



**UNIVERSIDAD NACIONAL DE LA PLATA
FACULTAD DE INFORMÁTICA**

**METODOLOGÍA BASADA EN MODELOS EN TIEMPO DE
EJECUCIÓN PARA LA CONSTRUCCIÓN Y OPERACIÓN
DE SISTEMAS AUTOCONSCIENTES
DE INTERNET DE LAS COSAS**

Tesis presentada para obtener el grado de Doctor en Ciencias
Informáticas

Autor: Lenin Xavier Erazo Garzón

Directora: Dra. Priscila Cedillo Orellana¹

Codirector: Dr. Gustavo Rossi²

Diciembre - 2024

¹ Facultad de Ingeniería. Universidad de Cuenca, Ecuador.

² Facultad de Informática. Universidad Nacional de La Plata, Argentina.

Dedicatoria

A mi esposa e hijos, por su cariño, comprensión y apoyo incondicional.

*A mis padres, por inculcarme que con disciplina, perseverancia,
humildad y determinación los sueños se hacen realidad.*

Agradecimientos

A Dios por ser mi guía y esperanza en el camino de la vida.

A mi Directora y Codirector de la tesis, Priscila y Gustavo, un agradecimiento especial por su orientación, motivación, apoyo, tiempo, enseñanzas y consejos valiosos que han sido decisivos para alcanzar los objetivos de este trabajo de investigación. Este agradecimiento será imperecedero como eternas serán sus enseñanzas.

A la Facultad de Informática de la Universidad Nacional de La Plata por abrirme las puertas para realizar mi doctorado en tan prestigiosa institución y brindarme la oportunidad de coincidir con extraordinarios profesionales durante mi formación.

A la Universidad del Azuay por la confianza depositada en mí persona y brindarme las facilidades y recursos necesarios para realizar mis estudios de doctorado, experiencia que ha enriquecido mi vida académica y profesional.

A los pasantes (Jaime, Nicolás) y tesistas (Adriana, José, Diego, Walter, Christian, Sergio, Alex, Brandon) de pregrado de la Universidad del Azuay y la Universidad de Cuenca que han colaborado en la implementación de las herramientas de software de la solución tecnológica propuesta en esta tesis doctoral. Asimismo, a todas las personas que han participado en las sesiones experimentales.

Resumen

El Internet de las Cosas (IoT) es una de las áreas de mayor crecimiento dentro de las tecnologías de información, debido a su incidencia técnica, social y económica en la sociedad, transformando favorablemente la forma en que vivimos y trabajamos. En la actualidad, uno de los desafíos más importantes para la comunidad científica y la industria es el desarrollo de software para sistemas IoT, ya que se caracterizan por ser altamente distribuidos, heterogéneos y dinámicos, siendo complejo identificar todos los requisitos del sistema en la etapa de desarrollo. Además, las metodologías y herramientas de ingeniería de software tradicionales han demostrado ser poco efectivas y productivas para afrontar las características intrínsecas de estos sistemas y, en particular el dinamismo y la incertidumbre asociados a los mismos. En sí, estos enfoques y herramientas se limitan a gestionar la evolución de los sistemas en tiempo de diseño, dejando de lado el tiempo de ejecución. Asimismo, se centran en los detalles de implementación (programación a bajo nivel) en lugar del dominio del problema, volviéndose una tarea compleja y tediosa el desarrollo de software para sistemas IoT.

Un enfoque apropiado es incorporar la autoconsciencia computacional a los sistemas IoT con el apoyo de la Ingeniería Dirigida por Modelos (MDE) y, particularmente los modelos en tiempo de ejecución (`models@run.time`), a fin de aumentar su autonomía y resiliencia frente a las situaciones imprevistas. Bajo este enfoque, un sistema IoT es considerado autoconsciente, si es capaz de obtener y mantener el conocimiento sobre sí mismo y el entorno (p. ej., estructura, estado, evolución), sin un control externo. Mientras que, los modelos en tiempo de ejecución constituyen una autorrepresentación causalmente conectada de un sistema IoT, siendo el medio a través del cual el sistema captura el conocimiento, es decir, es consciente.

Sin embargo, la autoconsciencia computacional en IoT es un área de investigación en desarrollo. Los esfuerzos de investigación se han concentrado fundamentalmente en trabajos teóricos orientados a establecer hojas ruta para la adopción de la autoconsciencia en sistemas IoT, así como soluciones prácticas sobre arquitecturas, frameworks, middlewares y algoritmos de aprendizaje automático para apoyar la autoconsciencia en aplicaciones particulares (ad hoc) de IoT, descuidando la concepción de metodologías e infraestructuras genéricas e integrales para construcción de este tipo sistemas. De igual manera, existen pocos estudios que consideran explícitamente a MDE y los modelos en tiempo de ejecución como parte de sus soluciones autoconscientes.

Frente a esta problemática, la presente tesis doctoral aborda las limitaciones y desafíos expuestos mediante la creación de `Aware-IoT@Run.Time`, un método de ingeniería de software basado en modelos en tiempo de ejecución que apoya a los stakeholders en la construcción, operación y mantenimiento de sistemas IoT autoconscientes. Específicamente, el proceso metodológico de `Aware-IoT@Run.Time` y sus actividades están fundamentados en el bucle de aprendizaje y razonamiento basado en modelos (LRA-M loop). Este proceso metodológico ha sido descrito a profundidad y representado gráficamente bajo la notación SPEM, incluyendo su descomposición jerárquica en actividades, tareas y subtareas, junto con la especificación de los roles de usuario, herramientas y artefactos de entrada / salida que intervienen en cada una de las actividades y tareas.

Además, en esta tesis doctoral se implementó un prototipo avanzado de una

infraestructura de soporte a la metodología Aware-IoT@Run.Time, compuesto por un conjunto de artefactos y herramientas de ingeniería de software que integran, automatizan y simplifican todas sus actividades y tareas. En primer lugar, el Lenguaje Específico de Dominio (DSL) Monitor-IoT y el configurador web de autoconsciencia permiten la construcción de modelos con especificaciones (conceptos) de alto nivel sobre las preocupaciones arquitectónicas, de monitoreo y autoconsciencia de un sistema IoT. Estas herramientas se basan en metamodelos construidos a partir de estándares (ISO/IEC 30141:2018), frameworks conceptuales y ontologías relevantes sobre arquitecturas de IoT, autoconsciencia computacional y medición de la calidad de software, lo cual asegura la formalidad y coherencia semántica de los modelos. A su vez, estos metamodelos poseen un vocabulario amplio para especificar una diversidad de entidades IoT (físicas y digitales), flujos de datos de monitoreo, así como aspectos y procesos de autoconsciencia, a nivel de las capas Edge, Fog y Cloud de un sistema IoT.

Como complemento, un middleware mantiene la relación causal entre los modelos y el sistema IoT gestionado, y viceversa. Este middleware ante cualquier cambio en las especificaciones de los modelos, implementa y/o reconfigura de forma automática y distribuida los artefactos de software de monitoreo y autoconsciencia de un sistema IoT, a nivel de las capas Edge, Fog y Cloud. Por su parte, los artefactos generados por el middleware se encargan de proveer el soporte necesario para que un sistema IoT pueda aprender y razonar a partir de la actualización y calibración continua de los parámetros (métricas, metadatos, etc.) de los modelos, volviéndose capaz de comprender su estado y posible evolución para llegar a recomendaciones (acciones correctivas) de ser necesario.

La metodología Aware-IoT@Run.Time y sus herramientas de soporte fueron evaluadas empíricamente mediante un estudio de caso y dos cuasi experimentos bajo el subdominio de ambientes de vida asistida. El estudio de caso ha demostrado que siguiendo paso a paso la guía metodológica y con el apoyo de las herramientas es factible construir sistemas IoT autoconscientes con un alto grado de abstracción, expresividad y flexibilidad. Mientras que, los cuasi experimentos basados en el Modelo de Evaluación de Métodos (MEM) y ejecutados en un entorno académico han determinado la alta efectividad y eficiencia de los participantes al utilizar la solución Aware-IoT@Run.Time. Además, estos cuasi experimentos han demostrado la intención de los participantes de utilizar la solución a futuro, ya que la consideran fácil de usar y útil.

Finalmente, la implementación y evaluación de la solución Aware-IoT@Run.Time ha servido para corroborar que un enfoque de ingeniería de software basado en la autoconsciencia computacional y los modelos en tiempo de ejecución resulta ser idóneo para abstraer la complejidad y elevar el nivel de automatización del desarrollo de los sistemas IoT, mejorar la gestión de su evolución y cambio, aumentar la independencia del dominio de aplicación y la plataforma tecnológica; y, en definitiva, reducir el esfuerzo y tiempo requerido para su puesta en operación. Sin embargo, a lo largo de la tesis doctoral se han evidenciado una serie de oportunidades de investigación para perfeccionar la solución Aware-IoT@Run.Time, mismas que deberán ser consideradas a futuro.

Palabras claves: Autoconsciencia Computacional, Ingeniería Dirigida por Modelos (MDE), Internet de las Cosas (IoT), Lenguaje Específico de Dominio (DSL), Modelos en Tiempo de Ejecución (models@run.time), Monitoreo.

Abstract

The Internet of Things (IoT) is one of the fastest-growing areas in information technologies due to its technical, social, and economic impact on society, which has favorably transformed how we live and work. Currently, one of the most important challenges for the scientific community and industry is the development of software for IoT systems, since they are highly distributed, heterogeneous, and dynamic, thus complicating the identification of all the system requirements in the development stage. Furthermore, traditional software engineering methodologies and tools have proven ineffective and unproductive in addressing the intrinsic characteristics of these systems and, in particular, the dynamism and uncertainty associated with them. These approaches and tools are limited to managing the evolution of systems at design time, leaving aside execution time. Likewise, they focus on implementation details (low-level programming) instead of the problem domain, thus making software development for IoT systems a complex and tedious task.

An appropriate approach is to incorporate computational self-awareness into IoT systems with the support of Model-Driven Engineering (MDE) and, particularly, models at runtime (`models@run.time`) to increase their autonomy and resilience in the face of unforeseen situations. Under this approach, an IoT system is considered self-aware if it can obtain and maintain knowledge about itself and the environment (e.g., structure, state, evolution) without external control. Meanwhile, models at runtime constitute a causally connected self-representation of an IoT system, serving as the means through which the system captures knowledge; meaning, it is aware.

However, computational self-awareness in IoT is a developing research area. Research efforts have mainly focused on theoretical work aimed at establishing roadmaps for adopting self-awareness in IoT systems, as well as practical solutions on architectures, frameworks, middlewares, and machine learning algorithms to support self-awareness in particular applications (ad hoc) of IoT. Hence, these efforts have neglected the conception of generic and comprehensive methodologies and infrastructures for constructing this type of system. Also, few studies explicitly consider MDE and runtime models as part of their self-aware solutions.

Faced with this problem, this doctoral thesis addresses the limitations and challenges presented by creating `Aware-IoT@Run.Time`, a software engineering method based on runtime models to support stakeholders in constructing, operating, and maintaining self-aware IoT systems. Specifically, the `Aware-IoT@Run.Time` methodological process and its activities are based on the model-based learning and reasoning loop (LRA-M). This methodological process is described in depth and graphically represented under the SPEM notation, which includes its hierarchical decomposition into activities, tasks, and subtasks, along with the specification of user roles, tools, and input/output artifacts that intervene in each activity and task.

Moreover, an advanced prototype of a support infrastructure for the `Aware-IoT@Run.Time` methodology was implemented in this doctoral thesis. It is composed of software engineering artifacts and tools that integrate, automate, and simplify all activities and tasks. First, the Monitor-IoT Domain-Specific Language (DSL) and the self-awareness web configurator enable the construction of models with high-level specifications (concepts) about the architectural, monitoring, and self-awareness concerns of an IoT system. These tools are based on metamodels built from relevant standards

(ISO/IEC 30141:2018), conceptual frameworks, and ontologies on IoT architectures, computational self-awareness, and software quality measurement, which ensures the formality and semantic coherence of the models. In turn, these metamodels have a broad vocabulary to specify a diversity of IoT entities (physical and digital), monitoring dataflows, as well as aspects and processes of self-awareness at the Edge, Fog, and Cloud layers of an IoT system.

As a complement, middleware maintains the causal relationship between the models and the managed IoT system and vice-versa. In the event of any change in the model specifications, this middleware automatically and distributedly implements and/or reconfigures the monitoring and self-awareness software artifacts of an IoT system at the Edge, Fog, and Cloud layers. On their part, the artifacts generated by the middleware are responsible for providing the necessary support so an IoT system can learn and reason from the continuous updating and calibration of the parameters (e.g., metrics, metadata) of the models, thus becoming capable of understanding its status and possible evolution to arrive at recommendations (corrective actions) if necessary.

The Aware-IoT@Run.Time methodology and its supporting tools were empirically evaluated through a case study and two quasi-experiments under the Ambient Assisted Living subdomain. The case study showed that, by following the methodological guide step by step and with the support of the tools, it is feasible to build self-aware IoT systems with a high degree of abstraction, expressiveness, and flexibility. Quasi-experiments based on the Method Evaluation Model (MEM) and executed in an academic environment determined the high effectiveness and efficiency of the participants when using the Aware-IoT@Run.Time solution. Furthermore, these quasi-experiments proved the participants' intention to use the solution in the future since they consider it easy to use and useful.

Finally, the implementation and evaluation of the Aware-IoT@Run.Time solution corroborates that a software engineering approach based on computational self-awareness and runtime models is suitable to abstract complexity and raise the automation level of IoT systems development, improve the management of their evolution and change, increase the independence of the application domain and the technological platform, and, ultimately, reduce the effort and time required to put it into operation. However, throughout this doctoral thesis, research opportunities to perfect the Aware-IoT@Run.Time solution were found, which should be considered in the future.

Keywords: Computational Self-Awareness, Model-Driven Engineering (MDE), Internet of Things (IoT), Domain-Specific Language (DSL), Runtime Models (models@run.time), Monitoring.

Fecha de defensa: 16 de diciembre de 2024

Jurado:

Dra. Marcela Genero, Universidad de Castilla-La Mancha, España

Dr. Luis Olsina, Universidad Nacional de La Pampa, Argentina

Dr. Mario Piattini, Universidad de Castilla-La Mancha, España

Tabla de Contenido

Capítulo 1: Introducción.....	1
1.1 Planteamiento del problema.....	1
1.2 Hipótesis	4
1.3 Objetivos.....	5
1.3.1 Objetivo general.....	5
1.3.2 Objetivos específicos	5
1.4 Método de investigación.....	5
1.5 Contexto de la investigación.....	8
1.6 Estructura de la tesis	9
Capítulo 2: Marco Teórico.....	12
2.1 Internet de las Cosas	12
2.1.1 Definición de IoT.....	13
2.1.2 Características de IoT	14
2.1.3 Arquitectura de IoT.....	15
2.2 Autoconsciencia computacional	17
2.2.1 Definición de sistemas informáticos autoconscientes	17
2.2.2 Autoconsciencia pública y privada.....	19
2.2.3 Niveles de autoconsciencia computacional	19
2.2.4 Autoconsciencia en sistema colectivos.....	20
2.2.5 Bucle de aprendizaje y razonamiento basado en modelos para sistemas autoconscientes (LRA-M loop)	21
2.2.6 Framework conceptual de autoconsciente computacional	22
2.3 Ingeniería Dirigida por Modelos (MDE).....	24
2.3.1 Paradigmas de MDE	26
2.3.2 Principios básicos de MDE.....	26
2.3.3 Modelos en tiempo de ejecución (models@run.time)	28
2.4 Medición del software	29
2.5 Conclusiones.....	30
Capítulo 3: Estado del Arte	32
3.1 Arquitecturas de IoT	32
3.1.1 Modelos de arquitectura de referencia para IoT	33
3.1.2 Metamodelos y DSLs para diseñar arquitecturas de IoT	33
3.1.3 Discusión	36
3.2 Autoconsciencia computacional en IoT y otros dominios relacionados.....	41
3.2.1 Estudios teóricos.....	42
3.2.2 Soluciones de autoconsciencia basadas en enfoques genéricos.....	44
3.2.3 Soluciones de autoconsciencia basadas en enfoques de MDE	46
3.2.4 Soluciones de autoconsciencia (ad hoc) para aplicaciones particulares de IoT.....	48
3.2.5 Discusión	50
3.3 Otras soluciones basadas en modelos en tiempo de ejecución para IoT	53
3.4 Conclusiones.....	53
Capítulo 4: Metodología Propuesta Aware-IoT@Run.Time	55
4.1 Definición de la metodología Aware-IoT@Run.Time	55
4.2 Descripción de las actividades de la metodología Aware-IoT@Run.Time.....	57

Actividad 1. Diseño de la arquitectura de monitoreo del sistema IoT	58
Actividad 2. Configuración de las capacidades de autoconsciencia del sistema IoT	62
Actividad 3. Sincronización entre los modelos de arquitectura - autoconsciencia y el sistema IoT observado (mantiene la relación causal).....	68
Actividad 4. Monitoreo	71
Actividad 5. Aprendizaje.....	71
Actividad 6. Razonamiento	72
4.3 Conclusiones.....	73
Capítulo 5: Infraestructura de Soporte a la Metodología.....	75
5.1 Arquitectura general de la infraestructura de soporte	75
5.2 DSL Monitor-IoT.....	77
5.2.1 Metamodelo Monitor-IoT	77
5.2.2 Diseñador gráfico de Monitor-IoT.....	78
5.3 Configurador web de capacidades de autoconsciencia de sistemas IoT.....	81
5.3.1 Metamodelo Aware-IoT	83
5.3.2 Arquitectura del configurador de autoconsciencia	84
5.4 Middleware de la infraestructura de soporte.....	87
5.4.1 Motor de sincronización de recursos de monitoreo	88
5.4.2 Motor de sincronización de recursos de autoconsciencia.....	90
5.4.3 Arquitectura del middleware de la infraestructura	93
5.5 Visor web de autoconsciencia.....	104
5.6 Estado actual y perspectivas futuras de ampliación de la infraestructura de soporte.....	104
5.7 Conclusiones.....	106
Capítulo 6: Aplicación de la Solución	108
6.1 Estudio de caso para la aplicación de la metodología Aware-IoT@Run.Time	108
6.1.1 Especificación de requerimientos de monitoreo del estudio de caso	108
6.1.2 Especificación de requerimientos de autoconsciencia del estudio de caso	110
6.1.3 Especificación de requerimientos de gestión de cambios del estudio de caso	111
6.2 Prerrequisitos de hardware y software base para el estudio de caso	113
6.3 Aplicación de la metodología Aware-IoT@Run.Time.....	115
6.3.1 Implementación de los requerimientos iniciales del estudio de caso	115
6.3.2 Implementación de los cambios de requeridos por el estudio de caso	156
6.4 Conclusiones.....	157
Capítulo 7: Evaluación Empírica de la Solución	159
7.1 Ámbito de la evaluación empírica de la solución.....	159
7.2 Modelo de Evaluación de Métodos (Method Evaluation Model - MEM).....	160
7.3 Cuasi experimento Nro. 1: Evaluación del DSL Monitor-IoT	162
7.3.1 Diseño del cuasi experimento	162
7.3.2 Adaptación del MEM para la evaluación del DSL Monitor-IoT.....	165
7.3.3 Ejecución del cuasi experimento	166
7.3.4 Análisis de los resultados del cuasi experimento	167
7.4 Cuasi experimento Nro. 2: Evaluación del configurador web de autoconsciencia y middleware de la infraestructura.....	173
7.4.1 Diseño del cuasi experimento	174
7.4.2 Adaptación del MEM para la evaluación del configurador web de autoconsciencia y middleware de la infraestructura de soporte	178

7.4.3	Ejecución del cuasi experimento	179
7.4.4	Análisis de los resultados del cuasi experimento	180
7.5	Amenazas a la validez de los cuasi experimentos	186
7.5.1	Validez interna.....	186
7.5.2	Validez externa	186
7.5.3	Validez del constructo	187
7.5.4	Validez de la conclusión.....	187
7.6	Conclusiones.....	188
Capítulo 8:	Conclusiones y Trabajo Futuro.....	190
8.1	Conclusiones.....	190
8.2	Resumen de las principales contribuciones	196
8.3	Difusión de resultados (publicaciones).....	199
8.4	Trabajo futuro	200
Bibliografía.....		203
Anexo A: Metamodelo Monitor-IoT		214
Anexo B: Metamodelo Aware-IoT		223
Anexo C: Modelo de Arquitectura de Monitoreo del Estudio de Caso		231
Anexo D: Boletín y Hoja de Trabajo del Cuasi Experimento Nro. 1.....		232
Anexo E: Boletín y Hoja de Trabajo del Cuasi Experimento Nro. 2.....		240
Anexo F: Resultados de los Cuasi Experimentos.....		249

Índice de Figuras

Figura 1-1. Modelo para la Transferencia de Tecnología adaptado para la tesis doctoral.....	6
Figura 1-2. Estructura general de la tesis doctoral.....	9
Figura 2-1. Modelo de arquitectura multicapa Edge-Fog-Cloud para IoT.....	16
Figura 2-2. Relación entre los tipos y niveles de autoconsciencia computacional.....	21
Figura 2-3. Bucle de aprendizaje y razonamiento basado en modelos para sistemas autoconscientes (LRA-M loop)	22
Figura 2-4. Elementos fundamentales de MDE.....	27
Figura 2-5. Relación causal entre un modelo en tiempo de ejecución y el sistema asociado	29
Figura 4-1. Proceso metodológico de alto nivel de Aware-IoT@Run.Time.....	56
Figura 4-2. Diseño de la arquitectura de monitoreo del sistema IoT: Recursos, herramientas y artefactos de entrada - salida.....	59
Figura 4-3. Diagrama de proceso para diseñar arquitecturas de monitoreo multicapa para sistemas IoT	60
Figura 4-4. Configuración de las capacidades de autoconsciencia del sistema IoT: Recursos, herramientas y artefactos de entrada - salida	63
Figura 4-5. Diagrama de proceso para configurar las capacidades de autoconsciencia de sistemas IoT	64
Figura 4-6. Sincronización entre los modelos de arquitectura - autoconsciencia y el sistema IoT observado: Herramientas y artefactos de entrada - salida	71
Figura 4-7. Monitoreo: Herramientas y artefactos de entrada - salida	71
Figura 4-8. Aprendizaje: Herramientas y artefactos de entrada - salida	72
Figura 4-9. Razonamiento: Herramientas y artefactos de entrada - salida	73
Figura 5-1. Arquitectura general de la infraestructura de soporte basada en models@run.time	76
Figura 5-2. Extracto del metamodelo Monitor-IoT con las principales metaclases y relaciones	79
Figura 5-3. Entorno de trabajo del DSL Monitor-IoT	80
Figura 5-4. Notación gráfica del DSL Monitor-IoT	80
Figura 5-5. Página principal del configurador web de capacidades de autoconsciencia.	82
Figura 5-6. Extracto del metamodelo Aware-IoT con sus metaclases y relaciones	85
Figura 5-7. Arquitectura del configurador web de capacidades de autoconsciencia de sistemas IoT.....	86
Figura 5-8. Diagrama de actividades del motor de sincronización de recursos de monitoreo de sistemas IoT	91
Figura 5-9. Diagrama de actividades del motor de sincronización de recursos de autoconsciencia de sistemas IoT	92
Figura 5-10. Arquitectura del middleware de la infraestructura de soporte	94
Figura 5-11. Visor web de autoconsciencia para sistemas IoT	105
Figura 6-1. Escenario de IoT del estudio de caso	110
Figura 6-2. Estructura jerárquica del sistema de gestión de emergencias	116
Figura 6-3. Modelado de la estructura jerárquica del Sistema de Gestión de Emergencia y sus subsistemas.....	122
Figura 6-4. Ventana de propiedades del DSL Monitor-IoT con los atributos del subsistema SCA	122
Figura 6-5. Modelado de las entidades físicas del escenario de IoT	122
Figura 6-6. Ventana de propiedades del DSL Monitor-IoT con los atributos de la entidad física dormitorio.....	123
Figura 6-7. Modelado de los nodos de computación del escenario de IoT.	123
Figura 6-8. Ventana de propiedades del DSL Monitor-IoT con los atributos del sensor DHT11	124
Figura 6-9. Modelado de recursos de hardware y software de los nodos de computación del escenario de IoT	125

Figura 6-10. Ventana de propiedades del DSL Monitor-IoT con los atributos de la base de datos PostgreSQLLocal	126
Figura 6-11. Ventana de propiedades del DSL Monitor-IoT con los atributos de la API obtenerTemperaturaHumedad	126
Figura 6-12. Ventana de propiedades del DSL Monitor-IoT con los atributos del servicio web guardarTemperaturaHumedad	126
Figura 6-13. Modelado de las redes y protocolos de comunicación del escenario de IoT	127
Figura 6-14. Ventana de propiedades con la configuración de la interfaz de red del nodo Fog HP Proliant Microserver	128
Figura 6-15. Ventana de propiedades con la configuración del broker Mosquitto	128
Figura 6-16. Modelado de categorías de entidades del escenario de IoT	129
Figura 6-17. Ventana de propiedades con los atributos de la categoría habitación	129
Figura 6-18. Ventana de propiedades con la configuración de la plantilla temperatura	130
Figura 6-19. Ventana de propiedades con la configuración de la propiedad temperatura a monitorear en el dormitorio	130
Figura 6-20. Modelado de flujos de datos y reglas de mapeo del escenario de IoT	131
Figura 6-21. Ventana de propiedades con la parametrización del flujo de recolección de datos de temperatura y humedad	131
Figura 6-22. Modelado de los enlaces de comunicación entre las entidades digitales del escenario de IoT	132
Figura 6-23. Registro del usuario de acceso	133
Figura 6-24. Creación del modelo de autoconsciencia del subsistema SCA	134
Figura 6-25. Definición de sujeto de autoconsciencia del escenario de IoT	134
Figura 6-26. Definición de objetos de autoconsciencia de tipo entidad física del escenario de IoT	135
Figura 6-27. Agregar aspecto de autoconsciencia individual: Concentración en tiempo real	136
Figura 6-28. Agregar métrica: Nivel de riesgo de CO	137
Figura 6-29. Configuración del criterio de decisión: nivel de riesgo de la concentración de CO	138
Figura 6-30. Configuración del servicio web obtenerModeloRegresionLinealSimple	140
Figura 6-31. Configuración del proceso pre-reflexivo que captura en tiempo real el conocimiento sobre la concentración y nivel de riesgo de CO	142
Figura 6-32. Configuración del proceso reflexivo que predice la carga de procesamiento de la CPU del nodo Fog	142
Figura 6-33. Configuración del método de recolección del proceso pre-reflexivo sobre la concentración actual de CO en el ambiente de la cocina	143
Figura 6-34. Configuración del método de cálculo del proceso reflexivo que predice la carga de procesamiento de la CPU del nodo Fog	145
Figura 6-35. Mapeo de parámetros del recurso de implementación (servicio web) obtenerRegresionLinealSimple	145
Figura 6-36. Configuración del escenario y variable de simulación para el servicio web obtenerRegresionLinealSimple	145
Figura 6-37. Configuración del modelo de análisis del proceso pre-reflexivo sobre la concentración y nivel de riesgo promedio diario de CO en el ambiente de la cocina	148
Figura 6-38. Configuración de las acciones o recomendaciones prescriptivas para el modelo de análisis del proceso pre-reflexivo sobre la concentración y nivel de riesgo promedio diario de CO en el ambiente de la cocina	148
Figura 6-39. Generación del archivo JSON con las especificaciones de arquitectura y autoconsciencia del subsistema SCA	149
Figura 6-40. Extracto de reporte de logs de las tareas ejecutadas por la aplicación supervisora en el nodo Fog	150
Figura 6-41. Extracto de reporte de logs de las tareas ejecutadas por la aplicación implementadora en la puerta de enlace IoT (Raspberry PI 3)	151
Figura 6-42. Extracto de reporte de logs de las tareas ejecutadas por la aplicación implementadora en el nodo Fog	151

Figura 6-43. Extracto de reporte de logs de las tareas ejecutadas por la aplicación implementadora en el nodo Cloud	151
Figura 6-44. Una muestra de los datos crudos recolectados y almacenados en la base de datos PostgreSQLLocal del nodo Fog.....	153
Figura 6-45. Logs de consola con información de las métricas directas producidas por los procesos de pre-reflexivos durante un periodo de tiempo.....	154
Figura 6-46. Logs de consola con información de las métricas indirectas producidas por los procesos de reflexivos durante un periodo de tiempo	154
Figura 6-47. Visualización de datos generados por el proceso reflexivo PID3 para el indicador nivel de carga de procesamiento de la CPU del nodo Fog en tiempo real.....	155
Figura 6-48. Visualización de datos generados por el proceso reflexivo PID6 para el indicador nivel de carga estimada de procesamiento de la CPU del nodo Fog	155
Figura 6-49. Extracto de reporte de logs de las tareas de reconfiguración ejecutadas por la aplicación implementadora en la puerta de enlace IoT (Raspberry PI 3)	156
Figura 6-50. Extracto de reporte de logs de las tareas de reconfiguración ejecutadas por la aplicación implementadora en el nodo Fog	157
Figura 7-1. Modelo de Evaluación de Métodos (Method Evaluation Model – MEM).	161
Figura 7-2. Modelo de la arquitectura de monitoreo del subsistema SS resultante de la ejecución de las tareas experimentales.....	164
Figura 7-3. Modelo adaptado del MEM para evaluar el DSL Monitor-IoT y su proceso metodológico.....	167
Figura 7-4. Diagramas de caja de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 1	168
Figura 7-5. Síntesis de los resultados del cuasi experimento Nro. 1	172
Figura 7-6. Plataforma de Docker desplegada durante el cuasi experimento Nro. 2.....	176
Figura 7-7. Modelo adaptado del MEM para evaluar el configurador de capacidad de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time	179
Figura 7-8. Diagramas de caja de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 2.....	181
Figura 7-9. Síntesis de los resultados del cuasi experimento Nro. 2.....	184

Índice de Tablas

Tabla 2-1. Niveles de autoconsciencia computacional	20
Tabla 3-1. Análisis comparativo entre los DSLs estudiados y Monitor-IoT (Parte 1)	39
Tabla 3-2. Análisis comparativo entre los DSLs estudiados y Monitor-IoT (Parte 2)	40
Tabla 5-1. Componentes de la capa de comunicación (síncrona o asíncrona)	96
Tabla 5-2. Componentes de la capa de lógica de negocio	98
Tabla 6-1. Descripción de los procesos pre-reflexivos y reflexivos a implementar en el estudio de caso	112
Tabla 6-2. Prerrequisitos de hardware y software base para la instanciación de la metodología Aware-IoT@Run.Time y su infraestructura de soporte.	114
Tabla 6-3. Plantilla para registrar sistemas y subsistemas del escenario de IoT	116
Tabla 6-4. Plantilla para registrar las especificaciones de las entidades físicas	117
Tabla 6-5. Plantilla para registrar las especificaciones de los dispositivos IoT (sensores).....	117
Tabla 6-6. Plantilla para registrar las especificaciones de las puertas de enlace IoT	117
Tabla 6-7. Plantilla para registrar las especificaciones de los nodos Fog	117
Tabla 6-8. Plantilla para registrar las especificaciones de los nodos Cloud.....	117
Tabla 6-9. Plantilla para registrar las especificaciones de las APIs	117
Tabla 6-10. Plantilla para registrar las especificaciones de las aplicaciones.....	118
Tabla 6-11. Plantilla para registrar las especificaciones de los middlewares.....	118
Tabla 6-12. Plantilla para registrar las especificaciones de los servicios web	118
Tabla 6-13. Plantilla para registrar las especificaciones de las bases de datos.....	119
Tabla 6-14. Plantilla para registrar las especificaciones de las tablas de base de datos	119
Tabla 6-15. Plantilla para registrar las especificaciones de los tópicos de broker.....	120
Tabla 6-16. Plantilla para registrar las especificaciones de las redes de comunicación y protocolos.....	120
Tabla 6-17. Plantilla para registrar las especificaciones de las interfaces de red	120
Tabla 6-18. Plantilla para registrar las especificaciones de las propiedades	121
Tabla 6-19. Plantilla para registrar las especificaciones de los flujos de datos	121
Tabla 6-20. Especificación de aspectos de autoconsciencia del objeto Dormitorio	135
Tabla 6-21. Especificación de aspectos de autoconsciencia del objeto Cocina.....	135
Tabla 6-22. Especificación de aspectos de autoconsciencia del objeto HP Proliant Microserver	136
Tabla 6-23. Especificación de métricas para evaluar los aspectos de autoconsciencia del objeto Dormitorio.....	136
Tabla 6-24. Especificación de métricas para evaluar los aspectos de autoconsciencia del objeto Cocina	137
Tabla 6-25. Especificación de métricas para evaluar los aspectos de autoconsciencia del objeto nodo Fog HP Proliant Microserver	137
Tabla 6-26. Especificación de criterios de decisión y sus umbrales para evaluar cualitativamente los aspectos de autoconsciencia	138
Tabla 6-27. Especificación de recursos de implementación genéricos	139
Tabla 6-28. Información general de los procesos de autoconsciencia pre-reflexivos	141
Tabla 6-29. Información general de los procesos de autoconsciencia reflexivos.....	141
Tabla 6-30. Parametrización de los métodos de recolección de los procesos pre-reflexivos	143
Tabla 6-31. Parametrización de los métodos de cálculo de los procesos reflexivos	144
Tabla 6-32. Especificación del escenario y variable de simulación para el servicio web obtenerRegresionLinealSimple.....	144
Tabla 6-33. Parametrización de los modelos de análisis de los procesos pre-reflexivos.....	146
Tabla 6-34. Parametrización de los modelos de análisis de los procesos reflexivos.....	147
Tabla 7-1. Goal-Question-Metric para el cuasi experimento Nro. 1	162
Tabla 7-2. Cuestionario para medir las variables de percepción del MEM para el cuasi experimento Nro. 1	166

Tabla 7-3. Niveles de significancia de Moody	168
Tabla 7-4. Estadística descriptiva y pruebas de hipótesis de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 1	169
Tabla 7-5. Estadística descriptiva de las variables de rendimiento (Eficiencia y Efectividad) para el cuasi experimento Nro. 1	169
Tabla 7-6. Modelo de regresión lineal simple entre Eficiencia y Facilidad de Uso Percibida para el cuasi experimento Nro. 1	170
Tabla 7-7. Modelo de regresión lineal simple entre Efectividad y Utilidad Percibida para el cuasi experimento Nro. 1	170
Tabla 7-8. Modelo de regresión lineal simple entre Facilidad de Uso Percibida y Utilidad Percibida para el cuasi experimento Nro. 1	171
Tabla 7-9. Modelo de regresión lineal simple entre Facilidad de Uso Percibida e Intención de Uso para el cuasi experimento Nro. 1	171
Tabla 7-10. Modelo de regresión lineal simple entre Utilidad Percibida e Intención de Uso para el cuasi experimento Nro. 1	171
Tabla 7-11. Resumen de hipótesis aceptadas y rechazadas en el cuasi experimento Nro. 1	172
Tabla 7-12. Goal-Question-Metric para el cuasi experimento Nro. 2	174
Tabla 7-13. Cuestionario para medir las variables de percepción del MEM para el cuasi experimento Nro. 2	178
Tabla 7-14. Estadística descriptiva y pruebas de hipótesis de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 2	181
Tabla 7-15. Estadística descriptiva de las variables de rendimiento (Eficiencia y Efectividad) para el cuasi experimento Nro. 2	182
Tabla 7-16. Modelo de regresión lineal simple entre Eficiencia y Facilidad de Uso Percibida para el cuasi experimento Nro. 2	182
Tabla 7-17. Modelo de regresión lineal simple entre Efectividad y Utilidad Percibida para el cuasi experimento Nro. 2	183
Tabla 7-18. Modelo de regresión lineal simple entre Facilidad de Uso Percibida y Utilidad Percibida para el cuasi experimento Nro. 2	183
Tabla 7-19. Modelo de regresión lineal simple entre Facilidad de Uso Percibida e Intención de Uso para el cuasi experimento Nro. 2	183
Tabla 7-20. Modelo de regresión lineal simple entre Utilidad Percibida e Intención de Uso para el cuasi experimento Nro. 2	184
Tabla 7-21. Resumen de hipótesis aceptadas y rechazadas en el cuasi experimento Nro. 2	184

Capítulo 1

Introducción

Este capítulo contextualiza el trabajo de investigación desarrollado durante la tesis doctoral. Particularmente, la sección 1.1 analiza la problemática y los aspectos que motivaron la realización de este trabajo. Luego, las secciones 1.2 y 1.3 plantean la hipótesis a contrastar y los objetivos de investigación, respectivamente. Por su parte, la sección 1.4 explica la metodología de investigación empleada; mientras que, la sección 1.5 describe el contexto en el que se ha desarrollado la tesis doctoral. Finalmente, la sección 1.6 presenta la organización de este documento.

1.1 Planteamiento del problema

El Internet de las Cosas (IoT) es un paradigma tecnológico utilizado para referirse a una infraestructura compuesta por un conjunto de objetos (cosas) inteligentes, ubicuos e interconectados mediante las tecnologías de información y comunicación para procesar datos del mundo físico y virtual; y, actuar con la mínima intervención humana [1][2][3]. El IoT promueve el desarrollo de innovadoras aplicaciones en una diversidad de dominios (p. ej., hogar, salud, ciudad inteligente, industria), agregando una nueva dimensión a la vida de las personas, centrada en la interacción permanente con los objetos inteligentes, en cualquier lugar y momento, lo cual impacta favorablemente en el bienestar y desarrollo sostenible de la sociedad [4][5][6][7]. Según Gartner [8], el número total de dispositivos IoT instalados en el año 2017 (8.400 millones), superó a la población total de seres humanos; mientras que, una previsión de la International Data Corporation (IDC) [9] estima que habrán alrededor de 41.600 millones de dispositivos IoT instalados en el 2025.

En la actualidad, uno de los desafíos más importantes para la comunidad científica y la industria es el desarrollo de software para sistemas IoT, dado que se caracterizan por ser altamente distribuidos, heterogéneos, escalables, dinámicos e inciertos [2][10]. En particular, los sistemas IoT operan en entornos ubicuos que cambian constantemente, debiendo ser controlados oportunamente para garantizar su correcto y continuo funcionamiento [10]. Debido a la incertidumbre intrínseca asociada a los cambios, resulta complejo identificar todos los requisitos de un sistema IoT durante la etapa de desarrollo [11]. Además, las metodologías y herramientas de ingeniería de software tradicionales han demostrado ser poco efectivas para afrontar el dinamismo y la escalabilidad de los sistemas IoT, siendo los desarrolladores los que gestionan la evolución de estos sistemas en el tiempo de diseño [12][13]. Esta limitación se agudiza aún más, ya que estos enfoques metodológicos y herramientas se centran en los detalles de implementación (programación a bajo nivel) en lugar del dominio del problema, volviéndose una tarea compleja y tediosa para los desarrolladores por la heterogeneidad de los nodos de computación y dispositivos IoT (sensores, actuadores) que deben ser integrados, a nivel de las capas Edge, Fog y Cloud de una infraestructura IoT [14]. Según Mineraud et al. [15], una de las brechas de investigación en IoT es la ausencia de herramientas (p. ej. Lenguajes Específicos de Dominio – DSLs) que ofrezcan primitivas funcionales para describir el espacio del problema y la solución con un alto nivel de abstracción, de tal forma que se simplifique y agilice el desarrollo y mantenimiento de las aplicaciones IoT.

Un enfoque apropiado para abordar los problemas señalados es incorporar la autoconsciencia computacional a los sistemas IoT con el apoyo de la Ingeniería Dirigida

por Modelos (MDE) y, particularmente los modelos en tiempo de ejecución (models@run.time), a fin de aumentar su capacidad autónoma y resiliente frente a las situaciones imprevistas [11][16][17][18]. Un sistema IoT es considerado autoconsciente, si es capaz de generar conocimiento sobre sí mismo y el entorno (p. ej., estructura, estado, evolución), sin un control externo [19]. Este conocimiento puede ser capturado directamente de los datos crudos recolectados desde los sensores físicos o virtuales (observación inmediata – perceptual) o extraído de modelos conceptuales internos, contruidos a partir de datos históricos [20].

El término autoconsciencia emerge desde la psicología y los primeros intentos por abordar explícitamente la autoconsciencia, en un contexto informático, comenzaron alrededor del año 2004 [11]. A su vez, la computación autoconsciente abarca varios aspectos que han sido estudiados implícitamente por otras áreas de investigación de la informática (p. ej., inteligencia artificial, sistemas multiagentes, computación autónoma, sistemas autoadaptativos, computación reflexiva, modelos en tiempo de ejecución); sin embargo, el área de los sistemas informáticos autoconscientes todavía es poco explorada; y, plantea importantes desafíos y oportunidades para la comunidad científica [21]. Estos desafíos están enfocados en trasladar varias de las funcionalidades de los sistemas que tradicionalmente se programan en tiempo de diseño al tiempo de ejecución [22], a través de la búsqueda de soluciones que incorporen el aprendizaje y razonamiento (capacidades de autoconsciencia) en línea basado en modelos; y, que a su vez sean viables técnica y económicamente, con respecto a la cantidad de recursos de hardware y software que necesitan para apoyar los grados de autoconsciencia de un sistema [21].

Particularmente, Elhabbash et al. [23] en su revisión sistemática sobre la adopción de la autoconsciencia en la ingeniería de software, evidencian que los trabajos de investigación sobre autoconsciencia dentro del dominio de IoT son escasos. Un alto porcentaje de estos estudios son teóricos, los cuales se centran en identificar los beneficios, desafíos y recomendaciones (hojas de ruta) en torno a la construcción de sistemas IoT autoconscientes [24][25][26][27]. A su vez, la mayoría de estudios de carácter práctico se han limitado a proponer soluciones (arquitecturas, frameworks, middlewares y algoritmos de aprendizaje automático) para apoyar la autoconsciencia en aplicaciones específicas (ad hoc) de IoT, dejando de lado la concepción de propuestas metodológicas e infraestructuras genéricas e integrales para la construcción de sistemas IoT con características autoconscientes.

Entre los trabajos destacables sobre autoconsciencia computacional que han servido de inspiración para la realización de la presente tesis doctoral se pueden citar los siguientes. Por un lado, el bucle de aprendizaje y razonamiento basado en modelos (LRA-M loop) [19] que describe los principales procesos de un sistema informático autoconsciente. Por otro lado, el framework conceptual propuesto por Lewis et al. [20], que organiza las diferentes formas y capacidades de autoconsciencia que un sistema informático puede poseer. Este framework proporciona un lenguaje conformado por los siguientes conceptos técnicos: i) niveles de autoconsciencia (pre-reflexiva, reflexiva y meta-reflexiva); ii) aspectos de autoconsciencia reflexiva; y, iii) dominio (objeto y sujeto) de autoconsciencia. Mediante la combinación de estos tres conceptos se pueden producir descripciones de requerimientos de autoconsciencia para un sistema específico.

En el contexto del desarrollo de software, el término modelo se refiere a una representación abstracta de un sistema y su entorno [28]. Tradicionalmente, se han utilizado los modelos para fines de documentación y comunicación en las actividades de desarrollo de un sistema. La Ingeniería Dirigida por Modelos (Model-Driven Engineering

- MDE) es un paradigma que ha incrementado la importancia de los modelos, ya que no solo se utilizan para los fines señalados, sino como artefactos centrales del proceso de desarrollo de software [28][29]. Además, en los últimos años, varias iniciativas de investigación han llevado la idea de utilizar los modelos, un paso más adelante, en tiempo de ejecución (*models@run.time*) [30], demostrando su utilidad para hacer frente a los aspectos complejos, dinámicos e inciertos de los sistemas modernos (p. ej., IoT) y su entorno [31][32][33].

Los modelos en tiempo de ejecución siguen el principio de la reflexión computacional, que según lo descrito por Pattie Maes, es el comportamiento exhibido por un sistema reflexivo, donde un sistema reflexivo es un sistema computacional que está relacionado consigo mismo de una manera causalmente conectada [34]. En este sentido, los modelos en tiempo de ejecución representan una capa de reflexión, la cual está causalmente conectada con el sistema, de modo que, cualquier cambio en el modelo en tiempo de ejecución, conduce a cambios en el sistema reflejado y viceversa [31]. Por lo tanto, los modelos en tiempo de ejecución pueden definirse como una autorrepresentación causalmente conectada del sistema asociado que enfatiza la estructura, el comportamiento y/o los objetivos del sistema y que puede ser manipulada en tiempo de ejecución para fines específicos [31][35].

En el ámbito de la autoconsciencia computacional, los modelos en tiempo de ejecución son el núcleo de los sistemas autoconscientes y proporcionan el medio para la autorrepresentación del sujeto (sistema) que es consciente [21]. Es así que, la información recopilada desde el sistema observado puede ser utilizada para retroalimentar los modelos en tiempo de ejecución. Luego, por medio de estos modelos se puede analizar el estado y la posible evolución del sistema; y, en caso de ser necesario recomendar acciones correctivas [36].

Por su parte, Bencomo et al. [33] en su revisión sistemática de la literatura sobre *models@run.time*, concluyen que uno de los propósitos y desafíos más relevantes de los modelos en tiempo de ejecución es la autoconsciencia computacional, debido a los pocos trabajos encontrados en la literatura, existiendo una importante brecha de investigación a ser aprovechada. Puntualmente, los autores consideran prioritario enfocarse en la aplicación de los modelos en tiempo de ejecución para construir sistemas autoconscientes de su entorno y de la interacción con otros sistemas o entre sus subsistemas o componentes (autoconsciencia colectiva).

Frente a esta problemática, surge la necesidad de abordar las limitaciones mencionadas mediante la creación de un método de ingeniería de software (*Aware-IoT@Run.Time*) basado en modelos en tiempo de ejecución que apoye a los stakeholders (p. ej., desarrolladores, expertos de dominio) en la construcción, operación y mantenimiento de sistemas IoT autoconscientes, con un alto grado de abstracción, automatización y flexibilidad.

Para ello, este enfoque metodológico debe estar soportado por una infraestructura conformada por un conjunto de artefactos y herramientas (p. ej., metamodelos, lenguajes específicos de dominio - DSLs, middleware, entre otros) que permitan a los stakeholders, por medio de lenguajes basados en conceptos del dominio del problema, construir modelos con especificaciones de alto nivel sobre cómo implementar los requisitos arquitectónicos, de monitoreo y autoconsciencia de un sistema IoT. Específicamente, los metamodelos de la solución metodológica deben basarse en estándares, frameworks conceptuales, ontologías y otros trabajos relevantes sobre arquitecturas de IoT,

autoconsciencia computacional y medición de la calidad de software, de modo que se asegure su formalidad y coherencia semántica; y, se disponga de lenguajes más completos, flexibles y extensibles para modelar o especificar una diversidad de alternativas de implementación de procesos de monitoreo y autoconsciencia (pre-reflexivos³, reflexivos⁴) para sistemas IoT. Estos procesos deben dotar a un sistema IoT de capacidades autónomas que le permitan evaluar (capturar el conocimiento) el estado y la posible evolución de sí mismo y su entorno, así como sugerir acciones para alcanzar un estado deseado de ser necesario.

A su vez, las herramientas de soporte a la solución metodológica deben minimizar la participación de los stakeholders en las tareas de programación a bajo nivel de los requisitos del sistema IoT, al igual que asegurar la relación causal entre los modelos y el sistema IoT observado, y viceversa. En primer lugar, a partir de las especificaciones contenidas en los modelos y los cambios que estos pueden presentar en el tiempo, las herramientas deben implementar, mantener y reconfigurar automáticamente los artefactos de software de monitoreo y autoconsciencia, a nivel de las capas Edge, Fog y Cloud, sin detener la operación de la infraestructura IoT. En segundo lugar, los artefactos de monitoreo y autoconsciencia deben actualizar en tiempo de ejecución las métricas, metadatos y otros parámetros especificados en los modelos, de tal forma que reflejen en todo momento el estado o la probable evolución del sistema en operación; y, a partir de estos, el sistema pueda determinar las respectivas recomendaciones prescriptivas incluidas en los modelos.

Por último, el enfoque metodológico debe ser lo suficientemente genérico, de tal manera que pueda aplicarse en cualquier dominio o plataforma tecnológica de IoT. En este sentido, debe ser factible implementar y desplegar las especificaciones de dominio de alto nivel de los modelos en cualquier plataforma tecnológica, únicamente creando a nivel del middleware de la infraestructura de soporte a la metodología, nuevos algoritmos que transformen dichas especificaciones en artefactos de software de monitoreo y autoconsciencia compatibles con la plataforma destino.

En síntesis, este trabajo de investigación busca crear un enfoque metodológico junto con su infraestructura de soporte que abstraiga la complejidad y eleve el nivel de automatización del proceso de desarrollo de sistemas IoT, mejore la gestión de la evolución y cambio de estos sistemas; y, en definitiva, reduzca el esfuerzo y tiempo requerido para su puesta en operación.

1.2 Hipótesis

La Ingeniería Dirigida por Modelos y, particularmente los modelos en tiempo de ejecución, en combinación con la autoconsciencia computacional constituyen un enfoque eficaz y eficiente para apoyar la construcción, operación y mantenimiento de sistemas IoT.

Por lo tanto, la aplicación de una metodología y sus herramientas de soporte basadas en modelos en tiempo de ejecución para la construcción, operación y mantenimiento de sistemas IoT autoconscientes son percibidas por los stakeholders como fáciles de usar, útiles; y, por ende, tienen la intención de adoptarlas en proyectos futuros.

³ Proceso capaz de obtener conocimiento subjetivo directamente de los datos crudos recolectados desde los sensores (observación perceptual), permitiendo al sistema IoT responder a los eventos de manera inmediata [36].

⁴ Proceso capaz de construir modelos internos conceptuales o reflexivos que capturan conocimiento a partir de los datos históricos recopilados por los sensores o calculados previamente por otros procesos [36].

1.3 Objetivos

1.3.1 Objetivo general

Proponer una metodología e infraestructura de soporte basada en modelos en tiempo de ejecución para la construcción, operación y mantenimiento de sistemas IoT autoconscientes.

1.3.2 Objetivos específicos

1. Analizar el estado del conocimiento tecnológico sobre la autoconsciencia computacional y la Ingeniería Dirigida por Modelos en IoT y otros dominios relacionados.
2. Definir un proceso metodológico sistemático, disciplinado y repetible, basado en modelos en tiempo de ejecución, que integre las actividades, recursos, artefactos y herramientas necesarias para la construcción, operación y mantenimiento de sistemas IoT autoconscientes.
3. Construir la infraestructura de soporte a la metodología, bajo una plataforma tecnológica que se adapte a las características intrínsecas de IoT (distribuido, heterogéneo, dinámico y escalable).
4. Utilizar la metodología y las herramientas de la infraestructura de soporte en un estudio de caso en IoT, a fin de demostrar la aplicabilidad de la solución tecnológica para la construcción, operación y mantenimiento de sistemas IoT autoconscientes.
5. Evaluar empíricamente la metodología y su infraestructura de soporte en un subdominio de IoT, mediante el empleo de cuasi experimentos. Esta evaluación permitirá recolectar evidencia suficiente para comprender el rendimiento (eficiencia, efectividad) de los usuarios al utilizar la solución tecnológica en la construcción de sistemas IoT, así como su percepción sobre la facilidad de uso, utilidad e intención de uso futuro de la misma.

1.4 Método de investigación

La metodología utilizada durante el desarrollo de la tesis doctoral está basada en el Modelo para la Transferencia de Tecnología propuesto por Gorschek et al. [37], el cual fomenta una estrecha cooperación y simbiosis entre los investigadores y profesionales de la industria, haciendo posible que las investigaciones no solo se basen en temas interesantes, sino también que sean relevantes para la industria. Desde un ámbito general, este modelo se fundamenta en un proceso conformado por siete actividades (ver Fig. 1-1), el cual inicia con la formulación del problema y la revisión del estado del arte. Luego se definen las soluciones candidatas, las cuales a través de un ciclo evolutivo son objeto de evaluaciones empíricas y una mejora continua hacia la búsqueda de una solución adecuada y realista.

En esta tesis doctoral se han cubierto las cinco primeras actividades del modelo de investigación y transferencia de tecnología, llegando hasta la validación inicial de la solución propuesta en el contexto de la academia. A su vez, las dos últimas actividades del modelo, consistentes en la validación realista y la liberación de la solución en la industria serán abordadas en trabajos futuros.

A continuación, se detallan cada una de las actividades ejecutadas a lo largo de la tesis doctoral:

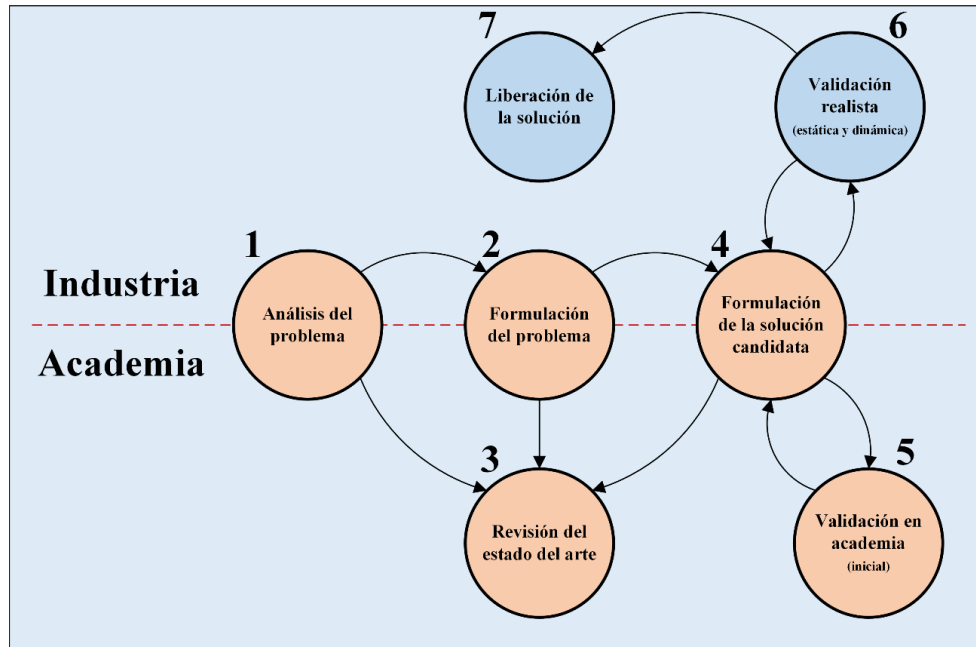


Figura 1-1. Modelo para la Transferencia de Tecnología adaptado para la tesis doctoral.
Fuente: Gorschek et al., 2006 [37].

1. **Análisis del problema.** Esta actividad se enfocó en comprender con claridad la problemática que atañe al desarrollo de software para sistemas IoT, profundizando particularmente en la identificación de las causas que originan la poca efectividad y eficiencia de la metodologías y herramientas de ingeniería de software tradicionales para la construcción de este tipo de sistemas, así como la determinación de las brechas, desafíos y oportunidades de investigación de esta área; y, las posibles formas (soluciones emergentes) de resolver este problema. Para ello, este análisis fue llevado a cabo mediante reuniones con la participación de investigadores, profesionales de software y expertos del dominio de IoT; y, complementado con una amplia y exhaustiva revisión de la literatura.
2. **Formulación del problema.** Una vez identificado y estudiado el problema a profundidad, este se formuló de manera precisa, entendible y con un sustento teórico adecuado. En la formulación del problema se delimitó el objeto de estudio e incorporó los antecedentes, características, factores contextuales y supuestos en los que se fundamenta.
3. **Revisión del estado del arte.** En paralelo con las actividades 1, 2 y 4 se realizó una revisión crítica de la literatura con el propósito de comprender el estado del conocimiento tecnológico sobre el uso de la Ingeniería Dirigida por Modelos y la autoconsciencia computacional en IoT y otros dominios relacionados. En particular, esta revisión permitió determinar las fortalezas y debilidades de las soluciones propuestas en la literatura, así como las brechas, desafíos y oportunidades de estas áreas de investigación. Para operativizar esta actividad, por un lado, se aprovecharon y analizaron a profundidad varios libros y revisiones relevantes y actuales disponibles en la literatura, relacionadas con los principales ejes temáticos de esta tesis doctoral, como son: Ingeniería Dirigida por Modelos (models@run.time) [30][33][38][39] y autoconsciencia computacional [13][23][40][41][42][43]. Por otro lado, se siguieron las directrices y procedimientos propuesto por Kitchenham y Charters [44] para recolectar, seleccionar y analizar los trabajos relacionados (estudios primarios) relevantes con el objeto de estudio de esta tesis doctoral.

4. **Formulación de la solución candidata.** Esta actividad se desarrolló de manera incremental y evolutiva. En los primeros incrementos se idearon soluciones potenciales a nivel del proceso metodológico, la arquitectura de la infraestructura y sus metamodelos, las cuales se construyeron a partir de estándares, arquitecturas de referencia, modelos conceptuales, ontologías y otros trabajos relacionados relevantes existentes en la literatura. En los siguientes incrementos se implementaron y completaron paulatinamente los restantes componentes y herramientas de la infraestructura de soporte, hasta llegar a tener una solución metodológica integral, soportada por un conjunto de herramientas operativas que abarquen todas las actividades de la metodología. Cabe recalcar que al final de cada incremento, se realizaron Revisiones Técnicas Formales (RTF), pruebas de software o evaluaciones empíricas dependiendo de los productos obtenidos, cuyos resultados fueron utilizados en los subsiguientes incrementos, a fin de alcanzar una mejora continua del proceso metodológico y la infraestructura. Siendo consistente con lo mencionado, para la construcción de las herramientas de software de la infraestructura se utilizaron metodologías de desarrollo de software ágiles, particularmente SCRUM [45], ya que resulta ser un enfoque metodológico flexible y adaptable para procesos de desarrollo empíricos u holístico, en donde se tiene un objetivo final; sin embargo, el camino para llegar a esa solución puede ser incierto o muy variante.
5. **Validación en academia (inicial).** En este punto se realizaron varias evaluaciones empíricas de la metodología y su infraestructura de soporte en un entorno de laboratorio universitario, con el propósito de probar el nivel de efectividad y eficiencia de los usuarios al utilizar la solución propuesta, así como su percepción sobre la facilidad de uso, utilidad e intención de uso futuro de la misma. Además, conforme a lo señalado con anterioridad, los resultados de las evaluaciones sirvieron para perfeccionar la solución candidata. Como estrategia de evaluación empírica se utilizaron cuasi experimentos, siguiendo el proceso experimental propuesto por Wohlin et al. [46] y seleccionando como sujetos a estudiantes de los últimos semestres de la carrera de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca. Un cuasi experimento es una investigación empírica, similar a un experimento, que mide el efecto de manipular una variable (denominada independiente o factor) en otra variable (denominada variable dependiente), habiendo aleatorizado las posibles variables perturbadoras [47]. Sin embargo, en un cuasi experimento los tratamientos no pueden asignarse de manera aleatoria a los sujetos; por el contrario, la asignación emerge de las características propias de los sujetos u objetos [48]. En términos generales, según Wohlin et al. [46] los pasos de un proceso experimental son: i) *Alcance*, se analiza el problema y definen los objetivos y metas del experimento; ii) *Planificación*, sienta las bases del experimento, para lo cual, se establece el contexto, formula las hipótesis, selecciona las variables independientes y dependientes, identifica los sujetos de estudio, escoge el tipo de diseño, prepara la instrumentación y evalúa las amenazas del experimento; iii) *Operación*, se ejecuta el experimento conforme a su planificación y diseño, recogiendo y validando las mediciones resultantes del mismo; iv) *Análisis e Interpretación*, se aplican medidas de estadística descriptiva, gráficos (p. ej., diagramas de cajas) y pruebas de hipótesis para comprender los datos y llegar a conclusiones en el experimento; y, v) *Presentación*, se elabora un informe detallado, incluyendo los resultados, hallazgos y conclusiones del experimento. Finalmente, en la formulación del cuasi experimento, se utilizó el Modelo de Evaluación de Métodos (MEM) propuesto por Moody [49], ya que contempla variables de rendimiento y percepción del usuario, así como la

relación entre estas, lo cual fue de gran valía para predecir la utilidad y posible adopción de la solución candidata en prácticas futuras.

6. **Validación realista (estática y dinámica).** Esta actividad tiene como alcance llevar a cabo estudios de caso y/o experimentos controlados en entornos industriales, con la participación de profesionales y expertos de dominio. En primer lugar, se realiza una presentación general de la solución candidata, con el objetivo de receptar comentarios de los profesionales (validación estática). Luego, se ejecutan proyectos pilotos con alcances limitados que cuentan con la participación conjunta de investigadores y profesionales hasta llegar a proyectos más grandes en los que se prescinde de la intervención directa de los investigadores (validación dinámica). Aunque, la validación realista está fuera del alcance de esta tesis doctoral, se debe resaltar que, la solución propuesta se aplicó en un estudio de caso sin la participación de actores de la industria, cuyo objetivo fue implementar, poner en operación y mantener un sistema con capacidades de monitoreo y autoconsciencia para gestionar las emergencias de los adultos mayores que viven solos en su hogar.
7. **Liberación de la solución.** En esta última actividad se evalúan los resultados obtenidos de las validaciones a la solución candidata y se toma la decisión de dar paso al lanzamiento oficial de la solución. Adicionalmente, se prepara documentación de soporte necesaria para el uso de la solución en la industria.

1.5 Contexto de la investigación

Esta tesis doctoral se ha desarrollado en el marco de varios proyectos de investigación ejecutados por el Laboratorio de Investigación y Desarrollo en Informática (LIDI) de la Universidad del Azuay y el Departamento de Ciencias de la Computación (DCC) de la Universidad de Cuenca, mismo que se detallan a continuación:

- Proyecto “Fog computing aplicado a monitoreo de dispositivos usados en ambientes de vida asistidos (ambient assisted living); caso de estudio: plataforma para el adulto mayor”, XVII Concurso Universitario de Proyectos de Investigación, financiado por la Universidad de Cuenca. Directora: Priscila Cedillo Orellana. Periodo de ejecución: 03 de septiembre de 2018 al 28 de febrero de 2021.
- Proyecto “Middleware de autoconsciencia basado en modelos en tiempo de ejecución para ecosistemas de Internet de las Cosas (Internet of Things - IoT)”, convocatoria 2020 de Proyectos de Investigación Científica, Desarrollo e Innovación Tecnológica (I+D+i) de la Universidad del Azuay (Grant: 2020-0059). Director: Lenin Erazo Garzón. Periodo de ejecución: 01 de abril de 2020 al 01 de abril de 2021.
- Proyecto “Metodología e infraestructura basada en modelos en tiempo de ejecución para la construcción y operación de sistemas autoconscientes de Internet de las Cosas”, ejecutado a través de varias etapas financiadas por las convocatorias 2021, 2022, 2023 y 2024 de Proyectos de Investigación Científica, Desarrollo e Innovación Tecnológica (I+D+i) de la Universidad del Azuay (Grants: 2021-0040, 2022-0030, 2023-0106 y 2024-0027). Periodo de ejecución: 01 de abril de 2021 al 01 de septiembre de 2024.

Como parte de estos proyectos han participado varios estudiantes de pregrado, los cuales mediante pasantías y trabajos de titulación han colaborado en la programación, implementación y validación de las herramientas de software que soportan la solución metodológica (Aware-IoT@Run.Time) propuesta en esta tesis doctoral.

1.6 Estructura de la tesis

En este capítulo se expuso la problemática que justifica el desarrollo de la tesis doctoral. A su vez, se definió la hipótesis a contrastar y los objetivos de investigación; y, finalmente, se describió la metodología de investigación empleada a lo largo de la tesis doctoral, así como el contexto en la que se desarrolló la misma.

El proceso de investigación y los resultados de la tesis doctoral se explican en varios capítulos, los cuales guardan relación con las actividades de la metodología utilizada. La Figura 1-2 presenta la estructura general de la tesis, en la cual se puede observar el flujo de las actividades de la metodología; y, su relación con los capítulos de la tesis.

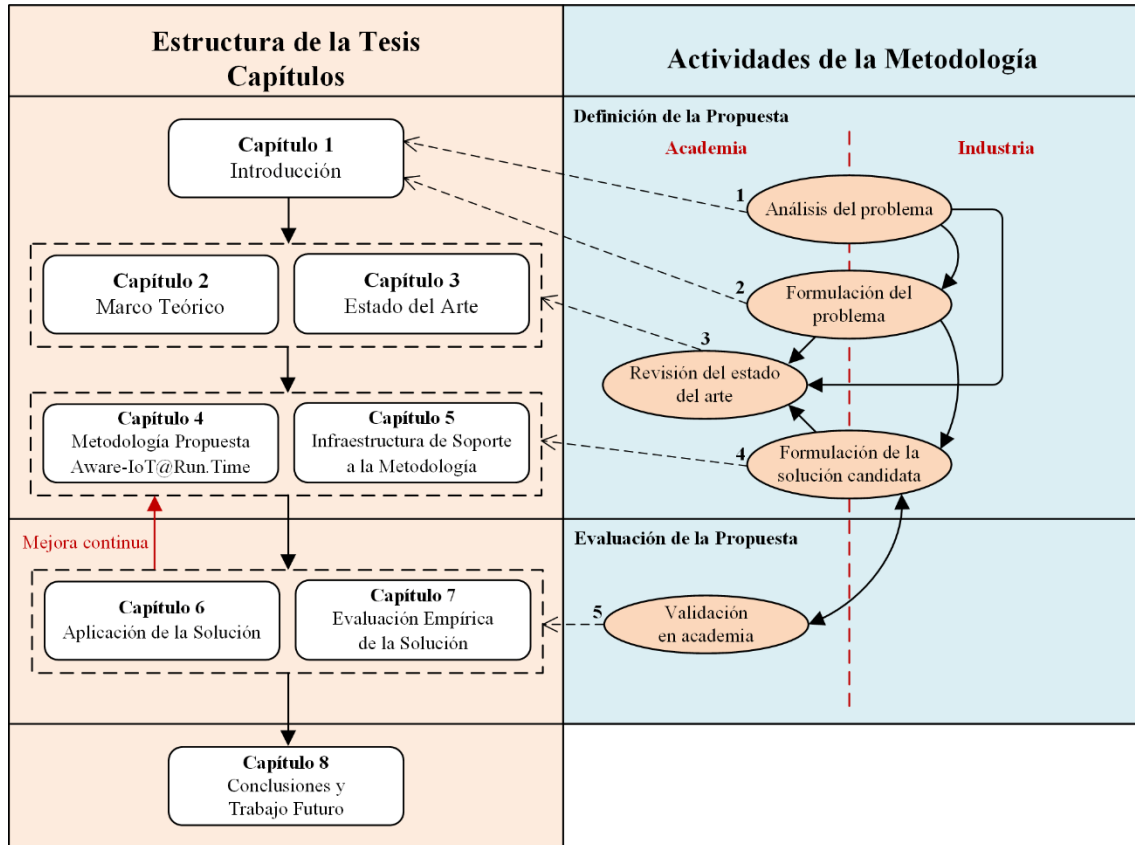


Figura 1-2. Estructura general de la tesis doctoral. Fuente: Elaboración propia.

A continuación, se presenta una breve descripción de cada uno de los capítulos subsiguientes:

- **Capítulo 2: Marco Teórico**

Aborda los aspectos conceptuales sobre las áreas temáticas en las que se desarrolla la tesis doctoral, como son: Internet de las Cosas (IoT), autoconsciencia computacional, medición de software e Ingeniería Dirigida por Modelos (MDE), haciendo hincapié en los modelos en tiempo de ejecución (models@run.time).

- **Capítulo 3: Estado del Arte**

Discute el estado del conocimiento tecnológico sobre la autoconsciencia computacional y la Ingeniería Dirigida por Modelos en IoT y otros dominios relacionados. Este capítulo identifica las fortalezas y limitaciones de los principales

estudios existentes en la literatura y resalta las brechas y desafíos abiertos que son abordados en esta tesis doctoral.

- **Capítulo 4: Metodología Propuesta Aware-IoT@Run.Time**

Comienza con una visión general sobre la contribución metodológica de este trabajo de investigación, denominada Aware-IoT@Run.Time, cuyo propósito es apoyar a los stakeholders en la construcción, operación y mantenimiento de sistemas IoT autoconscientes. Luego, se explica detalladamente cada una de las actividades principales que conforman la metodología Aware-IoT@Run.Time, descomponiéndolas en tareas y subtarefas de más bajo nivel. Además, se identifican los roles de usuario, guías, artefactos y herramientas que se deben utilizar en cada una de las actividades. Finalmente, el capítulo incluye diagramas de proceso a diferentes niveles de descomposición que muestran el orden de ejecución de las actividades y tareas de la metodología, así como sus artefactos de entrada y salida.

- **Capítulo 5: Infraestructura de Soporte a la Metodología**

Presenta la infraestructura basada en modelos en tiempo de ejecución que soporta las actividades de la metodología Aware-IoT@Run.Time. En primer lugar, se describe la arquitectura general de la infraestructura y sus componentes. Luego, se detallan a profundidad cada uno de los artefactos y herramientas de la infraestructura que permiten construir los modelos de arquitectura de monitoreo y capacidades de autoconsciencia de un sistema IoT, así como implementar y mantener de forma automática y distribuida los artefactos de software de monitoreo y autoconsciencia para asegurar la relación de causalidad entre los modelos y el sistema IoT observado. Por último, se explica el estado de implementación actual de la infraestructura y sus perspectivas futuras de ampliación.

- **Capítulo 6: Aplicación de la Solución**

Describe paso a paso la aplicación de la metodología Aware-IoT@Run.Time y sus herramientas de soporte en un estudio de caso que tiene como objetivo implementar, poner en operación y mantener un sistema con capacidades de monitoreo y autoconsciencia para gestionar las emergencias de los adultos mayores que viven solos en su hogar. Primero, se presenta el estudio de caso, detallando los requerimientos de monitoreo y autoconsciencia a implementar. Luego, se detallan los prerequisites de hardware y software base para la implementación del estudio de caso. Finalmente, se explica la ejecución de cada una de las actividades y tareas de la metodología y los resultados obtenidos en el estudio caso.

- **Capítulo 7: Evaluación Empírica de la Solución**

Explica la evaluación empírica de la metodología Aware-IoT@Run.Time y sus herramientas de soporte mediante la ejecución de dos cuasi experimentos, cuyo propósito es comprender la efectividad y eficiencia de esta solución tecnológica, así como la percepción del usuario sobre la facilidad de uso, utilidad e intención de uso futuro de la misma.

- **Capítulo 8: Conclusiones y Trabajo Futuro**

Expone las conclusiones en función de los objetivos y contribuciones alcanzados a lo largo de este trabajo de investigación. Además, se presentan las publicaciones que se

originaron a partir de esta tesis doctoral. Finalmente, se discuten las líneas y oportunidades de investigación futuras.

Adicionalmente, se incluyen en esta tesis doctoral, los anexos que amplían y respaldan el trabajo realizado:

- **Anexo A: Metamodelo Monitor-IoT**

Explica exhaustivamente las metaclases, atributos, relaciones y enumeraciones del metamodelo Monitor-IoT, el cual provee un lenguaje para modelar arquitecturas de IoT que soporten los procesos de monitoreo a través de las capas Edge, Fog y Cloud.

- **Anexo B: Metamodelo Aware-IoT**

Explica detalladamente las metaclases, atributos, relaciones y enumeraciones del metamodelo Aware-IoT, mismo que proporciona un lenguaje para especificar el dominio y las capacidades (niveles) de autoconsciencia de un sistema IoT, basado en métricas de calidad de software.

- **Anexo C: Modelo de Arquitectura de Monitoreo del Estudio de Caso**

Presenta el modelo de arquitectura de monitoreo completo del Subsistema de Control Ambiental (SCA), mismo que fue implementado en el estudio de caso presentado en el Capítulo 6.

- **Anexo D: Boletín y Hoja de Trabajo del Cuasi Experimento Nro. 1**

Describe el material experimental utilizado en el primer experimento, incluyendo la descripción de los objetivos de la evaluación, el enunciado del escenario de IoT a resolver en el experimento, los pre-requisitos para la ejecución del experimento; y, la hoja de trabajo con las instrucciones, campos y enlaces necesarios para la ejecución de las tareas experimentales y la recogida de los datos y productos resultantes del ejercicio experimental.

- **Anexo E: Boletín y Hoja de Trabajo del Cuasi Experimento Nro. 2**

Describe el material experimental empleado en el segundo experimento, conteniendo los mismos insumos citados en el anexo anterior.

- **Anexo F: Resultados de los Cuasi Experimentos**

Desglosa los resultados obtenidos en los cuasi experimentos para cada una de las variables de rendimiento (efectividad, eficiencia) y percepción (facilidad de uso, utilidad, intención de uso) del Modelo de Evaluación de Métodos (MEM).

Capítulo 2

Marco Teórico

Este capítulo presenta un marco conceptual sobre las áreas tecnológicas en las que se desarrolla la tesis doctoral, como son: Internet de las Cosas (IoT), autoconsciencia computacional, Ingeniería Dirigida por Modelos (MDE) y medición del software.

La sección 2.1 aborda el estudio del Internet de las Cosas, analizando las definiciones propuestas en la literatura, así como la importancia y las características intrínsecas de este paradigma tecnológico. Adicionalmente, se estudian los componentes, funcionalidades y beneficios de una arquitectura multicapa Edge-Fog-Cloud para sistemas IoT.

La sección 2.2 en primera instancia identifica los principales problemas y desafíos que motivaron el estudio de la autoconsciencia en un contexto informático. Luego, se explican los conceptos, tipos y niveles de autoconsciencia computacional que han sido inspirados, adaptados y propuestos a partir de la autoconsciencia humana. Además, en esta sección se estudian dos trabajos relevantes: el bucle de aprendizaje y razonamiento basado en modelos (LRA-M loop) [19] y el framework conceptual de autoconsciencia computacional propuesto por Lewis et al. [20], mismos que constituyen la base para el desarrollo de este trabajo de investigación.

La sección 2.3 contempla el estudio de los paradigmas y principios básicos de la Ingeniería Dirigida por Modelos (MDE), con énfasis en los modelos en tiempo de ejecución (models@run.time).

La sección 2.4 describe los principales términos y conceptos relacionados con la medición de software, los cuales se utilizan en este trabajo de investigación.

Finalmente, la sección 2.5 presenta las conclusiones del capítulo.

2.1 Internet de las Cosas

El término Internet de las Cosas o IoT, por sus siglas en inglés (*Internet of Things*), fue acuñado por primera vez en 1999 por Kevin Ashton [50] en el ámbito de la gestión de la cadena de suministro para ilustrar los beneficios de conectar las etiquetas de identificación por radio frecuencia (RFID) a Internet, de tal forma que sea posible realizar un seguimiento de las mercancías, sin necesidad de la intervención humana. Según Ashton, la esencia de IoT radica en que, si todos los objetos de la vida cotidiana estuvieran equipados con identificadores, sensores y conectividad, estos podrían ser gestionados por computadoras, de modo que puedan observar y comprender el mundo por sí mismos, en todo su esplendor aleatorio (gemelo digital) [50].

En la actualidad, IoT es uno de los paradigmas emergentes más relevantes dentro de las tecnologías de información, tal como lo confirma una encuesta publicada por Forbes Insights a más de 500 ejecutivos de una variedad de industrias reconocidas con sede en Europa, América y Asia-Pacífico, quienes consideran que IoT causará la mayor disrupción y brindará importantes oportunidades de negocio en el futuro [51][52]. En este sentido, el ritmo de crecimiento de IoT es exponencial, aumentando cada día su presencia en múltiples y variadas aplicaciones y aspectos de la vida diaria, con el único propósito de mejorar la forma en que trabajamos, interactuamos y vivimos [53][54]. En particular,

los dispositivos inteligentes constituyen la forma más visible de IoT, aplicada a una amplia gama de sectores o dominios [54]. Desde el punto de vista de los usuarios privados, los efectos más evidentes de IoT están en el ámbito doméstico y laboral, mediante la implementación de aplicaciones, tales como: domótica, vida asistida, e-salud, entretenimiento y ocupación en la vida. Desde la perspectiva empresarial y las ciudades, los escenarios más elocuentes están en la automatización industrial, logística, gestión de procesos de negocio, movilidad, servicios básicos, control ambiental, entre otros [5][6][7][55].

2.1.1 Definición de IoT

Con base en lo señalado, el ámbito de IoT es amplio y abarca una diversidad de tecnologías, constituyendo un concepto holístico y multidisciplinario, el cual ha sido definido y explicado numerosas veces en la literatura; sin embargo, no existe una definición única y universalmente aceptada para IoT [53]. Entre las principales definiciones, las cuales tienen una visión amplia de IoT, considerando sus más importantes rasgos y características se destacan:

1. *“Una red abierta e integral de objetos inteligentes que tienen la capacidad de autoorganizarse, compartir información, datos y recursos, reaccionando y actuando ante situaciones y cambios del entorno”* [1].
2. *“Una infraestructura de entidades físicas, sistemas y recursos de información interconectados junto con servicios inteligentes que pueden procesar información tanto del mundo físico como virtual; e, influir en las actividades del mundo físico”* [2].
3. *“Una infraestructura mundial para la sociedad de la información que propicia la prestación de servicios avanzados mediante la interconexión de objetos (físicos y virtuales) gracias a la interoperatividad de las tecnologías de información y comunicación presentes y futuras”* [3].

Las definiciones citadas y otras que se pueden encontrar en la literatura no necesariamente son contradictorias, sino que más bien resaltan diferentes aspectos asociados al fenómeno de IoT. Para el propósito de este trabajo de investigación, el término IoT se refiere a una infraestructura compuesta por un conjunto de objetos inteligentes, ubicuos (en cualquier lugar y momento) e interconectados mediante las tecnologías de información y comunicación para recolectar, transportar, analizar y almacenar datos del mundo físico y virtual, así como actuar con la mínima intervención humana. En este sentido, un sistema IoT está conformado tanto por entidades físicas como digitales. Las primeras representan cosas del mundo real que son monitoreadas por sensores y/o controladas por actuadores, pudiendo ser: seres vivos, objetos o el entorno (p. ej., edificios, personas, animales, vehículos, electrodomésticos, artículos, ropa, medio ambiente). Mientras que las entidades digitales, representan tecnologías de información y comunicación a nivel de hardware o software, tales como: servidores, microcontroladores, dispositivos IoT (sensores, tags, actuadores), middlewares, base de datos, servicios web, APIs y aplicaciones. En conjunción, una entidad física se extiende al mundo digital al integrarse con una o más entidades digitales, donde éstas últimas proporcionan las interfaces y mecanismos necesarios para obtener información sobre la entidad física, procesar dicha información y actuar sobre la misma (modificar su estado) a partir de instrucciones digitales [56].

Un aspecto recurrentemente abordado en varias definiciones de IoT, que merece especial atención, hace referencia a que un sistema IoT debe actuar autónomamente, es decir, en

gran medida sin la intervención de las personas. Por lo tanto, la autoconsciencia computacional pasa a ser un requisito prioritario en estos sistemas, de tal forma que puedan generar conocimiento sobre su propio estado y evolución, así como la de su entorno [57]; siendo al mismo tiempo, un facilitador para alcanzar otras propiedades deseables, como la autoadaptación, autoorganización, autoconfiguración y autorrecuperación (resiliencia ante fallos) [13].

2.1.2 Características de IoT

En esta subsección se describen las principales características que atañen a los sistemas IoT [2][10]:

- **Heterogeneidad.** Compuesto por un conjunto de dispositivos con variadas capacidades de procesamiento, almacenamiento, ancho de banda y energía. Además, estos dispositivos utilizan diferentes sistemas operativos, tecnologías de comunicación (p. ej., RFID, Wifi, Ethernet, Bluetooth, ZigBee, LoRaWAN), protocolos de aplicación (p. ej., CoAP, MQTT, HTTP), interfaces de programación (API) y estándares de intercambio de datos (p. ej., XML, JSON, RDF) [6][58].
- **Computación distribuida.** Conformado por una colección de nodos de computación, ubicados a nivel Edge, Fog y Cloud para alcanzar un equilibrio en el uso de recursos, que reduzca los costos de procesamiento y almacenamiento de los datos, la latencia y el tráfico de red, al mismo tiempo que, mejore la toma de decisiones y el control de los dispositivos cerca del borde de la infraestructura IoT. La búsqueda de este equilibrio puede llevar a escenarios de distribución muy variados, en los que se comparta en mayor o menor medida el procesamiento y almacenamiento de los datos, entre la nube y los nodos ubicados al borde de la red, ya sea en las puertas de enlace o incluso en los mismos sensores o actuadores, dependiendo de sus capacidades.
- **Escalabilidad.** Requieren expandir de forma transparente su infraestructura, descubriendo e incorporando en tiempo real nuevos dispositivos o servicios.
- **Ubicuidad.** Integración de los dispositivos inteligentes al entorno de las personas, acompañándolos en cualquier lugar y momento, de tal forma que no se perciban como objetos diferenciados.
- **Tiempo real (oportunidad).** Debido a que estos sistemas actúan en el mundo físico, la recolección y análisis de los datos sobre los eventos externos en progreso se produce en tiempo real, a fin de controlar y responder de manera oportuna (dentro de límites de tiempo específicos) a dichos eventos.
- **5Vs de los datos (volumen, velocidad, veracidad, variabilidad y variedad).** Esta característica es intrínseca a los sistemas de Big Data; sin embargo, a menudo ocurre que los sistemas IoT son fuentes de grandes volúmenes de datos, obtenidos a gran velocidad desde diversos lugares a través de enlaces de red, cuya veracidad debe ser validada (p. ej., debido a un mal funcionamiento de los sensores). Además, los sistemas IoT pueden contener una amplia variedad de tipos de datos dependiendo de las entidades físicas a monitorear. Finalmente, estos datos pueden ser almacenados de forma centralizada o distribuida, dependiendo de la naturaleza de los mismos, el procesamiento requerido y las características de los enlaces de comunicación.
- **Incertidumbre.** Operan en escenarios altamente cambiantes, siendo imposible identificar todos los requisitos en las etapas de desarrollo de un sistema IoT. En estos

sistemas la incertidumbre sobre su funcionamiento se ha convertido en suposiciones, ya que una funcionalidad del sistema puede conducir a diferentes resultados dependiendo de las circunstancias que pueden ocurrir durante su operación (p. ej., dispositivos o recursos que aparecen / desaparecen o que únicamente se conocen durante la ejecución del sistema).

