, 2010, pag. 41).

- **Heterogeneidad**: Los dispositivos son heterogéneos, por lo que la conectividad entre ellos debe hacerse de diversas maneras para soportar las distintas alternativas que presentan dichos dispositivos. Todos los dispositivos tienen diferentes sistemas operativos, interfaces de red, entrada y salida de datos. Algunos permiten la interacción con el humano, otros no, como el caso de los sensores.(Poslad, 2009, pag. 19,76), (Krumm, 2010, pag. 39,43,44).

- **Conectividad wireless**: Estos dispositivos poseen la capacidad de conectividad a redes sin cables, a través de distintas tecnologías como pueden ser bluetooth, Wifi (802.11), RFID, NFC (*Near Field Communication*, Comunicación cercana) (Poslad, 2009, pag. 4), (Lien et al., 2011).

- **Embebidos/Relacionados con el contexto**: Muchos dispositivos se encuentran embebidos en otros de mayor capacidad de procesamiento

o almacenamiento, del cual se alimentan; por ejemplo: sensores conectados a una placa arduino o Raspberry Pi. (Poslad, 2009, pag. 4).

- **Openness**: Permite que se le puedan agregar componentes al dispositivo, de una manera abierta, para su implementación final. Estas nuevas características o componentes pueden añadirse en cualquier momento de la vida del dispositivo, incluso cuando se encuentra funcionando. (Poslad, 2009, pag. 11,19).
- **Autonomía**: El dispositivo, por más pequeño que sea, tiene autonomía en el más amplio sentido de la palabra. Él mismo puede decidir cómo, cuándo y con quién comunicarse, así como a través de qué mecanismo (Poslad, 2009, pag. 15).
- **Invisibilidad**: Los dispositivos pueden ser totalmente invisibles tanto para el resto de los componentes de una coreografía como para los seres humanos. Tal es el caso de los dispositivos wereables, que se disimulan entre la ropa de las personas sin ser percibidos. (Krumm, 2010, pag. 48)
- **Seguridad y Privacidad**: La seguridad y privacidad son restricciones que se están imponiendo en todos los sistemas computacionales. Con los dispositivos ubicuos estos desafíos se incrementan debido a su volatilidad, espontaneidad, heterogeneidad y su naturaleza de invisibilidad (Krumm, 2010, pag. 49).

### 5.3.3. Convergencia entre SOA y los dispositivos ubicuos

Para lograr una convergencia entre SOA y los dispositivos ubicuos, es necesario armonizar las características de ambas tecnologías. A modo de ejemplo, la limitación en términos de memoria de los dispositivos ubicuos impide utilizar soluciones estándar, ej: *apache web server*, para la implementación del protocolo HTTP. Una solución a esta incompatibilidad es desarrollar un servidor web ad-hoc específico para los dispositivos que o bien no tienen un sistema operativo sobre el cual poder instalar soluciones ya probadas o que no tienen la memoria suficiente para poder ejecutar programas más complejos.

La tabla 5.1 detalla cómo las limitaciones de los dispositivos ubicuos puede manejarse técnicamente para converger con los conceptos y tecnologías SOA.

| **Limitaciones de los disp. ubicuos vs. requisitos técnicos para incluirlos en una composición SOA** | | |
|---|---|---|
| Característica | Implicancia | Solución planteada |
| *Recursos limitados* | Estas limitaciones implican que el dispositivo pueda:<br>- Quedarse sin batería mientras está en medio de una ejecución de una coreografía (la cual puede contener transacciones).<br>- No poseer memoria o capacidad computacional suficiente para ejecutar el framework que se propone para la ejecución de la coreografía. | Se propone realizar implementaciones ad-hoc en lugar de utilizar software disponible en el mercado. Por ejemplo, se podría desarrollar un servidor HTTP específico en lugar de usar un servidor Apache.<br>Esto permitiría hacer un uso optimizado de la memoria disponible. |
| *Movilidad y volatilidad en el acceso a servicios* | El hecho de que el dispositivo tenga la movilidad como una de sus características se puede ver como una ventaja por un lado, pero es una desventaja por el otro, ya que puede desaparecer (dejar de tener cobertura de conectividad wifi, bluetooth, etc.) mientras se encuentra realizando o participando de una coreografía, lo que implica que el resto de los integrantes dejen de tener respuesta por parte de este dispositivo. | Esto implica que debe existir un sistema alternativo para poder suplir su desaparición transitoria o definitiva.<br>Para ello existen distintas alternativas, algunas de las cuales se implementarán como parte de la solución, ej:<br>- Implementar las restricciones que propone la especificación WS-CDL, en base a time-outs, con lo cual se pueden implementar sistemas de rollback para que la coreografía culmine de manera correcta.<br>- Realizar extensiones en la especificación WS-CDL para poder utilizar dispositivos de reemplazo o clonado de los mismos. |

| **Limitaciones de los disp. ubicuos vs. requisitos técnicos para incluirlos en una composición SOA** | | |
|---|---|---|
| Característica | Implicancia | Solución planteada |
| *Heterogeneidad* | El hecho de que los distintos dispositivos sean heterogéneos implica que se debe encontrar una manera de que se comuniquen y puedan interactuar de una manera abierta, estándar y que se realice en capas de conectividad más altas. | Utilización de tecnología de servicios REST, la cual es estándar, abierta y se encuadra perfectamente dentro de SOA y de WS-CDL, si bien es cierto que deberán realizarse adaptaciones para los distintos dispositivos que compongan la coreografía. |
| *Conectividad Wireless* | La comunicación entre dispositivos puede hacerse de distintas maneras, siempre sin cables. Esto implica interconectar dispositivos con bluetooth o wifi o cualquier otra tecnología que sea wireless, de manera que por algún mecanismo los dispositivos puedan comunicarse entre ellos para llevar adelante la coreografía. | Programar interfaces de comunicación para las distintas alternativas de transferencia de información, dentro del framework de coreografías propuesto. |
| *Embebidos* | No posee, ya que la comunicación normalmente se realiza con el equipo que contiene al dispositivo de menor capacidad. | El framework propuesto se adaptaría perfectamente a este modelo, aunque no vamos a trabajar con dispositivos embebidos. |
| *Openness* | Se debe tener un framework basado en estándares abiertos para que la comunicación entre dispositivos de distintos fabricantes sea posible. | El framework propuesto, basado en SOA y WS-CDL es abierto por definición, lo cual encaja a la medida con esta característica. |
| *Autonomía* | El dispositivo pueda seleccionar la conectividad que mejor se adapte a sus necesidades o que pueda desplazarse de un lugar a otro o desaparecer. | Se encuentra dentro de las soluciones planteadas para las características anteriores. |

| **Limitaciones de los disp. ubicuos vs. requisitos técnicos para incluirlos en una composición SOA** | | |
|---|---|---|
| Característica | Implicancia | Solución planteada |
| *Invisibilidad* | Los dispositivos pueden ser totalmente invisibles tanto para el resto de los componentes de una coreografía como de los seres humanos. Tal es el caso de los dispositivos werables, que se disimulan entre la ropa de las personas sin ser percibidos. Por lo que, al ser tan pequeños, tienen limitantes de recursos. | Resolución en Recursos limitados, ya que al tener limitaciones de recursos por su tamaño se debe tomar la misma solución planteada. |
| *Seguridad y privacidad* | Las comunicaciones entre los dispositivos deben ser autenticadas y bajo capas de transporte encriptadas. | Se deben programar capas de autenticación en el framework, así como basarse en los casos que se pueda en capas seguras ya existentes como SSL. En los casos en que estas capas no existan (debido a la naturaleza del protocolo de conectividad utilizado) se deberán programar las mismas teniendo en cuenta muchas de las características ya enumeradas como son la heterogeneidad, recursos limitados, etc. |

Tabla 5.1: Características de los dispositivos ubicuos

## 5.4. Tipos de dispositivos ubicuos

La metodología de “Design Science” exige la evaluación continuada de prototipos o pruebas de concepto que implementen soluciones al problema planteado. Ello implica, entre otras cosas, seleccionar el hardware concreto que se debe utilizar en el marco de la tesis.

La tabla 5.2 muestra la lista de dispositivos que formarán parte de las pruebas de concepto que se llevarán adelante durante este trabajo de tesis. En dicha tabla hacemos una pequeña descripción de las características de cada dispositivo, así como una clasificación de los mismos de acuerdo a su capacidad.

| Clasificación | Descripción | Dispositivo comercial equiparable |
|---|---|---|
| Alta capacidad | - Gran cantidad de memoria RAM (8 GB mínimo)<br>- Gran capacidad de procesamiento (procesadores de 2 o más núcleos, 64 bits)<br>- Conexión de distintas maneras (wifi, red cableada, gran ancho de banda disponible)<br>- Fuente de energía constante con UPS | Servidores, equipos de escritorio, notebooks, etc. |
| Capacidad media - alta | - Memoria RAM de hasta 4GB<br>- Procesadores de más de 4 núcleos<br>- Alta movilidad<br>- Capacidad wireless únicamente | Móviles de alta gama<br>Tabletas<br>Raspberry pi o alternativas |
| Capacidad media - baja | - Memoria levemente escasa (32 kb máximo)<br>- Capacidad de procesamiento baja, único procesador con 16 bits de procesamiento como máximo. | Arduino Mega, Arduino UNO<br>Móviles de gama media o baja |
| Baja capacidad | - Escasa capacidad de procesamiento (8 bits a poca velocidad)<br>- Pequeños (entre 5 y 10 cms.)<br>- Escasa memoria (2kb máximo) | Arduino NANO o alternativos |
| Werables | - Muy pequeños (menos de 5 cms de diámetro)<br>- Capacidad de procesamiento muy baja<br>- Escasa memoria (512 bytes máximo) | Arduino GEMMA |

Tabla 5.2: Lista de dispositivos

De esta lista de dispositivos candidatos a ser utilizados durante la prueba de concepto, hemos seleccionado solamente algunos de ellos. Esta selección se basa principalmente en el tipo de dispositivos que se utilizan; si la prueba de concepto (ver sección 5.5.2) se implementase en la realidad. Para tal fin, se ha decidido utilizar dispositivos tanto de alta capacidad, dispositivos de capacidad media, dispositivos de media-baja y dispositivos de baja capacidad.

Los dispositivos denominados como werables no se han utilizado ya que poseen una capacidad de procesamiento muy similar a la de una placa Arduino Nano. Por lo tanto su única ventaja sería su escaso tamaño físico, lo cual no ha resultado relevante para este trabajo de tesis.

## 5.5. Solución planteada

A partir de la lista de características enumeradas anteriormente, en base al análisis que se hizo de cada una de las mismas, y de acuerdo al impacto que tendrían sobre la aplicación de estas características en la ejecución de una coreografía es que se diagramó las pruebas de concepto que se van a realizar. A su vez se tuvo en cuenta las características de cada uno de los dispositivos que son incorporadas en cada uno de los ciclos o pruebas de concepto. Dichas pruebas de concepto son las que se ajustan a la metodología de design-science seleccionada para este proyecto de tesis. Para llevar adelante estas pruebas de concepto se ha escogido un escenario de trabajo sobre el cual se realizarán las distintas evaluaciones luego de cada ciclo de investigación. A continuación detallamos el escenario de trabajo para luego abocarnos a las pruebas de concepto definidas.

### 5.5.1. Definición del escenario de trabajo

El escenario de trabajo que se plantea es específico del área de autopistas inteligentes o smart highways. Sin embargo, no deja de dar solución a un problema que se puede generalizar hacia prácticamente cualquier ambiente de trabajo. El escenario que planteamos se relaciona con los problemas de tránsito vehicular a lo largo de autopistas, rutas o carreteras, haciendo de todos ellos un entorno inteligente. Es decir, no solamente la carretera resulta inteligente, sino también todos los elementos que la rodean.

Este escenario puede esquematizarse tal y como indica la figura 6.14. Planteamos un ambiente donde los automóviles y ómnibus que transportan personas posean ciertos elementos de comunicación y sensores que permitan convertirlos en vehículos con seguridad que puedan prevenir accidentes. Dichos sensores podrían monitorear la salud del conductor en tiempo real, las funciones de seguridad del rodado, etc., y actuar en consecuencia. Por ejemplo, en caso de infarto del conductor, el vehículo podría iniciar una maniobra de frenado (dependiente de la tecnología del vehículo, la cual está

fuera del alcance de esta tesis, pero podría a su vez implementarse con la misma solución resultado de esta investigación), y advertir del posible peligro de accidente a otros vehículos que circulan por la misma ruta para que puedan tomar acciones preventivas. Para ello, la carretera deberá disponer de balizas de contacto con los vehículos que transitan por ella. Esto es visualizado en al Figura a través de las flechas que comunican los distintos actores del escenario. Este escenario puede complementarse con operaciones realizadas por sistemas de información clásicos. Por ejemplo, las balizas podrían disponer de la capacidad añadida de conectarse para solicitar automáticamente ayuda médica, policial, etc.

Figura 5.2: Esquema del escenario de trabajo

El escenario planteado posee, adicionalmente, ciertas características que lo hacen asimilable a los entornos reales donde los dispositivos ubicuos operan. En una carretera donde circulan vehículos, pueden darse circunstancias de:

- Zonas con poca conectividad.
- Escaso nivel de batería de los sensores o elementos de comunicación.
- Errores o desaparición de dispositivos durante la comunicación o manejo de la transacción.

El escenario planteado es comparable con los utilizados en distintas investigaciones acerca de sistemas ubicuos y pervasivos. Por ejemplo, en Ling Yibo et al. (2011) se mencionan distintos estudios de investigación que se centran en la utilización de dispositivos para la detección precoz de infartos de corazón, ya sean éstos por dispositivos que se implantan en la persona o mediante elementos que se disponen sobre el volante del vehículo para sensorizar al conductor. En cualquiera de los casos se trata de dispositivos que solamente interactúan con el vehículo (para tomar medidas correctivas), pero sin poder realizar avisos a otros participantes para actuar en consecuencia.

El escenario planteado puede ser fácilmente abstraído para obtener una solución genérica a problemas del mundo real, donde la ubicuidad sea el común denominador. Siendo más concretos en esta apreciación, nos referimos por ejemplo a la posibilidad de colaboración entre distintos dispositivos para lograr un mecanismo de comunicación más eficiente, como puede ser el caso de monitorización de personas con problemas de salud.

### 5.5.2. Pruebas de concepto

Se planificaron seis pruebas de concepto, las cuales enumeramos a continuación junto con una breve descripción de las características de dicho ciclo y con los dispositivos utilizados. La utilización de seis pruebas de concepto se debe a que es la cantidad de ciclos necesarios para poder implementar las características de los dispositivos ubicuos planteadas en la tabla 5.1. En el capítulo siguiente se hará un desarrollo más en profundidad y detalle de cada una de las pruebas de concepto utilizadas.

- **Primera prueba de concepto**: En esta primera prueba de concepto, se pondrá a prueba la primera versión del framework de ejecución de una coreografía escrita en WS-CDL. Los dispositivos involucrados han sido los catalogados dentro de la tabla (referencia) como de Alta capacidad y capacidad media. Los equipos utilizados son los siguientes:
  - Equipo servidor: PC con 16GB de memoria RAM, con procesador de 4 núcleos conectado a una dirección IP de acceso público. Este equipo dispone de un sistema operativo Linux, distribución KUbuntu versión 16.04. Posee instalado un servidor web Apache 2.4.18, PHP versión 7.0.15 y una base de Datos PostgreSQL 9.3 (esta es la que permite la grabación de una especie de bitácora de la ejecución de la coreografía).
  - Equipo Laptop: notebook de escritorio con 12GB de memoria RAM, procesador de doble núcleo conectado a una VPN (*Virtual Private Network*, Red Privada Virtual) con el servidor principal mencionado en el punto anterior. Este equipo dispone de un sistema operativo Linux, distribución KUbuntu versión 16.04. Posee instalado un servidor web 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 equipado con un sistema operativo RaspBian (distribución Linux adaptada para RaspberryPi). Posee instalado un servidor web Lighttpd versión 1.2, PHP versión 5.5 trabajando en modo CGI. Este equipo está conectado a la VPN que tiene como servidor al equipo server mencionado previamente.

  Como podemos observar en esta primera prueba no se han incorporado casi dispositivos ubicuos, salvo uno, debido a que la importancia de este ciclo es el de poner en funcionamiento el framework en un entorno simulado pero de características reales. De todas formas podemos observar que ya existe una heterogeneidad de dispositivos junto con algunas características limitadas tanto en tamaño, memoria y capacidad de procesamiento como es el caso de la Raspberry Pi.

- **Segunda prueba de concepto**: En esta segunda prueba de concepto presentaremos las distintas características que se han anexado a la primera versión del framework de ejecución de coreografías planteado previamente. Se incorporan en esta segunda etapa nuevos dispositivos ubicuos, cuyas prestaciones son menores a las que presentaron los equipos en la primera prueba de concepto. Las características que se incorporan son las de Recursos limitados, ya que se incorpora una placa Arduino Mega 2560, que posee 8k de memoria para datos y un procesador de 16 bits; Conectividad wireless ya que la placa Arduino hace su conexión a través de un módulo wifi (ESP8266-01) con el que se comunica a bajo nivel. Se tienen que hacer adaptaciones al framework de ejecución de coreografías ya que la escasa cantidad de memoria disponible no hace posible mantener en memoria toda la descripción de la coreografía, ni todas las variantes que implica. Estas adaptaciones se centrarán básicamente en hacer una síntesis de la coreografía, conteniendo información básica para el dispositivo o servicio que se esté representando.

- **Tercera prueba de concepto**: En esta tercera prueba de concepto presentaremos las distintas características que se han anexado a la segunda versión del framework de ejecución de coreografías planteado previamente 6.1 y 6.2. Se incorporan en esta tercera etapa nuevos dispositivos ubicuos, cuyas prestaciones son menores a las que presentaron los equipos en la segunda prueba de concepto. Se espera con esta nueva ejecución poder incorporar nuevas características de los sistemas ubicuos en la ejecución de la coreografía. Se sigue trabajando sobre las características de Recursos limitados, conectividad wireless y de heterogeneidad. Para este ciclo se incorpora una placa Arduino Nano V3 donde la memoria disponible para datos es de apenas 2K,

lo cual implica nuevos desafíos a nivel de programación y adaptación del código de ejecución de la coreografía, así como la de la arquitectura SOA implementada previamente para dar soporte a los servicios ubicuos disponibles en estos dispositivos. Se espera que se pueda encontrar un límite inferior que permita mostrar cuáles son las características mínimas necesarias para poder ejecutar coreografías bajo arquitectura SOA/REST en ambientes ubicuos.

- **Cuarta prueba de concepto**: En esta cuarta prueba de concepto abordaremos una de las características de los dispositivos ubicuos, que es su escaso nivel de batería lo cual puede provocar que el mismo desaparezca al momento de ser invocado o bien pueda truncarse la ejecución del servicio que brinda el mismo. En este caso se trabajará sobre agregados al framework de ejecución de coreografías sobre la placa Arduino, para que optimice y extienda la vida útil de la batería para no tener desconexiones repentinas o no controladas por la falta de energía. El hecho de poder controlar los niveles de batería facilitan la tarea de realizar desapariciones controladas, de manera tal que se puedan manejar alternativas para el normal cumplimiento de la coreografía.

- **Quinta prueba de concepto**: En esta quinta prueba de concepto trabajaremos con el manejo de una transacción distribuida a lo largo de la ejecución de la coreografía. Si bien el manejo de una transacción no es algo inherente únicamente a los dispositivos ubicuos, nos resultó interesante poder mostrar su implementación en un ambiente de dispositivos ubicuos y con servicios REST, lo que lo convierte en algo más complejo que en ambientes con servicios web SOAP. En este prueba trabajaremos con los mismos dispositivos y arquitectura que trabajamos en la segunda prueba de concepto.

- **Sexta prueba de concepto**: Esta prueba de concepto se centra fundamentalmente en las desapariciones no controladas de los dispositivos. Esta desaparición se puede advertir a través de la característica de timeout que se encuentra en la especificación de la coreografía, a través del lenguaje WS-CDL. El comportamiento habitual en este tipo de casos es la culminación de la coreografía o bien la decisión de no hacer ninguna tarea específica ante este evento. Si bien el manejo de timeouts y la implementación del mismo dentro de las coreografías no resulta innovador, ya que en el ámbito de los servicios web ha sido abordado desde sus comienzos, resulta interesante abordarlo ya que en muchos casos no son manejados de la manera correcta ni se tienen en cuenta a los fines prácticos, ya que en muchos lenguajes de programación pueden ser atrapados como un error y tratarlo como tal. Sin embargo en el manejo de coreografías resulta interesante porque estos pueden producirse por problemas de desapariciones no controladas de los dispositivos

ubicuos fundamentalmente. A partir de ello en este ciclo se abordará de dos maneras posibles: se finaliza la coreografía de manera inmediata, marcándola como no utilizable para futuras llamadas y como segunda opción la de llevar adelante alternativas que puedan atrapar de manera preventiva estas desapariciones. Para este último caso se pueden tomar distintas alternativas, las cuales se evalúan como parte de la solución específica dentro del siguiente capítulo de este trabajo de Tesis.

En cualquiera de los casos de la desaparición, se deberá dar aviso al manejador de la transacción para que pueda volver atrás la ejecución de la misma y que no haya conflictos posteriores.

A partir de cada prueba de concepto, se realizará un análisis de los resultados esperados para poder retroalimentar la siguiente fase o prueba de manera tal de ajustarse a la metodología de investigación seleccionada en este caso. Luego de la ejecución de las pruebas de concepto se podrá obtener como resultado un prototipo de ejecución de coreografías tanto para dispositivos normales como para dispositivos ubicuos, foco de la presente investigación. A su vez, se dispondrá de las mejoras, cambios y ajuste que se deben realizar al lenguaje de especificación y ejecución de coreografías WS-CDL.

De acuerdo a la metodología de investigación seleccionada, podemos observar en el diagrama 5.3 que las distintas pruebas de concepto o ciclos en el desarrollo del prototipo expuestas se equiparan con los pasos de **selección del escenario, prototipo, análisis de resultados y aplicación de mejoras** para obtener como resultado el modelo o framework de composición. Claramente estos pasos de investigación se relacionan con el ciclo de desarrollo planteado por la metodología “Design science”. Los resultados obtenidos serán parte del ciclo de rigor.

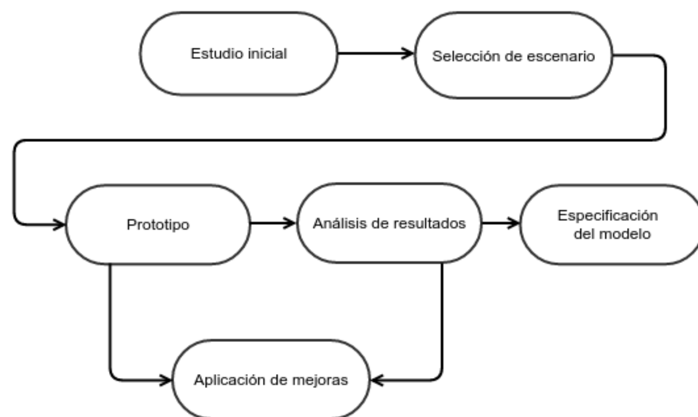


Figura 5.3: Diagrama de Flujo - Pasos de la investigación.

# Capítulo 6

# Resolución

Durante el presente capítulo nos abocaremos a la resolución propiamente dicha de la investigación planteada originalmente. En este punto es donde mostraremos cada uno de los ciclos de Design-Science que hemos implementado, los cuales ya comentamos en el capítulo anterior, junto al escenario de trabajo planteado. Para poder mostrar los resultados de esta resolución hemos dividido el capítulo en secciones, una por cada ciclo de diseño. En cada ciclo nos encargamos de mostrar y describir la definición de la coreografía, los dispositivos utilizados, qué partes se han implementado del framework desarrollado en dicho ciclo y por último se aborda una evaluación de los resultados que se obtienen luego de cada ciclo.

Como hemos mencionado, son 6 ciclos de diseño y por cada uno de ellos hacemos un análisis profundo tanto para mostrar cómo se ha llevado adelante como así también de los resultados que se obtuvieron.

## 6.1. Primer ciclo de Design-Science

En este primer ciclo de Design-Science, se pondrá a prueba la primera versión del framework de ejecución de una coreografía escrita en WS-CDL. A continuación, en las distintas subsecciones describiremos el diseño que hemos creado para esta primera etapa, donde se podrán apreciar desde las características utilizadas del lenguaje de definición de coreografías hasta los dispositivos y lenguajes empleados.

### 6.1.1. Características utilizadas de WS-Choreography

Para la definición de la coreografía se utilizó la especificación 1.0 de WS-CDL.

### 6.1.2. Definición de la coreografía del escenario planteado

Para el caso seleccionado, se han abordado los siguientes roles:

- **VehiculoTransito**: es cualquier vehículo que se encuentra circulando por la autopista o ruta, que puede recibir o enviar algún mensaje y participar de la coreografía.

- **VehiculoAccidentado**: es aquel vehículo que ha tenido algún tipo de incidente o accidente mientras transita o circula a lo largo de la carretera inteligente.

- **Baliza**: son aquellos dispositivos dispuestos a lo largo de la autopista, cada determinada cantidad de distancia, que permite tener comunicación hacia una central de la autopista, con capacidad de procesar peticiones de los dispositivos ubicuos dispuestos en los automóviles.

- **CentralBalizas**: equipo central, que reúne la información de todas las balizas que forman parte de la autopista, donde todas éstas se comunican para enviar sus novedades.

- **CentralEmergencias**: central donde se reúnen todas las peticiones de emergencias, las cuales deben activar el llamado a distintos entes y organismos para la solución del problema, de acuerdo a la gravedad que se informa.

La coreografía que se ha definido, la cual se puede encontrar en el Anexo E, emplea las siguientes interacciones:

- El iniciador es el rol de **VehiculoAccidentado**, el cual ante un incidente contacta a la **Baliza** más cercana, informando del problema ocurrido, donde uno de los datos principales es el número de patente del mismo.

- La **Baliza** se encarga de recibir el pedido de información del **VehiculoAccidentado** y contacta a la **CentralBalizas** y a los otros **VehiculoTransito** que se encuentren en la zona del incidente, informándoles lo sucedido y la ubicación del problema.