- **Auto-* (Self-*).** Requieren reaccionar de manera autónoma ante una amplia gama de situaciones diferentes y sensibles al contexto (p. ej., ubicación, tiempo, preferencias de usuario, requisitos de servicio), a fin de minimizar la intervención humana.

2.1.3 Arquitectura de IoT

Al momento de construir un sistema IoT, la tarea de elegir o diseñar una arquitectura que organice adecuadamente los componentes de hardware, comunicación y software conforme a las necesidades y restricciones particulares del problema (p. ej., capacidad de los dispositivos IoT), resulta esencial para brindar servicios de calidad. Según lo señalado con anterioridad, el paradigma de IoT es tan amplio y cubre tantos dominios y tecnologías que no existe una visión unificada de un modelo de arquitectura de IoT [5]. En la literatura existen varias propuestas de arquitecturas de IoT basadas en capas, que se diferencian en la forma como distribuyen y desacoplan, en más o menos capas, las funciones de los componentes de un sistema IoT (sensores, actuadores, puertas de enlace, nube, etc.), dependiendo del nivel de abstracción a alcanzar, los problemas a resolver u objetivos a lograr con la arquitectura (p. ej., reducción de costos de procesamiento y almacenamiento, mejora de la latencia y tráfico de red, seguridad, etc.) [1][53][58].

Tradicionalmente, la nube se ha utilizado en los sistemas IoT para el procesamiento, análisis y almacenamiento de los datos. Sin embargo, debido al crecimiento exponencial de los dispositivos IoT y datos que estos generan [9], una solución arquitectónica basada únicamente en la nube puede resultar poco práctica para algunos escenarios de IoT, ya que los servidores en la nube suelen estar ubicados lejos de las fuentes de datos o dispositivos IoT, lo que provoca problemas de rendimiento, latencia y congestión en la red, confiabilidad y seguridad, principalmente en aplicaciones IoT en tiempo real [59]. Por lo tanto, la tendencia actual es migrar las tareas de procesamiento y almacenamiento de los datos desde la nube hacia el borde de la red de IoT, es decir, hacia los dispositivos IoT. En este sentido, como complemento a la computación en la nube (*Cloud Computing*), se han propuesto dos soluciones o paradigmas tecnológicos: computación en el borde (*Edge Computing*) y computación en la niebla (*Fog Computing*) [60][61][62]. A continuación, se describen las características de un modelo de arquitectura multicapa Edge-Fog-Cloud para sistemas IoT (ver Figura 2-1) [63][64][65][66]:

- **Capa Edge.** También conocida como capa de red periférica o de proximidad, contiene los dispositivos IoT que recopilan datos (sensores) o realizan una acción en el entorno (actuadores). Además, esta capa incluye las puertas de enlace (gateways) IoT a las que se conectan los dispositivos IoT para acceder a los recursos de las redes de área local (LAN) o amplia (WAN). Aunque estos nodos Edge tienen capacidades limitadas de hardware (procesamiento, almacenamiento, memoria, energía, etc.), la computación en el borde tiene como valor agregado procesar aplicaciones o servicios que requieren pocos recursos informáticos, que no dependen de los datos provistos por otros dispositivos externos y que necesitan una respuesta rápida en tiempo real. Esto evita que ciertos datos sean transportados a través de la red y procesados en niveles superiores por los nodos Fog o Cloud. De ahí que, la computación en el borde garantiza la toma de decisiones oportuna, equilibra la carga computacional y la

cantidad de datos transmitidos hacia y desde los nodos Fog o Cloud; y, otorga autonomía a los dispositivos IoT, especialmente cuando pierden la conexión a la red.

- **Capa Fog.** La computación en la niebla desacopla las funciones de hardware y software de la nube y acerca los recursos de computación hacia el borde de la red. Por lo tanto, esta capa contempla un conjunto de nodos de computación descentralizados, ubicados en redes de área local o amplia (antes de la nube) que actúan como instancias mediadoras entre la nube y los dispositivos Edge (sensores, actuadores, puertas de enlace IoT). Entre las principales funciones de la capa Fog se tienen: i) reconfiguración dinámica basada en métricas (p. ej., carga de procesamiento, volumen de datos, tiempo de respuesta, tráfico de red) sobre qué datos se almacenan y procesan en los nodos Fog y qué datos se preparan y envían a la nube para su almacenamiento y análisis posterior; ii) análisis de datos complejos que no pueden ser ejecutados por los nodos Edge, debido a sus limitaciones computacionales; iii) análisis de datos en un contexto amplio, es decir, basado en datos provenientes de diversos dispositivos IoT; y, iv) operaciones previas de transformación y agregación de datos, cuyos resultados son enviados automáticamente a la nube. En consecuencia, la computación en la niebla minimiza la latencia de las aplicaciones y servicios al ejecutarlas más cerca de las fuentes de datos, reduce el tráfico de red hacia la nube y los costos de procesamiento y almacenamiento en esta última.
- **Capa Cloud.** Ofrece recursos y servicios con altas prestaciones, bajo demanda y de forma escalable. A su vez, estos servicios pueden ser accedidos desde cualquier lugar y momento a través del Internet. En particular, la capa Cloud brinda soporte tanto a los nodos Edge como Fog, especialmente en la ejecución de tareas de procesamiento, análisis y almacenamiento de grandes volúmenes de datos (Big Data), cuyos requisitos exceden la capacidad de las capas inferiores de la infraestructura.

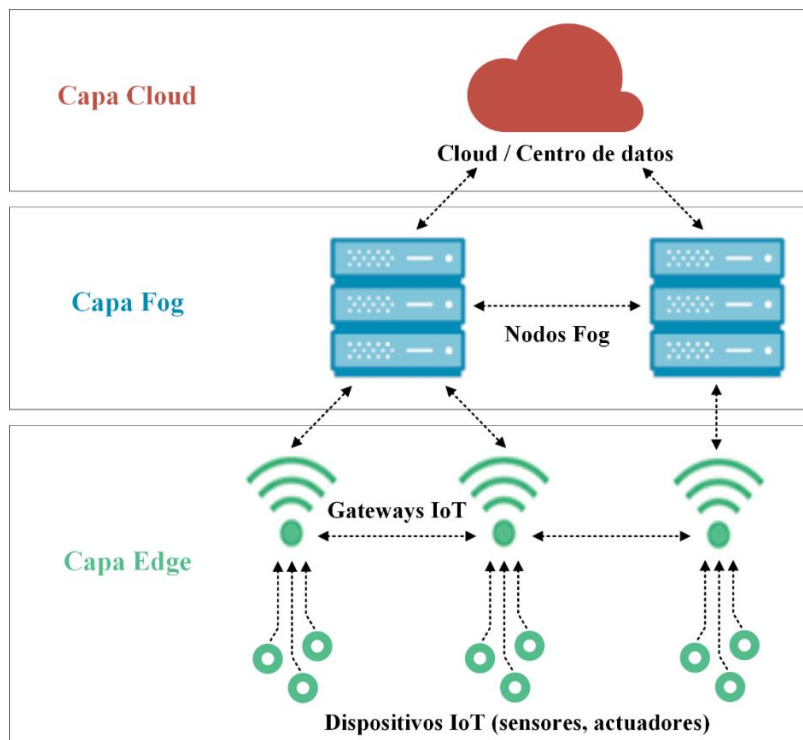


Figura 2-1. Modelo de arquitectura multicapa Edge-Fog-Cloud para IoT.

Fuente: Elaboración propia.

En conclusión, este modelo de arquitectura Edge-Fog-Cloud fomenta un equilibrio en el uso de los recursos distribuidos en las diversas capas, en procura de incrementar la eficiencia y efectividad de las aplicaciones y servicios de IoT. Asimismo, este modelo de arquitectura no tiene un único punto de fallo, al estar completamente descentralizada su operación en varios nodos; en contraposición, con una arquitectura soportada únicamente por la nube [64].

2.2 Autoconsciencia computacional

La complejidad de los sistemas modernos y particularmente del Internet de las Cosas, que a menudo es multifacética, abarcando altos niveles de descentralización, heterogeneidad, escalabilidad, dinamismo e incertidumbre, conducen a comportamientos durante el tiempo de ejecución que son difíciles de entender o predecir de forma a priori (etapa de desarrollo del sistema) [11]. A esto se suma, la ineficiencia de los métodos y herramientas de desarrollo de software actuales para gestionar la variedad y naturaleza cambiante de los requisitos de este tipo sistemas [67]. Una visión para afrontar este desafío es dotar a los sistemas de una mayor consciencia sobre sí mismo y el entorno, con el objeto de otorgarles la capacidad de tomar decisiones de forma inherente y autónoma [11].

Como antecedentes, la búsqueda de soluciones a la creciente complejidad de los sistemas informáticos no es nada nuevo. IBM en el año 2001, propuso la *computación autónoma*, un enfoque de inspiración biológica para desarrollar sistemas informáticos capaces de autogestionarse según los objetivos de alto nivel de los humanos [68]. En el “manifiesto” de IBM sobre la computación autónoma se menciona textualmente: “*para ser autónomo, un sistema informático necesita conocerse a sí mismo*” [69]. A su vez, en la literatura se han propuesto cuatro propiedades intrínsecas a los sistemas autónomos: autoconsciencia, consciencia del entorno, autocontrol y autoajuste [70][71][72]. Por lo tanto, los sistemas informáticos autoconscientes deben ser considerados como una subclase de sistemas informáticos autónomos [73].

Particularmente, la autoconsciencia computacional es un campo de investigación emergente que ha sido desarrollado a partir de la teoría de la autoconsciencia proporcionada por otras disciplinas, como la psicología y filosofía [74]. Los primeros intentos por abordar explícitamente la autoconsciencia, en un contexto informático, comenzaron alrededor del año 2004 [11]. Es así que, en un taller de DARPA [73] sobre “*Sistemas informáticos autoconscientes*” se discutieron varias de las contribuciones de una serie de investigadores cuyos documentos de posición propusieron desafíos claves para entender lo que podría significar la autoconsciencia en la computación.

En las siguientes subsecciones se estudian varios conceptos, tipos, niveles y enfoques de autoconsciencia computacional que han sido inspirados y adaptados desde la autoconsciencia humana.

2.2.1 Definición de sistemas informáticos autoconscientes

Según el diccionario de Oxford, la consciencia humana se define como “*conocimiento o percepción de una situación o hecho*”. Por lo tanto, informalmente se podría entender que los humanos acumulan conocimiento o se vuelven conscientes de las cosas, al percibir el mundo que los rodea. Por ejemplo, al observar los hechos de la realidad, escuchar a otras personas, leer libros; y, particularmente en los primeros años de vida, al aprender jugando.

Aunque existe una larga historia sobre el estudio de la naturaleza del yo en la filosofía, a principios del siglo XX [75][76], la psicología introdujo el concepto de autoconsciencia para comprender la variedad de conocimientos que los individuos poseen sobre sí

misimos. Quizás el momento más importante es cuando James [77] hace la distinción e interrelación entre las dos formas de “yo” propuestas por Hume y Kant. Por una parte, la visión Humeana (emocional) [78], el “yo implícito” o yo como sujeto de experiencias, consiste en una forma de autoconsciencia adquirida desde las experiencias o percepciones del individuo. Estas experiencias son exclusivas del individuo y, desde el propio punto de vista (subjetivo) del individuo, están determinadas por factores como su aparato sensorial, su situación dentro del mundo y otros factores asociados con su propio estado. Por otra parte, la visión Kantiana (racional) [79], el “yo explícito” o yo como objeto de conocimiento, engloba una forma de autoconsciencia basada en el conocimiento obtenido a partir de reconocer, modelar y razonar la información sobre las experiencias y los conceptos mantenidos en la mente e imaginación del individuo. Contrastando estas dos visiones, la consciencia que un individuo tiene de su yo explícito se considera una forma más avanzada de autoconsciencia, la cual está basada en la autoconsciencia implícita [74].

Particularizando el estudio al área de la informática, varias comunidades de investigación se han involucrado con la noción de la autoconsciencia computacional desde su propia perspectiva, tales como: computación autónoma, aprendizaje automático, inteligencia artificial, sistemas multiagentes, sistemas autoadaptativos, sistemas sensibles al contexto, computación reflexiva y *models@run.time* [19]. Sin embargo, varias revisiones [23][80][40] sobre autoconsciencia computacional encontraron que el término carecía de una definición precisa y consensuada que se aplique ampliamente en una multitud de dominios. En el año 2015 se organizó el Seminario Dagstuhl 15041, denominado “*Algoritmos y arquitecturas dirigidas por modelos para sistemas informáticos autoconscientes*” [81], el cual reunió a importantes investigadores pertenecientes a diferentes comunidades y que están relacionados con diversos aspectos de la autoconsciencia computacional.

Un aporte relevante de este seminario fue la formulación de una definición amplia sobre sistemas informáticos autoconscientes que integra las diferentes formas en que se utiliza este término en el ámbito de la investigación interdisciplinaria. Esta definición ha sido adoptada como guía para este trabajo de investigación y manifiesta textualmente [81]:

“*Los sistemas informáticos autoconscientes son sistemas informáticos que:*

- 1) *aprenden modelos que capturan conocimiento sobre sí mismos y su entorno (como su estructura, diseño, estado, posibles acciones y comportamiento en tiempo de ejecución) de forma continua; y,*
- 2) *razonan utilizando los modelos (p. ej., predecir, analizar, considerar y planificar), permitiéndoles actuar basados en su conocimiento y razonamiento (p. ej., explorar, explicar, informar, sugerir, autoadaptar o impactar su entorno),*

de acuerdo con los objetivos de nivel superior, los cuales pueden estar también sujetos a cambios”.

En la definición, el término “*modelo*”, en su sentido general, hace referencia a una representación abstracta y simplificada de cualquier aspecto del sistema, parte de él o su entorno que captura conocimiento y se puede utilizar para verificar, asegurar y alcanzar los objetivos del sistema [19]. Los modelos según su propósito se pueden clasificar en: i) *modelos descriptivos*, capturan un aspecto del sistema, tal y como es en un instante dado (p. ej.: capturar el estado de los recursos del sistema en un momento dado); ii) *modelos prescriptivos*, describen futuros previstos, es decir, cómo el sistema debería ser (p. ej.: cómo actuar después de un fallo de un componente); y, iii) *modelos predictivos*, admiten un razonamiento más complejo, como predecir el comportamiento del sistema en

determinadas condiciones o el impacto de una posible acción de adaptación considerada (p. ej. predecir el rendimiento del sistema para una determinada carga de trabajo) [19][82].

Por su parte, el término “*aprender*” implica que un sistema autoconsciente debe ser capaz de generar conocimiento de forma continua y a diferentes niveles (ver Subsección 2.2.2) por medio de la observación inmediata (percepción) y la reflexión. En primera instancia, el conocimiento subjetivo del sistema es adquirido directamente de los datos empíricos obtenidos desde los sensores físicos o virtuales (aparato sensorial) durante el tiempo de ejecución. En segunda instancia, un conocimiento más profundo es alcanzado mediante la construcción, refinamiento, calibración y/o aplicación de modelos conceptuales o reflexivos en tiempo de ejecución, los cuales utilizan la información histórica recolectada desde los sensores o procesada previamente. Cabe recalcar que además de la información obtenida directamente en tiempo de ejecución, se puede emplear información estática incorporada a los modelos durante el diseño del sistema [19].

Finalmente, el término “*razonar*” conlleva a que un sistema autoconsciente debe poseer una capacidad de análisis para contrastar el conocimiento extraído de los modelos aprendidos con sus objetivos a lograr, siendo el mecanismo para alcanzar una comprensión cuantitativa y cualitativamente sobre sí mismo y su entorno. Los resultados de este análisis pueden ser utilizados por el sistema para explicar o informar sobre su estado o probable evolución, para recomendar una acción o incluso para autoadaptarse ante los cambios, aunque este último no forma parte del alcance de la autoconsciencia [19].

2.2.2 Autoconsciencia pública y privada

Los conceptos sobre autoconsciencia pública y privada se traducen a un contexto informático de la siguiente manera [74][83]:

- **Autoconsciencia privada:** Capacidad de un sistema para obtener conocimiento basado en fenómenos internos a él mismo. Por lo tanto, un sistema necesita sensores internos para lograr este objetivo.
- **Autoconsciencia pública:** Es la capacidad de un sistema para obtener conocimiento basado en fenómenos externos a él mismo. Dicho conocimiento depende de cómo el propio sistema observa, percibe y mide los aspectos de su entorno (requiere sensores externos). Este conocimiento incluye la comprensión de la situación del sistema en el entorno, así como el rol que desempeña y el impacto (potencial) que tiene en el mismo.

Estos dos tipos de autoconsciencia son complementarios, ya que el conocimiento particular que un sistema tiene sobre su entorno social y físico puede sintetizarse con la autoconsciencia privada, con el fin de producir modelos conceptuales completos e integrales [74].

2.2.3 Niveles de autoconsciencia computacional

En la Tabla 2-1 se describen los cinco niveles de autoconsciencia computacional propuestos por Lewis et al. [74][83]. Estos niveles han sido mapeados y formulados a partir de los niveles de consciencia humana de Neisser [84]. A su vez, se detalla la relación entre los niveles de autoconsciencia y los conceptos de autoconsciencia pública y privada como fuentes de conocimiento relevantes basadas en sensores internos y externos, respectivamente (ver Figura 2-2).

Tabla 2-1. Niveles de autoconsciencia computacional [74].

Niveles de Lewis	Niveles de Neisser	Definición
Consciencia de estímulo	Yo ecológico (Ecological self)	El sistema conoce los estímulos que actúan sobre él, lo que le permite responder a los eventos. Sin embargo, no tiene conocimiento de estímulos pasados/futuros. Como los estímulos pueden originarse tanto interna como externamente, la consciencia del estímulo puede ser privada, pública o ambas.
Consciencia de interacción	Yo interpersonal (Interpersonal self)	El sistema puede aprender que los estímulos y sus propias acciones constituyen interacciones con otros sistemas y el entorno. Esta consciencia se basa típicamente en fenómenos externos, siendo de carácter pública. Aunque un sistema que aprende sobre la causalidad en las interacciones internas consigo mismo, constituye una forma de autoconsciencia privada.
Consciencia del tiempo	Yo extendido (Extended self)	El sistema puede obtener conocimiento sobre fenómenos históricos y/o probables futuros. La implementación de esta consciencia puede requerir que el sistema utilice memoria explícita, modelos de series de tiempo (regresión) o anticipación. Dado que la consciencia del tiempo puede aplicarse tanto a fenómenos internos como externos, puede ser privada, pública o ambas.
Consciencia de objetivos	Yo privado (Private self)	El sistema puede adquirir conocimiento sobre las metas, requisitos, preferencias y limitaciones actuales. En este sentido, el sistema tiene acceso a sus objetivos de forma que puede razonar sobre ellos o manipularlos. Cuando se combinan la consciencia de objetivos con la de interacción o tiempo, el sistema tiene la capacidad de razonar sobre las metas en relación con otros individuos o sobre metas futuras probables, respectivamente. Puesto que los objetivos pueden existir de forma privada para el sistema o colectivamente compartidos con otros sistemas, esta consciencia puede ser privada, pública o ambas.
Meta-autoconsciencia	Yo conceptual (Conceptual self)	El sistema puede obtener conocimiento de sus propios niveles de consciencia y de cómo estos se ejercitan. Esta consciencia permite que el sistema razone sobre los beneficios y costos de mantener ciertas capacidades y niveles de autoconsciencia. En vista de que la meta-autoconsciencia se ocupa únicamente del conocimiento de los procesos internos, constituye una forma de autoconsciencia privada.

2.2.4 Autoconsciencia en sistema colectivos

La autoconsciencia también puede estar presente a nivel individual o colectivo. El primer caso se refiere a la consciencia en el ámbito de un único sistema, donde el autoconocimiento se encuentra en un solo lugar [74]. Mientras que, el segundo caso se relaciona con la consciencia en sistemas colectivos compuestos por múltiples

subsistemas, donde las interacciones entre estos proporcionan un aprendizaje distribuido [74]. En estos sistemas colectivos no existe un punto central en el que se localiza la consciencia, sino más bien está distribuida en todas sus entidades [85]. Por lo tanto, la autoconsciencia colectiva se genera mediante una estrategia de abajo hacia arriba (Bottom – Up), es decir, las diferentes entidades dentro del colectivo aportan con consciencia de partes relevantes, las cuales se agregan y acumulan de manera estadística para dar lugar a una autoconsciencia de todo el sistema [85][86]. Luego, este conocimiento global es retroalimentado para que sea utilizado colectivamente por todas las entidades, de tal manera que todo el sistema tenga un sentido coherente y útil con su propio estado [85]. La Figura 2.1 ilustra conceptualmente como un sistema colectivo que comprende varios subsistemas puede verse como un sistema de consciencia global producto de la combinación y agregación de la consciencia de sus partes.

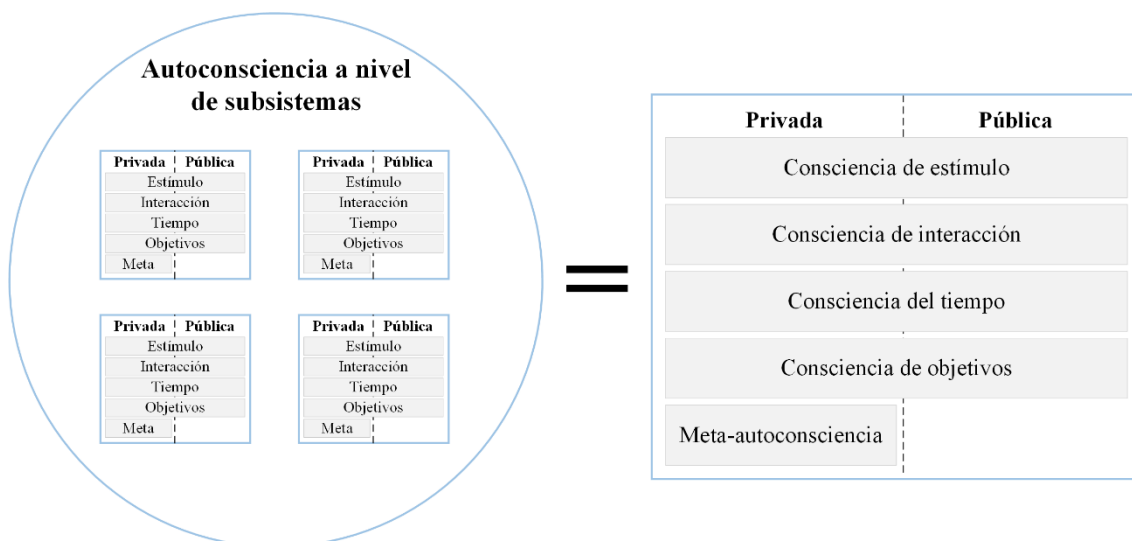


Figura 2-2. Relación entre los tipos y niveles de autoconsciencia computacional.

Fuente: Lewis et al., 2016 [83].

Finalmente, según Lewis et al. [74], la autoconsciencia descentralizada puede proporcionar mayor solidez y adaptabilidad al cambio que la autoconsciencia centralizada.

2.2.5 Bucle de aprendizaje y razonamiento basado en modelos para sistemas autoconscientes (LRA-M loop)

En la Figura 2-3 se presenta el bucle de aprendizaje y razonamiento basado en modelos (LRA-M loop) [19] que captura los principales procesos de un sistema informático autoconsciente. En la figura se puede observar que los procesos de aprendizaje y razonamiento dentro del sistema (“yo”) están impulsados por sus objetivos y las observaciones recopiladas en forma de datos empíricos sobre el propio sistema, sus usuarios, otros sistemas y el entorno en general. Por su parte, el proceso de aprendizaje continuo utiliza los datos empíricos para abstraer en los modelos un conocimiento más profundo sobre los aspectos relevantes del sistema y su entorno (p. ej., estructura, estado, posibles acciones y comportamiento en tiempo de ejecución). Los modelos aprendidos y el conocimiento que estos generan son la base para la ejecución del proceso de razonamiento. Este último proceso puede desencadenar acciones que afecten al comportamiento del sistema, al entorno; e, incluso a los procesos de aprendizaje y razonamiento. Sin embargo, la autoadaptación no es estrictamente obligatoria en la autoconsciencia, en su defecto un sistema informático autoconsciente puede únicamente

proporcionar explicaciones o recomendaciones sobre cómo actuar, pudiendo quedar la decisión final sobre qué acción tomar bajo la responsabilidad del usuario [19].

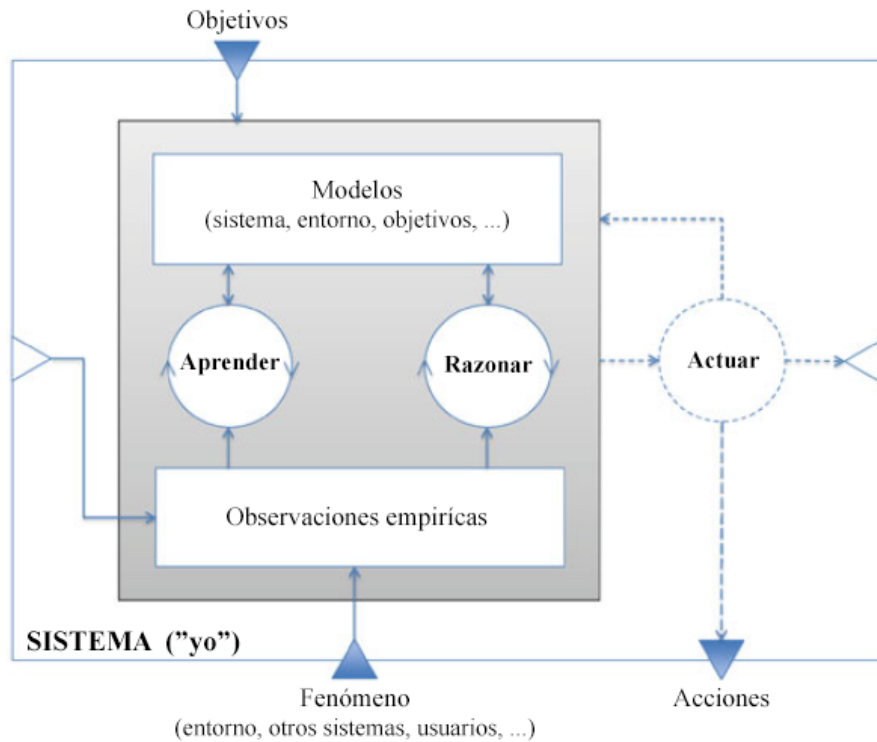


Figura 2-3. Bucle de aprendizaje y razonamiento basado en modelos para sistemas autoconscientes (LRA-M loop). Fuente: Kounev et al., 2017 [19].

2.2.6 Framework conceptual de autoconsciente computacional

En esta subsección se presenta el framework conceptual propuesto por Lewis et al. [20], el cual está conformado por un conjunto estructurado de conceptos técnicos que describen los tipos de capacidades de autoconsciencia que pueden ser deseables o estar presentes en los sistemas informáticos. Este framework está inspirado en la variedad de formas de autoconsciencia presentes en el ser humano; sin embargo, adopta un enfoque más inclusivo y extensible a un contexto informático, ya que no vincula directamente los niveles de autoconsciencia computacional con los niveles psicológicos de Neisser (como lo hace Lewis et al. en [74]).

Particularmente, el framework se basa en los siguientes conceptos de alto nivel: i) niveles de autoconsciencia; ii) aspectos de autoconsciencia reflexiva; y, iii) dominio (sujeto y objeto) de autoconsciencia. Estos conceptos se detallan en los siguientes apartados y pueden ser usados en conjunto para caracterizar las diferentes formas de procesos de autoconsciencia que podría poseer un sistema informático [20].

2.2.6.1 Niveles globales de autoconsciencia

El framework distingue tres niveles globales de autoconsciencia, mismos que se analizan a continuación [20]:

- **Autoconsciencia pre-reflexiva.** Consiste en la consciencia subjetiva que tiene un sistema a partir de su capacidad sensorial para percibir y hacer observaciones (sensores internos y externos). Este nivel se alinea con la consciencia de estímulo de Lewis et al. [74] y contempla únicamente la presencia de conocimiento subjetivo,

obtenido sin la aplicación de modelos internos o reflexivos basados en datos históricos. En términos de acción un sistema con autoconsciencia pre-reflexiva se limita a reglas de estímulo - acción, ya que cualquier razonamiento, si aún se le puede llamar así, está basado únicamente en la percepción u observación inmediata, más que en modelos internos preexistente (ya que no existen). Por lo tanto, en este nivel, el sistema no tiene ninguna conceptualización de sus experiencias, simplemente siente por sí mismo y posiblemente actúa.

La autoconsciencia pre-reflexiva incluye implícitamente la noción de subjetividad. En este sentido, dos sistemas diferentes, con sensores diferentes, ubicados en lugares diferentes detectarán el mismo fenómeno de manera diferente, es decir, obtendrán datos empíricos diferentes. De ahí que, en este nivel es crucial considerar los metadatos que describen el contexto en que se obtienen los datos empíricos; e, incluso estos metadatos deben ser utilizados en niveles avanzados de autoconsciencia, de tal forma que conduzcan a resultados más exactos en términos de conocimiento o acción. En definitiva, la autoconsciencia pre-reflexiva es una forma mínima de autoconsciencia computacional, siendo un requisito previo para lograr las subsiguientes formas de autoconsciencia.

- **Autoconsciencia reflexiva.** Proceso continuo y evolutivo capaz de producir una conceptualización (es decir, un modelo) de los conocimientos y experiencias del sistema. En otras palabras, a medida que el sistema opera a lo largo del tiempo y experimenta con su entorno a través de sus sensores, los procesos de autoconsciencia reflexiva construyen y refinan los modelos conceptuales mediante los datos empíricos y subjetivos recopilados por estos sensores, así como los metadatos que describen el contexto en que se obtuvieron. Al igual que ocurre en la autoconsciencia humana, los modelos o conceptualizaciones pueden capturar aspectos diferentes de una experiencia u objeto, tales como: estado, comportamiento, evolución en el tiempo, causalidad, etc. Estos aspectos de autoconsciencia se describen en la Subsección 2.2.6.2.
- **Autoconsciencia meta-reflexiva.** También conocida como meta-autoconsciencia, es en esencia un subconjunto de la autoconsciencia reflexiva, donde el objeto de la reflexión es un proceso de autoconsciencia reflexiva. En consecuencia, un sistema con consciencia meta-reflexiva es capaz de obtener conocimiento sobre los beneficios y costos de sus propios procesos de consciencia, a fin de determinar si sería conveniente mantenerlos, adaptarlos o cambiarlos.

2.2.6.2 Aspectos de autoconsciencia reflexiva

Los procesos reflexivos construyen modelos conceptuales que capturan diferentes aspectos de autoconsciencia sobre las experiencias y objetos de un sistema. Estos aspectos son considerados un conjunto más detallado de niveles de autoconsciencia, los cuales incluso se pueden combinar para formar capacidades complejas de autoconsciencia. Entre los principales aspectos de autoconsciencia abordados en el framework de Lewis et al. [20] se tienen:

- **Consciencia de identidad.** Capacidad de reconocer la identidad y los roles de las entidades del mundo real (posiblemente a lo largo del tiempo), tales como: humanos, objetos y otros sistemas.
- **Consciencia de estado.** Capacidad de reconocer los estados del propio sistema y su entorno.

- **Consciencia del tiempo.** Abarca el conocimiento histórico o futuro de fenómenos potenciales. Al combinar con otros aspectos de autoconsciencia se puede modelar la consciencia de la naturaleza temporal de estos. Por ejemplo, el sistema puede tener consciencia de identidades, estados e interacciones anteriores o futuras.
- **Consciencia de interacción.** Contempla el conocimiento sobre los patrones de interacción entre entidades. De ahí que, un sistema con este tipo de consciencia es capaz de reconocer que algunas acciones forman parte de interacciones que se encuentran casualmente conectadas entre ellas. Un ejemplo de esto, es el intercambio de mensajes entre entidades, donde un mensaje puede ser una respuesta a otro, en lugar de una acción aislada.
- **Consciencia de comportamiento.** Comprende el conocimiento sobre una acción o grupo de acciones realizadas por una entidad. Sin embargo, para que la consciencia del comportamiento sea más significativa, debería ir acompañada de la consciencia de identidad, tiempo y estado.
- **Consciencia de objetivos.** Capacidad de conceptualizar las metas, objetivos, motivaciones y limitaciones que impulsan el comportamiento de un sistema. Este tipo de consciencia puede hacer referencia a objetivos del propio sistema o del entorno (p. ej., usuarios u otros sistemas). Al combinarse con la consciencia de tiempo, el sistema puede llegar a entender cómo los objetivos evolucionan o cambian en el tiempo.

2.2.6.3 Dominio de autoconsciencia

En un sistema informático autoconsciente hay varias entidades involucradas en un aspecto particular de autoconsciencia. Estas entidades existen a diferentes niveles de abstracción o agregación, abarcando el sistema que es consciente, las partes del sistema que colaboran en proveer la consciencia y las cosas de las que es consciente. En el framework de Lewis et al. [20], este conjunto de entidades se denomina *dominio de autoconsciencia* y está conformado por dos conceptos relacionados:

- **Sujeto (Span) de autoconsciencia.** Entidad que es consciente, que hace la reflexión; y, en la que el conocimiento está disponible.
- **Objeto (Scope) de autoconsciencia.** Entidad sobre la que se reflexiona. Un sistema puede ser consciente de una amplia gama de objetos a lo largo de su vida. Por ejemplo, personas o cosas percibidas externamente, el aparato sensorial del sistema, los procesos internos de autoconsciencia, entre otros.

Finalmente, mediante la combinación de los tres conceptos o principios estudiados en este framework (niveles, aspectos y dominio de autoconsciencia) se pueden construir descripciones de autoconsciencia para un sistema informático. Por ejemplo, un sistema remoto de asistencia médica (sujeto) conoce el nivel de consumo de energía (aspecto) de un sensor de presión sanguínea (objeto) durante un periodo de tiempo de recolección de datos (aspecto).

2.3 Ingeniería Dirigida por Modelos (MDE)

Elevar el nivel de abstracción en el proceso de desarrollo de software ha sido una de las mayores preocupaciones de la comunidad científica y la industria frente a la creciente complejidad de los sistemas informáticos. El modelado es una de las formas de abstracción más ampliamente utilizadas en la ciencia y la tecnología [28][87]. En cualquier área de la ingeniería se crean modelos al momento de analizar y diseñar

sistemas complejos. Los modelos abstraen los detalles de interés de un sistema y permiten abordar las inquietudes sobre dicho sistema, en busca de respuestas y soluciones efectivas [28].

Particularmente, en la ingeniería de software el uso de modelos es una tradición de larga data y se volvió más popular desde el desarrollo del Lenguaje Unificado de Modelado (UML) [28]. Sin embargo, los modelos han sido utilizados generalmente con fines meramente de documentación, siendo la relación entre los modelos y la implementación del software de carácter intencional y no formal [28].

Con la aparición de la Arquitectura Dirigida por Modelos (Model-Driven Architecture - MDA) propuesta por el Object Management Group (OMG) en el año 2000 [88], los modelos han alcanzado un papel principal en el proceso de desarrollo de software, frente a las propuestas tradicionales basadas en lenguajes de programación y plataformas de objetos y componentes software [89]. Además, a raíz de este hito, ha crecido el interés de la comunidad de software en el uso sistemático y formal de los modelos en las diferentes etapas del ciclo de vida del software, dando lugar a un conjunto de paradigmas de desarrollo software bajo el nombre de Ingeniería Dirigida por Modelos (Model-Driven Engineering - MDE) [89].

Desde una perspectiva formal, MDE es una disciplina de la ingeniería de software orientada a proponer enfoques de desarrollo de software en los que se crean modelos abstractos de sistemas de software y se transforman sistemáticamente en implementaciones concretas (p. ej., código fuente, archivos de configuración) [90]. En este sentido, los conceptos centrales de MDE son los modelos y las transformaciones (operaciones de manipulación de modelos), es decir, bajo el contexto de MDE, estos conceptos se relacionan mediante la ecuación [87]:

$$\text{Modelos} + \text{Transformaciones} = \text{Software} \quad (2.1)$$

En donde, el sistema se modela en términos del dominio del problema. Luego, los modelos mediante transformaciones descienden en el nivel de abstracción (agregando nuevos detalles de la plataforma destino a los modelos) hasta llegar al código del sistema en desarrollo.

En consecuencia, MDE incrementa la importancia de los modelos, ya que no solo se utilizan con fines de documentación y comunicación, sino como artefactos centrales (ciudadanos de primera clase) del proceso de desarrollo de software [28][29][91]. Además, MDE abarca diversas técnicas que cubren todo el espectro de actividades de desarrollo de software, incluyendo: ingeniería de requisitos basada en modelos, diseño basado en modelos, generación de código a partir de modelos, pruebas basadas en modelos, evolución del software basada en modelos, entre otras [28].

Un enfoque de MDE se considera efectivo, si los modelos abstractos son comprensibles y tienen sentido para los expertos del dominio, pudiendo al mismo tiempo ser interpretados por un computador para implementar los artefactos de software [92]. De ahí que, MDE tiene un gran potencial para: i) reducir la brecha entre el dominio del problema y el dominio de implementación; ii) proporcionar un mayor nivel de abstracción y automatización del proceso de desarrollo; iii) mejorar la gestión de la complejidad y el cambio del software; e, iv) incrementar la productividad en general [90][93][94][95]. Estos beneficios no solo son aplicables en la construcción de nuevas aplicaciones sino en la reingeniería de aplicaciones heredadas y la configuración dinámica de sistemas en tiempo de ejecución [96].

2.3.1 Paradigmas de MDE

En la Ingeniería Dirigida por Modelos se pueden distinguir diferentes paradigmas, los cuales se agrupan en tres categorías principales [89]:

- **Desarrollo de software dirigido por modelos (MDD).** Ingeniería directa dirigida por modelos para crear nuevas aplicaciones. Entre los paradigmas inmersos en esta categoría se destacan la Arquitectura Dirigida por Modelos (MDA), el Desarrollo Específico del Dominio (DSM), también denominado “programación orientada a los lenguajes” y las factorías de software.
- **Reingeniería o modernización dirigida por modelos.** En este caso, los modelos son utilizados para representar el conocimiento extraído de los procesos de evolución de los sistemas existentes, a fin de mejorar su calidad. A su vez, una cadena de transformaciones de modelos automatiza los procesos de ingeniería inversa y generación de los artefactos de software del nuevo sistema. Una iniciativa particular destinada a definir metamodelos estándar para la modernización de software es ADM (*Architecture-Driven Modernization*), misma que ha sido propuesta por el consorcio OMG [97].
- **Modelos en tiempo de ejecución (models@run.time).** Este paradigma extiende la aplicabilidad de los modelos de software producidos durante la etapa de desarrollo (MDD) al entorno de ejecución. De hecho, los modelos son utilizados para razonar acerca del estado, comportamiento y evolución del sistema y su entorno operativo, proporcionando una base semántica para la toma de decisiones en tiempo de ejecución. En la subsección 2.3.3 se describe con mayor profundidad este paradigma.

2.3.2 Principios básicos de MDE

Todos los paradigmas de MDE comparten un conjunto de principios básicos o elementos fundamentales, mismos que se describen a continuación [87][89][93]:

- **Modelo.** Representación abstracta de un aspecto o incumbencia de un sistema de software. Los modelos de software deben ser diseñados o escritos en un lenguaje bien definido, es decir, conforme a una sintaxis y notación descrita formalmente (p. ej., mediante un metamodelo), a fin de que puedan ser interpretados por un computador para generar código o controlar un sistema. Entre los aspectos que se pueden especificar en un modelo de software se tienen: requisitos, estructura, estado o comportamiento.
- **Lenguaje Específico de Dominio (DSL).** Es un lenguaje diseñado específicamente para un determinado dominio, de tal forma que se puedan expresar soluciones de software con un alto nivel de abstracción. Un DSL consta de tres componentes principales: i) *sintaxis abstracta (metamodelo)*, define los conceptos del lenguaje y las relaciones entre estos, así como las reglas que establecen cuándo un modelo está bien formado; ii) *sintaxis concreta*, establece la notación del lenguaje; y, iii) *semántica*, usualmente definida a través de la traducción a conceptos de otro lenguaje destino cuya semántica se conoce (p. ej., Java) [89]. La notación o sintaxis concreta de un DSL puede ser textual o gráfica; sin embargo, la tendencia es crear DSLs híbridos que combinen texto y gráficos. Si un DSL está destinado al modelado, también se lo denomina *Lenguaje de Modelado Específico de Dominio* (DSML). Como ejemplos de DSLs comúnmente utilizados se tienen: HTML para el desarrollo

de páginas web, VHDL para la descripción de hardware; o, SQL para definición y manipulación de bases de datos.

- **Metamodelo.** Modelo conceptual que permite formalizar la definición (sintaxis abstracta) de un DSL. El metamodelo especifica los conceptos de un lenguaje, las relaciones entre estos conceptos y las reglas estructurales que limitan y determinan la validez de un modelo. Por consiguiente, un modelo es conforme a un metamodelo que establece su estructura. Actualmente, Ecore es el lenguaje de metamodelo más utilizado, mismo que está basado en Meta-Object Facility (MOF) [98]. En Ecore los conceptos del lenguaje se representan como metaclases del metamodelo, las propiedades de un concepto como atributos de una metaclase; y, las relaciones entre conceptos como referencias, agregaciones o generalizaciones entre metaclases [89]. A su vez, Ecore constituye el elemento central de las herramientas Obeo Designer Community Edition y Eclipse Sirius [99], las cuales proporcionan una infraestructura adecuada para crear soluciones de MDE, aprovechando las tecnologías de modelado incluidas en Eclipse Modeling Framework (EMF) y Graphical Modeling Framework (GMF) [100].
- **Transformaciones.** Conformado por motores o generadores de transformación que analizan ciertos aspectos de los modelos y luego los traducen a diferentes tipos de artefactos, como: código fuente, entradas de simulación, descripciones de uso XML o representaciones alternativas de los modelos [93]. Estas herramientas deben garantizar la coherencia entre los artefactos generados y los requisitos funcionales y no funcionales contenidos en los modelos. Existen tres tipos de transformaciones: modelo a texto (M2T), modelo a modelo (M2M) o texto a modelo (T2M). En síntesis, mientras que los modelos elevan el nivel de abstracción, las transformaciones aumentan el nivel de automatización del proceso de desarrollo de software.

En la Figura 2-4 se presenta un esquema que sintetiza la relación entre los elementos fundamentales de MDE descritos anteriormente.

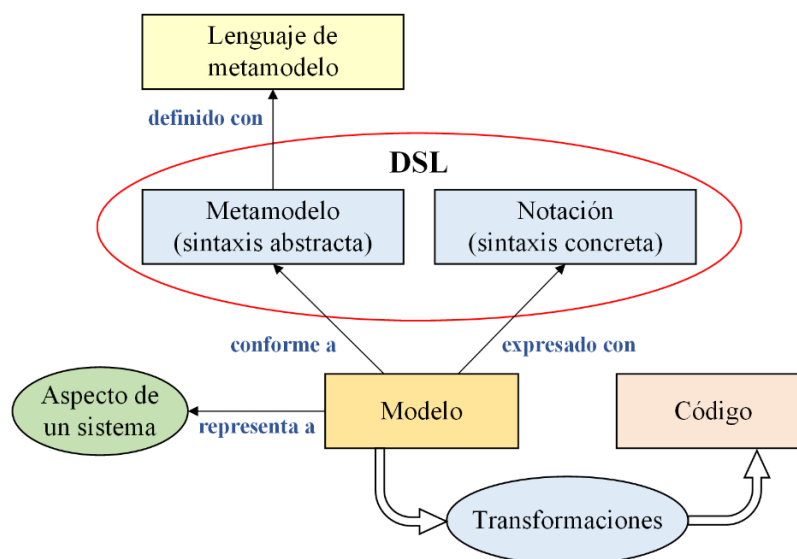


Figura 2-4. Elementos fundamentales de MDE.
Fuente: García et al., 2012 [89].

2.3.3 Modelos en tiempo de ejecución (models@run.time)

En la ingeniería de software tradicional las fases de desarrollo y ejecución del software se encuentran estrictamente separadas entre sí; y, los cambios del software en ejecución se realizan mediante la reimplementación de los artefactos de software [30]. Sin embargo, debido a la criticidad, complejidad y dinamismo creciente de los sistemas modernos, la distinción entre las fases de desarrollo y ejecución tiende a diluirse [101]. De hecho, estos sistemas requieren ser permanentemente reconfigurables y adaptables; no obstante, deben estar permanentemente disponibles, no siendo posible detener su ejecución para adaptar manualmente su funcionalidad [89].

Un enfoque adecuado para superar estas limitaciones es reutilizar los artefactos de modelado de las fases de desarrollo al tiempo de ejecución [31][102]. La información del sistema en ejecución puede utilizarse para retroalimentar los modelos de software. Luego, estos modelos pueden ser analizados en tiempo de ejecución para comprender el estado y la evolución del sistema; y, en caso de ser necesario tomar acciones correctivas a nivel de modelo [30]. Por lo tanto, al igual que los modelos de desarrollo tradicional, los modelos en tiempo de ejecución también permiten razonar sobre el sistema [89]. Por ejemplo, los usuarios del sistema pueden utilizar modelos en tiempo de ejecución para dar soporte al monitoreo y el control del sistema, observar el comportamiento del sistema en busca de patrones de conducta, apoyar la generación automática de artefactos de implementación para incorporarlos al sistema, todo esto durante el tiempo de ejecución [103]. En este sentido, el uso de los modelos en tiempo de ejecución contribuye a trasladar varias de las funcionalidades que tradicionalmente se programan en el momento del desarrollo al tiempo de ejecución, lo cual indudablemente mejora la gestión de la complejidad, la incertidumbre y el cambio del software [33]. Además, realizar los cambios a nivel de los modelos en tiempo de ejecución mejora la sincronización entre los artefactos de diseño y la implementación del sistema [104].

Por lo expuesto, los modelos en tiempo de ejecución son candidatos idóneos para soportar las propiedades auto-* (self-*) de los sistemas informáticos, tales como: autoconsciencia, autoadaptación, autoorganización, autoconfiguración y autorrecuperación [35].

2.3.3.1 Definición de los modelos en tiempo de ejecución

Con base en la utilización mencionada de los modelos, un modelo en tiempo de ejecución se define como: *“Una autorrepresentación causalmente conectada del sistema asociado que enfatiza la estructura, el comportamiento y los objetivos del sistema desde una perspectiva del espacio del problema”* [31].

Los modelos en tiempo de ejecución combinan los principios de la reflexión computacional con la Ingeniería Dirigida por Modelos. La reflexión computacional es la capacidad de un sistema computacional para observar y opcionalmente modificar su estructura, en otras palabras, es una actividad computacional que razona sobre su propia computación [34]. Por lo tanto, considerando el principio de reflexión computacional, los modelos en tiempo de ejecución deben representar a un sistema asociado en operación, de tal manera que reflejen constantemente su estado y comportamiento actual; es decir, si el sistema cambia, las representaciones del sistema (los modelos) también deberían cambiar y viceversa [31]. Sin embargo, a diferencia de la reflexión computacional, en los modelos en tiempo de ejecución se buscan representaciones del sistema (modelos) con un alto nivel abstracción, particularmente modelos relacionados con el espacio del problema, siendo concordante con los objetivos de MDE [31] (ver Figura 2-5).

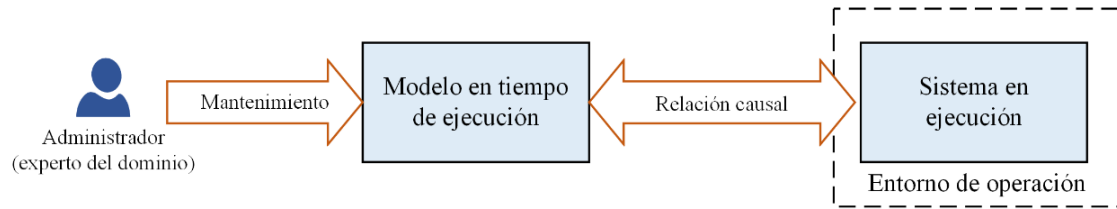


Figura 2-5. Relación causal entre un modelo en tiempo de ejecución y el sistema asociado. Fuente: Elaboración propia.

Otro aspecto clave de los modelos en tiempo de ejecución consiste en que estos deben estar intrínsecamente vinculados con los modelos producidos durante el proceso de desarrollo de software basado en MDD [89].

2.3.3.2 Dimensiones de los modelos en tiempo de ejecución

Los modelos pueden representar una variedad de aspectos o incumbencias de un sistema asociado, cuyas especificaciones pueden ser consultadas y actualizadas en tiempo de ejecución. Los modelos utilizados en tiempo de ejecución se categorizan en las siguientes dimensiones [31]:

- **Estructura versus comportamiento.** Los modelos comúnmente describen la estructura (p. ej., componentes, objetos, relaciones) o el comportamiento (p. ej., flujos de eventos) asociado al sistema.
- **Procedural versus declarativo.** Otra dimensión relevante es si el modelo es procedural, representando la estructura y comportamiento actual del sistema; o, declarativo, especificado en términos de los objetivos o metas a alcanzar con el sistema.
- **Funcional versus no funcional.** La gran mayoría de modelos se orientan a representar los requisitos funcionales del sistema; sin embargo, existe una necesidad creciente de disponer de modelos que capturen los requisitos no funcionales o de calidad del sistema (p. ej., rendimiento, seguridad, usabilidad).
- **Formal versus informal.** Existen modelos formales que se inspiran en enfoques de modelamiento matemático, mientras que, otros se derivan de modelos de programación o de abstracciones del dominio. Los modelos formales tienen como ventaja que pueden soportar el razonamiento automático sobre el estado del sistema; no obstante, estos modelos podrían no capturar o expresar apropiadamente los conceptos del dominio.

En la práctica es muy probable que se necesiten múltiples tipos de modelos en tiempo de ejecución para representar y capturar los diferentes aspectos o incumbencias de un sistema.

2.4 Medición del software

Una disciplina adicional dentro de la ingeniería de software en la que se apoya la presente tesis doctoral es la medición de software. Las métricas del software han demostrado ser muy eficaces en la construcción de sistemas de alta calidad, así como en la mejora continua de los procesos de desarrollo de software [105]. Particularmente, la solución metodológica propuesta que se describe en los capítulos posteriores hace uso de las métricas de software para apoyar la evaluación de los aspectos de autoconsciencia de un

sistema IoT. Por lo tanto, en esta sección se revisan los principales términos y conceptos relacionados con la medición de software que se utilizan en esta tesis doctoral.

Una *métrica de software* puede definirse como una forma de medir (*método de medición, función de cálculo, modelo de análisis*) uno o varios atributos (propiedades física o abstracta) de una entidad. Una métrica tiene una *escala* que define el conjunto de valores permitidos y dependiendo del tipo de escala (intervalo o ratio) puede tener una *unidad de medida* que representa una cantidad estandarizada, utilizada para expresar la magnitud de la métrica [106]. Además, las métricas pueden clasificarse en [106]:

- **Métrica directa.** Métrica cuya forma de medir es un *método de medición*, el cual no depende de otra métrica. Por ejemplo: puntos casos de uso programados, horas-programador diarias, costo por hora-programador, etc.
- **Métrica indirecta.** Métrica cuya forma de medir es una *función de cálculo*, la cual utiliza medidas obtenidas a partir de otras métricas directas o indirectas. Por ejemplo: horas programador totales, puntos casos de uso por hora de programador, costo total del proyecto, costo por punto caso de uso, etc.
- **Indicador.** Métrica cuya forma de medir es un *modelo de análisis*, el cual utiliza medidas obtenidas a partir de otras métricas (directas, indirectas, indicadores) junto con *criterios de decisión*. Los criterios representan umbrales, objetivos o patrones que son utilizados en los modelos de análisis para describir el nivel de confianza de un resultado o determinar la necesidad de una acción. Por ejemplo: productividad de los programadores, carestía del proyecto, etc.

Según lo mencionado con anterioridad, la formas de medir los atributos de una entidad pueden ser [106]:

- **Método de medición.** Secuencia lógica de operaciones usadas para realizar mediciones de un atributo respecto de una escala específica. Los métodos de medición pueden ser: *subjetivos*, si están basados en el juicio realizado por un ser humano; o, *objetivos*, si utilizan métodos numéricos.
- **Función de cálculo.** Algoritmo o cálculo realizado para combinar dos o más métricas directas y/o indirectas
- **Modelo de análisis.** Algoritmo o cálculo realizado para combinar una o más métricas con criterios de decisión asociados.

2.5 Conclusiones

La tesis doctoral contempla varias áreas tecnológicas, por lo que en este capítulo se ha presentado un marco conceptual integral que compila y sintetiza un conjunto de conocimientos que sustentan teóricamente y guían su desarrollo.

Por un lado, se han discutido las características y particularidades en las que operan los sistemas IoT, a fin de contextualizar el dominio de aplicación de esta tesis doctoral. A su vez, esto ha posibilitado resaltar la importancia y los beneficios de IoT para la comunidad, así como reconocer los problemas y desafíos que atañen al desarrollo de software para sistema IoT y que motivaron la ejecución de esta investigación. Asimismo, se han destacado varias directrices que deben considerarse al momento de construir sistemas IoT (p. ej., una arquitectura multicapa a nivel Edge, Fog y Cloud).

Por otro lado, se ha abordado el estudio de la autoconsciencia computacional, la Ingeniería Dirigida por Modelos (models@run.time) y la medición de software, cuyo uso integrado de los conocimientos (conceptos, principios, tipologías, enfoques, paradigmas, etc.) de estas áreas tecnológicas y su aplicación a IoT representa la principal contribución de esta tesis doctoral.

Particularmente, como se podrá apreciar en capítulos posteriores, tanto el bucle de aprendizaje y razonamiento basado en modelos para sistemas autoconscientes (LRA-M loop) [19] como el framework conceptual propuesto por Lewis et al. [20] estudiados en este capítulo, constituyen insumos valiosos que han servido de base para la construcción de la solución metodológica propuesta en esta investigación.

Finalmente, los temas tratados en este capítulo permiten al lector tener una idea clara de cada una de las tecnologías utilizadas a lo largo de este trabajo de investigación e incorporadas en la solución metodológica resultante.

Capítulo 3

Estado del Arte

Este capítulo se enfoca en comprender el estado del conocimiento tecnológico sobre el uso de la autoconsciencia computacional y la Ingeniería Dirigida por Modelos en IoT y otros dominios relacionados, como: sistemas ciber físicos, ciudad inteligente, computación ubicua, entre otros. En tal sentido, a lo largo de este capítulo se resaltan las fortalezas y limitaciones de los principales estudios en las áreas señaladas, así como las brechas y desafíos abiertos que son abordados en capítulos posteriores de este trabajo de investigación mediante la solución metodológica propuesta.

En la sección 3.1 se estudian los más relevantes modelos de referencia, metamodelos y DSLs de arquitecturas de IoT. Este estudio ha sido realizado en razón de que una de las actividades fundamentales que debe cubrir la solución metodológica es el diseño de la arquitectura de un sistema IoT, siendo esta arquitectura y sus componentes (físicos y digitales) la base para soportar los procesos de monitorización y autoconsciencia de dicho sistema.

En la sección 3.2 se analizan los principales estudios existentes sobre autoconsciencia computacional en IoT y otros dominios relacionados, con énfasis en aquellos que se apoyan en la Ingeniería Dirigida por Modelos y los modelos en tiempo de ejecución.

En la sección 3.3 se identifican varias revisiones sistemáticas sobre modelos en tiempo de ejecución con un alcance tanto general como orientado al dominio de IoT, que motivan a conocer y contrastar sobre el empleo de estos modelos para otros fines o propósitos distintos a la autoconsciencia computacional.

Finalmente, la sección 3.4 expone las conclusiones de este capítulo.

3.1 Arquitecturas de IoT

La arquitectura de un sistema define el conjunto de estructuras necesarias para razonar sobre el sistema, lo cual incluye los componentes que lo conforman, las relaciones entre estos y las propiedades de ambos [107]. La arquitectura es un puente entre los requerimientos de los stakeholders y un sistema resultante final (concreto). Si bien el camino entre los requerimientos y el sistema concreto puede ser complejo, el diseño, la documentación e implementación de arquitecturas puede reducir esta complejidad y mejorar la eficiencia y efectividad en el desarrollo del sistema [107]. Por lo tanto, una sólida arquitectura posibilita la planificación, diseño e implementación adecuada de los requisitos funcionales y no funcionales (rendimiento, flexibilidad, escalabilidad, integración, mantenibilidad, etc.) de un sistema; es decir, tiene un impacto positivo en la calidad de los servicios que proporciona un sistema.

Con base en lo mencionado, en las siguientes subsecciones se analizan las principales propuestas de modelos de arquitectura de referencia para IoT, así como los más destacados metamodelos y DSLs orientados a arquitecturas de IoT. Además, se incluye una discusión sobre las limitaciones y desafíos de investigación identificados en torno a este tipo de DSLs.

3.1.1 Modelos de arquitectura de referencia para IoT

Varios esfuerzos se han enfocado en construir arquitecturas de referencia para sistemas IoT. Primeramente, la ISO/IEC 30141:2018 [2] proporciona una Arquitectura de Referencia de IoT estandarizada (IoT RA), que incluye un modelo conceptual genérico que describe los conceptos de las entidades comunes de un sistema IoT y las relaciones entre estas. Del modelo conceptual se deriva un modelo de referencia de alto nivel, dividido en cinco vistas arquitectónicas: i) *funcional*, describe las funciones de un sistema IoT, desde una perspectiva independiente de la tecnología; ii) *sistema*, identifica los componentes genéricos (dispositivos, subsistemas y redes) que conforman un sistema IoT; iii) *información*, define a nivel conceptual la estructura (p. ej., relaciones, atributos, servicios) de la información sobre las entidades IoT; iv) *comunicaciones*, describe las principales redes de comunicaciones que intervienen en los sistemas IoT y las entidades que se conectan a través de ellas; y, v) *uso*, se centra en explicar cómo se debe desarrollar, probar, operar y utilizar un sistema IoT, desde la perspectiva del usuario.

A su vez, Bauer et al. [108] proponen un Modelo de Referencia Arquitectónico de IoT (IoT ARM) basado en cinco submodelos: i) *dominio* [109], incluye los principales conceptos de IoT (p. ej., entidades físicas, entidades virtuales, dispositivos, recursos, servicios) y las relaciones entre estos; ii) *información*, define la estructura de la información relacionada con IoT; iii) *funcional*, identifica grupos de funcionalidades de un sistema IoT; iv) *comunicación*, introduce conceptos para manejar la complejidad de las comunicaciones en entornos heterogéneos de IoT; y, v) *confianza, seguridad y privacidad*, describe las funcionalidades relevantes de estos aspectos y sus interdependencias e interacciones. Del mismo modo, Patel et al. [110][111] presentan una metodología que separa el desarrollo de una aplicación IoT en cuatro áreas: dominio, funcional, implementación y plataforma, que se representan en un modelo conceptual.

3.1.2 Metamodelos y DSLs para diseñar arquitecturas de IoT

Con base en las revisiones de la literatura sobre DSLs para IoT realizadas por Salman et al. [38] y Erazo-Garzón et al. [56], a continuación, se describen y analizan las principales propuestas enfocadas en el diseño de arquitecturas de IoT.

- 1) Eterovic et al. [112] se centran en crear un lenguaje de desarrollo de sistemas IoT lo suficientemente potente para los profesionales y comprensible para los usuarios finales que proporcionan los requisitos. Por lo tanto, proponen un Lenguaje de Modelado Específico de Dominio Visual (VDSML) para IoT utilizando un perfil UML basado en estereotipos. El VDSML permite modelar objetos reales y virtuales, los cuales pueden contener una colección de elementos virtuales, tales como: entradas (sensores), salidas (actuadores) y componentes de software. Una limitación es la poca especialización del lenguaje, haciendo falta estereotipos específicos para representar nodos Edge, Fog o Cloud; y, por otra parte, componentes de software, tales como: middleware, servicios, aplicaciones o bases de datos. Adicionalmente, este trabajo utilizó una prueba de usabilidad para validar el VDSML, por medio de la puntuación SUS (System Usability Scale), la tasa de éxito de la tarea, el tiempo dedicado a la tarea y la tasa de error del usuario.
- 2) Ciccozzi y Spalazzese [113] presentan MDE4IoT un enfoque que utiliza un conjunto de lenguajes basados en UML (fUML, ALF, Marte, OCL) para soportar el modelado, la generación de ejecutables en tiempo de diseño y la autoadaptación de configuraciones emergentes en tiempo de ejecución de sistemas IoT. MDE4IoT posibilita el modelado de las funcionalidades de software (componentes de software),

así como de los dispositivos o plataformas físicas (componentes de hardware) en los que se ejecutan dichas funcionalidades. Sin embargo, al igual que el enfoque anterior, requiere de una mayor especialización del lenguaje para representar las entidades propias de IoT. La utilidad MDE4IoT fue demostrada mediante la implementación de un estudio de caso de alumbrado público inteligente.

- 3) Salihbegovic et al. [114] proponen DSL-4-IoT para abordar la complejidad y heterogeneidad de los sistemas IoT. Este es un VDSML para diseñar la estructura de un sistema IoT utilizando bloques jerárquicos (sistema, subsistema, dispositivos, canales de dispositivos) enfocados principalmente en la capa Edge. Los bloques se pueden guardar en bibliotecas para fomentar la reutilización y escalabilidad. Los archivos de configuración generados en DSL-4-IoT se procesan en el motor de tiempo de ejecución OpenHAB. A su vez, para demostrar la viabilidad y usabilidad de la solución, se implementó un banco de pruebas de IoT que abarca una variedad de sensores estáticos y móviles en dos dominios de aplicación (hogar inteligente y monitoreo remoto de pacientes).
- 4) ThingML [115] incluye una metodología, un lenguaje de modelado y un conjunto de herramientas para soportar la generación de código multiplataforma para sistemas reactivos distribuidos. El lenguaje ThingML comprende dos estructuras: i) cosas que representan componentes de software; y ii) configuraciones que describen su interconexión. Los componentes de software pueden incluir propiedades, funciones, mensajes, puertos y máquinas de estado. Sin embargo, ThingML no permite modelar los nodos en los que se ejecutan los componentes del software a nivel de las capas Edge, Fog y Cloud. Con respecto a la evaluación de ThingML se han realizado varios estudios de caso, combinando diversas plataformas de implementación.
- 5) Pramudianto et al. [116] presentan una arquitectura para desarrollar prototipos de IoT, separando el modelado de dominio de las implementaciones tecnológicas. Mediante una herramienta basada en modelos, llamada IoTLink, los expertos pueden crear modelos de dominio, componiendo objetos virtuales vinculados a tecnologías de implementación (sensores, actuadores) para abstraer sus complejidades y especificidades. Además, IoTLink genera artefactos en Java a partir del modelo definido. Al igual que ThingML, una desventaja de IoTLink es que no contempla el modelado de los nodos informáticos en los que residen y se ejecutan los artefactos de Java. Finalmente, mediante un experimento controlado, IoTLink se evaluó y comparó con un proyecto de desarrollo clásico de Java en términos de tiempo de desarrollo y satisfacción del usuario.
- 6) Negash et al. [117] introducen un enfoque flexible y escalable que mejora la programabilidad y modificabilidad en la capa de percepción de un sistema IoT, especialmente en dispositivos con recursos limitados. El enfoque utiliza un lenguaje específico de dominio (DoS-IL) con notación textual para escribir scripts livianos y almacenarlos en una puerta de enlace. A su vez, los scripts son solicitados y ejecutados por los dispositivos de la capa de percepción mediante un navegador de recursos embebidos que utiliza un intérprete DoS-IL y manipula un Modelo de Objetos de Dispositivo (DOM). No obstante, esta propuesta no presenta una evaluación empírica.
- 7) Costa et al. [118] proponen un DSL (SoaML4IoT) para diseñar sistemas IoT basados en SOA. El DSL amplía el Lenguaje de Modelado de Arquitectura Orientada a Servicios (SoaML) con conceptos específicos del dominio de IoT. Los nodos Edge se

representan de forma general, requiriendo una mayor especialización en sensores, actuadores, etiquetas y puertas de enlace de IoT (controladores). En lugar de una evaluación empírica, se realizó un análisis comparativo con otras propuestas de DSLs para determinar el nivel de expresividad de SoaML4IoT.

- 8) Alulema et al. [119] presentan un enfoque dirigido por modelos que incluye un DSL, un editor gráfico y una transformación de modelo a texto (M2T) para generar el código para los nodos IoT, los cuales contienen controladores, sensores y actuadores. En consecuencia, esta solución solo apunta a la capa perimetral de una plataforma de IoT. A su vez, los autores diseñaron un escenario de hogar inteligente para demostrar la funcionalidad y viabilidad del enfoque propuesto.
- 9) González et al. [120] proponen un DSL gráfico (MOCSL) dirigido a personas sin conocimientos de programación, a fin de facilitar la creación de aplicaciones nativas para objetos inteligentes interconectados a través de la plataforma IoT Midgar. Por medio de MOCSL, los usuarios pueden crear la lógica necesaria para sus objetos, únicamente seleccionando los sensores y actuadores que desean usar en su teléfono inteligente o Arduino. Por lo tanto, MOCSL también se centra únicamente en la capa perimetral de una plataforma de IoT. Además, para validar la utilidad y eficiencia de la solución, los autores llevaron a cabo un experimento en el que dos perfiles de participantes, con diferentes niveles de conocimiento sobre objetos inteligentes, debían crear una aplicación Arduino básica utilizando MOCSL y otro editor gráfico.
- 10) Sharaf et al. [121] presentan un framework de generación de código basado en modelos, denominado CAPSml. Este framework utiliza modelos CAPS-SALM para representar la vista estructural y de comportamiento de la arquitectura de un sistema IoT. Los modelos CAPS-SALM están conformados por componentes y conexiones entre estos, mediante puertos de comunicación. Cada componente puede tener varios modos que denotan su estado y comportamiento por medio de un conjunto de eventos, acciones y condiciones. El framework CAPSml transforma el modelo CAPS-SALM en un modelo ThingML. A su vez, se reutiliza el framework ThingML para la generación de código multiplataforma utilizando diversos compiladores. Sin embargo, al igual que varios trabajos analizados previamente, la solución CAPSml no permite establecer en qué nodos de computación y capas de IoT se ejecutan los componentes. Los autores validaron la utilidad del framework mediante un estudio de caso de riego inteligente en el subdominio agrícola.
- 11) Kirchhof et al. [122] definen MontiThings, un lenguaje de modelado para arquitecturas de aplicaciones IoT basado en componentes organizados jerárquicamente que intercambian datos entre ellos, a través de conexiones entre sus interfaces o puertos dirigidos y tipificados. Los componentes compuestos definen su comportamiento mediante la composición e interrelación de sus subcomponentes, mientras que, los componentes atómicos describen su comportamiento con código escrito a mano, separado de la lógica de manejo de errores. Además, MontiThings dispone de herramientas para transformar los componentes a código en C++ y distribuirlos mediante contenedores ejecutables, adaptados a las características de los dispositivos IoT destino. Una limitante de MontiThings consiste en que cualquier entidad IoT (física o digital) se define como un componente genérico, restándole expresividad y semántica al lenguaje propuesto. Los autores se enfocaron en evaluar la escalabilidad de la solución mediante un estudio de caso y experimentos en el subdominio de la domótica.

- 12) Barriga et al. [123] han creado un DSL (SimulateIoT) para diseñar, codificar e implementar simulaciones de sistemas IoT. La solución comprende un metamodelo de dominio, una sintaxis gráfica concreta y algoritmos de transformación de modelo a texto. El modelo de simulación de IoT creado en el editor gráfico puede incluir sensores, actuadores, nodos Fog y Cloud, bases de datos y motores de procesamiento de eventos complejos. Sin embargo, estos elementos sólo pueden conectarse mediante protocolos de comunicación asíncronos (publicación-suscripción). La solución se implementó en dos estudios de caso de IoT (edificio inteligente y agricultura) para demostrar la expresividad de la solución por medio de la opinión e impresión de los autores.
- 13) Alfonso et al. [124][125] proponen un DSL para modelar los aspectos estáticos y dinámicos de una implementación de IoT. El DSL incluye un editor de proyección MPS con notación textual, tabular y de vista de árbol. Además, la solución comprende un prototipo de generador de manifiestos de Kubernetes para implementar los sistemas IoT modelados. En su primera publicación [124], este trabajo proporciona una prueba de concepto del prototipo de generador de código; mientras que, en su segunda publicación [125] se utilizan experimentos para validar la expresividad y facilidad de uso del DSL, con el apoyo de cuestionarios y estadística descriptiva básica (tabla de frecuencias).

3.1.3 Discusión

Existen importantes avances en el campo de los DSLs para diseñar arquitecturas de IoT, cuyo común denominador ha sido abstraer la complejidad y heterogeneidad de los sistemas IoT, de tal forma que se acelere su desarrollo; sin embargo, se han identificado varias limitaciones en el trabajo relacionado. En lo que resta de este apartado, se analizan estas limitaciones y los correspondientes desafíos de investigación, mismos que son abordados en esta tesis doctoral a través de la solución metodológica propuesta y, particularmente el DSL Monitor-IoT [56] que forma parte de esta solución. Asimismo, las Tablas 3-1 y 3-2 presentan un análisis comparativo entre los DSLs estudiados y el DSL Monitor-IoT.

- Aunque algunas propuestas de DSLs [116][117] se basan en el Modelo de Referencia Arquitectónica de IoT (IoT ARM) [108], no existe un metamodelo aceptado globalmente. Incluso se han observado ambigüedades entre las soluciones estudiadas respecto al significado y uso de diversos términos y conceptos de IoT, siendo la falta de consenso un problema abierto. Además, los DSLs propuestos no se basan en un metamodelo alineado con la arquitectura de referencia estandarizada para IoT ISO/IEC 30141:2018 (ver Tabla 2).
- Las propuestas estudiadas se basan en metamodelos con un lenguaje limitado para representar los conceptos claves que pueden ser necesarios para monitorear un sistema IoT, como: entidades físicas, entidades digitales (nodos de computación y sus recursos), redes y protocolos de comunicación, propiedades de entidades y flujos de datos entre entidades. En la Tabla 1 se observa en detalle que a diferencia de Monitor-IoT, los DSLs analizados no tienen la capacidad de especificar o modelar todos los conceptos claves mencionados.
- Con relación a las entidades físicas, tres propuestas [112][116][118] incluyen la posibilidad de representar cualquier entidad física en sus metamodelos. Por su parte, la propuesta de Alfonso et al. [125] permite representar únicamente los lugares (regiones) a monitorear en un escenario de IoT.

- La mayoría de los DSLs estudiados se centran en la capa Edge de una infraestructura IoT, permitiendo la especificación de sensores, actuadores, etiquetas; y, en menor medida, puertas de enlace de IoT ubicadas en la red de proximidad. Por el contrario, sólo existen pocos estudios [118][123][124] que soportan el modelado de nodos Fog y Cloud en sus metamodelos.
- Sobre los recursos de software y hardware que utilizan los nodos Edge, Fog y Cloud, algunas propuestas los representan mediante conceptos genéricos en sus metamodelos. Sin embargo, esto limita la posibilidad de especificar detalladamente los atributos y configuraciones de los recursos esenciales para monitorear y controlar los escenarios de IoT (APIs, aplicaciones, servicios, bases de datos, middlewares, brokers, interfaces de red), requiriéndose una especialización de estos recursos en los metamodelos. A su vez, aunque otras propuestas se centran en la especialización, no representan todos los recursos esenciales mencionados. Por lo tanto, se requieren metamodelos que combinen estos dos enfoques, incluyendo una relación de especialización/generalización entre subclases para definir en detalle los recursos esenciales de un sistema IoT y una superclase concreta para representar cualquier otro recurso de hardware o software (p. ej., CPU, RAM, batería).
- En particular, las soluciones estudiadas y sus metamodelos tienen limitaciones a la hora de representar el software o los recursos de datos utilizados por un escenario típico de monitoreo de IoT. Así, sólo dos propuestas [116][123] consideran el concepto de base de datos en sus metamodelos; sin embargo, estas propuestas no permiten modelar la estructura lógica de almacenamiento de la base de datos (tablas y columnas). A su vez, sólo un estudio [116] permite especificar operaciones de agregación y fusión sobre los datos monitoreados. Además, varios metamodelos no permiten especificar los nodos informáticos en los que residen o se ejecutan los recursos. Por ejemplo, aunque dos propuestas [123][124] contemplan el uso de tópicos de broker para soportar la comunicación asíncrona, no incorporan el concepto de broker para representar el recurso en el que se gestionan estos tópicos. Otra limitación es que las soluciones suponen que un sensor sólo puede recopilar datos para una sola propiedad de una entidad IoT, a diferencia de la realidad, donde hay sensores que pueden recopilar datos para múltiples propiedades (p. ej., el sensor DHT11 recopila temperatura y humedad) [123].
- Un grupo de DSLs [115]–[118] incluyen explícitamente el concepto de propiedad en sus metamodelos para caracterizar ciertos tipos de entidades IoT. Sin embargo, es necesario profundizar en este aspecto para proponer metamodelos con una estructura extensible, capaz de soportar la especificación de varios tipos de propiedades (telemetría, estado y metadatos) para cualquier entidad IoT, ya sea física o digital (nodos de computación y recursos), dependiendo de las necesidades de cada escenario de IoT. En este sentido, las propiedades de telemetría tienen información sobre una entidad física, recopilada a partir de sensores (p. ej., temperatura de una habitación, presión arterial de un paciente); las propiedades de estado contienen información que describe el estado actual de una entidad digital (p. ej., nivel de energía de la batería de un sensor, cantidad de memoria o procesamiento utilizado por una puerta de enlace de IoT, ancho de banda utilizado para transmitir los datos monitoreados); y, metadatos, que contienen información que rara vez cambia y que se conoce en tiempo de diseño (p. ej., número de serie, marca, modelo).
- Dado que un sistema IoT puede tener muchas entidades del mismo tipo, los metamodelos también deben admitir estructuras genéricas (p. ej., plantillas de

propiedades por categoría de entidad) para permitir que las propiedades comunes entre entidades se especifiquen por una sola vez. Sin embargo, este es un aspecto que no es considerado en el trabajo relacionado. Otro aspecto no cubierto es el mapeo de las propiedades a monitorear tanto con las APIs encargadas de recolectar los datos como con las estructuras de almacenamiento (base de datos, tablas, columnas) de dichos datos.

- Las propuestas [112][116][119][121]–[123] que soportan el modelado de flujos de datos no cubren la diversidad de posibilidades de enlaces de comunicación e intercambio de datos entre las entidades digitales (APIs, aplicaciones, servicios, bases de datos, middlewares, brokers) que los escenarios de IoT pueden requerir a través de las capas Edge, Fog y Cloud. Dependiendo de las capacidades de hardware de los dispositivos IoT, el ancho de banda de la red y el volumen de datos, es posible que se necesiten diferentes tipos de entidades digitales y configuraciones de comunicación entre estas para soportar la motorización de un entorno de IoT. Desde comunicaciones directas entre los dispositivos IoT (sensores) y los nodos Cloud hasta incluir nodos intermedios ubicados en el borde (puerta de enlace de IoT) o las capas de niebla, responsables de realizar las operaciones de enrutamiento, procesamiento (fusión, agregación) y almacenamiento de datos. Asimismo, pueden ser necesarias comunicaciones síncronas o asíncronas entre entidades digitales. Por lo tanto, uno de los desafíos es proponer DSLs lo suficientemente flexibles y extensibles para permitir el modelado gráfico (con un alto grado de abstracción) de flujos de datos, como una secuencia de diversos tipos de enlaces de comunicación entre las entidades digitales. A su vez, debe ser posible que los tipos de enlaces entre las entidades digitales puedan combinarse y ordenarse de diferentes maneras para formar flujos de datos según las necesidades de cada escenario de IoT.
- Por otro lado, apenas tres propuestas [115][116][121]–[123] (ver Tabla 2) han incluido como parte de su solución, la definición de un proceso metodológico que oriente las tareas de especificación, modelado e implementación de soluciones IoT con el apoyo de la herramienta DSL propuesta. En este sentido, resulta deseable disponer de un proceso metodológico asociado a una herramienta DSL para asegurar que los usuarios finales puedan utilizarla de manera disciplinada, eficiente y efectiva.
- Aunque la mayoría de los DSLs estudiados poseen un lenguaje de modelado visual (sintaxis gráfica concreta), pocas propuestas [112][116][120] han llevado a cabo una evaluación empírica rigurosa basada en la percepción del usuario para validar la facilidad de uso y utilidad de la herramienta DSL (ver Tabla 2). Sin embargo, estas propuestas citadas se centran únicamente en la capa Edge o disponen de un lenguaje de modelado limitado. Por el contrario, las propuestas que se enfocan en las capas Edge, Fog y Cloud [118][123][125] no incluyen una evaluación empírica o, en su lugar, validan sus soluciones a partir de la opinión e impresión de sus autores o empleando estadística básica. Esto sugiere la necesidad de utilizar técnicas más rigurosas (estadística inferencial - pruebas de hipótesis) para determinar la percepción del usuario final sobre la usabilidad y utilidad de la solución.

Tras la publicación del DSL Monitor-IoT en la revista IEEE Access, recientemente han surgido dos propuestas interesantes de lenguajes de modelado para representar sistemas IoT, denominadas ViLanIoT [126] y CHESSIOT [127]. La primera propuesta se ha preocupado de la sintaxis concreta del lenguaje, con miras a lograr una estandarización a nivel visual de un DSL para IoT, similar a lo que ocurre en otros dominios [126].

Tabla 3-1. Análisis comparativo entre los DSLs estudiados y Monitor-IoT [56].
(Parte 1: Capacidad de especificación y modelado de entidades, recursos, propiedades y flujos de datos)

Trabajo relacionado	Entidad física	Nodo Edge		Fog Node	Cloud Node	Red y protocolo de comunicación (síncrona, asíncrona)	Recurso		Propiedad de entidad	Categoría de entidad y plantilla de propiedad para promover la reutilización	Flujo de datos	Estructura Base de Datos (tablas, columnas)
		Dispositivo IoT (Sensor, Actuador, Tag)	Gateway IoT				Hardware (CPU, RAM, Interfaz de red, Batería, etc.)	Software				
Eterovic et al. [112]	Si	Si	No	No	No	No	No	Componente de software.	No	No	Si	No
MDE4IoT [113]	No	Si	No	No	No	No	Componente de hardware	Componente de software.	No	No	No	No
DSL-4-IoT [114]	No	Si	Si	No	Si	Si	No	Servicio.	No	No	No	No
ThingML [115]	No	No	No	No	No	Si	No	Componente de software, API.	Si	No	No	No
IoTLink [116]	Si	Si	No	No	No	Si	No	Objeto de software (instancia de clase), API, servicio, base de datos.	Si	No	Si	No
DoS-IL [117]	No	Si	No	No	No	Si	Recursos disponibles en el dispositivo restringido (DOM).	Recursos disponibles en el dispositivo restringido (DOM).	Si	No	No	No
SoaML4IoT [118]	Si	Si	Si	Si	Si	Parcial (síncrono)	No	Aplicación, componente de software, servicio.	Si	No	No	No
Alulema et al. [119]	No	Si	Si	No	No	Parcial (síncrono)	No	Servicio.	No	No	Si	No
MOCSL [120]	No	Si	Si	No	No	Parcial (síncrono)	No	Aplicación.	No	No	No	No
CAPSml [121]	No	No	No	No	No	Si	No	Componente de software, evento, servicio.	Si	No	Si	No
MontiThings [122]	No	No	No	No	No	Si	No	Componente de software.	Si	No	Si	No
SimulateIoT [123]	No	Si	No	Si	Si	Parcial (asíncrono)	No	Broker, Tópico de broker, Base de datos, motor CEP (Complex Event Processing), motor ESP (Event Stream Processing).	No	No	Si	No
Alfonso et al. [125]	Parcial (Región)	Si	Si	Si	Si	Parcial (asíncrono)	Representación de recursos no extensible. Los recursos (CPU, RAM, storage) se definen como atributos de las metaclasses <i>Node</i> , <i>Container</i> , y <i>Application</i> .	Contenedor, Aplicación, Tópico de broker.	No	No	No	No
Monitor-IoT [56]	Si	Si	Si	Si	Si	Si	Interfaz de red a través de la metaclass específica <i>NetworkInterface</i> y cualquier otro recurso de hardware a través de la metaclass genérica <i>Resource</i> .	API, aplicación, middleware, broker, tópico de broker, servicio y base de datos mediante metaclasses específicas y cualquier otro recurso a través de la metaclass genérica <i>Resource</i> .	Si (cualquier propiedad de entidades físicas o digitales a través de la metaclass <i>Property</i>)	Si	Si (cualquier combinación de flujos de datos entre entidades digitales a nivel de las capas Edge, Fog y Cloud)	Si

Tabla 3-2. Análisis comparativo entre los DSLs estudiados y Monitor-IoT [56].
(Parte 2: Estandarización, lenguaje de modelado visual, transformaciones, metodología y evaluación empírica)

Trabajo relacionado	Alineación con ISO/IEC 30141:2018	Lenguaje de Modelado Visual (Editor Gráfico)	Proceso metodológico que orienta el uso del DSL	Herramientas de transformación o generación de código	Evaluación empírica	
					Tipo	Propósito
Eterovic et al. [112]	No	Si	No	No	Experimento (Prueba de usabilidad)	Probar que el lenguaje es adecuado para usuarios técnicos y no técnicos en términos de simplicidad.
MDE4IoT [113]	No	Si	No	Si	Estudio de caso (Basado únicamente en la opinión e impresión de los autores. No se utilizan técnicas estadísticas)	Demostrar la utilidad de la solución.
DSL-4-IoT [114]	No	Si	No	Si	Prueba de concepto	Demostrar la viabilidad y usabilidad de la solución.
ThingML [115]	No	No	Si	Si	Estudio de caso	Demostrar la funcionalidad y factibilidad de la solución.
IoTLink [116]	No	Si	Si	Si	Experimento (Encuesta, estadística descriptiva y prueba de hipótesis)	Evaluar la efectividad, eficiencia y usabilidad de la herramienta.
DoS-IL [117]	No	No	No	Si	Ninguna	
SoaML4IoT [118]	No	Si	No	Si	Ninguna	
Alulema et al. [119]	No	Si	No	Si	Estudio de caso (Basado únicamente en la opinión e impresión de los autores. No se utilizan técnicas estadísticas)	Demostrar la funcionalidad y factibilidad de la solución.
MOCSL [120]	No	Si	No	Si	Experimento (Cuestionario, estadística descriptiva, diagrama de caja y prueba de hipótesis)	Validar la utilidad del DSL para usuarios sin conocimientos de programación y el nivel de eficiencia con respecto a otras alternativas.
CAPSml [121]	No	Si	Si	Si	Estudio de caso (Basado únicamente en la opinión e impresión de los autores. No se utilizan técnicas estadísticas)	Evaluar la utilidad de la solución.
MontiThings [122]	No	Si	Si (general)	Si	Estudio de caso, Experimento	Evaluar la escalabilidad de la solución.
SimulateIoT [123]	No	Si	Si (general)	Si	Estudio de caso (Basado únicamente en la opinión e impresión de los autores. No se utilizan técnicas estadísticas)	Demostrar la expresividad del DSL.
Alfonso et al. [125]	No	No	No	Si	Experimento (Cuestionario, estadística descriptiva básica)	Demostrar la expresividad y facilidad de uso del DSL.
Monitor-IoT [56]	Si	Si	Si (detallado)	Si	Cuasi experimento (Modelo de Evaluación de Métodos (MEM) [49], encuesta, estadística descriptiva, diagrama de caja, prueba de hipótesis y método de regresión lineal simple)	Determinar el rendimiento (efectividad y eficiencia) del usuario en el uso del DSL Monitor-IoT y su proceso metodológico. Comprender la percepción del usuario sobre la facilidad de uso, utilidad e intención de uso futuro del DSL Monitor-IoT y su proceso metodológico.