- La **CentralBalizas** se encarga de registrar el evento comunicado y de avisar a la **CentralEmergencias** para que pueda determinar las acciones y personal que debe intervenir en el incidente informado.

Un diagrama de ejecución se encuentra en la Figura 6.1 donde se muestra un diagrama de secuencia, donde en el mismo se pueden apreciar los distintos roles que participan de la coreografía y los mensajes y servicios que son invocados en cada paso.

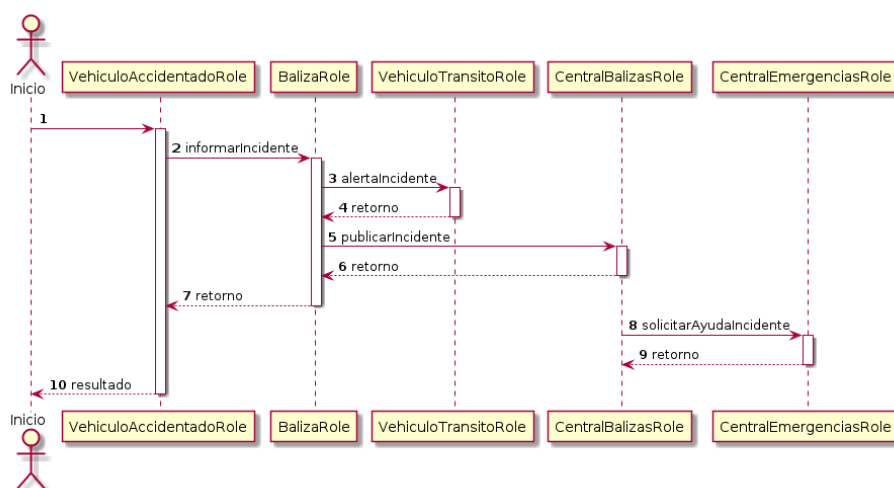


Figura 6.1: Diagrama de secuencia de ejecución de la coreografía

### 6.1.3. Dispositivos utilizados

En esta primera prueba de concepto los dispositivos utilizados han sido los catalogados dentro de la tabla 5.2 como de alta capacidad y capacidad media, ya que en este primer ciclo no se busca que sean de baja potencia sino que puedan ejecutar de manera exitosa una coreografía. Los equipos utilizados han sido los siguientes:

- Equipo servidor: PC con una dirección IP de acceso público. Este equipo dispone de un sistema operativo Linux, distribución KUbuntu versión 16.04. Posee instalado un servidor web Apache 2.4.18, PHP versión 7.0.15 y una base de Datos PostgreSQL 9.3 . Este equipo será el encargado de implementar los roles de: CentralBaliza y CentralEmergencia. La selección de este equipamiento para representar a estos roles se debe a dos motivos, en primer lugar a su alta capacidad de procesamiento para poder llevar adelante la ejecución de la coreografía en su primera versión del framework, y en segundo lugar debido a que representa roles que formarían parte de una aplicación clásica en SOA.

- 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. Este equipo dispone de un sistema operativo Linux, distribución KUbuntu versión 16.04. Posee instalado un servidor web Apache 2.4.18, PHP versión 7.0.15. Representará los roles de VehiculoAccidentado y VehiculoTransito. Esto se debe principalmente a su capacidad de procesamiento para la primera versión del framework.

- Equipo RaspberryPi B+: equipo con un procesador de 32 bits, con un

único núcleo, 1GB de memoria RAM equipado con un sistema operativo RaspBian (distribución Linux adaptada para RaspberryPi). Posee instalado un servidor web Lighttpd versión 1.2, PHP versión 5.5 trabajando en modo CGI (*Common Gateway Interface*, Interfaz de entrada común). Este equipo está conectado a la VPN que tiene como servidor al equipo server mencionado previamente. Implementará el rol de Baliza, siendo un dispositivo con características suficientes para poder procesar la primera versión del framework de ejecución de coreografías.

### 6.1.4. Implementación del framework de coordinación

En las distintas secciones que continúan, presentaremos las distintas decisiones que se llevaron adelante para poner en funcionamiento la primera prueba de concepto del escenario planteado.

#### 6.1.4.1. Decisiones arquitectónicas

La arquitectura seleccionada no es otra que SOA, ya que es la base sobre la cual se sustenta el WS-CDL, que es el lenguaje de especificación de coreografías bajo servicios web. Dentro de los servicios web, se ha adoptado la arquitectura REST. REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP y permite obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON (*JavaScript Object Notation*, Notación de Objetos en JavaScript). El formato más usado en la actualidad es JSON, ya que es más ligero y legible en comparación al formato XML.

Por lo tanto REST se apoya en HTTP, más específicamente en los verbos definidos por HTTP: GET, POST, PUT y DELETE. De aquí surge una alternativa a SOAP (el protocolo para la utilización de servicios web). Ahora bien, REST llega a solucionar esa complejidad que añade SOAP, haciendo mucho más sencillo el desarrollo de un servicio en el cual se va a almacenar la lógica de negocio y exponer los datos con una serie de recursos URL.

Los lenguajes de programación utilizados son PHP versión 7.x y 5.x dependiendo de la capacidad de procesamiento de los dispositivos seleccionados como así también de las versiones estables disponibles en los sistemas operativos correspondientes. Este lenguaje se encuentra instalado sobre dos servidores web a saber: por un lado Apache 2.4.x y por el otro Lighttpd 1.2. El segundo fue seleccionado por su escaso consumo de memoria al momento de su ejecución. En todos los casos se ha utilizado distribuciones Linux como sistema operativo. En el caso del servidor se ha instalado también un servidor de base de datos PostgreSQL 9.3, ya este equipo es el encargado de la grabación de una especie de bitácora de la ejecución de la coreografía, la cual permite luego hacer diagramas de secuencia mostrando la forma en que se ejecutó el framework.

La forma de conexionado de los dispositivos considerados, se puede apreciar en la Figura 6.2, el cual se presenta en forma de diagrama de despliegue de UML (*Unified Modeling Language*, Lenguage Unificado de Modelado). En dicho diagrama se puede apreciar que dos equipos implementan más de un rol, esto se debe principalmente a la decisión de minimizar la cantidad de dispositivos utilizados para este primer ciclo de diseño.

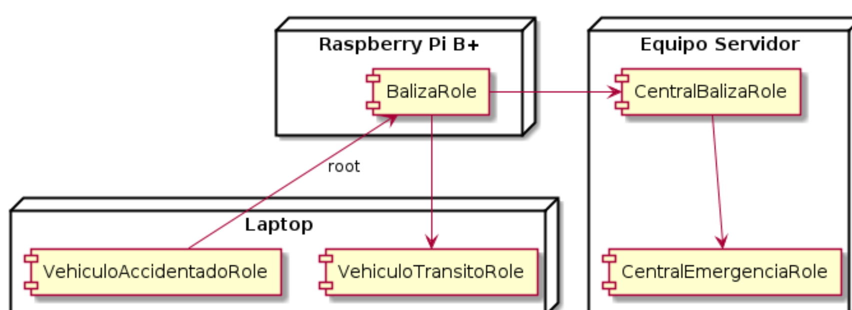


Figura 6.2: Diagrama de despliegue de dispositivos y roles

#### 6.1.4.2. Conectividad

Los distintos dispositivos que forman parte de esta primera versión de la implementación del lenguaje de coreografías, se encuentran todos bajo una misma red local, conectados a través de Internet. Existe un único equipo central conectado a Internet, el cual es el que actúa como servidor (tanto por su capacidad como su conectividad) y el resto forman parte de la red, ya que se encuentran conectados a otra red de área local, con conexión a Internet a través de un router.

#### 6.1.4.3. Lenguajes

El lenguaje seleccionado para la implementación de la primera versión del framework de ejecución de coreografías con lenguaje WS-CDL es PHP.

PHP es un lenguaje de script interpretado en el lado del servidor utilizado principalmente para la generación de páginas Web dinámicas, similar al ASP (*Active Server Pages*, Servidor de Páginas Activas) de Microsoft o el JSP (*Java Server Pages*, Servidor de Páginas Java) de Sun. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas particulares.

Las razones por las cuales se eligió este lenguaje se enumeran a continuación:

- Es un lenguaje de scripting que se ejecuta del lado del servidor.

- Es de código abierto y multiplataforma.
- Se ejecuta como un módulo dentro del servidor web Apache, lo cual logra una mejora en la velocidad de ejecución, mejor utilización de la memoria y un mantenimiento menor.
- Es orientado a objetos.
- Permite la codificación de servicios REST.
- Permite la integración con varias bibliotecas externas, lo cual permite analizar código tanto en XML como en JSON, base fundamental del desarrollo que necesitábamos realizar.
- Permite conexión a una gran variedad de motores de bases de datos (PostgreSQL, MySQL, Oracle, Microsoft SQL Server, DB2, etc).

#### 6.1.4.4. Servidores web

Debido a que la arquitectura seleccionada es SOA con implementación de los servicios con tecnología REST, se deben seleccionar servidores web que implementen dichas tecnologías del lado del servidor. En este caso los servidores seleccionados son dos:

- Apache: este servidor de código abierto y multiplataforma, además de poseer una gran solidez y robustez gracias a la comunidad de desarrolladores que posee, permite reglas de escritura para la escritura de direcciones o URLs, base fundamental de la programación de servicios REST. Adicionalmente este servidor permite que PHP se ejecute como un módulo dentro del espacio de memoria de este servidor, o también la posibilidad de ejecutarse de manera independiente como CGI, lo que evita que ocupe mayor espacio de memoria al ejecutarse.
- Lighttpd: es un servidor de código abierto, multiplataforma, rápido, seguro, flexible y que se apega a los estándares del W3C. Este servidor requiere muy poca memoria RAM para trabajar y menos capacidad de procesamiento. Solamente puede ejecutar PHP en modo CGI, ya que no hay módulos de este lenguaje para Lighttpd. Permite también reescritura de direcciones web, así como el redireccionamiento HTTP.

La selección se ha basado principalmente teniendo en cuenta como factores principales la fiabilidad, el apego a estándares, la velocidad de respuesta y el bajo consumo de recursos. Es por ello que se ha elegido Apache para que sea ejecutado en los equipos con mayor capacidad de procesamiento y Lighttpd se haya dejado para los equipos que poseen menos capacidad de procesamiento como es el caso de los equipos Raspberry Pi.

#### 6.1.4.5. Código

En este apartado hablaremos acerca del código utilizado en esta prueba de concepto. Ya hemos presentado la arquitectura seccionada, los lenguajes y los servidores web utilizados para esta primera prueba de concepto. El objetivo no es comentar todo el código utilizado sino describir la estructura del mismo, a través de diagramas de UML, en particular, el diagrama de clases, que se muestra en la Figura 6.3.

Se observa en la Figura 6.3 que existen varias clases, pero nos detendremos en dos de ellas, que son las que realizan la mayor parte del trabajo:

- `Clase Choreography`

  Esta es la clase que se encarga de procesar e interpretar la descripción en XML de la definición de la coreografía escrita en WS-CDL. Esta clase posee métodos que permiten saber qué acción o coreografía se encuentra ejecutando, los servicios que se deben ejecutar a continuación del actual, la forma en que se deben ejecutar los mismos (paralelo, secuencial, etc.) e incluso los detalles del servicio-dispositivo que se debe invocar a continuación. Para poder realizar este análisis, se debe conocer en cada dispositivo que forma parte de la coreografía el nombre del rol del mismo y el canal de ejecución. Esta es la primera clase que se debe instanciar cualquier dispositivo para poder conocer los detalles de ejecución de la coreografía.

- `Clase Choreography_service`

  Esta clase es la que nos permite interpretar la coreografía a nivel de un dispositivo y de la participación del mismo en la misma, es decir, esta clase se encarga de dialogar con la clase Choreography y determinar qué tareas le corresponde realizar al dispositivo en cuestión. De esta clase deben derivar todas las clases que quieran ser instanciadas en los dispositivos para llevar adelante un rol determinado.

Las restantes clases realizan funciones anexas a la Clase Choreography, permitiendo llevar adelante su tarea. Las otras clases que figuran en el diagrama corresponden a los distintos roles que son implementados en la ejecución del escenario planteado.

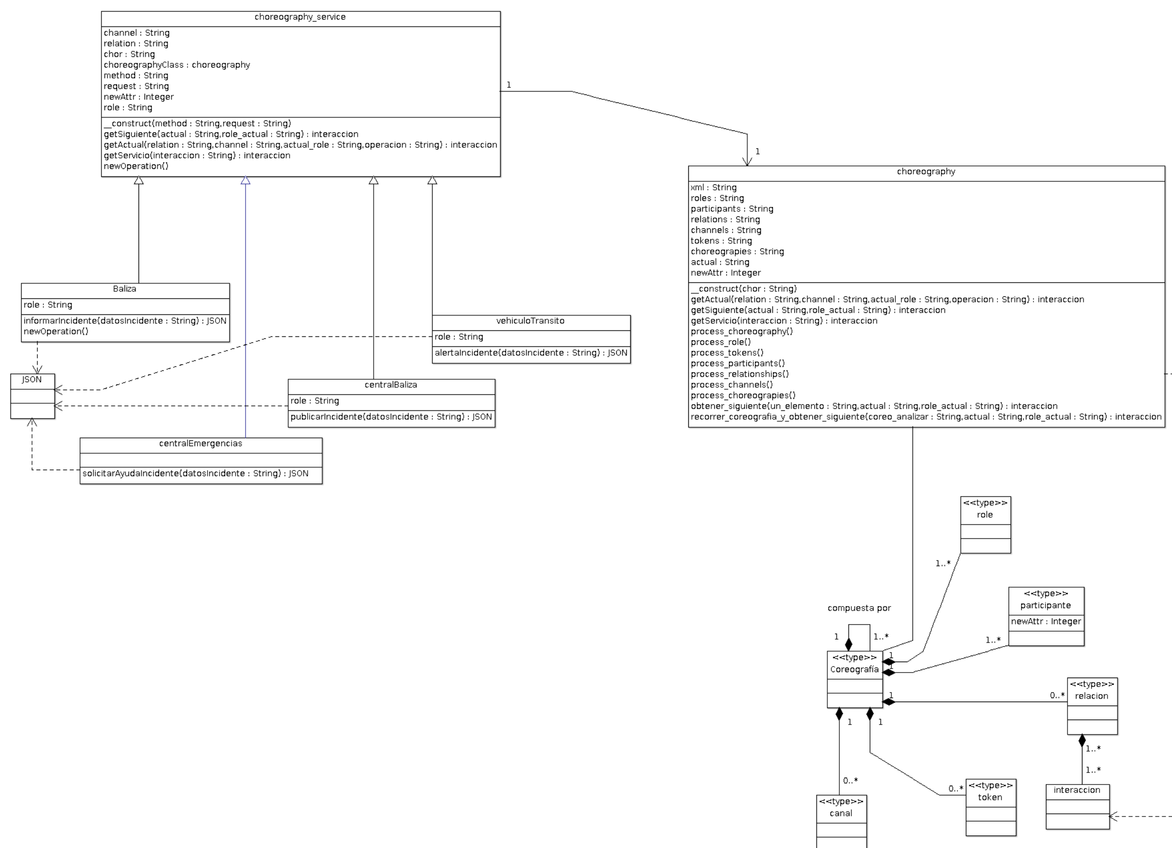


Figura 6.3: Diagrama de clases en PHP

### 6.1.5. Evaluación

#### 6.1.5.1. Prueba del software

Se produjeron errores simples propios de una primera prueba de concepto, los cuales fueron subsanados rápidamente, lo que permitió ejecutar exitosamente la coreografía. Dentro de estos errores destacan aquellos relacionados con la forma en que se han utilizado las distintas características del lenguaje de coreografías, como es el caso de la implementación de ejecución paralela, o la forma en que se debe manipular las llamadas a otras coreografías (a través de la cláusula PERFORM[1] del WS-CDL). Respecto del paralelismo, se produjo un problema para poder implementar dicha característica de una manera correcta en PHP. PHP no tiene una manera directa de implementar los hilos de ejecución (del inglés threads), por lo que su funcionamiento se tiene que hacer a través de módulos especiales, los cuales pueden introducir problemas de seguridad en el servidor donde se ejecutan.

#### 6.1.5.2. Mejoras pospuestas a futuras pruebas de concepto

Si bien la ejecución fue correcta, al evaluar el código ejecutado y las características de los equipos donde se ha ejecutado el prototipo, se hace necesario optimizar el código y adaptarlo para que pueda ser ejecutado en dispositivos que posean menos prestaciones (memoria, procesamiento, etc). Por lo tanto se debe revisar el código escrito en su totalidad para realizar una reingeniería del mismo para optimizar la utilización de memoria y procesamiento.

#### 6.1.5.3. Demostración

La figura 6.4 muestra el diagrama de la ejecución del framework en este primer ciclo de diseño. El mismo fue confeccionado automáticamente por el software que se encuentra desarrollado en el equipo servidor y que permite realizar una especie de bitácora de lo que se va ejecutando en la coreografía.

## 6.2. Segundo ciclo de Design-Science

En este segundo ciclo de Design-Science incorporaremos nuevos dispositivos, cuyas prestaciones son menores a las que poseen los equipos usados en el primer ciclo. En concreto utilizaremos dispositivos con capacidad media-baja, cuya especificación se puede observar en la tabla 5.2. Específicamente utilizaremos una placa Arduino Mega 2560. Si bien esta placa de desarrollo es la más avanzada dentro de la tecnología Arduino, tiene escasa capacidad tanto de procesamiento como de almacenamiento.

[1]Permite hacer llamadas a otras coreografías definidas en WS-CDL

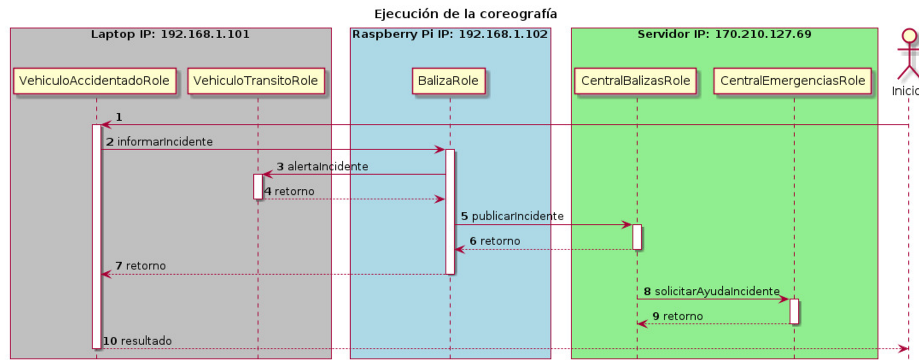


Figura 6.4: Diagrama de ejecución de la coreografía

### 6.2.1. Características utilizadas de WS-Choreography

Se deberán hacer adaptaciones al lenguaje de coreografías WS-CDL para que las ejecuciones en paralelo se lleven a cabo de una manera secuencial debido a las limitaciones de la placa Arduino, la cual no posee la capacidad de manejar hilos de ejecución en forma paralela.

### 6.2.2. Definición de la coreografía del escenario planteado

Se mantiene la misma definición de la coreografía que se ha planteado para el primer ciclo de diseño.

### 6.2.3. Dispositivos considerados

A los dispositivos ya considerados en la primera prueba de concepto, se anexa un dispositivo Arduino Mega 2560. Este dispositivo cuenta con 256KB de memoria flash para almacenar el código ejecutable, pero sólo 8KB para las variables de memoria. En este caso utilizar un dispositivo Arduino Mega con las características mencionadas es el paso directo para poder insertarnos en el espacio de los dispositivos ubicuos, ya que solo posee solamente el 0.05 % de capacidad de almacenamiento de la RaspberryPi B+ utilizada.

Para poder brindar conectividad a esta placa con el resto de los dispositivos que forman parte de la implementación, se ha decidido utilizar un módulo ESP8266-01 que brinda la posibilidad de unirse vía wi-fi, bajo el protocolo 802.11 b/g/n, a una red o router existente.

En este ciclo la placa Arduino Mega reemplazará a la RaspberryPi en la interpretación del rol Baliza (BalizaRole) que esta representaba en el primer ciclo de diseño. A su vez, la RaspberryPi tomará el lugar de la notebook y representará a los roles de VehiculoAccidentado y VehiculoTransito. De esta manera nos acercamos a los dispositivos ubicuos en la implementación del framework.

### 6.2.4. Implementación del framework de coordinación

En las distintas secciones que continúan, presentaremos las distintas decisiones que se llevaron adelante para poner en funcionamiento el segundo ciclo de diseño.

#### 6.2.4.1. Decisiones arquitectónicas

La arquitectura se mantiene la misma que en el primer ciclo de Design-Science. En este caso en particular ha sido necesario codificar parte de la infraestructura de servicios web REST a la plataforma de Arduino, ya que la misma no contiene ningún tipo de librería que permita la implementación ni de servicios REST como así tampoco la de un servidor web. Por lo tanto, se desarrolló una implementación "ad-hoc" de servidor web liviano específica para el manejo de recursos REST y para la ejecución de coreografías especificadas en WS-CDL.

Si bien existen algunas librerías de código abierto que implementan servicios REST a ser utilizadas para este caso particular, las mismas no se consideraron adecuadas ya que contaban con cierta cantidad de características que no iban a ser utilizadas, lo que sumaba espacio que no era utilizado a nivel código. Es importante destacar que a pesar de esta limitante de la capacidad de memoria, es posible realizar implementaciones parciales que aún mantienen la funcionalidad SOA.

De acuerdo a los dispositivos que se utilizarán (Arduino Mega), se necesitarán hacer adaptaciones al prototipo de ejecución de la coreografía, debido a que no se puede incluir la descripción de la misma en este tipo de placas Arduino, ya que su escasa capacidad de memoria no lo permite. Para ello se han generado fragmentos de la especificación que son inherentes al rol que cumple dentro de la coreografía la placa Arduino Mega (BalizaRole).

La forma de conexionado de los dispositivos considerados, se puede apreciar en la Figura 6.5, la cual se presenta en forma de diagrama de despliegue de UML.

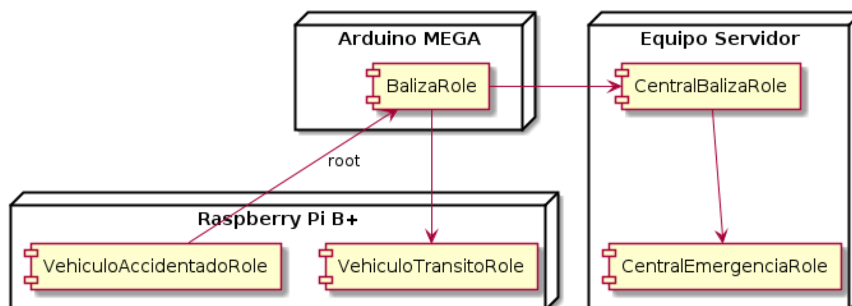


Figura 6.5: Diagrama de despliegue de dispositivos y roles

#### 6.2.4.2. Conectividad

El esquema de conectividad se mantiene de la misma manera que en el primer ciclo de diseño, con la salvedad que se incorpora dentro de la red a la placa Arduino Mega 2560.

#### 6.2.4.3. Lenguajes

A los lenguajes ya utilizados se anexó la programación en el lenguaje de programación que brinda Arduino a través de su entorno de desarrollo integrado. El mismo está basado en C++, por lo que es posible utilizar comandos estándar de C++ para la programación, lo que brinda una gran versatilidad y portabilidad del código. Por ejemplo, se podría portar fácilmente el desarrollo de soporte de servicios REST a otros dispositivos que soporte C++ en sus entornos de programación.

A su vez, para la programación del módulo ESP8266-01, se ha incorporado una librería de código abierto, denominada WiFiESP, la cual se puede descargar desde GitHub `https://github.com/bportaluri/WiFiEsp`, la que permitió programar el módulo a más alto nivel, sin tener que ejecutar directamente los comandos HAYES AT (Hay, 2010) que son los que reconoce dicho módulo.

#### 6.2.4.4. Servidores web

Los servidores web se han mantenido los mismos que fueron utilizados durante la ejecución de la primera prueba de concepto. Para el caso particular de la placa Arduino Mega en conjunto con el módulo ESP8266-01, se ha tenido que codificar un servidor web ad-hoc,

Este servidor web implementa todos los verbos disponibles en el protocolo HTTP: GET, PUT, POST y DELETE. Si bien, como hemos expresado anteriormente, nos hemos basado en la librería WifiEsp para el manejo principal de los flujos de entrada de bytes desde EL MÓDULO esp8266-01, se ha tenido que anexar todo lo que es el manejo de las peticiones e interpretación de las mismas. Nos referimos al hecho de que esta librería únicamente se encarga de comunicarse con el módulo ESP, pero no así de la interpretación de los distintos comandos que arriban o se envían desde la misma. Por lo tanto se ha tenido que trabajar en el desglosamiento de cada uno de los verbos implementados y su posterior procesamiento. A su vez, este servicio REST implementado, permite realizar el manejo de reglas de rewrite disponibles en servidores web avanzados como es el caso de Apache. Esto brinda la posibilidad de poder manejar distintas aplicaciones con el mismo código de procesamiento de dichas peticiones.

#### 6.2.4.5. Código

En este apartado hablaremos acerca del código utilizado en esta prueba de concepto, en particular, de las clases implementadas en el lenguaje C++, ya que para PHP se mantiene la misma estructuras de clases ya presentadas en la primera prueba de concepto. Las clases que se han codificado adicionalmente a las existentes son aquellas que están relacionadas directamente con la implementación del framework en el entorno de programación Arduino.

Las Figuras 6.6 y 6.7 muestran las clases más importantes que están involucradas tanto en el manejo del servidor web como en el manejo de la propia coreografía.

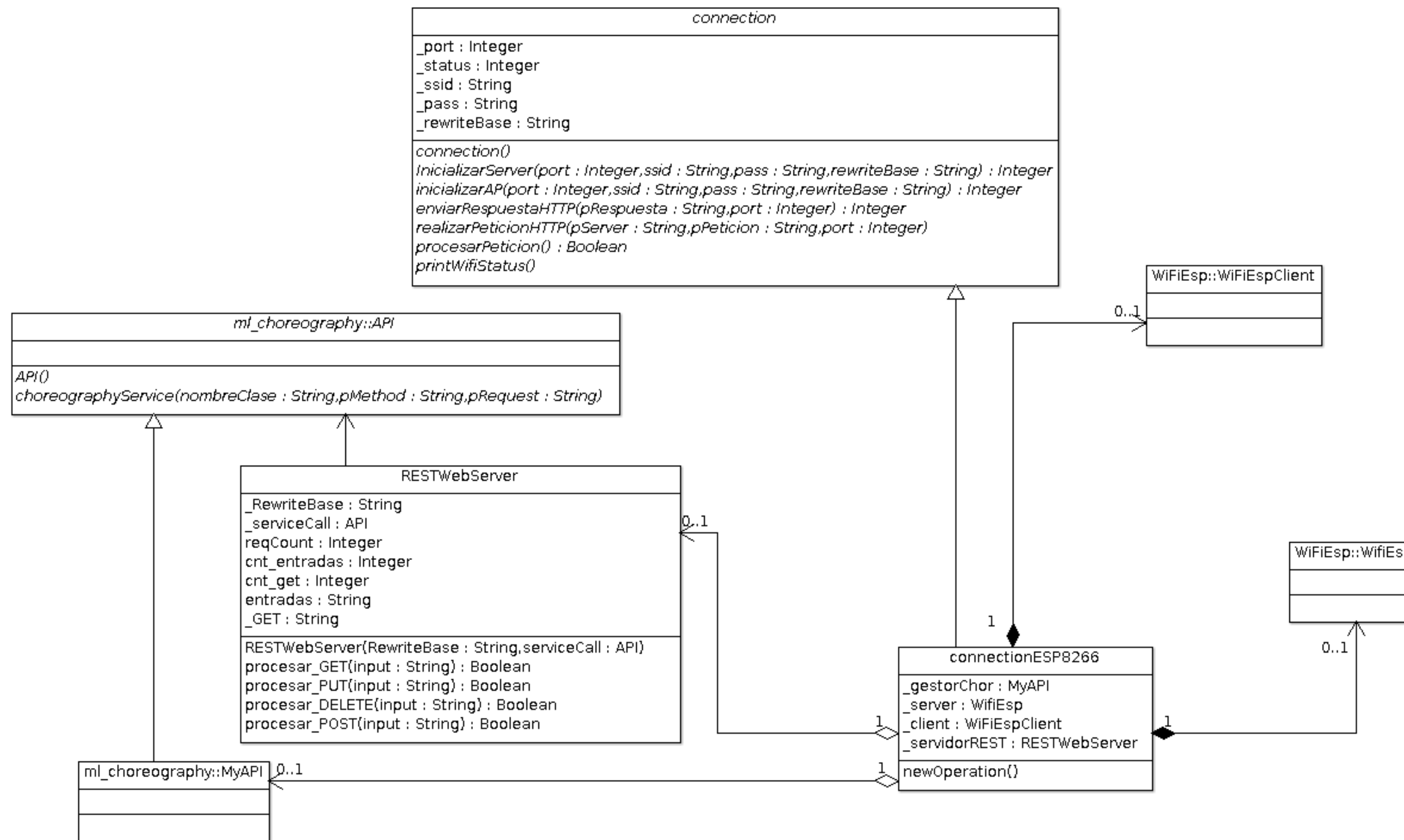


Figura 6.6: Diagrama de clases en C++

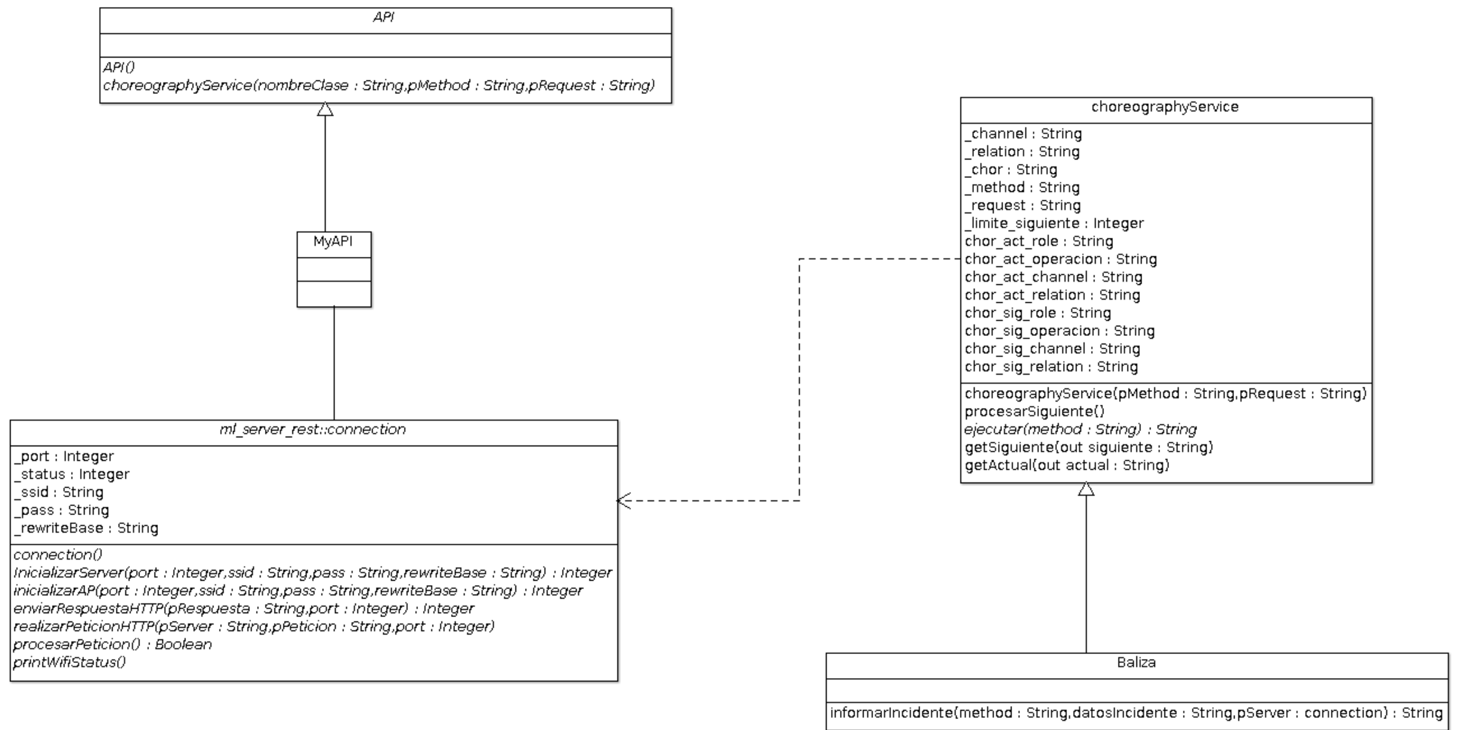


Figura 6.7: Diagrama de clases en C++

- `Clase connection`

  Esta clase es la encargada de administrar las conexiones que se produzcan a través de alguno de los módulos disponibles para la utilización de redes wifi. En este caso particular, se utiliza un módulo ESP8266-01, por lo que se instancia a través de la clase heredada connectionESP8266. Esta clase se encarga en principio de inicializar un servidor web y principalmente de recibir las peticiones HTTP y de realizar las respuestas a los clientes que se conectan.

- `Clase RESTWebServer`

  Esta clase es la encargada de procesar las peticiones de los servicios web a través de REST. Se encarga de administrar las llamadas a la API de coreografías.

- `Clase API`

  Esta clase, junto con choreographyService son las encargadas de instanciar las clases pertenecientes a los servicios implementados en el dispositivo. En este caso particular solamente se instancia el servicio del dispositivo Baliza (representado por la clase Baliza). Esta jerarquía de clases está diseñado a partir del patrón de diseño "Factory" (Pree, 1995).

### 6.2.5. Evaluación

Luego de transcurrido el segundo ciclo de diseño, podemos concluir que el mismo fue ejecutado de manera correcta por los dispositivos involucrados. Si bien se deben mencionar los inconvenientes propios de una codificación de este tipo (servidores web, servicios REST) sumado a la escasa capacidad de almacenamiento y procesamiento de los dispositivos involucrados, los mismos han sido subsanados para obtener la ejecución exitosa.

#### 6.2.5.1. Prueba del software

El código fuente utilizado en el primer ciclo de diseño, donde no se incluían dispositivos tan pequeños como la placa Arduino Mega, ha sido revisado y optimizado de manera tal que el mismo pueda ser trasladado al lenguaje C++, lenguaje por defecto de Arduino.

En esta primera versión en placas Arduino hubo inconvenientes simples a nivel de codificación propios del lenguaje C++, donde al no contar con un sistema operativo que sustente el desarrollo cada problema que se produce en la codificación implica que la placa se reinicie de forma completa y sin previo aviso.

El listado 6.1 muestra la cantidad de memoria utilizada por esta primera versión del framework, con una captura de los resultados de la compilación.

```
El Sketch usa 24888 bytes (9%) del espacio de almacenamiento de
programa
El maximo es 253952 bytes
Las variables Globales usan 1740 bytes (22%) de la memoria dinamica,
dejando 6452 bytes para las variables locales
El maximo es 8192 bytes
```

Listado 6.1: 1ra. Compilación en entorno Arduino

Como observamos en este listado, el código utiliza muy poca memoria y capacidad de la placa Arduino Mega. Si bien este espacio resulta suficiente para una placa de este estilo, para casos donde los dispositivos contengan menos capacidad puede ser problemático.

La placa Arduino Mega cuenta con mucha capacidad para el almacenamiento de programas (256KB), sin embargo la capacidad de almacenamiento para variables globales y locales no resulta ser suficiente en algunos casos. En algunos picos de consumo de memoria SRAM (*Static Random Access Memory*, Memoria estática de acceso aleatorio) (lugar donde se almacenan este tipo de variables) se ha llegado a utilizar más del 50 % de la misma. Para un análisis más exhaustivo del uso de la memoria en los dispositivos se recomienda ver el Anexo A de esta tesis.

#### 6.2.5.2. Mejoras pospuestas a futuras pruebas de concepto

Si bien la ejecución fue correcta y no hubo inconvenientes con la utilización de la memoria en la placa Arduino seleccionada, se hace necesario una revisión del código C++ para optimizar aún más la utilización de la memoria SRAM, ya que para ser utilizado en dispositivos con menor capacidad de almacenamiento de variables de programa se tornaría muy difícil su ejecución.

#### 6.2.5.3. Demostración

A continuación mostramos un diagrama de la ejecución del framework en este segundo ciclo de diseño. El mismo fue confeccionado automáticamente por el software que se encuentra desarrollado en el equipo servidor y que permite realizar una especie de bitácora de lo que se va ejecutando en la coreografía.

## 6.3. Tercer ciclo de Design-Science

En este tercer ciclo de Design-Science incorporaremos nuevos dispositivos, cuyas prestaciones son menores a las que poseen los equipos usados en el segundo ciclo. En concreto utilizaremos dispositivos con capacidad baja, cuya especificación se puede observar en la tabla 5.2. Específicamente utilizaremos una placa Arduino Nano V3. Esta placa ya posee muy pocas capacidades, tanto de memoria como de procesamiento.

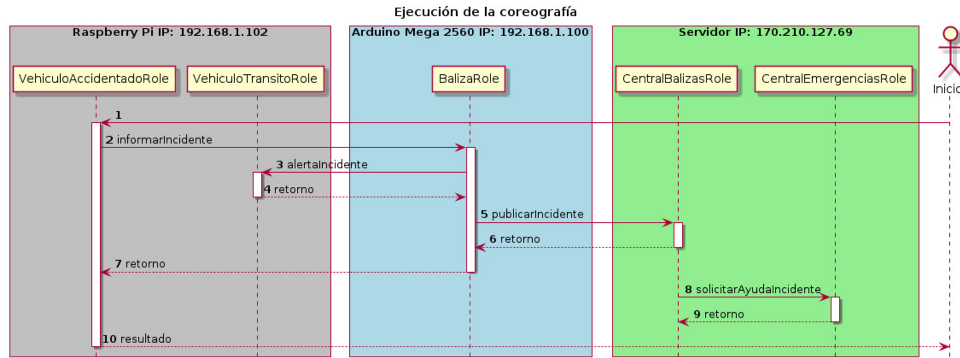


Figura 6.8: Diagrama de ejecución de la coreografía

### 6.3.1. Características utilizadas de WS-Choreography

En este ciclo se mantendrán las adaptaciones que se utilizaron durante el segundo ciclo, ya que la placa Arduino Nano posee la misma restricción la ejecución de distintos hilos.

### 6.3.2. Definición de la coreografía del escenario planteado

Se mantiene la misma definición de la coreografía que la planteada para el segundo ciclo de diseño.

### 6.3.3. Dispositivos considerados

A los dispositivos ya considerados en el primer ciclo de Design-Science, se anexa un dispositivo Arduino Nano V3. Este dispositivo cuenta con 32KB de memoria flash para almacenar el código ejecutable y con 2KB para el almacenamiento de variables de memoria, lo que representa tan solo el 12 % de la capacidad de memoria flash y el 25 % de la capacidad de almacenamiento de variables en memoria SRAM respecto de la placa Arduino Mega. La utilización de esta placa nos permite seguir avanzando en la utilización de dispositivos cada vez con menores capacidades. La figura 6.9 muestra una comparación de las capacidades de memoria de las dos placas Arduino, donde se puede apreciar la escasa capacidad de la placa Nano V3.

Para poder brindar conectividad a esta placa Arduino Nano se ha seguido con la utilización del módulo ESP8266-01, como se ha hecho durante el desarrollo del segundo ciclo de diseño.

En este tercer ciclo, la placa Arduino Nano reemplazará a la placa Arduino Mega en la interpretación del rol Baliza (BalizaRole), el resto de los dispositivos seguirán representando a los roles restantes.

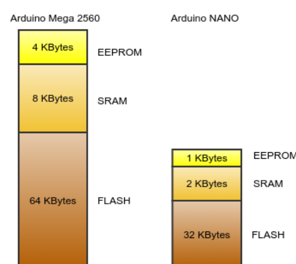


Figura 6.9: Comparación de memoria dispositivos Arduino

### 6.3.4. Implementación del framework de coordinación

En las distintas secciones que continúan, presentaremos las distintas decisiones que se llevaron adelante para poner en funcionamiento el tercer ciclo de diseño.

#### 6.3.4.1. Decisiones arquitectónicas

La arquitectura es la que se utilizó durante el segundo ciclo de Design-Science cuando se utilizó una placa Arduino Mega, reutilizando el código de implementación de un servidor Web con capacidad de procesar peticiones REST. Hacemos nuevamente hincapié en la posibilidad de realizar implementaciones parciales de la tecnología SOA a pesar de las importantes limitaciones de memoria disponibles en los dispositivos utilizados.

El esquema de conectividad se mantiene de la misma manera que en el segundo ciclo de diseño, con la salvedad que se reemplaza la placa Arduino Mega 2560 por la de Arduino Nano V3. La forma de conexionado de los dispositivos considerados, se puede apreciar en la Figura 6.10, la cual se presenta en forma de diagrama de despliegue de UML.

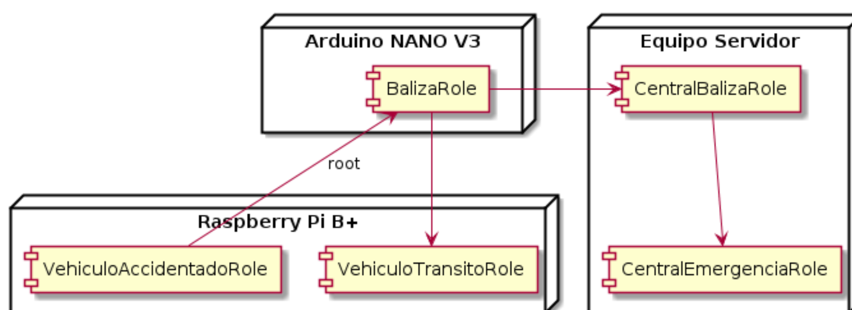


Figura 6.10: Diagrama de despliegue de dispositivos y roles

#### 6.3.4.2. Conectividad

El esquema de conectividad se mantiene de la misma manera que en el primer y segundo ciclo de Design-Science, con la salvedad que se incorpora dentro de la red a la placa Arduino Nano, reemplazando a la placa Arduino Mega utilizada en el segundo ciclo de diseño.

#### 6.3.4.3. Lenguajes

Los lenguajes utilizados siguen siendo PHP y C++, ya que la placa Arduino Nano tiene el mismo esquema de programación que la anterior.

#### 6.3.4.4. Servidores web

Se ha mantenido la misma arquitectura de servidores web que en la utilizada en segundo ciclo de Design-Science.

#### 6.3.4.5. Código

El código utilizado, si bien ha sido adaptado para poder ser ejecutado en este tipo de dispositivos de muy pocas prestaciones, las clases utilizadas tanto en PHP como en el lenguaje C++ son las mismas que se utilizaron tanto como para el primer y segundo ciclo de diseño, Figuras 6.6 y 6.7.

### 6.3.5. Evaluación

Luego de transcurrido el tercer ciclo de diseño, podemos expresar que la ejecución del mismo ha sido satisfactorio en el sentido que se ha encontrado un límite inferior respecto de la capacidad de los dispositivos ubicuos que pueden ser utilizados para ejecutar el framework desarrollado durante esta investigación. Específicamente la placa Arduino Nano V3 no ha sido capaz de ejecutar correctamente la ejecución de la coreografía tal como la hemos definido. A continuación en distintas subsecciones mostramos los detalles de los resultados obtenidos en esta ejecución.

Como conclusión de este tercer ciclo podemos resumir que los dispositivos deben contar con al menos 32kbytes de memoria flash para poder almacenar de manera correcta el código de ejecución del framework. A su vez estos deben contar con al menos 4kbytes de memoria SRAM para el almacenamiento de las variables y punteros de programa.

#### 6.3.5.1. Prueba del software

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 misma [2]. La ejecución bajo estas condiciones no es en absoluto aconsejable, por lo tanto, hubo que hacer adaptaciones para que finalmente ocupara el 81 % de la memoria flash disponible para programas.

Una vez solucionado el problema del tamaño del código objeto que se incorpora en la placa para que sea ejecutado, se tuvieron que hacer ajustes a la cantidad de memoria utilizada por las variables declaradas en el código fuente, las cuales se almacenan y consumen la memoria SRAM de la placa.

El código inicial del 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 solamente un 15 % máximo de memoria libre para el almacenamiento de variables locales y punteros. Realizando un análisis exhaustivo del código fuente con el fin de reducir el tamaño de memoria utilizada por variables globales, strings, etc., se logró hacer una reingeniería del código, optimizándolo de manera tal que tan solo demandaba el 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. 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.

En este punto se realizó un análisis en profundidad, tanto de manera teórica como práctica, para tener conocimiento sobre la cantidad mínima de memoria SRAM que se debía disponer para la ejecución del framework. El análisis en detalle puede ser consultado en el Anexo I, donde se realiza un pormenorizado estudio del código, para poder llegar a una conclusión de cantidad de memoria mínima necesaria. Aquí, solamente nos abocaremos a presentar un resumen de este estudio, haciendo hincapié en los puntos más importantes.

Para el análisis de la memoria consumida nos centraremos en las Figuras 6.6 y 6.7, que son las que muestran a través de diagramas de UML las clases implementadas en código C++.

Al iniciarse la ejecución del framework recordemos que ya se encontraban utilizados 1260 bytes de los 2048 disponibles. La clase connection realiza los ajustes necesarios en el módulo ESP8266 para poder recibir las peticiones de servicios web, para ello se invoca el método inicializarServer, el cual se encarga de realizar las configuraciones iniciales y de instanciar la clase RESTWebServer. Del análisis del código, podemos decir que se necesitan 43 bytes para la ejecución de este método, de los cuales quedan residentes 36 bytes. Por lo que la memoria ocupada pasaría a ser de 1296 bytes. A partir de este momento la placa Arduino junto con el módulo ESP8266 quedan a la espera de peticiones, las cuales al momento de arribar, son tratadas por el

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

método procesarPeticion de la clase connection. Este método es uno de los que más consume y hace que la memoria disponible llegue a su punto límite. Del análisis surge que este método por sí solo consume aproximadamente unos 540 bytes adicionales a los ya utilizados (1296), es decir, estaríamos en los 1836 bytes. A esto se le debe sumar la fragmentación que se produce en memoria por la utilización de los punteros y de los distintos registros utilizados para el control de la ejecución del programa. A su vez el framework para la ejecución y procesamiento de las peticiones de servicios, está necesitando un total de aproximadamente 800 bytes más, dato que surge del análisis teórico del código fuente, de las clases choreographyService, RESTWebServer y de la clase propia del manejo del servicio.