Mientras que, la segunda propuesta contempla un entorno de MDE que integra tres DSLs para modelar especificaciones de sistema (a nivel físico), software (a nivel funcional) y despliegue de soluciones IoT multicapa (Edge, Fog y Cloud) [127]. Estas propuestas, en su trabajo relacionado, destacan la contribución del DSL Monitor-IoT, en cuanto a la completitud de su lenguaje para representar los conceptos claves de un sistema IoT multicapa. De igual manera, Kohan et al. [128] en su reciente revisión sistemática sobre enfoques centrados en modelos para sistemas IoT, resaltan el metamodelo y lenguaje bien definido de Monitor-IoT, así como el enfoque empleado para su evaluación.

3.2 Autoconsciencia computacional en IoT y otros dominios relacionados

En la actualidad, tres libros recopilan las ideas y resultados de la investigación sobre la construcción de sistemas informáticos autoconscientes [13][40][41]; mientras que, varios estudios permiten trazar la progresión de esta área de investigación durante la última década y, particularmente determinar el estado del conocimiento tecnológico sobre la autoconsciencia computacional en IoT y otros dominios relacionados. Entre los más representativos, un primer estudio es la revisión sistemática de Elhabbash et al. [23] sobre la adopción de la autoconsciencia en la ingeniería de software, misma que aborda las siguientes interrogantes: i) cómo los trabajos han definido y caracterizado la autoconsciencia, ii) cuáles son los motivos y fuentes de inspiración para el uso de la autoconsciencia en la ingeniería de software; ii) qué prácticas y paradigmas de ingeniería de software han empleado la autoconsciencia; iii) qué enfoques de ingeniería se han utilizado para diseñar y codificar las propiedades de autoconsciencia en los sistemas de software; iv) cómo se han evaluado las soluciones de autoconsciencia propuestas; y, v) qué aplicaciones del mundo real han adoptado la autoconsciencia computacional. Un segundo estudio es el trabajo de Esterle et al. [129] que resume varias contribuciones en el ámbito de los sistemas ciber físicos autoconscientes, con el objeto de crear una imagen sobre el avance alcanzado en este campo. Un tercer estudio es la revisión de S-Julián et al. [42] que analiza el estado de la investigación sobre la incorporación de las capacidades self-* a nivel de la computación en el borde (Edge Computing) de un sistema inteligente. Particularmente, en esta revisión se abordan casos de prácticos relevantes sobre el uso de la autoconsciencia a nivel de los nodos Edge. Un último estudio consiste la encuesta de Jantsch et al. [43] que brinda un relato histórico y una discusión sobre los principales enfoques de autoconsciencia en sistemas en chip (SoC).

Como complemento a los trabajos señalados, se realizó una revisión bibliográfica en las principales bibliotecas digitales relacionadas con el área de la computación (IEEE Xplore, ACM Digital Library, SpringerLink y ScienceDirect). Además, se aplicó la estrategia de búsqueda de bola de nieve con el apoyo de la herramienta de mapeo de literatura - Litmaps [130], a fin de identificar otras publicaciones de interés, a partir de las referencias incluidas en los estudios previamente seleccionados. Cabe recalcar que la búsqueda y selección de los estudios primarios se centró únicamente en aquellos que abordan la autoconsciencia computacional de forma explícita. Estas actividades complementarias permitieron alcanzar una visión actualizada del estado de la cuestión e identificar estudios adicionales relevantes con la temática.

En las siguientes subsecciones se describen los principales estudios primarios sobre autoconsciencia dentro del dominio de IoT y otros relacionados. La presentación de los estudios se ha organizado en trabajos teóricos, soluciones prácticas basadas tanto en enfoques genéricos como paradigmas de MDE; y, finalmente, soluciones (ad hoc) orientadas a aplicaciones particulares de IoT. Además, se incluye una discusión sobre las

limitaciones encontradas en el trabajo relacionado, así como los desafíos aún abiertos en esta área de investigación.

3.2.1 Estudios teóricos

En la literatura se encuentran varios estudios conceptuales centrados fundamentalmente en exponer los beneficios, desafíos y recomendaciones (hojas de ruta) sobre la incorporación de la autoconsciencia en sistemas IoT, ciber físicos, ciudad inteligente y otros dominios similares.

En primer lugar, Bellman et al. [24] con base en los desafíos que representa la intersección entre los sistemas ciber físicos y la autoconsciencia computacional proponen una hoja de ruta para futuras investigaciones que integren estas tecnologías. Los autores aseveran que, a pesar de la enorme variedad (tamaño, recursos, nivel de distribución, accesibilidad) de sistemas ciber físicos, estos poseen principios y desafíos comunes. Por lo tanto, en lugar de diseñar sistemas específicos o a medida que sean autoconscientes, se requiere la creación de enfoques y métodos más generales que puedan ser aplicables a un amplio espectro de sistemas ciber físicos autoconscientes. Con este fin, los autores sostienen que es necesario ampliar los esfuerzos de investigación en los siguientes aspectos:

- i) *Desarrollar nuevos procesos de diseño e ingeniería.* Al trasladar las decisiones del tiempo de diseño al tiempo de ejecución, se debe cambiar la naturaleza de las preguntas de diseño hacia aquellas relevantes sobre cuándo, dónde y cómo incorporar las diferentes formas de autoconsciencia. Si bien algunos frameworks, arquitecturas de referencia y middlewares están disponibles para los sistemas autoconscientes, es necesario adaptarlos a los sistemas ciber físicos.
- ii) *Crear una infraestructura para los procesos de autoconsciencia.* Se requiere de plataformas, herramientas y componentes genéricos que faciliten y agilicen el desarrollo de sistemas ciber físicos de manera sistemática, reutilizable y confiable. En particular, es necesario crear nuevas herramientas de modelado o ampliar las existentes para que soporten la nueva dimensión de autoconsciencia. En síntesis, las nuevas metodologías y su infraestructura de soporte deben basarse en un conjunto de procesos y herramientas que se integren adecuadamente.
- iii) *Desarrollar una autoconsciencia sensible a los recursos.* La autoconsciencia no es gratuita; sin embargo, las funciones adicionales de aprendizaje y razonamiento pueden ahorrar más recursos generales de los que se necesitan para implementarlas. Por lo tanto, se deben explorar enfoques que permitan a un sistema tomar decisiones sobre cómo asignar de manera flexible los recursos finitos a diferentes procesos en tiempo de ejecución, incluidos los procesos de autoconsciencia, en busca de cumplir eficientemente con sus requisitos.
- iv) *Reconocer la situacionalidad y subjetividad de la autoconsciencia.* El conocimiento de un sistema es subjetivo, depende de su propia perspectiva de percepción, la cual está condicionada a su situación (p. ej., recursos disponibles de detección y procesamiento) que puede variar a lo largo del tiempo. De ahí que, también se requieren de técnicas que permitan a los sistemas reflexionar sobre su propia perspectiva para fundamentar adecuadamente su conocimiento y tomar decisiones acertadas, incluso que conlleven a cambiar su perspectiva.
- v) *Verificar la autoconsciencia y establecer garantías.* En un sistema autoconsciente, el comportamiento se ve afectado por sus propias capacidades de autoconsciencia. Para dar garantías en tiempo de ejecución sobre el comportamiento de un sistema, se

necesitan de enfoques de meta-autoconsciencia que le permitan al sistema determinar si resulta conveniente mantener, adaptar o cambiar un cierto nivel de autoconsciencia, a fin de asegurar el cumplimiento de los requisitos.

A su vez, Tammemäe et al. [25] describen varios ejemplos con el objeto de demostrar que la adopción de la autoconsciencia a nivel de los nodos Edge y Fog de una infraestructura de IoT resulta beneficioso para mejorar el rendimiento de los recursos durante el procesamiento de los datos, así como la confiabilidad y privacidad de los datos. Específicamente, los autores señalan que los nodos autoconscientes tienen una comprensión integral de su situación y del entorno, lo que conlleva a mejores decisiones sobre qué datos recopilar, cómo y dónde procesarlos, qué datos comunicar; y, finalmente, qué acciones ejecutar. Además, aportan recomendaciones sobre qué recursos se requieren y cómo deben ser organizados a nivel arquitectónico para distribuir adecuadamente la computación; y, particularmente la autoconsciencia entre los niveles jerárquicos (Edge, Fog, Cloud) de un sistema IoT. Al igual que el estudio anterior, se pone de manifiesto la necesidad de un método con herramientas de apoyo que guíe a los ingenieros en el diseño de aplicaciones IoT capaces de equilibrar la distribución de los requisitos de computación y autoconsciencia a través de las capas Edge, Fog y Cloud.

En otro estudio, Gurgen et al. [26] describen una arquitectura conceptual para sistemas ciber físicos autoconscientes, basada en el bucle de control MAPE-K [131] y un middleware orientado a servicios distribuidos. A partir de ello, identifican importantes desafíos de investigación y recomendaciones para la implementación de sistemas ciber físicos autoconscientes en ciudades inteligentes. En este sentido, resaltan que los aspectos de autoconsciencia deben considerarse desde la concepción hasta el despliegue y operación de un sistema ciber físico, sugiriendo utilizar enfoques basados en modelos, de tal forma que estos artefactos sean el vínculo para gestionar la autoconsciencia a lo largo de todas las etapas del ciclo de vida del sistema. Además, manifiestan que la autoconsciencia debe poder incorporarse en cualquier componente de software de un sistema ciber físico, sobre todo procurando asegurar que los dispositivos en el borde posean una inteligencia incorporada para razonar y actuar localmente.

Por otro lado, Esterle y Brown [27] amplían la idea de la autoconsciencia de sistemas individuales hacia una autoconsciencia en red con el propósito de aumentar la capacidad de los sistemas ciber físicos para aprender sobre su entorno y cómo se ven afectados por las interacciones con otros sistemas. Para ello, los autores proponen diferentes niveles (estimulo, interacción, tiempo, objetivos, meta-autoconsciencia) para categorizar la consciencia que un sistema puede desarrollar con el tiempo sobre su entorno de red, tal como lo hacen los humanos. Además, plantean una hoja de ruta en la que destacan a los *models@run.time* como un campo de investigación fundamental hacia la autoconsciencia en red para sistemas ciber físicos.

Desde un ámbito general de la autoconsciencia computacional, Lewis et al. [20] proponen un framework conceptual que define los niveles (pre-reflexiva, reflexiva, meta-reflexiva), aspectos y dominio (sujeto, objeto) de autoconsciencia de un sistema informático. Este framework ha sido descrito detalladamente en la Subsección 2.2.6. A su vez, se han propuesto dos arquitecturas de referencia ampliamente aceptadas para describir y diseñar la autoconsciencia en sistemas informáticos. La primera denominada EPiCS [83], proporciona una perspectiva arquitectónica que considera explícitamente los posibles procesos y niveles de autoconsciencia de un sistema informático, así como la interrelación entre estos. Esta arquitectura incluye tanto procesos de aprendizaje que generan modelos con diferentes niveles de autoconsciencia como procesos de razonamiento (con

capacidades de autoexpresión) impulsados por los objetivos del sistema para tomar decisiones sobre cómo actuar. Mientras que, la segunda arquitectura referencial es el bucle de aprendizaje y razonamiento basado en modelos (LRA-M loop) [19] que igualmente captura los principales procesos de un sistema informático autoconsciente. El bucle LRA-M fue explicado detenidamente en la subsección 2.2.5. Tanto el framework conceptual como las arquitecturas de referencia citadas, proporcionan una base común de principios y lineamientos sobre los cuales los investigadores y profesionales pueden estructurar sus trabajos y proponer nuevas soluciones tecnológicas. Tal es el caso de un trabajo reciente de Lewis y Sarkadi [132], en el cual se cuestiona que los sistemas actuales de inteligencia artificial carecen de una capacidad de autoconsciencia (reflexión). Por lo tanto, los autores proponen una arquitectura de agentes con inteligencia artificial reflexiva basada en teorías cognitivas, destacando los desafíos de investigación para su implementación. Esta solución integra la arquitectura de agente crítico [133] con la arquitectura LRA-M loop [19], con el propósito de dotar a los agentes de un conjunto de procesos de cognición reflexivas, de modo que, les permita alcanzar diferentes formas de autoconsciencia en tiempo de ejecución.

3.2.2 Soluciones de autoconsciencia basadas en enfoques genéricos

En esta subsección se describen estudios que van más allá de lo conceptual y proponen soluciones de autoconsciencia para IoT y otros dominios relacionados que han sido sujetas a algún tipo de evaluación empírica. A su vez, los estudios que se tratan en este apartado no consideran a MDE en sus propuestas.

Según Götzinger et al. [134], un obstáculo importante que debe superarse es el alto costo de implementar sistemas autoconscientes. La falta de un framework común hace que el desarrollo de cada sistema sea siempre desde cero, siendo redundante, ineficaz e improductivo. Por lo tanto, los autores proponen un framework de software, denominado RoSA (Research on Self-Awareness), que modela y evalúa la autoconsciencia en sistemas ciber físicos. Específicamente, RoSA implementa sistemas basados en una arquitectura jerárquica de agentes con funcionalidades de autoconsciencia, los cuales operan utilizando el bucle de control ODA (observar, decidir, actuar) [135]. Un agente RoSA es cualquier cosa con capacidades de percibir el entorno a través de sensores y actuar sobre el mismo mediante actuadores [133]. Además, estos agentes encapsulan funcionalidades basadas en conceptos de autoconsciencia, siendo capaces de generar conocimiento fino o grueso, dependiendo del nivel jerárquico en el que se encuentran. Los autores demostraron la aplicabilidad y flexibilidad de RoSA mediante dos estudios de caso orientados al monitoreo de la salud humana y diagnóstico de máquinas industriales.

Asimismo, Möstl et al. [136] presentan el framework IPF (Fábrica de Procesamiento de Información), mismo que está fundamentado en los principios de operación de una fábrica, en donde para mantener una producción eficaz y eficiente, los diversos aspectos del proceso de fabricación (logística de los suministros, maquinaria, instalaciones, transporte, etc.) deben controlarse y adaptarse continuamente a la carga de trabajo y restricciones actuales. De forma semejante, los sistemas ciber físicos basados en MPSoC (Multiprocessor System-on-Chip) son comparables con una fábrica. De ahí que, IPF propone una plataforma autoconsciente para el control autónomo y holístico de todos los componentes de hardware y software de un sistema ciber físico en chip, conforme a sus diversos objetivos (rendimiento, consumo de energía, confiabilidad, etc.). Esta plataforma tiene una arquitectura jerárquica compuesta por un conjunto de entidades de control SO/SA (autoorganización/autoconsciencia) distribuidas en varios niveles de abstracción (hardware, sistema operativo, middleware, aplicación) de un sistema ciber físico. Las

entidades SO/SA receptan y analizan los datos de los sensores; y, consecuentemente, generan directivas de actuación para afectar a los componentes del sistema. La correlación entre las múltiples entidades SO/SA, ubicadas en el mismo o diferentes niveles de abstracción, se establece mediante un conocimiento entre la actuación e implicación causada (p. ej., cómo una política de programación de tareas afecta el consumo de energía). En este estudio, el funcionamiento de IPF se ilustra mediante un caso de uso orientado a controlar la plataforma de un teléfono inteligente, a nivel de hardware y software. Cabe recalcar que otros estudios existentes también abordan la autoconsciencia en sistemas ciber físicos en chip, enmarcándose en la aplicación de la autoconsciencia como un medio para equilibrar de forma autónoma los recursos que usan este tipo de sistemas, considerando sus objetivos y restricciones cambiantes [137][138][139].

Por su parte, Habib et al. [140] desarrollan un framework de autoconsciencia y autorrediseño, denominado SASR, que incorpora un sistema de conocimiento de anomalías y autoadaptación para redes de sensores inalámbricos (WSN) implementadas en exteriores. El framework SASR crea una autoconsciencia híbrida dentro de cada sensor al incorporar una relación causal entre los cambios meteorológicos (consciencia pública) y las correspondientes pérdidas en la señal de transmisión (consciencia privada). Con base en este conocimiento, el sistema desencadena autoadaptaciones mediante mecanismos de saltos entre canales con diferentes potencias de transmisión. El simulador Contiki's Cooja fue utilizado para evaluar SASR a partir de las métricas: tasa de entrega de paquetes, latencia y consumo de energía.

Teniendo como propósito la autorrecuperación en aplicaciones IoT, Angarita [141] introduce el concepto de objetos inteligentes responsables y los requisitos para construir una API para este tipo de objetos. Según el autor, un objeto es responsable si combina capacidades transaccionales y de autoconsciencia para afrontar los entornos cambiantes y poco fiables en los que operan los sistemas IoT. Las propiedades transaccionales (recuperación hacia atrás y hacia adelante) son útiles para garantizar la ejecución confiable y atómica de cada uno de los componentes, asegurando en todo momento, la coherencia global del sistema IoT, incluso ante la presencia de fallas. Mientras que, la autoconsciencia permite que un objeto sea consciente del progreso de su ejecución, de los objetivos que debe alcanzar (criterios de calidad de servicio - QoS); y, de sus capacidades transaccionales y de otros objetos, a fin de tomar decisiones conjuntas de autorrecuperación. La utilidad de esta propuesta se ilustra mediante un caso de uso en el subdominio e-salud. En un trabajo más actual, Ettazi y Nassar [142] proporcionan un enfoque integral para abordar la confiabilidad, adaptabilidad y autoconsciencia en aplicaciones IoT, las cuales encapsulan la funcionalidad de las cosas mediante composición de servicios. Para ello, los autores establecen un modelo REASON conformado por cinco facultades cognitivas (razonamiento, reconocimiento, interacción, autocontrol, aprendizaje) que actúan sobre las funcionalidades de IoT (monitorear, actuar, comunicar). A su vez, el enfoque define mecanismos de recuperación ante fallos, fusionando las facultades cognitivas con las propiedades transaccionales, según las características funcionales de las cosas. Este artículo incluye una prueba de concepto sobre una aplicación IoT de monitoreo para pacientes diabéticos.

Por otro lado, Zoppi et al. [143] proponen una metodología basada en aprendizaje automático para implementar detectores de anomalías en dispositivos Edge o IoT con recursos limitados, a fin de que sean autoconscientes tanto de su correcto comportamiento como de estar bajo un ataque o intrusión. La metodología propuesta incluye cuatro etapas: i) identificar las anomalías que se desean detectar; ii) crear un sistema de monitoreo ligero para dispositivos IoT que recolecte indicadores sobre su rendimiento; iii) ejecutar el

sistema de monitoreo para tomar muestras del comportamiento del dispositivo objetivo en condiciones normales de funcionamiento e inyectando aleatoriamente errores; y, v) utilizar los datos recopilados para entrenar diversos algoritmos de aprendizaje de detección de anomalías e instalar el modelo aprendido con mejor rendimiento y precisión en los dispositivos IoT destino. La evaluación experimental se orientó a la detección de anomalías en el consumo de recursos y bloqueo de procesos en dispositivos IoT ARANCINO. Adicionalmente, existen otros trabajos como el de Mohamudally [144] y Jin et al. [145], que estudian y comparan diferentes modelos de inteligencia artificial con el propósito de volver viable la incorporación de la autoconsciencia colectiva y distribuida en los dispositivos ubicados al borde de una red de IoT.

3.2.3 Soluciones de autoconsciencia basadas en enfoques de MDE

Aunque la mayoría de los estudios existentes en la literatura sobre autoconsciencia en IoT y otros dominios relacionados no incorporan o consideran explícitamente alguno de los paradigmas de MDE como parte de sus soluciones, en este apartado se rescatan las siguientes propuestas.

Preden et al. [146] proponen una arquitectura que contempla los conceptos de autoconsciencia situacional y atención, como factores claves para implementar sistemas ciber físicos eficientes y confiables, sobre todo a nivel de la computación en el borde. Por lo general, la autoconsciencia situacional requiere de la fusión de datos y metadatos proveniente de múltiples sensores conectados a diferentes nodos de computación, con el fin de describir la situación espacial y temporal de un sistema y su entorno. Mientras que, la atención hace referencia a la asignación selectiva y progresiva de recursos limitados a tareas específicas. En este sentido, el estudio busca el equilibrio entre un nivel adecuado de autoconsciencia situacional y el procesamiento distribuido requerido para este fin. Para ello, los autores proponen una abstracción jerárquica de la información situacional, que va desde el sensado (propiedades internas y del entorno) hasta el contexto percibido (situaciones de bajo nivel), la comprensión (situaciones de nivel superior) y la proyección (estimación de situaciones futuras). Como complemento, la arquitectura incluye un modelo actualizado dinámicamente, contra el cual se comparan los datos monitoreados para determinar una situación inequívoca. En el caso de no llegar a una situación concluyente, un mecanismo de atención desencadena la recolección y análisis de datos adicionales, a fin de elevar el nivel de abstracción de la información situacional y refinar el modelo. La arquitectura se ejemplifica mediante un prototipo de sistema de monitoreo de salud.

El estudio de Möstl et al. [136], analizado en la subsección anterior, adicionalmente presenta el framework CCC (Control de Cambios Concurrentes) enfocado en construir sistemas ciber físicos autoconscientes con capacidades de monitoreo, automodelado y autoconfiguración. Este framework tiene una arquitectura compuesta de dos dominios. El dominio de ejecución contempla los componentes de hardware (plataforma) y software (funcional) de un sistema ciber físico observado. Además, este dominio posee capacidades de monitoreo y configuración de los parámetros de los componentes del sistema. El dominio de modelo incluye los modelos interrelacionados funcional y de plataforma del sistema. Todos los cambios en el dominio de ejecución son gestionados por el dominio del modelo mediante el controlador multicambio (MCC). Este controlador se basa en un formalismo de contratación, de tal forma que, si el comportamiento del dominio de ejecución cumple con los supuestos del dominio del modelo, el sistema observado funciona correctamente. Operativamente, el controlador recepta las métricas monitoreadas por el dominio de ejecución y detecta desviaciones entre el comportamiento

del sistema real y los modelos. En caso de desviaciones, el controlador actualiza los modelos para hacerlos coherente con el sistema observado y mediante mecanismos de análisis de dependencias entre modelos y análisis de contratos encuentra una configuración adecuada que garantice el cumplimiento de los requisitos contratados. El funcionamiento de CCC se ilustra mediante un ejemplo aplicado a un subsistema de automoción que gestiona el guiado lateral y longitudinal de un vehículo en carretera.

En el ámbito de la autoconsciencia colectiva, Esterle y Rinner [147] presentan un enfoque de diseño de aplicaciones IoT, el cual se basa en una arquitectura autoconsciente que utiliza modelos de interacción colectiva entre los nodos IoT para mejorar el desempeño y alcanzar los objetivos a nivel de toda la red de IoT. En esta arquitectura, cada nodo IoT posee un componente de autoconsciencia a nivel de nodo y otro a nivel de red. La información proveniente de los sensores físicos y virtuales de un nodo IoT alimenta la autoconsciencia a nivel de nodo, mientras que, la información recibida de otros nodos alimenta la autoconsciencia a nivel de red. Ambos componentes de autoconsciencia crean y refinan modelos compartidos a partir de las observaciones. Sin embargo, estos modelos pueden ser utilizados de forma diferente por los componentes, en función de los objetivos y limitaciones del nodo IoT. Los autores aplicaron el enfoque propuesto al problema de cubrir múltiples objetos con cámaras móviles, demostrando que la autoconsciencia en red maximiza la cobertura de los objetos.

En la computación ubicua o pervasiva, un área entrelazada con IoT, Gerbert-Gaillard et al. [148] demuestran cómo la autoconsciencia puede usarse pragmáticamente en la computación ubicua. Para ello, proponen un enfoque arquitectónico basado en MDE para gestionar y razonar en tiempo de ejecución sobre un tríptico de modelos (objetivo, core, contexto externo). En este trabajo se describe la función de cada uno de los modelos, tomando como ejemplo una aplicación autoconsciente de alerta de incendios en edificios. En trabajos posteriores, Gerbert-Gaillard et al. [149] y Lalanda et al. [150] proponen una solución autoconsciente para la gestión dinámica del contexto en la plataforma ubicua orientada a servicios, denominada iCasa. Los autores plantean una nueva arquitectura para iCasa compuesta por tres capas: comunicación, contexto y aplicación. Cada capa de la arquitectura dispone de un gestor autonómico, basado en tres modelos en tiempo de ejecución (objetivos, arquitectura, recursos), lo cual hace que cada capa sea consciente de sí misma; es decir, que conozca sus objetivos, estructura interna y recursos externos que puede utilizar. Por ejemplo, el gestor de la capa de contexto razona sobre los modelos para proporcionar una configuración apropiada que satisfaga las necesidades (objetivos) de la capa de aplicación. A su vez, esta configuración debe ser consistente con las propias capacidades internas de la capa de contexto y los recursos disponibles en el entorno, proveídos por la capa de comunicación. Esta solución autoconsciente se evaluó en términos de complejidad y consumo de recursos en varios casos de uso de Orange Labs.

En un ámbito general, Spinner et al. [151] presentan un análisis de los tipos de modelos en tiempo de ejecución que pueden ser usados para la gestión de recursos y rendimiento autoconsciente en centros de procesamiento de datos (datacenters). Los autores proporcionan un esquema de clasificación que permite comprender el propósito, los supuestos y la estructura de cada uno de los modelos. Los tipos de modelos que se abordan son: modelos de regresión estadística, redes de colas, modelos teóricos de control y modelos descriptivos (p. ej., Lenguaje de Modelado de Descartes - DML⁵). A su vez, este

⁵ DML (Descartes Modeling Language) [168] es un lenguaje a nivel de arquitectura para modelar aspectos relacionados con la gestión de recursos y rendimiento en sistemas de software dinámicos y su infraestructura de TI, formalizado por un conjunto de metamodelos basados en Meta-Object Facility (MOF).

trabajo incluye dos estudios de caso, uno sobre un modelo de control utilizado para desarrollar estrategias de control que mantengan el tiempo máximo de respuesta de una aplicación en la nube bajo los límites deseados; y, otro sobre un modelo en DML utilizado para predecir y optimizar los recursos de un sistema BlueYonder que brinda servicios de análisis de Big Data en la nube. Adicionalmente, Kounev et al. [152] y Huber et al. [153] proponen un enfoque basado en modelos para la gestión de recursos y rendimiento autoconsciente en sistemas de TI, con el apoyo del lenguaje DML y el bucle de control MAPE-K [131]. Este enfoque permite predecir en tiempo de ejecución: i) los cambios en la carga de trabajo de las aplicaciones de software; ii) los efectos de dichos cambios en el rendimiento de los recursos del sistema; y, iii) el impacto esperado de las acciones de adaptación en el rendimiento del sistema. A su vez, mediante la información de predicción, se implementa un bucle de control MAPE-K basado en modelos para la adaptación proactiva del sistema. Los autores evalúan la propuesta en el contexto de dos estudios de caso sobre sistemas empresariales, los cuales demuestran cómo el enfoque puede alcanzar mejoras en la eficiencia de los recursos. En cambio, Walter et al. [154] explica el aprendizaje en línea de los modelos en tiempo de ejecución en el ámbito de la gestión de recursos y rendimiento en centros de procesamiento de datos. Los autores clasifican el aprendizaje y mantenimiento de los modelos de rendimiento en tres áreas: i) estructura del modelo; ii) parametrización del modelo (estimación y calibración de los parámetros del modelo); y, iii) opciones de adaptación del modelo (detección de puntos de cambio y reconfiguración en tiempo de ejecución).

Por último, Lalanda et al. [18] utilizan modelos arquitectónicos de diseño y tiempo de ejecución para soportar la autoconsciencia. Para ello, construyen una relación causal entre modelos de diseño abstractos y modelos de tiempo de ejecución más concretos. Esto posibilita que los sistemas puedan comprobar la validez de su estado con respecto a los objetivos de diseño; y, como consecuencia, se administren por sí mismo y minimicen la necesidad de intervención humana. Este enfoque se ilustra mediante una aplicación de salud para un hogar inteligente llamada Actimetrics, desarrollada con Orange Labs. Un aspecto que resalta este trabajo, consiste en la falta de herramientas que gestionen todo el ciclo de vida del software; es decir, es fundamental que las herramientas arquitectónicas ayuden a los desarrolladores y administradores a modelar la arquitectura tanto en el momento del diseño como en la implementación y ejecución. La mayoría de las veces, estos artefactos se desarrollan o modelan con diferentes herramientas y se conservan por separado.

3.2.4 Soluciones de autoconsciencia (ad hoc) para aplicaciones particulares de IoT

El estudio de la autoconsciencia también se ha enfocado en aplicaciones individuales o específicas de IoT. A continuación, se abordan las principales propuestas de autoconsciencia organizadas por área de aplicación.

En la salud, el concepto de autoconsciencia se ha introducido mayoritariamente con el propósito de distribuir adecuadamente la carga de trabajo entre las capas Edge, Fog y Cloud; y, por ende, alcanzar una compensación entre el consumo de energía de los dispositivos Edge (wearables) y la calidad de los resultados de las soluciones de monitoreo de salud. Naeini et al. [155] proponen un sistema de monitoreo del dolor en tiempo real, basado en el bucle de control ODA (observar, decidir, actuar) [135] y dos modelos de aprendizaje automático con diferentes características de precisión y consumo de energía. El primer modelo evalúa el dolor mediante señales fisiológicas, a nivel de la capa de sensores. Mientras que, el segundo modelo multimodal analiza el dolor mediante reconocimiento de expresiones faciales y señales fisiológicas, a nivel de la capa de puerta

de enlace. ODA analiza el estado del sistema y su entorno (duración de la batería, nivel de dolor previsto); y, selecciona el modelo más adecuado para su ejecución. De igual manera, Forooghifar et al. en sus trabajos [156][157], plantean un sistema wearable de bajo consumo de energía para la detección de ataques epilépticos. Este sistema incluye un selector autoconsciente en el módulo de aprendizaje automático para cambiar a modo de bajo consumo, cuando una versión más simple del sistema puede cumplir con la calidad de la detección deseada. A su vez, los autores estudian diferentes estrategias de autoconsciencia para distribuir los complejos cálculos de los algoritmos de aprendizaje automático entre las capas Edge, Fog y Cloud, a fin de maximizar la vida útil de la batería de los dispositivos perimetrales [157]. Por su parte, Dell’Agnola et al. [158] presentan un sistema wearable autoconsciente para el monitoreo de la carga de trabajo cognitiva de operadores en situaciones de alto riesgo o que realizan tareas por mucho tiempo, a fin de comprender su estado mental y rendimiento. Este sistema posee una arquitectura multinivel y multimodal que permite la activación / desactivación de sensores fisiológicos (respiración, frecuencia cardíaca, temperatura de la piel, etc.) y algoritmos de aprendizaje automático con diferentes niveles de complejidad, en función de intervalos de confianza de la detección. Por último, Anzanpour et al. [159] presentan un sistema de puntuación de alerta temprana (Early Warning Score - EWS) autoconsciente para el monitoreo remoto de la salud de pacientes. Los autores discuten la utilización de la autoconsciencia en diferentes partes del sistema, como la recolección de datos desde sensores wearables, el razonamiento sobre la situación del paciente y la evaluación de la confianza de la puntuación calculada. Además, contemplan la propiedad de atención de la autoconsciencia para mejorar la eficiencia energética y la confiabilidad del sistema mediante el ajuste de prioridades en la recolección de datos sensoriales.

En el ámbito del transporte, Schlatow et al. [160] manifiestan que en sistemas críticos como el automotriz, la autoconsciencia es una propiedad transversal a todas las capas de un sistema (plataforma hardware/software, comunicación, sensores, funcionalidades propias de la conducción autónoma). Es así que, los autores presentan varios enfoques y casos de uso que ilustran la gestión de la autoconsciencia en cada una de estas capas, destacando el framework CCC (Control de Cambios Concurrentes) para gestionar los cambios de la plataforma de hardware/software en el campo automotriz y espacial; y, la arquitectura de virtualización de controladores automotrices para procesadores multinúcleo que permite la segregación temporal y espacial de aplicaciones autoconscientes que presentan cambios continuos. Por otra parte, Kaiser et al. [161] proponen el framework ToSaMuFo que combina la autoconsciencia computacional y la formación de pelotones de cuadricópteros (drones). El propósito del framework es permitir que los drones sean capaces de aprender de su entorno, evitar obstáculos o colisiones, de tal forma que puedan movilizarse con éxito como parte de un pelotón. Para ello, este framework propone una modificación al bucle de autoconsciencia LRA-M, denominada consciencia de pelotón. Este bucle LRA-M adaptado está compuesto por dos componentes: i) *módulo de pelotón*, encargado del ensamblaje de los drones sobre la marcha y de gestionar la distancia de seguridad entre ellos; y, ii) *módulo de consciencia*, responsable de mantener un mapa del entorno, encontrar una ruta adecuada a través de este mapa, resolver conflictos generados por el módulo de pelotón y evitar colisiones. Los autores implementan un prototipo de ToSaMuFo utilizando el sistema operativo para robots (ROS) y la plataforma de experimentación de robots virtuales (V-REP). En otro estudio, Akbar y Belhaouarie [162] presentan SaVE, un enfoque que incorpora la autoconsciencia en la computación de borde vehicular (VEC) para comprender mejor el estado interno y el entorno de los vehículos (movilidad, condiciones de red, consumo de energía, tiempo de respuesta, etc.). SaVE emplea un algoritmo de optimización por

enjambre de partículas exponencial adaptado al entorno VEC (VExPSO) para buscar de manera eficiente servidores de borde óptimos para la descarga de tareas, en caso de ser necesario. Además, SaVE utiliza un algoritmo de gradiente de política determinista profundo (DDPG) para optimizar el consumo de energía y la latencia; y, un algoritmo de aprendizaje federado (FL) jerárquico para entrenar de forma colaborativa el modelo DDPG en todos los vehículos.

En el área de la vigilancia y seguridad, Rinner et al. [163] y Esterle et al. [164] plantean un framework que adopta conceptos de autoconsciencia y autoexpresión en una red de cámaras inteligentes para analizar en el borde y colaborativamente el comportamiento dinámico de los objetos en escenarios parcialmente desconocidos, compensando al mismo tiempo, los objetivos contrapuestos de rendimiento, consumo de recursos y confiabilidad. El framework está compuesto por seis bloques: seguimiento de objetos, transferencia de objetos a otras cámaras, aprendizaje de la topología de red, selección de estrategias de seguimiento, monitoreo de recursos de la cámara; y, gestión de restricciones y objetivos de las cámaras. Estos bloques implementan diferentes niveles de autoconsciencia. Si bien el bloque de seguimiento de objetos sólo tiene en cuenta los estímulos, otros bloques, como la transferencia de objetos, el aprendizaje de topología o la selección de estrategias, tienen en cuenta la interacción, el tiempo y la meta-autoconsciencia. En la misma línea de investigación, Guettatfi et al. [165] desarrollan un enfoque de diseño autoconsciente para redes de sensores visuales (VSN). Una VSN consiste de un conjunto de nodos de cámaras inteligentes que integran: sensor de imagen, sistema de procesamiento y transceptor inalámbrico. Este enfoque adapta la arquitectura de referencia EPiCS [83] para un nodo VSN, la cual está compuesta por tres módulos: i) autoconsciencia privada, abstrae los datos desde los sensores internos del nodo y mantiene los modelos sobre las imágenes capturadas, los recursos utilizados y la calidad de servicio (QoS), ii) *autoconsciencia pública*, recibe la información de otros nodos y actualiza los modelos desde la perspectiva de red; y, iii) *autoexpresión*, emplea los modelos para determinar autoadaptaciones de la plataforma de hardware y software, a nivel del nodo y la red.

Por último, la autoconsciencia se ha aplicado también en la robótica, los trabajos de Akbar et al. [166][167] proponen un framework autoconsciente para robots móviles que optimiza el consumo de energía y uso de la red. Este framework busca en tiempo de ejecución configuraciones eficientes para tareas robóticas híbridas móvil-nube (MCH) que compensen adecuadamente el uso de la batería y la red, conforme a los cambios del entorno.

3.2.5 Discusión

La revisión bibliográfica realizada evidencia que la investigación sobre autoconsciencia en IoT y otros dominios relacionados se encuentra fragmentada en diferentes disciplinas y comunidades, tales como: inteligencia artificial, sistemas multiagentes, computación autónoma, computación orgánica, teoría de control, sistemas autoadaptativos, MDE, entre otros [21][134]. Como consecuencia, los esfuerzos de investigación y desarrollo de sistemas IoT autoconscientes se están duplicando y avanzan a un ritmo bastante lento, encontrándose aún importantes problemas poco explorados por la comunidad científica [23]. A continuación, se discuten las principales limitaciones y desafíos identificados a partir del trabajo relacionado, los cuales son abordados en esta tesis doctoral a través de la solución metodológica propuesta.

- Un gran porcentaje de estudios del trabajo relacionado proponen soluciones de

autoconsciencia dirigidas a aplicaciones particulares de IoT. Sin embargo, varios de los autores representativos del área de los sistemas autoconscientes [24][134][146] manifiestan que no es una buena práctica diseñar e implementar la autoconsciencia a medida (ad hoc) para cada nuevo sistema; de ahí que, incorporarla como una incumbencia (concern), independiente de la lógica de negocio de la aplicación, promete simplificar el desarrollo y operación de este tipo de sistemas. Por lo tanto, uno de los retos actuales consiste en la creación de enfoques metodológicos y herramientas de desarrollo genéricas que puedan ser aplicadas a cualquier subdominio de IoT.

- A su vez, el trabajo relacionado se ha centrado fundamentalmente en la construcción de arquitecturas, frameworks, middlewares y algoritmos de aprendizaje automático para apoyar la autoconsciencia en sistemas IoT y otros sistemas relacionados, descuidando la concepción de propuestas metodológicas integrales, basadas en paradigmas emergentes (p. ej., *models@run.time*), que cubran todo el ciclo de vida del software, desde el momento del diseño hasta la implementación y operación. En este sentido, existe la necesidad de proponer nuevos métodos de ingeniería para construir sistemas IoT con propiedades autoconscientes, adaptando frameworks y arquitecturas de referencia ampliamente aceptadas (p. ej., bucle LRA-M) [24]. Estos nuevos procesos metodológicos deben focalizar los esfuerzos de los desarrolladores y expertos de dominio en dar respuesta a las preguntas de diseño sobre cuándo, dónde y cómo incorporar las diferentes formas de autoconsciencia en sistema IoT (pre-reflexivas, reflexivas, meta-reflexivas), en procura de crear sistemas con mejores capacidades autonómicas ante los escenarios inciertos en que operan [24][25].
- Varios autores destacan la importancia de utilizar los enfoques de MDE y, particularmente los modelos en tiempo de ejecución en la construcción de sistemas IoT autoconscientes, debido a que los modelos son un medio potencial para capturar, representar y abstraer el conocimiento sobre el estado, comportamiento y evolución del sistema y su entorno operativo [21][26][27][33]. Sin embargo, como se puede observar en la revisión bibliográfica, existen pocos estudios que consideran explícitamente a MDE como parte de sus soluciones autoconscientes. Esto corrobora lo señalado por Bencomo et al. [33] en su revisión sistemática sobre *models@run.time*, cuando concluyen que uno de los propósitos y desafíos abiertos de los modelos en tiempo de ejecución es apoyar a la autoconsciencia computacional en los sistemas modernos (p. ej. IoT). Como complemento a lo mencionado en el párrafo anterior, se requiere de enfoques metodológicos, en donde los modelos sean el vínculo para gestionar con un alto grado de abstracción la autoconsciencia a lo largo de todas las etapas del ciclo de vida de un sistema IoT [26].
- Particularmente, las propuestas de DSLs y metamodelos para representar las preocupaciones de autoconsciencia de un sistema informático son escasas. Una herramienta en este ámbito es el Lenguaje de Modelado de Descartes (DML); sin embargo, se orienta únicamente a la gestión autoconsciente del rendimiento de los recursos informáticos en infraestructuras de TI [168]. Por tal motivo, resulta indispensable disponer de un metamodelo que proporcione un lenguaje capaz de representar una amplia variedad de niveles y aspectos de autoconsciencia en el contexto de IoT, más aún cuando estos sistemas están compuestos por múltiples nodos de computación con variadas capacidades y distribuidos a nivel de las capas Edge, Fog y Cloud de una infraestructura de IoT. Específicamente, el metamodelo debe ser lo suficientemente completo y flexible para: i) soportar la autorreflexión sobre

diversos aspectos (p. ej., estado, evolución) de las entidades IoT, tanto físicas como digitales; ii) definir una autoconsciencia centralizada en un único nodo de computación; o, distribuida y compartida entre múltiples nodos de computación, ubicados en las capas Edge, Fog o Cloud; iii) representar la autoconsciencia con diferentes niveles de abstracción y agregación, por una parte, a nivel de autoconsciencia pre-reflexiva, reflexiva o meta-reflexiva y, por otra parte, a nivel de autoconsciencia individual o colectiva; y, iv) valorar los aspectos de autoconsciencia de manera objetiva, utilizando técnicas cuantitativas y cualitativas.

- Existen estudios relevantes sobre autoconsciencia computacional que proporcionan una base conceptual sólida para proponer nuevas soluciones tecnológicas. Por consiguiente, los nuevos enfoques metodológicos y herramientas de soporte para la construcción de sistemas IoT autoconscientes deben aprovechar este tipo de estudios con el propósito de asegurar su formalidad, rigurosidad y calidad. Puntualmente, en este grupo de estudios se destacan el framework conceptual de niveles y aspectos de autoconsciencia de Lewis et al. [20] y la arquitectura de referencia para sistemas informáticos autoconscientes LRA-M Loop de Kounev et al. [19].
- En el trabajo relacionado se pudo observar una preferencia por utilizar las arquitecturas de referencia ODA [135] y MAPE-K [131] en la construcción de soluciones autoconscientes para el dominio de IoT y otros afines. Sin embargo, si bien los sistemas basados en estas arquitecturas poseen cierto nivel de autoconsciencia, no capturan explícitamente las preocupaciones sobre la autoconsciencia. Por el contrario, existen otras arquitecturas cuya finalidad es enfatizar explícitamente estas preocupaciones y proporcionar una perspectiva arquitectónica sobre las posibles capacidades de autoconsciencia de un sistema [24]. En esta línea, la arquitectura LRA-M Loop [19] ha sido el resultado de un gran esfuerzo comunitario en el seminario de Dagstuhl, denominado "Algoritmos y arquitecturas dirigidas por modelos para sistemas informáticos autoconscientes" [81], el cual reunió a importantes investigadores de diferentes comunidades relacionadas con diversos aspectos de la autoconsciencia computacional.
- De igual manera, varios autores [18][24][25] consideran necesario crear nuevas herramientas de desarrollo de sistemas IoT o ampliar las existentes para soportar la nueva dimensión de autoconsciencia. Entre los principales desafíos están proponer herramientas basadas en modelos que describan a alto nivel las preocupaciones de autoconsciencia, a fin de abstraer la complejidad del desarrollo y minimizar los detalles de implementación (programación a bajo nivel). Además, estas herramientas deben utilizar los modelos como artefactos transversales a todas las etapas del ciclo de vida del software (diseño, implementación, operación), lo cual resulta beneficioso para elevar el nivel de automatización del proceso y mejorar la gestión del cambio. Asimismo, las herramientas deben alinearse con patrones de diseño (p. ej. patrón Pub/Sub) que favorezcan la construcción de aplicaciones IoT ligeras y multiplataforma, de tal forma que sea viable técnicamente el despliegue de una autoconsciencia distribuida y colectiva a nivel de las capas Edge, Fog y Cloud de una infraestructura de IoT, sobre todo en aquellos dispositivos con capacidades limitadas de hardware, energía y ancho de banda.
- Un aspecto destacable son los numerosos estudios sobre modelos de minería de datos y aprendizaje automático orientados a soportar la autoconsciencia en sistemas IoT y otros similares. En tal virtud, los nuevos enfoques metodológicos y herramientas de desarrollo deben aprovechar estos modelos, tratándolos como algoritmos de caja

negra, cuya reutilización se logre mediante especificaciones de alto nivel de los parámetros de entrada y salida de dichos algoritmos.

- Por último, si bien el trabajo relacionado ha hecho uso de pruebas de concepto, estudios de caso y experimentos para evaluar empíricamente las soluciones autoconscientes en términos de funcionalidad, aplicabilidad, rendimiento, precisión, entre otros; no se ha observado en la literatura, estudios que validen la percepción del usuario final sobre la facilidad de uso, utilidad e intención de uso futuro de sus propuestas, mediante el empleo de técnicas basadas en estadística inferencial (pruebas de hipótesis). Este tipo de evaluaciones resulta fundamental para determinar la posible adopción de métodos o herramientas tecnológicas por parte de la industria de software.

3.3 Otras soluciones basadas en modelos en tiempo de ejecución para IoT

Dentro del dominio IoT se han propuesto otros estudios que utilizan los modelos en tiempo de ejecución con fines o propósitos distintos a la autoconsciencia computacional, tales como: abstracción de la plataforma, autoadaptación, autoorganización, autooptimización, autorrecuperación, interoperabilidad, aseguramiento de calidad, etc. Entre las principales fuentes para recabar mayor información sobre estos estudios están las revisiones sistemáticas sobre modelos en tiempo de ejecución de Szvetits y Zdun [30], Bencomo et al. [33] y, Erazo-Garzón et al. [39], en donde las dos primeras revisiones abordan el estudio de los *models@run.time* desde un ámbito general, mientras que la tercera revisión enmarca su estudio dentro del dominio de IoT.

3.4 Conclusiones

A partir de una revisión bibliográfica exhaustiva sustentada en importantes estudios secundarios (revisiones sistemáticas) disponibles en la literatura y complementada con la selección de relevantes estudios primarios obtenidos a partir de reconocidas bibliotecas digitales en el área de la informática (IEEE Xplore, ACM Digital Library, SpringerLink y ScienceDirect) y la técnica de bola de nieve, se construyó un estado del arte completo y actualizado sobre las principales áreas y preocupaciones que atañen a esta tesis doctoral, como son: autoconsciencia computacional e Ingeniería Dirigida por Modelos en IoT y otros dominios relacionados. En particular, a pesar de que la investigación sobre autoconsciencia computacional se encuentra fragmentada en diferentes disciplinas y comunidades, se hizo un gran esfuerzo por unificar y sintetizar en un solo estudio, los más importantes trabajos y avances de esta área de investigación, siendo un aporte destacable de esta tesis doctoral.

Adicionalmente, se analizó y discutió minuciosamente las fortalezas y limitaciones del trabajo relacionado; y, consecuentemente, se hicieron explícitas varias brechas y desafíos de investigación actuales sobre la autoconsciencia computacional que evidentemente conllevan a repensar sobre el diseño y desarrollo de los sistemas IoT.

En términos generales, mediante la revisión bibliográfica se comprobó que la investigación sobre autoconsciencia computacional en IoT y otros dominios relacionados se ha direccionado prioritariamente a la construcción de arquitecturas, frameworks, middlewares y algoritmos de aprendizaje automático; en contraposición, con la ausencia de propuestas de tipo metodológicas. Incluso muchos de los estudios existentes han concentrado sus esfuerzos en soluciones a medida (ad hoc) para aplicaciones específicas de IoT. Por lo tanto, producto de esta revisión bibliográfica se corroboró la necesidad de proponer enfoques metodológicos e infraestructuras de soporte genéricas para la

construcción, operación y mantenimiento de sistemas IoT autoconscientes, siendo consecuente con el objetivo que persigue esta tesis doctoral.

Siendo más específicos, la revisión bibliográfica ha servido para reconocer y/o revalidar ciertos aspectos fundamentales que deben considerarse en la construcción de la solución metodológica para sistemas IoT autoconscientes, mismos que han sido parcialmente o nada explorados en el trabajo relacionado. Entre los principales aspectos se tienen:

- Incorporar a los modelos como artefactos centrales de la solución metodológica y su infraestructura de soporte, siendo el medio para gestionar con un alto grado de abstracción, automatización y flexibilidad, los requerimientos de monitoreo y autoconsciencia a lo largo de todas las etapas (diseño, implementación, operación) del ciclo de vida de un sistema IoT;
- Reutilizar, adaptar e integrar como parte de la solución metodológica varias propuestas relevantes del trabajo relacionado, como son: la arquitectura de referencia para IoT ISO/IEC 30141:2018 [2], la arquitectura de referencia para sistemas informático autoconscientes LRA-M Loop [19] y, el framework conceptual de autoconsciencia de Lewis et al. [20].
- Crear metamodelos y DSLs basados en un vocabulario amplio, expresivo y extensible para modelar o especificar cómo implementar los requisitos arquitectónicos, de monitoreo y autoconsciencia de un sistema IoT, teniendo en consideración las características intrínsecas de estos sistemas, como: altamente distribuidos (a nivel de capas Edge, Fog, Cloud), heterogeneidad, escalabilidad, entre otros. En este sentido, estas herramientas deben permitir el modelado o especificación de una amplia variedad de flujos de datos de monitoreo y, de niveles y aspectos de autoconsciencia. En estas herramientas, la arquitectura del sistema IoT debe ser la base para soportar los procesos de monitorización y autoconsciencia.
- Emplear técnicas de evaluación empírica con un alto rigor científico, tendientes a demostrar el rendimiento del usuario con el uso de la solución metodológica, así como su percepción sobre la usabilidad, utilidad e intención de uso de la misma a futuro.

Como conclusión final, los resultados sistematizados en este capítulo trazan una hoja de ruta que ha sido de gran utilidad para abordar adecuadamente las actividades de investigación concernientes a la construcción y evaluación de la solución metodológica, mismas que se detallan en los capítulos subsiguientes. Inclusive, esta hoja de ruta podrá ser aprovechada por la comunidad científica para direccionar apropiadamente sus proyectos de investigación futuros.

Capítulo 4

Metodología Propuesta Aware-IoT@Run.Time

En este capítulo se describe Aware-IoT@Run.Time, una metodología basada en modelos en tiempo de ejecución para la construcción, operación y mantenimiento de sistemas IoT autoconscientes. Esta metodología apoya a los desarrolladores con un alto grado de abstracción y flexibilidad en la implementación y reconfiguración de recursos de software que soporten los procesos de monitorización y autoconsciencia de un sistema IoT. Estos recursos permiten que un sistema IoT, mediante la utilización de los modelos en tiempo de ejecución, sea capaz de evaluar y comprender su estado y posible evolución, así como la de su entorno; y, en caso de ser necesario determinar las respectivas recomendaciones o acciones correctivas.

La sección 4.1 presenta una visión general de la metodología propuesta. Luego, en la sección 4.2 se profundiza en la descripción de cada una de las actividades de la metodología, descomponiéndolas en tareas y subtarefas. Finalmente, la sección 4.3 presenta las conclusiones del capítulo.

4.1 Definición de la metodología Aware-IoT@Run.Time

La metodología propuesta fue presentada en Erazo-Garzón [36] y está conformada por un conjunto de actividades, recursos, artefactos y herramientas que indican cómo construir, operar y mantener un sistema IoT autoconsciente. Durante el trabajo de investigación, este método ha sido objeto de varias evaluaciones empíricas (ver Capítulo 8), que han conllevado a un refinamiento y mejora continua, a fin de garantizar un proceso sistemático, ordenado, eficaz, eficiente y repetible.

La metodología está construida a partir del bucle de aprendizaje y razonamiento basado en modelos, por sus siglas en inglés LRA-M loop [19], el cual describe los principales procesos de un sistema informático autoconsciente, iniciando desde la observación directa para recopilar datos empíricos sobre el propio sistema y su entorno. Luego, se producen procesos de aprendizaje continuo para capturar el conocimiento sobre los aspectos relevantes del sistema IoT y su entorno, por medio de la creación y/o aplicación de modelos conceptuales o reflexivos que utilizan el historial de los datos empíricos recolectados u otros datos derivados a partir de estos. Finalmente, se ejecutan procesos de razonamiento sobre el conocimiento adquirido para comprender cuantitativa y cualitativamente el estado actual y la evolución del sistema. Según LRA-M, un sistema autoconsciente no está obligado a actuar (autoadaptar), por lo que la metodología propuesta llega hasta determinar las recomendaciones o acciones que deben realizarse para asegurar o apoyar la consecución de los objetivos del sistema IoT.

A su vez, varios recursos y artefactos que forman parte de la metodología han sido contruidos a partir de estándares o trabajos de investigación relevantes, entre los más importantes: la arquitectura de referencia para IoT ISO/IEC 30141:2018 [2], el framework conceptual de autoconsciencia de Lewis et al. [20] y la ontología de medición de software de García et al. [106].

En la Figura 4-1 se presenta un diagrama de proceso de alto nivel basado en la notación SPEM 2.0 (Software & Systems Process Engineering Meta-Model Specification) [169],

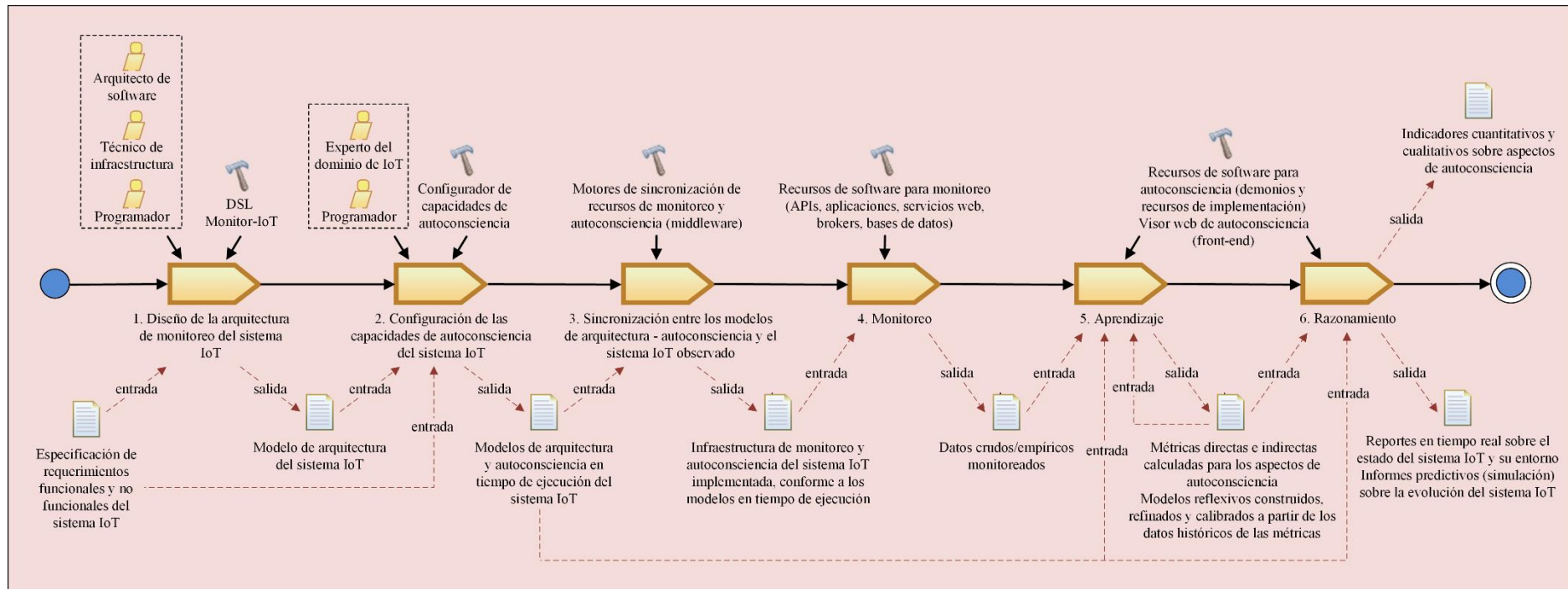


Figura 4-1. Proceso metodológico de alto nivel de Aware-IoT@Run.Time [36].

Fuente: Elaboración propia.

que muestra el orden de ejecución de las actividades que conforman la metodología, así como los roles de usuario, herramientas y artefactos de entrada y salida que se utilizan durante el proceso. La metodología está compuesta por seis actividades principales. Las dos primeras actividades requieren la intervención de los desarrolladores; mientras que, las cuatro restantes se ejecutan de manera automática con el apoyo de las herramientas de software propias de la infraestructura de soporte. Con relación a los roles de los usuarios, estos pueden solaparse en una misma persona o distribuirse en varias personas, dependiendo de los conocimientos y experiencia de los participantes del proyecto y el nivel de especialización del escenario de IoT a implementar.

Desde una visión general, el proceso metodológico propuesto comienza con dos actividades de modelado. La *primera actividad* está orientada al diseño visual del modelo de arquitectura de un sistema IoT que soporte los procesos de monitoreo, a nivel de las capas Edge, Fog y Cloud [64]. Mientras que, la *segunda actividad* se enfoca en la construcción de un modelo con especificaciones de alto nivel sobre las capacidades de autoconsciencia distribuidas de un sistema IoT. Estos dos modelos se encuentran interrelacionados y son construidos a partir de las herramientas DSL Monitor-IoT (ver Sección 5.2) y configurador de capacidades de autoconsciencia (ver Sección 5.3), las cuales utilizan un lenguaje de alto nivel conforme a los metamodelos Monitor-IoT (ver Anexo A) y Aware-IoT (ver Anexo B).

Por su parte, la *tercera actividad* utiliza los modelos de arquitectura y autoconsciencia en tiempo de ejecución para implementar de forma automática y distribuida los recursos de software que soportan la infraestructura de monitoreo y autoconsciencia del sistema IoT. Además, esta actividad ante cualquier cambio en los modelos en tiempo de ejecución es capaz de realizar reconfiguraciones sobre los recursos de software; es decir, mantiene la relación causal entre los modelos y el sistema IoT observado. Para lograr este cometido, en esta instancia se emplea un middleware compuesto por los motores de sincronización de recursos de monitoreo y autoconsciencia (ver Sección 5.4).

Las restantes actividades de la metodología utilizan los recursos de software de monitoreo y autoconsciencia implementados y puestos en operación. Estos recursos se encargan de recolectar, calcular y calibrar en tiempo de ejecución los parámetros (métricas, metadatos, etc.) especificados en los modelos, a fin de que reflejen en todo momento el estado y la evolución del sistema IoT en operación. En particular, la *cuarta actividad*, a través de los recursos de monitoreo, tiene como finalidad recolectar, transportar y procesar los datos crudos o empíricos de los sensores u otras fuentes; y, almacenarlos en las bases de datos previamente especificadas en el modelo de arquitectura. Por su parte, la *quinta actividad* utiliza los datos empíricos recopilados previamente y los recursos de autoconsciencia para producir las métricas directas e indirectas que valoran los aspectos de autoconsciencia de un sistema IoT. Finalmente, la *sexta actividad*, asimismo con el apoyo de los recursos de autoconsciencia, transforma las métricas directas e indirectas en indicadores cuantitativos y cualitativos, a fin de que el sistema IoT, a partir de estos, sea capaz de conocer su estado y posible evolución futura, así como proponer recomendaciones o acciones correctivas de ser necesario.

4.2 Descripción de las actividades de la metodología Aware-IoT@Run.Time

En esta sección se presenta una explicación completa de cada una de las actividades que conforman la metodología propuesta para la construcción, operación y mantenimiento de sistemas IoT autoconscientes, descomponiéndolas en tareas y subtareas de más bajo nivel. Además, se identifican los recursos (roles de usuario, guías), artefactos y herramientas

que se deben utilizar en cada una de las actividades y sus tareas. Como complemento, la aplicación formal, sistemática y ordenada del proceso metodológico a través de un estudio de caso real se ejemplifica en el Capítulo 7. Asimismo, dado que la metodología está basada en los metamodelos Monitor-IoT y Aware-IoT, los principales conceptos (metaclases y atributos) que se utilizan a lo largo de la descripción de la metodología se detallan a profundidad en los Anexo A y B.

Actividad 1. Diseño de la arquitectura de monitoreo del sistema IoT

El propósito de esta actividad es modelar de forma visual la arquitectura multicapa de un sistema IoT, la cual incluya principalmente: i) las entidades IoT (físicas y digitales) y sus propiedades a ser monitoreadas; ii) las entidades digitales (nodos de computación y sus recursos de hardware / software) encargadas de soportar los procesos de monitoreo (recolección, transporte, procesamiento y almacenamiento de datos), a nivel de las capas Edge, Fog y Cloud; y, iii) los flujos de datos entre las entidades digitales, mediante el empleo de protocolos de comunicación síncronos o asíncronos [56].

Para alcanzar este objetivo se requiere como insumo de entrada la *especificación de requerimientos funcionales y no funcionales* del sistema IoT. En este sentido, a partir de las necesidades de los stakeholders, debe ejecutarse un proceso previo de elicitación, priorización y documentación de los requerimientos del sistema IoT, el cual no está contemplado dentro del alcance de este trabajo de investigación, sino más bien consiste en un requisito para iniciar esta primera actividad.

Por otro lado, en esta actividad participa el *Arquitecto de Software*, quien con el apoyo de la herramienta *DSL Monitor-IoT* (ver Sección 5.2) debe completar de manera secuencial las siguientes tareas: i) Identificación de subsistemas, entidades, propiedades y flujos de datos; ii) Modelado de subsistemas, entidades y propiedades; y, iii) Modelado de flujos de datos. A su vez, cada tarea se divide en varias subtarear específicas [56]. Si bien, el *Arquitecto de Software* es el actor principal de esta actividad, se necesita el apoyo de dos roles adicionales: un *Técnico en infraestructura, redes y bases de datos*; y, un *Programador*. El primer rol apoya en la configuración de las entidades y parámetros del modelo de arquitectura, relacionados con las redes de comunicación, nodos de computación y sus recursos (p. ej., servidores de aplicaciones, bases de datos, brokers), a nivel de las capas Edge, Fog y Cloud. Mientras que, el segundo rol es el encargado de programar y encapsular en el modelo, las instrucciones de las APIs requeridas para obtener los datos empíricos desde un sensor u otra fuente.

Como resultado de esta actividad, se obtiene el modelo de arquitectura multicapa de un sistema IoT que soporte sus procesos de monitoreo. Este modelo se encuentra alineado con el estándar ISO/IEC 30141:2018 [2] y está serializado en formato XMI.

La Figura 4-2 presenta los recursos, herramientas y artefactos de entrada - salida de esta actividad; mientras que, la Figura 4-3 muestra un diagrama de proceso con el detalle de las tareas y subtarear que deben ejecutarse para completar esta actividad, así como los artefactos que se generan a lo largo de todo el proceso.

Tarea 1.1. Identificación de subsistemas, entidades, propiedades y flujos de datos

Esta tarea recibe como insumo la especificación de requerimientos funcionales y no funcionales de un sistema IoT con el propósito de analizarla y preparar la documentación requerida por el DSL Monitor-IoT para completar las tareas posteriores de modelado de la arquitectura de monitoreo. Por lo tanto, para estandarizar la preparación de esta documentación, se han propuesto un conjunto de plantillas (ver Tablas de la 6-3 a la 6-19)

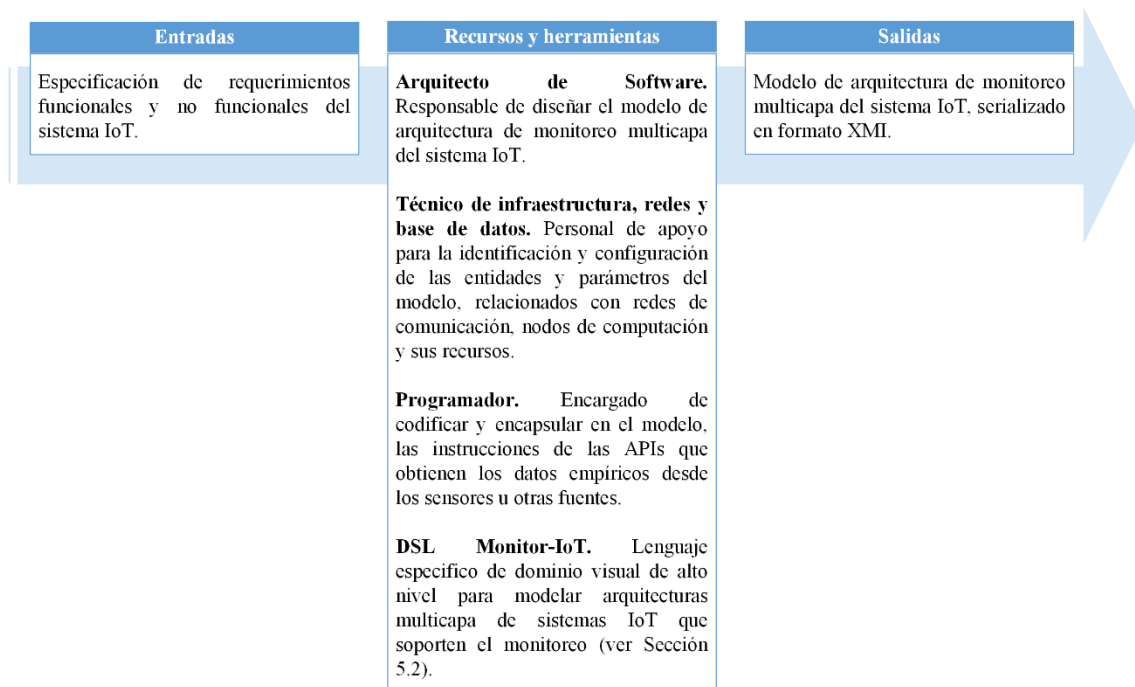


Figura 4-2. Diseño de la arquitectura de monitoreo del sistema IoT: Recursos, herramientas y artefactos de entrada - salida. Fuente: Elaboración propia.

que permiten identificar, clasificar y organizar adecuadamente los componentes y atributos a incluir en el modelo de arquitectura de monitoreo. Un ejemplo sobre la utilización de estas plantillas en un estudio de caso real se presenta en la subsección 6.3.1. Por último, aunque es recomendable que esta tarea se realice siempre para todo tipo de proyecto de construcción de sistemas IoT, puede ser opcional en aquellos proyectos, donde el equipo de desarrollo tiene un claro conocimiento sobre las entidades IoT, propiedades y flujos de datos a modelar como parte de la arquitectura de monitoreo del sistema IoT, pudiendo pasar directamente a las tareas de modelado de esta actividad. Esta tarea se divide en cuatro subtareas:

Subtarea 1.1.1. Descomposición del sistema IoT

Define la estructura jerárquica del sistema IoT, descomponiéndolo en subsistemas en caso de existirlos, a fin de promover la creación de arquitecturas de monitoreo que garanticen una alta cohesión y bajo acoplamiento entre sus partes; y de esta forma, potenciar la reutilización de las capacidades computacionales y físicas de todo o parte del sistema IoT, en otras aplicaciones o escenarios. Los datos a documentar para cada sistema o subsistema son: sigla (identificador), nombre, descripción y dominio de aplicación al que pertenece.

Subtarea 1.1.2. Identificación de entidades

Define las entidades físicas y/o digitales a monitorear. Además, determina las entidades digitales (nodos de computación y dispositivos IoT), sus recursos de hardware / software (p. ej., interfaces de red, middleware, servicios, APIs, aplicaciones, base de datos, tópicos de broker) y las redes de comunicación que son necesarios para soportar los procesos de monitorización, a nivel de las capas Cloud, Fog y Edge. Las entidades son organizadas y agrupadas en subsistemas, en caso de existir estos últimos. Los principales datos a documentar por entidad son: nombre, descripción, metaclass y subsistema al que pertenece.

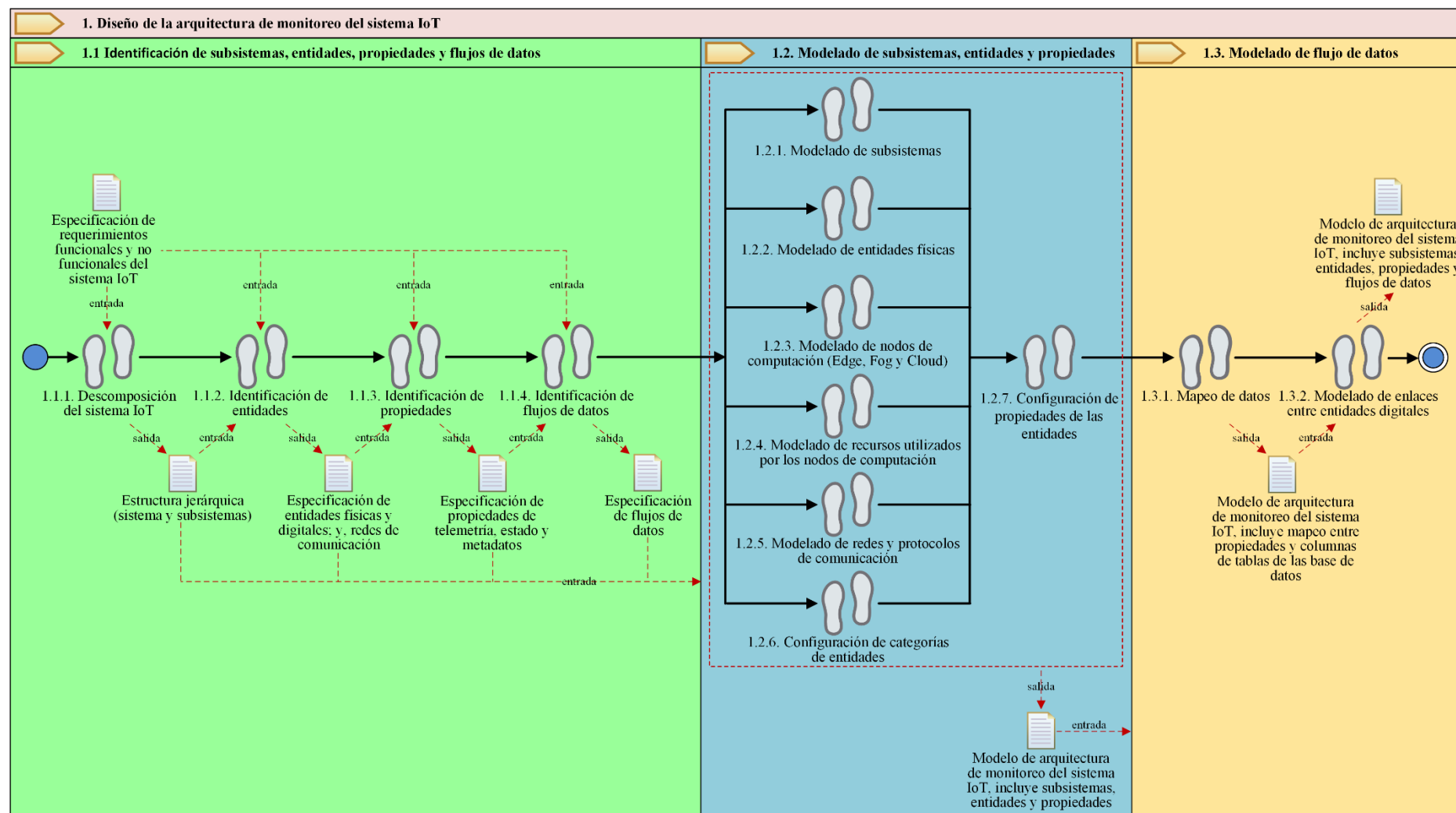


Figura 4-3. Diagrama de proceso para diseñar arquitecturas de monitoreo multicapa para sistemas IoT [56].

Fuente: Elaboración propia.

Subtarea 1.1.3. Identificación de propiedades

Describe las propiedades a ser monitoreadas o utilizadas como metadatos para cada entidad identificada. Los datos a especificar son: nombre y definición de la propiedad, tipo de propiedad (telemetría, estado, metadato), tipo de dato y unidad de medida de la propiedad, si la propiedad identifica de forma única a la entidad; y, si el valor de la propiedad se asigna en tiempo de diseño o ejecución.

Subtarea 1.1.4. Identificación de flujos de datos

En esta subtarea se especifican los flujos de datos, los cuales describen la forma como las entidades digitales interactúan entre sí para soportar los procesos de recolección, transporte, procesamiento y almacenamiento de datos. Los datos a documentar son: descripción y tipo de flujo de datos (recolección o agregación), tipo de comunicación que utiliza el flujo de datos (síncrona o asíncrona); y, frecuencia y hora de ejecución del flujo de datos.

Tarea 1.2. Modelado de subsistemas, entidades y propiedades

Primera tarea de modelado, la cual se ejecuta con el apoyo del DSL Monitor-IoT. En términos generales, a partir de la documentación generada en la tarea anterior, tiene como propósito modelar la estructura jerárquica entre el sistema y sus subsistemas, diseñar la estructura entre las entidades físicas y digitales (nodos de computación y sus recursos) y configurar las propiedades a monitorear por cada entidad. Esta tarea se divide en siete subtareas, varias de las cuales se ejecutan en paralelo:

Subtarea 1.2.1. Modelado de subsistemas

Diseña la estructura jerárquica entre el sistema IoT y sus subsistemas, en caso de existirlos. A su vez, agrupa las entidades en subsistemas, pudiendo una entidad física o digital ser reutilizada por varios subsistemas.

Subtarea 1.2.2. Modelado de entidades físicas

Diseña la estructura jerárquica entre las entidades físicas identificadas en la *Tarea 1.1*.

Subtarea 1.2.3. Modelado de nodos de computación a nivel Edge, Fog y Cloud

Agrega al modelo los nodos Cloud, nodos Fog, puertas de enlace y/o dispositivos IoT (sensores, tags, actuadores) especificados en la *Tarea 1.1*, y que son necesarios para soportar los procesos de monitoreo del sistema IoT. Los nodos de computación pueden estar contenidos dentro de las entidades físicas para dotarles de capacidades de detección, procesamiento, almacenamiento o actuación.

Subtarea 1.2.4. Modelado de recursos utilizados por los nodos de computación

Incorpora al modelo los recursos de hardware y software requeridos por los nodos de computación y que fueron especificados en la *Tarea 1.1*.

Subtarea 1.2.5. Modelado de redes y protocolos de comunicación

Agrega al modelo las redes y protocolos que soportan la comunicación síncrona o asíncrona entre las entidades digitales del sistema IoT. Además, asigna y configura las redes y protocolos a utilizar por cada uno de los recursos de software (p. ej., middlewares, bases de datos).

Subtarea 1.2.6. Configuración de categorías de entidades

Agrupar en categorías, aquellas entidades del mismo tipo que comparten propiedades comunes a ser monitoreadas o utilizadas como metadatos.

Subtarea 1.2.7. Configuración de propiedades de entidades

En esta subtarea se crean las propiedades a monitorear o ser utilizadas como metadatos, junto con sus plantillas que contienen las especificaciones generales de las propiedades. La información contenida en las plantillas puede ser reutilizada posteriormente para la configuración de propiedades comunes de otras entidades.

Tarea 1.3. Modelado de flujos de datos

Una vez incorporado al modelo las entidades físicas y digitales, en esta última tarea se diseña la manera como interactúan las entidades digitales para soportar los flujos de datos a través de las capas Edge, Fog y Cloud. Para alcanzar este objetivo, la tarea está conformada por dos subtareas:

Subtarea 1.3.1. Mapeo de datos

En esta subtarea se definen las reglas de mapeo, es decir, para cada flujo de datos se mapean las fuentes de datos (origen) con las ubicaciones de almacenamiento de los datos (destino).

Subtarea 1.3.2. Modelado de enlaces entre entidades digitales

Para cada flujo de datos, se diseña una secuencia ordenada de enlaces entre las entidades digitales. Esta secuencia determina la trayectoria de los datos y la forma como interactúan los nodos de computación y los recursos de software que intervienen en el flujo de datos, con el objetivo de transportar los datos, desde la fuente hasta su ubicación de almacenamiento lógico.

Actividad 2. Configuración de las capacidades de autoconsciencia del sistema IoT

El objetivo de esta actividad es construir el modelo de autoconsciencia de un sistema IoT, el cual incluya especificaciones sobre: i) el dominio de autoconsciencia, es decir, el sujeto (sistema IoT) que es consciente y los objetos (entidades IoT físicas y digitales) de los que es consciente un sistema IoT; ii) los aspectos a capturar sobre los objetos de autoconsciencia (p. ej., estado, evolución); iii) los procesos de autoconsciencia pre-reflexivos y reflexivos, con sus respectivos métodos de recolección, cálculo y modelos de análisis que determinan la forma como un sistema IoT ejecuta sus capacidades de autoconsciencia; iv) las métricas (directas, indirectas e indicadores) para evaluar los aspectos de autoconsciencia; v) los recursos de implementación (fórmulas, funciones o servicios web) que operacionalizan los métodos de autoconsciencia para calcular las métricas; y, vi) los criterios de decisión, umbrales y recomendaciones prescriptivas asociados a los modelos de análisis.

En esta actividad, participa principalmente el *experto del dominio de IoT*, quien utiliza un *configurador web de capacidades de autoconsciencia* (ver Sección 5.3) para especificar con un alto grado de abstracción los requerimientos de autoconsciencia del sistema IoT; es decir, estos requerimientos son expresados mediante conceptos abstractos del dominio de la autoconsciencia computacional (sujetos, objetos, aspectos, procesos y métodos de autoconsciencia) [20]; y, complementados con conceptos sobre medición de la calidad de software [106]. De igual manera que en la actividad anterior, se reduce notablemente las

tareas de programación a bajo nivel de los requerimientos de autoconsciencia. En este sentido, se requiere únicamente el apoyo de un *Programador* para codificar los algoritmos a encapsular en los recursos de implementación (funciones, servicios web) asociados tanto a los métodos de cálculo que producen las métricas indirectas mediante modelos reflexivos (modelos estadísticos, de minería de datos, aprendizaje automático, etc.), como a los modelos de análisis que obtienen los indicadores cuantitativos.

A su vez, los participantes del proyecto con el apoyo del configurador generan un modelo de autoconsciencia formal y serializado en formato JSON que puede ser interpretado y procesado por un computador. De esta forma, un sistema IoT puede operacionalizar autónomamente las especificaciones contenidas en este modelo; y, conocer su estado y posible evolución futura.

Para construir el modelo de autoconsciencia de un sistema IoT se requieren como insumos de entrada la especificación de requerimientos funcionales y no funcionales del sistema IoT, así como su modelo de arquitectura de monitoreo multicapa en formato XMI, obtenido de la Actividad 1.

Las tareas que contempla esta actividad son cuatro: i) Configuración del dominio de autoconsciencia; ii) Definición de los recursos de soporte a los procesos de autoconsciencia; iii) Definición de los procesos y métodos de autoconsciencia (aprendizaje y razonamiento); y; iv) Generación del modelo de autoconsciencia del sistema IoT. De la misma manera, cada tarea se divide en sus subtarear específicas. Cabe recordar que, la aplicación de estas tareas y subtarear en un estudio de caso real se ejemplifica en el Capítulo 7.

La Figura 4-4 presenta los recursos, herramientas y artefactos de entrada - salida de esta actividad; mientras que, la Figura 4-5 muestra un diagrama de proceso con el detalle de las tareas y subtarear que deben ejecutarse para completar esta actividad, así como los artefactos que se generan durante el proceso.

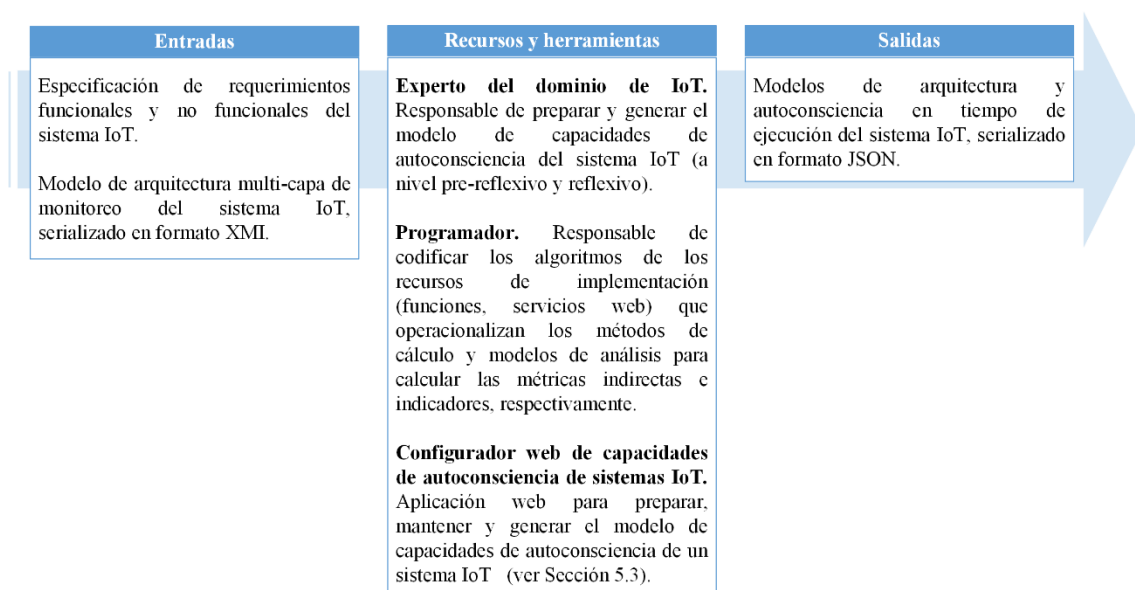


Figura 4-4. Configuración de las capacidades de autoconsciencia del sistema IoT: Recursos, herramientas y artefactos de entrada - salida. Fuente: Elaboración propia.