En los listados 6.2 y 6.3 podemos observar dos listados que nos muestran por un lado los resultados de la 1ra. compilación en un entorno Arduino y en segundo lugar la 2da. compilación en estos entornos.

```
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 6.2: 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 6.3: 2da. Compilación en entorno Arduino

#### 6.3.5.2. Mejoras pospuestas a futuras pruebas de concepto

Si bien el motor de ejecución de coreografías no puede ser ejecutado en este tipo de dispositivos, no invalida la posibilidad de que utilicemos un framework ad-hoc para este tipo de placas. Por lo tanto, la posibilidad de utilizar este tipo de dispositivos o incluso con menos prestaciones es posible, a través de la programación ad-hoc del rol que debe jugar este elemento dentro del conjunto de los otros dispositivos para poder llevar adelante de manera exitosa la ejecución de una coreografía.

#### 6.3.5.3. Demostración

La figura 6.8 muestra un diagrama de la ejecución del framework en este tercer ciclo de diseño. El mismo fue confeccionado automáticamente por el software que se encuentra desarrollado en el equipo servidor y que permite realizar una especie de bitácora de lo que se va ejecutando en la coreografía.

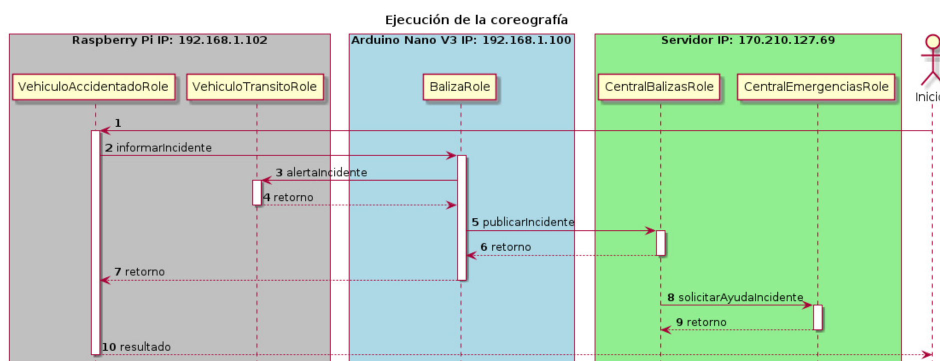


Figura 6.11: Diagrama de ejecución de la coreografía

## 6.4. Cuarto ciclo de Design-Science

En este cuarto ciclo de Design-Science incorporaremos una de las características más presente en este tipo de dispositivos ubicuos, que es su escaso nivel de batería lo cual puede provocar que el mismo desaparezca al momento de ser invocado. O incluso, pueda truncarse la ejecución del servicio que brinda el mismo. A continuación haremos una descripción de las características de esta ejecución, tal como se ha presentado en las anteriores.

### 6.4.1. Características utilizadas de WS-Choreography

En este ciclo se mantendrán las adaptaciones que se utilizaron durante el tercer ciclo. Se ha trabajado también con los tiempos de respuesta que se establecen en la declaración de la coreografía.

### 6.4.2. Definición de la coreografía del escenario planteado

Se mantiene la misma definición de la coreografía que la planteada para los anteriores ciclos de diseño.

### 6.4.3. Dispositivos considerados

Mantendremos los dispositivos utilizados durante el segundo ciclo de diseño, donde utilizamos una placa Arduino Mega, ya que la utilización de una placa Arduino Nano no fue posible por su escasa capacidad de almacenamiento en memoria SRAM.

### 6.4.4. Implementación del framework de coordinación

En las distintas secciones que continúan, presentaremos las distintas decisiones que se llevaron adelante para poner en funcionamiento este cuarto

ciclo de Design-Science.

#### 6.4.4.1. Decisiones arquitectónicas

La arquitectura es la que se utilizó durante el segundo ciclo de Design-Science cuando se utilizó una placa Arduino Mega, reutilizando el código de implementación de un servidor Web con capacidad de procesar peticiones REST.

El esquema de conectividad se mantiene de la misma manera que en el segundo ciclo de diseño, se puede apreciar en la Figura 6.10.

#### 6.4.4.2. Conectividad

El esquema de conectividad se mantiene de la misma manera que en el segundo ciclo de Design-Science, excepto porque en esta prueba se ha cambiado el conexionado del módulo ESP8266-01 de la placa Arduino Mega. Anteriormente el módulo estaba conectado a un puerto Serial1 de la placa Arduino, pero este tipo de puertos no permite el manejo de interrupciones, por lo que debió ser cambiado al puerto Serial estándar de la placa, ya que es el único que soporta el manejo de interrupciones.

Manejar interrupciones resulta importante para este ciclo, ya que la utilización de una interrupción es lo único que despierta a Arduino de un estado de ahorro de energía profundo como el que se utilizó en este caso. Al recibir una petición web sobre este puerto, Arduino despierta, repone todas sus funciones a modo normal y procesa la petición.

#### 6.4.4.3. Lenguajes

Se mantiene la misma estructura y utilización de lenguajes que en el segundo ciclo de diseño. Debido a que en este ciclo es necesario controlar los niveles de batería disponibles se utilizó una librería para tal fin, la cual puede encontrarse en `https://github.com/rlogiacco/BatterySense`.

#### 6.4.4.4. Servidores web

Se ha mantenido la misma arquitectura de servidores web que en la utilizada en el segundo ciclo.

#### 6.4.4.5. Código

El código utilizado para programar tanto para la placa Arduino Mega, como en la parte de PHP, es el mismo que el utilizado en el segundo ciclo de diseño. La única diferencia en el código de C++ para la placa Arduino Mega, es que se utilizó la librería de manejo de batería, además de la utilización de las librerías del fabricante del chip (ATMEL) para poder hacer que la placa

entre en un modo de ahorro de energía, de manera tal que su consumo sea mínimo. De todas formas las clases mantienen el mismo esquema que el de la Figura 6.6.

Con la librería de manejo de batería es posible saber el estado de la batería (tiempo disponible de acuerdo al consumo que se tiene en la placa), por lo que ante un estado de baja batería puede realizar acciones de ser relevado por otro dispositivo o hacer una desaparición controlada del mismo. La librería de manejo AVR [3], permite enviar a un modo de ahorro de energía, el máximo posible con este tipo de placas, para que en caso de recibir una petición web de participación del dispositivo en la coreografía, la misma despierte al Arduino y pueda procesar la llamada. Una vez finalizada la llamada y realizadas todas las tareas correspondientes a la coreografía, vuelve a su estado de ahorro de energía.

### 6.4.5. Evaluación

En este cuarto ciclo de Design-Science nos hemos encontrado con algunos inconvenientes para implementar el ahorro de energía en los dispositivos Arduino. Estos inconvenientes han podido ser subsanados para poder lograr finalmente con una ejecución exitosa, permitiendo que se ahorre la mayor cantidad de batería posible en los momentos en que el dispositivo no se encuentra realizando tareas concernientes a la ejecución de la coreografía.

#### 6.4.5.1. Prueba del software

Para poder optimizar la utilización de la batería de estos dispositivos, es indispensable poder lograr que el dispositivo ahorre la mayor cantidad de batería posible. Mientras el dispositivo no necesita realizar alguna tarea relacionada con la ejecución de la coreografía, se le debe dar la orden de ahorro de energía, tarea que realiza la librería AVR. En dicha librería se le indica a la placa Arduino que desactive todas sus funciones y quede latente a la espera de algún evento que la active. Para poder activarlo, se utilizan normalmente las interrupciones, que son indicadores que algo externo está necesitando su procesamiento. En este caso particular, recibir una petición de participación en la coreografía. Aquí es donde nos encontramos con el problema de que el módulo ESP8266 no despertaba a la placa Arduino, ya que el puerto Serial1 (donde se encontraba conectado el módulo wifi) no permite interrupciones. Una de las alternativas era conectar el módulo wifi emulando un puerto serie en alguno de los puertos digitales disponibles, que a su vez soportan el manejo de interrupciones. Tampoco fue posible, ya que se perdían datos al momento de procesar la petición cuando se despertaba la placa.

[3] Familia de microcontroladores con tecnología RISC del fabricante Atmel

Por lo tanto se optó por conectar el módulo wifi al puerto Serial que sí soporta interrupciones sin pérdida de datos.

Una vez que se pudo controlar el despertado de la placa a través de la utilización de las interrupciones, se procedió a la utilización de la librería que permite saber los niveles de batería disponibles.

#### 6.4.5.2. Mejoras pospuestas a futuras pruebas de concepto

Como hemos expuesto, la ejecución fue exitosa y permitió que la ejecución de la coreografía active al dispositivo Arduino para su participación. Por lo tanto, a través de este ciclo de diseño se pudo extender la autonomía del dispositivo ahorrando energía cuando no es necesario su utilización, y además poder realizar cálculos de cantidad de tiempo de autonomía disponible. Esto permite que en futuros ciclos se puedan implementar mecanismos de apagado controlado, invocando a otros dispositivos similares que puedan reemplazarlo.

#### 6.4.5.3. Demostración

El diagrama de ejecución es el mismo que el del segundo ciclo de diseño, ya que los dispositivos que se utilizaron son los mismos y respetando la misma conectividad entre ellos.

## 6.5. Quinto ciclo de Design-Science

En este quinto ciclo de Design-Science trabajaremos con el manejo de una transacción distribuida a lo largo de la ejecución de la coreografía. Si bien el manejo de una transacción no es algo inherente únicamente a los dispositivos ubicuos, nos resultó interesante poder mostrar su implementación en un ambiente pervasivo y trabajando con servicios REST.

### 6.5.1. Características utilizadas de WS-Choreography

Durante este ciclo de diseño se mantienen implementadas las mismas características que en los ciclos anteriores.

### 6.5.2. Definición de la coreografía del escenario planteado

Se mantiene la misma definición de la coreografía que la planteada para los anteriores ciclos de diseño.

El único cambio realizado, que no es parte de la coreografía, es el agregado del manejo de una transacción, que se ejecuta cuando el rol CentralBalizas recibe una petición de generación de un incidente informado desde alguna de las balizas. A partir de este momento se comienza con la transacción, que

se compone de dos dispositivos participantes: la Central de Emergencias y la Baliza. En el primer caso se debe registrar el evento en alguna base de datos y en el segundo se le informa a la Baliza que debe encenderse para alertar a los eventuales vehículos que transitan puedan estar alertas.

### 6.5.3. Dispositivos considerados

Mantendremos los dispositivos utilizados durante el cuarto ciclo.

### 6.5.4. Implementación del framework de coordinación

En las distintas secciones que continúan, presentaremos las distintas decisiones que se llevaron adelante para poner en funcionamiento el quinto ciclo de diseño.

#### 6.5.4.1. Decisiones arquitectónicas

La arquitectura es la que se utilizó durante el segundo y cuarto ciclo de diseño, cuando se utilizó una placa Arduino Mega, reutilizando el código de implementación de un servidor Web con capacidad de procesar peticiones REST. Se le agregó la posibilidad de una transacción, la cual se maneja en líneas generales de acuerdo al gráfico que se muestra en la siguiente figura 6.12.

En este caso se muestra la forma en que una transacción distribuida se realiza a través del método *TryConfirmCancel*, propuesto por (Pardon y Pautasso, 2014). En este diagrama podemos observar que se debe crear un *Coordinador de la Transacción* que es el elemento que se encargará de coordinar que la transacción sea ejecutada en su totalidad o en su defecto, la misma sea cancelada (por algún problema determinado). Este elemento coordinador es un servicio REST que se encarga de esta tarea. Este servicio debe ser activado con las direcciones de los recursos que formarán parte de la transacción, que también son servicios REST. Este tipo de soluciones al manejo de transacciones distribuidas debe manejar recursos (URIs) de forma dinámica, ya que las transacciones hacen uso de estos recursos dinámicos para poder llevar adelante el trabajo.

Nuevamente podemos observar que de manera acotada hemos implementado una transacción sobre dispositivos ubicuos en ambientes pervasivos sin perder las características de una aplicación SOA.

#### 6.5.4.2. Conectividad

El esquema de conectividad se mantiene de la misma manera que en el cuarto ciclo de diseño.

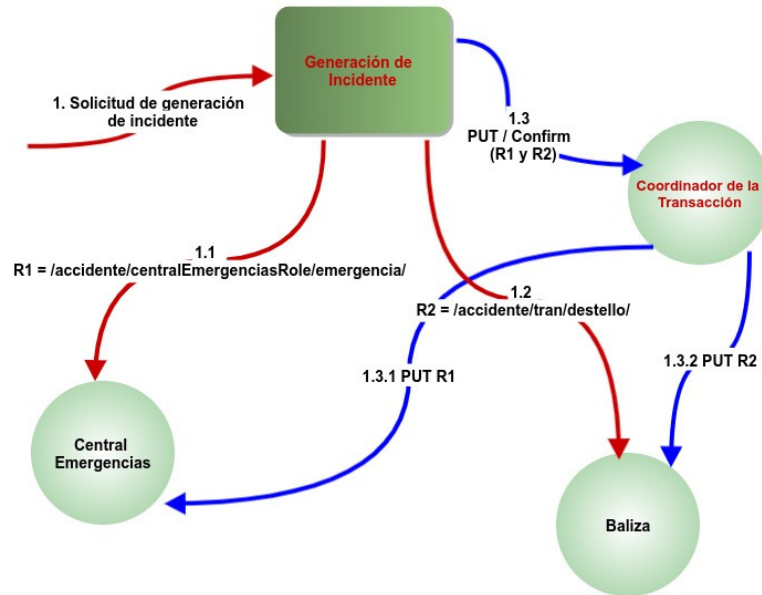


Figura 6.12: Diagrama de manejo de transacciones en REST

#### 6.5.4.3. Lenguajes

Se mantiene la misma estructura y utilización de lenguajes que en el cuarto ciclo de diseño.

#### 6.5.4.4. Servidores web

Se ha mantenido la misma arquitectura de servidores web que en la utilizada en el segundo y cuarto ciclo de diseño.

#### 6.5.4.5. Código

Se debe hacer una reingeniería de las clases involucradas en C++, para que las mismas soporten el manejo de transacciones. Si bien se deberán realizar modificaciones y reorganizar las clases existentes en C++, el código principal sigue siendo el mismo de los anteriores ciclos de diseño.

Como resultado de esta reorganización se obtuvo una jerarquía de clases mucho más robusta y que puede ser fácilmente extendida para soportar otro tipo de transacciones y no solamente la utilizada en este caso particular.

A continuación se puede observar un diagrama de clases en C++ que da soporte a este nuevo ciclo 6.13.

En el código PHP también se tuvieron que realizar adaptaciones a las

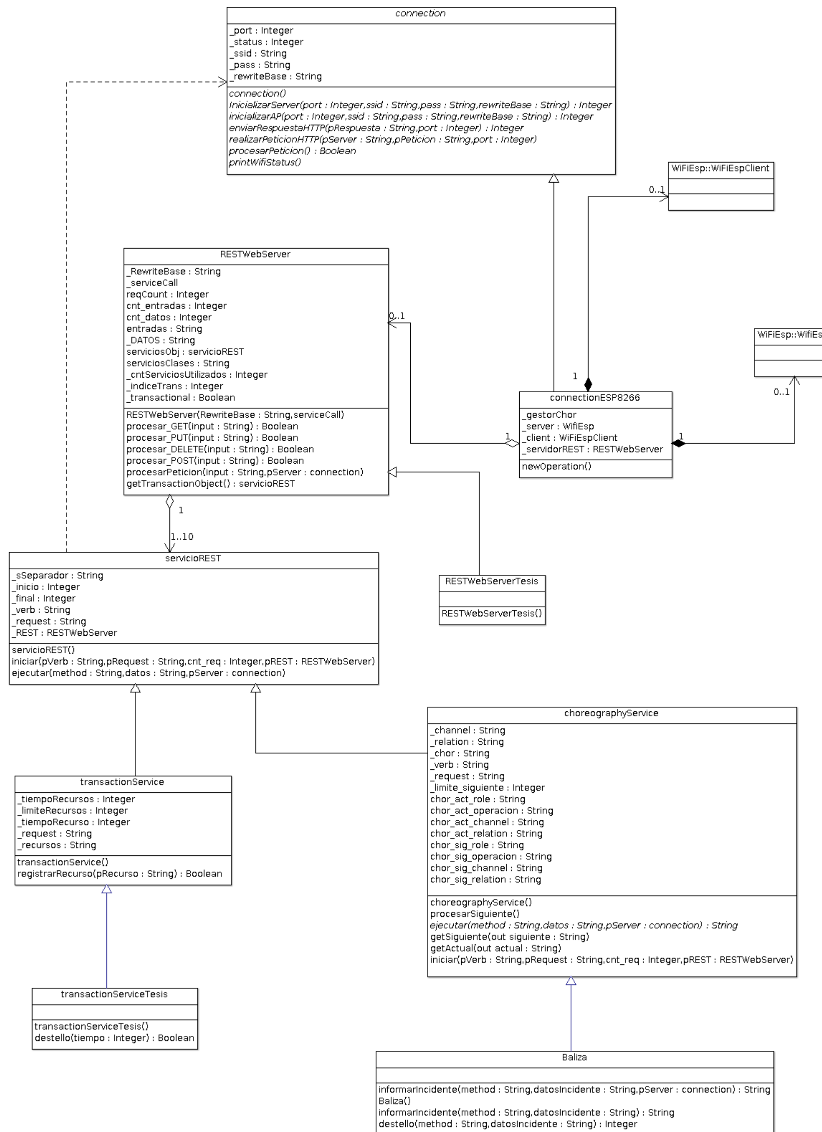


Figura 6.13: Diagrama de clases en C++ para el manejo de transacciones en REST

clases existentes para poder dar soporte al manejo de transacciones. En este lenguaje es que se codificó el servicio Coordinador de la Transacción. Esta selección se hizo debido a que este lenguaje es mucho más robusto para este tipo de tareas y además los equipos donde se ejecuta este código tienen mayores prestaciones, lo cual brindaba mayor seguridad a los fines de este trabajo de investigación.

A continuación se puede observar un diagrama de clases en PHP que da soporte a este nuevo ciclo 6.14.

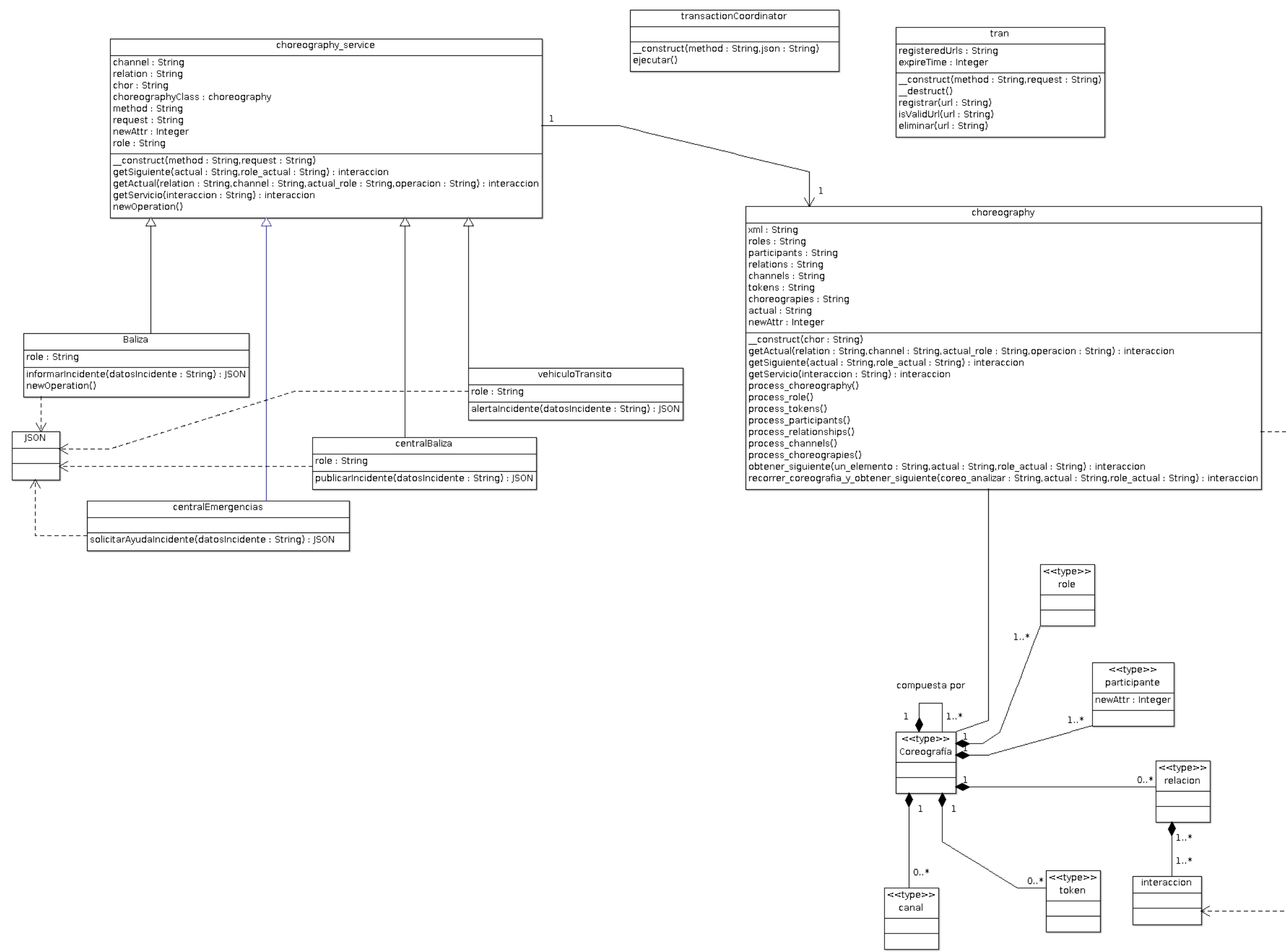


Figura 6.14: Diagrama de clases en PHP para el manejo de transacciones en REST

### 6.5.5. Evaluación

En este quinto ciclo de Design-Science nos hemos encontrado con algunos inconvenientes para implementar la transacción distribuida dentro de la ejecución de la coreografía. Principalmente el problema se suscitó en que la placa Arduino junto con el módulo ESP8266-01 que le brinda conectividad a la placa tiene tiempos de procesamiento más lentos que el resto de los dispositivos incluidos en la ejecución. Por esta limitación, propia de los dispositivos ubicuos, es que se tuvo que poner un retardo en las peticiones a la placa Arduino desde el código PHP, para que de esta manera el módulo wifi de la placa pudiese procesar correctamente todos los pedidos que recibía. Recordemos que en este caso de la ejecución de una transacción recibe más peticiones de ejecución de servicios que en los anteriores ciclos de diseño.

Una vez subsanados estos inconvenientes no hubo más inconvenientes y podemos afirmar que es totalmente posible realizar transacciones en ambientes pervasivos.

#### 6.5.5.1. Prueba del software

Durante el armado de la prueba se tuvo que realizar una reorganización de las clases en C++ utilizadas en Arduino para que den soporte al manejo de la transacción y sobre todo al manejo de recursos dinámicos para tal fin. Si bien se hizo una recodificación de algunas partes y clases, las mismas mantienen la estructura de los ciclos de diseño anteriores, es decir, se realizó una mejora en las mismas logrando que sean mucho más eficientes y extensibles. Del lado de PHP se codificaron algunas clases totalmente nuevas para ser anexadas a las ya existentes, es decir, se hizo una extensión del código para poder dar soporte al manejo de transacciones.

#### 6.5.5.2. Mejoras pospuestas a futuras pruebas de concepto

Como hemos expuesto, la ejecución fue exitosa y permitió que la ejecución de la coreografía maneje una transacción distribuida a lo largo de los dispositivos.

Se deberá en próximos ciclos implementar mecanismos para que en caso de que se produzca un error o inconveniente en la ejecución de la transacción se marque la coreografía como no disponible para su ejecución.

#### 6.5.5.3. Demostración

El diagrama de ejecución es el mismo que el del segundo ciclo de diseño, ya que los dispositivos que se utilizaron son los mismos y respetando la misma conectividad entre ellos.

## 6.6. Sexto ciclo de Design-Science

Este sexto ciclo de Disign-Science se enfoca en las desapariciones (controladas o no controladas) de los dispositivos. El comportamiento habitual en este tipo de casos es la culminación de la coreografía o bien la decisión de no hacer ninguna tarea específica ante este evento. Si bien el manejo de timeouts[4] y la implementación del mismo dentro de las coreografías no resulta innovador, ya que en el ámbito de los servicios web ha sido abordado desde sus comienzos, resulta interesante abordarlo en ambientes pervasivos y en especial con dispositivos ubicuos, ya que en muchos casos no son manejados de la manera correcta ni se tienen en cuenta a los fines prácticos, debido a que en muchos lenguajes de programación pueden ser atrapados como un error y tratarlo como tal. Sin embargo en el manejo de coreografías resulta interesante porque estos pueden producirse por problemas de desapariciones controladas o no controladas de los dispositivos ubicuos fundamentalmente. A partir de ello en este ciclo se abordará de dos maneras posibles: se finaliza la coreografía de manera inmediata, marcándola como no utilizable para futuras llamadas y como segunda opción la posibilidad de llevar adelante alternativas que puedan atrapar de manera preventiva estas desapariciones.

Por lo tanto, debemos, antes de comenzar con la realización de la prueba de concepto propiamente dicha, hacer una descripción de qué es lo que se entiende por desapariciones de los dispositivos, así como poder establecer un marco de referencia para tratarlas, donde es necesario tener una lista de problemas que pueden suceder que hagan desaparecer el dispositivo y las alternativas que pueden llevarse adelante para mitigar estos problemas en la medida de lo posible.

Es importante aclarar que en este estudio no se pretende encontrar una solución única y estándar al problema de las desapariciones, sino todo lo contrario, se espera poder tener un espectro de qué significa y qué puede acarrear un problema de este estilo en la ejecución de coreografías con dispositivos ubicuos. Además, cada coreografía representa la solución a un problema de negocio específico, el cual puede por definición no poder soportar algunas de las alternativas que se puedan plantear ante estos acontecimientos. También se debe dejar en claro que esta no es una lista exhaustiva ni de los problemas de desapariciones que se pueden producir ni de las alternativas planteadas en cada caso de desaparición, solo intenta plantear el problema y las aristas que puede conllevar su solución.

La tabla 6.1 muestra un resumen de lo planteado previamente.

[4]Tiempo que se espera antes de dar por terminada una petición web

| **Motivos de desaparición** | Descripción | Estrategias de solución |
|---|---|---|
| *Falta de batería* | Esta desaparición es CONTROLADA, ya que desde el dispositivo se puede testear el nivel de batería restante para poder saber qué tiempo de utilidad le queda.<br>Si bien la falta total de batería provoca una desconexión, la misma se puede predecir y tomar alternativas para la solución. | - Agregar en el mensaje de respuesta y de peticiones que realiza el dispositivo, la dirección del dispositivo que lo relevará. Esto no necesita cambios en la especificación.<br>- Realizar un agregado en la especificación del WS-CDL, donde cada dispositivo pueda definir quién lo reemplaza en caso de una desaparición.<br>- Enviar un mensaje de aviso al resto de los dispositivos de la especificación para que estén informados de que la coreografía puede sufrir problemas. |
| *Falta de conectividad* | Esta desaparición se la cataloga como NO CONTROLADA ya que no existe un dato fehaciente de que se pueda generar un corte de conectividad.<br>Este corte puede ser incluso un problema de la red a la cual se encuentra conectado el dispositivo y no una falla del propio elemento. | - Mantener la posibilidad de que el dispositivo esté clonado. En este caso se debería modificar la especificación WS-CDL a tal fin.<br>- En cada mensaje que envía (petición o respuesta) el dispositivo enviar como agregado la dirección del dispositivo que lo relevará. |
| *Rotura del dispositivo* | Esta desaparición se la cataloga como NO CONTROLADA ya que no se puede predecir, por lo que cuando sucede no hay manera de controlar la ausencia del dispositivo. | - Mantener la posibilidad de que el dispositivo esté clonado. En este caso se debería modificar la especificación WS-CDL a tal fin.<br>- En cada mensaje que envía (petición o respuesta) el dispositivo enviar como agregado la dirección del dispositivo que lo relevará. |
| *Timeouts* | En caso de producirse tiempos de espera por encima de los habituales o los preestablecidos en la coreografía. | - Contactar dispositivos alternativos brindados o especificados en la definición de la coreografía. |

Tabla 6.1: Alternativas de solución para desapariciones de los dispositivos

Existen otras alternativas de solución que son generales a cualquier problema que se pueda plantear y produzca una desaparición FORZOSA o NO FORZOSA, las cuales mencionamos a continuación:

- Utilizar un servicio de cloud computing como es el caso de AWS-IoT de Amazon, donde se puede guardar el estado de los dispositivos, así como una copia del mismo y simular que siempre se encuentra activo.
- Utilizar un sistema centralizador de la coreografía, donde los dispositivos que pueden ser parte de la misma se registran con una descripción del dispositivo que pueden clonar o representar. Este mecanismo centralizador puede ser el encargado de realizar búsquedas de nuevos dispositivos y el diálogo con ellos para ser incluidos dentro del catálogo de posibilidades.
- Trabajar con mecanismos de mensajes diferidos, donde el dispositivo no se encuentra obligado a responder inmediatamente.

Como podemos apreciar en esta tabla se plantean alternativas de posibles soluciones al problema planteado en la primer columna, pero siempre haciendo la salvedad de que la misma puede no ser aplicable al problema de negocio que se está dando solución con una coreografía de servicios y/o dispositivos.

En esta prueba de concepto en particular vamos a implementar la desaparición NO FORZOSA de cuando el dispositivo tiene una *Falta de batería*, y con la utilización de un dispositivo alternativo el cual se encuentra especificado dentro de los roles definidos en el lenguaje de especificación de coreografías WS-CDL. Para ello se hará una extensión del lenguaje específicamente para poder suplir esta característica que es específica de los sistemas ubicuos.

En lo que sigue se realizará una descripción y análisis de las distintas alternativas posibles para la implementación en esta prueba de concepto de la desaparición y su correspondiente solución, basándonos en la tabla 6.1.

### 6.6.1. Características utilizadas de WS-Choreography

Para poder llevar adelante o enfrentar los problemas de desapariciones de los dispositivos en el contexto de la ejecución de una coreografía, indudablemente necesitamos hacer cambios en la manera en que se ejecutan las mismas para dar soporte a estas características de los dispositivos ubicuos. Existen varias alternativas para abordar, basándonos en la posibilidad de contar con un dispositivo clon que permita llevar adelante o prestar los servicios del dispositivo que eventualmente puede desaparecer. La discusión de otras alternativas será abordado en capítulos siguientes, en este momento de la resolución se ha seleccionado esta posibilidad porque es la que más se

adapta para esta prueba de concepto. Una vez determinado que se utilizará un dispositivo alternativo que funciones como un clon de otro dispositivo participante de la coreografía, analizaremos a continuación las distintas alternativas disponibles para su implementación.

- Envío de mensaje: en esta alternativa, el dispositivo envía como respuesta o petición a otros participantes o roles de la coreografía la dirección donde se encuentra el dispositivo que va a reemplazarlo en caso de que el mismo desaparezca. Si bien esta alternativa no necesita cambios en la especificación WS- CDL (solamente bastaría con definir algún elemento **information** adicional para dar soporte al envío de la dirección del nuevo dispositivo), existen inconvenientes. El mayor inconveniente en este caso es que el dispositivo desaparezca antes que el mismo comience a participar de la coreografía, este hecho haría que el dispositivo clon no sea visualizado por el resto de los integrantes de la especificación, y por consiguiente, que la coreografía falle en su conjunto.

- Definir el dispositivo clon dentro de la especificación: en esta opción se declara dentro de la especificación de la coreografía un dispositivo que es el que tomaría el control del dispositivo que desaparece, de manera tal, que de antemano cada participante pueda saber qué dispositivos actúan como backup de otros en caso de una desaparición. Existen varias formas de realizar esta definición:

    1. Definirlo como un rol: en este caso se debería hacer una extensión del lenguaje de especificación adicionando atributos de si ese rol es el principal o no.
    2. Definirlo como un participante: estaríamos en el mismo caso que el punto anterior, o bien deberíamos definir un nuevo tipo de participante, lo cual no estaría reflejando correctamente lo que sucede en la coreografía ya que no es un participante nuevo sino uno clonado.
    3. Definir un nuevo elemento **Clones**: esto implica que se haga un agregado a la especificación WS-CDL, donde se incluye una sección para la definición de los dispositivos que actuarán como reemplazo de los originales o verdaderos roles dentro de la coreografía. En este caso habrá que definir a qué rol y participante estaría reemplazando y en caso de ser necesario, otros atributos que lo definan de la mejor manera posible como sería la interface que lo implementa, el tipo (a demanda o permanente), etc.

- Reemplazo en-línea: a través de esta alternativa estaríamos reemplazando el dispositivo en tiempo de ejecución de una manera totalmente transparente para la coreografía y la especificación de la misma.

Podrían utilizarse los servicios por ejemplo de AWS IoT que brinda Amazon, o bien otros alternativos o bien alguno implementado ad-hoc para tal caso. Esta posibilidad es más compleja de realizar y en algunos casos podría tener costos económicos de implementación, aunque es preciso mencionar que como tiene como ventaja que no se deben hacer cambios en la especificación.

Si bien pueden existir otras alternativas aquí hemos tomado la decisión de enumerar aquellas que consideramos las más importantes para su discusión en el dominio de conocimiento de este trabajo de tesis y de la prueba de concepto en particular con la que estamos trabajando. Para la implementación se ha seleccionado la alternativa de definir un nuevo elemento denominado **Clones** por sobre las otras alternativas, ya que la misma representa ventajas y sensibles mejoras respecto de las otras.

En cualquiera de las opciones que se plantearon previamente, debe existir un compromiso de parte del dispositivo que puede ser reemplazado por su propia desaparición, que mantendrá actualizado con todos sus estados al dispositivo clonador, de modo tal que la coreografía no se vea afectada por algún estado incorrecto. Para ello, el dispositivo deberá implementar los mecanismos para llevar adelante dicha correspondencia de los estados con el dispositivo clonado.

El nuevo elemento a incorporar en la especificación es el de clones y lo definimos de la siguiente manera:

```
<cloneType name="NCName"
           interface="QName"?
           type="on-demand"|"permanent">
           <roleType typeRef="QName"/> +
</cloneType>
```

El atributo *interface* se encuentra definido como opcional, identificando la inteerface WSDL.

El atributo *name* es utilizado para especificar un nombre distinto para cada elemento *cloneType* declarado dentro de un paquete coreografía.

El atributo *type* es utilizado para especificar qué tipo de utilización se va a hacer del elemento *clonType*, los valores posibles son:

1. "on-demand" - El dispositivo que actuará como clon solamente será utilizado en un caso específico de una relación en particular.

2. "permanent" - El dispositivo una vez que se activa como reemplazante continúa siendo visible para el resto de la coreografía. Este es el comportamiento por defecto.

Dentro del elemento *cloneType* uno o más elementos *roleType* identifican los roleTypes que DEBEN ser implementados por este cloneType. Cada roleType es especificado por un atributo typeRef del elemento roleType definido previamente en la declaración de la coreografía. El valor de QName del atributo typeRef del elemento roleType tiene que hacer referencia el nombre de un roleType declarado previamente.

En nuestro caso, vamos a utilizar este elemento cloneType para definir un dispositivo que reemplazará a una baliza en caso de que la misma tenga problemas de batería. Para ello definiremos el siguiente cloneType:

```
<cloneType name="BalizaClone"
           type="permanent">
           <roleType typeRef="BalizaRole"/>
</cloneType>
```

Donde al nuevo elemento lo denominamos *BalizaClone* y reemplazará al role *BalizaRole* en caso de desaparición del mismo.

### 6.6.2. Definición de la coreografía del escenario planteado

Se mantiene la misma definición de la coreografía que la planteada para los ciclos anteriores.

### 6.6.3. Dispositivos considerados

Mantendremos los dispositivos utilizados durante el segundo y cuarto ciclo de diseño. Para este caso también agregaremos un dispositivo clonador de la Baliza, por lo que se utilizará otra placa Arduino Mega 2560. Este nuevo dispositivo suplantará al dispositivo que implementa el rol de Baliza, el cual es una placa con las mismas características.

### 6.6.4. Implementación del framework de coordinación

En las distintas secciones que continúan, presentaremos las distintas decisiones que se llevaron adelante para poner en funcionamiento el sexto ciclo.

#### 6.6.4.1. Decisiones arquitectónicas

La arquitectura es la que se utilizó en la segunda prueba de concepto cuando se utilizó una placa Arduino Mega, reutilizando el código de implementación de un servidor Web con capacidad de procesar peticiones REST.

#### 6.6.4.2. Conectividad

El esquema de conectividad se mantiene de la misma manera que en el cuarto ciclo de diseño, donde se incorporó el manejo de interrupciones y de administración de niveles de batería.

#### 6.6.4.3. Lenguajes

Se mantiene la misma estructura y utilización de lenguajes que en el segundo ciclo.

#### 6.6.4.4. Servidores web

Se ha mantenido la misma arquitectura de servidores web que en la utilizada en el segundo y cuarto ciclo de diseño.

#### 6.6.4.5. Código

Se ha modificado el código que manejaba los timeouts incorporados en el quinto ciclo, para dar soporte a la definición de dispositivos clonados. De esta manera en caso de producirse un timeout en las llamadas hacia un dispositivo, se verifica la existencia de un dispositivo clonador y si existe, se reemplazan las llamadas hacia este nuevo dispositivo.

Este cambio se ha producido tanto en las clases de C++ para la programación de las placas Arduino, como también en PHP para el manejo de los servicios en dispositivos con mayores capacidades.

### 6.6.5. Evaluación

En este último ciclo de Design-Science hemos trabajado sobre las desapariciones de dispositivos. Para poder dar soporte a las desapariciones de los dispositivos dentro de la ejecución de la coreografía hemos tenido que realizar agregados a la especificación del lenguaje de especificación de coreografías WS-CDL, la cual fue expresada en 6.6.1.

La evaluación que hacemos es altamente positiva, ya que se pudo trabajar con las desapariciones de una manera transparente para la ejecución de la coreografía, reemplazando aquellos dispositivos que han desaparecido por alguna razón. En este caso particular se ha forzado la desaparición mediante el anexado de un código que produce un retardo mayor al timeout establecido para la respuesta de dicho dispositivo. Podemos concluir que es totalmente posible trabajar con dispositivos que actúan como reemplazo de aquellos que pueden tener desapariciones, y en caso de que estos también fallen, atrapar el error y cancelar la ejecución de la coreografía en su conjunto.

#### 6.6.5.1. Prueba del software

Tal como mencionamos, hemos tenido que realizar ajustes en el código de la ejecución de la coreografía para poder inducir el evento de una desaparición de un dispositivo. Este ajuste en el código no es más que un simple comando de retardo o aletargamiento en la ejecución. En el caso específico de PHP existe un comando a tal fin, donde se le especifica el tiempo deseado de retardo. Por otro lado en C++ simplemente se codificó un bucle con determinado tiempo de ejecución para tal fin.

Se puede apreciar que el código obtenido para la implementación del framework de ejecución de coreografías es robusto ya que permite adaptaciones sin mayores inconvenientes.

#### 6.6.5.2. Mejoras pospuestas a futuras pruebas de concepto

Queda como pendiente para futuros ciclos de diseño implementar otro tipo de soluciones a las desapariciones, como puede ser el caso de buscar en tiempo real un reemplazo al dispositivo que desapareció.

#### 6.6.5.3. Demostración

El diagrama 6.15 muestra un diagrama de secuencia de la ejecución de la coreografía utilizando un dispositivo de reemplazo para el rol Baliza, que es el que se ha diagramado que desaparezca.

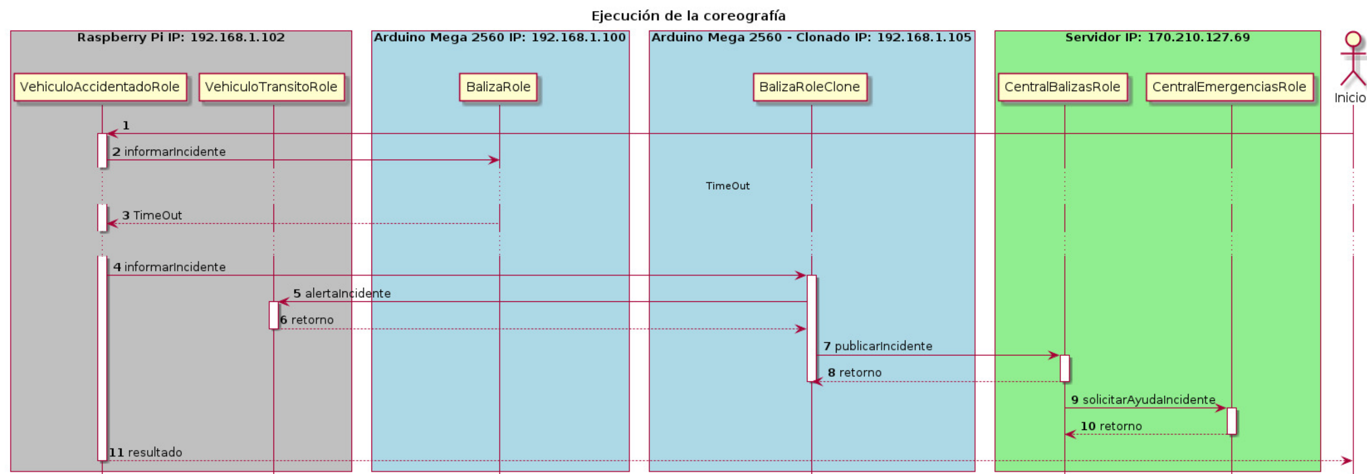


Figura 6.15: Diagrama de ejecución de la coreografía

# Capítulo 7

# Discusión

En este capítulo realizaremos un análisis de los resultados de investigación obtenidos en base a los objetivos de investigación, los cuales se describen en el capítulo de Planteamiento del problema y Objetivos de Investigación 3.

A continuación, indicaremos algunas limitaciones de la investigación realizada, tales como la imposibilidad de implementar una conectividad entre dispositivos distinta a la Wifi 802.11, o la falta de implementación de las especificaciones de seguridad como es el caso de WS-Security y WS-Reliability. Nótese que, en todos los casos, las limitaciones se deben a restricciones temporales o de recursos, y no a limitaciones de la investigación realizada..

Finalmente, describiremos las relaciones existentes entre la presente investigación y otras tecnologías/iniciativas existentes, tales como IoT, Microservicios y ciertos frameworks de dispositivos ubicuos como es el caso de Gaia, Aura o Amigo.

## 7.1. Cumplimiento de los Objetivos de investigación

Todos los objetivos de investigación, dentro de las restricciones temporales y de recursos propios de esta investigación, han sido alcanzados. El producto o framework obtenido funciona de manera correcta y dando soporte a las características propias de los dispositivos ubicuos. La solución planteada es simple, interoperable y extensible. Hemos logrado trasladar, a un espacio o entorno donde las soluciones realizadas eran ad-hoc, conceptos de una teoría que existía únicamente en SOA, cuyo propósito es el de realizar sistemas interoperables de forma estandarizada y transparente. Esto permite que los dispositivos ubicuos puedan cooperar a la hora de realizar sus tareas con ordenadores, teléfonos móviles, etc. que implementen la interfaz de SOA de forma transparente.

En las secciones siguientes realizaremos una descripción más pormenorizada.

### 7.1.1. Desarrollar un framework que implemente la especificación WS-CDL que pueda ser ejecutado por dispositivos ubicuos

El cumplimiento de este objetivo es uno de los primeros avances que se hicieron con este trabajo de investigación, ya que no existen al día de hoy dispositivos que puedan participar de la ejecución de una coreografía y menos aún si la misma se implementa con el lenguaje de descripción planteado por el W3C, ya que la sola incorporación de la descripción de la coreografía en la memoria del dispositivo sería imposible de realizar.

En la primer prueba de concepto, el framework fue desarrollado en PHP y ejecutado en una RaspberryPi B+. Si bien podemos considerar que este dispositivo posee características similares a los dispositivos ubicuos, su hardware es lo suficientemente potente como para permitir el desarrollo de aplicaciones muy similares a las que se pueden desarrollar en equipos avanzados. No obstante, a partir de la segunda prueba de concepto, incorporamos dispositivos de menores prestaciones, como es el caso de placas de desarrollo Arduino. Para ello, fue necesario adaptar el framework, ya que el lenguaje para programar este tipo de placas es habitualmente C++. Además de la traducción del código, hubo que tener en cuenta que las limitaciones de memoria de los dispositivos nos han obligado a:

- Seleccionar de los elementos más importantes de WS-CDL: En un principio el tamaño de memoria disponible de estas placas de desarrollo nos plantearon un desafío importante ya que con esas limitaciones era impensado poder incluir un motor de ejecución de coreografías de forma completa. Sin embargo, esta limitación fue superada escogiendo las características y elementos del lenguaje WS-CDL que forman parte del núcleo de la ejecución de coreografías, dejando para futuras implementaciones cuestiones relacionadas a variables, alertas, etc.

- Compactar la coreografía: La descripción de la coreografía debía limitarse en cada dispositivo a un trozo de la misma, que es la que indica al dispositivo cuáles son los pasos siguientes de la coreografía a ejecutar, y luego recodificada.

- Minimizar el uso de memoria: Discutiremos en la siguiente sección las estrategias de ahorro de la memoria disponible.

En contrapartida, las limitaciones de procesamiento de los dispositivos ubicuos no han resultado un problema. Se ha podido adaptar el código y la ejecución de la coreografía de manera tal que los dispositivos, por poca capacidad de procesamiento que tengan, puedan ser parte de la ejecución. En nuestra opinión, cualquier dispositivo existente en el mercado podría ejecutar el framework desarrollado en esta tesis.

### 7.1.2. Lograr la coordinación de dispositivos ubicuos teniendo en cuenta las características distintivas de los mismos, tales como escasa capacidad de memoria, de procesamiento, batería, problemas de conectividad, etc.

El aspecto más importante dentro de la investigación fue, probablemente, adaptar el lenguaje de especificación de coreografías WS-CDL a las distintas características propias de los dispositivos ubicuos. Para un listado de dichas características debe acudir a la tabla 5.1, en donde se pueden visualizar las características de los dispositivos ubicuos vs. los problemas que puede acarrear en la composición de una coreografía con elementos de este tipo.

Todas las características listadas en la tabla 5.1 han sido tomadas en consideración, a distintos niveles de detalle. Los recursos limitados han sido probados ampliamente (espacio de almacenamiento de programa, memoria, ancho de banda, batería, etc.). En todos estos casos ha sido posible adaptar el framework de coordinación a las limitaciones de los dispositivos, sin necesidad de hacer modificaciones al lenguaje de definición de coreografías.

Exceptuando el caso de la limitante de memoria que ha generado problemas para su solución, el resto de las limitaciones de recursos no han supuesto problema alguno, la conectividad sí ha supuesto un desafío relativamente importante. Un dispositivo puede desaparecer repentinamente, por asuntos relacionados con los modos de ahorro de batería, conectividad, etc. y volver a estar disponible en un cortísimo espacio de tiempo. De la misma forma, en ambientes ubicuos existen habitualmente múltiples dispositivos confiables que pueden llevar a cabo las mismas tareas que el dispositivo desaparecido. Para aprovechar estas características, realizamos una ampliación del lenguaje WS-CDL, la cual permite especificar dispositivos que son el respaldo de otros[1] que pueden tener desapariciones, de manera tal que no se resienta la ejecución de la coreografía. Esto incluso podría derivar en un sublenguaje de especificación de coreografías específico para dispositivos ubicuos, o incluso un nuevo lenguaje que determine la forma en que los dispositivos se coordinen para brindar soluciones a través de servicios. Alternativas similares se plantean también en (Dragoni et al., 2017a).

### 7.1.3. Asegurar/mantener interoperabilidad entre aplicaciones SOA ejecutadas en servidores arbitrarios y dispositivos ubicuos

Este objetivo también ha sido alcanzado satisfactoriamente. Desde la segunda prueba de concepto se encuentran integrados equipos que ejecutan aplicaciones SOA de manera tradicional (servidores y computadores de gran capacidad de procesamiento), ofreciendo los servicios web a través de una implementación REST.

[1] Incluso a través de la virtualización de los mismos en la nube

Si bien la distribución geográfica de los equipos no debería tener influencia en la ejecución final de la coreografía, hemos realizado pruebas con servidores y equipos a lo largo de todo el mundo, incluyendo países como España, Ecuador y Argentina. En caso de existir problemas de latencia en la red de transporte (en esta investigación hemos usado únicamente el protocolo TCP/IP) o time-outs, el framework implementado tomará las medidas especificadas en la declaración de la coreografía. Estas medidas pueden ir desde utilizar un dispositivo de reemplazo o a la cancelación de la ejecución por completo.

### 7.1.4. Conseguir que los dispositivos ubicuos soporten el manejo de distintas especificaciones basadas en SOA (transacciones, seguridad, etc.)

Durante la quinta prueba de concepto se logró implementar el manejo de una transacción (emulando la especificación WS-AtomicTransaction), incluyendo en la misma a dispositivos ubicuos. Dicha transacción involucró tanto a servidores tradicionales como a dispositivos ubicuos. En este caso particular, se implementó una transacción distribuida aplicando el método de TryConfirmCancel, propuesto por (Pardon y Pautasso, 2014). Al producirse el Rollback, los dispositivos deben implementar un servicio temporal y a demanda de cancelación de la transacción, el cual debe permanecer disponible por un tiempo determinado para que el mismo pueda ser invocado desde el manejador de la transacción.

Existen otros estándares planteados en SOA, como es el caso de las especificaciones WS-SECURITY, WS-RELIABILITY, etc. Si bien estas especificaciones no fueron implementadas durante las pruebas de concepto, las mismas podrían realizarse incluyendo el código necesario dentro del framework de ejecución, sin perjuicio de la especificación del lenguaje de especificación de coreografías. Para ello, únicamente sería necesario implementar una capa de seguridad SSL sobre el protocolo TCP/IP, sobre la cual se instalarían las correspondientes pilas de protocolos

## 7.2. Limitaciones de la solución propuesta

El framework propuesto posee varias limitaciones: pre-programación de la coreografía dentro del dispositivo, no considera dispositivos de escasas prestaciones como los dispositivos RFID, comunicación únicamente a través de la norma WIFI 802.11 y que la prueba de concepto utilizada debe integrar una mayor cantidad de dispositivos,las cuales describimos a continuación.

- La ejecución de las coreografías dentro de los dispositivos ubicuos, si bien se ajusta al lenguaje de definición de coreografías WS-CDL, se basa en una codificación pre-programada al momento de la escritura

del código dentro del dispositivo, es decir, de una manera estática y no dinámica en tiempo de ejecución. Esta limitación se debe a que estos dispositivos poseen escasa memoria RAM, lo que dificulta almacenar en memoria la especificación de la coreografía en formato XML.

Existen alternativas para que la definición de la coreografía no se realice de una manera estática dentro del dispositivo y que, a su vez, no implique un alto consumo de memoria. Estas alternativas están bajo estudio y merecen un análisis en particular cada una de ellas, valorando las virtudes y deficiencias para luego implementar aquella que tenga el mejor índice de costo/beneficio ponderando las características de los dispositivos en cuestión. Por ejemplo una alternativa podría ser solicitar a un elemento centralizador la parte de la coreografía que debe ejecutar el dispositivo, en tiempo real.

- Un punto importante también es poder incluir dispositivos con incluso menores prestaciones que los dispositivos ubicuos, tales como tarjetas RFID, dentro de la ejecución de la coreografía. Para ello será necesario hacer modificaciones y/o extensiones al WS-CDL para que estos puedan ser incorporados de manera especial y ser parte de la ejecución de la coreografía.

- Una de las características que no se ha implementado en las distintas pruebas llevadas adelante durante la investigación ha sido la de realizar la comunicación entre los dispositivos a través de otros elementos o módulos de conexión distintos de la norma 802.11, como puede ser Bluetooth o RFID. Cabe aclarar que esta investigación no tiene como objetivo evaluar cuál es el mejor método de interconectividad, sino por el contrario, lo que intenta es sentar las bases de la coordinación de dispositivos ubicuos a través de cualquier método o protocolo de conexión disponible, haciendo la ejecución de coreografías mucho más dinámico y versátil.

  Si bien puede ser motivo de un gran debate este punto, creemos que es totalmente posible la implementación de cualquier otro método de conectividad, distinto de la norma WIFI 802.11. En el caso del framework propuesto, ello implicaría generar una jerarquía de clases que implementen la recepción y envío de mensajes como se ha hecho con el módulo ESP8266-01. Del gráfico 6.13 se debería implementar una clase denominada connectionBluetooth (por poner un ejemplo de implementación con el estándar Bluetooth), similar a la clase existente connectionESP8266 para que lleve adelante el manejo de los mensajes recibidos y enviados. De esta forma se pueden implementar distintos modelos de conectividad de acuerdo a las características de cada dispositivo. Esto mejoraría a la interoperabilidad de la solución, lo cual

convertiría al framework de coordinación en un mecanismo muy versátil.

- Finalmente, deseamos poder realizar una prueba de concepto más grande, con más dispositivos y características mucho más ajustada a la realidad de un problema en particular, a partir de la cual podamos evolucionar la solución propuesta hacia un producto más industrial e incluso comercial.

## 7.3. Relación con otras tecnologías

En esta sección compararemos los resultados obtenidos durante nuestra investigación con otras tecnologías asentadas, como los frameworks Gaia u Oxigen, así como con tecnologías emergentes de este momento, más concretamente microservicios y IoT.

La similitud entre todas estas tecnologías es muy elevada, ya que el concepto subyacente es el mismo: utilizar dispositivos de reducidas prestaciones y equipos tradicionales al mismo tiempo, trabajando de manera coordinada. No obstante si bien a nivel general se puede apreciar una gran similitud, en la medida que ahondamos en la base subyacente de las tecnologías encontramos diferencias sustanciales.

### 7.3.1. Relación con Frameworks de dispositivos ubicuos

Si bien en este capítulo no discutiremos acerca de la relación entre este trabajo de investigación y los frameworks o ambientes existentes relacionados con computación ubicua o computación sensible al contexto, como los presentados durante el capítulo de estado de la cuestión 2.3, es destacable mencionar que la diferencia con los mismos es amplia, ya que dichos frameworks se centran únicamente en la forma en que pueden compartir información los dispositivos, y no en cómo estos se coordinan entre sí para llevar adelante una tarea. Además de que la arquitectura sobre la que se basan implica que existan dispositivos centrales y que permitan la conectividad de los restantes.

### 7.3.2. Relación con Microservicios

Un microservicio es un proceso independiente y cohesivo interactuando vía mensajes (Dragoni et al., 2017a). La arquitectura de microservicios es una aplicación distribuida donde todos sus módulos son microservicios (Dragoni et al., 2017a). Esta arquitectura no se apega a ningún lenguaje de programación en particular, sino que sólo provee una estrategia acerca de cómo distribuir los componentes de una aplicación distribuida.

Este enfoque ha sido construido sobre los conceptos de SOA pero, a su vez esta arquitectura tiene algunas características que la hacen distintiva de SOA, como son: el tamaño de los servicios, el contexto donde se ejecutan, la independencia que poseen los servicios, la flexibilidad, la modularidad y la evolución que pueden tener los microservicios desarrollados. Aunque, es importante mencionar también, que los microservicios no están orientados a ser utilizados en dispositivos ubicuos, sino como parte de un sistema basado en la arquitectura SOA, lo que lo deferencia de esta propuesta de investigación.

La coincidencia entre la arquitectura de microservicios y la solución propuesta es notable. Nuestro framework utiliza microservicios en los dispositivos, construyendo de esta manera una arquitectura análoga a la de microservicios como base para la coordinación de los mismos a través de coreografías. Además, tanto en microservicios como en este trabajo, se utilizan mecanismos de REST y XML con JSON como formato de intercambio de datos.

La investigación realizada sobre la utilización de coreografías descriptas en el lenguaje de especificación de coreografías WS-CDL para coordinar dispositivos ubicuos, es una mejora que realizamos a la arquitectura de microservicios, ya que según (Dragoni et al., 2017a) este enfoque de realizar colaboraciones entre microservicios a través de coreografías es de suma relevancia para esta tecnología.

### 7.3.3. Relación con IoT

Kevin Ashton fue quien primero utilizó el término de IoT, en el año 1999, donde lo describió como un sistema donde Internet está conectado al mundo físico y real a través de sensores ubicuos. Si bien el término ha ido evolucionando con el discurrir del progreso tecnológico, IoT básicamente sigue representando lo mismo: un mundo poblado de dispositivos (sin intervención humana) y que se comunican entre ellos. (Gubbi et al., 2013) deja en claro una situación muy importante, y es que IoT ha dejado o se ha salido del camino de la interconectividad a través de distintos mecanismos como Bluetooth, RFID, etc, para centrarse únicamente en Internet, haciendo hincapié en que la interconexión entre distintos objetos se hará usando principalmente TCP/IP.

Existe una similitud entre IoT y el framework propuesto en este trabajo de investigación, ya que en ambos casos se trata de coordinar dispositivos ubicuos como es el caso de sensores, actuadores, electrodomésticos con el fin de que realicen tareas sin intervención humana.

Como hemos indicado anteriormente, nuestra investigación no se basa únicamente en la interconectividad TCP/IP, sino por el contrario, plantea que sea posible utilizar distintos protocolos, siendo elegido el que más se adapte a las necesidades de los dispositivos que intentan coordinarse.

Además de los aspectos relativos a la conectividad, creemos que la coor-

dinación de los distintos elementos que componen una red de IoT podría ser llevada adelante con el framework aquí presentado y especificado a través del lenguaje de definición de coreografías WS-CDL. En muchos casos, los proyectos de IoT trabajan sobre la idea de una orquestación de dispositivos, controlados por un equipo central (por ej. una central de alarma). Con el framework propuesto, los dispositivos podrían ser no solo prestadores de un servicio sino también consumidores, realizando toda esta tarea sin la necesidad de la coordinación general a través de otro equipo o dispositivo central.

# Capítulo 8

# Conclusiones

La solución planteada, esto es, la utilización del estándar WS-CDL de coreografías para SOA, junto con el framework desarrollado, ha resuelto satisfactoriamente el problema planteado.

El principal problema que la presente tesis ha afrontado, el cual está descrito en detalle en la Sección de Planteamiento del problema, consistía en que la composición de dispositivos se realizaba a través de protocolos propietarios y sin seguir definiciones estándares. Esto ha provocado que dispositivos de distintos proveedores no puedan ser utilizados en una composición, presentando limitaciones importantes a la hora de realizar composiciones con estos dispositivos.

La solución planteada permite realizar composiciones con dispositivos ubicuos de una manera abierta, basada en estándares y escalable. Esto implica que diversos dispositivos, de diversos fabricantes, y con distintas capacidades, pueden ser incorporados fácilmente al framework y, por consiguiente, participar en coreografías con otros dispositivos de forma sencilla.

El framework desarrollado no posee gran complejidad. Es bastante simple desde el punto de vista de diseño, y no demasiado exigente en términos de procesamiento y memoria. Por ello, creemos que la solución planteada puede aplicarse en sectores comerciales como, por ejemplo, el de sistemas de seguridad hogareña, donde se podría utilizar el framework desarrollado para implementar sistemas más sofisticados que los actuales, con la posibilidad de manejar una mayor cantidad de dispositivos, realizando tareas más complejas que la generación de una simple señal de activación, como ocurre hoy en día. Otro sector en donde podría aplicarse el framework desarrollado es en la atención domiciliaria de pacientes con enfermedades que deban ser monitorizados a distancia, incluso con posibilidad de poder aplicar medicación en caso de ser necesario.

La solución planteada también es aplicable a IoT, donde los dispositivos no solamente estarían conectados entre sí, sino que también podrían utilizar otros protocolos de comunicación más allá de TCP/IP.

La solución propuesta permite, adicionalmente que los dispositivos ubicuos pueden interactuar no sólo con otros dispositivos ubicuos, sino también con aplicaciones basadas en SOA. De esta manera, los dispositivos pueden ser integrados en sistemas existentes ampliando enormemente sus capacidades.

Finalmente, la solución propuesta no es, ni mucho menos, completa. Es posible, y deseable, realizar toda una serie de mejoras y ampliaciones que describimos en la siguiente sección de Futuras Líneas.

# Capítulo 9

# Futuras líneas

Los resultados de esta investigación han sido satisfactorios y, al mismo tiempo, prometedores. De la mano con los hallazgos alcanzados, se han abierto tanto nuevas líneas de investigación como también posibilidades de producción de software de uso industrial. A continuación reportamos las futuras opciones de investigación y desarrollo que esta tesis ha propiciado.

## 9.1. Líneas futuras de investigación

En esta sección abordaremos las futuras líneas relacionadas principalmente con los logros obtenidos durante esta investigación, en lo concerniente a:

### 9.1.1. Descubrimiento de servicios

Durante la presente investigación se ha trabajado sobre servicios y dispositivos ubicuos estáticos, es decir, preestablecidos dentro de la coreografía y presentes en la red de comunicaciones antes del inicio de cualquier interacción. Resultaría una línea de investigación muy interesante la de poder coordinar a través de coreografías dispositivos ubicuos descubiertos “on the fly". Existen trabajos relacionados con el descubrimiento de servicios a demanda, indicados en el capítulo 2, los cuales podrían ser aplicados, con ajustes, al presente trabajo de investigación.

### 9.1.2. Introducción de capas de seguridad

Una línea de investigación que también resultaría en un notable progreso a este trabajo de tesis es la relacionada con la introducción dentro del framework de coordinación de las capas de seguridad existentes en servicios web como es el caso de WS-SECURITY. La inclusión de esta capa de seguridad significa un desafío importante ya que se debe estudiar de qué manera es posible incorporarla dentro de dispositivos con poca capacidad de

procesamiento y escasa memoria. Para alcanzar dicho fin probablemente será necesario realizar ajustes importantes a las especificaciones de seguridad, incluso mayores que las realizadas a las coreografías.

### 9.1.3. Implementación de transacciones de acuerdo a estándares

Otra línea de investigación, relacionada en cierta forma con la anterior, es la de implementar la totalidad de los estándares de transacciones disponibles en SOA dentro del framework de coordinación propuesto en esta tesis. En esta investigación se realizó la implementación de una transacción distribuida, pero sin ajustarse en su totalidad a los estándares WS-TRANSACTION y WS-COORDINATION. Los avances en esta línea serían desafiantes, ya que la aplicación de estos estándares a dispositivos con características limitadas es una tarea que demandará mucha investigación.

### 9.1.4. Convergencia con microservicios

Hemos mencionado durante la sección de Discusión que existe una relación bastante estrecha entre esta tesis y la tecnología de microservicios. En microservicios se discute sobre la forma en que se pueden coordinar los servicios para llevar adelante tareas en conjunto, donde una de las formas presentadas es la de coreografías. Por lo tanto, una futura línea de investigación consistiría en la convergencia entre los conceptos emanados de este trabajo de Tesis y la tecnología de microservicios.

### 9.1.5. Convergencia con IoT

Como hemos indicado en la sección de Discusión, IoT utiliza únicamente el protocolo TCP/IP para la interconectividad de los distintos componentes. También hemos mencionado que es una limitación que no se ha afrontado en el framework desarrollado en esta tesis. Por lo tanto, una futura línea de investigación consistiría en la convergencia entre ambas tecnologías ampliándolas a la utilización de otros protocolos como puede ser Bluetooth o RFID, por mencionar dos ejemplos destacados.

## 9.2. Líneas futuras de desarrollo

Estas líneas están basadas en la explotación de los resultados de esta investigación para el desarrollo de productos industriales.

### 9.2.1. Implementación del framwork de ejecución de coreografías sobre dispositivos ubicuos con calidad industrial

Para poder llevar adelante la investigación fue necesario implementar un framework de coordinación de dispositivos ubicuos tal y como exige la meto-

dología de "Design science" elegida. Si bien ello fue suficiente para los fines de la presente investigación, el prototipo desarrollado no alcanza la calidad necesaria para ser aplicado efectivamente sobre problemas de coordinación de dispositivos ubicuos en la industria.

Aunque sí deja las bases sentadas para que el desarrollo siga evolucionando hasta poder producir un framework sólido y de características industriales para que el mismo pueda ser utilizado en entornos reales.

### 9.2.2. Ampliación de las pilas de protocolos

El objetivo en este caso sería mejorar la conectividad de los dispositivos que forman parte de una coreografía. Para ello se debería extender el framework para utilizar otros protocolos de comunicaciones, como puede ser el caso de la utilización de protocolos Bluetooth, RFID, etc. Esta mejora permitiría ejecutar coreografías no sólo con una mejor diversidad de dispositivos ubicuos sino también probablemente de forma más versátil, aprovechando las posibilidades que dichos protocolos ofrecen, ej: la comunicación a través de redes de sensores.

# Bibliografía


Hayes at commands. http://michaelgellis.tripod.com/modem.html, 2010.

Amigo Project. http://www.hitech-projects.com/euprojects/amigo, 2019.

KNX. https://www.knx.org/knx-en/for-professionals/index.php, 2019.

Mosquito. http://www.mosquito-online.org/, 2019.

openHab Project. https://www.openhab.org/, 2019.

Bellifemine, F. L., Caire, G. y Greenwood, D. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley & Sons, Inc., USA, 2007. ISBN 0470057475.

Bouguettaya, A., Sheng, Q. y Daniel, F. *Web Services Foundations*. Springer New York, 2013. ISBN 9781461475187.

Campbell, R. H., Mickunas, D. M., Reed, D. y Nahrstedt, K. Active Spaces for Ubiquitous Computing. Available at http://gaia.cs.illinois.edu/, 2002.

Cassar, G., Barnaghi, P., Wang, W., De, S. y Moessner, K. Composition of services in pervasive environments: A Divide and Conquer approach. En *Computers and Communications (ISCC), 2013 IEEE Symposium on*, páginas 000226–000232. 2013.

Cherrier, S., Ghamri-Doudane, Y. M., Lohier, S. y Roussel, G. Services collaboration in wireless sensor and actuator networks: Orchestration versus choreography. En *2012 IEEE Symposium on Computers and Communications (ISCC)*, páginas 000411–000418. 2012. ISSN 1530-1346.

Dragoni, N., Giallorenzo, S., Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R. y Safina, L. Microservices: yesterday, today, and tomorrow. En *Present and Ulterior Software Engineering* (editado por M. Mazzara y B. Meyer). Springer, 2017a.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. y Safina, L. *Microservices: Yesterday, Today, and Tomorrow*, páginas 195–216. Springer International Publishing, Cham, 2017b. ISBN 978-3-319-67425-4.

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

Georgakopoulos, D. y Papazoglou, M. P. *Service-Oriented Computing*. The MIT Press, 2008. ISBN 0262072963, 9780262072960.

Gubbi, J., Buyya, R., Marusic, S. y Palaniswami, M. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, vol. 29(7), páginas 1645 – 1660, 2013. ISSN 0167-739X.

Harkes, J., Farbacher, T. y Miller, N. Project Aura Distraction-free Ubiquitous Computing. Available at `http://www.cs.cmu.edu/~./aura/people.html`, 2002.

Hayat, Z., Reeve, J. y Boutle, C. Ubiquitous security for ubiquitous computing. *Information Security Technical Report*, vol. 12(3), páginas 172 – 178, 2007. ISSN 1363-4127.

Hevner, A. R. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, vol. 19(2), páginas 87–92, 2007.

Hevner, A. R., March, S. T., Park, J. y Ram, S. Design science in information systems research. *MIS Q.*, vol. 28(1), páginas 75–105, 2004. ISSN 0276-7783.

Inaganti, S. y Behara, G. K. Service Identification: BPM and SOA Handshake. *Business Process Trends*, 2007.

Krumm, J. *Ubiquitous Computing Fundamentals*. Chapman and Hall/CRC, 2010. ISBN 9781420093612.

Lien, S., Chen, K. y Lin, Y. Toward ubiquitous massive accesses in 3gpp machine-to-machine communications. *IEEE Communications Magazine*, vol. 49(4), páginas 66–74, 2011. ISSN 0163-6804.

Ling Yibo, Pong Terrence, Vassiliou Christophoros C, Huang Paul L y Cima Michael J. Implantable magnetic relaxation sensors measure cumulative exposure to cardiac biomarkers. *Nat Biotech*, vol. 29(3), páginas 273–277, 2011. ISSN 1087-0156. 10.1038/nbt.1780.

Loke, S. W. Supporting ubiquitous sensor-cloudlets and context-cloudlets: Programming compositions of context-aware systems for mobile users. *Future Generation Computer Systems*, vol. 28(4), páginas 619–632, 2012. ISSN 0167-739X.

March, S. T. y Smith, G. F. Design and Natural Science Research on Information Technology. *Decis. Support Syst.*, vol. 15(4), páginas 251–266, 1995. ISSN 0167-9236.

Mostarda, L., Marinovic, S. y Dulay, N. Distributed orchestration of pervasive services. En *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, páginas 166–173. 2010. ISSN 2332-5658.

Mühlhäuser, M. y Gurevych, I., editores. *Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises*. IGI Global, 2008. ISBN 9781599048321.

Najar, S., Pinheiro, M. K. y Souveyet, C. A New Approach for Service Discovery and Prediction on Pervasive Information System. *Procedia Computer Science*, vol. 32, páginas 421–428, 2014. ISSN 1877-0509. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).

OASIS, O. f. A. S. I. S. 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, 2007.

OASIS, O. f. A. S. I. S. Web Services Coordination (WS-Coordination). http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html, 2009.

Palmieri, F. Scalable service discovery in ubiquitous and pervasive computing architectures: A percolation-driven approach. *Future Generation Computer Systems*, vol. 29(3), páginas 693–703, 2013. ISSN 0167-739X. Special Section: Recent Developments in High Performance Computing and Security.

Pant, K. y Juric, M. B. *Business Process Driven SOA Using BPMN and BPEL: From Business Process Modeling to Orchestration and Service Oriented Architecture*. Packt Publishing, Limited, 2008. ISBN 9781847191472.

Papazoglou, M. y van den Heuvel, W.-J. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, vol. 16(3), páginas 389–415, 2007. ISSN 1066-8888.

Pardon, G. y Pautasso, C. Atomic distributed transactions: A restful design. En *Proceedings of the 23rd International Conference on World Wide Web*, WWW 14 Companion. Association for Computing Machinery, New York, NY, USA, 2014. ISBN 9781450327459.

Peffers, K., Tuunanen, T., Rothenberger, M. A. y Chatterjee, S. A design science research methodology for information systems research. *Journal of Management Information Systems*, vol. 24(3), páginas 45–77, 2007. ISSN 07421222.

Poslad, S. *Ubiquitous Computing. Smart devices, Environments and Interactions*. Wiley, 2009. ISBN 9780470035603.

Pree, W. *Design Patterns for Object-Oriented Software Development*. ACM Press/Addison-Wesley Publishing Co., USA, 1995. ISBN 0201422948.

Quitadamo, R., Zambonelli, F. y Cabri, G. The service ecosystem: Dynamic self-aggregation of pervasive communication services. En *First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments (SEPCASE '07)*, páginas 1–1. 2007.

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

Sarshar, N., Boykin, O. y Roychowdhury, V. Scalable percolation search on complex networks. *Theoretical Computer Science*, vol. 355(1), páginas 48 – 64, 2006. ISSN 0304-3975. Complex Networks.

Sarshar, N., Boykin, P. O. y Roychowdhury, V. P. Percolation search in power law networks: making unstructured peer-to-peer networks scalable. En *Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings.*, páginas 2–9. 2004.

Science, M. L. f. C. y Laboratory, M. A. I. Pervasive Human-Centered Computing. Available at http://oxygen.csail.mit.edu/, 2002.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S. y Xu, X. Web services composition: A decade's overview. *Information Sciences*, vol. 280(0), páginas 218–238, 2014. ISSN 0020-0255.

Tapia, V. Industria 4.0 - internet de las cosas. *UTCiencia - Ciencia y TecnologÃa al servicio del pueblo*, vol. 1(1), páginas 51–60, 2017. ISSN 2602-8263.

Viroli, M. On competitive self-composition in pervasive services. *Science of Computer Programming*, vol. 78(5), páginas 556–568, 2013. ISSN 0167-6423. Special section: Principles and Practice of Programming in Java 2009/2010 & Special section: Self-Organizing Coordination.

Viroli, M. y Zambonelli, F. A biochemical approach to adaptive service ecosystems. *Information Sciences*, vol. 180(10), páginas 1876 – 1892, 2010. ISSN 0020-0255. Special Issue on Intelligent Distributed Information Systems.

W3C. Web Services Choreography Description Language: Primer. https://www.w3.org/TR/ws-cdl-10-primer/, 2006.

Wang, A. y Tonse, S. Announcing ribbon: Tying the netflix mid-tier services together. http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html, 2013.

Wang, S., Wan, J., Zhang, D., Li, D. y Zhang, C. Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, vol. 101, páginas 158 – 168, 2016. ISSN 1389-1286. Industrial Technologies and Applications for the Internet of Things.

Weiser, M. Hot topics-ubiquitous computing. *Computer*, vol. 26(10), páginas 71–72, 1993. ISSN 0018-9162.

Yu, Q., Liu, X., Bouguettaya, A. y Medjahed, B. Deploying and managing Web services: issues, solutions, and directions. *The VLDB Journal*, vol. 17(3), páginas 537–572, 2008. ISSN 1066-8888.

Yu, W. D. y Ong, C. H. A SOA Based Software Engineering Design Approach in Service Engineering. En *e-Business Engineering, 2009. ICEBE 2009. IEEE International Conference on*, páginas 409–416. 2009.

Zhou, Z., Zhao, D., Liu, L. y Hung, P. C. Energy-aware composition for wireless sensor networks as a service. *Future Generation Computer Systems*, vol. 80, páginas 299 – 310, 2018. ISSN 0167-739X.

Zimmermann O, S. N. W. G. P. M. Analysis and Design Techniques for Service-Oriented Development and Integration. Available at `http://www.perspect.ivesonwebservices.de/download/INF05-ServiceModelingv11.pdf`, 2013.

# Lista de acrónimos

API . . . . . . . . . . . *Application Programming Interface*, Interfaz de Programación de Aplicaciones

ASP . . . . . . . . . . . *Active Server Pages*, Servidor de Páginas Activas

BPEL . . . . . . . . . *Business Process Execution Language*, Lenguaje de Ejecución de Procesos de Negocio

BPMN . . . . . . . . *Business Process Model and Notation*, Modelo y Notación de Procesos de Negocio

CGI . . . . . . . . . . . *Common Gateway Interface*, Interfaz de entrada común

CORBA . . . . . . *Common Object Request Broker Architecture*

DCE . . . . . . . . . . *Distributed Computing Environment*, Ambiente Distribuído de Computación

EPROM . . . . . . *Erasable Programmable Read-Only Memory*, ROM programable borrable

FIPA . . . . . . . . . . *Foundation for Intelligent, Physical Agents*, Fundación para Agentes Físicos Inteligentes

HTTP . . . . . . . . *Hypertext Transfer Protocol*, Protocolo de Transferencia de Hypertexto

IoT . . . . . . . . . . . *Internet of Things*, Internet de las Cosas

JADE . . . . . . . . . *Java Agent DEvelopment Framework*, Framework de Desarrollo de Agentes Java

JSON . . . . . . . . . *JavaScript Object Notation*, Notación de Objetos en JavaScript

JSP . . . . . . . . . . . *Java Server Pages*, Servidor de Páginas Java

JVM . . . . . . . . . . *Java Virtual Machine*, Máquina Virtual Java

LAN .......... *Local Area Network*, Red de Area Local

MIT .......... *(Massachusetts Institute of Technology*, Instituto de Tecnología de Massachusetts

NFC .......... *Near Field Communication*, Comunicación cercana

OASIS ........ *Organization for Advanced Structured Information Systems*

OSGi.......... *Open Services Gateway initiative*, Open Services Gateway initiative

PDA .......... *Personal Digital Assistant*, Asistente Digital Personal

PHP .......... *Hypertext PreProcessor*, PreProcesador de Hypertexto

REST ......... *Representational State Transfer*, Transferencia de Estado Representacional

RFID ......... *Radio Frequency Identification*, Identificación por radiofrecuencia

SOA .......... *Service Oriented Architecture*, Arquitectura Orientada a Servicios

SOAP......... *Simple Object Access Protocol*, Protocolo Simple de Acceso a Objetos

SOC .......... *Software Oriented Computing*, Computación Orientada a Servicios

SRAM ........ *Static Random Access Memory*, Memoria estática de acceso aleatorio

TI............. *Information Technology*, Tecnologías de la Información

UML.......... *Unified Modeling Language*, Lenguage Unificado de Modelado

VM ........... *Virtual Machine*, Máquina Virtual

VPN .......... *Virtual Private Network*, Red Privada Virtual

W3C.......... *World Wide Web Consortium*, Consorcio Mundial Web

WAN.......... *Wide Area Network*, Red de Area Amplia

WS-BPEL .... *Web Service Business Process Execution Language*, Lenguaje de Ejecución de Procesos de Negocio con Servicios Web

WS-CDL...... *Web Service Choreography Description Language*, Lenguaje de Descripción de Coreografías con Servicios Web

WSCI......... *Web Service Choreography Interface*, Interface de Coreografías de Servicios Web

WSDL........ *Web Service Description Language*, Lenguaje de Descripción de Servicios Web

XML.......... *eXtensible Markup Language*, Lenguaje de Marcas Extensible

# Parte I

# Apéndices

# Análisis de la utilización de memoria

En este apéndice mostraremos un análisis de la utilización de la memoria en la placa Arduino, donde se toma como base la implementación y código utilizado durante la tercera prueba de concepto. Este análisis nos permitió saber cuál era el límite mínimo de memoria que debía disponer el dispositivo para poder ejecutar la coreografía y ser parte de ella.

Durante el desarrollo, se mostrará el código y un análisis de la memoria utilizada por Arduino, junto a la ejecución real para tener una comparación y poder apreciar cómo se va utilizando y consumiendo dicha memoria disponible.