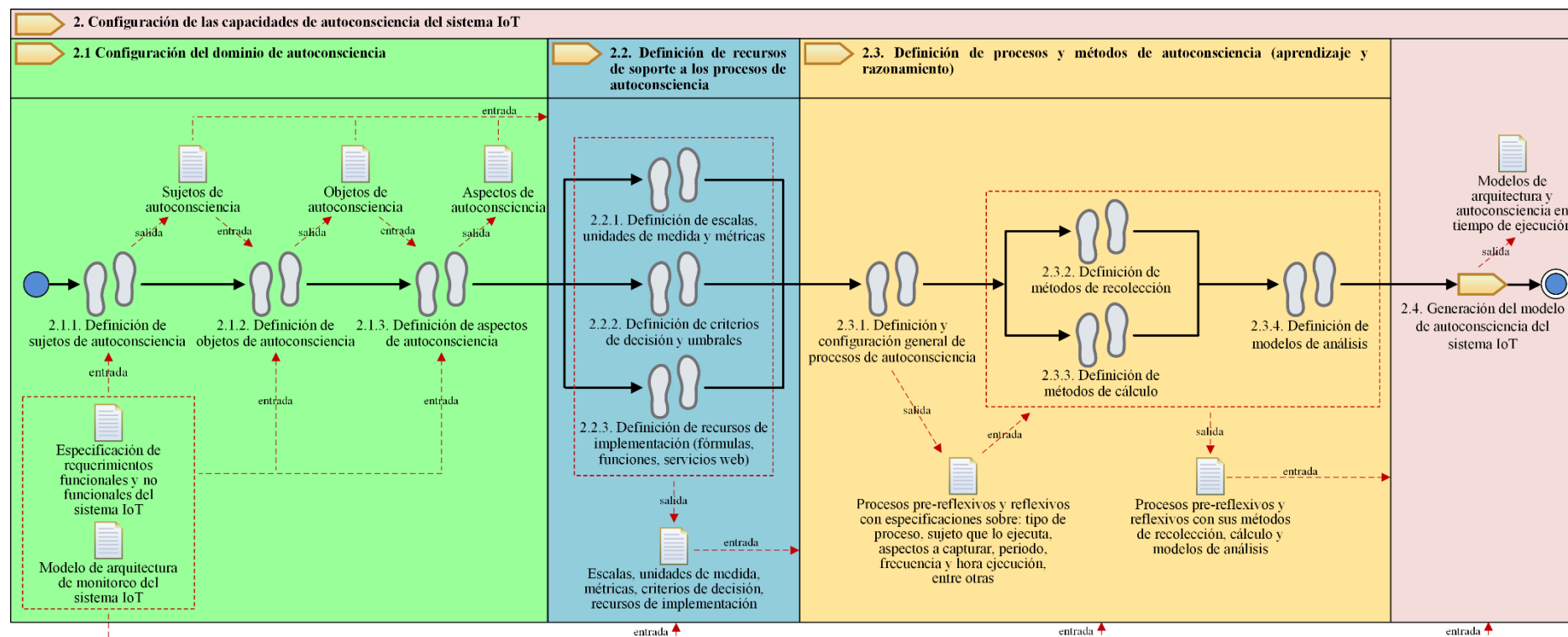


Figura 4-5. Diagrama de proceso para configurar las capacidades de autoconsciencia de sistemas IoT.
Fuente: Elaboración propia.

Tarea 2.1. Configuración del dominio de autoconsciencia

Contempla la definición de los sujetos, objetos y aspectos de autoconsciencia, por lo que se divide en tres subtareas:

Subtarea 2.1.1. Definición de sujetos de autoconsciencia

En esta subtask se seleccionan como sujetos de autoconsciencia al sistema IoT en general o a cualquiera de sus subsistemas, los cuales fueron definidos previamente en el modelo de arquitectura.

Subtarea 2.1.2. Definición de objetos de autoconsciencia

De igual manera, se seleccionan como objetos de autoconsciencia a cualquiera de las entidades IoT, sea física o digital, contenidas en el modelo de arquitectura.

Subtarea 2.1.3. Definición de aspectos de autoconsciencia

Especifica los aspectos de autoconsciencia sobre los cuales el sistema IoT reflexionará y será consciente. Un aspecto de autoconsciencia puede ser: *individual*, si contempla un único objeto y propiedad a evaluar, aislado de su entorno; y, *colectivo*, si abarca un conjunto de objetos o propiedades a evaluar de forma global. Por lo tanto, los aspectos de autoconsciencia se organizan jerárquicamente en una estructura de árbol, donde los aspectos individuales se encuentran a nivel de nodos hoja; mientras que, los aspectos colectivos pueden derivarse de aspectos individuales u otros colectivos de más bajo nivel. En este sentido, la metodología Aware-IoT@Run.Time utiliza el Modelo de Agregación Lógica de Preferencias (Logic Scoring of Preference – LSP) [170] para evaluar los aspectos colectivos, a partir de los indicadores obtenidos para los aspectos de más bajo nivel (estrategia botton – up). La forma como se aplica el modelo LSP en los procesos de autoconsciencia de un sistema IoT se explica más adelante. La información general a registrar en el configurador web por cada aspecto es la siguiente: nombre, descripción, tipo (p. ej. estado, predicción), nivel de importancia (peso) y sujeto que captura el aspecto. Adicionalmente, para el caso de los aspectos individuales se debe seleccionar la categoría de objeto (entidad física o digital) y el objeto sobre el que se captura dicho aspecto. Mientras que, para el caso de los aspectos colectivos se debe seleccionar los aspectos individuales u otros colectivos de más bajo nivel de los cuales se derivan, así como el operador de agregación del Modelo LSP que agrupa a estos aspectos de más bajo nivel.

Tarea 2.2. Definición de recursos de soporte a los procesos de autoconsciencia

Establece los mecanismos o recursos de soporte para la implementación y operacionalización de los procesos de autoconsciencia; y, sus métodos de aprendizaje y razonamiento. Estos recursos tienen un alcance general, por lo que pueden ser compartidos entre varios procesos y métodos de autoconsciencia. Los recursos a registrar son: escalas, unidades de medida, métricas, criterios de decisión, umbrales y recursos de implementación. Esta tarea se divide en tres subtareas:

Subtarea 2.2.1. Definición de escalas, unidades de medida y métricas

Especifica las métricas que serán utilizadas por los métodos de recolección, cálculo y modelo de análisis para evaluar los aspectos de autoconsciencia definidos previamente. Los datos a registrar por cada métrica son: nombre, descripción, abreviatura, tipo (directa, indirecta o indicador), escala y unidad de medida. Para el caso de la escala (conjunto de valores que puede tomar la métrica) se debe registrar: nombre, tipo de escala (ordinal,

nominal, intervalo, ratio) y valores válidos; mientras que, para la unidad de medida (cantidad estandarizada usada para expresar la magnitud de una métrica) se debe proporcionar: nombre, descripción y acrónimo.

Subtarea 2.2.2. Definición de criterios de decisión y umbrales

Establece los criterios de decisión y sus umbrales, los cuales serán utilizados por los modelos de análisis para transformar un indicador cuantitativo, de un aspecto de autoconsciencia, en una valoración cualitativa sobre el estado o la evolución del sistema IoT. Además, estos criterios están asociados con las recomendaciones prescriptivas, a fin de sugerir acciones correctivas para cada umbral, en caso de ser necesario. Los datos a proporcionar por cada criterio de decisión son: nombre y descripción. A su vez, un criterio de decisión puede tener dos o más umbrales, debiendo ingresar los siguientes datos por cada umbral: nombre, interpretación cualitativa, límites inferior y superior.

Subtarea 2.2.3. Definición de recursos de implementación (fórmulas, funciones, servicios web)

Define las fórmulas, funciones o servicios web que encapsulan las operaciones o algoritmos para operacionalizar los métodos de cálculo o modelos de análisis y obtener las métricas (indirectas, indicadores) respectivas. Los recursos de implementación pueden ser reutilizados por varios procesos y métodos de autoconsciencia. Para registrar las fórmulas se debe proporcionar los siguientes datos: nombre, descripción, expresión matemática, variables dependientes (nombre y tipo de dato) y tipo de dato de salida. Las fórmulas pueden hacer uso de funciones de agregación (p. ej., suma, cuenta, promedio). A su vez, para las funciones se debe especificar: nombre, descripción, instrucciones o código fuente a encapsular, parámetros de entrada (nombre, tipo de dato, si es requerido u opcional) y tipo de dato de salida. Finalmente, para los servicios web se debe ingresar: nombre, descripción, instrucciones o código fuente a encapsular, nodo de computación, servidor de aplicaciones en el que se publica, Endpoint, parámetros de entrada (nombre, tipo de dato, si es requerido u opcional), tipo de dato y formato (p. ej., JSON, XML) de salida.

Tarea 2.3. Definición de procesos y métodos de autoconsciencia (aprendizaje y razonamiento)

En esta tarea se especifican los procesos pre-reflexivos y reflexivos que serán ejecutados por los sujetos y que tienen como propósito capturar el conocimiento (evaluar métricas) sobre los aspectos de autoconsciencia de un sistema IoT. Los procesos pre-reflexivos desarrollan en el sujeto una consciencia subjetiva basada únicamente en información sensorial, es decir, datos recolectados directamente desde los sensores u otras fuentes (observación inmediata), a fin de crear métricas directas e indicadores; por lo que, hacen uso de los métodos de recolección y modelos de análisis. Por su parte, los procesos reflexivos construyen y/o aplican en el sujeto modelos internos conceptuales o reflexivos, a partir de la información histórica de las métricas (directas, indirectas) generada por otros procesos pre-reflexivos o reflexivos, con el propósito de obtener nuevas métricas indirectas e indicadores; por lo tanto, hacen uso de los métodos de cálculo y modelos de análisis. Para alcanzar este objetivo, la tarea se divide en cuatro subtareas:

Subtarea 2.3.1. Definición y configuración general de procesos de autoconsciencia

Para cada proceso de autoconsciencia tanto pre-reflexivo como reflexivo se especifica la siguiente información: i) id y descripción del proceso; ii) periodo de tiempo durante el

cual se ejecuta el proceso; ii) frecuencia u hora de ejecución del proceso; iii) sujeto y nodo de computación que ejecuta el proceso, posibilitando la implementación de una autoconsciencia computacional distribuida, a nivel de las capas Edge, Fog y Cloud; iv) aspecto de autoconsciencia a capturar; y, v) objetos sobre los que se captura el aspecto de autoconsciencia. Únicamente, los procesos reflexivos pueden asociarse con aspectos colectivos. En las siguientes subtareas, se configuran los métodos de autoconsciencia dependiendo del tipo de proceso.

Subtarea 2.3.2. Definición de métodos de recolección

Para cada proceso pre-reflexivo se configura un método encargado de recopilar desde un sensor u otra fuente, los datos crudos o empíricos de una propiedad determinada de un objeto de autoconsciencia (entidad IoT), produciendo como resultado una métrica directa. Para la configuración de este tipo de método se debe especificar la siguiente información: tipo de comunicación (síncrona o asíncrona), propiedad a recolectar y flujo de datos a utilizar; debiendo indicar que estos datos son obtenidos del modelo de arquitectura del sistema IoT. Además, se debe definir la métrica directa a obtener.

Subtarea 2.3.3. Definición de métodos de cálculo

Para cada proceso reflexivo se configura un método responsable de construir y/o aplicar un modelo conceptual o reflexivo (p. ej., modelos de regresión estadística, aprendizaje automático) que, a partir de los datos recopilados históricamente, obtenga como resultado una métrica indirecta. Para la parametrización de este tipo de método se debe proporcionar la siguiente información: periodo de tiempo de los datos históricos a utilizar en el cálculo, tipo (fórmula, función o servicio web) y nombre del recurso de implementación que operacionaliza el método. En caso de requerir configurar un método de cálculo predictivo se debe especificar los escenarios, variables y valores de simulación a utilizar. Además, se deben mapear los parámetros requeridos por el recurso de implementación con las métricas, metadatos o variables de simulación, estos últimos proveen los datos de entrada necesarios para la ejecución de las operaciones de cálculo. Finalmente, se debe definir la métrica indirecta a producir.

Subtarea 2.3.4. Definición de modelos de análisis

Para cada proceso pre-reflexivo o reflexivo se configura un modelo que relaciona una o más métricas previamente calculadas (directas, indirectas o indicadores) para producir como resultado un indicador cuantitativo porcentual. A su vez, como parte del modelo de análisis, el indicador cuantitativo es comparado con un criterio de decisión (compuesto por umbrales), a fin de obtener un indicador cualitativo que determine el estado o la posible evolución de un objeto de autoconsciencia de un sistema IoT, así como las correspondientes recomendaciones prescriptivas, en caso de ser necesario. Para la parametrización de este tipo de método se debe proporcionar la siguiente información: tipo (fórmula, función o servicio web) y nombre del recurso de implementación que operacionaliza el modelo. Asimismo, se debe mapear los parámetros requeridos por el recurso de implementación con las métricas o metadatos que proveen los datos de entrada necesarios para la ejecución de las instrucciones del modelo. Además, se debe definir el criterio de decisión a utilizar, especificar las recomendaciones prescriptivas asociadas a cada umbral del criterio de decisión seleccionado; y, establecer el indicador a producir. Finalmente, en este punto se debe indicar que los procesos reflexivos sobre aspectos de autoconsciencia colectivos utilizan únicamente modelos de análisis, es decir, no requieren aplicar previamente métodos de cálculo. Esto se debe a que los indicadores de este tipo de aspectos son calculados mediante el modelo LSP [170], aplicando directamente

operaciones de agregación ponderadas sobre los indicadores de los aspectos asociados de más bajo nivel.

Tarea 2.4. Generación del modelo de autoconsciencia del sistema IoT

El configurador web incluye de manera automática en el modelo de arquitectura de monitoreo del sistema IoT, las especificaciones de autoconsciencia configuradas en las tareas previas. La integración de estas especificaciones al modelo de arquitectura se realiza conforme al vocabulario y la sintaxis establecida en los metamodelos de arquitectura (Monitor-IoT) y autoconsciencia (Aware-IoT). Como resultado se genera un único archivo en formato JSON conteniendo los modelos de arquitectura y autoconsciencia en tiempo de ejecución del sistema IoT, el cual será interpretado y procesado en las actividades subsiguientes de la metodología (sincronización, aprendizaje y razonamiento), a fin de implementar y ejecutar los recursos de software que soportan los procesos de monitoreo y autoconsciencia del sistema IoT.

Actividad 3. Sincronización entre los modelos de arquitectura - autoconsciencia y el sistema IoT observado (mantiene la relación causal)

A partir de las especificaciones de los modelos de arquitectura - autoconsciencia y su evolución o cambios en el tiempo, esta actividad implementa y/o reconfigura de forma automática y distribuida los recursos de software que soportan la infraestructura de monitoreo y autoconsciencia de un sistema IoT, sin interrumpir su operación.

Los recursos de software de monitoreo de un sistema IoT hacen referencia a las APIs, aplicaciones, servicios web, brokers, tópicos y bases de datos que soportan la recolección, transporte, procesamiento y almacenamiento de los datos crudos de los sensores u otras fuentes, a nivel de las capas Edge, Fog y Cloud. A continuación, se describen cada uno de estos recursos de software [56]:

- Las APIs encapsulan instrucciones para obtener los datos crudos desde los sensores u otras fuentes.
- Las aplicaciones son artefactos de software que operan como intermediarios y encapsulan instrucciones para enrutar y controlar los flujos de datos entre las APIs, servicios web y brokers, así como para establecer con qué frecuencia se ejecutan automáticamente estos flujos de datos. En particular, las aplicaciones de software pueden controlar los flujos de recolección o agregación de datos. Una aplicación que controla un flujo de recolección de datos, primero incluye instrucciones de llamada a una API para obtener los datos crudos desde un sensor u otra fuente, luego dependiendo del tipo de comunicación: i) *síncrona*, contiene instrucciones para consumir un servicio web y enviar directamente los datos recopilados a un nodo de computación para su almacenamiento inmediato en una base de datos; o, ii) *asíncrona*, contienen instrucciones para publicar temporalmente los datos recolectados en un tópico de un Broker (comunicación asíncrona), de tal forma que estén disponibles para su almacenamiento posterior en una base de datos. A su vez, una aplicación que controla un flujo de agregación de datos incluye instrucciones para consumir periódicamente uno o varios servicios web, los cuales recuperan los datos crudos de monitoreo desde una tabla fuente, luego aplican operaciones de agregación sobre estos datos; y, finalmente, almacenan los datos resumidos en una tabla destino. Como ejemplo ilustrativo de los dos tipos de flujos de datos. En primer lugar, un flujo de recolección de datos para monitorear la temperatura de una habitación puede ser implementado mediante una aplicación que se ejecuta cada hora en una puerta de

enlace. Esta aplicación llama a una API para obtener los datos crudos desde un sensor de temperatura y luego consume un servicio web para almacenar dichos datos en una base de datos de un nodo Fog. Como complemento, un flujo de agregación de datos para obtener el promedio diario de la temperatura de la habitación puede ser implementado mediante otra aplicación que se ejecuta una vez al día en el nodo Fog. Esta aplicación consume un servicio web que recupera todos los datos de temperatura recolectados diariamente, luego calcula el promedio de estos datos; y, el resultado lo almacena en una base de datos ubicada en un nodo Cloud.

- Los servicios web facilitan el intercambio de datos entre los nodos de un sistema IoT, a nivel de las capas Edge, Fog y Cloud. A su vez, estos servicios web encapsulan el código necesario para recuperar o almacenar los datos de monitoreo (con diferentes niveles de agregación) y sus metadatos desde o en las bases de datos del sistema IoT.
- Los brokers y sus tópicos también facilitan el intercambio de datos entre los nodos de una infraestructura IoT; sin embargo, este intercambio se realiza de forma asíncrona. Como se mencionó, las aplicaciones pueden publicar temporalmente los datos recolectados en los tópicos de un broker. A su vez, el broker está asociado a un evento que se dispara cada vez que se publican nuevos datos en los tópicos. Por su parte, este evento consume servicios web para almacenar los datos de los tópicos en las bases de datos correspondientes, según lo establecido en el modelo de arquitectura.
- Las bases de datos pueden estar distribuidas en los diferentes nodos de computación de la infraestructura IoT. Además, estas bases de datos almacenan tanto los datos monitoreados (con diferentes niveles de agregación) como sus metadatos. Por lo tanto, esta actividad de sincronización se encarga de crear y/o modificar la estructura lógica de almacenamiento, conformada por las bases de datos, tablas y columnas requeridas para almacenar los propios datos de monitoreo y sus metadatos.

Por otro lado, los recursos de software de autoconsciencia de un sistema IoT están conformados por los demonios y los recursos de implementación, tales como: fórmulas, funciones y servicios web. Los demonios pueden residir de manera distribuida en los diferentes nodos de computación de una infraestructura IoT, a nivel de las capas Edge, Fog y Cloud. A su vez, estos demonios se ejecutan en segundo plano cada cierto intervalo de tiempo y encapsulan la lógica tanto de los procesos pre-reflexivos como reflexivos; de ahí que, dependiendo del tipo de proceso contienen diferentes instrucciones, mismas que se detallan a continuación:

- Un demonio para un proceso pre-reflexivo operacionaliza un método de recolección y un modelo de análisis. Por lo tanto, primero incluye instrucciones para obtener una métrica directa, a partir de los datos crudos o empíricos recolectados por los flujos de datos y recursos de monitoreo instanciados a través del modelo de arquitectura. Luego, contiene instrucciones de llamada a un recurso de implementación para transformar la métrica directa en un indicador cuantitativo. Finalmente, encapsula instrucciones para comparar el indicador cuantitativo con un criterio de decisión y sus umbrales, a fin de generar un indicador cualitativo y determinar las recomendaciones prescriptivas (acciones correctivas) que el caso lo amerite.
- Un demonio para un proceso reflexivo operacionaliza un método de cálculo y un modelo de análisis. En este sentido, primero encapsula instrucciones de llamada a un recurso de implementación para producir una métrica indirecta, utilizando los datos históricos de otras métricas (directas, indirectas, indicadores) o metadatos. También,

las instrucciones pueden admitir variables de simulación para obtener métricas indirectas que ayuden a predecir posibles escenarios futuros. Luego, al igual que un demonio pre-reflexivo, incluye instrucciones para transformar la métrica indirecta en indicadores cuantitativo y cualitativo, así como para determinar recomendaciones para llegar a un estado de operación deseado del sistema IoT, en caso de requerirlo.

- En el caso particular de un proceso reflexivo para un aspecto colectivo, el demonio asociado únicamente debe operacionalizar un modelo de análisis. En primer lugar, este demonio incluye instrucciones de llamada a un método disponible en la infraestructura de soporte a la metodología Aware-IoT@Run.Time, el cual automatiza el modelo LSP [170] (ver Subsección 5.4.3). Este método recibe como parámetro de entrada, el identificador del aspecto colectivo a calcular; y, devuelve como salida, el indicador porcentual obtenido del modelo LSP para dicho aspecto. Específicamente, el algoritmo LSP encapsulado en el método, aplica un mecanismo de agregación paso a paso, de abajo hacia arriba (estrategia Botton – Up), en donde el indicador del aspecto colectivo en cuestión se calcula mediante operaciones de agregación ponderadas sobre los indicadores de los aspectos asociados de más bajo nivel, pudiendo ser estos individuales o colectivos. Por lo tanto, este algoritmo utiliza las especificaciones contenidas en el modelo de autoconsciencia en tiempo de ejecución relacionadas con la estructura jerárquica de los aspectos, los operadores de agregación establecidos entre estos y los pesos (nivel de importancia) de cada aspecto. Finalmente, una vez que el indicador cuantitativo del aspecto colectivo ha sido devuelto por el método LSP, el demonio también incluye instrucciones para comparar dicho indicador con un criterio de decisión y sus umbrales, con el propósito de obtener un indicador cualitativo y determinar las recomendaciones prescriptivas que sean necesarias.

Con respecto a los recursos de implementación que son llamados o utilizados por los demonios de los procesos de autoconsciencia. Por una parte, esta actividad implementa automáticamente las funciones y los servicios web. Para ello, codifica las cabeceras de los artefactos (nombre, parámetros de entrada, variable de salida) y encapsula en su cuerpo las respectivas instrucciones, según las especificaciones del modelo de autoconsciencia. Por otra parte, la infraestructura de soporte a la metodología dispone de un método que analiza la sintaxis de los recursos de implementación de tipo fórmula, así como también las interpreta y resuelve (ver Subsección 5.4.3). Este método es consumido desde los demonios, los cuales envían la fórmula asociada y los argumentos que contienen los valores a ser reemplazados en las variables (parámetros) de la fórmula para su resolución.

Para la ejecución de esta actividad se utilizan los *motores de sincronización de recursos de monitoreo y autoconsciencia* que forman parte del middleware de la infraestructura de soporte a la metodología (ver Sección 5.4). Estos motores se encargan de detectar e interpretar periódicamente las especificaciones contenidas en la última versión de los modelos de arquitectura y autoconsciencia en tiempo de ejecución. A su vez, con base en estas especificaciones, los motores ejecutan automáticamente tareas de implementación y/o reconfiguración de los artefactos de software de monitoreo y autoconsciencia, encapsulando en estos artefactos las instrucciones descritas en los párrafos anteriores. La Figura 4-6 presenta las herramientas empleadas y los artefactos de entrada – salida de esta actividad.

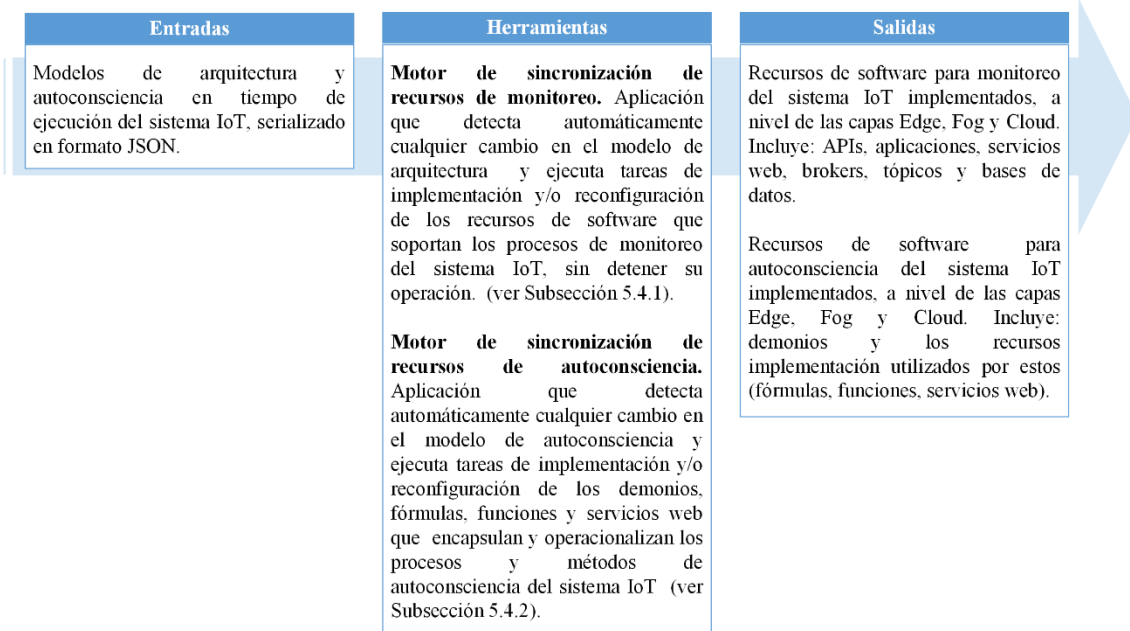


Figura 4-6. Sincronización entre los modelos de arquitectura - autoconsciencia y el sistema IoT observado: Herramientas y artefactos de entrada - salida. Fuente: Elaboración propia.

Actividad 4. Monitoreo

Durante esta actividad, los recursos de software de monitoreo implementados se encuentran en operación, conforme a las especificaciones establecidas en el modelo de arquitectura en tiempo de ejecución. Estos recursos tienen como propósito recolectar, transportar, procesar y almacenar los datos crudos de los sensores u otras fuentes, por medio de comunicaciones síncronas o asíncronas. La Figura 4-7 presenta las herramientas utilizadas y los artefactos de entrada – salida de esta actividad.

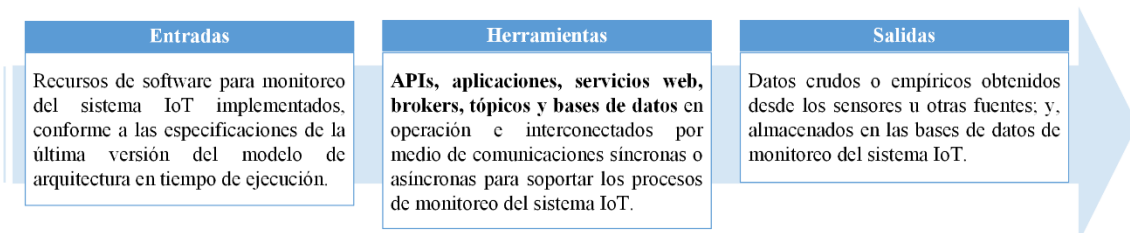


Figura 4-7. Monitoreo: Herramientas y artefactos de entrada - salida. Fuente: Elaboración propia.

Actividad 5. Aprendizaje

Esta actividad tiene como alcance producir las métricas directas e indirectas como resultado de la ejecución de los procesos de autoconsciencia pre-reflexivos y reflexivos, respectivamente. Como se ha mencionado con anterioridad, las métricas directas son obtenidas mediante los métodos de recolección de los procesos pre-reflexivos y representan concretamente los datos crudos o empíricos recolectados desde los sensores u otras fuentes. Mientras que, las métricas indirectas son producidas por los métodos de cálculo de los procesos reflexivos. Estas métricas indirectas se derivan de otras métricas previamente calculadas (directas, indirectas o indicadores), por medio de la construcción y/o aplicación de modelos reflexivos (p. ej., modelos de regresión estadística, minería de datos, aprendizaje automático), especificados en los métodos de cálculo. En este sentido, la actividad de aprendizaje puede conllevar a extraer, refinar y calibrar continuamente los

modelos reflexivos a partir de los datos históricos de las métricas para obtener valoraciones más ajustadas a la realidad.

Por lo tanto, esta actividad requiere como insumos de entrada los datos crudos generados por la actividad de *Monitoreo*, las métricas calculadas previamente y las especificaciones contenidas en los modelos de arquitectura y autoconsciencia en tiempo de ejecución. A su vez, los demonios pre-reflexivos y reflexivos son los encargados de automatizar esta actividad mediante la operacionalización de los métodos de recolección y cálculo, conforme a lo establecido en la última versión del modelo de autoconsciencia en tiempo de ejecución. La Figura 4-8 presenta las herramientas y artefactos de entrada – salida de esta actividad.

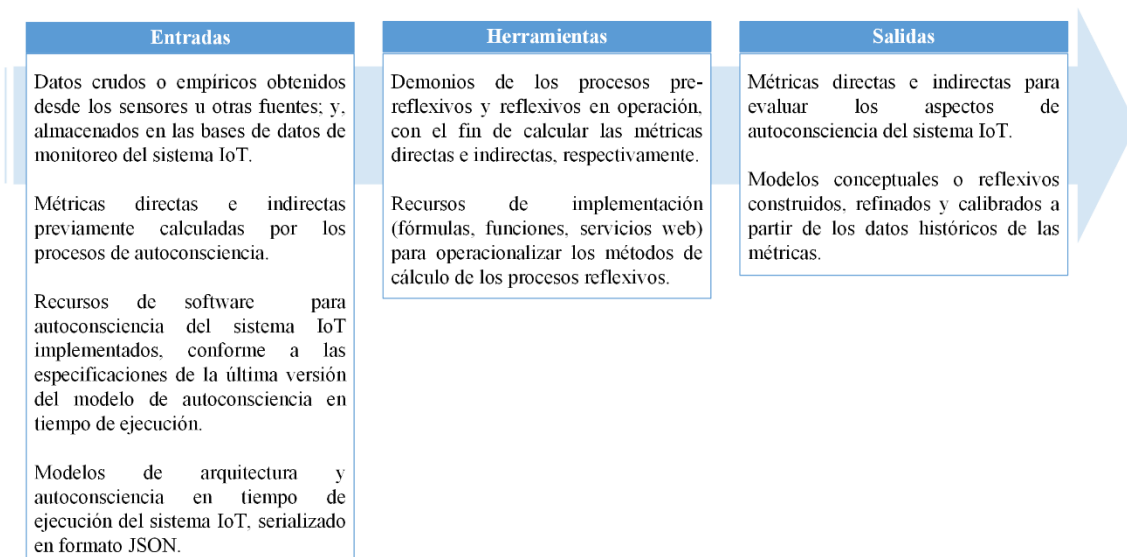


Figura 4-8. Aprendizaje: Herramientas y artefactos de entrada - salida.

Fuente: Elaboración propia.

Actividad 6. Razonamiento

Como complemento a la actividad de *Aprendizaje*, a partir de las métricas directas e indirectas obtenidas previamente, la actividad de *Razonamiento* tiene como objetivo producir los indicadores asociados a cada uno de los procesos pre-reflexivos y reflexivos. Para ello se aplican los modelos de análisis que combinan y comparan una o más métricas previamente calculadas (directas, indirecta o indicadores) con los criterios de decisión (umbrales) asociados a los modelos, a fin de obtener nuevos indicadores cuantitativos (porcentual) y cualitativos que valoren y reflejen el estado o la posible evolución de los objetos de autoconsciencia de un sistema IoT. A su vez, con el apoyo de los indicadores cualitativos se determinan las recomendaciones prescriptivas que deben ser consideradas por el sistema IoT.

Como se mencionó anteriormente, esta actividad requiere como insumos de entrada las métricas directas e indirectas generadas por los procesos de autoconsciencia, así como las especificaciones incluidas en el modelo de autoconsciencia en tiempo de ejecución. De igual manera, esta actividad es automatizada por los demonios de los procesos pre-reflexivos y reflexivos, mediante la operacionalización de los modelos de análisis, conforme a lo establecido en la última versión del modelo de autoconsciencia en tiempo de ejecución. Finalmente, un *visor de autoconsciencia* (aplicación web), a nivel de front-end, presenta reportes en tiempo real sobre el estado y la evolución (predicción) del sistema IoT y su entorno (objetos de autoconsciencia). Estos reportes contienen tablas y

gráficos sobre las métricas directas, indirectas, indicadores (cuantitativos y cualitativos), metadatos (fecha, ID de proceso, etc.) y recomendaciones prescriptivas resultantes (ver Sección 5.5). La Figura 4-9 muestra las herramientas y artefactos de entrada – salida de esta actividad.

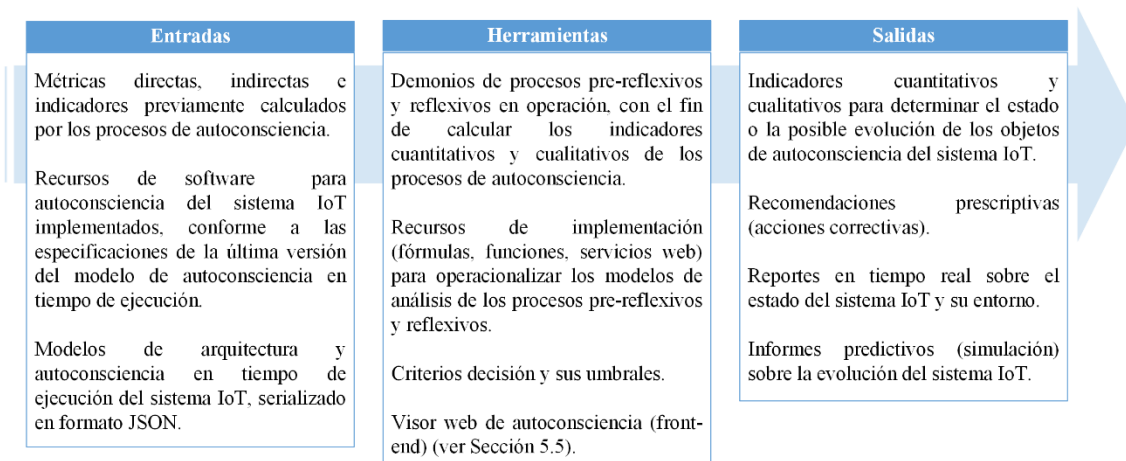


Figura 4-9. Razonamiento: Herramientas y artefactos de entrada - salida.

Fuente: Elaboración propia.

4.3 Conclusiones

En este capítulo, se ha presentado una metodología que utiliza el paradigma de Ingeniería Dirigida por Modelos y, particularmente los modelos en tiempo de ejecución como artefactos centrales del proceso de desarrollo, operación y mantenimiento de un sistema IoT autoconsciente. El enfoque propuesto ha pasado por un proceso de evaluación experimental y mejora continua con el propósito de alcanzar un método formal, ordenado, eficaz, eficiente y repetible, cuyos detalles son tratados en el Capítulo 7.

La metodología ha sido descrita textual y gráficamente, incluyendo la descomposición jerárquica del proceso metodológico en actividades, tareas y subtarear, la especificación de los roles de usuarios que deben participar en las actividades manuales, la relación con las herramientas que soportan el proceso; y, la descripción de los artefactos de entrada y salida de las actividades. En este sentido, la profundidad y completitud con la que se ha detallado la metodología, facilita su entendimiento y aplicación por parte de los usuarios, lo cual fue comprobado durante la evaluación experimental de la misma.

Las dos primeras actividades del proceso metodológico permiten a los desarrolladores diseñar los modelos de arquitectura y autoconsciencia de un sistema IoT, con un alto grado de abstracción, expresividad y flexibilidad, ya que se enfocan en el uso de lenguajes de alto nivel basados en conceptos del dominio del problema (p. ej. entidades físicas y digitales de IoT, flujos de datos, sujetos, objetos, aspectos y procesos de autoconsciencia, métricas de calidad). Al mismo tiempo, este enfoque metodológico minimiza los detalles de implementación (programación a bajo nivel), lo cual reduce la complejidad; y, acelera el desarrollo y mantenimiento de este tipo de sistemas. Por su parte, las subsiguientes actividades de la metodología, elevan el nivel de automatización del proceso de implementación de un sistema IoT, ya que no requieren la participación de los desarrolladores, en su defecto transforman automáticamente las especificaciones de alto nivel de los modelos en artefactos de software que soportan los procesos de monitoreo y autoconsciencia, a lo largo de las capas Edge, Fog y Cloud de un sistema IoT.

Asimismo, el enfoque metodológico es lo suficientemente flexible, dependiendo del

alcance de las especificaciones de los modelos, la operación de los artefactos de software de monitoreo y autoconsciencia puede dotar a un sistema IoT de una diversidad de capacidades de autoconsciencia que van desde lo individual a lo colectivo y de lo pre-reflexivo a lo reflexivo. Estas capacidades también pueden reflexionar sobre una amplia variedad de objetos (entidades físicas y digitales) y aspectos (estado, causalidad, predicción, etc.) de autoconsciencia.

En otro punto, este método facilita la gestión de la mantenibilidad de un sistema IoT, ya que cualquier cambio debe realizarse a nivel de los modelos en tiempo de ejecución. Luego, con el soporte de las herramientas (motores de sincronización del middleware de la infraestructura) se detectan automáticamente estos cambios y se ejecutan las tareas de reconfiguraciones de los artefactos de monitoreo y autoconsciencia, sin necesidad de realizar implementaciones manuales o detener la operación de la infraestructura IoT.

En consecuencia, por todo lo expuesto en el capítulo, este método operativiza la relación causal entre los modelos en tiempo de ejecución y el sistema IoT, y viceversa. Por una parte, ante cualquier cambio en las especificaciones de los modelos se implementan y/o reconfiguran automáticamente a nivel de software, los correspondientes artefactos de monitoreo y autoconsciencia del sistema IoT. Por otra parte, una vez que estos artefactos entran en operación, se encargan de actualizar y calibrar en tiempo de ejecución los parámetros (métricas, metadatos, etc.) especificados en los modelos, logrando de esta manera reflejar a nivel de los modelos, el estado o la probable evolución del sistema IoT. Además, por medio del análisis de los parámetros actualizados de los modelos, el sistema puede llegar a conclusiones sobre qué recomendaciones o acciones correctivas son necesarias para alcanzar un estado deseado. Así pues, este enfoque metodológico también contribuye a elevar la capacidad autónoma y resiliente de los sistemas IoT frente a las situaciones imprevistas.

Por último, los modelos en tiempo de ejecución constituyen una capa de abstracción que actúa como puente entre los requisitos de monitorización y autoconsciencia de un sistema IoT y sus detalles de implementación. De ahí que, esta capa de abstracción permite que el enfoque metodológico propuesto pueda ser aplicado en cualquier dominio o plataforma tecnológica de IoT. Como los modelos en tiempo de ejecución son especificaciones de dominio de alto nivel, la plataforma tecnológica de destino se puede cambiar rápidamente, mediante la implementación de nuevos motores de transformación y sincronización; sin embargo, las especificaciones de los modelos de arquitectura y autoconsciencia permanecen constantes.

Capítulo 5

Infraestructura de Soporte a la Metodología

En este capítulo se presenta la infraestructura basada en modelos en tiempo de ejecución que soporta la metodología *Aware-IoT@Run.Time* descrita en el capítulo anterior. Esta infraestructura posibilita la construcción, operación y mantenimiento de sistemas IoT autoconscientes con un alto grado de abstracción y flexibilidad. En este sentido, las herramientas de la infraestructura permiten generar modelos con especificaciones de monitoreo y autoconsciencia de alto nivel para cualquier dominio de IoT. Asimismo, estas especificaciones pueden ser rápidamente implementadas y desplegadas en cualquier plataforma tecnológica de IoT, únicamente creando, a nivel del middleware de la infraestructura, nuevos algoritmos (motores) de transformación y sincronización de los artefactos de software de monitoreo y autoconsciencia del sistema IoT.

La sección 5.1 presenta la arquitectura general de la infraestructura de soporte y sus componentes. Las subsiguientes secciones describen a profundidad cada uno de los componentes de la infraestructura. De ahí que, en la sección 5.2 se detalla el DSL *Monitor-IoT* para modelar arquitecturas de monitoreo de sistemas IoT. La sección 5.3 expone el configurador web de capacidades de autoconsciencia para sistemas IoT. La sección 5.4 describe el middleware de la infraestructura, conformado por los motores de sincronización de recursos de software de monitoreo y autoconsciencia. La sección 5.5 muestra el visor de autoconsciencia. Por su parte, la sección 5.6 explica el estado actual y las perspectivas futuras de ampliación de la infraestructura de soporte. Finalmente, la sección 5.7 presenta las conclusiones del capítulo.

5.1 Arquitectura general de la infraestructura de soporte

La infraestructura de soporte está conformada por un conjunto de componentes (artefactos o herramientas) que apoyan a los usuarios y técnicos en la ejecución de las actividades y tareas de la metodología *Aware-IoT@Run.Time*. En la Figura 5-1 se presenta la arquitectura de la infraestructura de soporte, incluyendo los componentes y sus interrelaciones. Estos componentes se enumeran a continuación [36]:

- 1) Metamodelo de arquitectura de monitoreo de sistemas IoT (*Monitor-IoT*).
- 2) Metamodelo de capacidades de autoconsciencia de sistemas IoT (*Aware-IoT*).
- 3) Lenguaje específico de dominio (DSL) *Monitor-IoT*.
- 4) Configurador web de capacidades de autoconsciencia para sistemas IoT.
- 5) Middleware de la infraestructura, conformado por los motores de sincronización de recursos de software de monitoreo y autoconsciencia para sistema IoT.
- 6) Visor web de métricas de autoconsciencia.

En un contexto general, las herramientas *DSL Monitor-IoT* y *configurador web de autoconsciencia* apoyan a los usuarios en la realización de las dos primeras actividades de la metodología, las cuales tienen como propósito la construcción de los modelos de arquitectura y autoconsciencia de un sistema IoT, respectivamente. Estas herramientas utilizan la sintaxis y semántica contenida en los *metamodelos Monitor-IoT* y *Aware-IoT*; de ahí que, los modelos de arquitectura y autoconsciencia resultantes son conformes a estos metamodelos. A su vez, el *configurador web de autoconsciencia* utiliza como insumo el modelo de arquitectura creado en el *DSL Monitor-IoT* y dispone de una base de datos para almacenar tanto las configuraciones particulares de un sistema IoT,

relacionadas con su dominio, procesos y métodos de autoconsciencia, como las configuraciones generales sobre las métricas, criterios de decisión y recursos de implementación (fórmulas, funciones, servicios web) que pueden ser reutilizados en la construcción de diversos modelos de autoconsciencia para sistemas IoT (ver Fig. 5-1(a-e)).

Por su parte, los *motores del middleware de la infraestructura* tienen como alcance la automatización de la tercera actividad de la metodología. Por lo tanto, estos motores utilizan los modelos de arquitectura y autoconsciencia en tiempo de ejecución para implementar y/o reconfigurar automáticamente los artefactos y recursos de software que soportan los procesos de monitorización y autoconsciencia del sistema IoT observado, a nivel de las capas Edge, Fog y Cloud. De esta forma, cualquier cambio en los modelos se refleja automáticamente en el sistema IoT en operación, garantizando la relación de causalidad entre estos (ver Fig. 5-1(f-h)).

En particular, los artefactos de software implementados por el *motor de sincronización de recursos de monitoreo* incluyen APIs, aplicaciones, servicios web, tópicos de brokers y bases de datos, los cuales operan de manera interconectada para soportar la recolección, transporte, procesamiento y almacenamiento de los datos crudos de los sensores u otras fuentes. Mientras que, los artefactos de software implementados por el *motor de sincronización de recursos de autoconsciencia* incluyen demonios (aplicaciones que se ejecutan cada cierto intervalo de tiempo) que encapsulan y ejecutan de manera distribuida y en segundo plano los procesos pre-reflexivos y reflexivos, a fin de calcular las métricas para evaluar los aspectos de autoconsciencia. Además, como parte de los artefactos de autoconsciencia se tienen las funciones y servicios web que son llamados o consumidos desde los demonios para operacionalizar los métodos de cálculo y modelos de análisis, así como las bases de datos que almacenan las métricas directas, indirectas e indicadores obtenidos por los procesos de autoconsciencia (ver Fig. 5-1(i)).

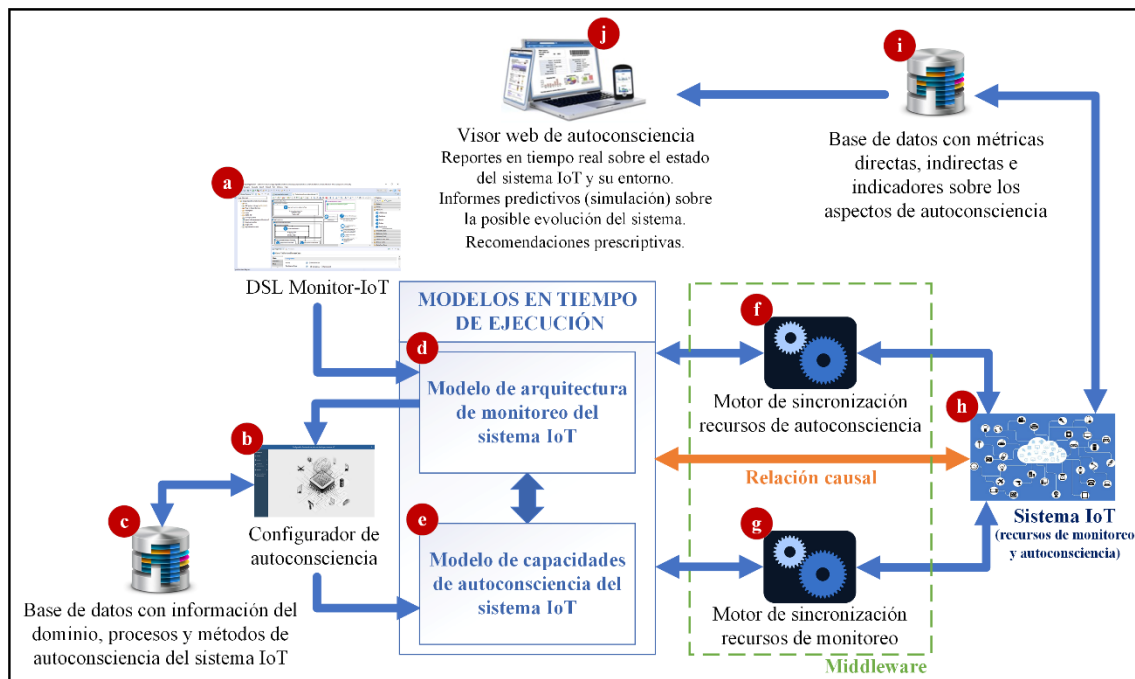


Figura 5-1. Arquitectura general de la infraestructura de soporte basada en models@run.time [36]. Fuente: Elaboración propia.

Por último, la infraestructura de soporte dispone de un *visor web de autoconsciencia*, a nivel de front-end, que presenta reportes basados en tablas y gráficos con información sobre las métricas directas, indirectas, indicadores (cuantitativos y cualitativos), metadatos (fecha, ID de proceso, etc.) y recomendaciones prescriptivas resultantes de la ejecución de los procesos de autoconsciencia (ver Fig. 5-1(j)).

En las subsiguientes secciones se describe en detalle cada uno de los componentes de la infraestructura de soporte a la metodología Aware-IoT@Run.Time.

5.2 DSL Monitor-IoT

En esta sección, se describe el lenguaje específico de dominio denominado Monitor-IoT, el cual apoya a los desarrolladores en el diseño de arquitecturas de monitoreo multicapa para sistemas IoT, con un alto nivel de abstracción, expresividad y flexibilidad, a fin de que estén exentos en gran medida de los detalles de implementación de estos sistemas, facilitando y agilizando sus tareas de desarrollo. Monitor-IoT está conformado por un diseñador gráfico (lenguaje de modelado visual de alto nivel) y un metamodelo alineado con la arquitectura de referencia ISO/IEC 30141:2018. Este metamodelo proporciona un lenguaje capaz de modelar arquitecturas con una amplia variedad de entidades digitales y flujos de datos (sincrónicos y asincrónicos) entre estas, a través de las capas Edge, Fog y Cloud; y, de esta forma soportar el monitoreo en una diversidad de escenarios de IoT [56].

Específicamente, Monitor-IoT provee un lenguaje para soportar principalmente: i) la definición de las entidades físicas y digitales a ser monitoreadas; ii) la definición de las entidades digitales (nodos de computación y sus recursos) que soportan los procesos de monitoreo (recolección, transporte, procesamiento y almacenamiento de datos), a nivel de las capas Edge, Fog y Cloud; iii) la especificación de las propiedades a monitorear para cada entidad IoT; iv) la definición de los flujos de datos entre las entidades digitales, basados en comunicaciones síncronas o asíncronas; y, v) el establecimiento de las operaciones de agregación para los datos recolectados [56].

Cabe mencionar que si bien Monitor-IoT está orientado en primera instancia al diseño de arquitecturas que soporten procesos de monitoreo de sistemas IoT, el metamodelo propuesto es altamente flexible y extensible. Por lo tanto, la ampliación y validación de Monitor-IoT para soportar otros tipos de procesos (p. ej., análisis, generación de reportes para la toma de decisiones, actuación) será abordada en trabajos futuros.

En las siguientes subsecciones, se explica cómo se construyó el metamodelo (sintaxis abstracta) en el que se basa el DSL Monitor-IoT. Además, se presenta el diseñador gráfico y la notación de los componentes (sintaxis concreta) que puede contener una arquitectura de IoT modelada con Monitor-IoT.

5.2.1 Metamodelo Monitor-IoT

El metamodelo Monitor-IoT fue construido a partir de la arquitectura de referencia para IoT ISO/IEC 30141:2018. Esta arquitectura incluye un modelo conceptual genérico y abstracto que describe las entidades claves (físicas y digitales) de un sistema IoT típico, así como las relaciones entre estas [2]. De ahí que, las entidades y relaciones claves de este modelo conceptual se reutilizaron para crear el metamodelo. Esta reutilización (alineación) proporciona al metamodelo un vocabulario común que puede ser empleado, sin ambigüedades, en cualquier subdominio o implementación de IoT. En concreto, se reutilizaron 18 de los 21 conceptos contenidos en el modelo conceptual. La mayoría de los conceptos reutilizados (*entity*, *physical entity*, *digital entity*, *network*, *IoT user*, *human*

user, non-human user, IoT device, sensor, tag, actuator, IoT gateway, application, service, database) se han representado como metaclases, excepto tres conceptos (*domain, endpoint, identifier*) que se han incluido como atributos de metaclase. Luego, con base en el análisis de los trabajos relacionados existentes en la literatura [112]–[124], se incluyeron en el metamodelo las entidades más comunes entre las propuestas, así como relevantes para los procesos de recolección, transporte, procesamiento y almacenamiento de datos de un sistema IoT. A su vez, se incorporaron al metamodelo entidades adicionales, necesarias tanto para integrar las entidades previamente identificadas como para soportar importantes primitivas funcionales de monitoreo no cubiertas previamente (por ejemplo, flujos de datos, plantillas de propiedades, estructura lógica de las bases de datos, middlewares, brokers, mapeo de propiedades a monitorear con API y bases de datos, operaciones de agregación de datos). Finalmente, para definir las relaciones entre las entidades (metaclases) fue fundamental comprender cómo estas colaboran durante los procesos de monitoreo de un ambiente de IoT [56].

Este enfoque de construcción ha permitido asegurar la coherencia semántica del metamodelo, alineándolo con la ISO/IEC 30141:2018, así como dotarlo de un lenguaje más completo y flexible, capaz de modelar una diversidad de alternativas de implementación de procesos de monitoreo y flujos de datos en sistemas IoT.

Adicionalmente, Monitor-IoT permite la construcción de arquitecturas de monitoreo para IoT con comunicación asíncrona y síncrona entre los nodos de computación. En el primer caso, el envío y la recepción de datos se separan temporalmente, lo cual resulta crucial para mejorar el rendimiento de los sistemas IoT a gran escala. Mientras que, en el segundo caso, el intercambio de datos (envío - recepción) se realiza en tiempo real, con el fin de priorizar aquellas solicitudes que requieren una respuesta inmediata.

El metamodelo Monitor-IoT se ha especificado en Ecore, utilizando las herramientas incluidas en Eclipse Modeling Framework (EMF) [100]. La Figura 5-2 presenta un extracto del metamodelo Monitor-IoT con sus principales metaclases y relaciones. Las metaclases incluidas en la figura se identifican con un número encerrado en un círculo de diferente color. El rojo identifica las metaclases alineadas con ISO/IEC 30141:2018, el azul identifica las metaclases seleccionadas de las propuestas estudiadas (trabajo relacionado); y, el verde identifica las metaclases adicionales incorporadas por los investigadores de este trabajo. Por su parte, en el Anexo A se describen detalladamente cada una de las metaclases de Monitor-IoT. Además, este Anexo contiene las Figuras A-1, A-2, A-3 y A-4 que presentan el metamodelo completo, incluyendo todas sus metaclases, relaciones, atributos y enumeraciones.

5.2.2 Diseñador gráfico de Monitor-IoT

El DSL Monitor-IoT proporciona una herramienta de diseño gráfico (lenguaje de modelado visual de alto nivel) que simplifica y facilita la creación y mantenimiento de modelos de arquitectura de monitoreo multicapa para sistemas IoT conformes con el vocabulario de dominio (metamodelo) presentado en el Anexo A. El diseñador gráfico fue construido en Obeo Designer Community Edition y Eclipse Sirius [99] para aprovechar las tecnologías de modelado incluidas en EMF (Eclipse Modeling Framework) y GMF (Graphical Modeling Framework) [100]. De esta forma, Monitor-IoT soporta la creación de un banco de trabajo de modelado personalizado para diseñar arquitecturas de monitoreo para IoT. El banco de trabajo de modelado está compuesto por un conjunto de editores (diagramas, tablas, árboles) cuya estructura y comportamiento están determinados por el metamodelo. Cabe mencionar que este diseñador gráfico fue

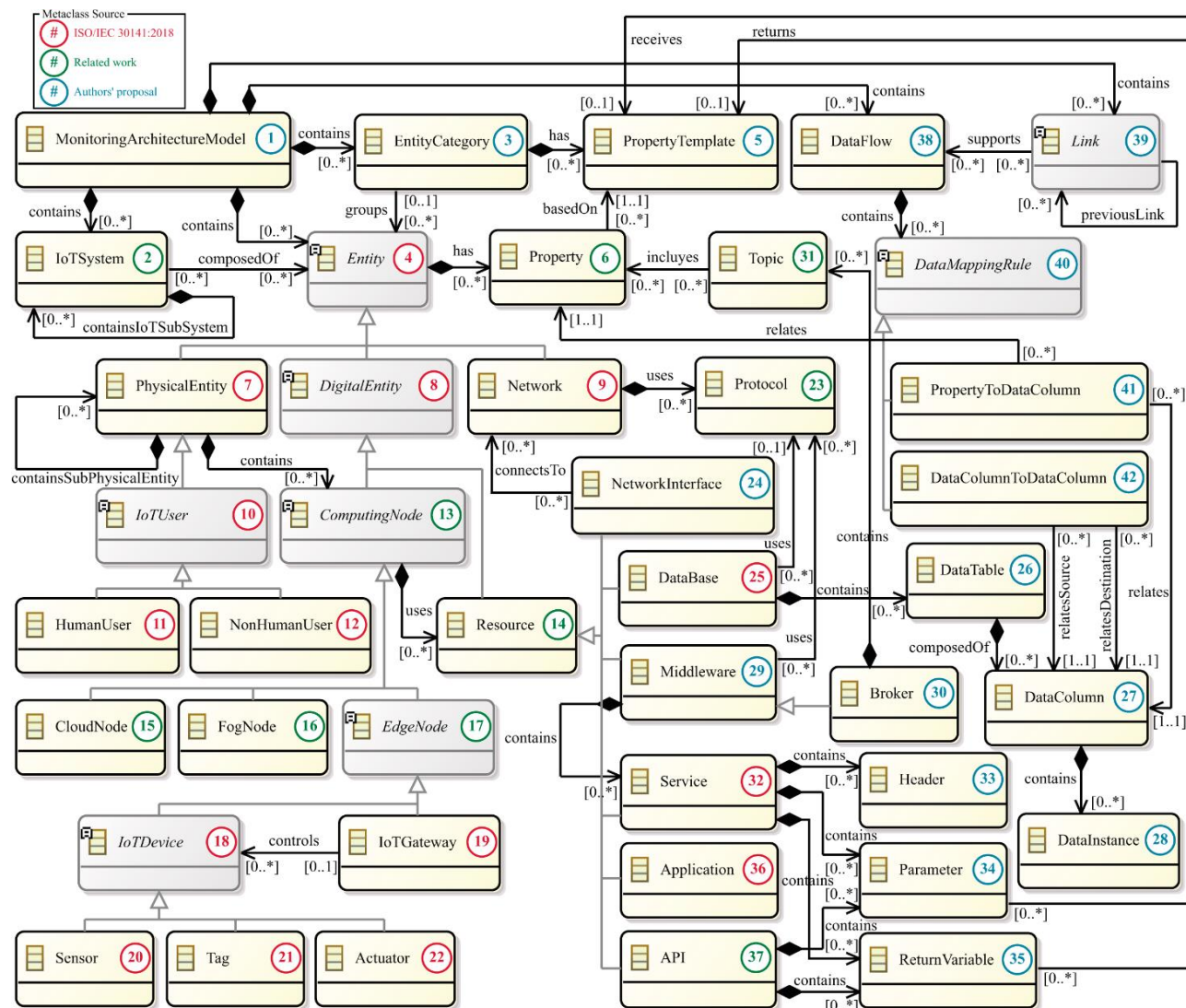


Figura 5-2. Extracto del metamodelo Monitor-IoT con las principales metaclasses y relaciones [56]. Fuente: Elaboración propia.

desarrollado con el apoyo de estudiantes de pregrado de la Universidad de Cuenca a través de un trabajo de titulación [171]. La Figura. 5-3 muestra el entorno de trabajo de Monitor-IoT, incluyendo el área de modelado, la paleta de herramientas y la ventana de propiedades; mientras que, la Figura 5-4 presenta la notación gráfica (sintaxis concreta) para cada una de las metaclasses de Monitor-IoT.

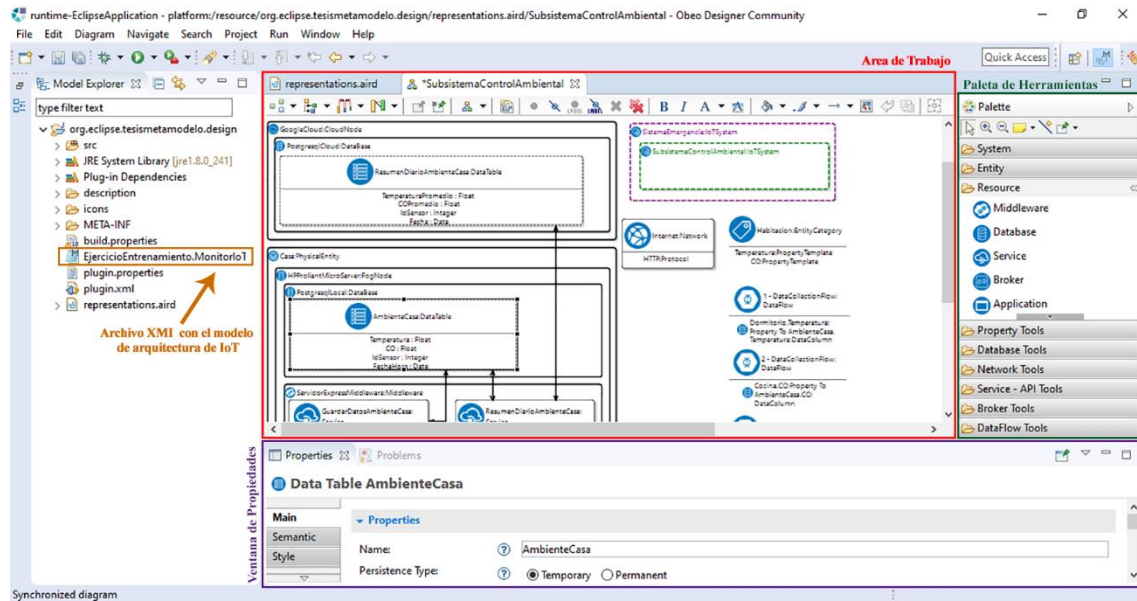


Figura 5-3. Entorno de trabajo del DSL Monitor-IoT.
Fuente: Elaboración propia.

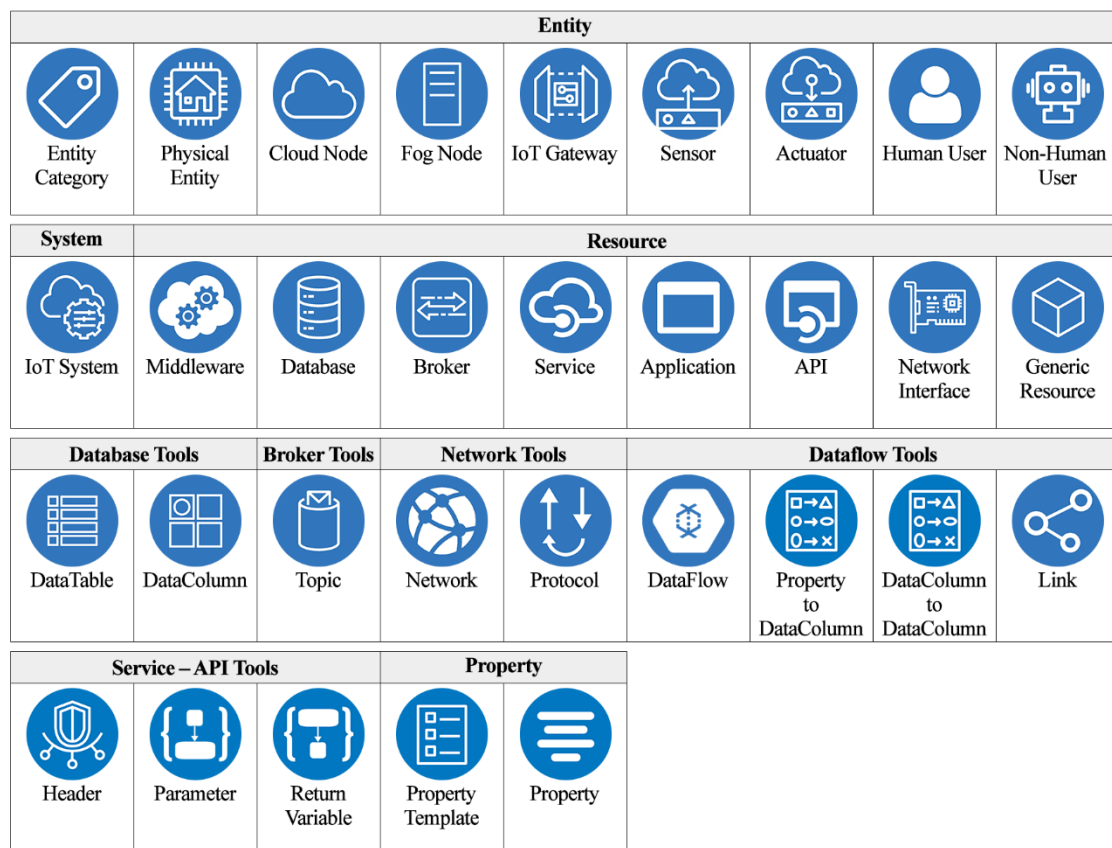


Figura 5-4. Notación gráfica del DSL Monitor-IoT [56].
Fuente: Elaboración propia.

A través de la interfaz de modelado visual de Monitor-IoT, los desarrolladores pueden diseñar modelos de arquitectura de IoT, brindando solo especificaciones de alto nivel sobre las entidades digitales (por ejemplo, sensores, API, aplicaciones, servicios, brokers, bases de datos) y las interrelaciones entre estas (flujos de datos), mismas que son requeridas para soportar los procesos de monitoreo de un sistema IoT, a nivel de las capas Edge, Fog y Cloud; es decir, los desarrolladores no tienen que preocuparse por cómo estas entidades y flujos de datos deben implementarse o codificarse a bajo nivel.

Monitor-IoT crea un modelo de arquitectura de monitoreo serializado en XMI para que una computadora pueda interpretarlo. De ahí que, las especificaciones incluidas en estos modelos pueden ser procesadas por algoritmos de transformación, a fin de generar automáticamente los recursos de software que soportan las operaciones de monitoreo de un sistema IoT. Particularmente, el componente de la infraestructura encargado de interpretar y procesar este modelo es el motor de sincronización de recursos de monitoreo, el cual se describe en la subsección 5.4.1.

En el sitio web de la herramienta DSL Monitor-IoT se pueden descargar los archivos fuente y un video que explica los pasos para su configuración y ejecución: <https://sites.google.com/uazuay.edu.ec/dslmonitoriot>.

5.3 Configurador web de capacidades de autoconsciencia de sistemas IoT

El configurador de capacidades de autoconsciencia es una aplicación web desarrollada en Node.js, un entorno de tiempo de ejecución de JavaScript, multiplataforma, de código abierto, orientado a eventos asincrónicos y basado en el motor V8 de Google [172][173]. Este configurador web de autoconsciencia tiene como propósito apoyar al equipo de desarrollo (expertos del dominio y programadores) en la definición y mantenimiento de las especificaciones de autoconsciencia de alto nivel de un sistema IoT, así como en la generación del correspondiente modelo de autoconsciencia y su integración con el modelo de arquitectura de monitoreo, obtenido del DSL Monitor-IoT [36]. Para este cometido, el configurador incluye las siguientes funcionalidades (ver Fig. 5-5):

- Establecer el dominio de autoconsciencia, es decir, el sujeto (sistema o subsistema IoT) que es consciente y los objetos (entidades IoT físicas y digitales) de los que es consciente un sistema IoT.
- Definir los aspectos de autoconsciencia individuales y colectivos a capturar sobre las entidades IoT u objetos de autoconsciencia (p. ej., conocer el estado actual del monóxido de carbono en el ambiente de una casa, predecir la carga de procesamiento de un Gateway de IoT para un número determinado de sensores).
- Configurar los procesos de autoconsciencia pre-reflexivos y reflexivos, junto con sus métodos de autoconsciencia (recolección, cálculo y modelos de análisis), así como la forma de operacionalizar estos procesos y métodos mediante los recursos de implementación.
- Definir las escalas, unidades de medida y métricas (directas, indirectas e indicadores) a emplear para la evaluación de los aspectos de autoconsciencia.
- Establecer los criterios de decisión y umbrales a utilizar por los modelos de análisis para transformar un indicador cuantitativo en una valoración cualitativa de un aspecto de autoconsciencia.
- Definir los recursos de implementación genéricos (fórmulas, funciones o servicios web) que podrán ser utilizados para operacionalizar los procesos y métodos de autoconsciencia.
- Integrar y generar los modelos de arquitectura y autoconsciencia en tiempo de

ejecución en un único archivo en formato JSON, de tal forma que las especificaciones de autoconsciencia configuradas se integren e interrelacionen con los componentes del modelo de arquitectura de monitoreo del sistema IoT.



Figura 5-5. Página principal del configurador web de capacidades de autoconsciencia.
Fuente: Elaboración propia.

El configurador utiliza un metamodelo denominado Aware-IoT, construido a partir del framework conceptual de autoconsciencia de Lewis et al. [20] y la ontología de medición de software de García et al. [106]; de ahí que, los modelos de autoconsciencia generados por el configurador son conformes a este metamodelo.

Además, el configurador es multiusuario y permite crear áreas de trabajo para configurar de forma independiente las especificaciones de autoconsciencia de cada sistema IoT. En este sentido, el configurador permite asociar a cada área de trabajo el modelo de arquitectura de monitoreo del sistema IoT en cuestión, a fin de que la información sobre sus subsistemas, entidades y flujos de datos pueda ser utilizada durante las tareas de configuración de las capacidades de autoconsciencia de dicho sistema.

Por su parte, la información sobre las métricas, criterios de decisión (umbrales) y recursos de implementación (fórmulas, funciones o servicios web) es de propósito general; es decir, se encuentra disponible en todas las áreas de trabajo, con el objetivo de que estos recursos puedan ser reutilizados durante la configuración de las especificaciones de autoconsciencia de cualquier sistema IoT.

El configurador almacena las especificaciones de autoconsciencia en una base de datos en MySQL [174]. De ahí que, a partir de la información contenida en la base de datos se puede generar en cualquier momento el archivo JSON con los modelos de arquitectura y autoconsciencia en tiempo de ejecución.

Por último, este configurador también fue desarrollado con la colaboración de estudiantes de pregrado de la Universidad del Azuay mediante pasantías y un trabajo de titulación [175].

En las siguientes subsecciones, se describe cómo se construyó el metamodelo (sintaxis abstracta) en el que se basa el configurador web de autoconsciencia. Además, se detalla la arquitectura del configurador web de autoconsciencia y sus principales componentes.

5.3.1 Metamodelo Aware-IoT

Como se ha mencionado con anterioridad, el metamodelo de autoconsciencia (Aware-IoT) para sistemas IoT fue construido a partir del framework conceptual de niveles y aspectos de autoconsciencia propuesto por Lewis et al. [20]. Este framework enmarca las capacidades de autoconsciencia de un sistema computacional dentro de tres niveles: i) *pre-reflexivo (consciencia del estímulo)*, si un sistema es capaz de obtener conocimiento subjetivo directamente de los datos empíricos recolectados desde los sensores u otras fuentes (observación inmediata), permitiéndole responder a los eventos de manera inmediata. ii) *reflexivo*, si un sistema es capaz de construir y/o aplicar modelos conceptuales a partir de los datos históricos recopilados desde los sensores, los cuales capturan el conocimiento sobre los diferentes aspectos de un objeto (p. ej., estado, comportamiento, evolución, predicción); y, iii) *meta-reflexivo*, si un sistema es capaz de obtener conocimiento de sus propios niveles y procesos de consciencia, a fin de razonar sobre los beneficios y costos de mantener o adaptar un cierto nivel de consciencia [20].

El metamodelo propuesto se centra en los dos primeros niveles de autoconsciencia y proporciona un lenguaje conformado por un conjunto de conceptos para describir el dominio (sujetos, objetos y aspectos) de autoconsciencia, así como los procesos de autoconsciencia, a nivel pre-reflexivo y reflexivo, de un sistema IoT.

Adicionalmente, en el metamodelo de autoconsciencia se han incorporado y adaptado varios conceptos de la ontología de medición de software propuesta por un grupo de investigadores reconocidos en el área de la calidad del software [106], tales como: unidad, escala, métrica directa e indirecta, indicador, método de medición (modificado por método de recolección), función de cálculo (modificado por método de cálculo), modelo de análisis y criterio de decisión. El nombre de algunos conceptos ha sido modificado para asegurar la pertinencia del mismo con su función. En concreto, estos conceptos tienen como propósito soportar la representación de los métodos de autoconsciencia (aprendizaje y razonamiento), los cuales son los encargados de definir la forma como los procesos pre-reflexivos y reflexivos evalúan (capturan el conocimiento) los aspectos de autoconsciencia de un sistema IoT y su entorno. En este sentido, el metamodelo a nivel pre-reflexivo permite especificar métodos de recolección de datos desde sensores u otras fuentes para obtener métricas directas. Mientras que, a nivel reflexivo posibilita la definición de métodos de cálculo basados en modelos conceptuales para a partir de métricas directas y/o indirectas obtenidas con anterioridad, producir nuevas métricas indirectas. A su vez, el metamodelo incluye la representación de modelos de análisis para comparar las métricas directas o indirectas con los criterios de decisión (umbrales) y obtener indicadores que sirvan para valorar cuantitativa y cualitativamente los aspectos de autoconsciencia de un sistema IoT; y, de esta forma respaldar acciones (recomendaciones prescriptivas) en caso de ser necesarias.

Por otro lado, el metamodelo de autoconsciencia se integra con el metamodelo Monitor-IoT [56] descrito en la subsección 5.2.1. Para ello, se han definido relaciones con las metaclases *IoTSystem*, *Entity*, *Property*, *DataFlow*, *ComputingNode*, *Middleware*, *DataBase*, *DataTable* y *DataColumn* del metamodelo Monitor-IoT con el propósito de obtener las especificaciones sobre: i) la estructura del sistema IoT (subsistemas, entidades IoT físicas y digitales, propiedades), a partir de la cual se definirán los sujetos, objetos y aspectos de autoconsciencia; ii) los nodos de computación y middlewares del sistema IoT, en los cuales se ejecutarán de manera distribuida los procesos de autoconsciencia y recursos de implementación (funciones, servicios); iii) los flujos de datos que emplea el sistema IoT, mismos que podrán ser utilizados por los métodos de recolección; y, iv) la

estructura de almacenamiento (base de datos) utilizada por el sistema IoT, misma que se empleará para definir el lugar donde se almacenarán las métricas, metadatos y recomendaciones resultantes de la ejecución de los procesos de autoconsciencia.

Al igual que Monitor-IoT, el metamodelo Aware-IoT ha sido especificado en Ecore, utilizando las herramientas incluidas en el Eclipse Modeling Framework (EMF) [100]. La Figura 5-6 presenta un extracto del metamodelo Aware-IoT con sus metaclases y relaciones. Las metaclases incluidas en la figura se identifican con un número encerrado en un círculo de diferente color. El rojo identifica las metaclases incorporadas a partir del framework conceptual de autoconsciencia [20], el verde representa las metaclases seleccionadas desde la ontología de medición de software [106], el marrón representa las metaclases de Monitor-IoT [56]; y, el azul identifica las metaclases propias incorporadas por los investigadores de este trabajo. A su vez, en el Anexo B se explican detalladamente cada una de las metaclases de Aware-IoT. Este Anexo contiene las Figuras B-1, B-2, B-3 y B-4 que muestran el metamodelo completo, incluyendo sus metaclases, atributos, relaciones y enumeraciones.

5.3.2 Arquitectura del configurador de autoconsciencia

El configurador web de autoconsciencia tiene una arquitectura compuesta por cuatro capas: cliente, lógica de presentación, lógica de negocio y datos. Estas capas se encuentran lo más desacopladas posibles, con el propósito de contribuir a la reutilización del código y mejorar la interoperabilidad, escalabilidad y mantenibilidad de la aplicación. En la Figura 5-7 se presenta la arquitectura del configurador, incluyendo sus capas, componentes e interfaces de comunicación entre estos. En lo que resta de la subsección, se describe la funcionalidad de cada una de las capas y sus componentes:

- **Capa de cliente.** Representa la interfaz de usuario del configurador de autoconsciencia, compuesta por un conjunto de páginas HTML dinámicas. En este sentido, por medio de un navegador web, el equipo de desarrollo interactúa con las páginas HTML (formularios web) para proporcionar las especificaciones necesarias (basadas en conceptos de alto nivel) para la construcción del modelo de autoconsciencia. Además, a través de la interfaz de usuario se provee el modelo de arquitectura de monitoreo del sistema IoT construido en el DSL Monitor-IoT, con el propósito de que la información sobre sus subsistemas, entidades IoT y flujos de datos esté disponible para la configuración de las capacidades de autoconsciencia del sistema en cuestión. Particularmente, la capa de cliente envía las solicitudes al servidor de aplicaciones (framework Express [176]) mediante el protocolo HTTP para que sean procesadas por las capas inferiores. Una vez procesadas las solicitudes, el servidor de aplicaciones devuelve la respuesta en formato HTML. Las páginas HTML también incluyen código JavaScript y hojas de estilo CSS (Cascading Style Sheets) para validar el formato de los datos de entrada y preparar la forma de presentar la información resultante al usuario.
- **Capa de lógica de presentación.** Contiene el servidor de aplicaciones web implementado en el framework Express de Node.js [176], compuesto por un conjunto de APIs RESTful (APIs Web) y plantillas de vistas (páginas y parciales) en EJS (Embedded JavaScript templating) [177]. Las APIs sirven de enlace entre la capa de cliente y la capa de lógica de negocio, ocultando el código de esta última. De ahí que, las APIs son consumidas desde las plantillas de vistas; y, en su cuerpo invocan a los métodos de la capa de negocio para procesar las solicitudes provenientes de la capa

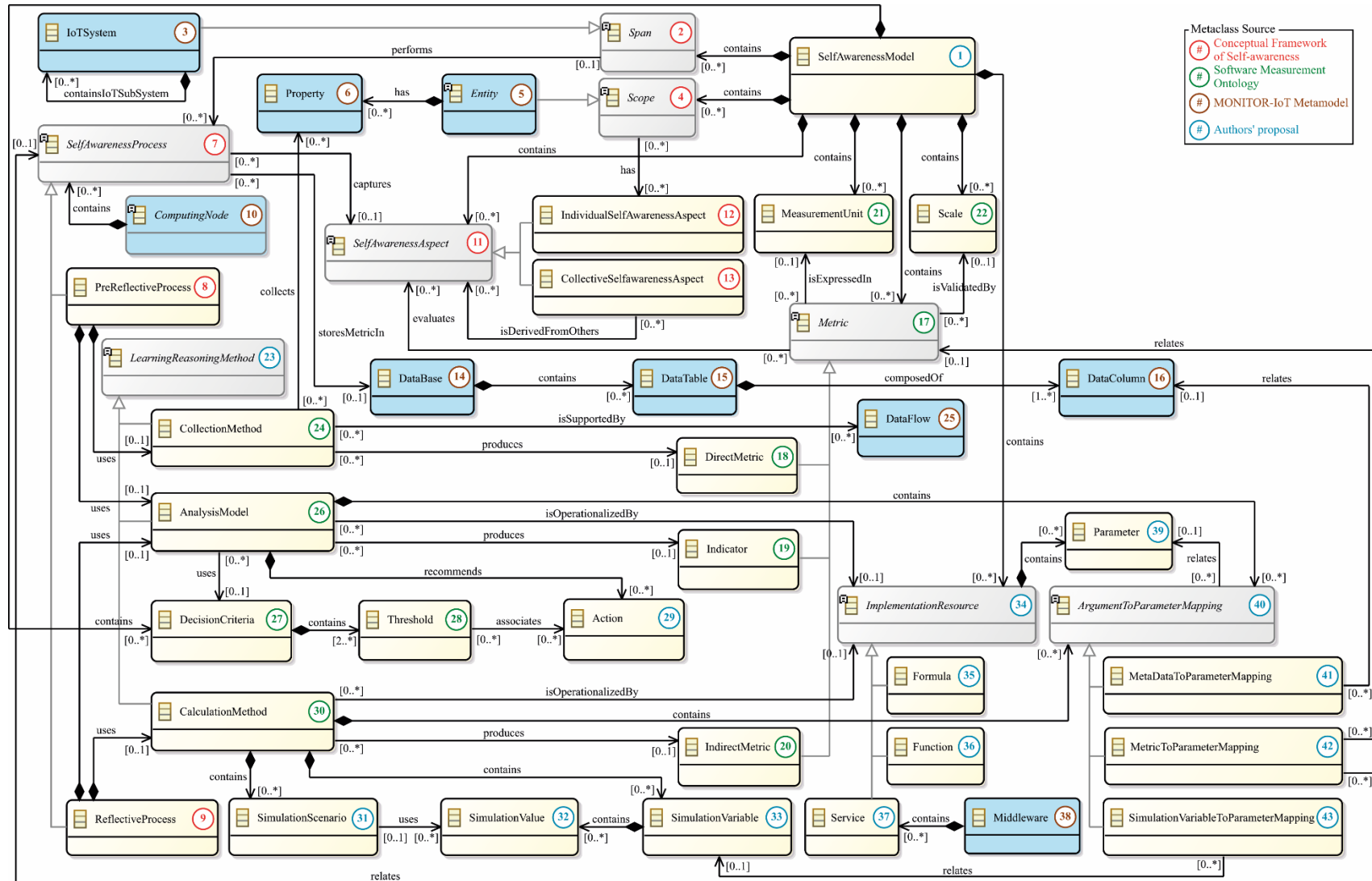


Figura 5-6. Extracto del metamodelo Aware-IoT con sus metaclasses y relaciones. Fuente: Elaboración propia.

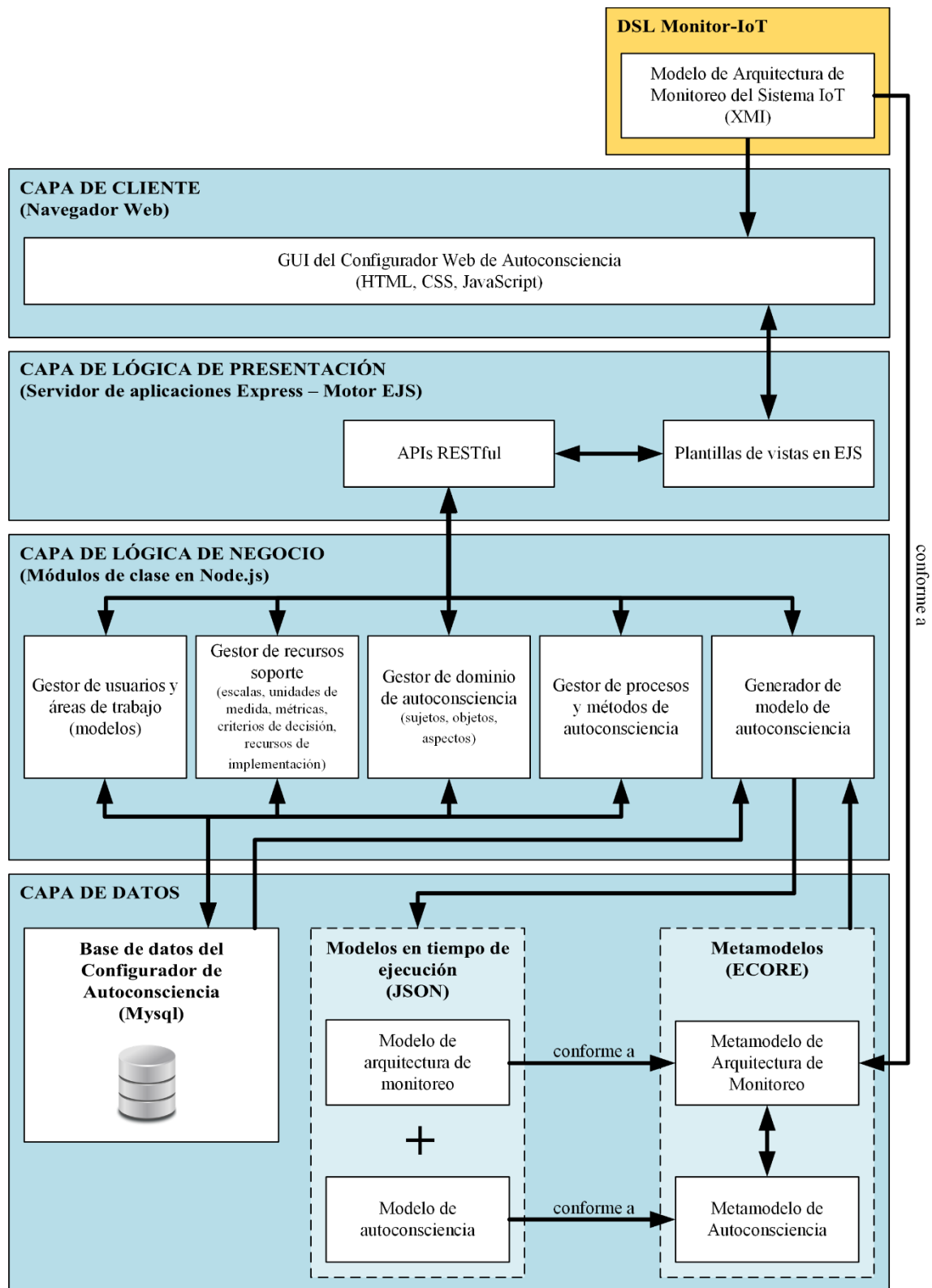


Figura 5-7. Arquitectura del configurador web de capacidades de autoconsciencia de sistemas IoT. Fuente: Elaboración propia.

de cliente. Finalmente, las APIs devuelven las respuestas en formato JSON. Por su parte, las plantillas EJS son archivos de texto que combinan HTML con código JavaScript delimitado entre etiquetas. Estas plantillas son renderizadas por el motor EJS utilizando las respuestas en JSON devueltas por las APIs, con el fin de generar y enviar las páginas HTML a la capa de cliente. Como complemento, en esta capa se utiliza el middleware Router de Express para crear manejadores de rutas de acceso montables y modulares a las APIs y páginas web del configurador [178].

- **Capa de lógica de negocio.** Incluye un conjunto de módulos de clases en Node.js que encapsulan la lógica de negocio del configurador de autoconsciencia; y, por ende, proporcionan métodos para gestionar: i) los usuarios de la aplicación y sus áreas de trabajo (modelos); ii) los recursos de soporte genéricos a los procesos de autoconsciencia (escalas, unidades de medida, métricas, criterios de decisión, recursos de implementación); iii) el dominio de autoconsciencia (sujetos, objetos, aspectos); y, iv) los procesos y métodos de autoconsciencia. Además, esta capa provee un método para generar un archivo en formato JSON con los modelos de arquitectura y autoconsciencia en tiempo de ejecución, asegurando que estos modelos sean conformes con los metamodelos Monitor-IoT y Aware-IoT. Finalmente, los métodos incluyen instrucciones para consultar, validar, procesar y/o almacenar la información gestionada por el configurador en una base de datos relacional en MySQL.
- **Capa de datos.** Conformada por la base de datos relacional en MySQL que almacena de manera persistente las especificaciones de autoconsciencia configuradas en cada una de las áreas de trabajo. Además, en esta capa se almacena el archivo JSON generado por el configurador, conteniendo los modelos de arquitectura y autoconsciencia en tiempo de ejecución del sistema IoT gestionado, mismo que será detectado y utilizado automáticamente por el middleware de autoconsciencia para planificar las tareas de creación y/o reconfiguración de los recursos de monitoreo y autoconsciencia del sistema IoT.

5.4 Middleware de la infraestructura de soporte

Según lo mencionado con anterioridad, el middleware de la infraestructura tiene como objetivo automatizar la tercera actividad de la metodología Aware-IoT@Run.Time. Por consiguiente, el middleware es una parte esencial de la infraestructura que tiene como propósito mantener la relación causal entre los modelos de arquitectura - autoconsciencia y el sistema IoT gestionado; es decir, detecta en tiempo de ejecución cualquier cambio en las especificaciones de los modelos y dispara automáticamente acciones de implementación y/o reconfiguración de los recursos de software que soportan los procesos de monitorización y autoconsciencia, a nivel de las capas Edge, Fog y Cloud, sin detener la operación del sistema IoT.

Al igual que el configurador de autoconsciencia, un prototipo del middleware ha sido desarrollado progresivamente en Node.js con la colaboración de varios tesis de pregrado de la Universidad del Azuay [179][180][181][182]. Node.js es una herramienta diseñada para construir aplicaciones de red altamente distribuidas, heterogéneas, dinámicas y escalables, semejantes a las aplicaciones de software requeridas por los sistemas IoT [172][173]. En este sentido, Node.js es un entorno de tiempo de ejecución multiplataforma, basado en el motor V8 de Google, cuyas aplicaciones ligeras pueden ejecutarse de manera sencilla y eficiente en una gran cantidad de sistemas operativos tanto para servidores como dispositivos móviles o Edge (ubicados en el borde de una red IoT) con limitadas capacidades de hardware y energía [173][183]. A su vez, Node.js está

basado en el lenguaje JavaScript que soporta la metaprogramación reflexiva, es decir, una técnica de programación para crear aplicaciones que sean capaces de analizar, generar y transformar otros programas, e incluso modificar su propio código mientras se ejecuta [184]. Esto resulta apropiado para el enfoque propuesto basado en modelos en tiempo de ejecución, ya que se requiere crear o modificar en tiempo de ejecución la base del código de los recursos de software de monitoreo o autoconsciencia, según las especificaciones dinámicas contenidas en los modelos. Por otro lado, Node.js está basado en un modelo de entrada y salida sin bloqueos controlado por eventos asíncronos, lo que le permite manejar una gran cantidad de conexiones simultáneas, con un alto nivel de rendimiento y escalabilidad [173][185]. En particular, los sistemas IoT suelen estar compuestos por múltiples sensores que se incrementan rápidamente en el tiempo y generan una cantidad considerable de peticiones y volumen de datos, siendo el modelo de programación de Node.js ideal para gestionar eficientemente estas peticiones concurrentes [185][186].

Más allá de los justificativos enumerados que conllevaron a elegir Node.js como la plataforma tecnológica para implementar el prototipo del middleware de la infraestructura que soporta la metodología Aware-IoT@Run.Time, se debe ser enfático en que la solución propuesta, tanto a nivel de la metodología como de la arquitectura de la infraestructura, al estar basado en paradigmas de MDE (models@run.time), es independiente del dominio de implementación, por lo que esta solución puede ser rápidamente replicada en cualquier otra plataforma tecnológica, mediante la implementación de un nuevo middleware que se encargue de transformar las especificaciones de los modelos en artefactos de software de monitoreo y autoconsciencia para la plataforma tecnológica destino.

El middleware de la infraestructura está conformado por los motores de sincronización de recursos de monitoreo y autoconsciencia. A su vez, las funcionalidades de estos motores están organizadas en componentes o módulos, debiendo recalcar que existen componentes que son compartidos y reutilizados por ambos motores, ya que encapsulan funcionalidades comunes. El middleware es lo suficientemente flexible y parametrizable, de tal forma que los motores puedan ser ejecutados de manera independiente o en conjunto; es decir, dependiendo del alcance de los requerimientos de los usuarios finales se puede únicamente configurar y ejecutar el motor de recursos de monitoreo; o, por el contrario, ejecutar de forma integral y coordinada ambos motores. En las siguientes subsecciones, se detallan las funcionalidades de cada uno de los motores y se describen los componentes de la arquitectura del middleware y sus motores, así como las interfaces de comunicación entre estos.

5.4.1 Motor de sincronización de recursos de monitoreo

Este motor tiene como alcance interpretar en tiempo de ejecución las especificaciones contenidas en el modelo de arquitectura de monitoreo; y, a partir de estas, implementar, reconfigurar y poner en operación los artefactos de software (APIs, aplicaciones, servicios web, tópicos de broker, bases de datos y tablas) que soportan los procesos de monitoreo del sistema IoT. A su vez, estos artefactos son implementados de forma distribuida en los diversos nodos de computación de un sistema IoT, ubicados a nivel de las capas Edge, Fog y Cloud. Puntualmente, las actividades que debe realizar el motor de sincronización de recursos de monitoreo se pueden dividir en dos grupos. En el primer grupo se encuentran un conjunto de actividades relacionadas con la planificación, distribución y control de las tareas de implementación y reconfiguración de los recursos de monitoreo del sistema IoT. Estas actividades son ejecutadas por un nodo de

computación que opera en calidad de supervisor del sistema IoT. A continuación, se enumeran las principales actividades de este grupo:

- Observar y detectar de forma periódica, cualquier cambio en el archivo JSON que contiene los modelos de arquitectura de monitoreo y autoconsciencia del sistema IoT.
- Identificar todos los nodos de computación (a nivel Edge, Fog y Cloud) especificados en el nuevo modelo de arquitectura de monitoreo del sistema IoT.
- Comparar la nueva versión del modelo de arquitectura de monitoreo con la configuración en ejecución del sistema IoT. De ser necesario, planificar las tareas de implementación de los nuevos artefactos de monitoreo requeridos o de reconfiguración de los artefactos existentes, a ser ejecutadas en cada uno de los nodos de computación identificados en paso anterior.
- Distribuir la nueva versión del modelo de arquitectura y el plan de implementación y reconfiguración de los recursos de monitoreo a todos los nodos de computación del sistema IoT.
- Controlar el cumplimiento del plan de implementación y reconfiguración de los recursos de monitoreo, incluso en el caso de operar en conjunto con el motor responsable de los recursos de autoconsciencia, esta actividad contempla también la verificación del cumplimiento de las tareas de implementación y reconfiguración de los recursos de autoconsciencia, ya que el objetivo final es garantizar un funcionamiento sincronizado entre los motores, que asegure la integridad y consistencia de las operaciones de monitoreo y autoconsciencia en todos los nodos de computación del sistema IoT. Para ello, el nodo supervisor revisa los reportes individuales recibidos de cada nodo de computación sobre el cumplimiento de los planes de implementación y reconfiguración de los recursos de monitoreo y autoconsciencia. En caso existir en alguno de los nodos, problemas en la ejecución de estos planes, el nodo supervisor envía un mensaje a todos los nodos para que retornen a la última configuración válida de monitoreo del sistema IoT; caso contrario, envía un mensaje indicando que la ejecución de los planes a nivel de todos los nodos ha sido exitosa, por lo que se debe establecer la nueva versión del modelo de arquitectura como la nueva configuración de monitoreo en ejecución del sistema IoT.
- Visualizar en la consola del nodo supervisor, los reportes de logs sobre la ejecución de las tareas de implementación y reconfiguración de los recursos de monitoreo.

Por su parte, el segundo grupo está conformado por actividades orientadas propiamente a la implementación y/o reconfiguración de los recursos de monitoreo, las cuales son ejecutadas localmente en cada uno de los nodos de computación del sistema IoT. A continuación, se describen las actividades de este grupo:

- Receptar la nueva versión del modelo de arquitectura de monitoreo enviada por el nodo supervisor, así como el plan de implementación y reconfiguración de los recursos de monitoreo para su respectiva ejecución.
- Crear o alterar la estructura de las bases de datos y tablas que almacenan los datos resultantes de los procesos de monitoreo del sistema IoT. En caso de que el plan de reconfiguración incluya tareas de eliminación de columnas, previo a la ejecución de estas tareas, el motor debe respaldar la base de datos de monitoreo.
- Crear nuevas suscripciones a tópicos de broker o actualizar las existentes, a fin soportar los flujos de comunicación asíncrona entre los artefactos de software de monitoreo.
- Implementar o reconfigurar las funciones APIs que recuperan los datos crudos desde los sensores u otras fuentes.

- Implementar o reconfigurar los servicios web que recuperan, procesan y almacenan los datos recolectados y sus metadatos con diferentes niveles de agregación. Para el caso de los flujos de recolección de datos, estos servicios web almacenan los datos crudos recibidos desde los sensores u otras fuentes en las bases de datos de monitoreo del sistema IoT; mientras que, para el caso de los flujos de agregación, estos servicios web recuperan los datos crudos previamente almacenados en una tabla fuente, aplican operaciones de agregación sobre estos datos; y, el resultado lo almacenan en una tabla destino.
- Implementar o reconfigurar las aplicaciones que enrutan y controlan los flujos de datos entre las APIs, servicios web y brokers. Como ya se ha mencionado, para el caso de los flujos de recolección de datos, las aplicaciones enrutan los datos obtenidos desde las APIs hacia los respectivos servicios web (comunicación síncrona) o brokers (comunicación asíncrona) para gestionar su almacenamiento en una base de datos. Mientras que, para el caso de los flujos de agregación de datos, las aplicaciones consumen periódicamente uno o varios servicios web que procesan los datos crudos de monitoreo para almacenarlos con diferentes niveles de agregación.
- Enviar al nodo supervisor un reporte sobre el estado de cumplimiento del plan de implementación y reconfiguración de los recursos de monitoreo ejecutado en el nodo de computación en cuestión.
- Visualizar en la consola del nodo de computación, los logs resultantes de la ejecución local de las tareas de implementación y reconfiguración de los recursos de monitoreo.

En la Figura 5-8 se presenta un diagrama con la secuencia de ejecución de las actividades del motor de sincronización de recursos de monitoreo, tanto a nivel del nodo supervisor como del resto de nodos del sistema IoT.

5.4.2 Motor de sincronización de recursos de autoconsciencia

Este motor tiene como propósito implementar, reconfigurar y poner en operación de forma distribuida los artefactos de software (demonios, fórmulas, funciones, servicios web, bases de datos, tablas) que operacionalizan los procesos de autoconsciencia pre-reflexivos y reflexivos, a nivel de las capas Edge, Fog y Cloud de un sistema IoT. Para ello, el motor analiza e interpreta en tiempo de ejecución las especificaciones tanto del modelo de arquitectura de monitoreo como del modelo de autoconsciencia del sistema IoT. De igual manera, el motor de recursos de autoconsciencia contempla un conjunto de funcionalidades y actividades similares al motor de recursos de monitoreo, mismas que se ejecutan tanto a nivel del nodo supervisor como del resto de nodos de computación del sistema IoT. En la Figura 5-9 se presenta un diagrama con la secuencia de ejecución de las actividades del motor de sincronización de recursos de autoconsciencia.

En el nodo supervisor, las actividades relacionadas con la observación y detección de cambios en el archivo JSON de los modelos, la identificación de los nodos de computación en el modelo; y, la distribución y control del plan de implementación y reconfiguración de los recursos de autoconsciencia son gestionadas a través de componentes compartidos con el motor de recursos de monitoreo (ver Fig. 5-10), ya que tienen un mismo comportamiento funcional. Las únicas actividades que se ejecutan a través de componentes independientes son las concernientes a la comparación de la nueva versión del modelo de autoconsciencia con la configuración en ejecución del sistema IoT; y, la creación del plan de implementación y reconfiguración de los artefactos de autoconsciencia; ya que estas actividades realizan un análisis sobre el contenido propio del modelo de autoconsciencia.

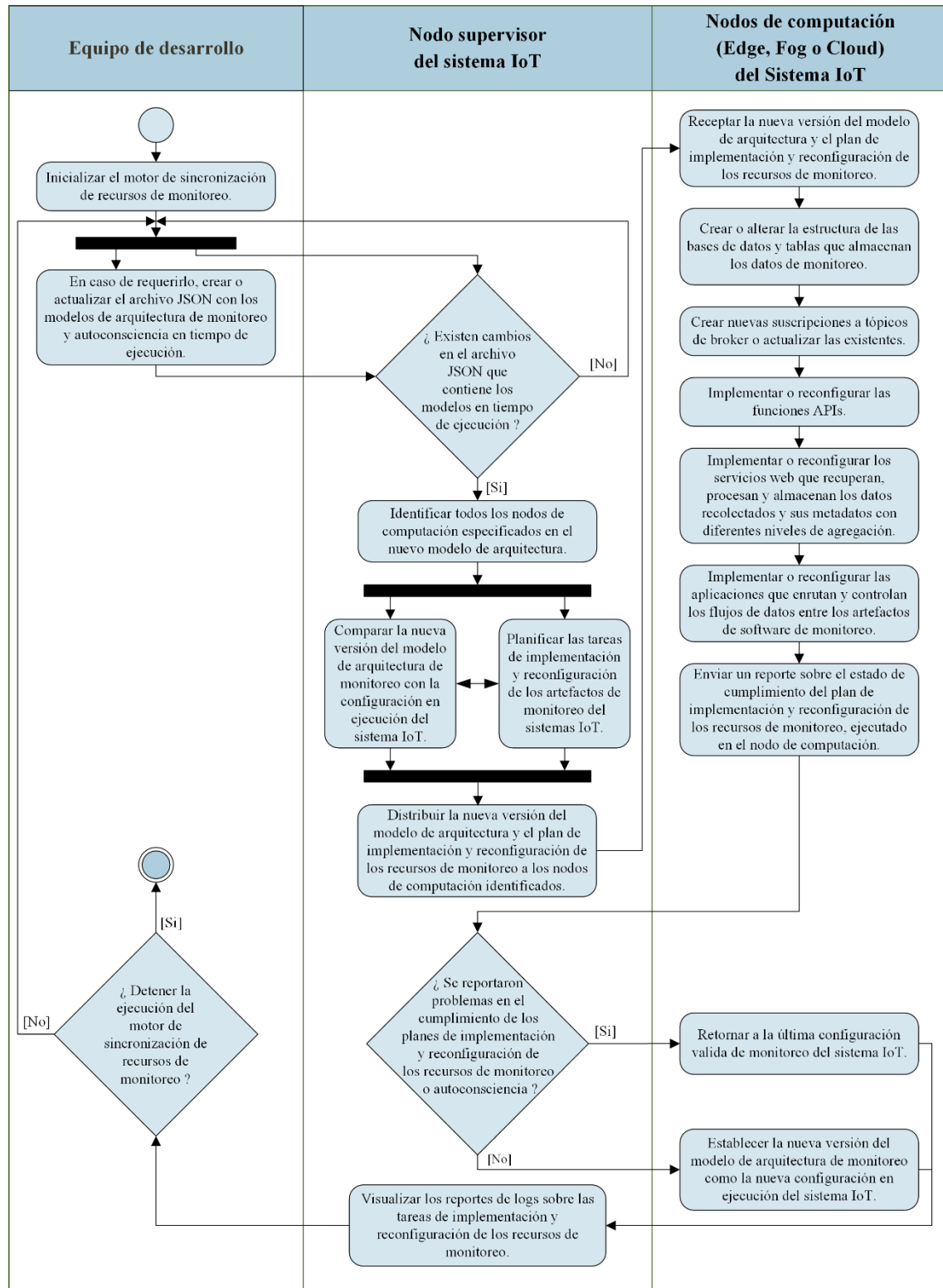


Figura 5-8. Diagrama de actividades del motor de sincronización de recursos de monitoreo de sistemas IoT. Fuente: Elaboración propia.

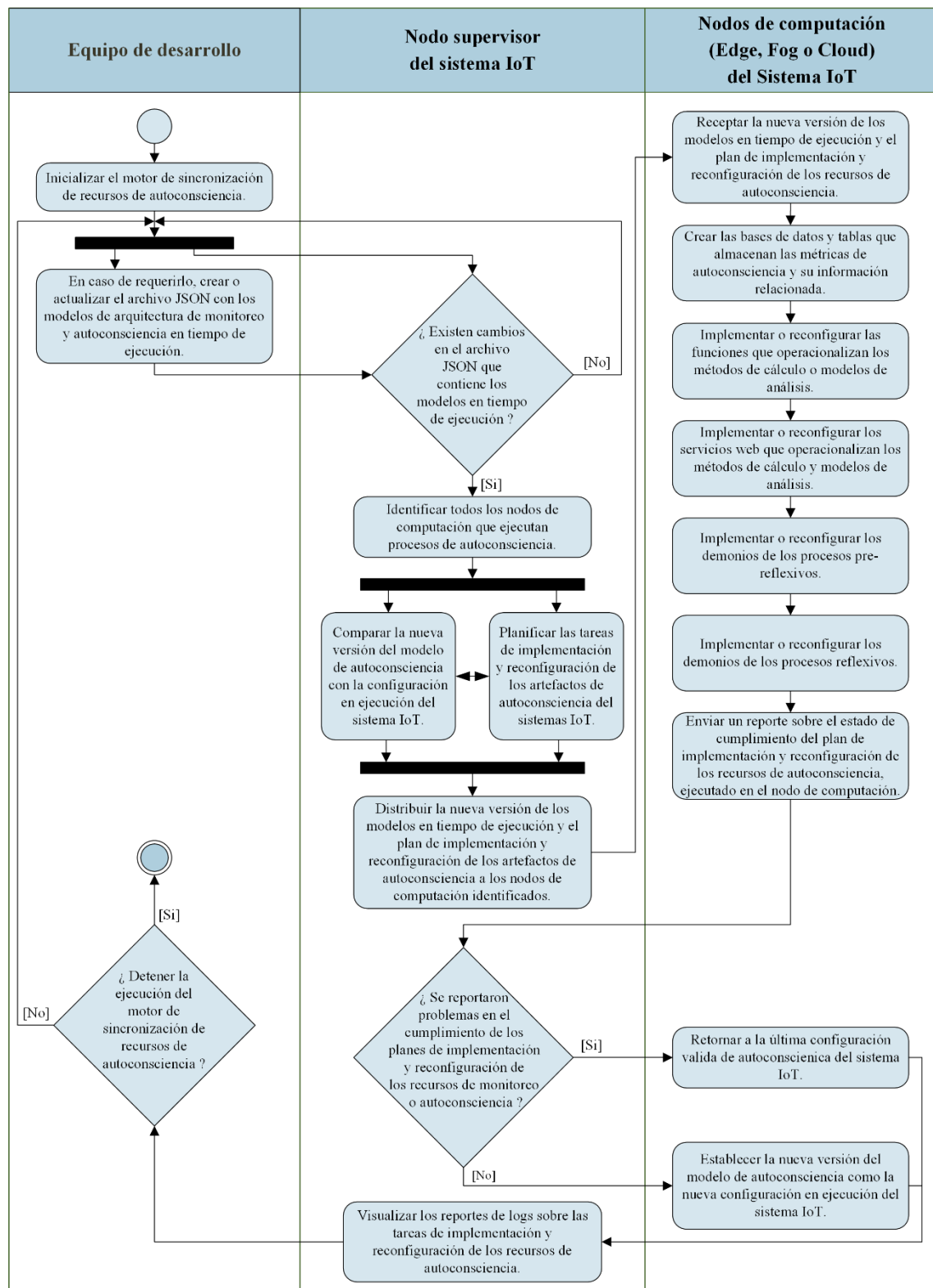


Figura 5-9. Diagrama de actividades del motor de sincronización de recursos de autoconsciencia de sistemas IoT. Fuente: Elaboración propia.

A nivel de los nodos de computación del sistema IoT, las actividades de recepción y control del plan son gestionadas de manera unificada con el motor de recursos de monitoreo. Adicionalmente, el motor de recursos de autoconsciencia contempla otras actividades a nivel de los nodos de computación, enfocadas propiamente a la creación o reconfiguración distribuida de los artefactos de autoconsciencia del sistema IoT, mismas que se enumeran a continuación:

- Crear las bases de datos y tablas que almacenan las métricas, metadatos y recomendaciones resultantes de la ejecución de los procesos de autoconsciencia.
- Implementar o reconfigurar las funciones que operacionalizan los métodos de cálculo o modelos de análisis.
- Implementar o reconfigurar los servicios web que operacionalizan los métodos de cálculo y modelos de análisis.
- Implementar o reconfigurar los demonios de los procesos pre-reflexivos.
- Implementar o reconfigurar los demonios de los procesos reflexivos.

Como se puede observar en las Figuras 5-8 y 5-9, ambos motores validan el cumplimiento tanto del plan de implementación y reconfiguración de los recursos de monitoreo como de los recursos de autoconsciencia. Esta validación integral garantiza que los recursos de monitoreo y autoconsciencia del sistema IoT operen siempre en función de una misma versión de los modelos de arquitectura y autoconsciencia.

5.4.3 Arquitectura del middleware de la infraestructura

En razón de que los motores de sincronización de recursos de monitoreo y autoconsciencia deben realizar actividades tanto a nivel del nodo supervisor como del resto de nodos de computación del sistema IoT, las actividades de estos motores se automatizan a través de dos aplicaciones de software: i) una aplicación denominada *supervisora*, que corre en el nodo supervisor, encargada de las actividades de detección, planificación, distribución y control de las tareas de implementación y reconfiguración de los recursos de monitoreo y autoconsciencia; y, ii) una aplicación denominada *implementadora*, que se ejecuta en el resto de nodos del sistema IoT, responsable de las actividades propias de implementación y reconfiguración de los recursos de software.

Estas aplicaciones están conformadas por un conjunto de componentes distribuidos en cuatro capas: cliente, comunicación (síncrona o asíncrona), lógica de negocio y datos (ver Fig. 5-10). De igual manera, los componentes presentan una alta cohesión y bajo acoplamiento, con el fin de beneficiar la reutilización, escalabilidad y mantenibilidad de las aplicaciones de software. Además, varios de los componentes del middleware están codificados mediante la técnica de metaprogramación reflexiva, brindándoles la capacidad de auto modificar su propio código; es decir, sobrescribir o generar nuevo código en tiempo de ejecución con base en las especificaciones de los modelos [184]. Puntualmente, en la plataforma Node.js se utiliza la función *eval* [187] de JavaScript para generar el código de los artefactos de software de monitoreo y autoconsciencia en tiempo de ejecución, tales como: funciones (APIs), aplicaciones, servicios web y demonios.

Por otro lado, la arquitectura del middleware está diseñada con un tipo de comunicación *push asíncrona* [188] entre el nodo supervisor y el resto de nodos del sistema IoT, para lo cual utiliza como intermediario un broker. Específicamente, este tipo de comunicación permite que el nodo supervisor, tan pronto como detecte un cambio en los modelos en tiempo de ejecución, publique en el broker tanto la nueva versión de estos modelos como los planes de implementación y reconfiguración de los artefactos de monitoreo y

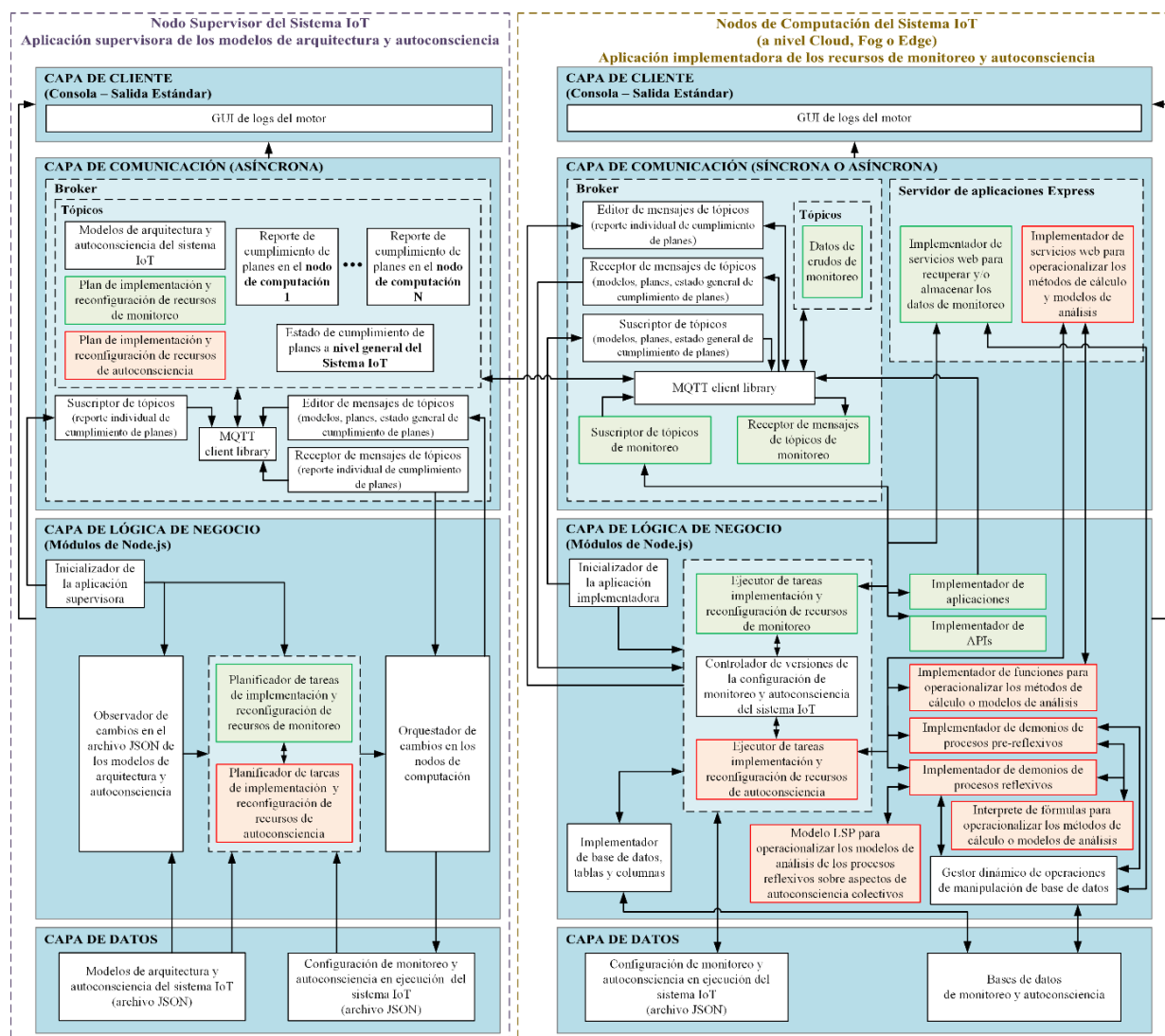


Figura 5-10. Arquitectura del middleware de la infraestructura de soporte. Fuente: Elaboración propia.

autoconsciencia, a fin de que todos los nodos del sistema IoT puedan obtenerlos para su respectiva implementación. De igual manera, la información de control sobre el estado de cumplimiento de los planes en cada uno de los nodos del sistema IoT es compartida por medio del broker.

En este caso particular, al tratarse de sistemas IoT altamente escalables, los cuales pueden estar compuestos por una gran cantidad de nodos de computación con diversas capacidades de procesamiento, ancho de banda y energía, la comunicación *push* es idónea para simplificar el intercambio de datos entre estos nodos, reducir el tráfico de red y la carga del nodo supervisor, ya que este último enviará (publicará) las nuevas versiones de los modelos y planes, únicamente cada que cambien estos modelos en tiempo de ejecución.

En la Figura 5-10 se presenta la arquitectura de las aplicaciones supervisora e implementadora del middleware, incluyendo sus capas, componentes e interfaces de comunicación entre estos. Los componentes compartidos por los dos motores se presentan con un rectángulo de color blanco; mientras que, los componentes propios del motor de recursos de monitoreo con un rectángulo de color verde; y, los componentes pertenecientes al motor de recursos de autoconsciencia con un rectángulo de color rojo. A continuación, se explica la funcionalidad de cada una de las capas del middleware y sus componentes:

- **Capa de cliente.** Representa la interfaz de usuario del middleware y sus motores. En este sentido, por medio de la consola o salida estándar se presentan permanentemente al usuario, los logs resultantes de la ejecución de las aplicaciones supervisora e implementadora del middleware. Específicamente, la aplicación implementadora presenta únicamente los logs de las tareas ejecutadas localmente en el nodo de computación, mientras que, la aplicación supervisora presenta un reporte global de las tareas ejecutadas en todos los nodos de computación del sistema IoT.
- **Capa de comunicación.** El objetivo principal de esta capa es implementar y desplegar los artefactos de software (servicios web, tópicos de broker) especificados en los modelos de arquitectura y autoconsciencia que soportan la comunicación síncrona o asíncrona entre los nodos de computación del sistema IoT, ubicados en las capas Edge, Fog y Cloud. Adicionalmente, esta capa contiene un conjunto de componentes que gestionan el intercambio de información sobre los planes de implementación y reconfiguración de los artefactos de software entre el nodo supervisor y el resto de nodos de computación, con el apoyo de un broker (comunicación *push asíncrona*). Para ello, la arquitectura del middleware contempla varios tópicos de broker que comparten la siguiente información: i) la nueva versión de los modelos y los planes son publicados en único tópico, con el objeto de simplificar los algoritmos de intercambio de información entre los nodos de computación; ii) los reportes individuales de cumplimiento de los planes son reenviados al nodo supervisor mediante tópicos independientes, uno para cada nodo de computación; y, iii) el estado de cumplimiento de los planes a nivel general del sistema IoT es publicado en un tópico adicional. Para el caso particular de la plataforma Node.js, esta capa ejecuta el servidor aplicaciones Express de Node.js [176], en el cual se publican servicios web para el intercambio de datos directo entre los nodos de computación (comunicación síncrona). A su vez, esta capa reutiliza la librería cliente del protocolo MQTT (MQTT.js) [189], a fin de gestionar los servicios de mensajería del broker (comunicación asíncrona). En la Tabla 5-1 se describe detalladamente la funcionalidad de cada uno de los componentes incluidos en esta capa.

Tabla 5-1. Componentes de la capa de comunicación (síncrona o asíncrona).

Componente	Funcionalidad	Alcance
Aplicación Supervisora		
Suscriptor de tópicos (reportes individuales de cumplimiento de planes).	Módulo que contiene métodos para establecer la conexión a un broker; y, suscribir el nodo supervisor a los tópicos que publican los reportes individuales sobre el cumplimiento de los planes de implementación y reconfiguración de los artefactos de software en cada uno de los nodos de computación.	Ambos motores.
Editor de mensajes de tópicos (modelos, planes, estado de cumplimiento general de planes).	Módulo que contiene métodos para que el nodo supervisor publique: la nueva versión detectada de los modelos de arquitectura y autoconsciencia, los planes de implementación y reconfiguración de los recursos de monitoreo y autoconsciencia; y, el estado de cumplimiento de los planes a nivel general del sistema IoT.	Ambos motores.
Receptor de mensajes de tópicos (reporte individual de cumplimiento de planes)	Evento que se dispara en el nodo supervisor, cada vez que nuevos reportes individuales de cumplimiento de los planes de implementación y reconfiguración de los artefactos de software son publicados por los nodos de computación del sistema IoT.	Ambos motores.
Aplicación Implementadora		
Suscriptor de tópicos (modelos, planes, estado de cumplimiento general de planes).	Módulo que contiene métodos para establecer la conexión a un broker; y, suscribir el nodo de computación en el que se ejecuta la aplicación implementadora a los tópicos que publican: la nueva versión detectada de los modelos de arquitectura y autoconsciencia, los planes de implementación y reconfiguración de los recursos de monitoreo y autoconsciencia; y, el estado de cumplimiento de los planes a nivel general del sistema IoT.	Ambos motores.
Receptor de mensajes de tópicos (modelos, planes, estado de cumplimiento general de planes).	Evento que se dispara en los nodos de computación del sistema IoT, cada vez que el nodo supervisor publica: una nueva versión de los modelos de arquitectura y autoconsciencia, los planes de implementación y reconfiguración de los recursos de monitoreo y autoconsciencia; y, el estado de cumplimiento de los planes a nivel general del sistema IoT. A su vez, este evento tras recibir nuevos mensajes en los tópicos, invoca a los métodos de la capa lógica de negocio, encargados de ejecutar y controlar las tareas de implementación y reconfiguración de los artefactos de monitoreo y autoconsciencia.	Ambos motores.
Editor de mensajes de tópicos (reporte individual de cumplimiento de planes).	Módulo que contiene un método para que los nodos de computación del sistema IoT publiquen sus reportes individuales sobre el cumplimiento de los planes de implementación y reconfiguración de los artefactos de software.	Ambos motores.
Implementador de servicios web para recuperar y/o almacenar los datos de monitoreo.	Componente encargado de implementar, reconfigurar y publicar dinámicamente los servicios web que facilitan el intercambio de datos de monitoreo entre los nodos Edge, Fog o Cloud de un sistema IoT. A su vez, estos servicios web pueden contemplar instrucciones para procesar y almacenar los datos recolectados (con diferentes niveles de agregación) y sus metadatos en las bases de datos de monitoreo del sistema IoT, o para recuperarlos posteriormente. En este punto, se debe precisar que este componente construye los servicios web, de tal forma que se oculte la lógica programación relacionada con las operaciones de manipulación de las bases de datos, mediante la llamada a métodos de la capa de lógica de negocio que se encargan propiamente de encapsular y ejecutar estas operaciones. A continuación, se describe cómo se construyen automáticamente los servicios web mediante la información disponible en las instancias de las metaclases del modelo de arquitectura: • La URL del servicio web es construida mediante la metaclase <i>Service</i> del modelo de arquitectura. Esta metaclase contiene como atributos, el endpoint y tipo de método (GET, POST, etc.) del servicio web. Particularmente, en la plataforma Node.js se utiliza el middleware Router de Express para crear dinámicamente los endpoints o enrutadores a los servicios web [176]. • El cuerpo del servicio web, dependiendo del tipo de flujo de datos (recolección o agregación) que soporta, puede contener: i) instrucciones para extraer los datos de monitoreo (a ser almacenados) de la solicitud http del servicio web.; ii) instrucciones para invocar a los métodos encargados de recuperar o almacenar los datos de monitoreo desde o en las bases de datos del sistema IoT; e, iii) instrucciones para enviar la respuesta http del servicio web, pudiendo contener los datos de monitoreo recuperados desde las bases de datos	Motor de recursos de monitoreo.

	o errores presentados en la ejecución. En la plataforma Node.js, estas instrucciones son encapsuladas en controladores (funciones), los cuales son invocados desde los enrutadores de los servicios web. • La estructura de los datos de la solicitud y respuesta http de un servicio web se encuentran especificadas en las instancias de las metaclasses <i>Parameter</i> y <i>ReturnVariable</i>, respectivamente. A su vez, por medio de las relaciones entre las metaclasses <i>PropertyTemplate</i>, <i>Property</i>, <i>Parameter</i> y <i>ReturnVariable</i>, se puede determinar a qué propiedades de las entidades IoT hacen referencia los datos de monitoreo contenidos en las solicitudes y respuestas de los servicios web. • Los métodos invocados desde los controladores de los servicios web encapsulan la lógica de programación para recuperar o almacenar los datos de monitoreo desde o en las bases de datos. Estos métodos forman parte del componente “Gestor dinámico de operaciones de manipulación de base de datos”, ubicado en la capa de lógica de negocio de la infraestructura del middleware, la cual se describe más adelante. En términos generales, para la construcción de estos métodos se utilizan las instancias de las metaclasses <i>DataFlow</i>, <i>LinkServiceToDatatable</i>, <i>DataBase</i>, <i>DataTable</i>, <i>DataColumn</i>, <i>PropertyToDataColumn</i> y <i>DataColumnToDataColumn</i>.	
Suscriptor de tópicos de monitoreo.	Módulo que contiene métodos para establecer la conexión a un broker; y, suscribir nuevos tópicos de monitoreo o cancelar los existentes, según las especificaciones de la última versión del modelo de arquitectura de monitoreo. Las instancias de las metaclasses del modelo que son utilizadas en la lógica de programación de este módulo son: <i>ComputingNode</i> , <i>NetworkInterface</i> , <i>Broker</i> , <i>Protocol</i> y <i>Topic</i> .	Motor de recursos de monitoreo.
Receptor de mensajes de tópicos de monitoreo.	Evento asociado a un broker que se dispara en los nodos de computación, cada vez que se publican nuevos mensajes (datos crudos recolectados) en los tópicos de monitoreo a los cuales dichos nodos están suscritos. A su vez, este evento incluye instrucciones para consumir servicios web que guardan los datos de los tópicos en las correspondientes bases de datos de monitoreo. En la lógica de programación de este evento se utilizan las siguientes metaclasses del modelo de arquitectura: <i>Broker</i> , <i>Topic</i> , <i>LinkServiceToTopic</i> , <i>Service</i> , <i>ComputingNode</i> , <i>Middleware</i> , <i>NetworkInterface</i> , <i>Protocol</i> , <i>Header</i> , <i>Parameter</i> , <i>ReturnVariable</i> , <i>PropertyTemplate</i> y <i>Property</i> .	Motor de recursos de monitoreo.
Implementador de servicios web para operacionalizar los métodos de cálculo y modelos de análisis.	Este componente tiene como propósito crear o reconfigurar los recursos de implementación de tipo servicios web, los cuales encapsulan un conjunto de instrucciones para calcular las métricas indirectas o indicadores asociados a los métodos de cálculo o modelos de análisis, respectivamente. En este sentido, la lógica de programación de estos servicios web se estructura de forma similar a los servicios web que soportan el monitoreo. A continuación, se explica cómo se implementan los diferentes elementos de estos servicios: • La URL del servicio web es construida mediante los atributos endpoint y tipo de método de la metaclass <i>Service</i> del modelo de autoconsciencia. Asimismo, se utiliza el middleware Router de Express para crear dinámicamente los endpoints o enrutadores a los servicios web [176]. • El cuerpo del servicio web incluye: i) instrucciones para extraer los datos de la solicitud http, mismos que son utilizados para el cálculo de la métrica; ii) instrucciones de llamada a una función que encapsula el código del cálculo de la métrica; e, iii) instrucciones para enviar la respuesta http del servicio web, pudiendo ser el valor de métrica calculada o errores presentados durante la ejecución. De igual manera, en la plataforma Node.js, estas instrucciones son encapsuladas en controladores (funciones), mismos que son invocados desde los enrutadores de los servicios web. • Los datos de la solicitud http del servicio web son extraídos según las especificaciones contenidas en la metaclass <i>Parameter</i>. Mientras que, el resultado de la métrica se estructura en la salida http, según los atributos <i>returnDataType</i> y <i>returnDataFormatType</i> de las metaclasses <i>ImplementationResource</i> y <i>Service</i>, respectivamente. • Las funciones invocadas desde los servicios web y que encapsulan las instrucciones de cálculo de las métricas son	Motor de recursos de autoconsciencia.

	creadas mediante el componente “ <i>Implementador de funciones para operacionalizar los métodos de cálculo o modelos de análisis.</i> ”, ubicado en la capa de lógica de negocio. Este componente será analizado más adelante.	
--	--	--

- **Capa de lógica de negocio.** Contiene un conjunto de módulos y métodos en Node.js distribuidos a través de las aplicaciones supervisora e implementadora que proporcionan las siguientes funcionalidades: i) inicializar y parametrizar la operación de los motores del middleware; ii) observar y detectar cambios en el archivo JSON de los modelos en tiempo de ejecución; iii) comparar los modelos y la configuración en operación del sistema IoT; iv) planificar, ejecutar y controlar las tareas de implementación y/o reconfiguración de los artefactos de software de monitoreo y autoconsciencia. En la Tabla 5-2 se detallan cada uno de los componentes o módulos incluidos en esta capa.

Tabla 5-2. Componentes de la capa de lógica de negocio.

Componente	Funcionalidad	Alcance
Aplicación Supervisora		
Inicializador de la aplicación supervisora.	Programa principal que inicia los motores del middleware en el nodo supervisor, según los parámetros establecidos por el usuario. En la línea de comandos, al momento de llamar a este programa, el usuario debe establecer los siguientes parámetros: i) modo de operación del middleware, es decir, si ejecuta únicamente el motor de sincronización de recursos de monitoreo; o, por el contrario, si operan los dos motores en conjunto; ii) directorio de ubicación del archivo JSON que contiene los modelos de arquitectura y autoconsciencia en tiempo de ejecución; y, iii) dirección IP y puerto del broker utilizado como intermediario para intercambiar la información con el resto de nodos de computación del sistema IoT. En el caso de no indicar el último parámetro, la aplicación asume por defecto que el broker se encuentra instalado en el nodo supervisor, puerto 1883.	Ambos motores.
Observador de cambios en el archivo JSON de los modelos de arquitectura y autoconsciencia.	Componente encargado de observar y detectar cualquier cambio en el archivo JSON que contiene los modelos de arquitectura y autoconsciencia en tiempo de ejecución. Particularmente, en Node.js se utiliza la API <i>watch</i> de la librería <i>chokidar</i> [190], con el objeto de crear un agente asíncrono que observe los cambios en el directorio que contiene el archivo JSON. Los cambios que detecta pueden ser: agregar por primera vez el archivo JSON en el directorio; o, modificar / eliminar un archivo JSON existente.	Ambos motores.
Planificador de tareas de implementación y reconfiguración de recursos de monitoreo.	En primera instancia, este componente se encarga de comparar la nueva versión detectada del modelo de arquitectura de monitoreo con la configuración de monitoreo en ejecución del sistema IoT, con el propósito de identificar diferencias en las especificaciones técnicas de los artefactos de software de monitoreo. Luego, a partir de estas diferencias se planifican las tareas de creación, modificación o eliminación de los artefactos de software, tales como: APIs, aplicaciones, servicios web, tópicos de brokers, bases de datos de monitoreo, incluyendo sus tablas y columnas. Como resultado, se obtiene un plan de implementación y reconfiguración de los artefactos de software de monitoreo, cuyas tareas se encuentran agrupadas por nodo de computación.	Motor de recursos de monitoreo.
Planificador de tareas de implementación y reconfiguración de recursos de autoconsciencia.	Este componente tiene una funcionalidad similar al antes descrito, con la diferencia que la comparación se realiza entre la nueva versión detectada del modelo de autoconsciencia y la configuración de autoconsciencia en operación del sistema IoT. Asimismo, con base en las diferencias encontradas en la comparativa, se planifican las tareas de creación, modificación o eliminación de los artefactos de software de autoconsciencia, tales como: demonios de los procesos de autoconsciencia, funciones y servicios web que operacionalizan los métodos de autoconsciencia, bases de datos y tablas que almacenan las métricas, metadatos y recomendaciones resultantes de la ejecución de los procesos de autoconsciencia. Como resultado, se obtiene un plan de implementación y reconfiguración de los artefactos de	Motor de recursos de autoconsciencia.

	software de autoconsciencia, cuyas tareas se encuentran agrupadas por nodo de computación.	
Orquestador de cambios en los nodos de computación.	Este módulo tiene como función principal la distribución y control de los planes de implementación y reconfiguración de los artefactos de software, a nivel de todas las capas (Edge, Fog, Cloud) y nodos de computación de la infraestructura IoT. A continuación, se enumeran sus principales tareas: i) invocar a los métodos encargados de publicar en los respectivos tópicos del broker, la nueva versión de los modelos de arquitectura y autoconsciencia; y, los planes de implementación y reconfiguración de los artefactos de software; ii) invocar al método que suscribe el nodo supervisor a los tópicos que publican los reportes individuales de cumplimiento de los planes; iii) analizar los reportes individuales de cumplimiento de los planes recibidos desde los nodos de computación, conforme a las directrices descritas en las subsecciones anteriores (5.4.1 y 5.4.2), a fin de determinar si la nueva versión de los modelos se establece como la nueva configuración de monitoreo y autoconsciencia en ejecución del sistema IoT; o, en su defecto, se retoma a la última configuración válida del sistema IoT; y; iv) llamar al método encargado de publicar en el tópico correspondiente, el estado de cumplimiento general de los planes; de tal forma que, coordinada y coherentemente se establezca la misma configuración en todos los nodos de computación.	Ambos motores.
Aplicación Implementadora		
Inicializador de la aplicación implementadora.	Programa principal que inicia la aplicación implementadora con las funcionalidades de los motores, a nivel de los nodos de computación del sistema IoT. Entre las principales tareas que ejecuta este componente se tienen: i) inicializar el servidor de aplicaciones Express de Node.js, en el cual se publican los servicios web requeridos por los procesos de monitoreo y autoconsciencia; y, ii) llamar al módulo encargado de suscribir el nodo de computación a los tópicos de broker que publican los modelos de arquitectura y autoconsciencia en tiempo de ejecución, los planes de implementación y reconfiguración de los recursos de software; y, el estado de cumplimiento de estos planes a nivel general del sistema IoT. Por último, al momento de ejecutar la aplicación implementadora, el usuario debe establecer en la línea de comando los siguientes parámetros: dirección IP y puerto del broker utilizado como intermediario para intercambiar la información con el nodo supervisor.	Ambos motores.
Ejecutor de tareas implementación y reconfiguración de recursos de monitoreo.	Módulo que clasifica las tareas según el tipo de artefacto de software de monitoreo (API, aplicación, servicio web, tópico de broker, base de datos) a implementar o reconfigurar. Con base en esta clasificación, distribuye las tareas a los componentes encargados de implementar cada tipo de artefacto.	Motor de recursos de monitoreo.
Ejecutor de tareas implementación y reconfiguración de recursos de autoconsciencia.	Módulo que clasifica las tareas según el tipo de artefacto de software de autoconsciencia (demonios, funciones, servicios web, bases de datos) a implementar o reconfigurar. Con base en esta clasificación, distribuye las tareas a los componentes encargados de implementar cada tipo de artefacto.	Motor de recursos de autoconsciencia.
Implementador de APIs.	Este componente tiene como propósito implementar o reconfigurar las funciones APIs que encapsulan el código para recolectar los datos crudos desde los sensores u otras fuentes. Tanto la cabecera (nombre, parámetros de entrada, variables de salida) como el cuerpo (instrucciones) de las funciones APIs son creadas a partir de las especificaciones contenidas en las instancias de las metaclasses <i>API</i> , <i>Parameter</i> y <i>ReturnVariable</i> del modelo de arquitectura de monitoreo.	Motor de recursos de monitoreo.
Implementador de aplicaciones.	Componente responsable de implementar o reconfigurar las aplicaciones de software que encapsulan el código necesario para enrutar y controlar los flujos de datos entre las APIs, servicios y brokers. Al ser las aplicaciones componentes centrales de los flujos de datos, para la creación de su lógica de programación se requiere la información de una gran cantidad de instancias de las metaclasses del modelo de arquitectura. A continuación, se detallan las principales metaclasses, relaciones y atributos que se utilizan para crear las instrucciones de una aplicación: Los flujos de datos que controla una aplicación, así como las APIs, servicios web o brokers que intervienen en dichos	Motor de recursos de monitoreo.

	flujos son identificados a partir de las metaclasses <i>DataFlow</i> y <i>Link</i>. • Una aplicación puede controlar uno o varios flujos de datos. Por lo tanto, cada flujo de datos es un fragmento de código que se invoca periódicamente dentro de una aplicación. En particular, para implementar esta funcionalidad en Node.js, se utiliza el método <i>setInterval</i> [191]. Por medio de este método, se asigna una función asíncrona a cada flujo de datos, la misma que se ejecuta automáticamente cada cierto intervalo de tiempo. La frecuencia de tiempo con que se ejecuta la función asíncrona de un flujo de datos es obtenida de los atributos <i>unitOfTime</i>, <i>executionTimeInterval</i> y <i>executionTime</i> de la metaclass <i>DataFlow</i>. • El fragmento de código que se encapsula en cada función asíncrona asociada a un flujo de datos puede contener instrucciones para llamar a una API, consumir servicios web o publicar tópicos en un broker. • Las instrucciones para invocar a una API (nombre, parámetros de entrada, variables de salida) se implementan a partir de las metaclasses <i>API</i>, <i>Parameter</i>, <i>ReturnVariable</i>, <i>PropertyTemplate</i> y <i>Property</i>. • Las instrucciones para consumir un servicio web (endpoint, método, cabecera, parámetros de entrada, variables y tipo de formato de salida) se construyen mediante las metaclasses <i>ComputingNode</i>, <i>Middleware</i>, <i>NetworkInterface</i>, <i>Protocol</i>, <i>Service</i>, <i>Header</i>, <i>Parameter</i>, <i>ReturnVariable</i>, <i>PropertyTemplate</i> y <i>Property</i>. • Las instrucciones para publicar los datos recolectados en un tópico (nombre, estructura y tipo de formato del payload) de un broker determinado (dirección IP, protocolo, puerto) se implementan por medio de las metaclasses <i>Topic</i>, <i>ComputingNode</i>, <i>Broker</i>, <i>NetworkInterface</i>, <i>Protocol</i>, <i>PropertyTemplate</i> y <i>Property</i>. • La relación <i>previousLink</i> que define el orden de los enlaces que conforman un flujo de datos, permite construir la secuencia lógica de programación (orden de invocación de los artefactos de software) encapsulada en el cuerpo de la función asíncrona asociada al flujo de datos. • Asimismo, mediante las relaciones existentes entre las metaclasses <i>PropertyTemplate</i>, <i>Property</i>, <i>Parameter</i>, <i>ReturnVariable</i> y <i>Topic</i>, se puede determinar las propiedades de las entidades IoT que son utilizadas (metadatos) o monitoreadas (telemetría, estado) en los flujos de datos; y, cómo estas propiedades se relacionan con los parámetros de entrada y las variables de salida de las APIs o servicios web, así como con la estructura del payload de los tópicos. Con esta información se parametrizan las instrucciones contenidas en las funciones asíncronas, a fin de asegurar el intercambio coherente de los datos de las propiedades entre los artefactos de software (APIs, servicios web, brokers).	
Implementador de funciones para operacionalizar los métodos de cálculo o modelos de análisis.	Este módulo tiene como objetivo crear o reconfigurar el código los recursos de implementación de tipo función en tiempo de ejecución. Asimismo, estas funciones encapsulan una secuencia de instrucciones para calcular las métricas indirectas o indicadores asociados a los métodos de cálculo o modelos de análisis, respectivamente. La cabecera (nombre, parámetros de entrada, variables de salida) y cuerpo (instrucciones) de las funciones son creadas mediante la información de las instancias de las metaclasses <i>Function</i> y <i>Parameter</i> del modelo de autoconsciencia.	Motor de recursos de autoconsciencia.
Intérprete de fórmulas para operacionalizar los métodos de cálculo o modelos de análisis.	Módulo encargado de analizar (sintácticamente), interpretar y resolver los recursos de implementación de tipo fórmula asociados a los métodos de cálculo y modelos de análisis de los procesos de autoconsciencia. En particular, este módulo tiene la capacidad de resolver fórmulas con varias variables (parámetros), operaciones matemáticas y funciones de agregación (suma, promedio, desviación estándar, etc.). A su vez, este módulo provee un método principal que recibe como parámetros de entrada: una cadena con la expresión de la fórmula; y, los valores de las variables de la fórmula. Finalmente, el método devuelve el valor resultante de la resolución de la fórmula. Finalmente, la lógica de	Motor de recursos de autoconsciencia.

	programación de este módulo utiliza la información de las instancias de las metaclasses <i>Formula</i> y <i>Parameter</i> del modelo de autoconsciencia.	
Modelo LSP para operacionalizar los modelos de análisis de los procesos reflexivos relacionados con aspectos de autoconsciencia colectivos.	Componente que implementa el algoritmo del Modelo de Agregación Lógica de Preferencias - LSP [170] para calcular los indicadores cuantitativos de los aspectos de autoconsciencia colectivos. Estos indicadores se calculan mediante operaciones de agregación ponderadas entre los indicadores obtenidos para los aspectos de autoconsciencia relacionados de más bajo nivel. Para ello, este módulo proporciona un método principal que recibe como parámetro de entrada, el identificador del aspecto colectivo a calcular; y, devuelve como salida, el indicador porcentual obtenido por el modelo LSP para el aspecto colectivo en cuestión. Además, el algoritmo del Modelo LSP hace uso de las especificaciones del modelo de autoconsciencia, en lo referente a la estructura jerárquica, operadores de agregación y pesos de los aspectos de autoconsciencia.	Motor de recursos de autoconsciencia.
Implementador de demonios de procesos pre-reflexivos.	Este componente implementa o reconfigura los demonios que operacionalizan los procesos pre-reflexivos. A continuación, se detalla cómo se construyen estos demonios, con base en las especificaciones contenidas en el modelo de autoconsciencia: • Los demonios deben ejecutarse periódicamente en segundo plano. De ahí que, bajo la plataforma Node.js, se utiliza el método <i>setInterval</i> [191] para crear una función asíncrona por cada demonio. Estas funciones asíncronas contienen la secuencia de instrucciones de los procesos pre-reflexivos, mismas que se ejecutan repetidamente, cada cierto intervalo de tiempo. La frecuencia de tiempo con que se ejecutan los demonios es obtenida de los atributos <i>unitOfTime</i>, <i>executionTimeInterval</i> y <i>executionTime</i> de la metaclass <i>SelfAwarenessProcess</i>. • Un demonio para un proceso pre-reflexivo incluye instrucciones para operacionalizar un método de recolección, un modelo de análisis; y, finalmente almacenar toda la información resultante en las bases de datos de autoconsciencia. • La lógica de programación del método de recolección contiene instrucciones para producir una métrica directa, a partir de los datos crudos obtenidos por los flujos de recolección de datos y que se encuentran almacenados en las bases de datos de monitoreo. Principalmente, estas instrucciones se implementan mediante la información de las instancias de las metaclasses <i>PreReflexiveProcess</i>, <i>CollectionMethod</i>, <i>DirectMetric</i>, <i>DataFlow</i>, <i>Property</i>, <i>PropertyToDataColumn</i>, <i>DataColumn</i>, <i>DataTable</i> y <i>DataBase</i>. • La lógica de programación del modelo de análisis contempla instrucciones de llamada a un recurso de implementación (fórmula, función o servicio web) para transformar la métrica directa en un indicador cuantitativo. Para el caso de los recursos de implementación de tipo fórmula, se invoca al método encargado de interpretar y resolver las fórmulas. Adicionalmente, incluye instrucciones para comparar el indicador cuantitativo con los umbrales de un criterio de decisión, de tal forma que se obtenga un indicador cualitativo y las recomendaciones prescriptivas asociadas. Entre las principales metaclasses que se utilizan para implementar las instrucciones del modelo de análisis se tienen: <i>PreReflexiveProcess</i>, <i>AnalysisModel</i>, <i>Indicator</i>, <i>Formula</i>, <i>Function</i>, <i>Service</i>, <i>Parameter</i>, <i>MetricToParameterMapping</i>, <i>DecisionCriteria</i>, <i>Threshold</i> y <i>Action</i>. Por su parte, la metaclass <i>MetricToParameterMapping</i> permite parametrizar la instrucción de llamada al recurso de implementación asociado al modelo de análisis; es decir, pasar como parámetros al recurso de implementación, los valores de las métricas a ser utilizadas en el cálculo del indicador. • Finalmente, el código del demonio contiene instrucciones para invocar a un método del componente “<i>Gestor dinámico de operaciones de manipulación de base de datos</i>”, a fin de almacenar la métrica directa, el indicador cuantitativo y cualitativo (su interpretación), los metadatos; y, las recomendaciones prescriptivas.	Motor de recursos de autoconsciencia.

Implementador de demonios de procesos reflexivos.	Este componente implementa o reconfigura los demonios que operacionalizan los procesos reflexivos. La forma como se construyen estos demonios es similar al de los procesos pre-reflexivos, misma que se detalla a continuación: • Los demonios reflexivos también se implementan como funciones asíncronas mediante el método <i>setInterval</i> [191]. • Por su parte, un demonio para un proceso reflexivo incluye instrucciones para operacionalizar un método de cálculo, un modelo de análisis; y, finalmente almacenar toda la información resultante en las bases de datos de autoconsciencia. • La lógica de programación del método de cálculo incluye instrucciones de llamada a un recurso de implementación (fórmula, función o servicio web) para producir una métrica indirecta, a partir de los datos históricos de otras métricas, metadatos o variables de simulación. Asimismo, para el caso de los recursos de implementación de tipo fórmula, se invoca al método encargado de interpretar y resolver las fórmulas. Entre las principales metaclases que se utilizan para implementar las instrucciones del método de cálculo se tienen: <i>ReflexiveProcess</i>, <i>CalculationMethod</i>, <i>IndirectMetric</i>, <i>SimulationScenario</i>, <i>SimulationVariable</i>, <i>SimulationValue</i>, <i>Formula</i>, <i>Function</i>, <i>Service</i>, <i>Parameter</i>, <i>MetaDataToParameterMapping</i>, <i>MetricToParameterMapping</i> y <i>SimulationVariableToParameterMapping</i>. En particular, las últimas tres metaclases citadas, permiten parametrizar la instrucción de llamada al recurso de implementación asociado al método de cálculo; es decir, pasar como parámetros al recurso de implementación, los valores de los metadatos, métricas y/o variables de simulación requeridos para el cálculo de la métrica indirecta. • La lógica de programación del modelo de análisis es idéntica al de los demonios pre-reflexivos. Con la única variante que, para el caso de los demonios sobre aspectos de autoconsciencia colectivos, se añaden instrucciones para invocar al método principal del componente LSP, a fin de calcular directamente los respectivos indicadores cuantitativos, a partir de aplicar operaciones de agregación sobre los indicadores de aspectos relacionados de más bajo nivel. • Finalmente, estos demonios también incluyen instrucciones para invocar a un método del componente “Gestor dinámico de operaciones de manipulación de base de datos”, a fin de almacenar la métrica indirecta, el indicador cuantitativo y cualitativo (su interpretación), los metadatos, los escenarios, variables y valores de simulación utilizados; y, las recomendaciones prescriptivas.	Motor de recursos de autoconsciencia.
Implementador de base de datos, tablas y columnas.	Módulo que dispone de métodos para mantener la estructura de las bases de datos de monitoreo y autoconsciencia, de tal forma que sean conformes con la última versión de los modelos. Particularmente, este módulo incluye métodos para: i) crear bases de datos; ii) crear tablas; y; iii) añadir, alterar (tipo de dato) o eliminar una columna. Los métodos construyen y ejecutan sentencias de definición de datos en SQL, según la información contenida en las instancias de las metaclases <i>DataBase</i>, <i>DataTable</i> y <i>DataColumn</i> del modelo de arquitectura. En el caso de que el plan de reconfiguración incluya tareas de eliminación de columnas, este componente incluye métodos para respaldar previamente la base de datos en cuestión. En este punto, es necesario indicar que, la tabla que almacena la información de las métricas resultantes de los procesos de autoconsciencia, siempre tiene el mismo nombre (“<i>MetricaAutoconsciencia</i>”) y una estructura fija, conformada por las siguientes columnas: <i>idMetrica</i>, <i>tipoMetrica</i>, <i>valorMetrica</i>, <i>fechaHora</i>, <i>idProceso</i>, <i>idSujeto</i>, <i>idObjeto</i>, <i>idAspecto</i>, <i>umbralIndicador</i>, <i>interpretacionIndicador</i>, <i>acciones</i>, <i>idEscenarioSimulacion</i>, <i>nombreVariableSimulacion</i> y <i>valorVariableSimulacion</i>. Sin embargo, la relación <i>storesMetricIn</i> entre las metaclases <i>SelfAwarenessProcess</i> y <i>Database</i>, define la base de datos en donde debe ser creada esta tabla para almacenar los resultados de un determinado proceso de autoconsciencia.	Ambos motores.

Gestor dinámico de operaciones de manipulación de base de datos.	Este componente dispone de métodos de acceso y manipulación de las bases de datos de monitoreo y autoconsciencia. A continuación, se describen los principales métodos y cómo se construyen mediante las especificaciones de los modelos en tiempo de ejecución. • Métodos para establecer la conexión con las bases de datos especificadas en el modelo de arquitectura. Para ello, se utiliza la información de las instancias de las metACLases <i>ComputingNode</i>, <i>Database</i>, <i>NetworkInterface</i> y <i>Protocol</i>. • Métodos para recuperar o almacenar los datos desde o en las bases de datos de monitoreo. Estos métodos incluyen instrucciones para construir y ejecutar sentencias SQL (Select, Insert) con base en las especificaciones del modelo de arquitectura. Las sentencias SQL pueden manipular columnas que almacenan datos de telemetría, estado y metadatos. Los datos de telemetría o estado se transportan a través de todos los artefactos de monitoreo (APIs, aplicaciones, servicios web, brokers). Mientras que, los metadatos son obtenidos a partir de fórmulas de expresión (p. ej., la fecha y hora de recolección) o de datos establecidos en el modelo de arquitectura durante el tiempo de diseño (p. ej., número de serie de un sensor). Además, las consultas SQL pueden incluir columnas de agrupación (Group by), funciones de agregación (SUM, AVG, COUNT, etc.), condicionales (Where y Having) con el objeto de soportar los flujos de agregación de datos. El código de este tipo de métodos utiliza las instancias de las metACLases <i>DataFlow</i>, <i>LinkServiceToDatatable</i>, <i>DataBase</i>, <i>DataTable</i>, <i>DataColumn</i>, <i>PropertyTo DataColumn</i> y <i>DataColumnToDataColumn</i> del modelo de arquitectura. • Métodos para almacenar las métricas en las bases de datos de autoconsciencia. De igual forma, estos métodos incluyen instrucciones para construir y ejecutar sentencias SQL (Insert) para almacenar la información sobre las métricas directas, indirectas e indicadores en las correspondientes bases de datos asignadas a los procesos de autoconsciencia. Para ello, la lógica de programación de estos métodos utiliza instancias de las metACLases <i>SelfAwarenessProcess</i> y <i>DataBase</i> de los modelos en tiempo de ejecución.	Ambos motores.
Controlador de versiones de la configuración de monitoreo y autoconsciencia del sistema IoT.	Este controlador de versiones se encarga de validar que la misma configuración de monitoreo y autoconsciencia se ejecute en todos los nodos de computación del sistema IoT. Para ello, el controlador utiliza la información publicada por el nodo supervisor sobre el estado de cumplimiento de los planes de implementación y reconfiguración, a nivel general del sistema. En este sentido, si los planes se han ejecutado satisfactoriamente en todos los nodos de computación, la nueva versión de los modelos de arquitectura y autoconsciencia se establece como la configuración en ejecución del sistema IoT; caso contrario, se retorna a la última configuración en operación válida del sistema IoT.	Ambos motores.

- **Capa de datos.** Integrada por tres componentes. En primer lugar, un archivo JSON con los modelos de arquitectura y autoconsciencia del sistema IoT, el cual es permanentemente observado por la aplicación supervisora para detectar cambios en sus especificaciones. Estos cambios pueden provenir de los ajustes que realiza el equipo de desarrollo a los modelos a través del DSL Monitor-IoT y el configurador web de autoconsciencia. En segundo lugar, un archivo JSON con las especificaciones de la configuración de monitoreo y autoconsciencia en ejecución del sistema IoT. De igual manera, este archivo de configuración es conforme a los metamodelos Monitor-IoT y Aware-IoT. De ahí que, los dos archivos JSON comparten una misma estructura, siendo posible compararlos entre sí mediante la aplicación supervisora, a fin de identificar cambios en sus especificaciones que requieran de la ejecución de tareas de actualización de los artefactos de software de monitoreo y autoconsciencia del sistema IoT; y, de esta forma, mantener la relación causal entre los modelos y el sistema observado. Finalmente, el tercer componente son las bases de datos que

almacenan los datos resultantes de los procesos de monitoreo y autoconsciencia del sistema IoT. Las bases de datos de monitoreo guardan los datos recolectados (con diferentes niveles de agregación) y sus metadatos; mientras que, las bases de datos de autoconsciencia guardan las métricas directas, indirectas e indicadores de los aspectos de autoconsciencia, sus metadatos y las recomendaciones prescriptivas. Estas bases de datos pueden estar distribuidas en varios nodos de computación del sistema IoT.

5.5 Visor web de autoconsciencia

A nivel del front-end de la infraestructura de autoconsciencia, se dispone de una aplicación web para visualizar la información sobre las métricas resultantes de los procesos de autoconsciencia pre-reflexivos y reflexivos. Específicamente, esta aplicación ofrece las siguientes funcionalidades (ver Fig. 5-11):

- Reportar en tiempo real el estado de un determinado aspecto de autoconsciencia. Este reporte visualiza de forma tabular y gráfica, los valores que obtiene una métrica (directa, indirecta o indicador) cada cierto intervalo de tiempo. Con respecto a las métricas de tipo indicador, adicional a su valoración cuantitativa, se incluye su valoración cualitativa e interpretación de la misma, así como las recomendaciones o acciones prescriptivas, en caso de existir.
- Visualizar los datos históricos de una métrica durante un periodo de tiempo determinado (rango de fechas). Estos reportes históricos presentan la misma información que los reportes en tiempo real.
- Presentar informes predictivos (simulación) sobre la posible evolución de un aspecto de autoconsciencia, considerando los escenarios, variables y valores de simulación disponibles en el modelo de autoconsciencia.
- Las métricas disponibles para su presentación en el visor pueden ser filtradas por: sujeto, objeto y aspectos de autoconsciencia. Los objetos de autoconsciencia se presentan en un árbol, de conformidad con la estructura jerárquica establecida para las entidades IoT en el modelo de arquitectura.

En la Figura 5-11, a manera de ejemplo, se presenta en el visor web de autoconsciencia, un reporte del indicador que evalúa el nivel de riesgo de la concentración de monóxido de carbono en el ambiente (aspecto) de una cocina (objeto), el cual es obtenido mediante la ejecución de un proceso pre-reflexivo de un Sistema de Control Ambiental (sujeto). Además, en el reporte se muestra la correspondiente valoración e interpretación cualitativa del indicador; y, las recomendaciones prescriptivas de ser necesario.

5.6 Estado actual y perspectivas futuras de ampliación de la infraestructura de soporte

Como resultado de la tesis doctoral, se ha implementado un prototipo avanzado de la infraestructura de soporte, con el propósito de hacer factible la evaluación empírica, mediante cuasi experimentos, de todas las actividades de la metodología Aware-IoT@Run.Time. Específicamente, este prototipo incluye un conjunto de funcionalidades para diseñar y construir los modelos de arquitectura y autoconsciencia de un sistema IoT; y, a partir de estos modelos, implementar y mantener automáticamente los recursos de software que soportan los procesos de monitoreo y autoconsciencia (pre-reflexivos y reflexivos) de un sistema IoT. Sin embargo, el alcance del prototipo se ha enmarcado únicamente en los aspectos de autoconsciencia individuales; postergando las funcionalidades relacionadas con los aspectos colectivos para ser implementadas en trabajos futuros.

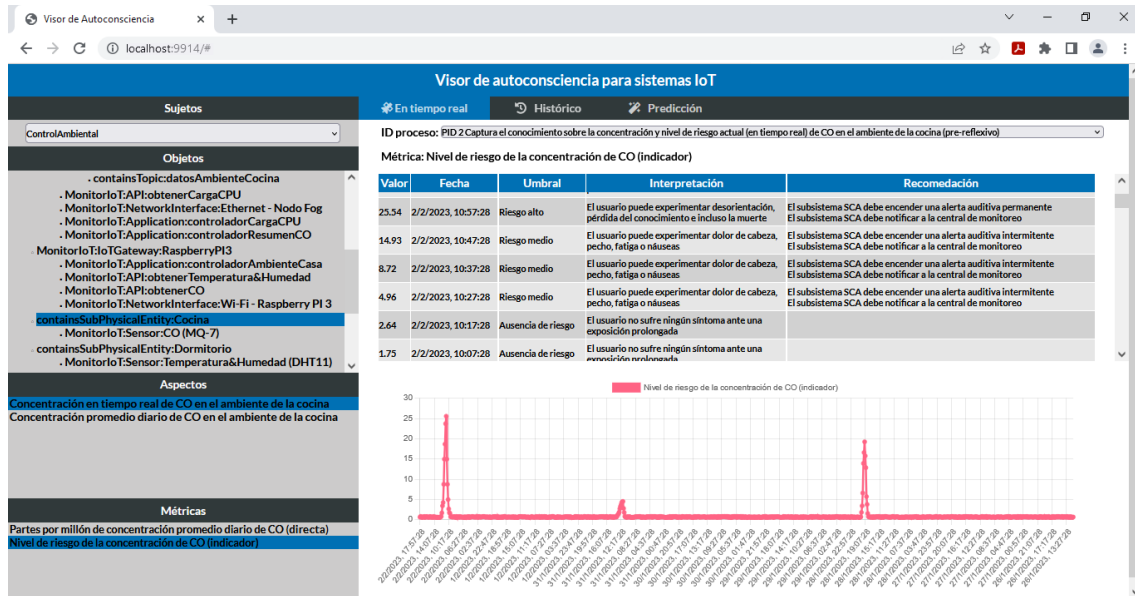


Figura 5-11. Visor web de autoconsciencia para sistemas IoT.

Fuente: Elaboración propia.

Adicionalmente, existen importantes oportunidades a corto, mediano y largo plazo para ampliar y mejorar las funcionalidades de la infraestructura de soporte; y, en general, el enfoque metodológico Aware-IoT@Run.Time. A continuación, se enlistan las principales tareas a ser consideradas en proyectos futuros:

- Completar la implementación de las funcionalidades de la infraestructura de soporte, tales como: el formulario de definición de la estructura jerárquica de los aspectos colectivos, como parte del configurador web de autoconsciencia; el módulo LSP para la evaluación de los aspectos colectivos, el intérprete de fórmulas para operacionalizar los métodos de cálculo o modelos de análisis, entre los principales.
- Construir un DSL visual para modelar gráficamente las capacidades de autoconsciencia de un sistema IoT y compararlo con el configurador web de autoconsciencia existente en la actualidad, en términos de usabilidad, funcionalidad y reutilización de componentes. Para ello, resulta indispensable profundizar en la investigación de herramientas para la construcción de DSLs orientados a megamodelos [192], los cuales permitan modelar visualmente por perspectivas o en capas las diferentes incumbencias (concerns) de un sistema IoT (p. ej., arquitectura, autoconsciencia, adaptación); y, a su vez, soporten la interrelación entre los conceptos modelados en las diferentes capas. Este requerimiento es fundamental para modularizar el modelado y facilitar su organización visual y entendimiento.
- La solución tecnológica propuesta aplica el aprendizaje de modelos basado en la parametrización, es decir, los datos monitoreados sobre el sistema IoT y su entorno son utilizados para actualizar, estimar y calibrar en tiempo de ejecución los parámetros (métricas, metadatos) de los modelos [154]. Si bien estos parámetros reflejan en todo momento el estado actual y la posible evolución del sistema IoT, manteniendo así la relación causal entre los modelos en tiempo de ejecución y el sistema IoT observado, existe la oportunidad de ir un paso más adelante, hacia el aprendizaje de modelos a nivel estructural [154]. En este sentido, uno de los desafíos a futuro es implementar nuevos componentes como parte de la infraestructura de soporte a la metodología Aware-IoT@Run.Time, a fin de dotar a los sistemas IoT de capacidades de automodelado [193], de manera que más allá de poder actualizar los parámetros de sus modelos, actúen como modeladores no humanos para cambiar la

estructura de sus modelos, según los eventos que ocurren durante la operación del propio sistema. Por ejemplo, agregar nuevas entidades IoT, propiedades, metadatos, flujos de datos, procesos de autoconsciencia, métricas, entre otros. Asimismo, la infraestructura de soporte deberá disponer de componentes adicionales que permitan a los sistemas IoT descubrir en tiempo real nuevas entidades IoT y sus propiedades; o, detectar entidades IoT que han sufrido cambios o dejado de operar.

- Ampliar el alcance de la infraestructura de soporte con el objeto de que sea capaz de soportar los procesos de actuación (adaptación) de un sistema IoT.

5.7 Conclusiones

A lo largo de este capítulo, se estudió a profundidad todos los componentes de la infraestructura de soporte a la metodología Aware-IoT@Run.Time. Esta infraestructura fue diseñada considerando a los modelos de arquitectura y autoconsciencia como componentes centrales de los procesos de implementación, operación y mantenimiento de los artefactos de software de monitorización y autoconsciencia de un sistema IoT, lo cual abstrae la complejidad y eleva la automatización de estos procesos.

Los metamodelos Monitor-IoT y Aware-IoT se construyeron en base a estándares, frameworks conceptuales y ontologías relevantes sobre arquitecturas de IoT, autoconsciencia computacional y medición de la calidad de software. Esto asegura la formalidad y coherencia semántica de los modelos utilizados por la infraestructura. En particular, el metamodelo Monitor-IoT proporciona un lenguaje de modelo de arquitecturas de monitoreo de IoT más completo, flexible y extensible, capaz de modelar una amplia variedad de entidades digitales y flujos de datos entre ellas para soportar los procesos de recolección, transporte, procesamiento, agregación y almacenamiento de datos, a nivel de las capas Edge, Fog y Cloud. Por su parte, el metamodelo Aware-IoT, hasta donde se conoce en la literatura, resulta ser el primer lenguaje para modelar el dominio y las capacidades de autoconsciencia de un sistema IoT, basado en métricas de calidad de software.

A su vez, las herramientas DSL Monitor-IoT y configurador web de autoconsciencia, basadas en los metamodelos antes indicados, se implementaron para ofrecer funcionalidades que describan el problema y espacio de la solución con un alto nivel de abstracción. Es así que, el DSL Monitor-IoT posee una interfaz de modelado visual de arquitecturas de monitoreo multicapa para IoT; mientras que, el configurador de autoconsciencia trabaja con especificaciones (conceptos) de alto nivel sobre las capacidades de autoconsciencia de un sistema IoT. En consecuencia, las herramientas citadas contribuyen a facilitar y acelerar las tareas diseño, implementación y mantenimiento de los sistemas IoT autoconscientes, capaces de evaluar su estado y posible evolución. Esta afirmación fue comprobada durante la evaluación experimental de estas herramientas (ver Capítulo 7).

Por otro lado, un prototipo del middleware de la infraestructura, encargado de implementar y/o reconfigurar de forma distribuida los artefactos de software de monitoreo y autoconsciencia, fue implementado bajo el entorno de tiempo de ejecución multiplataforma Node.js. No obstante, la arquitectura del middleware fue diseñada para que pueda ser implementada en cualquier plataforma tecnológica. En este sentido, se explicó a detalle cada uno de los componentes del middleware, lo cual será una guía para aquellos desarrolladores que pretendan implementar o instanciar la solución en otra plataforma tecnológica. Incluso la arquitectura del middleware es tan general que, puede ser utilizada para la implementación distribuida de artefactos de software que soporten

otras incumbencias (concerns) de los sistemas IoT; por ejemplo, la creación de artefactos de software que soporten los procesos de actuación en un sistema IoT. Para este caso, se deberían construir nuevos metamodelos sobre el área de interés a representar y los componentes propios de transformación de las especificaciones de los modelos a artefactos de software.

En el caso de requerir implementar el middleware en otro lenguaje de programación, se recomienda considerar los siguientes aspectos técnicos para su elección, mismos que son intrínsecos a los entornos en los que operan los sistemas IoT: i) *multiplataforma*, posibilidad de operar en una gran cantidad de sistemas operativos tanto para servidores como dispositivos Edge con limitadas capacidades de hardware y energía; ii) *metaprogramación reflexiva*, capacidad de auto modificar el código en tiempo de ejecución, ante los cambios detectados en las especificaciones de los modelos; y, iii) *programación asíncrona*, capacidad de manejar sin bloqueos una gran cantidad de solicitudes o eventos concurrentes, con un alto nivel de rendimiento y escalabilidad.

Otro aspecto destacable de la arquitectura del middleware es el tipo de comunicación push asíncrona que emplea para intercambiar la información (modelos, planes, etc.) entre el nodo supervisor y el resto de nodos de computación del sistema IoT. Este tipo de comunicación ha permitido que el nodo supervisor, por intermedio de un broker, envíe (publique) la información al resto de nodos, únicamente cuando se detectan cambios en los modelos en tiempo de ejecución; lo cual indiscutiblemente, reduce el intercambio de datos (solicitudes) entre estos nodos; y, por ende, minimiza el uso de sus recursos. Además, los programas utilizados por los nodos para publicar u obtener la información de los tópicos del broker son simples y ligeros, siendo una ventaja en términos de procesamiento y consumo de memoria. En consecuencia, la comunicación push asíncrona resulta ser útil y eficiente para sistemas IoT conformados por grandes cantidades de dispositivos IoT heterogéneos, con capacidades limitadas de procesamiento, memoria, energía y ancho de banda.

En términos generales, una ventaja de la infraestructura de soporte a la metodología Aware-IoT@Run.Time es que puede ser desplegada en cualquier subdominio de IoT, si bien en los capítulos posteriores, la solución propuesta se aplica y evalúa en el subdominio de ambientes de vida asistida, esta puede ser utilizada en otros subdominios, tales como: control ambiental, industria, ciudad inteligente, etc.

Finalmente, se recomienda establecer una hoja de ruta para aprovechar las oportunidades de mejora de la infraestructura de soporte descritas en la sección 5.6.

Capítulo 6

Aplicación de la Solución

Este capítulo describe cómo aplicar la metodología Aware-IoT@Run.Time y las herramientas de su infraestructura de soporte. Para ello, se propone un estudio de caso dentro del subdominio de ambientes de vida asistida, con la finalidad de demostrar la aplicabilidad y utilidad de la solución tecnológica. En términos generales, el estudio de caso tiene como propósito implementar, poner en operación y mantener un sistema con capacidades de monitoreo y autoconsciencia para gestionar las emergencias de los adultos mayores que viven solos en su hogar.

La sección 6.1 presenta el estudio de caso propuesto. Luego, en la sección 6.2 se detallan los prerequisites de hardware y software, que servirán de base para la implementación del estudio de caso utilizando la metodología y sus herramientas de soporte. A su vez, la sección 6.3 describe la ejecución y los resultados de cada una de las actividades de la metodología. Finalmente, la sección 6.4 presenta las conclusiones del capítulo.

6.1 Estudio de caso para la aplicación de la metodología Aware-IoT@Run.Time

El escenario de IoT propuesto para la aplicación de la metodología es una adaptación de los estudios de caso presentados en [56] y [194]. Estos estudios de caso contemplan un sistema de Sistema de Gestión de Emergencias (SGE) compuesto por dos subsistemas (ver Fig. 6): i) un Subsistema de Control Ambiental (SCA), encargado de monitorear la temperatura, humedad y presencia de monóxido de carbono (CO) en el ambiente de una casa; y, ii) un Subsistema de Salud (SS) para monitorear la frecuencia cardíaca de un adulto mayor.

Específicamente, el estudio de caso a desplegar en este capítulo se enmarca dentro del subsistemas de Control Ambiental y tiene como alcance ejemplificar la implementación, puesta en operación y mantenimiento de un conjunto de requerimientos de monitoreo y autoconsciencia en este subsistema, con el apoyo de la metodología Aware-IoT@Run.Time y sus herramientas de soporte. Cabe recalcar que el Subsistema de Salud fue utilizado como parte de un cuasi experimento que evalúa empíricamente el DSL Monitor-IoT, mismo que se describe en el siguiente capítulo (Sección 7.3) y Anexo D.