```
void connectionESP8266::inicializarServer(int port, char *ssid,
     char *pass, char *rewriteBase,Stream* espS, Stream* logS)
{
    uint8_t status;              // the Wifi radio's status

  _espSerial = logS;

    status = WL_IDLE_STATUS;
    this->_server = new WiFiEspServer(port);

    WiFi.init(_espSerial);

    if (WiFi.status() == WL_NO_SHIELD) {
        while (true);
    }

    while ( status != WL_CONNECTED) {
        status = WiFi.begin(ssid, pass);
    }

    this->_server->begin();

    MyAPI *_gestorChor = new MyAPI();

    this->_servidorREST = new RESTWebServer(rewriteBase, _gestorChor);
    return;
}
```

Memoria disponible al comienzo: 6823 (surge de los 8k disponibles menos los 1300 de uso de variables globales, más alguna inicialización de la placa y el programa)

En este punto la memoria de variables locales se incrementa en aproximadamente en 1 byte

```
  1byte -------------------> status
```

Y la memoria heap en aproximadamente en 16 bytes

```
2bytes ------------------> *_gestorChor
2bytes  -----------------> this->_server
2bytes ------------------> this->_servidorREST
2bytes ------------------> port
2bytes ------------------> *ssid
2bytes ------------------> *pass
2bytes ------------------> *rewriteBase
4bytes ------------------> punteros a Serial
```

Esto es solamente la inicialización, luego de ello quedaría como residente una cantidad de aproximadamente 10 bytes

```
2bytes ------------------> *_gestorChor
2bytes  -----------------> this->_server
2bytes ------------------> this->_servidorREST
4bytes ------------------> punteros a Serial
```

Memoria disponible al finalizar: 6777 (calculado segun la función de memoryfree)

A su vez se deben considerar que al crearse dos objetos en memoria, disparan la ejecución de sus respectivos constructores de clase, consumiendo memoria de acuerdo a las declaraciones que existen de propiedades de los mismos. A continuación se hace un estudio de la memoria consumida.

```
RESTWebServer::RESTWebServer(const char *RewriteBase,
              API *serviceCall)
{
    this->_RewriteBase = RewriteBase;
    this->_serviceCall = serviceCall;
}
```

Memoria disponible al comienzo: 6777.

Y las propiedades declaradas en la definición de la clase es la siguiente:

```
    int reqCount = 0;
    int cnt_entradas = -1, cnt_get = -1;
    char *entradas[SIZE_ENTRADAS];
    char *_GET[SIZE_GET];
```

En este punto la memoria de variables locales se incrementa en aproximadamente en 6 bytes

```
2bytes --------------------> reqCount
2bytes --------------------> cnt_entradas
2bytes --------------------> cnt_get
```

Y la memoria heap en aproximadamente en 20 bytes.

```
10bytes -------------------> *entradas
   (SIZE_ENTRADAS definido en 5)
   (cada entrada consta de 2 bytes ya que es un puntero a una cadena)
10bytes -------------------> *_GET
   (SIZE_GET definido en 5)
   (cada entrada consta de 2 bytes ya que es un puntero a una cadena)
```

Debemos sumar a la cantidad residente cantidad de 26 bytes, lo que acumula un total de 36 bytes.

Memoria disponible al finalizar: 6777 bytes.

Ahora analizamos el constructor de la clase MyAPI (almacenada en la variable *_gestorChor)

```
MyAPI::MyAPI(){

}
```

Como podemos apreciar esta clase no tiene consumo de memoria, porque tampoco tiene declaradas propiedades de la clase, es una especie de clase respondiendo al patrón de diseño factory.

Luego de esto se llama a la función procesarPeticion del objeto connectionESP8266.