6.1.1 Especificación de requerimientos de monitoreo del estudio de caso

La casa donde vive el adulto mayor tiene tres habitaciones: sala, cocina y dormitorio (ver Fig. 6-1). De ahí que, se necesita monitorear cada 10 minutos la temperatura y humedad en el dormitorio del adulto mayor; y, la presencia de CO en la cocina. Además, con el objeto de registrar el historial del rendimiento de la CPU de un servidor local ubicado en la casa del usuario, el cual opera como nodo Fog, se requiere monitorear cada 10 minutos el porcentaje de uso (carga de procesamiento) de la CPU de este servidor.

A continuación, se detallan los dispositivos IoT, nodos de computación; y, sus recursos de hardware y software necesarios para la implementación de los requerimientos de monitoreo, a nivel de las capas Edge, Fog y Cloud:

- **Capa Edge.** Para cumplir con los requerimientos de monitoreo, la plataforma IoT incluye dos tipos de sensores. En el dormitorio del adulto mayor se debe instalar un

sensor de temperatura y humedad *DHT11* [195] (ver Fig. 6-1 (a)), mientras que, en la cocina un sensor de CO *MQ-7* [196] (ver Fig. 6-1 (b)). La plataforma IoT también contempla la utilización de un microcontrolador *Raspberry PI 3* que actúa como puerta de enlace. Este microcontrolador ejecuta periódicamente la aplicación de software *controladorAmbienteCasa* que llama a las APIs *obtenerTemperaturaHumedad* y *obtenerCO* para recolectar los datos desde los sensores de temperatura, humedad y CO por medio de la interfaz de comunicación GPIO de la *Raspberry PI 3* [197] (ver Fig. 6-1 (c)). Además, la aplicación de software, por un lado, envía de forma síncrona los datos recolectados de temperatura y humedad al servidor ubicado en la casa del usuario (nodo Fog), consumiendo un servicio RESTful (*guardarTemperaturaHumedad*) proveído por este; y, por otro lado, envía de forma asíncrona los datos recolectados de CO al nodo Fog, utilizando como intermediario un broker (tópico *datosAmbienteCocina*). La puerta de enlace y el nodo Fog se comunican mediante una red de área local Wi-Fi, empleando los protocolos de aplicación HTTP (comunicación síncrona) y MQTT (comunicación asíncrona).

- **Capa Fog.** Como se mencionó, esta capa contiene un servidor *HP Proliant Microserver* ubicado en la casa del usuario, con capacidades para procesar y almacenar localmente los datos provenientes de los sensores, a fin de reducir la latencia, tráfico de red y controlar oportunamente los dispositivos ubicados al borde de la infraestructura IoT (ver Fig. 6-1 (d)). Este nodo Fog incluye los siguientes recursos:
 - Un sistema de gestión de base de datos *PostgreSQL* [198], cuyo objetivo es almacenar los datos crudos provenientes de todos los sensores físicos y virtuales de la plataforma IoT. Esta base de datos contiene las siguientes tablas: i) una tabla *AmbienteDormitorio* para almacenar la temperatura y humedad del dormitorio (*temperatura*, *humedad*), la fecha y hora (*fechaHora*) en que se recolecta el dato y el identificador del sensor (*idSensor*) que recolecta el dato; ii) una tabla *AmbienteCocina* para almacenar la concentración de CO de la cocina (*CO*), la fecha y hora (*fechaHora*) en que se recolecta el dato y el identificador del sensor (*idSensor*) que recolecta el dato; y, iii) una tabla *CargaCPUFog* para la almacenar el porcentaje de uso de la CPU (*cargaCPU*) del nodo Fog, la fecha y hora (*fechaHora*) en que se obtiene el dato, el identificador de la CPU (*idCPU*) que es monitoreada y el número de sensores activos al momento del monitoreo (*numeroSensoresControlados*).
 - Un broker *Eclipse Mosquitto* [199] que implementa un modelo de comunicación asíncrono de datos (modelo publicación / suscripción).
 - Un servidor de aplicaciones, basado en el *framework Express de Node.js* [176], que contiene servicios RESTful de intercambio, agregación y almacenamiento de datos;
 - Una aplicación de software *controladorCargaCPU* que se ejecuta periódicamente y llama en primera instancia a la API *obtenerCargaCPU* para conseguir el porcentaje de uso de la CPU del nodo Fog. Luego, consume un servicio RESTful (*guardarCargaCPU*) para almacenar dicho valor porcentual en la base de datos ubicada en el mismo nodo Fog.
 - Una aplicación de software *controladorResumenDiarioCO* que invoca una vez al día (23h55) a un servicio RESTful (*guardarResumenDiarioCO*) para almacenar el resumen (promedio, máximo, mínimo) diario de concentración de CO en una base de datos del nodo Cloud.
 - Un servicio RESTful *guardarTemperaturaHumedad* que recibe los datos recolectados de temperatura y humedad para su almacenamiento en la base de datos de monitoreo del nodo Fog. Este servicio es consumido directamente por la

- aplicación de software *controladorAmbienteCasa* (soporta flujos de recolección de datos síncronos).
 - Un servicio RESTful *guardarCO* que es consumido desde un evento asociado al broker, con el propósito de receptar los datos de CO publicados en el tópico respectivo (*datosAmbienteCocina*) y almacenarlos en la base de datos de monitoreo del nodo Fog (soporta flujos de recolección de datos asíncronos).
 - Un servicio RESTful *guardarCargaCPU* que es consumido directamente por la aplicación de software *controladorCargaCPU*, con el fin de almacenar los datos sobre la carga de la CPU del nodo Fog.
 - Un servicio RESTful *guardarResumenDiarioCO* que calcule el valor promedio, máximo y mínimo diario de CO en el ambiente de la cocina; y, los datos resultantes los almacena en la base de datos del nodo Cloud (soporta flujos de agregación de datos).
- Capa Cloud.** El escenario de IoT requiere de un nodo Cloud implementado en la *plataforma de CEDIA* [200] (ver Fig. 6-1 (e)). Este nodo Cloud dispone de un sistema de gestión de base de datos *PostgreSQL* [198] para almacenar los datos de resumen diario de CO, resultantes de las operaciones de agregación ejecutadas en el nodo Fog. Puntualmente, la base de datos contiene una tabla *ResumenDiarioCO* para almacenar el promedio, máximo y mínimo diario de CO (*promedioCO*, *maximoCO*, *minimoCO*), el identificador del sensor (*idSensor*) que recolecta el dato; y, la fecha de medición (*fecha*). La comunicación entre los nodos Fog y Cloud se realiza a través de la red de Internet.

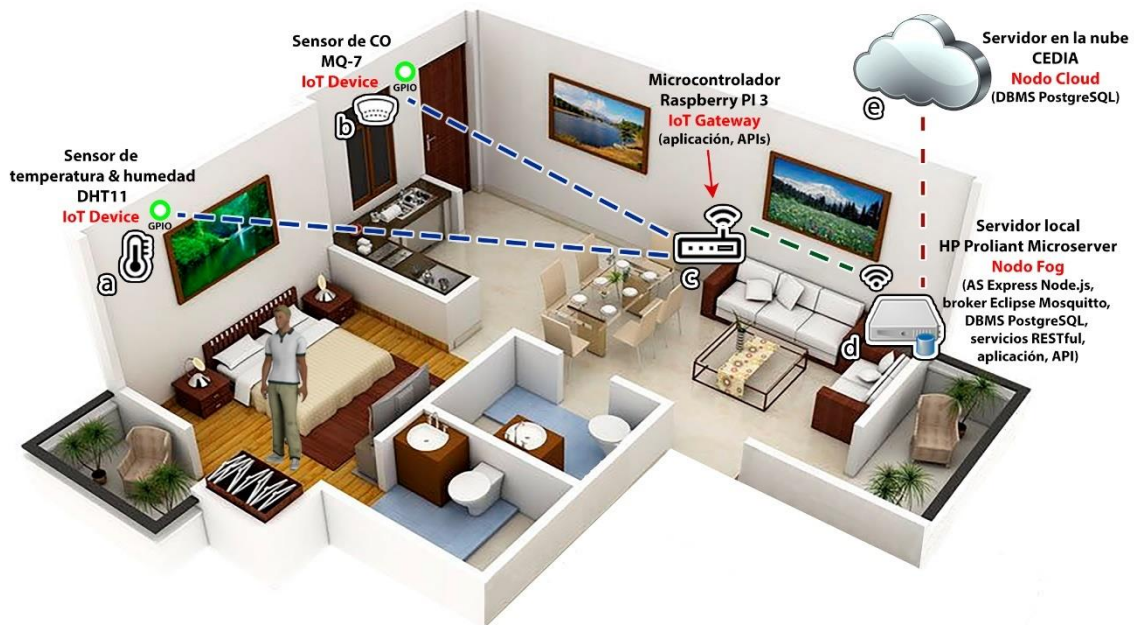


Figura 6-1. Escenario de IoT del estudio de caso.
Fuente: Elaboración propia.

6.1.2 Especificación de requerimientos de autoconsciencia del estudio de caso

Tal como se mencionó con anterioridad, los requerimientos de autoconsciencia se enmarcan únicamente dentro del Subsistema de Control Ambiental (SCA) y abarcan una amplia variedad de objetos, aspectos y procesos de autoconsciencia, relacionados tanto con entidades físicas como digitales. En este sentido, el estudio de caso tiene como objetivo dotar al subsistema SCA de capacidades de autoconsciencia pre-reflexiva y reflexiva, de tal forma que le permitan, mediante la evaluación de métricas, comprender

el estado o la evolución (predicción) de sus entidades físicas y digitales; y, en caso de ser necesario llegar a determinar acciones correctivas.

En particular, se busca desarrollar en el subsistema SCA una autoconsciencia pre-reflexiva sobre: i) la temperatura ambiente en tiempo real del dormitorio del adulto mayor y comprender el nivel de satisfacción del usuario; ii) la concentración en tiempo real de CO en el ambiente de la cocina e intuir el nivel de riesgo que tiene para la salud y vida del adulto mayor; y, iii) la carga de procesamiento en tiempo real de la CPU del nodo Fog, es decir, determinar qué tan ocupada está la CPU en un momento dado.

Además, se requiere que el subsistema SCA, a partir de los datos históricos, pueda desarrollar una autoconsciencia reflexiva sobre: i) la variabilidad diaria de la temperatura ambiente en el dormitorio del adulto mayor y el nivel de aceptabilidad del usuario; y, ii) el promedio diario de concentración de CO en el ambiente de la cocina y su nivel de riesgo.

Asimismo, se necesita que el subsistema SCA sea capaz de crear y aplicar un modelo basado en la técnica de regresión lineal simple, con el propósito de predecir el nivel de uso o carga de procesamiento requerido por la CPU del Nodo Fog para procesar los datos de monitoreo de un determinado número sensores (variable de simulación).

Finalmente, el subsistema SCA debe ser capaz de decidir sobre sus acciones futuras como respuesta a los procesos de autoconsciencia. En la Tabla 6-1 se detalla la información referente a los procesos de autoconsciencia a configurar, implementar y mantener en este escenario, así como los objetos, aspectos, métricas, criterios de decisión (umbrales) y acciones (recomendaciones prescriptivas) asociadas a estos procesos.

Los demonios de los procesos de autoconsciencia deben ejecutarse en el nodo Fog de la infraestructura IoT. De igual manera, las métricas resultantes de estos procesos deben almacenarse en la base de datos PostgreSQL ubicada en el nodo Fog. Para ello, esta base de datos contiene la tabla *MetricaAutoconsciencia*, la cual debe almacenar la siguiente información general por cada métrica: el identificador, tipo (directa, indirecta, indicador) y valor de la métrica calculada (*idMetrica*, *tipoMetrica*, *valorMetrica*), la fecha de cálculo de la métrica (*fechaHora*); y, los identificadores del proceso, sujeto, objeto y aspecto de autoconsciencia relacionados con la métrica (*idProceso*, *idSujeto*, *idObjeto*, *idAspecto*). Además, para el caso de las métricas de tipo indicador, esta tabla debe almacenar: el umbral en el que se encuentra el indicador (*umbralIndicador*), la interpretación del indicador (*interpretacionIndicador*); y, las recomendaciones o acciones correctivas (*recomendaciones*). Asimismo, en el caso de que el proceso utilice simulaciones, se debe guardar: el identificador del escenario de simulación (*idEscenarioSimulacion*); y, el nombre y valor de la variable de simulación (*nombreVariableSimulacion*, *valorVariableSimulacion*).

Finalmente, cabe señalar que el estudio de caso propuesto tiene como alcance la implementación y mantenimiento de procesos relacionados con aspectos de autoconsciencia individuales, dejando los aspectos colectivos para ser abordados en trabajos futuros, una vez que, se implementen las funcionalidades y componentes de la infraestructura de soporte relacionados con estos aspectos.

6.1.3 Especificación de requerimientos de gestión de cambios del estudio de caso

Uno de los objetivos principales del estudio de caso es demostrar la capacidad de la solución Aware-IoT@Run.Time para gestionar los cambios y la evolución del subsistema SCA, a través de la relación de causalidad que existe entre los modelos en tiempo de

Tabla 6-1. Descripción de los procesos pre-reflexivos y reflexivos a implementar en el estudio de caso.

Id Proceso	Tipo de proceso de autoconsciencia	Frecuencia de ejecución del proceso	Aspectos de Autoconsciencia a capturar	Criterios de decisión (umbrales) ¹	Interpretación	Acciones (recomendaciones prescriptivas)
1	Pre-reflexivo	10 minutos	Temperatura ambiente en tiempo real del dormitorio.	$\geq 18^{\circ}\text{C}$ Y $\leq 22^{\circ}\text{C}$ Caso contrario	Satisfactorio , temperatura ambiente confortable para el usuario Insatisfactorio , temperatura ambiente no confortable para el usuario.	No es necesario implementar acciones. El subsistema SCA debe calibrar el aire acondicionado de la habitación (autoadaptación).
2	Reflexivo	24 horas	Variabilidad diaria de la temperatura ambiente del dormitorio.	$\geq 0^{\circ}\text{C}$ Y $\leq 4^{\circ}\text{C}$ Caso contrario	Aceptable nivel de variabilidad diaria de la temperatura ambiente. Inaceptable nivel de variabilidad diaria de la temperatura ambiente.	No es necesario implementar acciones. El subsistema SCA debe notificar a la central de monitoreo.
3	Pre-reflexivo	10 minutos	Concentración en tiempo real de monóxido de carbono (CO) en el ambiente de la cocina.	≥ 0 ppm Y ≤ 35 ppm > 35 ppm Y ≤ 200 ppm	Ausencia de riesgo , el usuario no sufre ningún síntoma ante una exposición prolongada. Riesgo medio , el usuario puede experimentar dolores de cabeza, pecho, fatiga o náuseas.	No es necesario implementar acciones. El subsistema SCA debe encender una alerta auditiva intermitente y notificar a la central de monitoreo.
4	Reflexivo	24 horas	Concentración promedio diario de monóxido de carbono (CO) en el ambiente de la cocina.	> 200 ppm	Riesgo alto , el usuario puede experimentar desorientación, pérdida del conocimiento e incluso la muerte.	El sistema SCA debe encender la alerta auditiva permanente y notificar a la central de monitoreo.
5	Pre-reflexivo	10 minutos	Carga de procesamiento en tiempo real de la CPU del nodo Fog.	$\geq 0\%$ Y $\leq 30\%$	Baja carga de procesamiento de la CPU.	El subsistema SCA debe verificar la disponibilidad de los sensores, la frecuencia de lectura de los sensores y la correcta operación de la red de comunicación (autorrecuperación). Además, debe notificar a la central de monitoreo.
6	Reflexivo	24 horas	Predecir la carga de procesamiento de la CPU del nodo Fog para gestionar un cierto número de sensores (p. ej., duplicar la cantidad de sensores que operan actualmente).	$> 30\%$ Y $\leq 80\%$ $> 80\%$	Normal carga de procesamiento de la CPU. Alta carga de procesamiento de la CPU.	No es necesario implementar acciones. El subsistema SCA debe verificar la frecuencia de lectura de los sensores o en su defecto incrementar la capacidad de procesamiento del nodo Fog (autoadaptación). Además, debe notificar a la central de monitoreo.

¹ °C: Grados Celsius de temperatura ambiente.

ppm: Partes por millón de concentración de monóxido de carbono (CO).

ejecución y este subsistema, la cual es operacionalizada por el middleware de la infraestructura. En este sentido, una vez que estén en operación todos los requerimientos de monitoreo y autoconsciencia descritos en las subsecciones anteriores se deberán realizar los siguientes cambios:

- Implementar un flujo de datos asíncrono en lugar de síncrono que recolecte los datos de temperatura y humedad desde el sensor *DHT11* y los envíe al nodo Fog para su almacenamiento en la correspondiente tabla implementada (*AmbienteDormitorio*). Para ello, debe publicar el tópico *DatosAmbienteDormitorio* en el broker *Mosquitto*, a fin de almacenar temporalmente los datos recolectados. A su vez, es necesario incrementar la frecuencia de ejecución de este flujo de datos a 30 minutos.
- Consecuentemente, se requiere incrementar la frecuencia de ejecución del proceso pre-reflexivo que *captura el conocimiento sobre la temperatura ambiente actual (en tiempo real) del dormitorio y el nivel de satisfacción del usuario* a 30 minutos; así como también modificar el valor mínimo y máximo del umbral *Satisfactorio* del criterio de decisión asociado a este proceso a 15°C y 25°C, respectivamente.

6.2 Prerrequisitos de hardware y software base para el estudio de caso

Con base en la descripción del estudio de caso, es necesario instalar y configurar el hardware y software base, a nivel de las capas Edge, Fog y Cloud, a fin de que se puedan ejecutar las herramientas de la infraestructura de soporte a la metodología *Aware-IoT@Run.Time*. En la Tabla 6-2 se enlistan los dispositivos IoT (sensores) y nodos de computación (puerta de enlace, nodo Fog y nodo Cloud) requeridos por el escenario de IoT, junto con el detalle de las interfaces de comunicación (dirección de red, protocolos y puertos) que deben utilizar cada uno de ellos. Además, en la Tabla 6-2 se detalla el sistema operativo, software base y las herramientas de la infraestructura de soporte a instalar en cada uno de los nodos de computación para la correcta implementación del escenario de IoT.

Como se puede observar, en el nodo Fog de la infraestructura de soporte se tendrá disponible el DSL Monitor-IoT y configurador web de autoconsciencia, necesarios para diseñar y construir los modelos de arquitectura y autoconsciencia en tiempo de ejecución del escenario de IoT. Por lo tanto, el nodo Fog contendrá la base de datos en MySQL del configurador de autoconsciencia.

Adicionalmente, el nodo Fog actuará en calidad de supervisor del sistema IoT, por lo que ejecutará la aplicación supervisora del middleware de la infraestructura, encargada de detectar los cambios en los modelos en tiempo de ejecución; y, a su vez, planificar y distribuir las tareas de implementación y reconfiguración de los recursos de monitoreo y autoconsciencia a la puerta de enlace, el propio nodo Fog y el nodo Cloud del escenario de IoT.

Como complemento, todos los nodos de computación (puerta de enlace, nodo Fog y nodo Cloud) ejecutarán la aplicación implementadora del middleware de la infraestructura, responsable de recepcionar la nueva versión de los modelos en tiempo de ejecución; y, los planes de implementación y reconfiguración de los recursos de monitoreo y autoconsciencia para su respectiva ejecución.

Con respecto al intercambio de información (modelos, planes, reportes de cumplimiento) entre el nodo supervisor y el resto de nodos de computación del sistema IoT se utilizará como intermediario el broker *Mosquitto*, el cual estará instalado en el mismo nodo supervisor (Fog).

Tabla 6-2. Prerrequisitos de hardware y software base para la instanciación de la metodología Aware-IoT@Run.Time y su infraestructura de soporte.

Dispositivo IoT o nodo de computación	Interfaz de comunicación	Sistema Operativo	Herramientas de la infraestructura de soporte instaladas	Protocolo / Puerto	Software base requerido
Sensor de temperatura y humedad (DHT11)	- GPIO 17 (Pin 11) [197]				
Sensor de CO (MQ-7)	- GPIO 18 (Pin 12) [197]				
Gateway IoT (Raspberry PI 3)	- Interfaz GPIO [197] - Wi-Fi, IP: 192.168.1.10	Raspbian	- Aplicación implementadora de recursos de monitoreo y autoconsciencia del middleware de la infraestructura.		- Node.js Linux (ARMv7) versión 14.17.0 o superior.
Nodo Fog (HP Proliant Microserver)	- Ethernet, IP: 192.168.1.11	Windows 10 Pro	- DSL Monitor-IoT para modelar la arquitectura de monitoreo multicapa del escenario de IoT.		- Java SE Development Kit 8 o superior
			- Configurador web de capacidad de autoconsciencia.	- HTTP (80)	- Node.js Windows (x64) versión 14.17.0 o superior.
			- Aplicación supervisora de recursos de monitoreo y autoconsciencia del middleware de la infraestructura.		- Servidor de aplicaciones web Express de Node.js versión 4.17.2 o superior.
			- Aplicación implementadora de recursos de monitoreo y autoconsciencia del middleware de la infraestructura.	- HTTP (9999)	
			- Visor de autoconsciencia	- HTTP (9914)	
			- Base de datos del configurador de capacidades de autoconsciencia.	- mysql (3306)	- MySQL Community Server Windows (x64) version 5.7.37 o superior.
			- Base de datos de monitoreo y autoconsciencia	- postgresql (5432)	- PostgreSQL Windows (x64) versión 14.7 o superior.
Nodo Cloud (CEDIA)	- IP: 172.16.1.10	Windows 10 Pro	- Broker	- MQTT (1883)	- Eclipse Mosquitto Windows (x64) version 2.0.14 o superior.
			- Aplicación implementadora del middleware de la infraestructura.		- Node.js Windows (x64) versión 14.17.0 o superior.
			- Base de datos de resumen diario de monitoreo.	- postgresql (5432)	- PostgreSQL Windows (x64) versión 14.7 o superior.

Por otro lado, conforme a lo descrito en el estudio de caso, el nodo Fog también contendrá las bases de datos en PostgreSQL para almacenar los datos crudos de monitoreo y las métricas (directas, indirectas, indicadores) generadas por los demonios de los procesos de autoconsciencia. A su vez, la aplicación implementadora del nodo Fog desplegará el servidor de aplicaciones Express de Node.js para publicar los servicios web requeridos por el estudio de caso. De igual manera, los flujos de recolección de datos asíncronos estarán soportados por el broker Mosquitto, instalado en el nodo Fog.

Finalmente, el nodo Cloud dispondrá de una base de datos en PostgreSQL para almacenar los datos de resumen diario resultantes de los flujos de agregación.

6.3 Aplicación de la metodología Aware-IoT@Run.Time

Con base en el proceso metodológico para la construcción, operación y mantenimiento de sistemas IoT autoconscientes presentado en el Capítulo 5, en esta sección se aplican de forma ordenada y sistemática, todas las actividades, tareas y subtarefas de este proceso en el estudio de caso propuesto.

Por consiguiente, el proceso metodológico a ejecutar contempla seis actividades (ver Fig. 4-1): i) *Diseño de la arquitectura de monitoreo del sistema IoT*; ii) *Configuración de las capacidades de autoconsciencia del sistema IoT*; iii) *Sincronización entre los modelos de arquitectura – autoconsciencia y el sistema IoT observado*; iv) *Monitoreo*; v) *Aprendizaje*; y, vi) *Razonamiento*. Las dos primeras actividades son desarrolladas por el investigador con el apoyo de las herramientas DSL Monitor-IoT y configurador web de capacidades de autoconsciencia, a fin obtener un archivo JSON con las especificaciones de los modelos de arquitectura y autoconsciencia en tiempo de ejecución del escenario de IoT. Las cuatro actividades restantes se ejecutan automáticamente por medio del middleware de la infraestructura, una vez detectada la existencia del archivo JSON o una nueva versión del mismo en un directorio preestablecido, el cual es periódicamente observado por el middleware. Estas cuatro actividades tienen como alcance la implementación, reconfiguración y operación de los recursos de software de monitoreo y autoconsciencia requeridos por el escenario de IoT, conforme a las especificaciones contenidas en el archivo JSON. En lo que resta de esta sección, en primer lugar, se aborda la implementación de los requerimientos iniciales de monitoreo y autoconsciencia del estudio de caso (Subsección 6.3.1). Luego, se describe cómo se gestionaron los cambios requeridos por el estudio de caso, con el apoyo de la solución Aware-IoT@Run.Time (Subsección 6.3.2).

6.3.1 Implementación de los requerimientos iniciales del estudio de caso

En esta subsección se explica detalladamente la ejecución de cada una de las actividades de la metodología Aware-IoT@Run.Time, en el ámbito de los requerimientos iniciales del estudio de caso.

Actividad 1. Diseño de la arquitectura de monitoreo del sistema IoT

En esta actividad se modela la arquitectura de monitoreo multicapa del Subsistema de Control Ambiental, utilizando el DSL Monitor-IoT. Para ello, se ejecutan secuencialmente tres tareas: i) *Identificación de subsistemas, entidades, propiedades y flujos de datos*; ii) *Modelado de subsistemas, entidades y propiedades*; y, iii) *Modelado de flujos de datos*.

Tarea 1.1. Identificación de subsistemas, entidades, propiedades y flujos de datos

Analiza los requerimientos del escenario de IoT y prepara la documentación requerida por el DSL Monitor-IoT para el modelado de la arquitectura de monitoreo. En este sentido, para el levantamiento de las especificaciones de los componentes y atributos que deben incluirse en el modelo de arquitectura de monitoreo del escenario de IoT, se utiliza un conjunto de plantillas (ver Tablas de la 6-3 a la 6-19), las cuales permiten clasificar y organizar de manera estructurada y ordenada esta documentación. La documentación generada en esta instancia, facilita y agiliza las tareas posteriores de modelado. Para completar esta tarea se ejecutan cuatro subtareas:

Subtarea 1.1.1: Descomposición del sistema IoT

Desde un ámbito general, se define la estructura jerárquica del Sistema de Gestión de Emergencias (SGE), dividiéndole en los Subsistemas de Control Ambiental (SCA) y Salud (SS) (ver Fig. 6-2). A su vez, especifica los siguientes datos por cada sistema o subsistema: sigla (identificador), nombre, descripción y dominio de aplicación al que pertenece (ver Tabla 6-3). Las siguientes actividades y tareas se concentrarán únicamente en el subsistema SCA; sin embargo, la inclusión de la estructura jerárquica del sistema y sus subsistemas dentro del modelo de arquitectura, permite que a futuro estos subsistemas puedan compartir y reutilizar las entidades digitales (p. ej., dispositivos IoT, nodos de computación, recursos de hardware y software) previamente definidas.

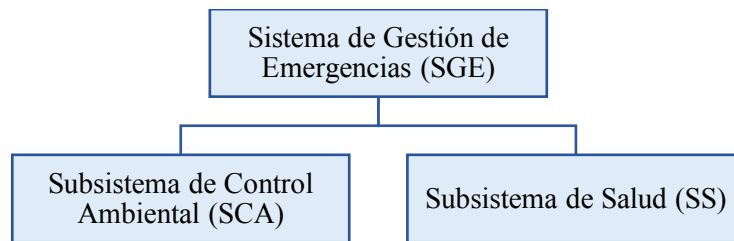


Figura 6-2. Estructura jerárquica del sistema de gestión de emergencias.
Fuente: Elaboración propia.

Tabla 6-3. Plantilla para registrar sistemas y subsistemas del escenario de IoT.

Id. (Sigla)	Nombre	Descripción	Dominio
SGE	Sistema de Gestión de Emergencias.	Gestiona las emergencias de adultos mayores que viven solos en su hogar.	AAL
SCA	Subsistema de Control Ambiental.	Monitorea la temperatura, humedad y concentración de monóxido de carbono (CO) en el ambiente de la casa.	AAL
SS	Subsistema de Salud.	Monitorea los signos vitales (frecuencia cardíaca) del adulto mayor.	AAL

Subtarea 1.1.2: Identificación de entidades

Determina las entidades físicas o digitales a monitorear, así como las entidades digitales (nodos de computación y dispositivos IoT), recursos de hardware / software (p. ej., interfaces de red, middleware, servicios, APIs, aplicaciones, base de datos, tablas, tópicos de broker) y redes de comunicación que son necesarios para soportar los procesos de monitorización (recolección, transporte, procesamiento y almacenamiento) del escenario de IoT, a nivel de las capas Edge, Fog y Cloud (ver Tablas de la 6-4 a la 6-17).

Tabla 6-4. Plantilla para registrar las especificaciones de las entidades físicas.

Entidad física	Descripción	Atributos	
		Sistema o Subsistema que lo utiliza	Tipo de entidad (metaclase)
Casa	Casa donde vive el adulto mayor.	SCA	PhysicalEntity
Dormitorio	Dormitorio del adulto mayor, cuya temperatura y humedad es monitoreada.	SCA	PhysicalEntity
Cocina	Lugar donde cocina el adulto mayor, cuya presencia de monóxido de carbono (CO) es monitoreada.	SCA	PhysicalEntity

Tabla 6-5. Plantilla para registrar las especificaciones de los dispositivos IoT (sensores).

Dispositivo IoT	Descripción	Atributos	
		Sistema o subsistema que lo utiliza	Tipo de sensor
Sensor de Temperatura y Humedad (DHT11)	Dispositivo IoT que monitorea la temperatura y humedad del dormitorio del adulto mayor.	SCA	Resistivo
Sensor de CO (MQ-7)	Dispositivo IoT que monitorea la presencia de monóxido de carbono (CO) en la cocina.	SCA	Conductivo

Tabla 6-6. Plantilla para registrar las especificaciones de las puertas de enlace IoT.

Puerta de enlace (Gateway) IoT	Descripción	Atributos	
		Sistema o subsistema que lo utiliza	Tipo de Gateway
Raspberry PI 3	Puerta de enlace en la que se ejecutan las aplicaciones y APIs de software que recolectan los datos crudos desde los sensores DHT11 y MQ-7.	SCA	Microcontrolador

Tabla 6-7. Plantilla para registrar las especificaciones de los nodos Fog.

Nodo Fog	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Modelo de servicio	Modelo de despliegue	Plataforma
HP Proliant Microserver (servidor local)	Nodo de computación ubicado en la casa del adulto mayor con capacidad de almacenar y procesar los datos crudos provenientes de los sensores.	SCA	No aplica	Privado	Node.js, PostgreSQL, Mosquitto

Tabla 6-8. Plantilla para registrar las especificaciones de los nodos Cloud.

Nodo Cloud	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Modelo de servicio	Modelo de despliegue	Plataforma
CEDIA Cloud	Nodo de computación en la nube que almacena los datos resumidos de monitoreo del sistema IoT (p. ej. resumen diario de la concentración de CO en la cocina).	SCA	PaaS	Público	Node.js, PostgreSQL

Tabla 6-9. Plantilla para registrar las especificaciones de las APIs.

API	Descripción	Atributos			
obtenerTemperaturaHumedad	Función API que obtiene la temperatura y humedad desde el sensor DHT11.	Sistema o subsistema que lo utiliza: SCA			
		Nodo de computación en el que opera: Raspberry PI 3			
		Parámetros de entrada: idSensor: Integer			
		Instrucciones que encapsula: var sensor = require('node-dht-sensor').promises; var valor=sensor.readSync(11,idSensor); return {'temperatura':valor.temperature, 'humedad':valor.humidity};			
		Variables de salida: temperatura: Float, humedad: Float			
obtenerCO	Función que obtiene la concentración de CO desde el sensor MQ-7.	Tipo de formato de los datos de salida: JSON			
		Sistema o subsistema que lo utiliza: SCA			
		Nodo de computación en el que opera: Raspberry PI 3			
		Parámetros de entrada: idSensor: Integer			
		Instrucciones que encapsula: var sensor = require('adc-pi-gpio'); var valor = gpio.read(idSensor); return {'CO':valor}			

obtenerCargaCPU	Función API que obtiene el porcentaje de uso de la CPU del nodo Fog.	Variables de salida: CO: Float
		Tipo de formato de los datos de salida: JSON
		Sistema o subsistema que lo utiliza: SCA
		Nodo de computación en el que opera: HP Proliant Microserver
		Parámetros de entrada: Ninguno
		Instrucciones que encapsula: const child = require('child_process'); let com='wmic cpu get loadpercentage /value'; let valor=child.spawnSync('cmd.exe',['/c',com]); valor=valor.output[1].toString().trim().split('=')[1]; return {'cargaCPU':valor};
		Variables de salida: cargaCPU: Float
		Tipo de formato de los datos de salida: JSON

Tabla 6-10. Plantilla para registrar las especificaciones de las aplicaciones.

Aplicación	Descripción	Atributos		
		Sistema o subsistema que lo utiliza	Nodo de computación en el que opera	Tipo de aplicación
controladorAmbienteCasa	Aplicación de software que invoca a las APIs correspondientes, a fin de obtener los datos de temperatura, humedad y CO desde los sensores. Luego, envía estos datos al nodo Fog, por medio de un servicio web o broker, para su respectivo almacenamiento.	SCA	Raspberry PI 3	Desktop
controladorCargaCPU	Aplicación que invoca a la API correspondiente, a fin de obtener el porcentaje de uso de CPU del nodo Fog. Luego, consume un servicio web para el almacenamiento de estos datos en el nodo Fog.	SCA	HP Proliant Microserver	Desktop
controladorResumenDiarioCO	Aplicación que consume un servicio web para obtener diariamente los datos de CO almacenados en la base de datos del nodo Fog, aplicar operaciones de agregación (promedio, máximo, mínimo) sobre los mismos; y, almacenar los datos resumidos en la base de datos del nodo Cloud.	SCA	HP Proliant Microserver	Desktop

Tabla 6-11. Plantilla para registrar las especificaciones de los middlewares.

Middleware	Descripción	Atributos		
		Sistema o subsistema que lo utiliza	Nodo de computación en el que opera	Protocolo que usa
Express (servidor de aplicaciones)	Servidor de aplicaciones web que publica los servicios RESTful para el intercambio, agregación y almacenamiento de datos.	SCA	HP Proliant Microserver	HTTP
Mosquitto (broker)	Broker que soporta la comunicación asíncrona entre la puerta de enlace y el nodo Fog.	SCA	HP Proliant Microserver	MQTT

Tabla 6-12. Plantilla para registrar las especificaciones de los servicios web.

Servicio web	Descripción	Atributos		
		Sistema o subsistema que lo utiliza	Nodo de computación en el que opera	Protocolo que usa
guardarTemperaturaHumedad	Servicio RESTful que almacena los datos recolectados de la temperatura y humedad del dormitorio, en la base de datos de monitoreo del nodo Fog.	SCA	HP Proliant Microserver	HTTP
		Endpoint:	temperaturahumedad	
		Método:	POST	
		Tipo de formato de los datos de salida:	JSON	
		Parámetros de entrada:	temperatura: Float, humedad: Float	
		Variables de salida:	Ninguna	
guardarCO	Servicio RESTful que almacena los datos recogidos del CO de la cocina, en la base de datos de monitoreo del nodo Fog.	SCA	HP Proliant Microserver	MQTT
		Endpoint:	co	
		Método:	POST	
		Tipo de formato de los datos de salida:	JSON	
		Parámetros de entrada:	CO: Float	
		Variables de salida:	Ninguna	

guardarCargaCPU	Servicio RESTful que almacena los datos recolectados de la carga de CPU, en la base de datos de monitoreo del nodo Fog.	Sistema o subsistema que lo utiliza: SCA Nodo de computación en el que opera: HP Proliant Microserver Endpoint: cargacpu Método: POST Tipo de formato de los datos de salida: JSON Parámetros de entrada: cargaCPU: Float Variables de salida: Ninguna
guardarResumenDiarioCO	Servicio RESTful que calcula el promedio, máximo y mínimo diario de la concentración de CO en la cocina y lo almacena en la base de datos del nodo Cloud.	Sistema o subsistema que lo utiliza: SCA Nodo de computación en el que opera: CEDIA Cloud Endpoint: resumendiarioco Método: POST Tipo de formato de los datos de salida: JSON Parámetros de entrada: Ninguno Variables de salida: Ninguna

Tabla 6-13. Plantilla para registrar las especificaciones de las bases de datos.

Base de datos	Descripción	Atributos
PostgreSQL Local	Base de datos ubicada en el nodo Fog. Almacena los datos crudos provenientes de los sensores.	Sistema o subsistema que lo utiliza: SCA
		Nodo de computación en el que opera: HP Proliant Microserver
		Uri: postgresql://postgres:admin@192.168.1.11:5432/postgresqllocal
		User: postgres
		Password: admin
PostgreSQL Cloud	Base de datos ubicada en el nodo Cloud. Almacena el resumen diario de la concentración de CO en la cocina.	Protocolo que usa: PostgreSQL
		Sistema o subsistema que lo utiliza: SCA
		Nodo de computación en el que opera: CEDIA Cloud
		Uri: postgresql://postgres:admin@172.16.1.10:5432/postgresqlcloud
		User: postgres
		Password: admin
		Protocolo que usa: PostgreSQL

Tabla 6-14. Plantilla para registrar las especificaciones de las tablas de base de datos.

Tabla: AmbienteDormitorio			
Descripción		Propiedades de tabla	
Almacena los datos del monitoreo de la temperatura y humedad del dormitorio del adulto mayor.		Sistema o subsistema que lo utiliza: SCA	
		Base de datos que almacena: PostgreSQL Local	
		Tipo de persistencia: Permanente	
		Unidad de tiempo persistencia: No aplica	
		Tiempo de persistencia: No aplica	
Columnas			
Nombre	Tipo de columna	Tipo de dato	Fórmula/Instrucciones
temperatura	Data	Float	
humedad	Data	Float	
idSensor	MetaData	Integer	
fechaHora	MetaData	Date	return new Date().toISOString();
Tabla: AmbienteCocina			
Descripción		Propiedades de tabla	
Almacena los datos del monitoreo de la concentración de CO de la cocina.		Sistema o subsistema que lo utiliza: SCA	
		Base de datos que almacena: PostgreSQL Local	
		Tipo de persistencia: Permanente	
		Unidad de tiempo persistencia: No aplica	
		Tiempo de persistencia: No aplica	
Columnas			
Nombre	Tipo de columna	Tipo de dato	Fórmula/Instrucciones
CO	Data	Float	
idSensor	MetaData	Integer	
fechaHora	MetaData	Date	return new Date().toISOString();
Tabla: CargaCPUFog			
Descripción		Propiedades de tabla	
Almacena los datos crudos del monitoreo de la carga de CPU del nodo Fog.		Sistema o subsistema que lo utiliza: SCA	
		Base de datos que almacena: PostgreSQL Local	
		Tipo de persistencia: Permanente	
		Unidad de tiempo persistencia: No aplica	
		Tiempo de persistencia: No aplica	

Columnas			
Nombre	Tipo de columna	Tipo de dato	Fórmula/Instrucciones
cargaCPU	Data	Float	
nroSensoresControlados	MetaData	Integer	
idCPU	MetaData	Integer	
fechaHora	MetaData	Date	return new Date().toISOString();
Tabla: ResumenDiarioCO			
Descripción		Propiedades de tabla	
Almacena el promedio, máximo y mínimo diario de la concentración de CO en la cocina.		Sistema o subsistema que lo utiliza: SCA	
		Base de datos que almacena: PostgreSQL Cloud	
		Tipo de persistencia: Permanente	
		Unidad de tiempo persistencia: No aplica	
		Tiempo de persistencia: No aplica	
Columnas			
Nombre	Tipo de columna	Tipo de dato	Fórmula/Instrucciones
promedioCO	Data	Float	
maximoCO	Data	Float	
minimoCO	Data	Float	
idSensor	MetaData	Integer	
fecha	MetaData	Date	return new Date().toISOString();

Tabla 6-15. Plantilla para registrar las especificaciones de los tópicos de broker.

Tópico	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Broker en el que se publica	Tipo de tópico	Tipo de formato de dato
datosAmbienteCocina	Tópico (tema) en el que se publican los datos crudos de la concentración de CO de la cocina.	SCA	Mosquitto	Atómico	JSON

Tabla 6-16. Plantilla para registrar las especificaciones de las redes de comunicación y protocolos.

Red de comunicación	Descripción	Atributos		
		Sistema o subsistema que lo utiliza	Protocolo	Puerto
ProximidadGPIO	Interfaz de comunicación GPIO que comunica el sensor de DHT11 y el sensor MQ-7 con el microcontrolador Raspberry PI 3.	SCA		
LAN	Red de área Local Wifi que comunica la puerta de enlace (microcontrolador Raspberry PI 3) con el nodo Fog y sus recursos.	SCA	HTTP	9999
			postgresql	5432
			MQTT	1883
Internet	Red de Internet que interconecta los nodos Fog y Cloud para acceder a sus recursos.	SCA	postgresql	5432

Tabla 6-17. Plantilla para registrar las especificaciones de las interfaces de red.

Interfaz de red	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Red a la que se conecta	Network Address	Mac Address
Wi-Fi – Raspberry PI 3	Adaptador de red Wi-Fi de la puerta de enlace Raspberry PI 3.	SCA	LAN	192.168.1.10	DC-A6-32-4A-FC-68
Ethernet – HP Proliant Microserver	Adaptador de red Ethernet del nodo Fog HP Proliant Microserver.	SCA	LAN Internet	192.168.1.11	50-5B-C2-D5-AE-8D
CEDIA Cloud	Adaptador de red del nodo Cloud de CEDIA.	SCA	Internet	172.16.1.10	No conocida

Subtarea 1.1.3: Identificación de propiedades

Especifica las propiedades a ser monitoreadas o utilizadas como metadatos por cada entidad física o digital, según el escenario de IoT (ver Tabla 6-18).

Tabla 6-18. Plantilla para registrar las especificaciones de las propiedades.

Propiedad	Definición	Atributos					
		Entidad física o digital relacionada	Identificador	Tipo de propiedad	Unidad de medida	Tipo de dato	Valor asignado en tiempo de diseño
temperatura	Temperatura ambiente de una habitación de la casa.	Dormitorio	No	Telemetría	Grados Celsius	Float	No
humedad	Humedad relativa de una habitación de la casa.	Dormitorio	No	Telemetría	Porcentual	Float	No
CO	Concentración de monóxido de carbono presente en una habitación de la casa.	Cocina	No	Telemetría	Parte por millón	Float	No
idSensor	Identificador de los sensores físicos. En este caso se utiliza el número de pin GPIO para identificarlos.	Sensores temperatura, humedad, CO	Si	Metadato	No aplica	Integer	Si
cargaCPU	Carga de procesamiento utilizada por el servidor.	HP Proliant Microserver	No	Estado	Porcentual	Float	No
idCPU	Identificador de la CPU del servidor.	HP Proliant Microserver	Si	Metadato	No aplica	Integer	Si
nroSensoresControlados	Cantidad de sensores que controla el servidor.	HP Proliant Microserver	No	Metadato	No aplica	Integer	No

Subtarea 1.1.4: Identificación de flujos de datos

Determina los flujos de datos requeridos por el escenario de IoT para transportar los datos recolectados desde las fuentes (sensores físicos o digitales) hasta su almacenamiento, con diferentes niveles de agregación, en las bases de datos correspondientes (ver Tabla 6-19).

Tabla 6-19. Plantilla para registrar las especificaciones de los flujos de datos.

Flujo de dato	Atributos				
	Tipo de flujo de dato	Tipo de comunicación	Unidad de tiempo	Frecuencia de ejecución del flujo de datos	Hora de ejecución del flujo de datos (opcional)
Flujo de recolección de datos de temperatura y humedad del dormitorio.	Flujo de recolección de datos	Síncrono	Minuto	10	No aplica
Flujo de recolección de datos de concentración de CO en la cocina.	Flujo de recolección de datos	Asíncrono	Minuto	10	No aplica
Flujo de recolección de datos de carga de CPU del nodo Fog.	Flujo de recolección de datos	Síncrono	Minuto	10	No aplica
Flujo de agregación que calcula el valor promedio, máximo y mínimo diario de la concentración de CO en la cocina.	Flujo de agregación de datos	Síncrono	Día	1	23h55

Tarea 1.2: Modelado de subsistemas, entidades y propiedades

A partir de la documentación generada en la Actividad 1 y utilizando el DSL Monitor IoT, por un lado, se modela la estructura jerárquica del sistema de gestión de emergencias y sus subsistemas; y, por otro lado, se diseña la estructura entre las entidades físicas y digitales (nodos de computación y sus recursos de hardware/software) requeridas por el escenario de IoT. A su vez, se organizan en categorías aquellas entidades del mismo tipo que comparten propiedades comunes. Finalmente, se configuran las propiedades a monitorear o ser utilizadas como metadatos en el escenario de IoT. Durante las tareas de modelado, los atributos de los componentes de la arquitectura son configurados utilizando la información contenida en las plantillas de la Actividad 1. Para cumplir con esta tarea se ejecutan siete subtareas:

Subtarea 1.2.1. Modelado de subsistemas

Diseña la estructura jerárquica entre el Sistema de Gestión de Emergencias (SGE) y los Subsistemas de Control Ambiental (SCA) y Salud (SS) (ver Fig. 6-3). Además, agrupa las entidades IoT por subsistemas, en función de su utilidad. A manera de ejemplo, en la Figura 6-4 se muestra la ventana de propiedades del DSL Monitor-IoT con los atributos establecidos para el subsistema SCA.

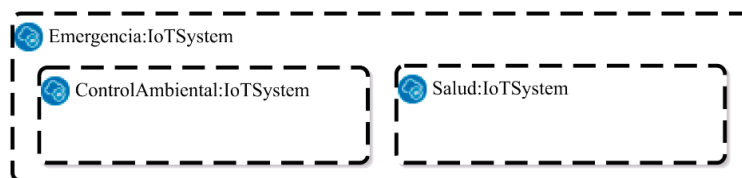


Figura 6-3. Modelado de la estructura jerárquica del Sistema de Gestión de Emergencia y sus subsistemas. Fuente: Elaboración propia.

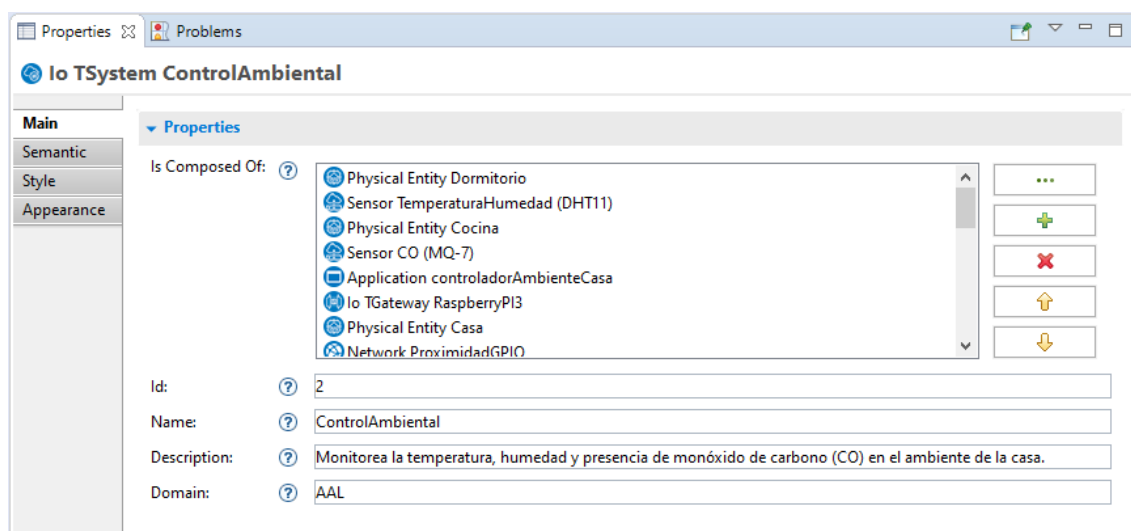


Figura 6-4. Ventana de propiedades del DSL Monitor-IoT con los atributos del subsistema SCA. Fuente: Elaboración propia.

Subtarea 1.2.2. Modelado de entidades físicas

Diseña la estructura jerárquica entre las entidades físicas del escenario de IoT. En la Figura 6-5 se observa que la entidad física *casa* contiene las entidades físicas *dormitorio* y *cocina*. Como ejemplo, en la Figura 6-6 se presentan los atributos especificados para la entidad física *dormitorio*, asimismo, se puede apreciar que incluye las propiedades *temperatura* y *humedad* a monitorear en el dormitorio.

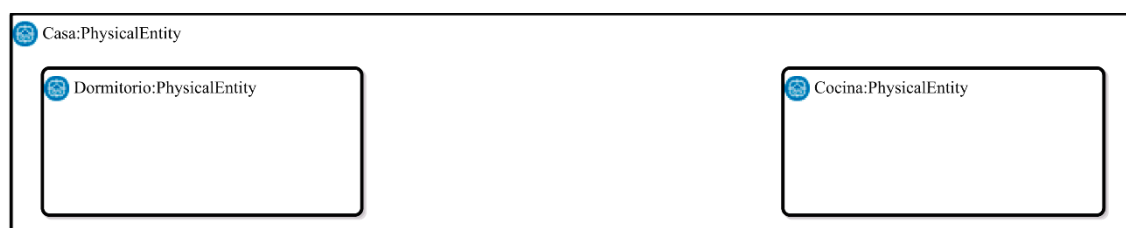


Figura 6-5. Modelado de las entidades físicas del escenario de IoT. Fuente: Elaboración propia.

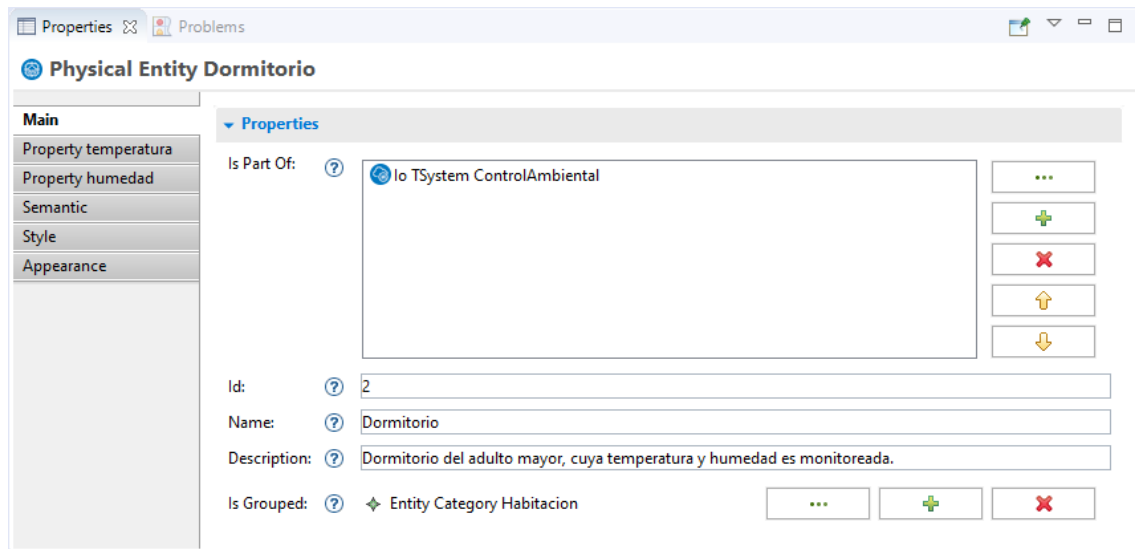


Figura 6-6. Ventana de propiedades del DSL Monitor-IoT con los atributos de la entidad física dormitorio. Fuente: Elaboración propia.

Subtarea 1.2.3. Modelado de nodos de computación a nivel Edge, Fog y Cloud

Añade los sensores, puertas de enlace, nodo Fog y nodo Cloud requeridos por el escenario de IoT. Estas entidades digitales pueden estar distribuidas y contenidas en las entidades físicas, según los requisitos del escenario de IoT. En este sentido, la Figura 6-7 muestra que el sensor de temperatura y humedad *DHT11* y el sensor de CO *MQ-7* están ubicados en el dormitorio y cocina de la casa, respectivamente. A su vez, la puerta de enlace *Raspberry PI 3* y el nodo Fog *HP Proliant Microserver* están instalados localmente en la casa. Finalmente, el nodo Cloud de *CEDIA* al ser una entidad externa se encuentra modelado fuera de la casa. A modo de ejemplo, en la Figura 6-8 se presentan parte de los atributos configurados para el sensor de temperatura y humedad *DHT11*.

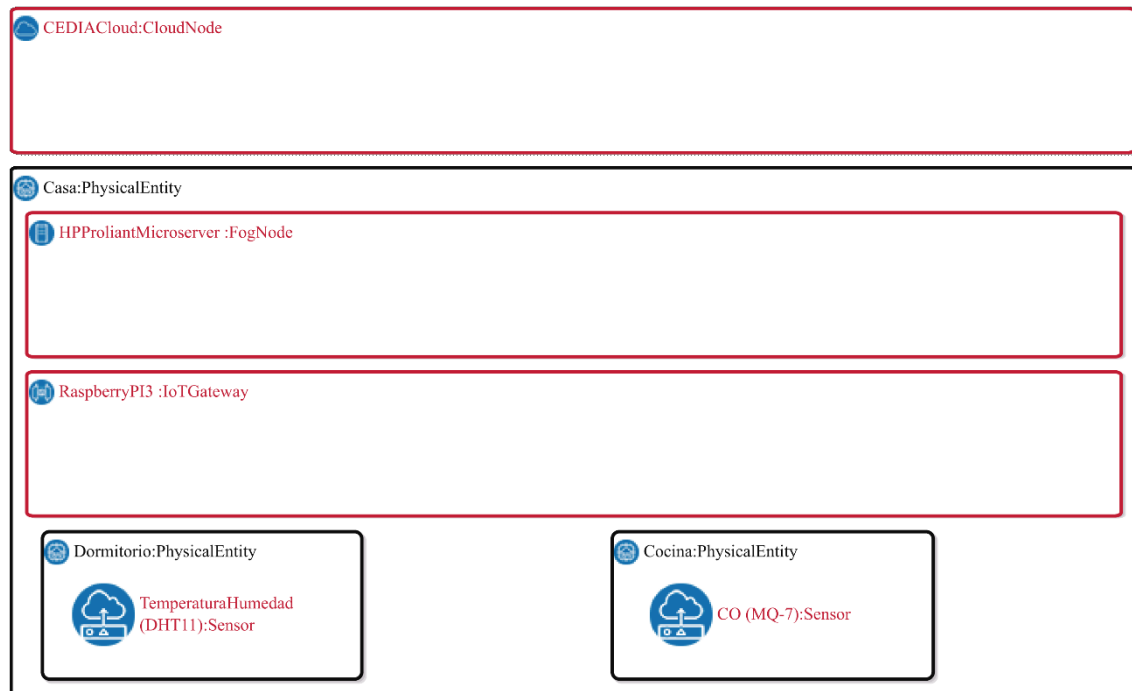


Figura 6-7. Modelado de los nodos de computación del escenario de IoT. Fuente: Elaboración propia.

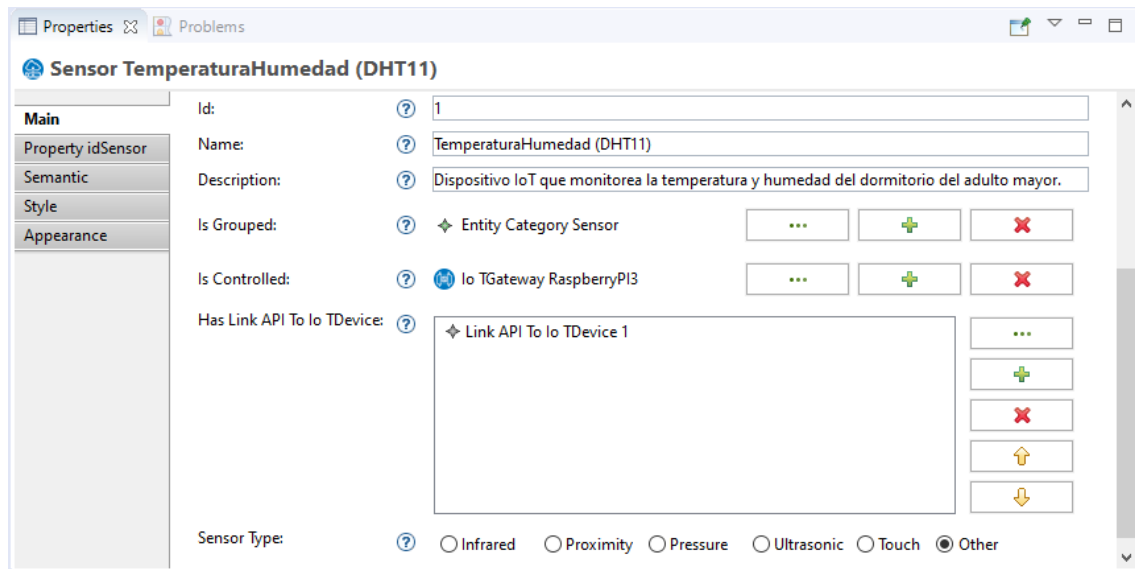


Figura 6-8. Ventana de propiedades del DSL Monitor-IoT con los atributos del sensor DHT11. Fuente: Elaboración propia.

Subtarea 1.2.4. Modelado de recursos utilizados por los nodos de computación

Incluye los recursos necesarios de hardware y software en los nodos de computación para soportar los procesos de monitorización (recolección, transporte, procesamiento y almacenamiento) del escenario de IoT. En la Figura 6-9 se puede apreciar que: i) las interfaces de red están incorporadas en la puerta de enlace *Raspberry PI 3*, nodo *Fog HP Proliant Microserver* y nodo *Cloud de CEDIA*, posibilitando la conexión de estos nodos a las redes de comunicación para compartir sus recursos y datos; ii) la aplicación *controladorAmbienteCasa*; y, las APIs *obtenerTemperaturaHumedad* y *obtenerCO* de los sensores físicos se ejecutan en la puerta de enlace; iii) la aplicación *controladorCargaCPU* y API *obtenerCargaCPU* que monitorea la carga de CPU del nodo *Fog* se ejecutan en este; iv) la aplicación *controladorResumenDiarioCO* que resume los datos de CO también se ejecuta en el nodo *Fog*; iv) el servidor de aplicaciones *Express* y broker *Mosquitto* igualmente se ejecutan en el nodo *Fog*; iv) los servicios web *guardarCargaCPU*, *guardarTemperaturaHumedad*, *guardarCO* y *guardarResumenDiarioCO* están publicados dentro del servidor de aplicaciones; v) el tópico *DatosAmbienteCocina* que almacena temporalmente los datos de monitoreo de CO está definido en el broker; y; vi) las bases de datos *PostgreSQLLocal* y *PostgreSQLCloud* están implementadas en los nodos *Fog* y *Cloud*, respectivamente; incluyendo la definición de las tablas que almacenan.

En particular, como se puede observar en la Figura 6-9, las APIs y servicios web contienen las especificaciones de los parámetros (datos) de entrada a utilizar por las instrucciones de los mismos, así como las especificaciones sobre las variables de retorno que deben devolver estos recursos. Finalmente, como ejemplo, en la Figura 6-10 se muestra la configuración de la base de datos local implementada en el nodo *Fog* (*PostgreSQLLocal*); y, en las Figuras 6-11 y 6-12 se presentan parte de las especificaciones de los atributos de la API *obtenerTemperaturaHumedad* y del servicio web *guardarTemperaturaHumedad*, debiendo mencionar que el código incluido en el atributo *Instrucciones* será encapsulado en la función de la API, al momento de su implementación automática en la Actividad 3 de la metodología.

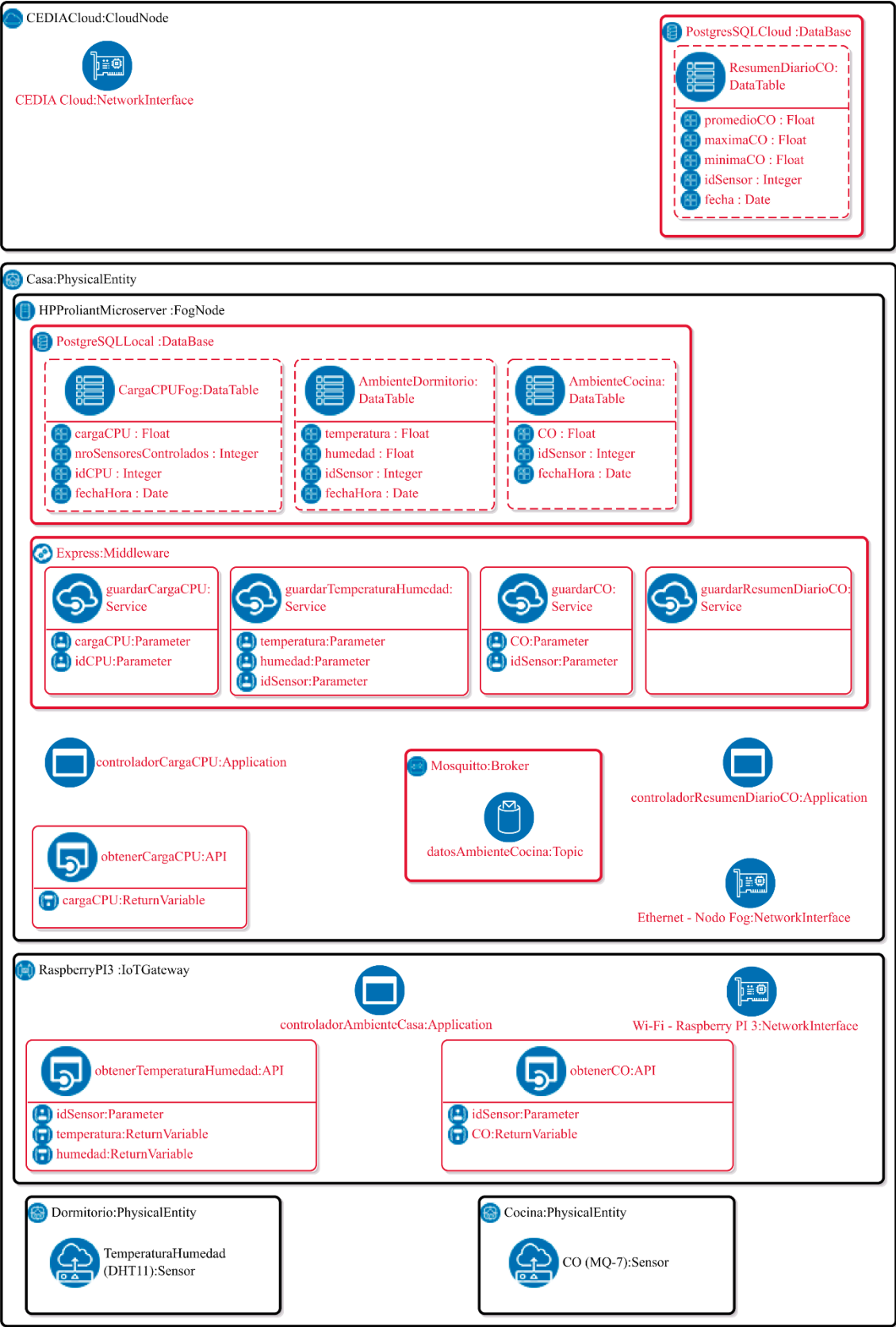


Figura 6-9. Modelado de recursos de hardware y software de los nodos de computación del escenario de IoT. Fuente: Elaboración propia.

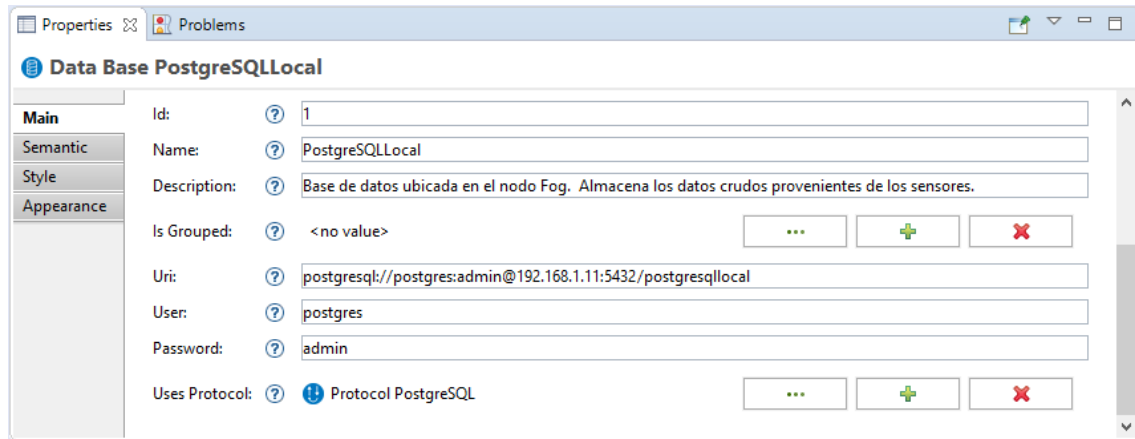


Figura 6-10. Ventana de propiedades del DSL Monitor-IoT con los atributos de la base de datos PostgreSQLLocal. Fuente: Elaboración propia.

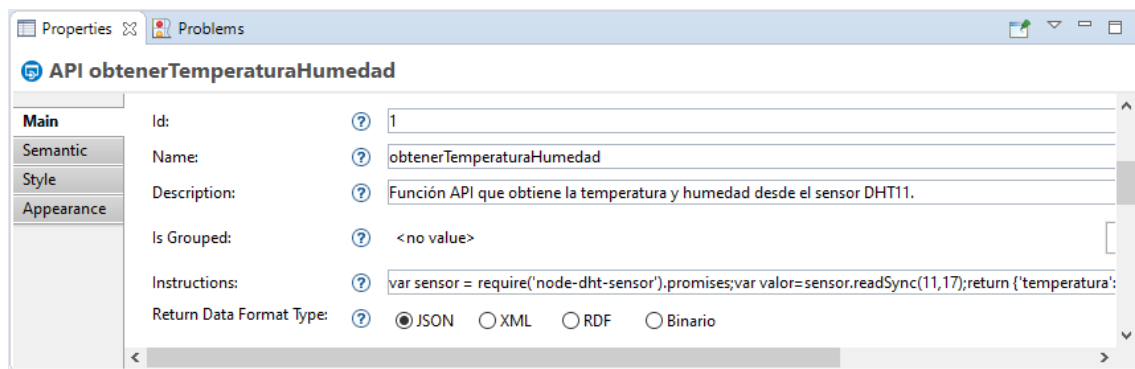


Figura 6-11. Ventana de propiedades del DSL Monitor-IoT con los atributos de la API obtenerTemperaturaHumedad. Fuente: Elaboración propia.

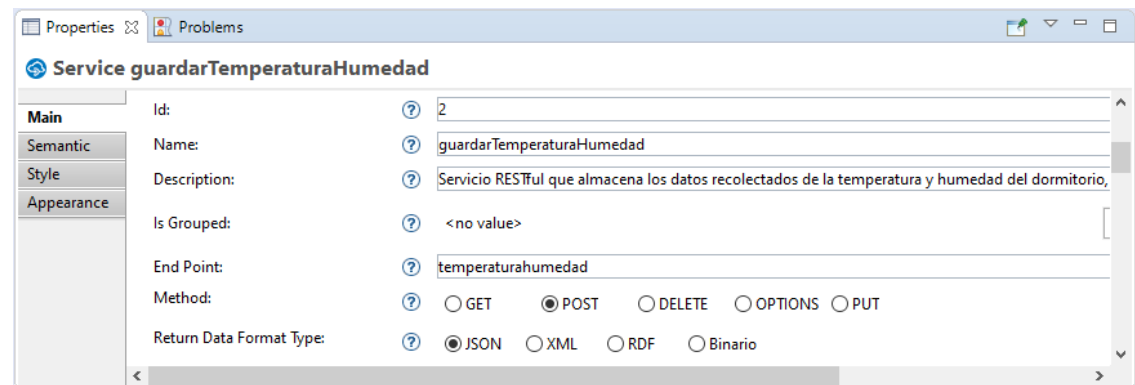


Figura 6-12. Ventana de propiedades del DSL Monitor-IoT con los atributos del servicio web guardarTemperaturaHumedad. Fuente: Elaboración propia.

Subtarea 1.2.5. Modelado de redes y protocolos de comunicación

Incorpora las redes y protocolos que soportan la comunicación síncrona y asíncrona entre las entidades digitales del escenario de IoT. En la Figura 6-13 se observan las tres redes de comunicación incorporadas al modelo de arquitectura: i) una *red de proximidad* que conecta los sensores *DHT11* y *MQ-7* con la puerta de enlace *Raspberry PI 3*, mediante la interfaz de comunicación *GPIO*; ii) una *red de área local Wi-Fi* que conecta la puerta de enlace *Raspberry PI3* con el nodo Fog *HP Proliant Microserver*, empleando los protocolos *HTTP* (comunicación síncrona) o *MQTT* (comunicación asíncrona); y iii) una *red de internet* que comunica el nodo Fog *HP Proliant Microserver* con el nodo *Cloud*

de *CEDIA*. Además, en esta subtarea se asignan y configuran las redes de comunicación a las interfaces de red de los nodos de computación, así como los protocolos a emplear por los recursos de software, tales como: servidor de aplicaciones *Express*, broker *Mosquitto*; y, bases de datos *PostgreSQLLocal* y *PostgreSQLCloud*. Como ejemplo, en las Figuras 6-14 y 6-15 se presentan la configuración de la interfaz de red del nodo Fog y la configuración del protocolo de red del broker, respectivamente.

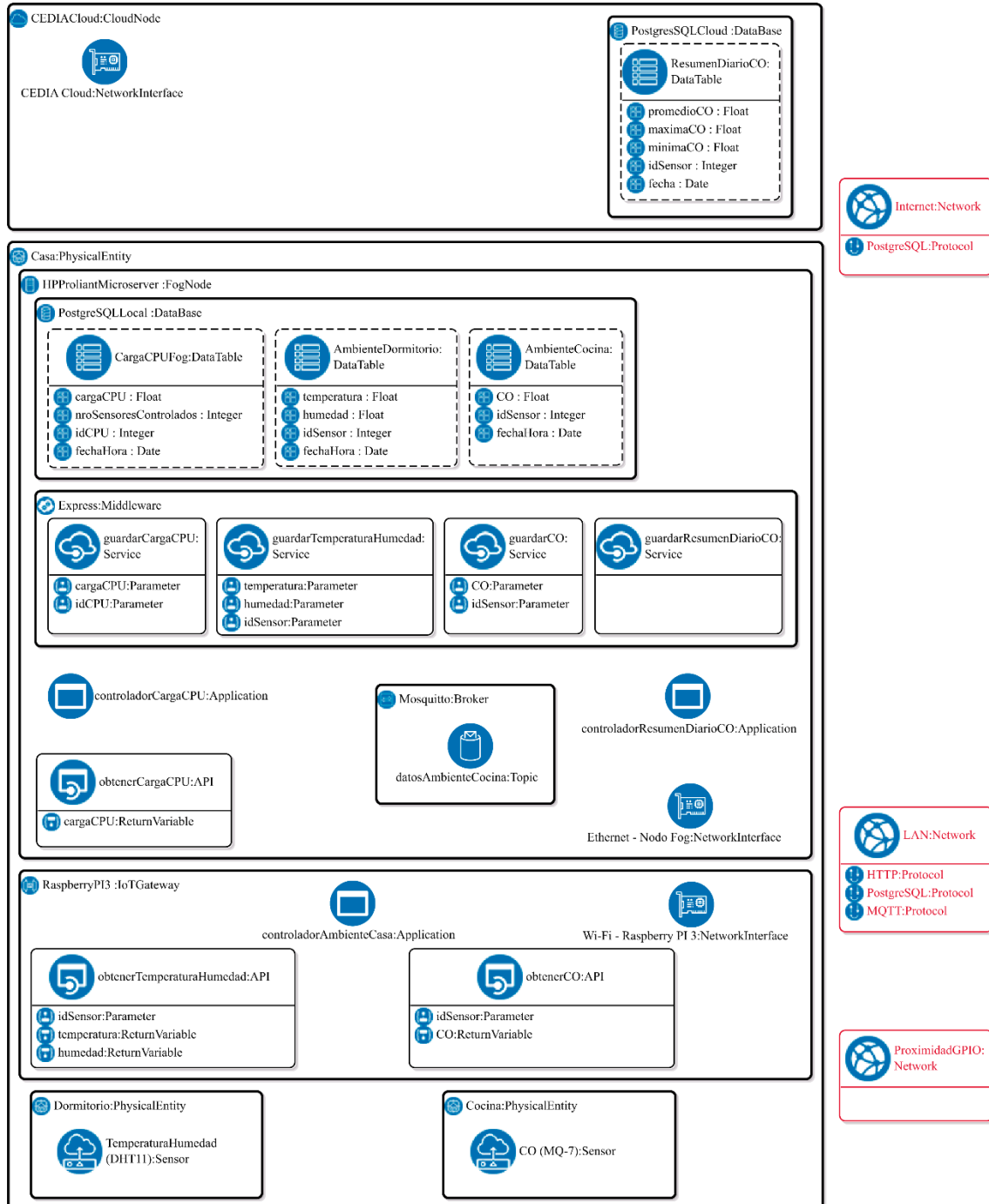


Figura 6-13. Modelado de las redes y protocolos de comunicación del escenario de IoT. Fuente: Elaboración propia.

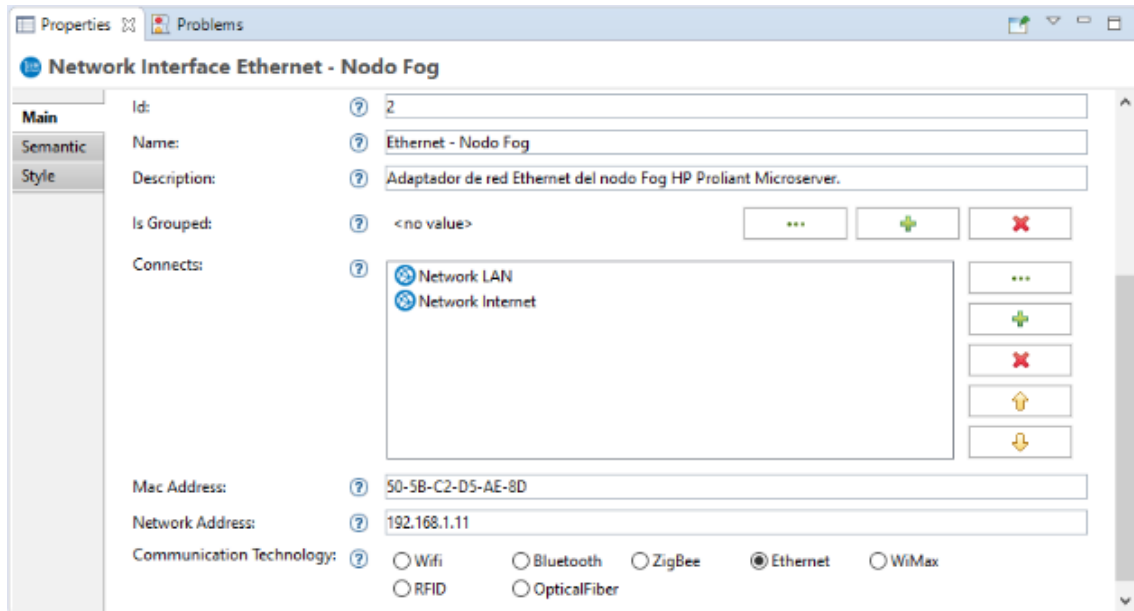


Figura 6-14. Ventana de propiedades con la configuración de la interfaz de red del nodo Fog HP Proliant Microserver. Fuente: Elaboración propia.

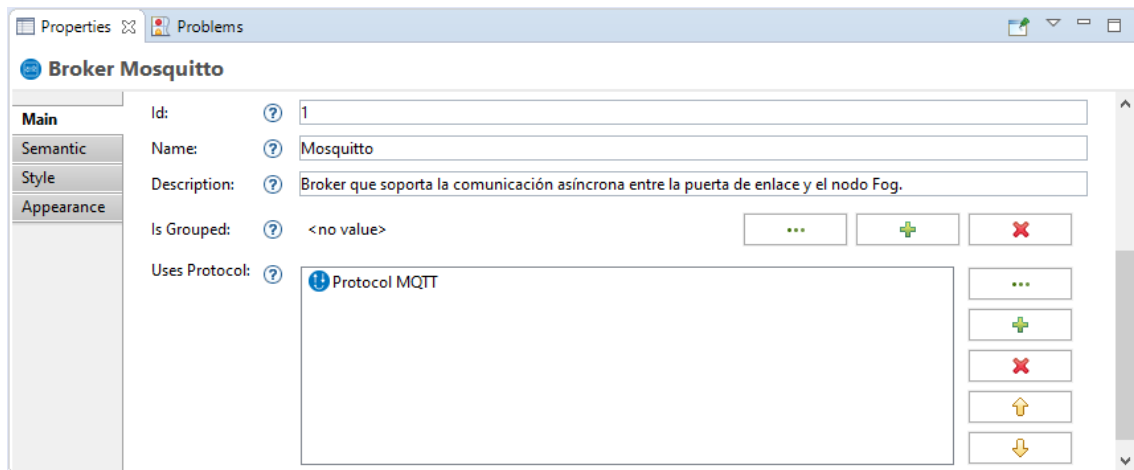


Figura 6-15. Ventana de propiedades con la configuración del broker Mosquitto. Fuente: Elaboración propia.

Subtarea 1.2.6. Configuración de categorías de entidades

Como una buena práctica para organizar de mejor manera el modelo de arquitectura se recomienda agrupar las propiedades y sus plantillas en categorías de entidades. A manera de ejemplo, en este escenario se han creado tres categorías de entidad: i) *habitación*, que agrupa el dormitorio y la cocina de la casa; ii) *sensor*, que agrupa los sensores físicos *DHT11* y *MQ-7*; y, ii) *servidor*, que agrupa la puerta de enlace *Raspberry PI 3* y los nodos Fog y Cloud. Estas categorías contienen las plantillas de las propiedades que pueden ser monitoreadas o utilizadas como metadatos para este tipo de entidades (ver Fig. 6-16). A su vez, en la Figura 6-17 se presentan los atributos establecidos para la categoría *habitación*.

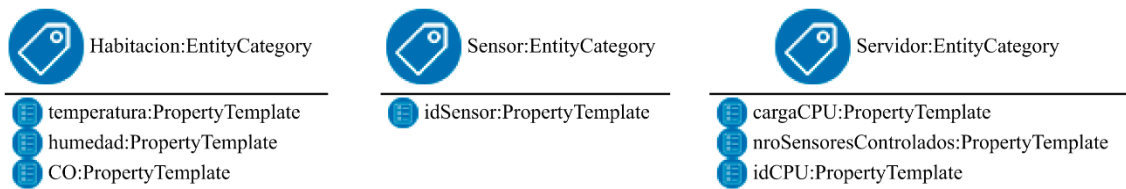


Figura 6-16. Modelado de categorías de entidades del escenario de IoT.

Fuente: Elaboración propia.

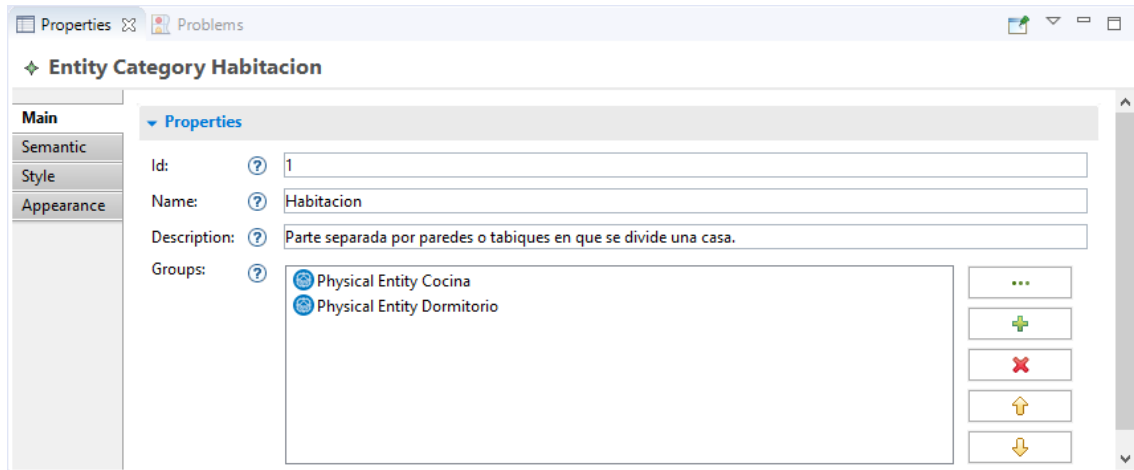


Figura 6-17. Ventana de propiedades con los atributos de la categoría habitación.

Fuente: Elaboración propia.

Subtarea 1.2.7. Configuración de propiedades de entidades

En primer lugar, se crean las plantillas con la información general de las propiedades a monitorear (telemetría, estado) y metadatos a utilizar en el escenario de IoT. Además, mediante las plantillas se mapean, en un solo paso, todas las propiedades comunes con los parámetros de entrada o variables de retorno de las APIs y servicios web en las que serán utilizadas. Luego, con base en las plantillas, se crean las propiedades y metadatos particulares para cada una de las entidades (física o digital) modeladas previamente. En la Figura 6-16 se presentan las plantillas de las propiedades *Temperatura*, *Humedad*, *CO*, *idSensor*, *CargaCPU*, *NroSensoresControlados*, *idCPU* agrupadas dentro las categorías de entidad correspondientes. De las plantillas creadas, *temperatura*, *humedad* y *CO* son de tipo telemetría, *cargaCPU* de tipo estado; y, *nroSensoresControlados*, *idSensor* e *idCPU* de tipo metadato. Finalmente, a manera de ejemplo, en las Figuras 6-18 y 6-19 se observan los atributos establecidos para la plantilla *temperatura* y la correspondiente propiedad particular asignada a la entidad *dormitorio*, respectivamente.

Tarea 1.3. Modelado de flujos de datos

En esta última tarea se diseña cómo interactúan las entidades digitales para soportar los flujos de datos requeridos por el escenario de IoT, de tal forma que los datos recolectados por las fuentes (sensores físicos o digitales) sean transportados, procesados y almacenados, con diferentes niveles de agregación, en las bases de datos destino. Para completar esta tarea se ejecutan dos subtareas:

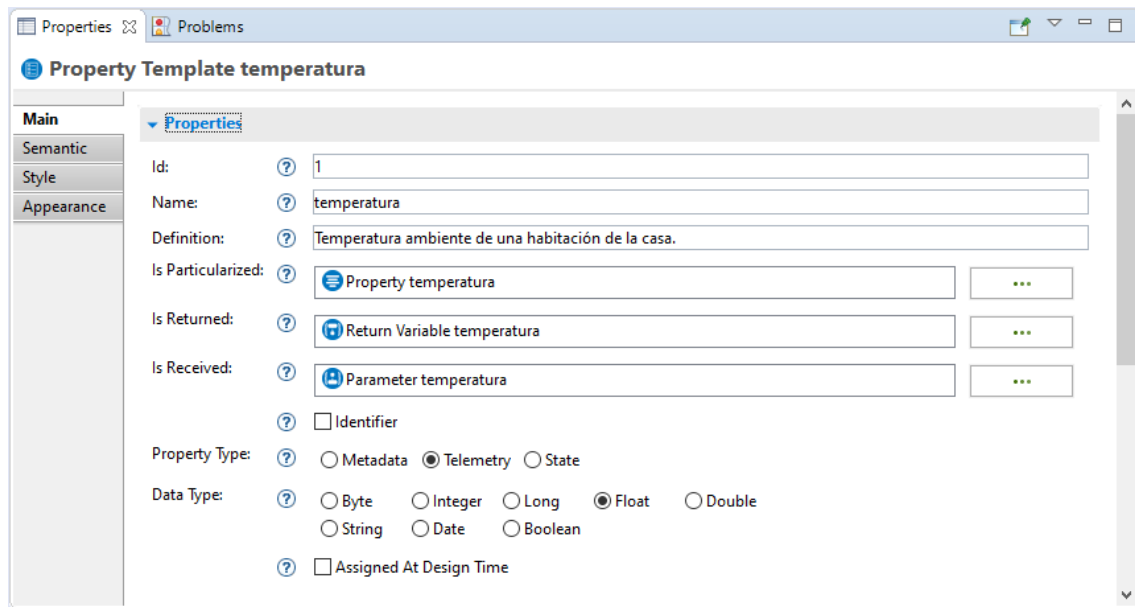


Figura 6-18. Ventana de propiedades con la configuración de la plantilla temperatura.
Fuente: Elaboración propia.

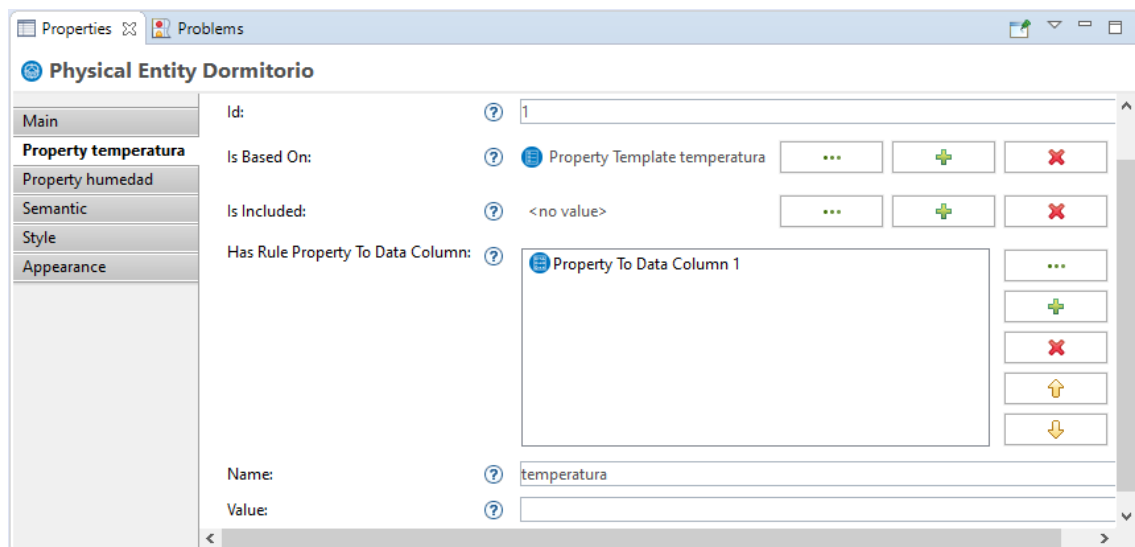


Figura 6-19. Ventana de propiedades con la configuración de la propiedad temperatura a monitorear en el dormitorio. Fuente: Elaboración propia.

Subtarea 1.3.1. Mapeo de datos

Crea los flujos de recolección y agregación de datos identificados en la subtarea 1.1.4. A su vez, para cada flujo de recolección de datos se agregan las reglas de mapeo que asocian las propiedades (a monitorear o los metadatos a utilizar en el flujo de datos) con las correspondientes tablas y columnas de base de datos donde se almacenarán. Asimismo, para cada flujo de agregación de datos se añaden las reglas de mapeo que asocian las columnas origen, a las que se aplican las operaciones de agregación, con las columnas destino donde almacenarán los datos resumidos. En la Figura 6-20 se presentan los cuatro flujos de recolección de datos y un flujo de agregación de datos que soportan los requerimientos de monitoreo del escenario de IoT, conteniendo sus reglas de mapeo. Por su parte, en la Figura 6-21 se muestra la parametrización del flujo de recolección de datos de temperatura y humedad, incluyendo entre otros atributos: el tipo de flujo de datos

(*recolección*), el tipo de comunicación que utiliza (*síncrona*); y, la unidad e intervalo de tiempo de ejecución (*cada 10 minutos*).

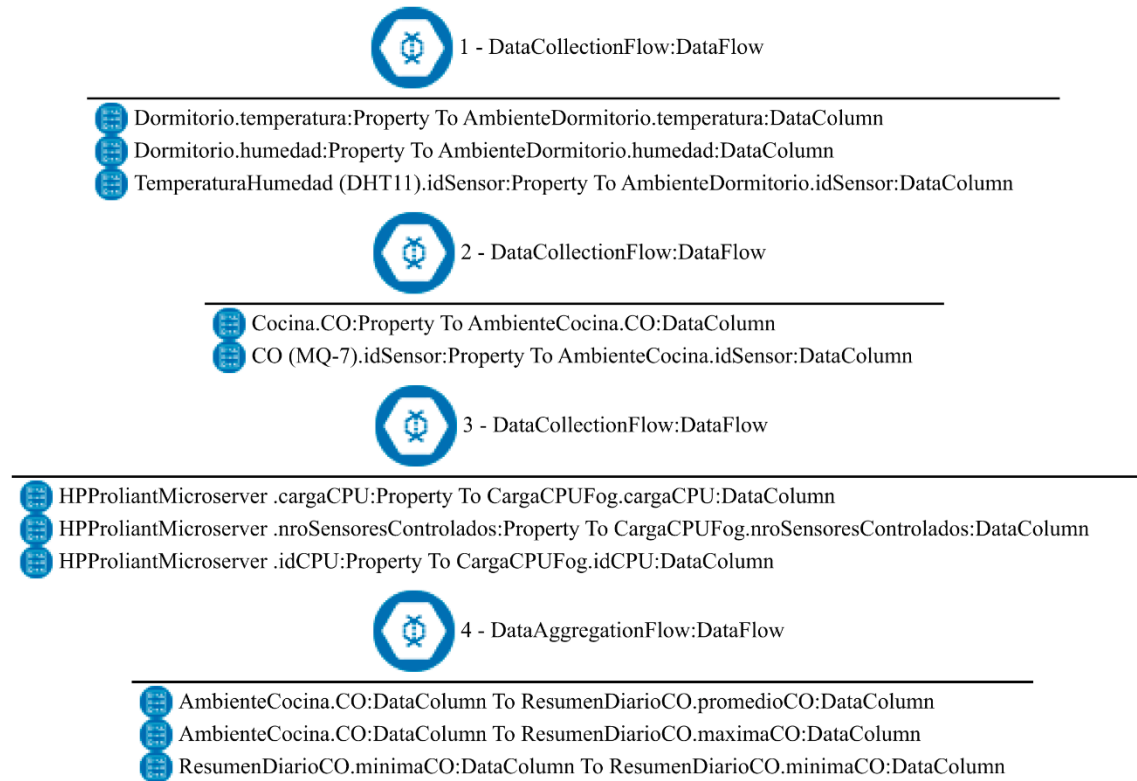


Figura 6-20. Modelado de flujos de datos y reglas de mapeo del escenario de IoT.
Fuente: Elaboración propia.

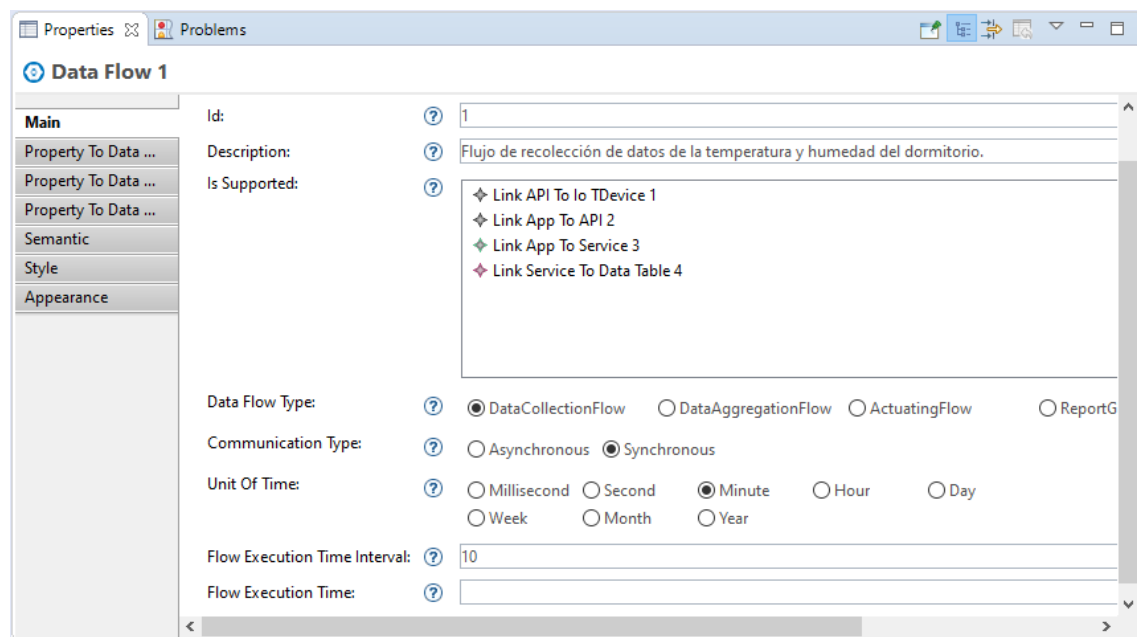


Figura 6-21. Ventana de propiedades con la parametrización del flujo de recolección de datos de temperatura y humedad. Fuente: Elaboración propia.

Subtarea 1.3.2. Modelado de enlaces entre entidades digitales

Diseña los enlaces de comunicación entre las entidades digitales. A su vez, estos enlaces son organizados secuencialmente en cada uno de los flujos de datos, a fin de determinar su trayectoria y las entidades que intervienen. En la Figura 6-22 se observan los enlaces de comunicación modelados entre las entidades digitales para el escenario de IoT; mientras que, en la Figura 6-21 se presenta de forma ordenada los enlaces de comunicación que soportan (atributo *Is Supported*) el flujo de recolección de datos de temperatura y humedad.

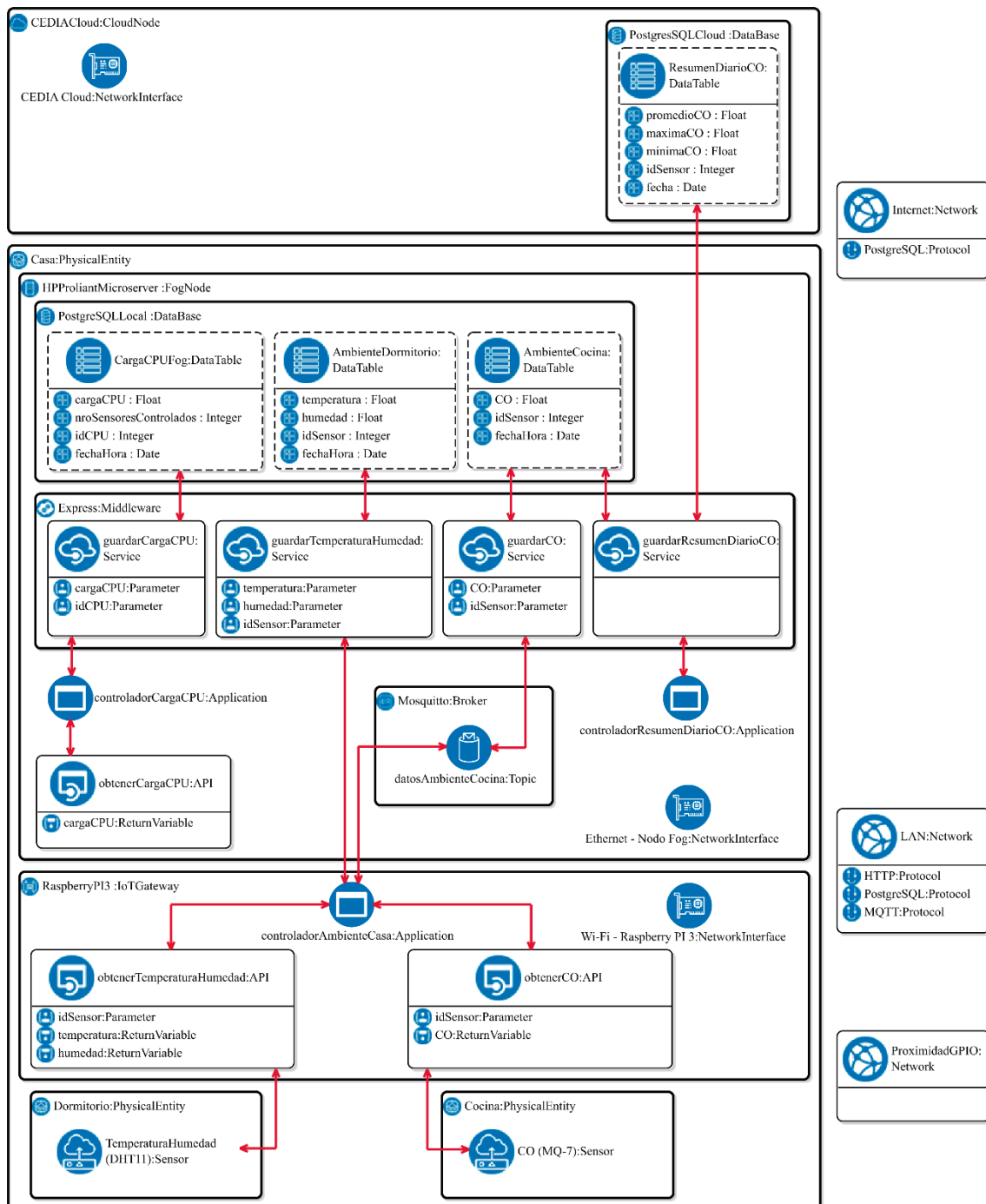


Figura 6-22. Modelado de los enlaces de comunicación entre las entidades digitales del escenario de IoT. Fuente: Elaboración propia.

Finalmente, en el Anexo C se presenta el modelo de arquitectura de monitoreo resultante del estudio de caso, conteniendo todos los componentes obtenidos de las tareas ejecutadas en la actividad 1.

Actividad 2. Configuración de las capacidades de autoconsciencia del sistema IoT

De acuerdo con los requerimientos de autoconsciencia del estudio de caso, esta actividad tiene como alcance la construcción del modelo de capacidades de autoconsciencia del Subsistema de Control Ambiental (SCA), con el apoyo del configurador web de autoconsciencia. Para este cometido, se deben realizar cuatro tareas: i) *Configuración del dominio de autoconsciencia*; ii) *Definición de los recursos de soporte a los procesos de autoconsciencia*; iii) *Definición de los procesos y métodos de autoconsciencia (aprendizaje y razonamiento)*; y; iv) *Generación del modelo de autoconsciencia*. Como resultado de esta actividad, las especificaciones de autoconsciencia se incorporan automáticamente al modelo de arquitectura de monitoreo y se genera un único archivo en formato JSON conteniendo ambos modelos.

Previo a realizar las tareas mencionadas, es necesario crear un usuario para acceder al configurador web de autoconsciencia. A su vez, se requiere crear un área o modelo de trabajo para configurar de forma independiente las especificaciones de autoconsciencia del subsistema SCA. Esta área de trabajo debe estar vinculada con el modelo de arquitectura de monitoreo resultante de la Actividad 1, a fin de que sus recursos estén disponibles durante las tareas posteriores de configuración. En la Figura 6-23 se presenta el formulario *Registro* con los datos del usuario a utilizar para la construcción del modelo de autoconsciencia del subsistema SCA. Mientras que, en la Figura 6-24 se muestra un formulario, sección *Crear un nuevo modelo (área de trabajo) de autoconsciencia*, en el que se especifica el nombre, descripción y autor del modelo de autoconsciencia a construir, así como el archivo XMI (*ModeloArquitecturaMonitoreoSCA.MonitorIoT*) con el modelo de arquitectura de monitoreo del escenario de IoT a vincular.

Figura 6-23. Registro del usuario de acceso.
Fuente: Elaboración propia.

Figura 6-24. Creación del modelo de autoconsciencia del subsistema SCA.
Fuente: Elaboración propia.

Tarea 2.1. Configuración del dominio de autoconsciencia

Define los sujetos, objetos y aspectos de autoconsciencia, según los requerimientos del escenario de IoT. Para completar esta tarea se ejecutan tres subtareas:

Subtarea 2.1.1. Definición de sujetos de autoconsciencia

De acuerdo con el alcance del escenario de IoT, se selecciona como sujeto de autoconsciencia al subsistema SCA (ver Fig. 6-25).

Figura 6-25. Definición de sujeto de autoconsciencia del escenario de IoT.
Fuente: Elaboración propia.

Subtarea 2.1.2. Definición de objetos de autoconsciencia

Selecciona los objetos (entidades físicas o digitales del modelo de arquitectura) sobre los que el subsistema SCA será consciente, siendo específicamente el *dormitorio* y la *cocina* de la casa (ver Fig. 6-26), así como el nodo Fog *HP Proliant Microserver*.

Figura 6-26. Definición de objetos de autoconsciencia de tipo entidad física del escenario de IoT. Fuente: Elaboración propia.

Subtarea 2.1.3. Definición de aspectos de autoconsciencia

Especifica los aspectos sobre los cuales el subsistema SCA será consciente. De ahí que, según lo establecido en el escenario de IoT, las Tablas de la 6-20 a la 6-22 contienen las especificaciones de los aspectos de autoconsciencia de los objetos dormitorio, cocina y nodo Fog, respectivamente. Asimismo, como ejemplo, la Figura 6-27 presenta el formulario *Agregar aspecto de autoconsciencia individual* del configurador web con el detalle de los atributos del aspecto *concentración en tiempo real de monóxido de carbono (CO) en el ambiente de la cocina*.

Tabla 6-20. Especificación de aspectos de autoconsciencia del objeto Dormitorio.

Aspecto de autoconsciencia	Descripción	Atributos				
		Peso (nivel de importancia)	Tipo	Sujeto que captura el aspecto	Categoría de objetos	Objetos sobre los que se captura el aspecto
Temperatura ambiente en tiempo real del dormitorio.	Conocer la temperatura ambiente en tiempo real del dormitorio del adulto mayor y comprender el nivel de satisfacción del usuario.	0.4	Estado	SCA	Entidad física	Dormitorio
Variabilidad diaria de la temperatura ambiente del dormitorio.	Conocer la variabilidad diaria de la temperatura ambiente del dormitorio del adulto mayor y el nivel de aceptabilidad del usuario.	0.4	Estado	SCA	Entidad física	Dormitorio

Tabla 6-21. Especificación de aspectos de autoconsciencia del objeto Cocina.

Aspecto de autoconsciencia	Descripción	Atributos				
		Peso (nivel de importancia)	Tipo	Sujeto que captura el aspecto	Categoría de objetos	Objetos sobre los que se captura el aspecto
Concentración en tiempo real de CO en el ambiente de la cocina.	Conocer la concentración en tiempo real de monóxido de carbono (CO) en el ambiente de la cocina e intuir el nivel de riesgo que tiene para la salud y vida del adulto mayor.	0.6	Estado	SCA	Entidad física	Cocina
Concentración promedio diario de CO en el ambiente de la cocina.	Conocer el promedio diario de concentración de monóxido de carbono (CO) en el ambiente de la cocina y entender su nivel de riesgo.	0.6	Estado	SCA	Entidad física	Cocina

Tabla 6-22. Especificación de aspectos de autoconsciencia del objeto HP Proliant Microserver.

Aspecto de autoconsciencia	Descripción	Atributos				
		Peso (nivel de importancia)	Tipo	Sujeto que captura el aspecto	Categoría de objetos	Objetos sobre los que se captura el aspecto
Carga de procesamiento en tiempo real de la CPU del nodo Fog.	Determinar la carga de procesamiento en tiempo real (alta, media, baja) de la CPU del nodo Fog.	0.6	Estado	SCA	Nodo Fog	HP Proliant Microserver
Predecir la carga de procesamiento de la CPU del nodo Fog.	Predecir la carga de procesamiento de la CPU del nodo Fog para gestionar un cierto número de sensores a futuro.	0.4	Predicción	SCA	Nodo Fog	HP Proliant Microserver

Agregar Aspecto de Autoconsciencia Individual

Nombre:

Concentración en tiempo real de CO en el ambiente de la cocina

Descripción:

Conocer la concentración en tiempo real de monóxido de carbono (CO) en el ambiente de la cocina e intuir el nivel de riesgo que tiene para la salud y vida del adulto mayor

Peso:

0.6

Tipo:

ESTADO

Activo:

☒

Sujeto que captura el aspecto:

☐ Emergencia
☒ ControlAmbiental

Categoría de objetos:

Entidades Físicas

Objetos sobre los que se captura el aspecto:

☐ Casa
☐ Dormitorio
☒ Cocina

Guardar

Cancelar

Figura 6-27. Agregar aspecto de autoconsciencia individual: Concentración en tiempo real de monóxido de carbono (CO) en el ambiente de la cocina. Fuente: Elaboración propia.

Tarea 2.2. Definición de recursos de soporte a los procesos de autoconsciencia

En esta tarea, se registran las escalas, unidades de medida, métricas, criterios de decisión, umbrales y recursos de implementación (funciones o servicios web) necesarios para la operacionalización de los procesos de autoconsciencia pre-reflexivos y reflexivos requeridos por el subsistema SCA dentro del escenario de IoT. Para este cometido se ejecutan tres subtareas:

Subtarea 2.2.1. Definición de escalas, unidades de medida y métricas

Determina las métricas que utilizarán los procesos y métodos (recolección, cálculo y modelo de análisis) para evaluar los aspectos de autoconsciencia previamente definidos. Las Tablas de la 6-23 a la 6-25 contienen las especificaciones de las métricas a utilizar en el escenario de IoT, organizadas por objeto de autoconsciencia. A su vez, a modo de ejemplo, la Figura 6-28 muestra el formulario *Agregar métrica* con los atributos de la métrica *nivel de riesgo de la concentración de CO*.

Tabla 6-23. Especificación de métricas para evaluar los aspectos de autoconsciencia del objeto Dormitorio.

Métrica	Descripción	Atributos				
		Abreviatura	Escala	Unidad de medida	Tipo	Perspectiva
Grados Celsius de temperatura ambiente actual.	Grados Celsius de temperatura ambiente en el momento actual.	GCTAA	Cuantitativa, continua, intervalo	Grados Celsius	Directa	

Grados Celsius de variabilidad diaria de la temperatura ambiente.	Grados Celsius de variabilidad diaria (desviación estándar) de la temperatura ambiente.	GCVDTA	Cuantitativa, continua, intervalo	Grados Celsius	Indirecta	
Nivel de satisfacción del usuario sobre la temperatura ambiente.	Nivel de satisfacción de la temperatura ambiente para el usuario, a fin de determinar si es necesario alguna acción.	NSUTA	Cuantitativa, discreta, ratio	Porcentaje	Indicador	Positiva
Nivel de aceptabilidad de la variabilidad diaria de la temperatura ambiente.	Nivel de aceptabilidad de la variabilidad diaria de la temperatura ambiente, a fin de determinar si es necesario alguna acción.	NAVDTA	Cuantitativa, discreta, ratio	Porcentaje	Indicador	Positiva

Tabla 6-24. Especificación de métricas para evaluar los aspectos de autoconsciencia del objeto Cocina.

Métrica	Descripción	Atributos				
		Abreviatura	Escala	Unidad de medida	Tipo	Perspectiva
Partes por millón de concentración actual de CO.	Partes por millón de concentración actual de monóxido de carbono (CO) en el ambiente.	PPMCACO	Cuantitativa, continua, ratio	Partes por millón	Directa	
Partes por millón de concentración promedio diario de CO.	Partes por millón de concentración promedio diario de monóxido de carbono (CO) en el ambiente.	PPMCPDO	Cuantitativa, continua, ratio	Partes por millón	Indirecta	
Nivel de riesgo de la concentración de CO.	Nivel de riesgo de la concentración de monóxido de carbono (CO) para el usuario, a fin de determinar si es necesario alguna acción.	NRCCO	Cuantitativa, continua, ratio	Porcentaje	Indicador	Negativa

Tabla 6-25. Especificación de métricas para evaluar los aspectos de autoconsciencia del objeto nodo Fog HP Proliant Microserver.

Métrica	Descripción	Atributos				
		Abreviatura	Escala	Unidad de medida	Tipo	Perspectiva
Porcentaje de carga de procesamiento actual de la CPU.	Porcentaje de uso o carga de procesamiento de la CPU en el momento actual.	PCPACPU	Cuantitativa, continua, ratio	Porcentaje	Directa	
Porcentaje estimado de carga de procesamiento de la CPU.	Porcentaje estimado o predicho de uso de la CPU para controlar una cierta cantidad de sensores en el futuro.	PECPCPU	Cuantitativa, continua, ratio	Porcentaje	Indirecta	
Nivel de carga de procesamiento de la CPU.	Nivel de uso de la CPU, a fin de determinar si es necesario alguna acción.	NCPCPU	Cuantitativa, continua, ratio	Porcentaje	Indicador	Negativa

Agregar Métrica

Nombre: Nivel de riesgo de la concentración de CO

Descripción:

Nivel de riesgo de la concentración de monóxido de carbono (CO) para el usuario, a fin de determinar si es necesario alguna acción

Abreviatura: NRCCO

Escala:

CUANTITATIVA, CONTINUA, RATIO

Unidad de medida:

PORCENTAJE

Tipo:

INDICADOR

Perspectiva:

NEGATIVA

Activo:

☒

Guardar

Cancelar

Figura 6-28. Agregar métrica: Nivel de riesgo de CO.
Fuente: Elaboración propia.

Subtarea 2.2.2. Definición de criterios de decisión y umbrales

Establece los criterios de decisión (umbrales) para evaluar cualitativamente los aspectos de autoconsciencia, conforme a los condicionantes establecidos en el escenario de IoT (ver Tabla 6-1). En este sentido, la Tabla 6-26 incluye las especificaciones de los criterios de decisión a utilizar en los modelos de análisis de los procesos de autoconsciencia. Además, la Figura 6-29 presenta en el configurador web, el criterio de decisión *nivel de riesgo de la concentración de CO*, junto con el detalle de sus umbrales.

Tabla 6-26. Especificación de criterios de decisión y sus umbrales para evaluar cualitativamente los aspectos de autoconsciencia.

Criterio de decisión	Descripción	Umbrales			
		Nombre	Interpretación	Inferior	Superior
Nivel de satisfacción del usuario sobre la temperatura ambiente.	Evalúa cualitativamente el nivel de satisfacción del usuario con la temperatura ambiente.	Satisfactorio	Temperatura ambiente confortable para el usuario.	100%	100%
		Insatisfactorio	Temperatura ambiente no confortable para el usuario.	0%	0%
Nivel de aceptabilidad de la variabilidad de la temperatura ambiente.	Evalúa cualitativamente el nivel de aceptabilidad de la variabilidad de la temperatura ambiente.	Aceptable	Aceptable nivel de variabilidad de la temperatura ambiente.	100%	100%
		Inaceptable	Inaceptable nivel de variabilidad de la temperatura ambiente.	0%	0%
Nivel de riesgo de la concentración de CO.	Evalúa cualitativamente el nivel de riesgo para el usuario de la concentración de monóxido de carbono (CO) en el ambiente.	Ausencia de riesgo	El usuario no sufre ningún síntoma ante una exposición prolongada.	0%	3.5%
		Riesgo medio	El usuario puede experimentar dolor de cabeza, pecho, fatiga o náuseas.	3.5%	20%
		Riesgo alto	El usuario puede experimentar desorientación, pérdida del conocimiento e incluso la muerte.	20%	
Nivel de carga de procesamiento de la CPU.	Evalúa cualitativamente el nivel de uso de la CPU.	Baja	Baja carga de procesamiento de la CPU.	0%	30%
		Normal	Normal carga de procesamiento de la CPU.	30%	80%
		Alta	Alta carga de procesamiento de la CPU.	80%	



Figura 6-29. Configuración del criterio de decisión: nivel de riesgo de la concentración de CO. Fuente: Elaboración propia.

Subtarea 2.2.3. Definición de recursos de implementación

Determina las funciones y servicios web a utilizar por los métodos de cálculo o modelos de análisis para calcular las métricas indirectas e indicadores cuantitativos, respectivamente. Como resultado de esta subtarea, en la Tabla 6-27 se presenta la configuración de estos recursos de implementación genéricos. Asimismo, en la Figura 6-30 se ejemplifica la configuración del servicio web *obtenerModeloRegresion LinealSimple* que construye un modelo de regresión lineal simple para extrapolar una variable dependiente (p. ej., carga de procesamiento de la CPU) en función de una variable independiente (p. ej., cantidad de sensores controlados). Cabe mencionar que el recurso de implementación de tipo fórmula no ha sido considerado en este estudio de caso, por lo que será abordado y validado en trabajos futuros.

Tabla 6-27. Especificación de recursos de implementación genéricos.

Recurso de implementación	Descripción	Atributos			
obtenerPromedio	Obtiene la media o promedio de un conjunto de datos.	Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	datos	Float	Falso
		Instrucciones: const math = require('mathjs'); return math.mean(datos); Tipo de dato de salida: Float			
obtenerDesviacionEstandar	Obtiene la desviación estándar de un conjunto de datos.	Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	datos	Float	Falso
		Instrucciones: const math = require('mathjs'); return math.std(datos); Tipo de dato de salida: Float			
obtenerRegresion LinealSimple	Construye un modelo de regresión lineal simple para extrapolar una variable dependiente a partir de una variable independiente.	Tipo de recurso: Servicio web			
		Nodo de computación: Nodo Fog HP Proliant Microserver			
		Middleware (servidor de aplicaciones): Express			
		Endpoint: obtenerregresionlinealsimple			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	varDependiente	Float	Falso
		2	varIndependiente	Float	Falso
		3	varSimulacion	Float	Falso
		Instrucciones: const ml = require('ml-regression'); const regression = new ml.SimpleLinearRegression(varIndependiente,varDependiente); return regression.predict(varSimulacion); Tipo de dato de salida: Float Formato de salida: JSON			
obtenerIndicadorMetrica Porcentual	Asigna una métrica porcentual (sea directa o indirecta) como indicador.	Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	metricaPorcentual	Float	Falso
		Instrucciones: return metricaPorcentual; Tipo de dato de salida: Float			
obtenerNivelSatisfaccion Temperatura	Obtiene el nivel de satisfacción porcentual del usuario sobre la temperatura ambiente.	Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	temperatura	Float	Falso
		Instrucciones: if (temperature>=18 && temperature<=22) return 100; else return 0; Tipo de dato de salida: Integer			
obtenerNivelAceptabilidad Temperatura	Obtiene el nivel de aceptabilidad sobre la variabilidad de la temperatura ambiente.	Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	temperatura	Float	Falso
		Instrucciones: if (temperature>=0 && temperature<=4) return 100; else return 0; Tipo de dato de salida: Integer			

obtenerNivelRiesgoCO	Obtiene el nivel de riesgo porcentual de la concentración de monóxido de carbono (CO).	Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	monoxidoCarbono	Float	Falso
		Instrucciones:			
		return (monoxidoCarbono/1000)*100;			
		Tipo de dato de salida: Float			

Agregar Recurso de Implementación

Nombre:

obtenerRegresionLinealSimple

Descripción:

Construye un modelo de regresión lineal simple para extrapolar una variable dependiente, a partir de una variable independiente

Tipo de recurso

SERVICIO

EndPoint

Parámetros

Instrucciones

Tipo de salida

☐ Pre existente

```
const ml = require('ml-regression');
const regression = new ml.SimpleLinearRegression(varIndependiente,varDependiente);
return regression.predict(varSimulacion);
```

Guardar

Cancelar

Figura 6-30. Configuración del servicio web obtenerModeloRegresionLinealSimple.
Fuente: Elaboración propia.

Tarea 2.3. Definición de procesos y métodos de autoconsciencia (aprendizaje y razonamiento)

Con base en la información contenida en la Tabla 6-1 sobre los procesos pre-reflexivos y reflexivos que se deben implementar en el escenario de IoT, en esta tarea se procede a configurar dichos procesos, así como los métodos de recolección, cálculo y modelos de análisis que emplean. A su vez, en este punto se utilizan e integran como parte de los procesos, todos los recursos de soporte configurados en la tarea anterior, tales como: métricas, criterios de decisión y recursos de implementación. Para alcanzar este objetivo, la tarea se divide en tres subtareas:

Subtarea 2.3.1 Definición y configuración general de procesos de autoconsciencia

Específicamente, tres procesos pre-reflexivos y tres procesos reflexivos son configurados para dotar al subsistema SCA de capacidades de autoconsciencia, de acuerdo a lo requerido por el estudio de caso. En las Tablas 6-28 y 6-29 se presenta la configuración general de cada uno de estos procesos, clasificados por pre-reflexivos y reflexivos, respectivamente. De igual manera, las Figura 6-31 y 6-32 ejemplifican la parametrización del proceso pre-reflexivo que *captura en tiempo real el conocimiento sobre la concentración y nivel de riesgo de CO* y del proceso reflexivo que *construye un modelo de regresión lineal para predecir la carga de procesamiento requerida por la CPU del nodo Fog para controlar el doble de sensores que operan actualmente*.

Tabla 6-28. Información general de los procesos de autoconsciencia pre-reflexivos.

PID	Descripción	Atributos generales
1	Captura el conocimiento sobre la temperatura ambiente actual (en tiempo real) del dormitorio y el nivel de satisfacción del usuario.	Tipo de ejecución: Predefinido
		Fecha de inicio de ejecución: 01/Enero/2023
		Fecha de fin de ejecución: 31/Diciembre/2023
		Unidad de tiempo de ejecución: Minuto
		Intervalo de ejecución: 10
		Hora de ejecución (opcional): No aplica
		Sujeto que ejecuta el proceso de autoconsciencia: Subsistema SCA.
		Nodo de computación en el que se ejecuta el proceso: Nodo Fog HP Proliant Microserver
		Aspecto de autoconsciencia a capturar: Temperatura ambiente en tiempo real del dormitorio.
2	Captura el conocimiento sobre la concentración y nivel de riesgo actual (en tiempo real) de CO en el ambiente de la cocina.	Objetos sobre los que se captura el aspecto de autoconsciencia: Dormitorio.
		Tipo de ejecución: Predefinido
		Fecha de inicio de ejecución: 01/Enero/2023
		Fecha de fin de ejecución: 31/Diciembre/2023
		Unidad de tiempo de ejecución: Minuto
		Intervalo de ejecución: 10
		Hora de ejecución (opcional): No aplica
		Sujeto que ejecuta el proceso de autoconsciencia: Subsistema SCA.
		Nodo de computación en el que se ejecuta el proceso: Nodo Fog HP Proliant Microserver
3	Captura el conocimiento sobre la carga de procesamiento de la CPU del nodo fog en tiempo real.	Aspecto de autoconsciencia a capturar: Concentración en tiempo real de CO en el ambiente de la cocina.
		Objetos sobre los que se captura el aspecto de autoconsciencia: Cocina.
		Tipo de ejecución: Predefinido
		Fecha de inicio de ejecución: 01/Enero/2023
		Fecha de fin de ejecución: 31/Diciembre/2023
		Unidad de tiempo de ejecución: Minuto
		Intervalo de ejecución: 10
		Hora de ejecución (opcional): No aplica
		Sujeto que ejecuta el proceso de autoconsciencia: Subsistema SCA.
		Nodo de computación en el que se ejecuta el proceso: Nodo Fog HP Proliant Microserver
		Aspecto de autoconsciencia a capturar: Carga de procesamiento en tiempo real de la CPU del nodo Fog.
		Objetos sobre los que se captura el aspecto de autoconsciencia: Nodo Fog HP Proliant Microserver.

Tabla 6-29. Información general de los procesos de autoconsciencia reflexivos.

PID	Descripción	Atributos generales
4	Captura el conocimiento sobre la variabilidad diaria y nivel de aceptabilidad de la temperatura ambiente del dormitorio.	Tipo de ejecución: Predefinido
		Fecha de inicio de ejecución: 01/Enero/2023
		Fecha de fin de ejecución: 31/Diciembre/2023
		Unidad de tiempo de ejecución: Día
		Intervalo de ejecución: 1
		Hora de ejecución (opcional): 23h55
		Sujeto que ejecuta el proceso de autoconsciencia: Subsistema SCA.
		Alcance de la autoconsciencia: Individual
		Nodo de computación en el que se ejecuta el proceso: Nodo Fog HP Proliant Microserver
5	Captura el conocimiento sobre la concentración y nivel de riesgo promedio diario de CO en el ambiente de la cocina.	Aspecto de autoconsciencia a capturar: Variabilidad diaria de la temperatura ambiente del dormitorio.
		Objetos sobre los que se captura el aspecto de autoconsciencia: Dormitorio.
		Tipo de ejecución: Predefinido
		Fecha de inicio de ejecución: 01/Enero/2023
		Fecha de fin de ejecución: 31/Diciembre/2023
		Unidad de tiempo de ejecución: Día
		Intervalo de ejecución: 1
		Hora de ejecución (opcional): 23h55
		Sujeto que ejecuta el proceso de autoconsciencia: Subsistema SCA.
		Alcance de la autoconsciencia: Individual
		Nodo de computación en el que se ejecuta el proceso: Nodo Fog HP Proliant Microserver
		Aspecto de autoconsciencia a capturar: Concentración promedio diario de CO en el ambiente de la cocina.

6	Predice la carga de procesamiento requerida por la CPU del nodo Fog para controlar el doble de sensores que operan actualmente.	Objetos sobre los que se captura el aspecto de autoconsciencia: Cocina.
		Tipo de ejecución: Predefinido
		Fecha de inicio de ejecución: 01/Enero/2023
		Fecha de fin de ejecución: 31/Diciembre/2023
		Unidad de tiempo de ejecución: Día
		Intervalo de ejecución: 1
		Hora de ejecución (opcional): 23h55
		Sujeto que ejecuta el proceso de autoconsciencia: Subsistema SCA.
		Alcance de la autoconsciencia: Individual
		Nodo de computación en el que se ejecuta el proceso: Nodo Fog HP Proliant Microserver
		Aspecto de autoconsciencia a capturar: Prededir la carga de procesamiento de la CPU del nodo Fog.
		Objetos sobre los que se captura el aspecto de autoconsciencia: Nodo Fog HP Proliant Microserver.

Configurador de Autoconsciencia para Sistemas IoT

Área de Trabajo: Modelo Autoconsciencia SCA

Agregar Proceso de Autoconsciencia Pre-reflexivo

Paso 1: Información General

ID: PID2

Descripción: Captura el conocimiento sobre la concentración y nivel de riesgo actual (en tiempo real) de CO en el ambiente de la cocina.

Tipo Ejecución: PREDEFINIDO

Periodo de ejecución:

Inicio: 01/01/2023 Fin: 31/12/2023

Frecuencia de ejecución

Unidad de Tiempo: MINUTO Intervalo de ejecución: 10 Hora ejecución: --:--

Sujeto que ejecuta el proceso:

☐ Emergencia ☒ ControlAmbiental

Nodo de computación en el que se ejecuta el proceso: HPProliantMicroserver

Aspecto de autoconsciencia a capturar: Concentración en tiempo real de CO en el ambiente de la cocina

Categoría de objetos: Entidades Físicas

Objetos sobre los que se captura el aspecto autoconsciencia: Cocina

Guardar (Información General)

Figura 6-31. Configuración del proceso pre-reflexivo que captura en tiempo real el conocimiento sobre la concentración y nivel de riesgo de CO. Fuente: Elaboración propia.

Configurador de Autoconsciencia para Sistemas IoT

Área de Trabajo: Modelo Autoconsciencia SCA

Agregar Proceso de Autoconsciencia Reflexivo

Paso 1: Información General

ID: PID6

Descripción: Predice la carga de procesamiento requerida por la CPU del nodo Fog para controlar el doble de sensores que operan actualmente.

Tipo Ejecución: PREDEFINIDO

Periodo de ejecución:

Inicio: 01/01/2023 Fin: 31/12/2023

Frecuencia de ejecución

Unidad de Tiempo: DIA Intervalo de ejecución: 1 Hora ejecución: 23:55

Sujeto que ejecuta el proceso:

☐ Emergencia ☒ ControlAmbiental

Alcance de la autoconsciencia: INDIVIDUAL

Nodo de computación en el que se ejecuta el proceso: HPProliantMicroserver

Aspecto de autoconsciencia a capturar: Prededir la carga de procesamiento de la CPU del nodo Fog

Categoría de objetos: Nodos Fog

Objetos sobre los que se captura el aspecto autoconsciencia: HPProliantMicroserver

Guardar (Información General)

Figura 6-32. Configuración del proceso reflexivo que predice la carga de procesamiento de la CPU del nodo Fog. Fuente: Elaboración propia.

Subtarea 2.3.2. Definición de métodos de recolección

Para cada proceso pre-reflexivo, se configura un método de recolección, responsable de producir una métrica directa a partir de los datos recolectados desde los sensores físicos y virtuales. Como resultado, la Tabla 6-30 incluye la parametrización de estos métodos de recolección. Particularmente, la Figura 6-33 muestra la configuración del método de

recolección asociado al proceso pre-reflexivo que *captura el conocimiento sobre la concentración actual de CO en el ambiente de la cocina.*

Tabla 6-30. Parametrización de los métodos de recolección de los procesos pre-reflexivos.

PID	Proceso pre-reflexivo	Método de recolección asociado Atributos
1	Captura el conocimiento sobre la temperatura ambiente actual (en tiempo real) del dormitorio y el nivel de satisfacción del usuario.	Tipo de comunicación: Síncrona Propiedad: temperatura Flujo de Datos: Flujo de recolección de datos de la temperatura y humedad del dormitorio. Métrica directa de salida: Grados Celsius de temperatura ambiente actual.
2	Captura el conocimiento sobre la concentración y nivel de riesgo actual (en tiempo real) de CO en el ambiente de la cocina.	Tipo de comunicación: Asíncrona Propiedad: CO Flujo de Datos: Flujo de recolección de datos de la concentración de CO en la cocina. Métrica directa de salida: Partes por millón de concentración actual de CO.
3	Captura el conocimiento sobre la carga de procesamiento de la CPU del nodo fog en tiempo real.	Tipo de comunicación: Asíncrona Propiedad: cargaCPU Flujo de Datos: Flujo de recolección de datos de la carga de CPU del nodo Fog. Métrica directa de salida: Porcentaje de carga de procesamiento actual de la CPU.

Figura 6-33. Configuración del método de recolección del proceso pre-reflexivo sobre la concentración actual de CO en el ambiente de la cocina. Fuente: Elaboración propia.

Subtarea 2.3.3. Definición de métodos de cálculo

Para cada proceso reflexivo, se configura un método de cálculo, encargado de construir y/o aplicar un modelo conceptual o causal que, a partir de los datos recopilados históricamente, obtenga una métrica indirecta con el soporte de un recurso de implementación. La Tabla 6-31 presenta la parametrización de estos métodos de cálculo para el escenario de IoT, incluyendo el mapeo entre las métricas, metadatos o variables de simulación; y, los parámetros requeridos por cada uno de los recursos de implementación asociados a los métodos de cálculo (*obtenerDesviaciónEstandar*, *obtenerPromedio*, *obtenerRegresionLinealSimple*), a fin pasar los datos necesarios a estos recursos para su procesamiento interno. Como caso particular, previo al mapeo de los parámetros del servicio web *obtenerRegresionLinealSimple*, se crea el escenario y la variable de simulación requerida por este, empleando la información expuesta en la Tabla 6-32.

Tabla 6-31. Parametrización de los métodos de cálculo de los procesos reflexivos.

PID	Proceso reflexivo	Método de cálculo asociado			
		Atributos			
4	Captura el conocimiento sobre la variabilidad diaria y nivel de aceptabilidad de la temperatura ambiente del dormitorio.	Periodo de datos:			
		Unidad (opcional)	Intervalo (opcional)	Fecha de inicio (opcional)	Fecha de fin (opcional)
		Día	1		
		Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerDesviacionEstandar			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		datos	Float	Métrica Directa	Grados Celsius de temperatura ambiente actual.
		Métrica indirecta de salida: Grados Celsius de variabilidad diaria de la temperatura ambiente.			
5	Captura el conocimiento sobre la concentración y nivel de riesgo promedio diario del monóxido de carbono (CO) en el ambiente de la cocina.	Periodo de datos:			
		Unidad	Intervalo	Fecha de inicio (opcional)	Fecha de fin (opcional)
		Día	1		
		Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerPromedio			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		datos	Float	Métrica Directa	Partes por millón de concentración actual de CO.
		Métrica indirecta de salida: Partes por millón de concentración promedio diario de CO.			
6	Predice la carga de procesamiento requerida por la CPU del nodo Fog para controlar el doble de sensores que operan actualmente.	Periodo de datos:			
		Unidad (opcional)	Intervalo (opcional)	Fecha de inicio (opcional)	Fecha de fin (opcional)
				01/enero/2023	
		Tipo de recurso de implementación: Servicio web			
		Recurso de implementación: obtenerRegresionLinealSimple			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		varDependiente	Float	Métrica Directa	Porcentaje de carga de procesamiento actual de la CPU.
varIndependiente		Float	Metadata	nroSensoresControlados	
varSimulación		Float	Variable Simulación	nroSensoresSimular	
		Métrica indirecta de salida: Porcentaje estimado de carga de procesamiento de la CPU.			

Tabla 6-32. Especificación del escenario y variable de simulación para el servicio web obtenerRegresionLinealSimple.

Escenario de simulación	Descripción	Variable de simulación	Valor de simulación
Duplicar la cantidad de sensores que operan actualmente.	Escenario utilizado para predecir la carga de procesamiento de la CPU del nodo Fog para controlar el doble de sensores que operan actualmente.	nroSensoresSimular	4

A manera de ejemplo, la Figura 6-34 ilustra la configuración del método de cálculo vinculado al proceso reflexivo que *predice la carga de procesamiento requerida por la CPU del nodo Fog para controlar el doble de sensores que operan actualmente*. Asimismo, las Figuras 6-35 y 6-36 muestran el mapeo de parámetros del servicio web *obtenerRegresionLinealSimple*, así como la configuración del escenario y variable de simulación que se utiliza como argumento de entrada de este recurso.

Menú

Modelos

Dominio

Procesos de autoconsciencia

Pre-reflexivos

Reflexivos

Generales

Generar modelo de autoconsciencia

Área de Trabajo: Modelo Autoconsciencia SCA

Agregar Proceso de Autoconsciencia Reflexivo

Paso 1: Información General

Paso 2: Métodos de aprendizaje y razonamiento

Método de cálculoModelo de análisis de datos

Periodo de Datos

UnidadNINGUNO

Intervalo

Inicio01/01/2023

Findd/mm/aaaa

Tipo de recursoSERVICIO

RecursoobtenerRegresionLinealSimple

Variables de simulación

Escenarios de simulación

Mapeo de parámetros

Métrica indirecta de salida:Porcentaje estimado de carga de procesamiento de la CPU

Guardar (Métodos de aprendizaje y razonamiento)

Figura 6-34. Configuración del método de cálculo del proceso reflexivo que predice la carga de procesamiento de la CPU del nodo Fog. Fuente: Elaboración propia.

Mapeo Parámetros

Recurso Implementación:

Nombre:obtenerRegresionLinealSimple

Tipo dato de salida:FLOAT

Descripción:Construye un modelo de regresión lineal simple para extrapolar una variable dependiente a partir de una variable independiente

ID	Nombre	Tipo de dato	Opcional	Tipo de mapeo	Argumento de entrada	Procesos
3	varDependiente	FLOAT	true	DIRECTA	Porcentaje de carga de procesamiento actual de la CPU	PID3-Captura
4	varIndependiente	FLOAT	true	METADATA	nroSensoresControlados	
5	varSimulacion	FLOAT	true	VARIABLE SIMULACION	nroSensoresSimular	

GuardarCancelar

Figura 6-35. Mapeo de parámetros del recurso de implementación (servicio web) obtenerRegresionLinealSimple. Fuente: Elaboración propia.

Escenario de simulación

AgregarModificarEliminar

#	Id	Nombre	Descripción	Activo
1		Duplicar la cantidad de sensores que operan actualmente	Escenario utilizado para predecir la carga de procesamiento de la CPU del nodo Fog para controlar el doble de sensores que operan actualmente	☒

Variables y valores de simulación

AgregarModificarEliminar

#	Variable	Valor
	nroSensoresSimular	4

Cerrar

Figura 6-36. Configuración del escenario y variable de simulación para el servicio web obtenerRegresionLinealSimple. Fuente: Elaboración propia.

Subtarea 2.3.4. Definición de modelos de análisis

Tanto para los procesos pre-reflexivos como reflexivos, se configura un modelo de análisis para transformar la métrica (directa o indirecta) previamente calculada en un indicador cuantitativo (porcentual), con el apoyo de un recurso de implementación. Además, cada modelo de análisis se asocia con un criterio de decisión con el objeto de obtener el respectivo indicador cualitativo; y, de requerir el escenario de IoT, se establecen las posibles acciones o recomendaciones prescriptivas para cada umbral. En este sentido, las Tabla 6-33 y 6-34 contienen la parametrización de los modelos de análisis para los procesos pre-reflexivos y reflexivos, respectivamente. De igual manera, estas tablas incluyen el mapeo entre las métricas y los parámetros requeridos por cada uno de los recursos de implementación asociados a los modelos de análisis (*obtenerNivelSatisfaccionTemperatura*, *obtenerNivelAceptabilidadTemperatura*, *obtenerNivelRiesgoCO*, *obtenerIndicadorMetricaPorcentual*). Como se puede observar en las Tablas varios recursos de implementación son utilizados por más de un modelo de análisis. Finalmente, estas tablas también muestran la configuración de las recomendaciones prescriptivas para aquellos umbrales que lo requieran.

Tabla 6-33. Parametrización de los modelos de análisis de los procesos pre-reflexivos.

PID	Proceso reflexivo	Modelo de análisis asociado Atributos			
1	Captura el conocimiento sobre la temperatura ambiente actual (en tiempo real) del dormitorio y el nivel de satisfacción del usuario.	Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerNivelSatisfaccionTemperatura			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		temperatura	Float	Métrica Directa	Grados Celsius de temperatura ambiente actual.
		Indicador de salida: Nivel de satisfacción del usuario sobre la temperatura ambiente.			
		Criterio de decisión: Nivel de satisfacción del usuario sobre la temperatura ambiente.			
		Umbral	Acción / Recomendación prescriptiva		
	Insatisfactorio	El subsistema SCA debe calibrar el aire acondicionado de la habitación (autoadaptación).			
2	Captura el conocimiento sobre la concentración y nivel de riesgo actual (en tiempo real) de CO en el ambiente de la cocina.	Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerNivelRiesgoCO			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		monoxidoCarbono	Float	Métrica Directa	Partes por millón de concentración actual de CO.
		Indicador de salida: Nivel de riesgo de la concentración de CO.			
		Criterio de decisión: Nivel de riesgo de la concentración de CO.			
		Umbral	Acción / Recomendación prescriptiva		
	Riesgo medio	El subsistema SCA debe encender una alerta auditiva intermitente. El subsistema SCA debe notificar a la central de monitoreo.			
	Riesgo alto	El subsistema SCA debe encender una alerta auditiva permanente. El subsistema SCA debe notificar a la central de monitoreo.			
3	Captura el conocimiento sobre la carga de procesamiento de la CPU del nodo fog en tiempo real.	Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerIndicadorMetricaPorcentual			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		metricaPorcentual	Float	Métrica Directa	Porcentaje de carga de procesamiento actual de la CPU.
		Indicador de salida: Nivel de carga de procesamiento de la CPU.			
		Criterio de decisión: Nivel de carga de procesamiento de la CPU.			
		Umbral	Acción / Recomendación prescriptiva		
	Baja	El subsistema SCA debe verificar la disponibilidad o frecuencia de lectura de los sensores (autorrecuperación). El subsistema SCA debe verificar la correcta operación de la red de comunicación (autorrecuperación). El subsistema SCA debe notificar a la central de monitoreo.			

		Alta	El subsistema SCA debe verificar la frecuencia de lectura de los sensores o en su defecto incrementar la capacidad de procesamiento del nodo fog (autoadaptación).
			El subsistema SCA debe notificar a la central de monitoreo.

Tabla 6-34. Parametrización de los modelos de análisis de los procesos reflexivos.

PID	Proceso reflexivo	Modelo de análisis asociado Atributos			
4	Captura el conocimiento sobre la variabilidad diaria y nivel de aceptabilidad de la temperatura ambiente del dormitorio.	Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerNivelAceptabilidadTemperatura			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		temperatura	Float	Métrica Indirecta	Grados Celsius de variabilidad diaria de la temperatura ambiente.
		Indicador de salida: Nivel de aceptabilidad de la variabilidad diaria de la temperatura ambiente.			
		Criterio de decisión: Nivel de aceptabilidad de la variabilidad de la temperatura ambiente.			
		Umbral	Acción / Recomendación prescriptiva		
	Inaceptable	El subsistema SCA debe notificar a la central de monitoreo.			
5	Captura el conocimiento sobre la concentración y nivel de riesgo promedio diario del monóxido de carbono (CO) en el ambiente de la cocina.	Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerNivelRiesgoCO			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		monoxidoCarbono	Float	Métrica Indirecta	Partes por millón de concentración promedio diario de CO..
		Indicador de salida: Nivel de riesgo de la concentración de CO.			
		Criterio de decisión: Nivel de riesgo de la concentración de CO.			
		Umbral	Acción / Recomendación prescriptiva		
		Riesgo medio	El subsistema SCA debe encender una alerta auditiva intermitente.		
	El subsistema SCA debe notificar a la central de monitoreo.				
Riesgo alto	El subsistema SCA debe encender una alerta auditiva permanente.				
	El subsistema SCA debe notificar a la central de monitoreo.				
6	Predice la carga de procesamiento requerida por la CPU del nodo Fog para controlar el doble de sensores que operan actualmente.	Tipo de recurso de implementación: Función			
		Recurso de implementación: obtenerIndicadorMetricaPorcentual			
		Parámetro del recurso de implementación	Tipo de dato del parámetro	Tipo de mapeo	Argumento de entrada
		metricaPorcentual	Float	Métrica Indirecta	Porcentaje estimado de carga de procesamiento de la CPU.
		Indicador de salida: Nivel de carga de procesamiento de la CPU.			
		Criterio de decisión: Nivel de carga de procesamiento de la CPU.			
		Umbral	Acción / Recomendación prescriptiva		
		Alta	Se deberá incrementar la capacidad de procesamiento del nodo Fog (autoadaptación).		

Finalmente, la Figura 6-37 contiene la configuración del modelo de análisis vinculado al proceso pre-reflexivo sobre la *concentración y nivel de riesgo actual del monóxido de carbono (CO) en el ambiente de la cocina*; mientras que, la Figura 6-38 presenta el registro de las recomendaciones prescriptivas para el modelo de análisis antes citado.

Tarea 2.4. Generación del modelo de autoconsciencia del sistema IoT

Con el apoyo del configurador, las especificaciones de autoconsciencia definidas en las tareas previas de esta actividad, se incorporan e integran al modelo de arquitectura de monitoreo del escenario de IoT, conforme al vocabulario y la sintaxis establecida en los metamodelos respectivos. A su vez, se genera un archivo en formato JSON (*ModelosArquitecturaAutoconsciencia.json*) conteniendo los modelos de arquitectura y autoconsciencia en tiempo de ejecución del escenario de IoT. Este archivo JSON es almacenado en el subdirectorio *modelos* ubicado en el directorio raíz de la aplicación supervisora, el cual es observado periódicamente por esta aplicación, con el propósito de

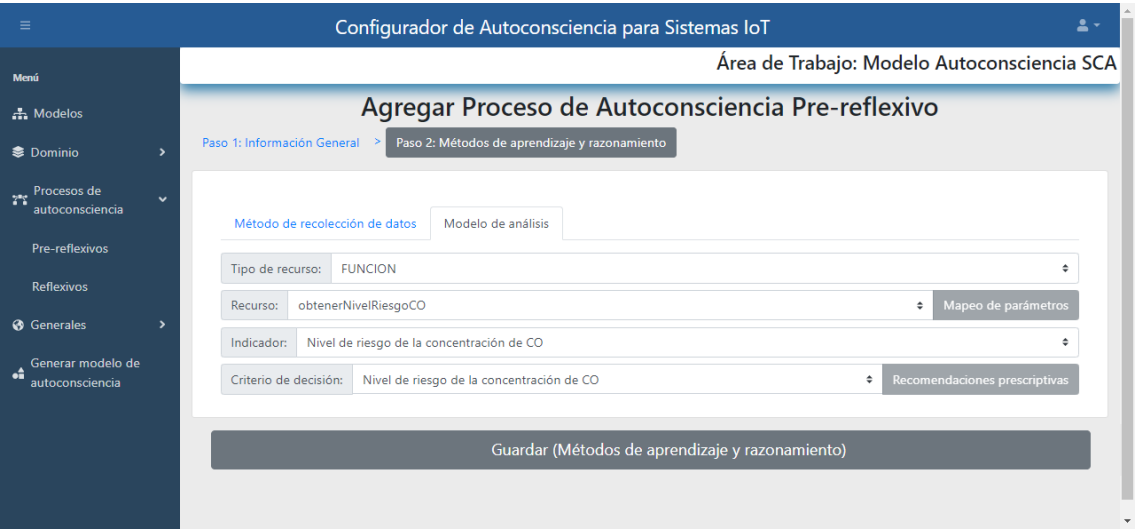


Figura 6-37. Configuración del modelo de análisis del proceso pre-reflexivo sobre la concentración y nivel de riesgo promedio diario de CO en el ambiente de la cocina.
Fuente: Elaboración propia.

Recomendaciones Prescriptivas (Acciones)

✕

Umbrales

Id	Nombre	Interpretación	Inferior	Superior
1	Ausencia de riesgo	El usuario no sufre ningún síntoma ante una exposición prolongada	0.0	3.5
2	Riesgo medio	El usuario puede experimentar dolor de cabeza, pecho, fatiga o náuseas	3.5	20.0
3	Riesgo alto	El usuario puede experimentar desorientación, pérdida del conocimiento e incluso la muerte	20.0	

Agregar

Modificar

Eliminar

#	Id	Descripción	Activo
☐	4	El subsistema SCA debe encender una alerta auditiva permanente	☒
☐	5	El subsistema SCA debe notificar a la central de monitoreo	☒

Cerrar

Figura 6-38. Configuración de las acciones o recomendaciones prescriptivas para el modelo de análisis del proceso pre-reflexivo sobre la concentración y nivel de riesgo promedio diario de CO en el ambiente de la cocina. Fuente: Elaboración propia.

detectar cualquier cambio en las especificaciones de los modelos; y, en consecuencia, disparar la ejecución automática de las actividades posteriores de la metodología.

En la Figura 6-39 se visualiza el archivo JSON generado con las especificaciones de los modelos de arquitectura y autoconsciencia del subsistema SCA. Además, este archivo puede ser descargado localmente por el usuario utilizando el botón *Descargar archivo JSON*.



Figura 6-39. Generación del archivo JSON con las especificaciones de arquitectura y autoconciencia del subsistema SCA. Fuente: Elaboración propia.

Actividad 3. Sincronización entre los modelos de arquitectura – autoconciencia y el sistema IoT observado (mantiene la relación causal)

En esta actividad, la aplicación supervisora del middleware que se ejecuta en el nodo *Fog HP Proliant Microserver*, observa periódicamente cualquier cambio en las especificaciones de los modelos de arquitectura y autoconciencia contenidas en el archivo JSON. A su vez, al momento de detectar algún cambio en los modelos, esta aplicación planifica y distribuye las tareas de implementación o reconfiguración de los recursos de software de monitoreo y autoconciencia para que sean ejecutadas por la puerta de enlace *Raspberry PI 3*, el propio nodo Fog y, el nodo Cloud de CEDIA, con el propósito de mantener la relación causal entre los modelos y el subsistema SCA. Por su parte, la ejecución de estas tareas se realiza con el apoyo de la aplicación implementadora del middleware, instalada en cada uno de los nodos de computación del escenario de IoT.

Particularmente, la Figura 6-40 muestra la consola del nodo Fog con un extracto de los logs resultantes de las tareas ejecutadas por la aplicación supervisora. En esta Figura se observa por orden de ocurrencia los siguientes eventos: i) detección automática de una nueva versión del archivo JSON con los modelos de arquitectura y autoconciencia; ii) identificación de los nodos de computación especificados en la nueva versión del modelo de arquitectura del escenario de IoT (*HP Proliant Microserver*, *Raspberry PI 3*, *nodo Cloud de CEDIA*); iii) comparación de la nueva versión de los modelos con la configuración en tiempo de ejecución del subsistema SCA; iii) planificación de las tareas de implementación y/o reconfiguración de los recursos de monitoreo y autoconciencia a ser ejecutadas en cada uno de los nodos de computación identificados; iv) suscripción del nodo supervisor a los tópicos del broker mediante los cuales se receptan los reportes individuales de cumplimiento de los planes ejecutados en cada uno de los nodos de computación; v) publicación en el broker de la nueva versión de los modelos, así como de los planes de implementación y reconfiguración de los recursos de software, con el objeto de que puedan ser receptados e implementados por cada uno de los nodos de computación; v) recepción y revisión de los reportes individuales de cumplimiento de los planes, obteniéndose como resultado la implementación satisfactoria de los recursos de monitoreo y autoconciencia en todos los nodos de computación; y, vi) finalmente, notificación por medio del broker sobre el cumplimiento exitoso de los planes a nivel

general del subsistema SCA, por lo que se establece la nueva versión de los modelos como la configuración en ejecución de este subsistema.

```

C:\Windows\System32\cmd.exe - node startup.js --modooperacion 2 --rutamodelos ./models
C:\Infraestructura\middleware\AplicacionSupervisora>node startup.js --modooperacion 2 --rutamodelos ./models
Aplicación supervisora inicializada
Modo operación: En conjunto los motores de sincronización de recursos de monitoreo y autoconsciencia
Observando nueva versión de los modelos de arquitectura y autoconsciencia ...
Se ha detectado una nueva versión del archivo JSON: models\ModelosArquitecturaAutoconsciencia.json
Identificando los nodos de computación (Edge, Fog y Cloud) especificados en el nuevo modelo de arquitectura de monitoreo ...
RaspberryPI3 - IP: 192.168.1.10
HPProliantMicroserver - IP: 192.168.1.11
CEDIACloud - IP: 172.16.1.10
MOTOR DE SINCRONIZACIÓN DE RECURSOS DE MONITOREO
Comparando la nueva versión del modelo de arquitectura de monitoreo con la versión actual en ejecución ...
Planificando tareas de implementación/reconfiguración de recursos de monitoreo para los nodos de computación identificados ...
MOTOR DE SINCRONIZACIÓN DE RECURSOS DE AUTOCONSCIENCIA
Comparando la nueva versión del modelo de autoconsciencia con la versión actual en ejecución ...
Planificando tareas de implementación/reconfiguración de recursos de autoconsciencia para los nodos de computación identificados ...
Suscribiendo el nodo supervisor a los tópicos del broker ...
Tópico: Reporte individual de cumplimiento de planes en RaspberryPI3
Tópico: Reporte individual de cumplimiento de planes en HPProliantMicroserver
Tópico: Reporte individual de cumplimiento de planes en CEDIACloud
Publicando en el broker, la nueva versión de los modelos y los planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia ...
Receptando (por medio del broker) y analizando los reportes individuales de cumplimiento de los planes de implementación/reconfiguración de recursos ...
Implementación exitosa de los recursos de monitoreo y/o autoconsciencia en RaspberryPI3 (OK)
Implementación exitosa de los recursos de monitoreo y/o autoconsciencia en HPProliantMicroserver (OK)
Implementación exitosa de los recursos de monitoreo y/o autoconsciencia en CEDIACloud (OK)
Publicando en el broker, un mensaje sobre el estado de cumplimiento exitoso de los planes a nivel general del sistema IoT ...
Se ha establecido la nueva versión de los modelos como la configuración en ejecución del sistema IoT
Aplicación supervisora observando nueva versión de modelos de arquitectura y autoconsciencia ...
  
```

Figura 6-40. Extracto de reporte de logs de las tareas ejecutadas por la aplicación supervisora en el nodo Fog. Fuente: Elaboración propia.

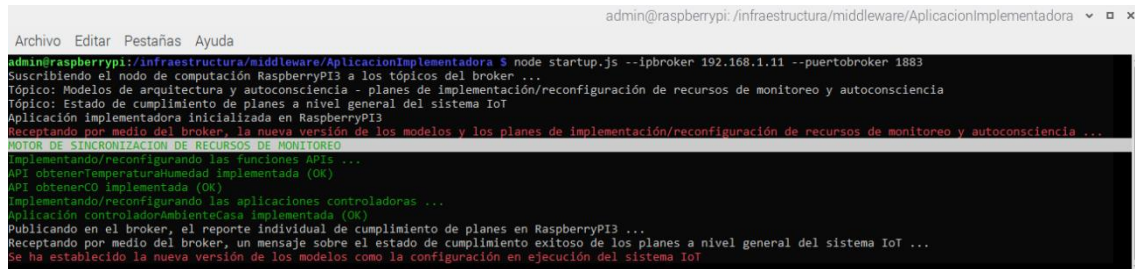
A su vez, las Figuras de la 6-41 a la 6-43 presentan un extracto de los logs resultantes de las tareas ejecutadas por la aplicación implementadora en cada uno de los nodos de computación del escenario de IoT. Como eventos iniciales se tienen: i) suscripción de cada nodo de computación a los tópicos del broker por medio de los cuales se reciben la nueva versión de los modelos y los planes de implementación y reconfiguración de los recursos de software; ii) para el caso particular del nodo Fog, se inicializa el servidor de aplicaciones Node.js Express en el puerto 9999, conforme a lo especificado en el modelo de arquitectura, con el propósito de soportar la publicación de los servicios web relacionados con los procesos de monitoreo y autoconsciencia; y, iii) concretamente, la detección y recepción de una nueva versión de los modelos y de los correspondientes planes de implementación y reconfiguración de los recursos de software.

A continuación, en la consola de la puerta de enlace IoT (Raspberry PI 3) se visualizan los eventos relacionados con la implementación satisfactoria de las APIs *obtenerTemperaturaHumedad* y *obtenerCO*; y, la aplicación *controladorAmbienteCasa*.

Mientras que, en la consola del nodo Fog se observan los siguientes eventos. En primer lugar, aquellos relacionados con la creación de la base de datos *PostgreSQLLocal* y las tablas utilizadas por los procesos tanto de monitoreo como de autoconsciencia (*AmbienteDormitorio*, *AmbienteCocina*, *CargaCPUFog*, *MetricaAutoconsciencia*). En segundo lugar, por orden de ocurrencia, los eventos relacionados con la implementación de los recursos de monitoreo: i) tópico de broker *datosAmbienteCocina*; ii) API *obtenerCargaCPU*; iii) servicios web *guardarCargaCPU*, *guardarTemperaturaHumedad*, *guardarCO* y *guardarResumenDiarioCO*; y, iv) aplicaciones *controladorCargaCPU* y *controladorResumenDiarioCO*. En tercer lugar, asimismo por orden de ocurrencia, los eventos concernientes a la implementación de los recursos de autoconsciencia: i) funciones *obtenerPromedio*, *obtenerDesviacionEstandar*, *obtenerIndicadorMetricaPorcentual*, *obtenerNivelSatisfaccionTemperatura*, *obtenerNivelAceptabilidadTemperatura* y *obtenerNivelRiesgoCO*; ii) servicio web *obtenerRegresionLinealSimple*; y, iii) demonios - funciones asíncronas (*PID1* al *PID6*) que encapsulan los procesos de autoconsciencia.

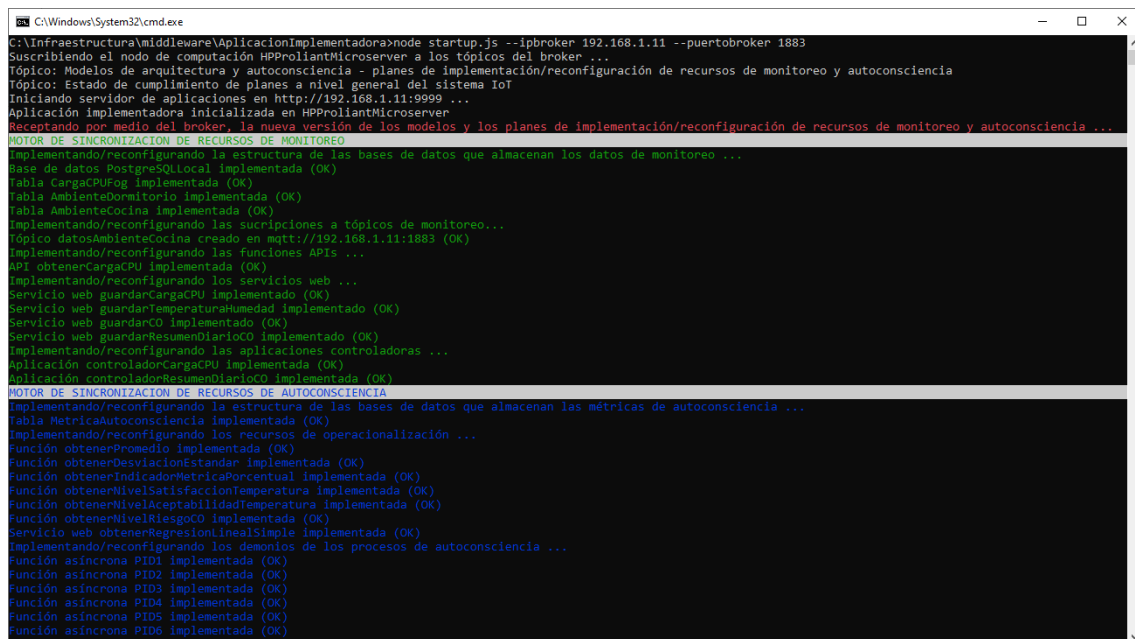
Luego, en la consola del nodo Cloud se presentan los eventos relacionados con la creación de la base de datos *PostgreSQLCloud* y la tabla *ResumenDiarioCO* empleada para almacenar los datos resultantes del flujo de agregación.

Como eventos finales, comunes entre todos los nodos, se tienen: i) publicación en el broker de los reportes individuales de cumplimiento de los planes; ii) recepción del mensaje publicado por el nodo supervisor sobre el cumplimiento exitoso de los planes a nivel general del subsistema SCA; y, iii) finalmente, establecimiento de la nueva versión de los modelos como la configuración en ejecución del subsistema SCA.



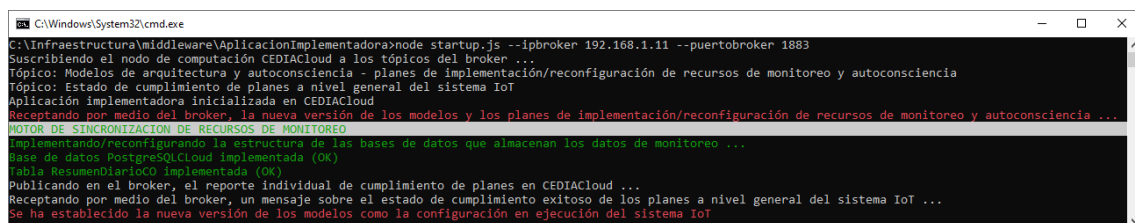
```
admin@raspberrypi:~/infraestructura/middleware/AplicacionImplementadora $ node startup.js --ipbroker 192.168.1.11 --puertobroker 1883
Suscribiendo el nodo de computación RaspberryPi3 a los tópicos del broker ...
Tópico: Modelos de arquitectura y autoconsciencia - planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia
Tópico: Estado de cumplimiento de planes a nivel general del sistema IoT
Aplicación implementadora inicializada en RaspberryPi3
Receptando por medio del broker, la nueva versión de los modelos y los planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia ...
MOTOR DE SINCRONIZACION DE RECURSOS DE MONITOREO
Implementando/reconfigurando las funciones APIs ...
API obtenerTemperaturaHumedad implementada (OK)
API obtenerCO implementada (OK)
Implementando/reconfigurando las aplicaciones controladoras ...
Aplicación controladorAmbienteCasa implementada (OK)
Publicando en el broker, el reporte individual de cumplimiento de planes en RaspberryPi3 ...
Receptando por medio del broker, un mensaje sobre el estado de cumplimiento exitoso de los planes a nivel general del sistema IoT ...
Se ha establecido la nueva versión de los modelos como la configuración en ejecución del sistema IoT
```

Figura 6-41. Extracto de reporte de logs de las tareas ejecutadas por la aplicación implementadora en la puerta de enlace IoT (Raspberry PI 3). Fuente: Elaboración propia.



```
C:\Windows\System32\cmd.exe
C:\Infraestructura\middleware\AplicacionImplementadora>node startup.js --ipbroker 192.168.1.11 --puertobroker 1883
Suscribiendo el nodo de computación HPProliantMicroserver a los tópicos del broker ...
Tópico: Modelos de arquitectura y autoconsciencia - planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia
Tópico: Estado de cumplimiento de planes a nivel general del sistema IoT
Iniciando servicio de aplicaciones en http://192.168.1.11:9999 ...
Aplicación implementadora inicializada en HPProliantMicroserver
Receptando por medio del broker, la nueva versión de los modelos y los planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia ...
MOTOR DE SINCRONIZACION DE RECURSOS DE MONITOREO
Implementando/reconfigurando la estructura de las bases de datos que almacenan los datos de monitoreo ...
Base de datos PostgreSQLLocal implementada (OK)
Tabla CargaCPUfog implementada (OK)
Tabla AmbienteDormitorio implementada (OK)
Tabla AmbienteCocina implementada (OK)
Implementando/reconfigurando las suscripciones a tópicos de monitoreo...
Tópico datosAmbienteCocina creado en mqtt://192.168.1.11:1883 (OK)
Implementando/reconfigurando las funciones APIs ...
API obtenerCargaCPU implementada (OK)
Implementando/reconfigurando los servicios web ...
Servicio web guardarCargaCPU implementado (OK)
Servicio web guardarTemperaturaHumedad implementado (OK)
Servicio web guardarCO implementado (OK)
Servicio web guardarResumenDiarioCO implementado (OK)
Implementando/reconfigurando las aplicaciones controladoras ...
Aplicación controladorCargaCPU implementada (OK)
Aplicación controladorResumenDiarioCO implementada (OK)
MOTOR DE SINCRONIZACION DE RECURSOS DE AUTOCONSCIENCIA
Implementando/reconfigurando la estructura de las bases de datos que almacenan las métricas de autoconsciencia ...
Tabla MetricaAutoconsciencia implementada (OK)
Implementando/reconfigurando los recursos de operacionalización ...
Función obtenerPromedio implementada (OK)
Función obtenerDesviacionEstandar implementada (OK)
Función obtenerIndicadorMetricaPorcentual implementada (OK)
Función obtenerNivelSatisfaccionTemperatura implementada (OK)
Función obtenerNivelAceptabilidadTemperatura implementada (OK)
Función obtenerNivelRiesgoCO implementada (OK)
Servicio web obtenerRegresionLinealSimple implementada (OK)
Implementando/reconfigurando los demonios de los procesos de autoconsciencia ...
Función asíncrona PID1 implementada (OK)
Función asíncrona PID2 implementada (OK)
Función asíncrona PID3 implementada (OK)
Función asíncrona PID4 implementada (OK)
Función asíncrona PID5 implementada (OK)
Función asíncrona PID6 implementada (OK)
```

Figura 6-42. Extracto de reporte de logs de las tareas ejecutadas por la aplicación implementadora en el nodo Fog. Fuente: Elaboración propia.



```
C:\Windows\System32\cmd.exe
C:\Infraestructura\middleware\AplicacionImplementadora>node startup.js --ipbroker 192.168.1.11 --puertobroker 1883
Suscribiendo el nodo de computación CEDIACloud a los tópicos del broker ...
Tópico: Modelos de arquitectura y autoconsciencia - planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia
Tópico: Estado de cumplimiento de planes a nivel general del sistema IoT
Aplicación implementadora inicializada en CEDIACloud
Receptando por medio del broker, la nueva versión de los modelos y los planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia ...
MOTOR DE SINCRONIZACION DE RECURSOS DE MONITOREO
Implementando/reconfigurando la estructura de las bases de datos que almacenan los datos de monitoreo ...
Base de datos PostgreSQLCloud implementada (OK)
Tabla ResumenDiarioCO implementada (OK)
Publicando en el broker, el reporte individual de cumplimiento de planes en CEDIACloud ...
Receptando por medio del broker, un mensaje sobre el estado de cumplimiento exitoso de los planes a nivel general del sistema IoT ...
Se ha establecido la nueva versión de los modelos como la configuración en ejecución del sistema IoT
```

Figura 6-43. Extracto de reporte de logs de las tareas ejecutadas por la aplicación implementadora en el nodo Cloud. Fuente: Elaboración propia.

Actividad 4. Monitoreo

Una vez completada la actividad anterior, los recursos de software de monitoreo implementados entran en funcionamiento para operacionalizar los flujos de recolección y agregación de datos relacionados con las propiedades: temperatura y humedad de la habitación del adulto mayor, monóxido de carbono de la cocina; y, carga de la CPU del

nodo Fog. Estos recursos de monitoreo operan según las condicionantes establecidas en el escenario de IoT y reflejadas en el modelo de arquitectura de monitoreo en tiempo de ejecución.

En los flujos de recolección de datos, las aplicaciones instanciadas *controladorAmbienteCasa* y *controladorCargaCPU* disparan y controlan periódicamente los flujos de datos entre las APIs, servicios web y broker, dependiendo del tipo de comunicación síncrona o asíncrona. Por su parte, las APIs *obtenerTemperaturaHumedad*, *obtenerCO* y *obtenerCargaCPU* obtienen los datos crudos directamente desde los sensores físicos (*DHT11*, *MQ-7*) o virtuales (*carga de CPU*) y devuelven estos datos a las aplicaciones que las invocan. A su vez, el broker *Mosquitto* almacena temporalmente los tópicos con los datos de monitoreo publicados por las aplicaciones. Finalmente, los servicios web publicados *guardarTemperaturaHumedad*, *guardarCO* y *guardarCargaCPU* reciben los datos de monitoreo directamente desde las aplicaciones (comunicación síncrona) o por intermedio de un evento que se dispara ante la presencia de nuevos datos publicados en el broker (comunicación asíncrona). Estos servicios tienen como finalidad almacenar los datos de monitoreo en tablas correspondientes (*AmbienteDormitorio*, *AmbienteCocina*, *CargaCPUFog*) de la base de datos *PostgreSQLLocal*, ubicada en el nodo Fog.

En el flujo de agregación de datos, la aplicación *controladorResumenDiarioCO* consume periódicamente (una vez al día – 23h55) el servicio web *guardarResumenDiarioCO*. En primer lugar, este servicio obtiene desde tabla *AmbienteCocina* de la base de datos *PostgreSQLLocal*, los datos de CO monitoreados durante el día; luego, calcula el valor promedio, máximo y mínimo diario de estos datos; y, finalmente, almacena los datos resumidos en la tabla *ResumenDiarioCO* de la base de datos *PostgreSQLCloud*, ubicada en el nodo Cloud.

La Figura 6-44 presenta una muestra de los datos crudos recolectados periódicamente por el subsistema SCA (artefactos de monitoreo), relacionados con la temperatura, humedad, CO y carga de la CPU del nodo Fog, mismos que se encuentran almacenados en la base de datos *PostgreSQLLocal* del nodo Fog. Además, en esta Figura se observan los metadatos (*IdSensor*, *idCPU*, *nroSensoresControlados*, *fechaHora*) asociados a los datos recolectados.