```
void connectionESP8266::procesarPeticion(){
  char input[INPUT_SIZE + 1];        // string with get input
  int pos = 0;                   // Position of the input string

  this->_client = _server->available();
  if (this->_client) {
    boolean currentLineIsBlank = true;
    pos=0;
    while (this->_client.connected()) {
      if (this->_client.available()) {
        char c = this->_client.read();
        input[pos++] = c;
        if (c == '\n' && currentLineIsBlank) {
          // Cierro la cadena de entrada
          input[pos] = '\0';
          break;
        }
        if (c == '\n') {
           currentLineIsBlank = true;
        }else if (c != '\r') {
           currentLineIsBlank = false;
        }
      }
    }

    _servidorREST->procesarPeticion(input, this);
  }

}
```

En este punto la memoria de variables locales se incrementa en aproximadamente en 521 bytes

```
  513bytes --------------------> input
  2bytes   --------------------> pos
  1byte --------------------> currentLineIsBlank
  1byte   --------------------> c
  4bytes  --------------------> distintas comparaciones
                del tipo '\r', '\n', etc
```

Y la memoria heap en aproximadamente en 16 bytes

```
  2bytes --------------------> this->_client
```

Esto es solamente la inicialización, luego de ello quedaría como residente una cantidad de aproximadamente 2 bytes (sumados a los anteriores 36 bytes) nos da un total de 38 bytes

```
  2bytes  ------------------> this->_client
```

En este subproceso se han utilizado en total unos 523 bytes adicionales a los que ya tiene en memoria SRAM, lo que estaría sumando un total aproximadamente de 559 bytes sumados a los 1294 de memoria estática que tiene al momento de la compilación el sketch. Es decir, un total de 1853 bytes, muy cercano a los 2k disponibles. A esto se le debe sumar la fragmentación de la memoria.

Luego de ello se llama a la función _servidorREST->procesarPeticion, la cual analizaremos a continuación

```
void RESTWebServer::procesarPeticion(const char *input,
                    connection *pServer)
{
```

```
    bool res = false;
    choreographyService *chorService;

    if (input[0] == '\0'){
        return false;
    }

    if (existe_subcadena(input,"GET")){
        res = this->procesar_GET(input);
    }
    if (res){
        chorService = _serviceCall->getService(this->entradas[0],
                                  this->_GET, this->cnt_get);
        chorService->ejecutar(this->entradas[1],
                    this->entradas[2], pServer);
    }
    return;
}
```

En este punto la memoria de variables locales se incrementa en aproximadamente en 5 bytes

```
  1byte  -----------------------> res
  4bytes  ----------------------> distintas comparaciones
               del tipo '\0', 'GET', etc
```

Y la memoria heap en aproximadamente en 6 bytes

```
  2bytes --------------------> *chorService
  2bytes --------------------> *input
  2bytes --------------------> *pServer
```

Residente luego de esta llamada no queda nada, ya que son todas variables locales o de punteros locales, que una vez finalizada la ejecución del método se liberan. Por lo tanto se sigue con una ocupación de memoria de unos 559 bytes aproximados. De todas formas estas variables incrementan la cantidad de memoria utilizada de manera temporal a unos 570 bytes (559 + 11).

En este método se invoca al método procesar_GET de esta misma clase, lo analizaremos a continuación

```
bool RESTWebServer::procesar_GET(const char *input){
  char *parte;
  bool res = false;

  cnt_entradas = -1;
  cnt_get = -1;

  parte = strstr(input, "GET ");
  if (parte != NULL){
    parte = strtok(parte,"\n");
    parte = substr(parte,0,
      (strlen(parte) - strlen(strstr( input, "HTTP/1.1"))));
    parte = substr(parte,4,0);

    if (existe_subcadena(parte,_RewriteBase)){
       char *token = strtok(parte, "/");
       while(token != NULL){
         entradas[++cnt_entradas] = token;
         if ((cnt_entradas == 0) &&
            (strcmp(strlwr(entradas[cnt_entradas]),_RewriteBase) == 0))
          {
            cnt_entradas--;
        }
        token = strtok(NULL, "/");
      }

      parte = substr(strstr(entradas[cnt_entradas], "?"),1,0);
      if (strlen(parte) > 0){
         entradas[cnt_entradas] = substr(entradas[cnt_entradas],
                0,strpos(entradas[cnt_entradas],"?"));
         token = strtok(parte, "&");
         while(token != NULL){
           _GET[++cnt_get] = token;
```

```
            token = strtok(NULL, "&");
          }
        }
        res = true;
      }

    }
    return res;
}
```

En este punto la memoria de variables locales se incrementa en aproximadamente en 1 byte

```
  1byte  --------------------> res
  19bytes --------------------> distintas comparaciones
                del tipo '\0', 'GET', etc
```

Y la memoria heap en aproximadamente en 16 bytes

```
  2bytes ------------------> *parte
  2bytes ------------------> *toquen
```

Esto suma unos 24 bytes adicionales, los cuales no quedan residente en la memoria, pero sí incrementan el uso temporal de la memoria SRAM, la cual tiene un límite determinado durante toda la vida del programa. Es decir, que en este momento estaríamos utilizando unos 594 bytes.

Para continuar con el cálculo, volvemos al método procesarPetición de la clase RESTWebServer, y nos centramos en la siguientes líneas:

```
    if (res){
        chorService = _serviceCall->getService(this->entradas[0],
                this->_GET, this->cnt_get);
        chorService->ejecutar(this->entradas[1],
                this->entradas[2],pServer);
    }
```

Es decir que ahora, en caso de que el verbo solicitado sea un GET, debemos analizar el código getService, lo cual nos devuelve un objeto que se lo asignamos al puntero chorService.

```
choreographyService *MyAPI::getService(char *nombreClase,
          char *pRequest[], int cnt_req)
{
  if (strcmp(nombreClase,"balizarole") == 0){
    return new BalizaRole(pRequest,cnt_req);
  }
}
```

En este punto la memoria de variables locales se incrementa en aproximadamente en 1 byte

```
  2bytes  --------------------> cnt_req
  10bytes --------------------> distintas comparaciones
                del tipo 'balizarole'
```

Y la memoria heap en aproximadamente en 16 bytes

```
  2bytes ------------------> *nombreClase
  2bytes ------------------> *pRequest
```

Esto incrementaría en unos 16 bytes la memoria que se consume en este momento, tomando un tope de 610 bytes. y luego analizamos el método ejecutar

```
void choreographyService::ejecutar(char *method,
    char *datosIncidente, connection *pServer)
{
  char *respuesta, *sRespuestaPeticion;
```

```
  if (strcmp(method,"informarIncidente") == 0){
  respuesta = this->informarIncidente(method,datosIncidente);
  pServer->enviarRespuestaHTTP(respuesta,_token);
  }
  for (int i=0; i < this->_limite_siguiente; i++){
    char sPeticion[512];

    // Armo la uri del servicio a llamar
    strcpy(sPeticion,"/accidente/");
    strcat(sPeticion,chor_sig_role[i]);
    strcat(sPeticion,"/");
    strcat(sPeticion,chor_sig_operacion[i]);
    strcat(sPeticion,"/");
    strcat(sPeticion,datosIncidente);
    strcat(sPeticion,"?chor=accidente&channel=");
    strcat(sPeticion,chor_sig_channel[i]);
    strcat(sPeticion,"&relation=");
    strcat(sPeticion,chor_sig_relation[i]);
    strcat(sPeticion,"&token=");
    strcat(sPeticion,_token);
    pServer->realizarPeticionHTTP(chor_sig_ip[i],sPeticion,80);
  }
  return;
}
```

En este punto la memoria de variables locales se incrementa en aproximadamente en 1 byte

```
  2bytes  --------------------> cnt_req
  10bytes --------------------> distintas comparaciones
                del tipo 'balizarole'
```

Y la memoria heap en aproximadamente en 16 bytes

```
  2bytes ------------------> *method
  2bytes ------------------> *datosIncidente
  2bytes ------------------> *pServer
```

Aquí tenemos un caso especial donde los punteros respuesta y sRespuestaPeticion, inicialmente ocupan 2 bytes cada uno, pero las cadenas a las que apuntan son de carácter variable, pero puede llegar a ocupar varios bytes, sumado a los literales que se van concatenando. Suponiendo que las respuestas ocupen unos 200 bytes, en este proceso la memoria utilizada se eleva considerablemente, llegando en casos a 1024 bytes, sumado a lo que ya se encuentra en memoria estamos en los límites de los 3kbytes.

Ahora analizamos el llamado al método de enviarRespuestaHTTP, correspondiente al objeto connectionESP8266:

```
void connectionESP8266::enviarRespuestaHTTP(char *pRespuesta,
        int token)
{
  StaticJsonBuffer<50> jsonBuffer;

  JsonObject& root = jsonBuffer.createObject();
  root["resultado"] = pRespuesta;
  root["token"] = token;

  if (this->_client){
    this->_client.print(F(
     "HTTP/1.1 200 OK\r\n"
       "Content-Type: application/json\r\n"
     "Content-Length: "));
    this->_client.print(root.measureLength());
    this->_client.print(F("\r\n"
            "Connection: close\r\n"
            "\r\n"));
    root.printTo(this->_client);

    delay(15);

    this->_client.stop();
  }
}
```

donde utilizamos un objeto del tipo JSON para la respuesta del servicio REST que se implementa. Aquí, podemos observar que se hace una reserva de 50 bytes y luego existen literales, los cuales han podido ser elevados a la memoria EPROM (*Erasable Programmable Read-Only Memory*, ROM programable borrable) de la placa, lo cual no utiliza memoria SRAM.

Ahora analizamos el llamado al método de realizarPeticionHTTP, correspondiente al objeto connectionESP8266:

```
void connectionESP8266::realizarPeticionHTTP(char *pServer,
          char *pPeticion, int port)
{
  char sPedido[256], sHost[256];
  WiFiEspClient client;
  unsigned long time;

  client.stop();

  if (client.connect(pServer, port)) {

    strcpy(sPedido,"GET ");
    strcat(sPedido,pPeticion);
    strcat(sPedido," HTTP/1.1");
    strcpy(sHost,"Host: ");
    strcat(sHost,pServer);
    Serial.println(sPedido);
    Serial.println(sHost);
    client.println(sPedido);
    client.println(sHost);
    client.println(F("Connection: close"));
    client.println();
    time = millis();
    while(!client.available() && (millis() - time) < 3000);
    while (client.available()) {
      char c = client.read();
    }
    delay(15);
  }
}
```

Donde podemos analizar que el consumo de memoria es bastante elevado ya que tenemos dos variables locales declaradas de arreglos de caracteres de 256 bytes cada una, sumado al cliente Wifi, y los literales del tipo "GET", "HTTP", etc.

A su vez se debe sumar a todo este proceso las referencias y almacenamientos a puntos de retorno, puntos de próxima instrucción a ejecutarse, etc, las cuales son concernientes a los lenguajes de programación y suelen ocupar un lugar importante en este tipo de placas con escasa memoria disponible. Esto incrementaría en unos 16 bytes la memoria que se consume en este momento, tomando un tope de 610 bytes.

Por todo lo expuesto, podemos determinar que la cantidad de memoria mínima necesaria para la ejecución de este framework es de 4 kbytes.

# Diagramas de clases y código relacionado

## .1. Diagramas de clases en PHP

A continuación se presentará los diagramas de clase en lenguaje PHP, que representan la forma en que se han organizado las clases para la implementación del framework de ejecución de coreografías. A continuación se mostrará el código más relevante de dicha jerarquía.

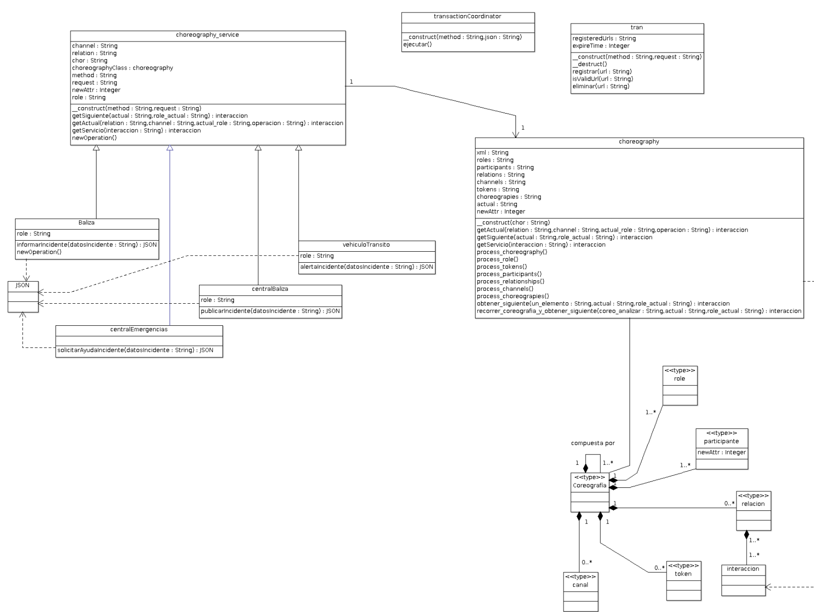


Figura 1: Diagrama de clases en PHP

El código que se muestra a continuación pertenece a la clase **choreography**, y pertenece a los métodos que se encargan de controlar la ejecución del framework.

```
/*
 * Obtiene el elemento inicial de la coreografía, para poder saber qué debe ejecutar
 */
public function getInicial($actual_role){
  $interaccion_inicial = array();

  foreach($this->xml->choreography as $fila){
    if ((string)$fila['root'] == "true"){
      foreach($fila->relationship as $relacion){

        // Si hay secuencias definidas en la coreografía,
        // busco a ver si encuentro la interacción correspondiente
        if (isset($fila->sequence)){
          foreach($fila->sequence as $secuencia){
            // Dentro de cada secuencia recorro todas las interacciones que tiene definidas
            foreach($secuencia->interaction as $interaccion){
              foreach($interaccion as $participantes){
                if (substr((string)$participantes['fromRole'],4) == $actual_role){
                  $interaccion_inicial['sequence'][] = $interaccion;
                }
              }
            }
          }
        }

        // Si hay paralelas definidas en la coreografía,
        // busco a ver si encuentro la interacción correspondiente
        if (isset($fila->parallel)){
          foreach($fila->parallel as $paralela){
            // Dentro de cada secuencia recorro todas las interacciones que tiene definidas
            foreach($paralela->interaction as $interaccion){
              foreach($interaccion as $participantes){
                if (substr((string)$participantes['fromRole'],4) == $actual_role){
                  $interaccion_inicial['parallel'][] = $interaccion;
                }
              }
            }
          }
        } // De todos los parallel


        // Si hay choices definidas en la coreografía,
        // busco a ver si encuentro la interacción correspondiente
        if (isset($fila->choice)){
          foreach($fila->choice as $elegida){
            // Dentro de cada secuencia recorro todas las interacciones que tiene definidas
            foreach($elegida->interaction as $interaccion){
              foreach($interaccion as $participantes){
                if (substr((string)$participantes['fromRole'],4) == $actual_role){
```

```
                  $interaccion_inicial['choice'][] = $interaccion;
                }
              }
            }
          }
        } // De todos los choices

        // Si hay workunits definidas en la coreografía,
        // busco a ver si encuentro la interacción correspondiente
        if (isset($fila->workunit)){
          foreach($fila->workunit as $elegida){
            // Dentro de cada work recorro todas las interacciones que tiene definidas
            foreach($elegida->interaction as $interaccion){
              foreach($interaccion as $participantes){
                if (substr((string)$participantes['fromRole'],4) == $actual_role){
                  $interaccion_inicial['workunit'][] = $interaccion;
                }
              }
            }
          }
        } // De todos los workunits

      } // Bucle de relationships
    } // If coreografía es root
  } // Choreography cycle
  return $interaccion_inicial;
} // Function

/*
 * Obtiene el elemento actual que corresponde a la coreografía,
 * es decir qué relación estamos implementenado
 */
public function getActual($relation,$channel,$actual_role,$operacion){
  foreach($this->xml->choreography as $fila){
    $encontrado = false;
    foreach($fila->relationship as $relacion){
      if (substr((string)$relacion['type'],4) == $relation){
        $encontrado = true;
      }

      // Si la relación está implementada en esta coreografía
      // entonces busco adentro de las secuencias, parallel, worksunits etc.
      if ($encontrado){

        // Si hay secuencias definidas en la coreografía,
        // busco a ver si encuentro la interacción correspondiente
        if (isset($fila->sequence)){
          foreach($fila->sequence as $secuencia){
            // Dentro de cada secuencia recorro todas las interacciones que tiene definidas
            foreach($secuencia->interaction as $interaccion){
```

```
      if ($interaccion['operation'] == $operacion){
        foreach($interaccion as $participantes){
          if (substr((string)$participantes['toRole'],4) == $actual_role){
            $this->actual['choreography'] = $this->choreograpies[(string)$fila['name']];
            $this->actual['interaction'] = $interaccion;
            $this->actual['struct'] = 'sequence';
            $this->actual['operacion'] = $operacion;
            return $this->actual;
          }
        }
      }
    }
  }
}

// Si hay paralelas definidas en la coreografía,
// busco a ver si encuentro la interacción correspondiente
if (isset($fila->parallel)){
  foreach($fila->parallel as $paralela){
    // Dentro de cada secuencia recorro todas las interacciones que tiene definidas
    foreach($paralela->interaction as $interaccion){
      if ($interaccion['operation'] == $operacion){
        foreach($interaccion as $participantes){
          if (substr((string)$participantes['toRole'],4) == $actual_role){
            $this->actual['choreography'] = $this->choreograpies[(string)$fila['name']];
            $this->actual['interaction'] = $interaccion;
            $this->actual['struct'] = 'parallel';
            $this->actual['operacion'] = $operacion;
            return $this->actual;
          }
        }
      }
    }
  }
} // De todos los parallel


// Si hay choices definidas en la coreografía,
// busco a ver si encuentro la interacción correspondiente
if (isset($fila->choice)){
  foreach($fila->choice as $elegida){
    // Dentro de cada secuencia recorro todas las interacciones que tiene definidas
    foreach($elegida->interaction as $interaccion){
      if ($interaccion['operation'] == $operacion){
        foreach($interaccion as $participantes){
          if (substr((string)$participantes['toRole'],4) == $actual_role){
            $this->actual['choreography'] = $this->choreograpies[(string)$fila['name']];
            $this->actual['interaction'] = $interaccion;
            $this->actual['struct'] = 'choice';
            $this->actual['operacion'] = $operacion;
```

```
                    return $this->actual;
                  }
                }
              }
            }
          }
        } // De todos los choices

        // Si hay workunits definidas en la coreografía,
        // busco a ver si encuentro la interacción correspondiente
        if (isset($fila->workunit)){
          foreach($fila->workunit as $elegida){
            // Dentro de cada work recorro todas las interacciones que tiene definidas
            foreach($elegida->interaction as $interaccion){
              if ($interaccion['operation'] == $operacion){
                foreach($interaccion as $participantes){
                  if (substr((string)$participantes['toRole'],4) == $actual_role){
                    $this->actual['choreography'] = $this->choreograpies[(string)$fila['name']];
                    $this->actual['interaction'] = $interaccion;
                    $this->actual['struct'] = 'workunit';
                    $this->actual['operacion'] = $operacion;
                    return $this->actual;
                  }
                }
              }
            }
          }
        } // De todos los parallel

      } // If encontró relation
    } // Bucle de relationships
  } // Choreography cycle
} // Function

/*
 * Obtiene el elemento siguiente para ser llamado,
 * si es que existiera...
 */
public function getSiguiente($actual,$role_actual){
  // Saco el nombre de la interacción actual
  $interaccion_actual = get_object_vars($actual['interaction']);

  // Primero saco la coreografía correspondiente ...
  $coreografia = get_object_vars($actual['choreography']);

  // Inicializo variables
  $siguiente = false;
  $interaccion_siguiente = [];

  // Recorro la coreografía actual
```

```
foreach($coreografia as $elemento => $valor){
  if ($elemento == 'sequence' || $elemento == 'parallel' || $elemento == 'workunit'){
    $keys = array_keys(get_object_vars($valor));
    $arreglo_elementos = get_object_vars($valor);

    // Recorro todos los elementos en orden de aparición en la coreografía
    foreach($valor as $key => $value){
      $un_elemento = get_object_vars($value);
      if (!$siguiente){
        if ($un_elemento['@attributes']['name'] == $interaccion_actual['@attributes']['name']){
          $siguiente = true;
        }
      }else{
        switch (strtolower($elemento)){
          case "sequence":
            if ($key == "perform"){
              $coreo_analizar = substr($un_elemento['@attributes']['choreographyName'],4);

              // Recorro la coreografía para saber cuál o cuáles son los siguientes pasos
              $resultado_s = $this->recorrer_coreografia_y_obtener_siguiente(
                        $this->choreograpies[$coreo_analizar],$actual,$role_actual);
              foreach($resultado_s as $resul => $resul_value){
                $interaccion_siguiente[$resul] = $resultado_s[$resul];
              }
            }
            if ($key == "interaction"){
              // Envio a extraer el siguiente de la interaccion
              $temp = $this->obtener_siguiente($un_elemento,$actual,$role_actual);
              if (count($temp) > 0)
                $interaccion_siguiente['sequence'][] = $temp;
            }
            break;
          case "parallel":
            if ($key == "perform"){
              $coreo_analizar = substr($un_elemento['@attributes']['choreographyName'],4);

              // Recorro la coreografía para saber cuál o cuáles son los siguientes pasos
              $resultado_s = $this->recorrer_coreografia_y_obtener_siguiente(
                        $this->choreograpies[$coreo_analizar],$actual,$role_actual);
              foreach($resultado_s as $resul => $resul_value){
                $interaccion_siguiente[$resul] = $resultado_s[$resul];
              }
            }
            if ($key == "interaction"){
              // Envio a extraer el siguiente de la interaccion
              $temp = $this->obtener_siguiente($un_elemento,$actual,$role_actual);
              if (count($temp) > 0)
                $interaccion_siguiente['sequence'][] = $temp;
            }
            break;
```

```
            case "workunit":
              if ($key == "perform"){
                $coreo_analizar = substr($un_elemento['@attributes']['choreographyName'],4);

                // Recorro la coreografía para saber cuál o cuáles son los siguientes pasos
                $resultado_s = $this->recorrer_coreografia_y_obtener_siguiente(
                            $this->choreograpies[$coreo_analizar],$actual,$role_actual);
                foreach($resultado_s as $resul => $resul_value){
                  $interaccion_siguiente[$resul] = $resultado_s[$resul];
                }
              }
              if ($key == "interaction"){
                // Envio a extraer el siguiente de la interaccion
                $temp = $this->obtener_siguiente($un_elemento,$actual,$role_actual);
                if (count($temp) > 0)
                  $interaccion_siguiente['sequence'][] = $temp;
              }
              break;
          }

        }
      }

    }
  }

  // Ahora recorro todas las coreografías para saber
  // si hay alguna que tenga un workunit que se deba ejecutar
  foreach($this->xml->choreography as $fila){
    // Dentro de cada coreografía recorro todas los workunit que tiene definidas
    if (isset($fila->workunit)){
      foreach($fila->workunit as $unidad){
        // Si se cumple la condición de guarda


        // Dentro de cada WorkUnit recorro todas las interacciones que tiene definidas
        foreach($unidad->interaction as $interaccion){
          $fromRole = $interaccion->participate['fromRole'];
          $fromRole = strval(substr($fromRole,(strpos($fromRole,":")+1)));
          if ($fromRole == $role_actual)
            $interaccion_siguiente['workunit'][] = $interaccion;
        }
      }
    }
  }

  return $interaccion_siguiente;

}
```

El código que se muestra a continuación pertenece a las clases **transactionCoordinator** y **tran**.

```php
class transactionCoordinator
{
  protected $method;
  protected $request;
  protected $json;

  function __construct($method, $json){
    $this->method = $method;
    $this->json = $json;

  }

  function __destruct() {
  }

  public function ejecutar() {
      foreach($this->json as $reg){
        // Control el expires time y ejecuto
        if ($reg['expires'] >= time()){
          $resultado = "";
          $i = 0;
          $resultado = CallAPI("PUT",$reg['datos']['uri'],false);
          while (($resultado['estado'] != 204) && ($i<10)){
            file_put_contents(__DIR__ . "/../coordinator.log","paso" . $i,FILE_APPEND);
            $i++;
          }
        }
      }
  }

}//Fin de la clase

class tran
{
  protected $method;
  protected $request;
  protected $registeredUrls;
  protected $expireTime = 0; // 0 Unlimited time (expressed in minutes)

  function __construct($method,$request){
      $this->request = $request;
      $this->method = $method;
      if (file_exists(__DIR__ . '/../registeredUrls.json')){
        $json = file_get_contents(__DIR__ . '/../registeredUrls.json');
        $this->registeredUrls = json_decode($json,true);
      }else{
        $this->registeredUrls = [];
      }
      $this->expireTime = 60;
```

```
      // Limpiar los urls que ya no son válidos ...
      foreach($this->registeredUrls['transaction'] as $reg => $valor){
        if (($this->registeredUrls['transaction'][$reg]['expires'] < time())){
          unset($this->registeredUrls['transaction'][$reg]);
        }
      }
  }

  function __destruct() {
    // Serializo el json con las urls registradas ...
    file_put_contents(__DIR__ . "/../registeredUrls.json",json_encode($this->registeredUrls));
  }

  public function registrar ($url) {
    $registro = array("uri" => $url, "expires" => time() + ($this->expireTime * 60));
    $this->registeredUrls['transaction'][] = $registro;
    return $registro;
  }

  public function isValidUrl($url) {
    $valido = false;
    foreach($this->registeredUrls['transaction'] as $reg){
      if (($reg['uri'] == $url) && ($reg['expires'] >= time())){
        $valido = true;
      }
    }
    return $valido;
  }

  public function eliminar($url) {
    $indice = -1;
    foreach($this->registeredUrls['transaction'] as $reg => $valor){
      if (trim($this->registeredUrls['transaction'][$reg]['uri']) == trim($url)){
        $indice = $reg;
      }
    }
    if ($indice != -1){
      unset($this->registeredUrls['transaction'][$indice]);
    }
    return;
  }
}
```

## .2. Diagramas de clases en C++

A continuación se presentará los diagramas de clase en lenguaje C++, que representan la forma en que se han organizado las clases para la implementación del framework de ejecución de coreografías. A continuación se mostrará el código más relevante de dicha jerarquía.

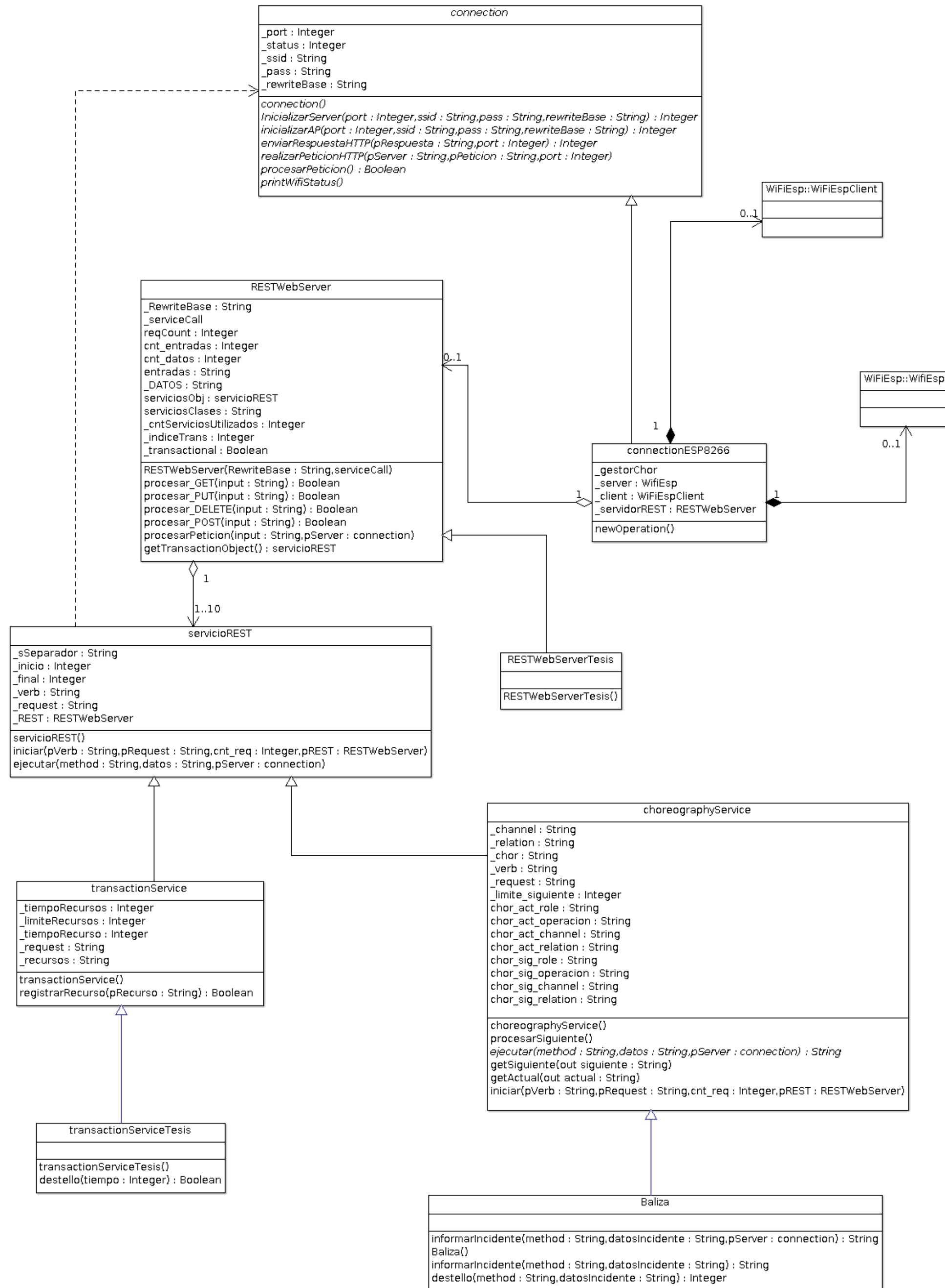


Figura 2: Diagrama de clases para el manejo de transacciones en REST

El código que se muestra a continuación pertenece a las clases más relevantes del framework que se ejecuta en los dispositivos Arduino.

```
#include <string_ext.h>
#include <Baliza.h>
#include <string.h>
#include <Arduino.h>
#include <connection.h>
#include <choreographyService.h>

choreographyService::choreographyService() : servicioREST()
{
}

int choreographyService::iniciar(char pVerb, char *pRequest[], int cnt_req,RESTWebServer *pREST)
{
    // En request viene un arreglo con los parámetros GET de la operación
    // El orden es el siguiente:
    // Posicion 0: coreografía
    // Posicion 1: channel
    // Posicion 2: relation
    // Método: nombre del método de la clase a ejecutar
    bool flag=false;
    int j=0, res;
  int comienzo = 0;
  char *tmp;

  // Llamo al ancestro ...
  res = servicioREST::iniciar(pVerb,pRequest,cnt_req,pREST);

  if (res != 1)
    return -1; // Error del ancestro...

  // Recorro el arreglo de datos que vinieron ...
    for(int i=0; i<=cnt_req; i++){
        if (existe_subcadena(pRequest[i],"chor=") || existe_subcadena(pRequest[i],"\"chor\":")){
      comienzo = 4 + this->_inicio;
      tmp = pRequest[i] + comienzo;
      strncpy(this->_chor,tmp, (strlen(tmp) - 1));
      this->_chor[(strlen(tmp) - 1)] = '\0';
            flag = true;
        }
        if (existe_subcadena(pRequest[i],"channel=") || existe_subcadena(pRequest[i],"\"channel\":")){
      comienzo = 7 + this->_inicio;
      tmp = pRequest[i] + comienzo;
      strncpy(this->_channel,tmp, (strlen(tmp) - 1));
      this->_channel[(strlen(tmp) - 1)] = '\0';
            flag = true;
        }
        if (existe_subcadena(pRequest[i],"relation=") || existe_subcadena(pRequest[i],"\"relation\":")){
      comienzo = 8 + this->_inicio;
      tmp = pRequest[i] + comienzo;
      strncpy(this->_relation,tmp, (strlen(tmp) - 1));
```

```
      this->_relation[(strlen(tmp) - 1)] = '\0';
            flag = true;
        }
        if (existe_subcadena(pRequest[i],"token=") || existe_subcadena(pRequest[i],"\"token\":")){
      comienzo = 5 + (this->_inicio - 1);
      tmp = pRequest[i] + comienzo;
      if ((strcmp(tmp,this->_token) == 0) && !this->_utilizable){
        Serial1.println("Coreografía inutilizable aún");
        return -2; // Error de coreografía inutilizable por error de timeout o algo anterior
      }
      this->_utilizable = true;
      strcpy(this->_token,tmp);
      this->_token_set = true;
            flag = true;
        }
        if (!flag){
            this->_request[j] = pRequest[i];
            flag = false;
            j++;
        }
    }

  if (!_token_set){
    ltoa(random(1000),this->_token,10);
  }

    // Verifico que hayan venido los datos de la coreografía
    if ((this->_chor[0] == '\0') || (this->_channel[0] == '\0') || (this->_relation[0] == '\0'))
        return -3;  // Error en la recepción de los parámetros, no se puede seguir con la ejecución

  return 1;
}

void choreographyService::getActual(char *actual[4]){

  actual[0] = chor_act_role;
  actual[1] = chor_act_operacion;
  actual[2] = chor_act_channel;
  actual[3] = chor_act_relation;

}

void choreographyService::getSiguiente(char *siguiente[10][4]){

  for(int i=0;i < _limite_siguiente; i++){
    siguiente[i][0] = chor_sig_role[i];
    siguiente[i][1] = chor_sig_operacion[i];
    siguiente[i][2] = chor_sig_channel[i];
    siguiente[i][3] = chor_sig_relation[i];
  }
```

```
  return;

}

void choreographyService::ejecutar(char *method, char *datosIncidente, connection *pServer)
{
  char sPeticion[512];
  char sJson[128];
  char resultado[3][100];

  // Recorro la coreografía para ejecutar lo que corresponda...
  for (int i=0; i < this->_limite_siguiente; i++){

    // Registro las llamadas
    strcpy(sPeticion,"/accidente/bitacora/agregar/datos?chor=");
    strcat(sPeticion,this->_chor);
    strcat(sPeticion,"&sd=");
    strcat(sPeticion,chor_act_role);
    strcat(sPeticion,"&md=");
    strcat(sPeticion,chor_act_operacion);
    strcat(sPeticion,"&sh=");
    strcat(sPeticion,chor_sig_role[i]);
    strcat(sPeticion,"&mh=");
    strcat(sPeticion,chor_sig_operacion[i]);
    strcat(sPeticion,"&disp=");
    strcat(sPeticion,pServer->urlencode("server-griisa").c_str());
    strcat(sPeticion,"&token=");
    strcat(sPeticion,_token);
    strcat(sPeticion,"&ip=170.210.127.69");
    strcat(sPeticion,"&peticion=");
    strcat(sPeticion,pServer->urlencode("http://").c_str());
    strcat(sPeticion,pServer->urlencode(chor_sig_ip[i]).c_str());
    //strcat(sPeticion,pServer->urlencode(sPeticion).c_str());
    pServer->realizarPeticionHTTP('G',"170.210.127.69",sPeticion,80, resultado, atoi(chor_sig_timeout[i]));

    // Armo la uri del servicio a llamar
    trim(datosIncidente);
    strcpy(sPeticion,"/accidente/");
    strcat(sPeticion,chor_sig_role[i]);
    strcat(sPeticion,"/");
    strcat(sPeticion,chor_sig_operacion[i]);
    strcat(sPeticion,"/");
    strcat(sPeticion,datosIncidente);

    // Armo el json
    strcpy(sJson,"{\"chor\":\"");
    strcat(sJson,this->_chor);
    strcat(sJson,"\",\"channel\":\"");
    strcat(sJson,chor_sig_channel[i]);
    strcat(sJson,"\",\"relation\":\"");
```

```
strcat(sJson,chor_sig_relation[i]);
strcat(sJson,"\",\"token\":");
strcat(sJson,this->_token);
strcat(sJson,"}");

// Hago el llamado del servicio propiamente dicho
pServer->realizarPeticionHTTP('O',chor_sig_ip[i],sPeticion,80,sJson,resultado,atoi(chor_sig_timeout[i]));

// De acuerdo al código verifico qué pasó y cómo sigue
switch (atoi(resultado[0])){
  case -1:
      // Verifico si tiene reemplazo en la coreografía...
      if (chor_sig_ip_clone[i][0] != '\0'){
        Serial1.println("adentro por timeout");
        // Hago el llamado del servicio propiamente dicho
        pServer->realizarPeticionHTTP('O',chor_sig_ip_clone[i],sPeticion,80,sJson,resultado,atoi(chor_sig_timeout[i]));
        if (atoi(resultado[0]) != 200){
          Serial1.println("ChoreographyService: Timeout");
          this->_utilizable = false;
          break;
        }else{
          Serial1.println("ChoreographyService: OK");
          Serial1.print("ChoreographyService Datos: ");
          Serial1.println(resultado[2]);
          this->_utilizable = true;
          break;
        }
      }else{
        Serial1.println("ChoreographyService: Timeout");
        this->_utilizable = false;
        break;
      }
  case -2:
      Serial1.println("ChoreographyService: No conectado");
      this->_utilizable = false;
      break;
  case 200:
      Serial1.println("ChoreographyService: OK");
      Serial1.print("ChoreographyService Datos: ");
      Serial1.println(resultado[2]);
      this->_utilizable = true;
      break;
  case 404:
      Serial1.println("ChoreographyService: 404");
      break;
  default:
      Serial1.println(resultado[0]);
      Serial1.println(resultado[1]);
      break;
}
```

```
  }

  return;
}


#include <Arduino.h>
#include <connection.h>
#include <servicioREST.h>
#include <RESTWebServer.h>

servicioREST::servicioREST()
{
}

int servicioREST::iniciar(char pVerb, char *pRequest[], int cnt_req,RESTWebServer *pREST)
{
    this->_verb = pVerb;
  this->_REST = pREST;
  if (this->_verb == 'G'){
    strcpy(this->_sSeparador,"=");
    this->_inicio = 1;
    this->_final = 0;
  }else{
    strcpy(this->_sSeparador,":");
    this->_inicio = 4;
    this->_final = 1;
  }
    for(int i=0; i<=cnt_req; i++){
    this->_request[i] = pRequest[i];
  }

  return 1;
}

#include <string_ext.h>
#include <string.h>
#include <Arduino.h>
#include <connection.h>
#include <transactionService.h>

transactionService::transactionService() : servicioREST()
{
}

int transactionService::iniciar(char pVerb, char *pRequest[], int cnt_req,RESTWebServer *pREST)
{
  return servicioREST::iniciar(pVerb,pRequest,cnt_req,pREST);
}
```

```
void transactionService::ejecutar(char *method, char *datosIncidente, connection *pServer)
{

  char *respuesta, *sRespuestaPeticion;

  return respuesta;

}

bool transactionService::registrarRecurso(char *pRecurso){
  if (this->_limiteRecursos == SIZE_RECURSOS){
    return false;
  }else{
    Serial1.print("Recurso a registrar: ");
    Serial1.println(pRecurso);
    strcpy(this->_recursos[this->_limiteRecursos],pRecurso);
    //this->_recursos[this->_limiteRecursos] = pRecurso;
    Serial1.println(this->_recursos[this->_limiteRecursos]);
    this->_tiempoRecursos[this->_limiteRecursos] = millis();
    this->_limiteRecursos++;
  }
  return true;
}
```

# Instalación del Framework

En este anexo presentaremos una especie de Manual de Instalación del framework para su correcta ejecución. La instalación consta de dos partes, por un lado la instalación del framework para la utilización con PHP, y por otro lado la instalación en Arduino.

## .3. Instalación en PHP

Esta librería permite la ejecución de coreografías. Tiene un motor o framework base (clase choreography_service y choreography) que permite derivar las clases que implementan los servicios que luego serán invocados por la coreografía.

La característica más importante de esta librería es que permite la lectura y ejecución totalmente autónoma de la coreografía que se quiere implementar. En este caso particular se entrega la librería con las clases base y con una implementación de ejemplo, denominada accidente.xml.

Se debe descomprimir el archivo .zip en una carpeta accesible por el servidor web (se ha probado hasta el momento con Apache y Lighttpd). La siguiente guía de instalación está basada en un servidor web Apache.

La librería es compatible con PHP 5.x o mayor.

Como este es un proyecto basado en servicios web REST, se debe tener activado en Apache el modulo de rewrite activado. Para ello se debe seguir las siguientes instrucciones:

- Linux

  Desde la consola bastará con introducir el siguiente comando:

  ```
  sudo a2enmod rewrite
  sudo /etc/init.d/apache2 restart
  ```

  A continuación editamos el archivo /etc/apache2/sites-enabled/000-default y buscamos la línea AllowOverride None y la cambiamos por AllowOverride All

  Luego reiniciamos el servidor apache.

- Windows

En Windows debemos modificar el archivo httpd.conf que dependiendo del servidor que hayamos instalado su ubicación será diferente. En este caso la instalación que se tiene es con appserv. Su ubicación es C:/AppServ/Apache2.2/conf/httpd.conf. En este archivo buscamos la linea

```
LoadModule rewrite_module modules/mod_rewrite.so
```

y si tiene el carácter # es porque está comentado, borramos ese carácter. Después buscamos esta sección:

```
    Options FollowSymLinks ExecCGI Indexes
    AllowOverride All
    Order deny,allow
    Deny from all
    Satisfy all
```

Y debemos poner AllowOverride All

Si la versión de PHP instalada es menor a 7.0, necesitamos activar la librería JSON. Para ello realizamos los siguientes pasos:

- Linux

```
    sudo aptitude install php5-json
```

- Windows

  Se debe descargar la librería php_json.dll, ubicarla en la carpeta de extensiones de php y luego habilitarla en php.ini

También se debe habilitar la librería curl, que permite la ejecución de servicios en otros servidores.

- Linux

```
    sudo apt-get install php5-curl
```

- Windows

  Descargar la librería desde esta dirección `https://curl.haxx.se/dlwiz/?type=bin`. Luego habilitar la extensión en el archivo php.ini.

Se deberán tener activas las extensiones de xml, postgresql, gd. Para ello desde Linux se ejecuta el siguiente comando

```
    sudo apt-get install php5-mcrypt php5-cli php5-xml php5-zip php5-pgsql php5-
        gd php5-xmlrpc
```

En este caso se utilizó la versión 5 de PHP, pero en caso de ser otra se debe cambiar por la versión correspondiente.

A continuación se muestra una configuración para el sitio, el cual puede ser luego incluida en el archivo 000-default.conf de Apache

```
    Alias /accidente "/home/oscar/var/websites/Tesis/Accidente/"
    <Directory "/home/oscar/var/websites/Tesis/Accidente/">
      Options Indexes MultiViews
        Options FollowSymLinks
        AllowOverride All
          <IfModule !mod_authz_core.c>
                        Order allow,deny
                        Allow from all
                </IfModule>
                <IfModule mod_authz_core.c>
                        Require all granted
                </IfModule>
    </Directory>
```

Una vez descomprimido el archivo .zip, se debe acceder y modificar los siguiente archivos:

`https://github.com/GRISE-UPM/choreography/blob/master/.htaccess`: este archivo es el que contiene las reglas de rewrite de Apache para el manejo de llamadas REST (en este caso). El archivo tiene el siguiente contenido:

```
RewriteEngine On
RewriteBase /accidente/
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ api.php?request=$1 [QSA,NC,L]
```

donde como vemos aquí se debe cambiar /**accidente**/ por la dirección correspondiente al servicio que se desea llamar.

El resto de los archivos que se deben modificar corresponden al ejemplo de ejecución de coreografía que se agrega a este framework, accidente.xml.

- Baliza.php

  `https://github.com/GRISE-UPM/choreography/blob/master/Baliza.php`

- CentralBaliza.php

  `https://github.com/GRISE-UPM/choreography/blob/master/CentralBaliza.php`

- CentralEmergencias.php

  `https://github.com/GRISE-UPM/choreography/blob/master/CentralEmergencias.php`

- VehiculoAccidentado.php

  `https://github.com/GRISE-UPM/choreography/blob/master/VehiculoAccidentado.php`

- VehiculoTransito.php

  `https://github.com/GRISE-UPM/choreography/blob/master/VehiculoTransito.php`

En cualquiera de ellos se puede reemplazar el código para que realice otra funcionalidad a la que se expresa en ellos. También cada uno de ellos puede

servir como base para la realización e implementación de otras especificaciones de coreografías.

Para el correcto funcionamiento de la coreografía, es necesario establecer en el archivo configuracion_wsdl.ini las direcciones ip de cada uno de los participantes de la coreografía. Este archivo es ad-hoc en reemplazo de las descripciones en wsdl de cada uno de los servicios que aquí se presentan. El archivo tiene el siguiente contenido:

```
[accidente]
BalizaRole = 192.168.1.119
VehiculoAccidentadoRole = 192.168.1.102
CentralBalizasRole = 192.168.1.102
VehiculoTransitoRole = 192.168.1.102
CentralEmergenciasRole = 192.168.1.102
bitacora = 170.210.127.69
BalizaClone = 192.168.1.102
```

Como vemos, el nombre del Role que figura en el archivo de definición de la coreografía (en este caso accidente.xml) es el que debe figurar en cada línea de este archivo, junto a la dirección Ip en la cual se encuentra funcionando.

Es recomendable que en toda instalación de estas librerías, al ser un proyecto de desarrollo, es muy deseable que se habilite en PHP la posibilidad de visualizar los erroes. Esto se realiza habilitando la directiva display_error=true en el archivo php.ini.

## .4. Instalación en Arduino

Librerías para la utilización de un servidor REST basado en el módulo ESP8266-01 Esta librería permite el agregado de realizar un servidor REST al módulo WIFI que se le adicione al proyecto. Inicialmente soporta el módulo ESP8266-01, basado en la librería WiFiEsp.

- Permite configurar la URL base del servicio REST
- Permite utilizar (a futuro) más de un módulo de conectividad (Wifi, wired, bluetooth, etc)
- Realiza las llamadas a los servicios a partir de las carpetas especificadas, en modo de clases y objetos.

En este caso, para poder utilizarlas se deberá descargar y descomprimir el archivo .zip en la carpeta de librerías de Arduino.

Las librerías a descargar son las siguientes:

- ml_server_rest

  `https://github.com/GRISE-UPM/ml_server_rest`

- ml_string

  `https://github.com/GRISE-UPM/ml_string`

Una vez bajadas e instaladas las librerías en la carpeta correspondiente, se deben modificar el archivo Baliza.cpp. El código a sufrir modificaciones es el siguiente:

```
BalizaRole::BalizaRole() : choreographyService()
{
    chor_act_role = "BalizaRole";
    chor_act_operacion = "informarIncidente";
    chor_act_channel = "Baliza";
    chor_act_relation = "Vehiculo_Baliza";

    chor_sig_role[1] = "centralBalizasRole";
    chor_sig_operacion[1] = "publicarIncidente";
    chor_sig_channel[1] = "CentralBaliza";
    chor_sig_relation[1] = "Baliza_CentralBaliza";
    chor_sig_ip[1] = "192.168.1.102";
    chor_sig_ip_clone[1] = "192.168.1.132";
    chor_sig_timeout[1] = "5000";

    chor_sig_role[0] = "VehiculoTransitoRole";
    chor_sig_operacion[0] = "alertaIncidente";
    chor_sig_channel[0] = "Vehiculo";
    chor_sig_relation[0] = "Baliza_Vehiculo";
    chor_sig_ip[0] = "192.168.1.102";
    chor_sig_ip_clone[0] = "\0";
    chor_sig_timeout[0] = "5000";

    this->_limite_siguiente = 2;
    return;
}
```

En particular, se debe cambiar la dirección IP del siguiente Rol a ejecutar. En este caso, como en la coreografía desde el Rol Baliza se deben ejecutar dos servicios (centralBalizasRole y VehiculoTransitoRole) es que se deben modificar dos direcciones IP. La variable _limite_siguiente se establece en el valor 2, ya que son dos los roles a ejecutar, en caso de querer que se ejecute uno solo se debe modificar también la asignación a esta variable.

# Arquitectura de ejecución

Presentamos ahora un esquema arquitectónico de cómo se produce la ejecución del framework.

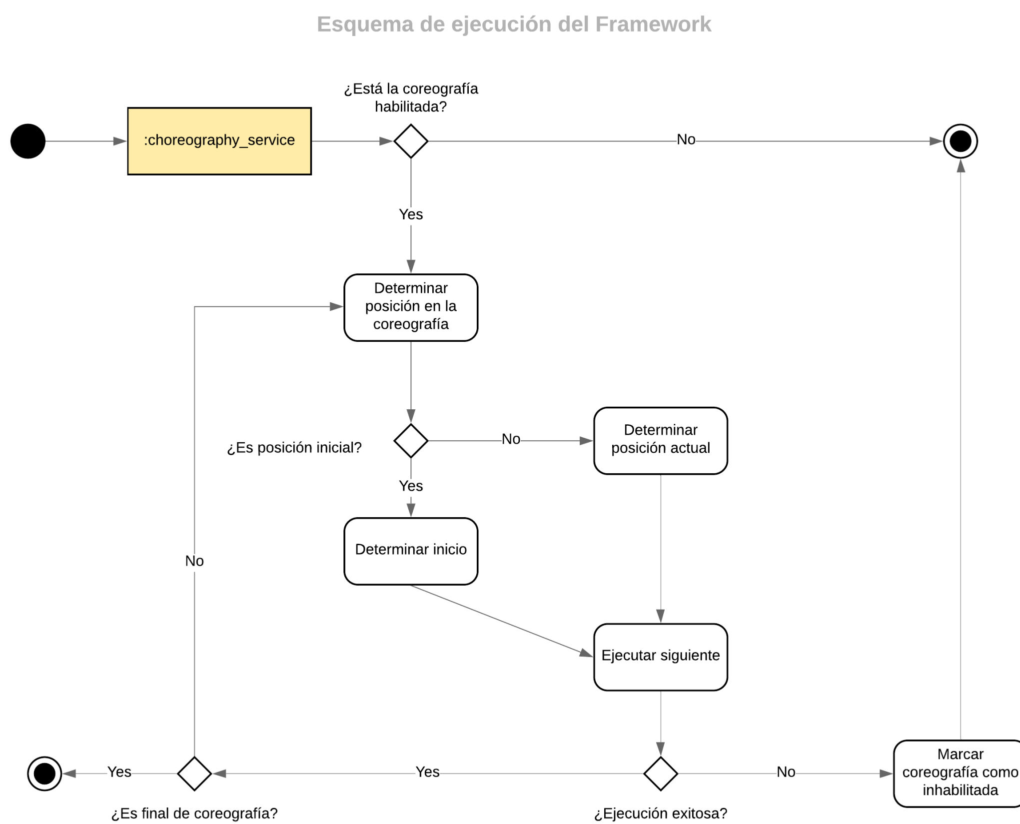


Figura 3: Diagrama de ejecución del Framework

En el diagrama podemos apreciar la manera en que se ejecuta el framework de coreografías que hemos desarrollado. Lo primero que se produce en la ejecución es la instanciación de la clase choreography_service, a partir de la cual se ejecuta el resto del framework. Esta ejecución es a modo general, en un gráfico siguiente se mostrará más en particular la tarea de **Ejecutar**

**siguiente**, donde se realizan algunas tareas adicionales, las cuales no fueron presentadas en este diagrama por simplificación y una mejor visualización del contexto general de ejecución. En el caso de que se produzcan errores en la ejecución, o en alguna parte de la misma, la coreografía se marca como inhabilitada (más específicamente la instancia actual) para que futuras llamadas con esta instancia no produzcan resultados o comportamientos no deseados.

Toda ejecución comienza con la tarea de determinar el paso actual donde se encuentra la ejecución (el que pertenece al dispositivo que está en ese punto de la ejecución) para luego determinar cuál es el paso siguiente y así sucesivamente hasta encontrar el final de la misma.

En el siguiente gráfico 4, donde se expone con mayor detalle la tarea de **Ejecutar siguiente**.

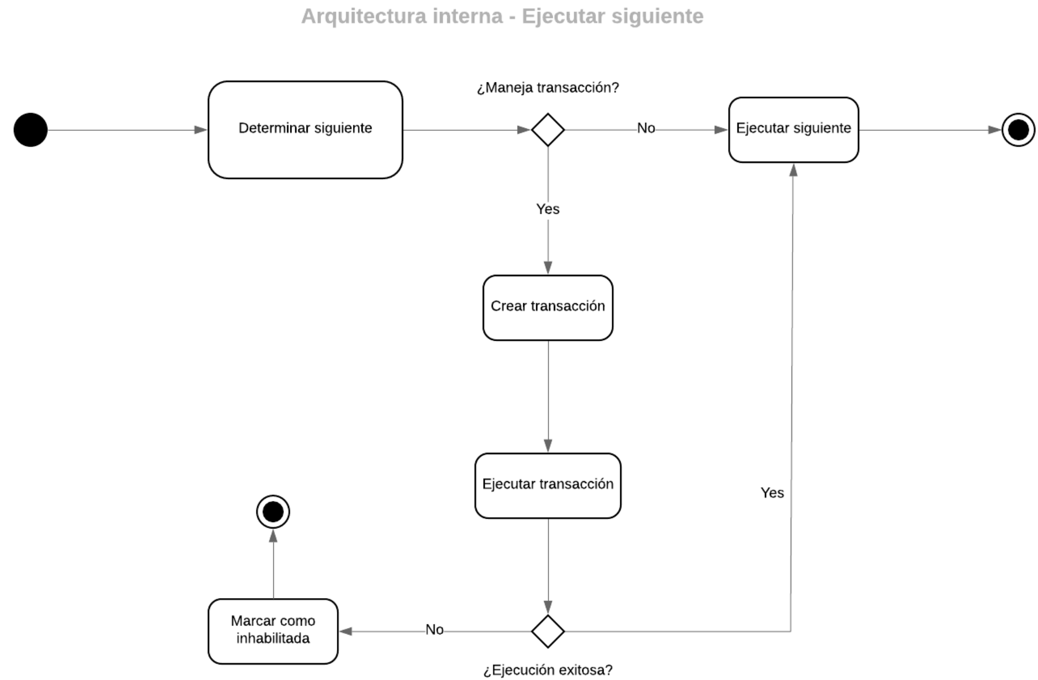


Figura 4: Diagrama en detalle

En este diagrama 4 se puede apreciar que la ejecución del paso siguiente de la coreografía contempla la posibilidad de que el dispositivo sea parte de una transacción en conjunto con otros dispositivos dentro de la coreografía. Para ello, evalúa dicha condición y en caso de existir, a su vez, contempla la posibilidad de que la ejecución de la misma sea exitosa, para lo cual puede realizar un rollback de la transacción e inhabilitar la coreografía.