Actividad 5. Aprendizaje

Asimismo, una vez completada la Actividad 3 de la metodología, los recursos de software de autoconsciencia implementados inician su operación, conforme a las especificaciones establecidas en los modelos en tiempo de ejecución. Durante esta actividad de *Aprendizaje*, los seis demonios (funciones asíncronas) instanciados tienen como alcance producir las métricas directas e indirectas relacionadas con los aspectos de autoconsciencia a evaluar. En particular, tres demonios ejecutan los procesos pre-reflexivos *PID1*, *PID2*, y *PID3* que producen las métricas directas: *grados Celsius temperatura ambiente actual del dormitorio del adulto mayor*, *partes por millón de concentración actual de CO en la cocina* y *porcentaje de carga de procesamiento actual de la CPU del nodo Fog*, respectivamente. Mientras que, los restantes demonios ejecutan los procesos reflexivos *PID4*, *PID5* y *PID6* que producen las métricas indirectas: *grados Celsius de variabilidad diaria de la temperatura ambiente del dormitorio del adulto mayor*, *partes por millón de concentración promedio diario de CO en la cocina* y *porcentaje estimado de carga de procesamiento de la CPU del nodo Fog*, respectivamente.

public.ambientedormitorio/postgreSQLLocal/postgres@NodoFog				
	temperatura real	humedad real	idsensor integer	fechahora timestamp without time zone
1	15.48	41.36	17	2023-06-27 07:13:14.353
2	16.15	41.51	17	2023-06-27 08:13:14.349
3	18.37	42.48	17	2023-06-27 09:13:14.362
4	19.01	41.82	17	2023-06-27 10:13:14.418
5	20.93	40.63	17	2023-06-27 11:13:14.479
6	22.75	48.13	17	2023-06-27 12:13:14.515
7	24.62	54.98	17	2023-06-27 13:13:14.169
8	23.85	63.88	17	2023-06-27 14:13:14.179
9	22.73	63.94	17	2023-06-27 15:13:14.212
10	22.24	54.56	17	2023-06-27 16:13:14.278
11	20.45	49.39	17	2023-06-27 17:13:14.332
12	19.12	47.69	17	2023-06-27 18:13:14.375
13	18.35	45.21	17	2023-06-27 19:13:14.429
14	17.41	41.75	17	2023-06-27 20:13:14.483
15	17.09	39.1	17	2023-06-27 21:13:14.332

public.ambientecocina/postgreSQLLocal/postgres@NodoFog			
	co real	idsensor integer	fechahora timestamp without time zone
1	7.22	12	2023-06-27 07:03:14.355
2	5.4	12	2023-06-27 08:03:14.352
3	7.49	12	2023-06-27 09:03:14.364
4	5.04	12	2023-06-27 10:03:14.419
5	5.45	12	2023-06-27 11:03:14.481
6	5.05	12	2023-06-27 12:03:14.516
7	7.63	12	2023-06-27 13:03:14.172
8	5.94	12	2023-06-27 14:03:14.182
9	6.61	12	2023-06-27 15:03:14.213
10	5.67	12	2023-06-27 16:03:14.28
11	5.04	12	2023-06-27 17:03:14.334
12	6.64	12	2023-06-27 18:03:14.377
13	5.75	12	2023-06-27 19:03:14.432
14	6.28	12	2023-06-27 20:03:14.485
15	5.62	12	2023-06-27 21:03:14.338

public.cargacpufog/postgreSQLLocal/postgres@NodoFog				
	cargacpu real	idcpu integer	nrosensorescontrolados integer	fechahora timestamp without time zone
1	42	1	2	2023-06-27 07:03:14.350
2	39	1	2	2023-06-27 08:03:14.347
3	37	1	2	2023-06-27 09:03:14.360
4	21	1	1	2023-06-27 10:03:14.417
5	17	1	1	2023-06-27 11:03:14.476
6	9	1	0	2023-06-27 12:03:14.511
7	23	1	1	2023-06-27 13:03:14.166
8	17	1	1	2023-06-27 14:03:14.176
9	41	1	2	2023-06-27 15:03:14.210
10	38	1	2	2023-06-27 16:03:14.275
11	37	1	2	2023-06-27 17:03:14.328
12	15	1	1	2023-06-27 18:03:14.371
13	7	1	0	2023-06-27 19:03:14.425
14	21	1	1	2023-06-27 20:03:14.482
15	43	1	2	2023-06-27 21:03:14.328

Figura 6-44. Una muestra de los datos crudos recolectados y almacenados en la base de datos PostgreSQLLocal del nodo Fog. Fuente: Elaboración propia.

Los demonios de los procesos pre-reflexivos utilizan los datos crudos o empíricos recopilados y almacenados en la actividad anterior de *Monitoreo* con el objeto de generar las métricas directas. Mientras que, los demonios de los procesos reflexivos invocan a las respectivas funciones implementadas (*obtenerPromedio*, *obtenerDesviacionEstandar*) o al servicio web publicado (*obtenerRegresionLinealSimple*) para operacionalizar los métodos de cálculo, a fin de generar las métricas indirectas, mediante la aplicación de operaciones estadísticas sobre los datos históricos de las métricas directas previamente obtenidas.

Para ilustrar la ejecución de los procesos de autoconsciencia, en las Figuras 6-45 y 6-46 se presentan capturas de la consola del nodo Fog con información sobre las métricas directas e indirectas producidas por los demonios de los procesos pre-reflexivos y reflexivos, respectivamente.

Actividad 6. Razonamiento

Como complemento a la actividad anterior, en este punto los demonios de los procesos de autoconsciencia, a partir de las métricas directas e indirectas obtenidas, generan los indicadores cuantitativos y cualitativos que valoran el estado y la evolución de los objetos y aspectos de autoconsciencia del escenario de IoT. Específicamente, los demonios de los procesos pre-reflexivos *PID1*, *PID2*, y *PID3* producen los indicadores cuantitativos: *nivel de satisfacción del adulto mayor sobre la temperatura ambiente del dormitorio*, *nivel de*

```

C:\Windows\System32\cmd.exe
-----PROCESO DE AUTOCONSCIENCIA-----
PID: 1 Tipo: Pre-reflexivo - Estado Fecha: 28/6/2023 20:43:25.294
Descripción: Captura el conocimiento sobre la carga de procesamiento de la CPU del nodo fog en tiempo real.
Métrica directa: Porcentaje de carga de procesamiento actual de la CPU. Valor: 38.00%
Indicador: Nivel de carga de procesamiento de la CPU. Valor: 38.00%
Umbral: Normal
Interpretación: Normal carga de procesamiento de la CPU.
Recomendación/acción: Ninguna.

-----PROCESO DE AUTOCONSCIENCIA-----
PID: 1 Tipo: Pre-reflexivo - Estado Fecha: 28/6/2023 20:43:25.304
Descripción: Captura el conocimiento sobre la temperatura ambiente actual (en tiempo real) del dormitorio y el nivel de satisfacción del usuario.
Métrica directa: Grados Celsius temperatura ambiente actual. Valor: 17.59°C
Indicador: Nivel de satisfacción del usuario sobre la temperatura ambiente. Valor: 0.00%
Umbral: Insatisfactorio
Interpretación: Temperatura ambiente no confortable para el usuario.
Recomendación/acción:
- El subsistema SCA debe calibrar el aire acondicionado de la habitación (auto-adaptación).

-----PROCESO DE AUTOCONSCIENCIA-----
PID: 2 Tipo: Pre-reflexivo - Estado Fecha: 28/6/2023 20:43:25.310
Descripción: Captura el conocimiento sobre la concentración y nivel de riesgo actual (en tiempo real) de CO en el ambiente de la cocina.
Métrica directa: Partes por millón de concentración actual de CO. Valor: 5.13PPM
Indicador: Nivel de riesgo de la concentración de CO. Valor: 0.51%
Umbral: Ausencia de riesgo
Interpretación: El usuario no sufre ningún síntoma ante una exposición prolongada.
Recomendación/acción: Ninguna.

-----PROCESO DE AUTOCONSCIENCIA-----
PID: 3 Tipo: Pre-reflexivo - Estado Fecha: 28/6/2023 20:53:25.313
Descripción: Captura el conocimiento sobre la carga de procesamiento de la CPU del nodo fog en tiempo real.
Métrica directa: Porcentaje de carga de procesamiento actual de la CPU. Valor: 17.60%
Indicador: Nivel de carga de procesamiento de la CPU. Valor: 17.60%
Umbral: Baja
Interpretación: Baja carga de procesamiento de la CPU.
Recomendación/acción:
- El subsistema SCA debe verificar la disponibilidad o frecuencia de lectura de los sensores (auto-recuperación).
- El subsistema SCA debe verificar la correcta operación de la red de comunicación (auto-recuperación).
- El subsistema SCA debe notificar a la central de monitoreo.

-----PROCESO DE AUTOCONSCIENCIA-----
PID: 1 Tipo: Pre-reflexivo - Estado Fecha: 28/6/2023 20:53:25.324
Descripción: Captura el conocimiento sobre la temperatura ambiente actual (en tiempo real) del dormitorio y el nivel de satisfacción del usuario.
Métrica directa: Grados Celsius temperatura ambiente actual. Valor: 17.49°C

```

Figura 6-45. Logs de consola con información de las métricas directas producidas por los procesos de pre-reflexivos durante un periodo de tiempo. Fuente: Elaboración propia.

```

C:\Windows\System32\cmd.exe
-----PROCESO DE AUTOCONSCIENCIA-----
PID: 4 Tipo: Reflexivo - Estado Fecha: 28/6/2023 23:55:0.167
Descripción: Captura el conocimiento sobre la variabilidad diaria y nivel de aceptabilidad de la temperatura ambiente del dormitorio.
Métrica indirecta: Grados Celsius de variabilidad diaria de la temperatura ambiente. Valor: 4.13°C
Indicador: Nivel de aceptabilidad de la variabilidad diaria de la temperatura ambiente. Valor: 0.00%
Umbral: Inaceptable.
Interpretación: Inaceptable nivel de variabilidad de la temperatura ambiente.
Recomendación/acción:
- El subsistema SCA debe notificar a la central de monitoreo.

-----PROCESO DE AUTOCONSCIENCIA-----
PID: 5 Tipo: Reflexivo - Estado Fecha: 28/6/2023 23:55:0.274
Descripción: Captura el conocimiento sobre la concentración y nivel de riesgo promedio diario del monóxido de carbono (CO) en el ambiente de la cocina.
Métrica indirecta: Partes por millón de concentración promedio diario de CO. Valor: 5.83PPM
Indicador: Nivel de riesgo de la concentración de CO. Valor: 0.58%
Umbral: Ausencia de riesgo.
Interpretación: El usuario no sufre ningún síntoma ante una exposición prolongada.
Recomendación/acción: Ninguna.

-----PROCESO DE AUTOCONSCIENCIA-----
PID: 6 Tipo: Reflexivo - Predicción Fecha: 28/6/2023 23:55:0.280
Descripción: Predice la carga de procesamiento requerida por la CPU del nodo Fog para controlar el doble de sensores que operan actualmente.
Escenario de simulación: Duplicar la cantidad de sensores que operan actualmente.
Variable de simulación (nroSensoresSimular): Valor: 4
Métrica indirecta: Porcentaje estimado de carga de procesamiento de la CPU. Valor: 71.34%
Indicador: Nivel de carga de procesamiento de la CPU. Valor: 71.34%
Umbral: Normal.
Interpretación: Normal carga de procesamiento de la CPU.
Recomendación/acción: Ninguna.

```

Figura 6-46. Logs de consola con información de las métricas indirectas producidas por los procesos de reflexivos durante un periodo de tiempo. Fuente: Elaboración propia.

riesgo de la concentración actual de CO en la cocina y nivel de carga de procesamiento actual de la CPU del nodo Fog, respectivamente. Mientras que, los demonios de los procesos reflexivos PID4, PID5 y PID6, producen los indicadores cuantitativos: nivel de aceptabilidad de la variabilidad diaria de la temperatura ambiente del dormitorio del adulto mayor, nivel de riesgo de la concentración promedio diario de CO en la cocina y nivel de carga estimada de procesamiento de la CPU del nodo Fog, respectivamente.

Las funciones implementadas (*obtenerIndicadorMétricaPorcentual*, *obtenerNivelSatisfaccionTemperatura*, *obtenerNivelAceptabilidadTemperatura*, *obtenerNivelRiesgoCO*) son invocadas desde los demonios de los procesos pre-reflexivos y reflexivos para operacionalizar los modelos de análisis y producir los indicadores cuantitativos. A continuación, como parte de los modelos de análisis, los demonios comparan los indicadores cuantitativos resultantes con los criterios de decisión y sus umbrales, a fin de obtener los indicadores cualitativos. Estos indicadores describen la situación actual o estiman la posible evolución futura de los aspectos de autoconsciencia analizados por el subsistema SCA y le permiten determinar acciones a ejecutar (recomendaciones prescriptivas), de ser necesario. En las Figuras 6-45 y 6-46 citadas con anterioridad, se presentan adicionalmente los indicadores (cuantitativos y cualitativos) y las

recomendaciones generadas por los demonios de los procesos de autoconsciencia, durante un periodo de tiempo.

Cabe recalcar que los demonios de los procesos de autoconsciencia almacenan las métricas (directas, indirectas e indicadores) generadas tanto en esta actividad como la anterior, en la tabla *MetricaAutoconsciencia* de la base de datos *PostgreSQLLocal*, ubicada en el nodo Fog.

Finalmente, a modo de ejemplo en las Figuras 6-47 y 6-48, se presentan en el visor web de autoconsciencia los valores obtenidos para el indicador *nivel de carga de procesamiento de la CPU* por parte de los procesos pre-reflexivo PID3 y reflexivo PID4, respectivamente. Adicionalmente, se puede observar las correspondientes interpretaciones y recomendaciones para estos valores, así como el escenario y valor de simulación asociado, en caso de existir.

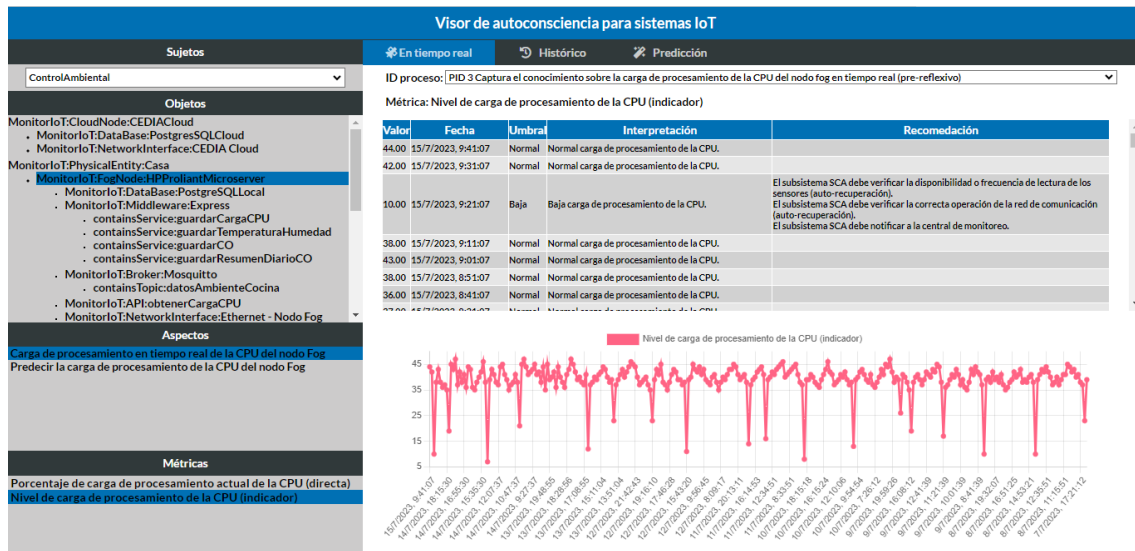


Figura 6-47. Visualización de datos generados por el proceso pre-reflexivo PID3 para el indicador nivel de carga de procesamiento de la CPU del nodo Fog en tiempo real. Fuente: Elaboración propia

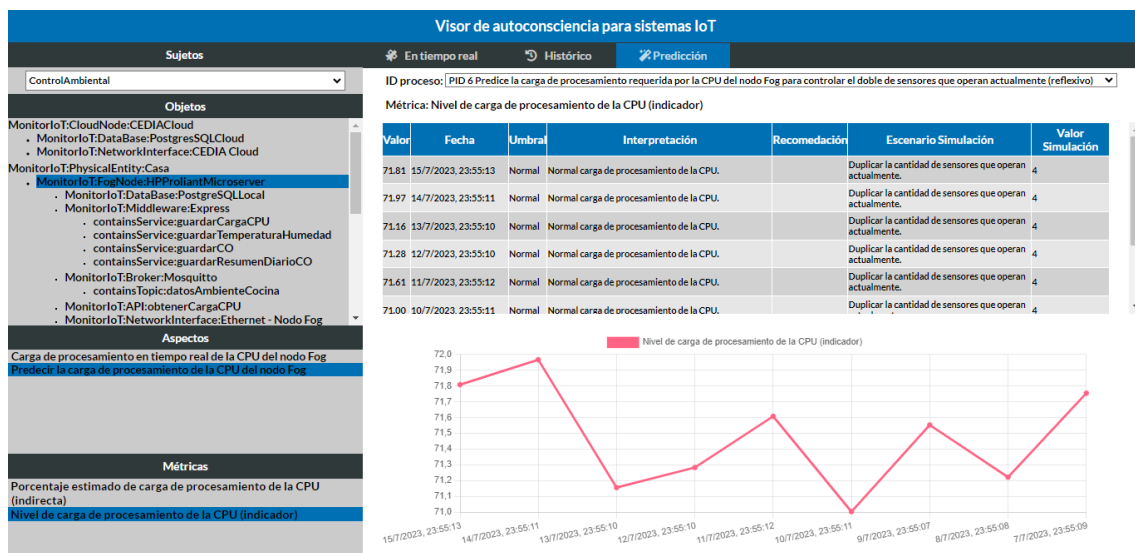


Figura 6-48. Visualización de datos generados por el proceso reflexivo PID6 para el indicador nivel de carga estimada de procesamiento de la CPU del nodo Fog. Fuente: Elaboración propia.

6.3.2 Implementación de los cambios de requeridos por el estudio de caso

Una vez que se ha puesto en operación el subsistema SCA con todos los requerimientos de monitoreo y autoconsciencia establecidos inicialmente en el estudio de caso, en esta subsección, se demuestra cómo se gestionan los cambios en este subsistema sin detener su operación, con el apoyo de la metodología *Aware-IoT@Run.Time*. De igual manera, se requiere ejecutar de forma ordenada todas las actividades de la metodología, pero en el contexto de un sistema implementado y en funcionamiento. En primer lugar, empleando las herramientas *DSL Monitor-IoT* y configurador de autoconsciencia, se actualizan las especificaciones de los modelos de arquitectura de monitoreo y capacidades de autoconsciencia de conformidad con los cambios solicitados en el estudio de caso. Luego, se genera el archivo JSON con la nueva versión de los modelos en tiempo de ejecución. Consecuentemente, el middleware de la infraestructura automáticamente detecta los cambios en estos modelos; y, planifica y ejecuta las tareas reconfiguración y puesta en operación de la nueva versión de los artefactos de software de monitoreo y autoconsciencia en los nodos de computación que lo requieran.

En este estudio de caso, las tareas de reconfiguración afectan a la puerta de enlace (*Raspberry PI 3*) y nodo Fog (*HP Proliant Microserver*). En las Figuras 6-49 y 6-50 se presentan los logs generados por la aplicación implementadora que corre en estos nodos. En las Figuras, de igual manera, se puede apreciar como evento inicial la recepción de la nueva versión de los modelos en tiempo de ejecución y de los planes de implementación y reconfiguración de los artefactos de software, de acuerdo con los cambios propuestos en el caso de estudio.

Por un lado, en la consola de la puerta de enlace se muestra como tarea ejecutada, la reconfiguración (modificación de su código fuente) de la aplicación *controladorAmbienteCasa*, de tal forma que los datos de temperatura y humedad obtenidos desde el sensor *DHT11* se publiquen temporalmente en el tópico *DatosAmbienteDormitorio* del broker *Mosquitto* (comunicación asíncrona), en lugar de enviarlos directamente a través de un servicio web (comunicación síncrona) al nodo Fog para su almacenamiento. Además, como esta aplicación controla el flujo de recolección de datos de la temperatura y humedad del dormitorio, también se reconfigura para que este flujo se ejecute automáticamente cada 30 minutos.

Por otro lado, en la consola del nodo Fog se observan las siguientes tareas ejecutadas: i) creación o publicación del nuevo tópico *DatosAmbienteDormitorio* en el broker *Mosquitto*; ii) reconfiguración de la función *obtenerNivelSatisfaccionTemperatura* que operacionaliza el modelo de análisis del proceso *PIDI - Captura el conocimiento sobre la temperatura ambiente actual (en tiempo real) del dormitorio y el nivel de satisfacción del usuario*, a fin de que transforme la métrica directa en el correspondiente indicador, según el nuevo intervalo requerido para el umbral *Satisfactorio* (entre 15°C y 25°C); y, iii) reconfiguración de la función asíncrona asociada al proceso *PIDI*, de tal manera que su frecuencia de ejecución sea también cada 30 minutos.

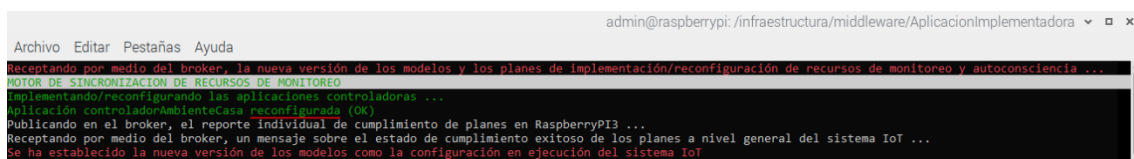


Figura 6-49. Extracto de reporte de logs de las tareas de reconfiguración ejecutadas por la aplicación implementadora en la puerta de enlace IoT (Raspberry PI 3).

Fuente: Elaboración propia.

```

C:\Windows\System32\cmd.exe
Receptando por medio del broker, la nueva versión de los modelos y los planes de implementación/reconfiguración de recursos de monitoreo y autoconsciencia ...
MOTOR DE SINCRONIZACIÓN DE RECURSOS DE MONITOREO
Implementando/reconfigurando las suscripciones a tópicos de monitoreo ...
Tópico datosAmbienteDormitorio creado en mqtt://192.168.1.11:1883 (OK)
MOTOR DE SINCRONIZACIÓN DE RECURSOS DE AUTOCONSCIENCIA
Implementando/reconfigurando los recursos de operacionalización ...
Función obtenerNivelSatisfacciónTemperatura reconfigurada (OK)
Implementando/reconfigurando los demonios de los procesos de autoconsciencia ...
Función asincrónica PID1 reconfigurada (OK)
Publicando en el broker, el reporte individual de cumplimiento de planes en HPProliantMicroserver ...
Receptando por medio del broker, un mensaje sobre el estado de cumplimiento exitoso de los planes a nivel general del sistema IoT ...
Se ha establecido la nueva versión de los modelos como la configuración en ejecución del sistema IoT
  
```

Figura 6-50. Extracto de reporte de logs de las tareas de reconfiguración ejecutadas por la aplicación implementadora en el nodo Fog. Fuente: Elaboración propia.

6.4 Conclusiones

Con la implementación del estudio de caso se ha demostrado que utilizando la solución metodológica Aware-IoT@Run.Time y su infraestructura de soporte es factible construir, operar y mantener sistemas IoT autoconscientes con un alto grado de abstracción, expresividad y flexibilidad. Entre las principales fortalezas de esta solución, observadas y constatadas durante la ejecución del estudio de caso se tienen:

- El proceso metodológico propuesto por Aware-IoT@Run.Time, fundamentado en seis actividades principales, ha orientado de manera formal y disciplinada al equipo de desarrollo en la implementación de los requisitos de monitoreo y autoconsciencia del estudio de caso. El disponer de una guía metodológica, que utiliza en su etapa inicial, plantillas para el levantamiento y organización de las especificaciones de los componentes de la arquitectura de un sistema IoT, facilitó notablemente el entendimiento de la problemática inmersa en el estudio de caso y agilizó la ejecución de las actividades posteriores de modelado de la arquitectura de monitoreo y configuración de las capacidades de autoconsciencia del sistema IoT.
- Las herramientas DSL Monitor-IoT y configurador de autoconsciencia han posibilitado que el equipo de desarrollo se concentre principalmente en la descripción de especificaciones funcionales de monitoreo y autoconsciencia mediante lenguajes visuales y formularios basados en conceptos abstractos de dominio, minimizando los detalles de implementación propios de la infraestructura.
- En conjunto, los metamodelos Monitor-IoT y Aware-IoT han proporcionado un lenguaje lo suficientemente completo para especificar e implementar los artefactos de software que automatizan los procesos de monitoreo y autoconsciencia (pre-reflexivos y reflexivos) de un sistema IoT.
- El middleware de la infraestructura ha posibilitado la implementación y despliegue automático y distribuido de los artefactos de software de monitoreo y autoconsciencia del estudio de caso, a nivel de las capas Edge, Fog y Cloud. Para ello, esta herramienta utilizó como único insumo las especificaciones contenidas en los modelos en tiempo de ejecución, prescindiendo de la intervención del equipo de desarrollo en las tareas de codificación de estos artefactos. Adicionalmente, el código fuente (lógica de monitorización y autoconsciencia) encapsulado en los artefactos generados por el middleware ha sido implementado siguiendo los lineamientos de modularidad especificados en los modelos en tiempo de ejecución, a fin de asegurar la reutilización de estos artefactos (alta cohesión y bajo acoplamiento), así como facilitar su mantenibilidad.
- La solución tecnológica ha permitido que el equipo de desarrollo incorpore directamente los cambios requeridos en el estudio de caso, a nivel de los modelos en tiempo de ejecución. Consecuentemente, la relación de causalidad existente entre los modelos y el sistema IoT observado, operacionalizada por medio del middleware de la infraestructura, ha viabilizado de forma automática, ágil y transparente la evolución

del sistema IoT, conforme a las especificaciones de los modelos; y, sin requerir la paralización de su operación y la participación de los desarrolladores.

Las fortalezas antes citadas han repercutido positivamente en ciertos aspectos deseables en el desarrollo de sistemas IoT, tales como: aumento de la automatización del proceso de desarrollo, disminución de la complejidad del desarrollo, mejora de la gestión del cambio y mantenibilidad; y, reducción del esfuerzo y tiempo requerido para la puesta en operación. En términos generales, esto se ha traducido en la mejora de la eficiencia y efectividad del equipo de desarrollo, así como en su percepción favorable sobre la usabilidad y utilidad de la solución Aware-IoT@Run.Time, lo cual incluso es corroborado empíricamente en el Capítulo 7 mediante la ejecución de dos cuasi experimentos. Sin embargo, tras la implementación del estudio de caso se pueden avizorar varias recomendaciones para ser consideradas en trabajos futuros:

- Aunque en el estudio de caso se han incluido diferentes tipos de entidades IoT (físicas, digitales), sensores (físicos, virtuales), flujos de datos (síncrono, asíncrono), procesos de autoconsciencia (pre-reflexivos, reflexivos), aspectos de autoconsciencia (estado, predicción), métricas; y, recursos de implementación (funciones, servicios web), como se ha mencionado con anterioridad, la infraestructura de soporte aún se encuentra en estado de prototipado; por lo tanto, conforme se complete la implementación de todas sus funcionalidades y componentes se deberá ampliar el estudio de caso para incorporar otros requerimientos, tales como: aspectos de autoconsciencia colectivos, recursos de implementación de tipo fórmula, sensores utilizando otras tecnologías o protocolos de comunicación (p. ej., Wifi, Bluetooth).
- Este estudio de caso ha tenido como alcance la implementación de una relación causal entre los modelos en tiempo de ejecución y el sistema IoT observado, fundamentada en el aprendizaje de modelos mediante la actualización y calibración de sus parámetros (métricas, metadatos, etc.) a partir de los datos crudos monitoreados sobre el sistema IoT y su entorno. Un desafío a futuro que permitirá alcanzar mayores ventajas del enfoque propuesto es plantear estudios de caso, donde nuevas entidades IoT, propiedades, metadatos, flujos de datos, procesos de autoconsciencia, métricas, entre otros, puedan ser descubiertos, incorporados o actualizados automáticamente en los modelos en tiempo de ejecución. Previo a ello, será necesario desarrollar algunos componentes adicionales como parte del middleware de la infraestructura, los cuales brinden capacidades de automodelado a los sistemas IoT para cambiar la estructura de sus modelos, como respuesta al entorno cambiante en el que operan estos sistemas.

Como conclusión final, la aplicación de la solución Aware-IoT@Run.Time en el estudio de caso, indica que el uso de enfoques basados en modelos en tiempo de ejecución es ampliamente beneficioso para construir sistemas IoT autoconscientes. Además, ha dejado en evidencia una serie de oportunidades de investigación que se pueden abordar a futuro para perfeccionar la solución tecnológica propuesta.

Capítulo 7

Evaluación Empírica de la Solución

Este capítulo presenta la evaluación empírica de la metodología Aware-IoT@Run.Time y las herramientas de su infraestructura de soporte. Esta evaluación está basada en cuasi experimentos que tienen como propósito conocer la efectividad y eficiencia de este enfoque metodológico, así como la percepción del usuario sobre la facilidad de uso, utilidad e intención de uso futuro de la solución propuesta para la construcción, mantenimiento y operación de sistemas IoT autoconscientes.

La sección 7.1 describe desde una perspectiva general, el alcance y la forma como se evalúa empíricamente la metodología Aware-IoT@Run.Time y sus herramientas. La sección 7.2 presenta una breve descripción del Modelo de Evaluación de Métodos (Method Evaluation Model - MEM) [49], empleado durante esta evaluación. Por su parte, las secciones 7.3 y 7.4 describen los cuasi experimentos aplicados tanto al DSL Monitor-IoT como al configurador web de autoconsciencia y middleware de la infraestructura, respectivamente. La sección 7.5 explica las principales amenazas a la confiabilidad de los resultados de la evaluación empírica de la solución y cómo estas fueron mitigadas durante el desarrollo de los cuasi experimentos. Finalmente, en la sección 7.6 se exponen las conclusiones del capítulo.

7.1 Ámbito de la evaluación empírica de la solución

Un estudio empírico sirve para evaluar un proceso, método, herramienta o cualquier otro tipo de tecnología creado para el desarrollo, mantenimiento o aseguramiento de la calidad del software; y, consiste en un acto u operación que permite descubrir algo que no se conoce o poner a prueba una hipótesis [201]. De ahí que, en este trabajo de investigación se realizó una evaluación empírica de la metodología Aware-IoT@Run.Time y su infraestructura de soporte, conducente a comprobar el rendimiento del usuario al utilizar la solución propuesta, así como su percepción sobre la utilidad y posible adopción de la misma en proyectos futuros de construcción de sistemas IoT autoconscientes.

Particularmente, la evaluación empírica se enfocó en validar las herramientas DSL Monitor-IoT [56], configurador web de capacidades de autoconsciencia y middleware de la infraestructura, este último, conformado por los motores de sincronización de recursos de software de monitoreo y autoconsciencia, de tal forma que se examinen todas las actividades de la metodología Aware-IoT@Run.Time.

Como estrategia de evaluación empírica, se empleó el cuasi experimento basado en el proceso experimental propuesto por Wohlin et al. [46]. Un cuasi experimento es una investigación empírica, similar a un experimento, que mide el efecto de manipular una variable (denominada independiente o factor) en otra variable (denominada variable dependiente), habiendo aleatorizado las posibles variables perturbadoras [47]. Sin embargo, en un cuasi experimento los tratamientos no pueden asignarse de manera aleatoria a los sujetos; por el contrario, la asignación emerge de las características propias de los sujetos u objetos [48]. Por lo tanto, en esta investigación, los cuasi experimentos fueron realizados con la participación de estudiantes de los últimos semestres de las carreras de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca, quienes ejecutaron las actividades de la metodología Aware-IoT@Run.Time con el apoyo de sus herramientas de soporte, con el fin de diseñar, implementar y/o poner en

operación los requerimientos de monitoreo del Subsistema de Salud (SS) descritos en el Anexo D, así como varios de los requerimientos funcionales de monitoreo y autoconsciencia del Subsistema de Control Ambiental (SCA) descritos en el estudio de caso, presentado en el capítulo anterior. Cabe mencionar que según varios estudios [202][203][204] no existe una diferencia significativa entre el tipo de estudiantes que intervinieron en los cuasi experimentos y los profesionales juniors, siendo la capacidad para ejecutar las actividades de los cuasi experimentos plenamente comparables entre estos.

Si bien la metodología Aware-IoT@Run.Time y su infraestructura de soporte ha sido sujeta a varias pruebas (unitarias, integración) y cuasi experimentos (incluidas pruebas pilotos) a lo largo de este trabajo de investigación, conduciendo a una mejora continua de la solución; particularmente, este capítulo se centra en la descripción de dos cuasi experimentos, los cuales abarcan todas las actividades y herramientas de Aware-IoT@Run.Time. Un primer cuasi experimento orientado a la evaluación del DSL Monitor-IoT (*actividad 1 de la metodología*) [56], cuyo ejercicio experimental contempló el diseño de la arquitectura de monitoreo del subsistema SS (ver Fig. 7-2); y, un segundo cuasi experimento enmarcado en la evaluación del configurador web de capacidades de autoconsciencia y el middleware de la infraestructura (*actividades de la 2 a la 6 de la metodología*), cuyo ejercicio experimental consistió en configurar e implementar dos procesos de autoconsciencia en el subsistema SCA, uno pre-reflexivo y otro reflexivo que evalúen la concentración actual y promedio diario de CO en el ambiente de la cocina, respectivamente.

Cabe mencionar que previo a la implementación del segundo cuasi experimento se diseñó y ejecutó una prueba piloto orientada a evaluar únicamente el motor de sincronización de recursos de software de monitoreo. Para ello, se utilizó una plataforma basada en máquinas virtuales que simuló un escenario distribuido de nodos de computación Edge, Fog y Cloud, conteniendo los componentes de software de la infraestructura de soporte; sin embargo, debido al gran tamaño de las máquinas virtuales, la instanciación de los nodos de computación y sus componentes tomó demasiado tiempo durante la prueba piloto. Como alternativa, para que sea factible desplegar de forma rápida el escenario de evaluación del segundo cuasi experimento en cada uno de los computadores de los participantes, se optó por configurar, implementar e integrar todos los componentes de la infraestructura de soporte bajo la plataforma Docker; esto permitió que los componentes puedan ejecutarse dentro de contenedores de software más livianos, facilitando la portabilidad y despliegue de la infraestructura durante la evaluación empírica. En la subsección 7.4.1 se explica cómo los componentes de la infraestructura, requeridos por el escenario de evaluación, se organizaron e implementaron en contenedores de Docker.

Adicionalmente, para la formulación de los cuasi experimentos se utilizó el Modelo de Evaluación de Métodos (MEM) propuesto por Moody [49], ya que integra variables de rendimiento y percepción del usuario como mecanismo para predecir la utilidad y posible adopción de la metodología Aware-IoT@Run.Time y su infraestructura en prácticas futuras. La estructura de este modelo se describe en la siguiente sección.

7.2 Modelo de Evaluación de Métodos (Method Evaluation Model - MEM)

El MEM fue originalmente concebido como un medio para evaluar el rendimiento actual, la aceptación y la posible adopción de métodos de diseño de sistemas de información [49]; sin embargo, este modelo ha sido exitosamente aplicado para la evaluación de otros tipos de métodos en el campo de la ingeniería de software [205].

El MEM es un modelo teórico que incorpora dos aspectos de “éxito” para un método, la eficacia actual y la adopción en la práctica [49][206]. En consecuencia, este modelo combina el Pragmatismo Metodológico, una teoría para validar el conocimiento metodológico [207] y el Modelo de Aceptación de Tecnológica (Technology Acceptance Model - TAM) propuesto por Davis [208], un modelo teórico para analizar y explicar la percepción del usuario sobre facilidad de uso, utilidad e intención de uso de una tecnología o método determinado.

La Figura 7-1 resume los constructores del MEM y las relaciones causales entre ellos. En esta Figura se observa que la adopción de un método en la práctica no solamente depende de su eficacia actual (éxito pragmático), sino además de si el usuario de un método lo percibe como eficaz (éxito percibido) [205].

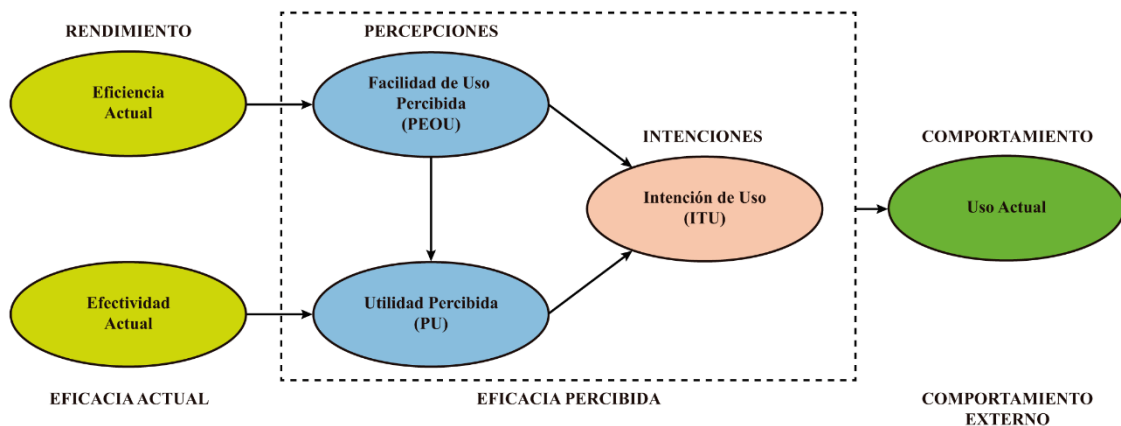


Figura 7-1. Modelo de Evaluación de Métodos (Method Evaluation Model – MEM).

Fuente: Moody (2003) [49].

A continuación, se exponen las definiciones de cada uno de los constructores del MEM [49]:

- **Eficacia Actual.** Está conformado por dos variables de desempeño o rendimiento del usuario.
 - **Eficiencia Actual.** Esfuerzo requerido para aplicar un método, el cual puede ser medido como el tiempo requerido para aplicar un método.
 - **Efectividad Actual.** Grado con el que un método alcanza sus objetivos, mismo que puede ser medido evaluando la calidad de los artefactos obtenidos al aplicar un método.
- **Eficacia Percibida.** Está basada en dos variables de percepción del usuario.
 - **Facilidad de Uso Percibida.** Grado en que una persona cree que utilizando un método requerirá menos esfuerzo para desempeñar sus tareas. Esta variable representa un juicio perceptivo sobre el esfuerzo requerido para aprender y usar un método determinado.
 - **Utilidad Percibida.** Grado en que una persona cree que utilizando un método mejorará su desempeño en el trabajo. Esta variable representa un juicio perceptivo sobre la efectividad de un método. El modelo contiene una relación causal, la cual indica que la utilidad percibida puede estar condicionada por la facilidad de uso percibida.
 - **Intención de Uso.** Grado en que una persona tiene la intención de usar un método en particular. Esta variable representa un juicio perceptivo sobre la eficacia y efectividad del método y es utilizada para medir la probabilidad de adopción del

mismo en la práctica. El modelo incluye dos relaciones causales, las cuales indican que la facilidad de uso percibida y la utilidad percibida pueden afectar la intención de usar un método.

- **Uso Actual.** Representa una variable basada en el comportamiento, definida como el grado en que un método es utilizado en la práctica. El modelo contempla una relación causal hipotética, donde el uso actual está condicionado por la intención de uso.

En el modelo MEM, la predicción de la variable intención de uso está orientada a métodos propuestos recientemente, mientras que, la medición de la variable uso actual (adopción en la práctica) solo puede ser utilizada en métodos que ya han sido adoptados por la industria [205].

Finalmente, para utilizar el MEM en la evaluación de la metodología Aware-IoT@Run.Time y su infraestructura de soporte fue necesario operacionalizar o adaptar este modelo teórico para su aplicación en este tipo específico de métodos y herramientas. En las siguientes secciones se explica cómo se operacionalizó el MEM en cada uno de los cuasi experimentos ejecutados.

7.3 Cuasi experimento Nro. 1: Evaluación del DSL Monitor-IoT

Este cuasi experimento se enfoca en la evaluación de la primera actividad de la metodología Aware-IoT@Run.Time, relacionada con el diseño visual de la arquitectura de monitoreo de un sistema IoT con el apoyo de la herramienta DSL Monitor-IoT [56]. Por lo tanto, en esta sección se describe el diseño, ejecución y análisis de los resultados de este experimento, a fin de proporcionar evidencias sobre el rendimiento (eficiencia y efectividad) del usuario, así como su percepción sobre la facilidad de uso, utilidad e intención de uso futuro del DSL Monitor-IoT y el proceso metodológico en el que se basa.

7.3.1 Diseño del cuasi experimento

El cuasi experimento fue diseñado con base en el proceso experimental propuesto por Wohlin et al. [46]. Como primer paso se formuló el *objetivo del cuasi experimento* de acuerdo con el paradigma Goal-Question Metric (GQM) propuesto por Basili et al. [209]. En la Tabla 7-1 se presenta el objetivo del cuasi experimento.

Tabla 7-1. Goal-Question-Metric para el cuasi experimento Nro. 1.

<i>Evaluar:</i>	El DSL Monitor-IoT en el diseño de arquitecturas de monitoreo para sistemas IoT.
<i>Con el propósito de:</i>	• Determinar el rendimiento (efectividad y eficiencia) del usuario en el uso del DSL Monitor-IoT y su proceso metodológico. • Comprender la percepción del usuario sobre la facilidad de uso, utilidad e intención de uso futuro del DSL Monitor-IoT y su proceso metodológico.
<i>Desde el punto de vista de:</i>	Investigador.
<i>En el contexto de:</i>	Un grupo de estudiantes de los últimos semestres de la carrera de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca.

A su vez, se definieron las *preguntas de investigación* a responder por medio del cuasi experimento:

- RQ1.** ¿El DSL Monitor-IoT es percibido como fácil de usar y útil? De ser así ¿La percepción del usuario es el resultado de su rendimiento al utilizar el DSL Monitor-IoT?

RQ2. ¿Existe la intención de usar el DSL Monitor-IoT en el futuro? De ser así ¿La intención de uso es el resultado de la percepción del usuario?

Por su parte, las preguntas de investigación fueron evaluadas a través de pruebas de hipótesis. En particular, la primera pregunta fue estudiada mediante las hipótesis:

- $H1_0$: El DSL Monitor-IoT se percibe como difícil de usar, $H1_0 = \neg H1_1$.
- $H2_0$: El DSL Monitor-IoT no se percibe como útil, $H2_0 = \neg H2_1$.
- $H4_0$: La facilidad de uso percibida no está determinada por la eficiencia actual, $H4_0 = \neg H4_1$.
- $H5_0$: La utilidad percibida no está determinada por la efectividad actual. $H5_0 = \neg H5_1$.
- $H6_0$: La utilidad percibida no está determinada por la facilidad de uso percibida, $H6_0 = \neg H6_1$.

Mientras que, la segunda pregunta fue analizada por medio de las hipótesis:

- $H3_0$: No existe la intención de utilizar el DSL Monitor-IoT en el futuro, $H3_0 = \neg H3_1$.
- $H7_0$: La intención de uso no está determinada por la facilidad de uso percibida, $H7_0 = \neg H7_1$.
- $H8_0$: La intención de uso no está determinada por la utilidad percibida, $H8_0 = \neg H8_1$.

El *contexto del cuasi experimento* estuvo determinado por:

- i) La herramienta DSL Monitor-IoT y su proceso metodológico.
- ii) El escenario de IoT utilizado para la evaluación de Monitor-IoT, el cual se enmarca dentro del subdominio de ambientes de vida asistida y consistió en el diseño de la arquitectura de monitoreo del subsistema SS, cuyos requerimientos se describen en el Anexo D. Por su parte, en la Figura 7-2 se muestra el modelo de arquitectura de monitoreo resultante del subsistema SS, el cual debe ser diseñado por los participantes como parte del ejercicio experimental.
- iii) La participación de 33 estudiantes de los últimos semestres de la carrera de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca, con sólidos conocimientos sobre Arquitecturas de Software, Desarrollo Dirigido por Modelos e Internet de las Cosas.

Las *tareas experimentales del cuasi experimento* fueron definidas conforme al proceso propuesto para diseñar arquitecturas de monitoreo para sistemas IoT con el DSL Monitor-IoT, mismo que abarca la Actividad 1 de la metodología Aware-IoT@Run.Time (ver Capítulo 4). En este sentido, las tareas principales del cuasi experimento fueron:

- **Tarea 1.** Identificación de subsistemas, entidades, propiedades y flujos de datos.
- **Tarea 2.** Modelado de subsistemas, entidades y propiedades.
- **Tarea 3.** Modelado de flujos de datos.

Las *variables dependientes del cuasi experimento* para evaluar Monitor-IoT fueron las variables de rendimiento (eficiencia y efectividad) y percepción de usuario (facilidad de uso, utilidad e intención de uso) definidas en el MEM. Estas variables permitieron probar las hipótesis y son medidas mediante las ecuaciones y el cuestionario presentados en la subsección 7.3.2.

El *material experimental del cuasi experimento* incluyó la documentación y las herramientas de software necesarias para realizar y evaluar las tareas experimentales. Este material fue distribuido a los participantes mediante un sitio web y estuvo conformado por:

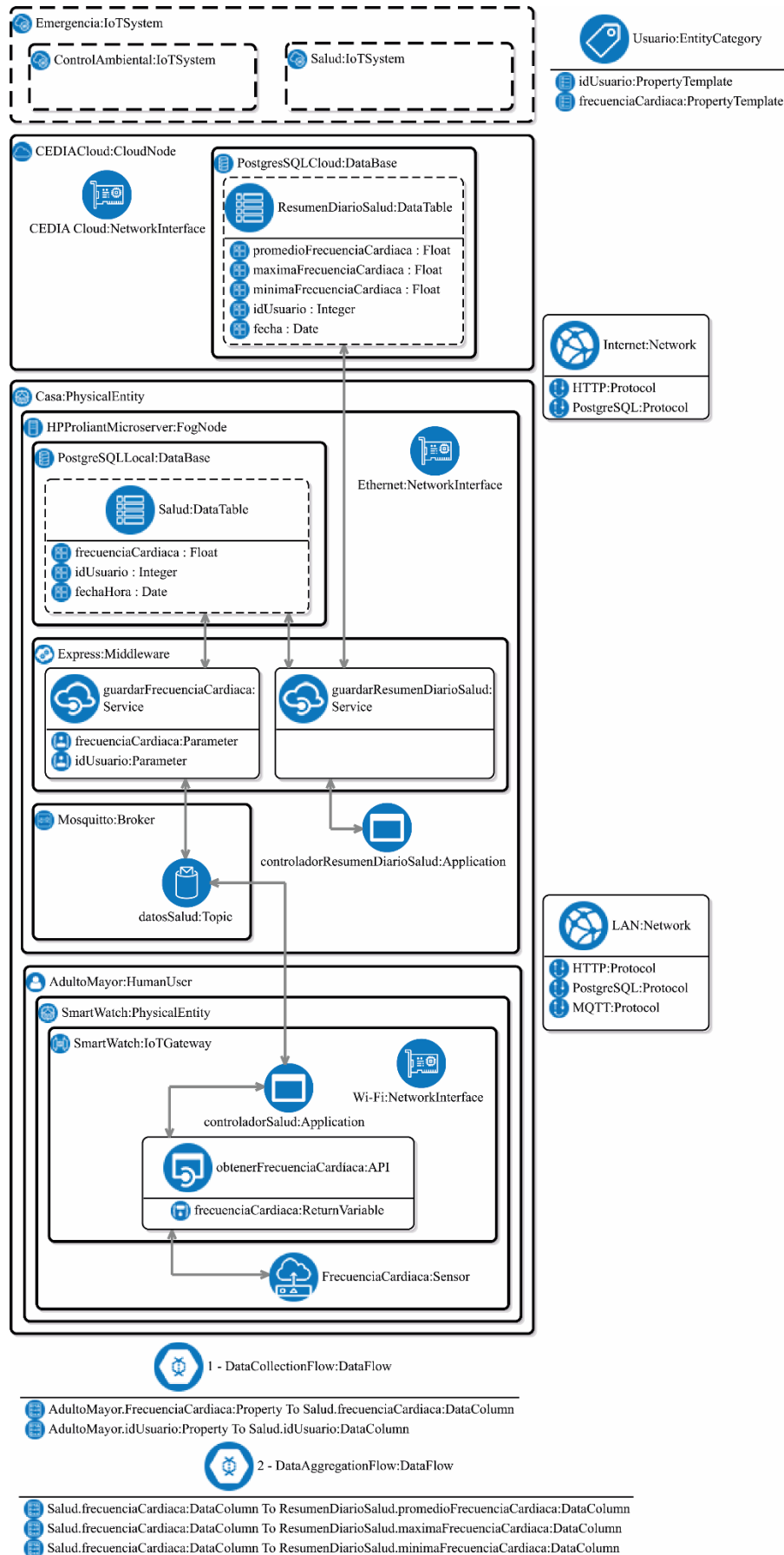


Figura 7-2. Modelo de la arquitectura de monitoreo del subsistema SS resultante de la ejecución de las tareas experimentales. Fuente: Elaboración propia.

- i) Una presentación utilizada por el investigador para capacitar a los participantes. En esta presentación se describen: los principales conceptos relacionados con el DSL Monitor-IoT, el software base (Obeo Designer Community Edition) utilizado por Monitor-IoT, el entorno de trabajo (notación) del diseñador gráfico de Monitor-IoT; y, un ejercicio de entrenamiento que contempla el diseño de la arquitectura de monitoreo del subsistema SCA, detallado en el estudio de caso propuesto en el Capítulo 6.
- ii) Un video tutorial con la descripción de los pasos para la instalación y configuración de la herramienta Monitor-IoT.
- iii) Un video tutorial sobre el ejercicio de entrenamiento.
- iv) Una guía metodológica de Aware-IoT@Run.Time con la descripción de las tareas y sub tareas que contemplan la Actividad 1, relacionada con el diseño de arquitecturas de monitoreo para sistemas IoT, utilizando Monitor-IoT (ver Capítulo 4).
- v) Un boletín y hoja de trabajo con la descripción e instrucciones del ejercicio experimental (diseño de la arquitectura de monitoreo del subsistema SS). Esta hoja de trabajo incluye varios campos para recolectar métricas sobre las actividades ejecutadas por los participantes y los productos obtenidos (ver Anexo D).
- vi) Un cuestionario en Google Forms para medir la percepción del usuario sobre el uso de la solución propuesta, una vez que haya concluido el experimento.

7.3.2 Adaptación del MEM para la evaluación del DSL Monitor-IoT

Con el propósito de utilizar el MEM en la evaluación del DSL Monitor-IoT se requiere operacionalizar este modelo teórico para su aplicación en este tipo específico de herramientas. La operacionalización consiste en definir los constructores del modelo en función de variables de rendimiento y percepción relevantes para Monitor-IoT; y, a su vez, adaptar los instrumentos de medición para evaluar estas variables bajo el contexto de Monitor-IoT.

Con respecto a las variables de rendimiento, la *Eficiencia Actual* contempla la medición del esfuerzo requerido para diseñar la arquitectura de monitoreo de un sistema IoT con el apoyo de Monitor-IoT y su proceso metodológico; mientras que, la *Efectividad Actual* comprende la medición de la calidad del modelo de arquitectura de monitoreo resultante de la utilización de Monitor-IoT. A continuación, se presenta la forma de evaluar cada una de las variables de rendimiento citadas:

- **Eficiencia Actual.** Tiempo empleado para completar todas las tareas experimentales del cuasi experimento. Para ello, se utiliza la Ecuación 7-1:

$$Eficiencia = \sum_{i=1}^n tiempo\ tarea_i \quad (7-1)$$

En donde, n es el número de tareas experimentales.

- **Efectividad Actual.** Grado de cumplimiento de las tareas experimentales del cuasi experimento. Para lo cual, se utiliza la Ecuación 7-2:

$$Efectividad = \frac{\sum_{i=1}^n puntaje\ tarea_i}{n} \quad (7-2)$$

En donde, n es el número de tareas experimentales, mientras que, el puntaje de cada tarea se asigna de la siguiente manera:

- 1, cuando la tarea fue realizada correctamente.
- 0.5, cuando la tarea fue realizada parcialmente.
- 0, cuando la tarea fue realizada incorrectamente.

Con relación a la medición de las variables de percepción del MEM, en la Tabla 7-2 se presenta un cuestionario adaptado, el cual está conformado por doce preguntas, distribuidas de la siguiente manera: i) cinco preguntas para medir la *Facilidad de Uso Percibida (PEOU)*; ii) cuatro preguntas para evaluar la *Utilidad Percibida (PU)*; iii) dos preguntas enfocadas en la *Intención de Uso (ITU)*; y, iv) una pregunta abierta para recopilar sugerencias sobre el DSL Monitor-IoT. A su vez, las preguntas fueron formuladas utilizando la escala de Likert (5 puntos), asumiendo 3 como valor neutro, a partir del cual se puede aceptar o rechazar la herramienta Monitor-IoT según las variables de percepción.

Tabla 7-2. Cuestionario para medir las variables de percepción del MEM para el cuasi experimento Nro. 1.

Nro.	Pregunta
PEOU1	El diseñador gráfico de Monitor-IoT es intuitivo y fácil de entender.
PEOU2	La notación gráfica (representación de los componentes) utilizada por Monitor-IoT es intuitiva y fácil de entender.
PEOU3	El diseñador gráfico de Monitor-IoT es fácil de usar.
PEOU4	Las tareas y subtarear a seguir para diseñar una arquitectura de monitoreo para sistemas IoT con el apoyo de Monitor-IoT son claras y fáciles de entender.
PEOU5	Las tareas y subtarear a seguir para diseñar una arquitectura de monitoreo para sistemas IoT con el apoyo de Monitor-IoT son fáciles de implementar.
PU1	El uso de Monitor-IoT podría reducir el tiempo y esfuerzo requerido en el desarrollo, despliegue y mantenimiento de sistemas IoT.
PU2	Monitor-IoT es útil para apoyar las actividades de desarrollo, despliegue y mantenimiento de sistemas IoT.
PU3	El uso de Monitor-IoT podría mejorar su desempeño en el desarrollo, despliegue y mantenimiento de sistemas IoT.
PU4	El modelo de arquitectura de monitoreo obtenido desde Monitor-IoT es útil para representar como los requisitos de monitorización de un sistema IoT deben ser implementados.
ITU1	En caso de necesitar diseñar una arquitectura de monitoreo para sistemas IoT en el futuro, utilizaría Monitor-IoT.
ITU2	Recomendaría el uso de Monitor-IoT para diseñar arquitecturas de monitoreo para sistemas IoT.
OQ1	Tiene alguna sugerencia con respecto a la herramienta Monitor-IoT (<i>pregunta abierta</i>).

De acuerdo con la adaptación del MEM, la probabilidad de que el DSL Monitor-IoT sea aceptado en la práctica puede ser predicho probando las hipótesis expuestas en la subsección anterior. Estas hipótesis contienen implícitamente las variables del MEM (Eficiencia, Efectividad, PEOU, PU y ITU). En la Figura 7-3 se presenta el modelo adaptado del MEM para evaluar el rendimiento (eficiencia y efectividad) del usuario, así como su percepción sobre la facilidad de uso, utilidad e intención de uso futuro del DSL Monitor-IoT y el proceso metodológico en el que se basa.

7.3.3 Ejecución del cuasi experimento

El cuasi experimento fue realizado de manera virtual en la plataforma Zoom con la participación de 33 estudiantes de la carrera de Ingeniería de Sistemas, 12 pertenecientes a la Universidad del Azuay y 21 a la Universidad de Cuenca. Además, el cuasi experimento tuvo una duración de cuatro horas, divididas en dos sesiones.

La primera sesión se centró en la instalación de Monitor-IoT, así como la capacitación y entrenamiento de los participantes. Esta sesión tuvo una duración de una hora y media; e, incluyó las siguientes actividades: i) presentación de los principales conceptos relacionados con Monitor-IoT (p. ej., Internet de las Cosas, Arquitectura de Sistemas, Lenguajes

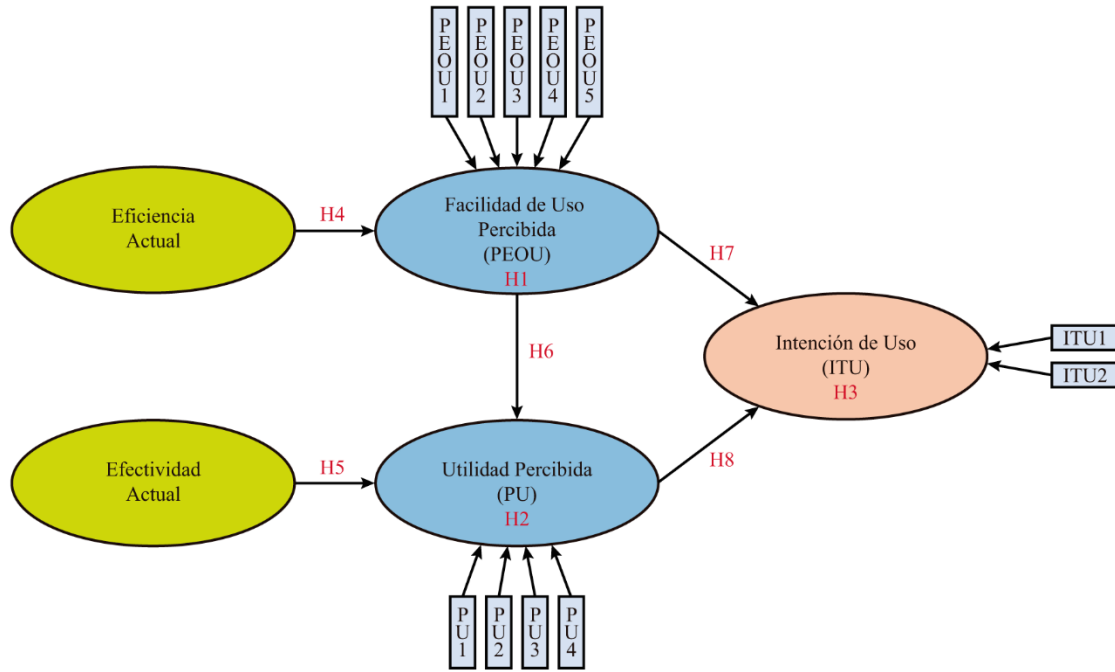


Figura 7-3. Modelo adaptado del MEM para evaluar el DSL Monitor-IoT y su proceso metodológico. Fuente: Elaboración propia.

Específicos de Dominio); ii) explicación y acompañamiento en el proceso de instalación y configuración de Monitor-IoT, iii) revisión del entorno de trabajo (área de diseño, paleta de herramientas, ventana de propiedades) de Monitor-IoT, la forma de instanciar los elementos del DSL, definir sus propiedades e interconectarlos entre sí; y, iv) resolución del ejercicio de entrenamiento con el propósito de demostrar cómo se debe utilizar Monitor-IoT para diseñar una arquitectura de monitoreo para sistemas IoT, según el proceso metodológico de Aware-IoT@Run.Time (ver Capítulo 4).

La segunda sesión se orientó al desarrollo del ejercicio experimental por parte de los participantes. Esta sesión tuvo una duración de dos horas y media, distribuidas de la siguiente manera: i) 20 minutos para presentar el caso práctico (subsistema SS) a resolver con la herramienta Monitor-IoT (ver Anexo D); ii) 120 minutos para diseñar la arquitectura de monitoreo planteada en el ejercicio experimental; sin embargo, se debe recalcar que se permitió a los participantes terminar el experimento fuera del tiempo indicado, con la finalidad de mitigar el efecto techo; y, iii) 10 minutos para responder un cuestionario en Google Forms con las preguntas presentadas en la Tabla 7-2, así como para enviar la hoja de trabajo resuelta del ejercicio experimental y el archivo con el modelo de arquitectura obtenido en formato XMI. El investigador responsable de realizar el experimento aclaró cualquier duda durante el desarrollo del ejercicio experimental.

7.3.4 Análisis de los resultados del cuasi experimento

Los datos recogidos en el experimento fueron analizados utilizando estadística descriptiva, diagramas de caja (box plots) y pruebas con el soporte de la herramienta SPSS V20 (con $\alpha = 0.05$), a fin de contrastar las hipótesis establecidas en el experimento, que como se mencionó con anterioridad incluyen las variables de rendimiento y percepción del MEM (Eficiencia, Efectividad, PEOU, PU y ITU). Para la aceptación o rechazo de las hipótesis se utilizaron los niveles significancia sugeridos por Moody [210], mismos que se enlistan en la Tabla 7-3.

Tabla 7-3. Niveles de significancia de Moody [210].

Nivel	Rango
No significativo	$p > 0.1$
Baja significancia	$p < 0.1$
Media significancia	$p < 0.05$
Alta significancia	$p < 0.01$
Muy alta significancia	$p < 0.001$

Por su parte, en el Anexo F (Tabla F-1) se presentan en forma detallada los datos recogidos y calculados por participante para cada una de las variables de rendimiento y percepción del MEM.

7.3.4.1 *Análisis de la percepción del usuario*

La Figura 7.4 presenta los diagramas de caja de las variables de percepción. Como común denominador se observa que la media de cada variable es mayor al valor neutro (3) de la escala de Likert (ver también Tabla 7-4). A su vez, el diagrama de caja de la variable Intención de Uso (ITU) muestra un único dato atípico perteneciente al participante con identificador 1. Este participante ha sido excluido para evitar distorsiones en los análisis estadísticos subsiguientes.

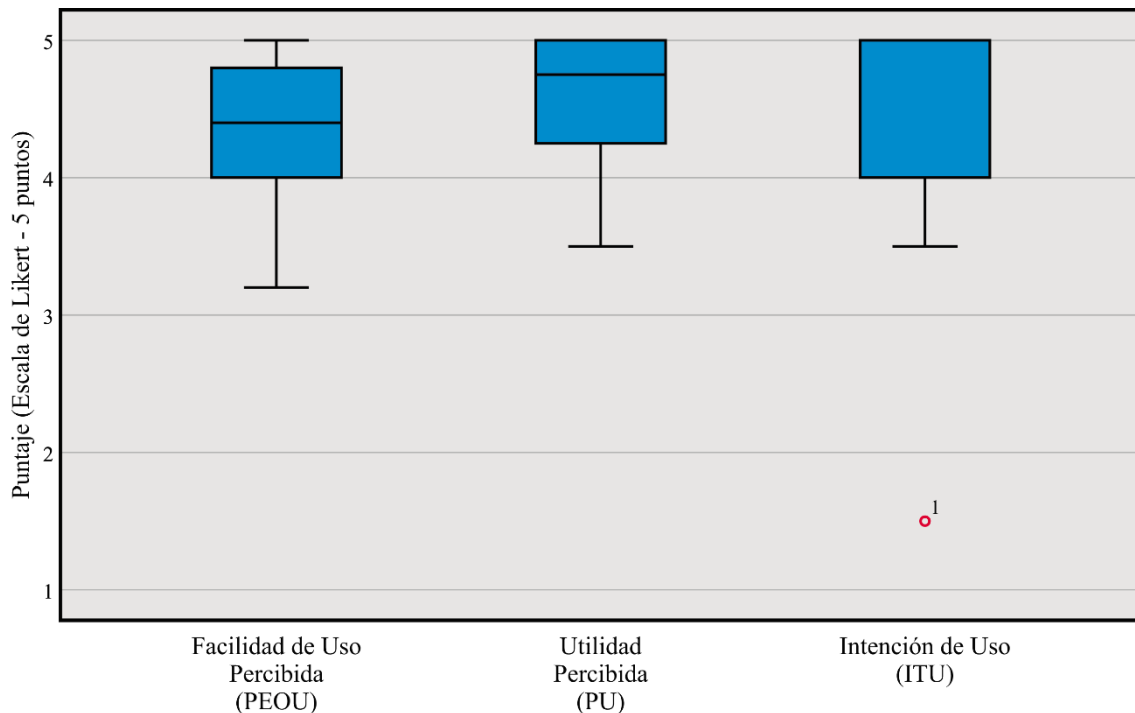


Figura 7-4. Diagramas de caja de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 1. Fuente: Elaboración propia.

Como parte del análisis se aplicó la prueba de Shapiro-Wilk para comprobar si las variables de percepción tienen una distribución normal; y, a su vez, determinar el tipo de prueba a emplear para contrastar las hipótesis nulas $H1_0$, $H2_0$ y $H3_0$. En la penúltima columna de la Tabla 7-4 se muestran los resultados de la prueba de Shapiro-Wilk, donde se puede observar que todas las variables estudiadas (PEOU, PU y ITU) no tienen una distribución normal ($p < 0.05$). Por lo tanto, para contrastar las hipótesis se seleccionó la prueba de Wilcoxon one-tailed one-sample (prueba no paramétrica) con un valor de prueba igual a 3, correspondiente al valor neutral de la escala de Likert.

Los resultados de la prueba de Wilcoxon one-tailed one-sample para cada variable se presentan en la última columna de la Tabla 7-4, donde se puede observar que todas las variables de percepción (PEOU, PU y ITU) tienen un muy alto nivel de significancia ($p < 0.001$) con relación al valor de prueba 3. Por lo tanto, estos resultados permiten rechazar las hipótesis nulas H_{10} , H_{20} y H_{30} , lo cual significa que los participantes perciben al DSL Monitor-IoT como una herramienta fácil de usar y útil; y, además, tienen la intención de utilizar Monitor-IoT para diseñar arquitecturas de monitoreo para sistemas IoT en proyectos futuros.

Tabla 7-4. Estadística descriptiva y pruebas de hipótesis de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 1.

Variable	Mín.	Máx.	Media	Desviación Estándar	Shapiro-Wilk test p-value	Wilcoxon one-tailed one-sample test p-value
PEOU	3.2000	5.0000	4.4250	0.5099	0.013 < 0.05	0.000 < 0.001
PU	3.5000	5.0000	4.5469	0.4854	0.000 < 0.05	0.000 < 0.001
ITU	3.5000	5.0000	4.5625	0.5499	0.000 < 0.05	0.000 < 0.001

7.3.4.2 Análisis del rendimiento del usuario

La Tabla 7-5 presenta las medidas de estadística descriptiva para las variables de rendimiento (Efectividad y Eficiencia). De igual manera, los datos atípicos identificados previamente fueron eliminados en este análisis.

La efectividad promedio de los participantes fue de 86.98%, demostrando que en su gran mayoría fueron capaces de cumplir correctamente con las tareas experimentales relacionadas con el diseño de la arquitectura de monitoreo de IoT propuesta en este cuasi experimento. Concretamente, las tareas experimentales 1, 2 y 3 alcanzaron niveles de cumplimiento del 87.88%, 95.45% y 78.79%, respectivamente (ver Anexo F). Los errores más frecuentes cometidos por los participantes fueron: identificación parcial de las entidades IoT y sus flujos de datos, equivocaciones en la clasificación de las entidades IoT y la configuración incorrecta de las reglas de mapeo de los flujos de datos.

A su vez, la eficiencia de los participantes osciló entre 52 y 126 minutos, con un tiempo de promedio de 86.78 minutos y una desviación estándar (nivel de dispersión) de 16.75 minutos. Esto evidencia que, la eficiencia en el uso de Monitor-IoT puede variar considerablemente entre los participantes dependiendo de algunos factores, tales como: la capacidad de discernimiento, la prolijidad con la que realizan las tareas; y, la experiencia y destreza en el uso de las herramientas de modelado de software. Sin embargo, a pesar de esas particularidades, el propósito principal de este cuasi experimento consistió en probar si el rendimiento del usuario influye en su percepción sobre la facilidad de uso y utilidad del DSL Monitor-IoT.

Tabla 7-5. Estadística descriptiva de las variables de rendimiento (Eficiencia y Efectividad) para el cuasi experimento Nro. 1.

Variable	Mín.	Máx.	Media	Desviación Estándar
Efectividad	0.5000	1.0000	0.8698	0.1569
Eficiencia (minutos)	52.0000	126.0000	86.7813	16.7520

Cabe recalcar que a pesar de que la capacitación sobre el uso del DSL Monitor-IoT tuvo una duración mínima, los resultados de rendimiento fueron muy satisfactorios; de ahí que, la reducción de los errores cometidos por los participantes se puede lograr mediante una capacitación más profunda, basada en una mayor cantidad de ejercicios prácticos.

Además, los resultados sobre el rendimiento de los participantes, complementados con las respuestas obtenidas para la pregunta abierta del cuestionario de percepción, fueron de gran utilidad para el refinamiento y la mejora continua del DSL Monitor-IoT y el proceso metodológico Aware-IoT@Run.Time en general.

7.3.4.3 Análisis de las relaciones causales

En este punto se valida la parte estructural del MEM, en términos de las relaciones causales entre las variables Eficiencia, Efectividad, Facilidad de Uso Percibida (PEOU), Utilidad Percibida (PU) e Intención de Uso (ITU). Para esto, se ha utilizado el método estadístico de regresión lineal simple como mecanismo para probar las hipótesis H4₀, H5₀, H6₀, H7₀ y H8₀, las cuales están definidas como relaciones causales entre las variables antes indicadas. A continuación, se describen los resultados obtenidos para cada una de las relaciones causales:

- **Eficiencia vs. Facilidad de Uso Percibida.** El modelo de regresión lineal construido se presenta en la Ecuación 7-3, donde la Eficiencia es la variable independiente y PEOU la variable dependiente. El modelo resultante no es significativo ($p > 0.1$) y el coeficiente de determinación (R^2) indica que la variable Eficiencia permite explicar únicamente el 2.4% de la varianza de PEOU (ver Tabla 7-6). En consecuencia, los resultados no permiten rechazar la hipótesis nula H4₀ y aceptar su hipótesis alternativa, lo cual significa que la eficiencia actual de los participantes no influye en su percepción sobre la facilidad de uso.

$$PEOU = 4.016 + 0.005 * Eficiencia \quad (7-3)$$

Tabla 7-6. Modelo de regresión lineal simple entre Eficiencia y Facilidad de Uso Percibida para el cuasi experimento Nro. 1.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R ²
Constante	4.016	0.485	8.281	0.000		
Eficiencia	0.005	0.005	0.858	0.398 > 0.1	0.155	0.024

- **Efectividad vs Utilidad Percibida.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-4, donde la Efectividad es la variable independiente y PU la variable dependiente. El modelo resultante tiene una alta significancia ($p < 0.01$) y el coeficiente de determinación (R^2) indica que la variable Efectividad permite explicar el 29.3% de la varianza de PU (ver Tabla 7-7). En consecuencia, los resultados permiten rechazar la hipótesis nula H5₀ y aceptar su hipótesis alternativa, lo cual significa que la efectividad actual de los participantes incide parcialmente en su percepción sobre la utilidad.

$$PU = 3.096 + 1.676 * Efectividad \quad (7-4)$$

Tabla 7-7. Modelo de regresión lineal simple entre Efectividad y Utilidad Percibida para el cuasi experimento Nro. 1.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R ²
Constante	3.090	0.420	7.364	0.000		
Efectividad	1.676	0.475	3.528	0.001 < 0.01	0.542	0.293

- **Facilidad de Uso Percibida vs Utilidad Percibida.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-5, donde PEOU es la variable independiente y PU la variable dependiente. El modelo resultante tiene una alta significancia ($p < 0.01$) y el coeficiente de determinación (R^2) indica que la variable PEOU permite explicar el 33.1% de la varianza de PU (ver Tabla 7-8). En consecuencia, los resultados

permiten rechazar la hipótesis nula H_{60} y aceptar su hipótesis alternativa, lo cual significa que la utilidad percibida sí está determinada en parte por la facilidad de uso percibida por los participantes.

$$PU = 2.124 + 0.547 * PEOU \quad (7-5)$$

Tabla 7-8. Modelo de regresión lineal simple entre Facilidad de Uso Percibida y Utilidad Percibida para el cuasi experimento Nro. 1.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R²
<i>Constante</i>	2.124	0.633	3.355	0.002		
<i>PEOU</i>	0.547	0.142	3.850	0.001 < 0.01	0.575	0.331

- **Facilidad de Uso Percibida vs Intención de Uso.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-6, donde PEOU es la variable independiente e ITU la variable dependiente. El modelo resultante tiene una muy alta significancia ($p < 0.001$) y el coeficiente de determinación (R^2) indica que la variable PEOU permite explicar el 50.1% de la varianza de ITU (ver Tabla 7-9). De ahí que, los resultados permiten rechazar la hipótesis nula H_{70} y aceptar su hipótesis alternativa, lo cual corrobora que la intención de uso sí está determinada en parte por la facilidad de uso percibida por los participantes.

$$ITU = 1.186 + 0.763 * PEOU \quad (7-6)$$

Tabla 7-9. Modelo de regresión lineal simple entre Facilidad de Uso Percibida e Intención de Uso para el cuasi experimento Nro. 1.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R²
<i>Constante</i>	1.186	0.620	1.914	0.065		
<i>PEOU</i>	0.763	0.139	5.483	0.000 < 0.001	0.707	0.501

- **Utilidad Percibida vs Intención de Uso.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-7, donde PU es la variable independiente e ITU la variable dependiente. El modelo resultante tiene una muy alta significancia ($p < 0.001$) y el coeficiente de determinación (R^2) indica que la variable PU permite explicar el 48.8% de la varianza de ITU (ver Tabla 7-10). Por lo tanto, los resultados permiten rechazar la hipótesis nula H_{80} y aceptar su hipótesis alternativa, lo cual significa que la intención de uso sí está determinada en parte por la utilidad percibida por los participantes.

$$ITU = 0.964 + 0.791 * PU \quad (7-7)$$

Tabla 7-10. Modelo de regresión lineal simple entre Utilidad Percibida e Intención de Uso para el cuasi experimento Nro. 1.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R²
<i>Constante</i>	0.964	0.677	1.425	0.165		
<i>PU</i>	0.791	0.148	5.348	0.000 < 0.001	0.699	0.488

7.3.4.4 Síntesis de los resultados del cuasi experimento

La Figura 7-5 presenta una síntesis de los resultados obtenidos en la evaluación del DSL Monitor-IoT con el apoyo del MEM; mientras que, la Tabla 7-11 muestra un resumen de las hipótesis aceptadas y rechazadas durante el cuasi experimento. Con base en estos resultados, a continuación, se exponen las principales conclusiones a las que se ha llegado para cada una de las preguntas de investigación planteadas en este cuasi experimento:

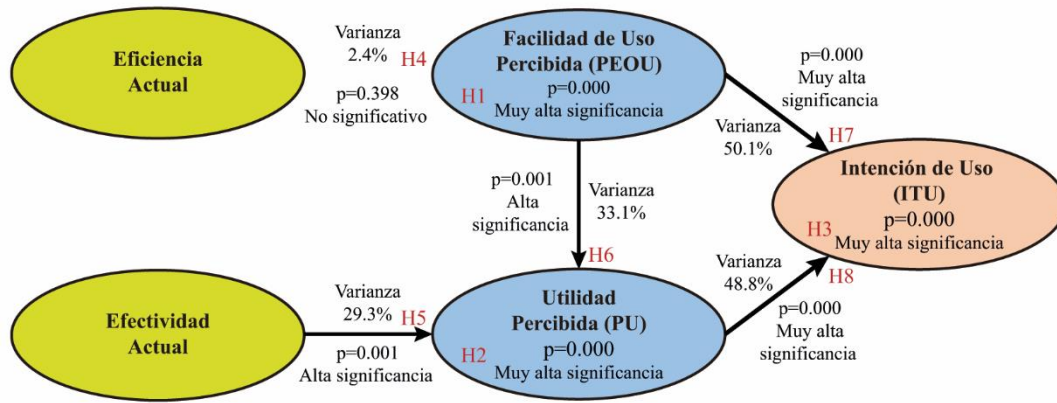


Figura 7-5. Síntesis de los resultados del cuasi experimento Nro. 1.

Fuente: Elaboración propia.

Tabla 7-11. Resumen de hipótesis aceptadas y rechazadas en el cuasi experimento Nro. 1.

Participantes	Número de participantes	Hipótesis aceptadas	Hipótesis rechazadas
Estudiantes de los últimos semestres de la carrera de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca.	33	H1 ₁ , H2 ₁ , H3 ₁ , H4 ₀ , H5 ₁ , H6 ₁ , H7 ₁ , H8 ₁ .	H1 ₀ , H2 ₀ , H3 ₀ , H4 ₁ , H5 ₀ , H6 ₀ , H7 ₀ , H8 ₀ .

RQ1. *¿El DSL Monitor-IoT es percibido como fácil de usar y útil? De ser así ¿La percepción del usuario es el resultado de su rendimiento al utilizar el DSL Monitor-IoT?*

Las hipótesis nulas H1₀ y H2₀ fueron rechazadas, demostrando que la mayoría de los participantes perciben al DSL Monitor-IoT y su proceso metodológico como fácil de usar y útil. Incluso, esto se puede corroborar con la efectividad media alcanzada por los participantes del 86.98%, lo cual significa que en su mayoría fueron capaces de cumplir correctamente con las tareas experimentales relacionadas con el diseño de la arquitectura de monitoreo del subsistema IoT en cuestión.

Con respecto a la influencia del rendimiento del usuario sobre su percepción, se analizaron las hipótesis nulas H4₀ y H5₀. Por su parte, la hipótesis H4₀ no pudo ser rechazada, por lo que la facilidad de uso no está determinada por la eficiencia actual de los participantes. Esto puede deberse a que, aunque la mayoría de los participantes perciben a la solución como fácil de usar, existe mucha variabilidad en la eficiencia de los participantes al ejecutar las tareas experimentales; sin embargo, esta brecha podrá reducirse con una capacitación más profunda y a medida que los usuarios conozcan mejor la solución y alcancen destrezas en su uso. Por el contrario, la hipótesis H5₀ fue rechazada, lo cual implica que la efectividad actual de los participantes influye en su percepción sobre la utilidad de la solución.

En cuanto a la evaluación cualitativa del experimento (pregunta abierta del cuestionario de percepción), un aspecto destacable es que varios participantes mencionaron la importancia de disponer de un proceso metodológico (Aware-IoT@Run.Time) asociado al DSL Monitor-IoT, ya que les fue de gran ayuda para cumplir correctamente con las tareas y requerimientos del experimento. Adicionalmente, los participantes han proporcionado sugerencias valiosas, mismas que han sido consideradas para mejorar la calidad de la solución.

Por otro lado, los escenarios de IoT utilizados durante la prueba piloto y los ejercicios de entrenamiento y experimental demostraron que Monitor-IoT tiene un lenguaje de dominio lo suficientemente completo para representar los conceptos clave que pueden requerir los diversos entornos de monitoreo de IoT. En este sentido, la evaluación de Monitor-IoT contempló: i) diferentes tipos de entidades físicas y propiedades a monitorear; ii) diversidad de nodos de computación, junto con sus recursos esenciales (APIs, aplicaciones, servicios, bases de datos y su estructura, brokers, tópicos, interfaces de red), a nivel de las capas Edge, Fog y Cloud; y, iii) flujos de comunicación, tanto síncronos como asíncronos. Asimismo, la evaluación evidenció un ahorro de esfuerzo y tiempo en el modelado de las arquitecturas de los escenarios de evaluación, debido al uso de un lenguaje que soporta la definición de plantillas de propiedades para entidades IoT del mismo tipo.

RQ2. ¿Existe la intención de usar el DSL Monitor-IoT en el futuro? De ser así ¿La intención de uso es el resultado de la percepción del usuario?

La hipótesis nula H_{30} fue rechazada, evidenciando la disposición favorable de los participantes de utilizar el DSL Monitor-IoT para diseñar arquitecturas de monitoreo de IoT en proyectos futuros.

Con relación a la influencia de la percepción del usuario sobre su intención de uso, se analizaron las hipótesis nulas H_{60} , H_{70} y H_{80} . En este sentido, la hipótesis H_{60} fue rechazada, demostrando que los participantes al percibir a Monitor-IoT como fácil de usar, también lo consideran útil. Asimismo, las hipótesis H_{70} y H_{80} fueron rechazadas, concluyendo que la percepción de los participantes sobre la facilidad de uso y utilidad de Monitor-IoT condiciona parcialmente sus intenciones de usar esta herramienta a futuro; sin embargo, pueden existir otras variables que influyen en la intención de uso de los participantes, las cuales serán objeto de análisis en trabajos futuros.

En definitiva, aunque la herramienta DSL Monitor-IoT se encuentra en estado de prototipo, los resultados alcanzados son alentadores, por tal motivo, se tiene previsto continuar con la mejora de la solución y su evaluación en otros contexto o subdominios de IoT con la participación de la industria.

7.4 Cuasi experimento Nro. 2: Evaluación del configurador web de autoconsciencia y middleware de la infraestructura

Este cuasi experimento se enfoca en evaluar empíricamente el configurador de capacidades de autoconsciencia y el middleware de la infraestructura, este último compuesto por los motores de sincronización de recursos de software de monitoreo y autoconsciencia de un sistema IoT; es decir, cubre la evaluación de las actividades 2 a la 6 de la metodología Aware-IoT@Run.Time. La actividad 2 contempla que el participante del experimento utilice el configurador, con el objeto de construir un modelo de capacidades de autoconsciencia para un sistema IoT. Para ello, se le entrega al participante como insumo inicial, el correspondiente modelo de arquitectura de monitoreo de dicho sistema. Además, en esta actividad el participante debe generar el archivo JSON con los modelos de arquitectura y autoconsciencia en tiempo de ejecución. Mientras tanto, las actividades de la 3 a la 6 se ejecutan automáticamente por medio del middleware, el cual detecta el archivo JSON, analiza las especificaciones de los modelos de arquitectura y autoconsciencia en tiempo de ejecución; y, consecuentemente, implementa y pone en operación los recursos de monitoreo y autoconsciencia, encargados de producir las métricas directas, indirectas e indicadores que valoran los aspectos de autoconsciencia del sistema IoT observado. En este punto, el participante verifica, tanto en la consola del

middleware como en el visor web de autoconsciencia, que los procesos pre-reflexivos y reflexivos estén ejecutándose y generando las métricas y recomendaciones prescriptivas, conforme a las especificaciones de los modelos en tiempo de ejecución.

Específicamente, en esta sección se describe el diseño, ejecución y análisis de los resultados de este cuasi experimento, con el propósito de comprender el rendimiento (eficiencia y efectividad) del usuario con el uso del configurador de autoconsciencia, el middleware de la infraestructura y el proceso metodológico Aware-IoT@Run.Time en el que se basan, así como su percepción sobre la facilidad de uso, utilidad e intención de uso futuro de esta solución.

7.4.1 Diseño del cuasi experimento

Este cuasi experimento, al igual que el anterior, fue diseñado siguiendo el proceso experimental propuesto por Wohlin et al. [46]. En la Tabla 7-12 se presenta el objetivo del cuasi experimento, de acuerdo con el paradigma Goal-Question Metric (GQM) propuesto por Basili et al. [209].

Tabla 7-12. Goal-Question-Metric para el cuasi experimento Nro. 2.

<i>Evaluar:</i>	El configurador de capacidad de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time.
<i>Con el propósito de:</i>	• Determinar el rendimiento (efectividad y eficiencia) del usuario en el uso del configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time. • Comprender la percepción del usuario sobre la facilidad de uso, utilidad e intención de uso futuro del configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time.
<i>Desde el punto de vista de:</i>	Investigador.
<i>En el contexto de:</i>	Un grupo de estudiantes de los últimos semestres de la carrera de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca.

Las *preguntas de investigación* a responder por medio del cuasi experimento son:

- RQ1.** ¿El configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time son percibidos como fáciles de usar y útiles? De ser así ¿La percepción del usuario es el resultado de su rendimiento al utilizar esta solución tecnológica?
- RQ2.** ¿Existe la intención de usar el configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time? De ser así ¿La intención de uso es el resultado de la percepción del usuario?

Asimismo, las preguntas de investigación fueron evaluadas a través de pruebas de hipótesis. En particular, la primera pregunta fue estudiada mediante las hipótesis:

- H1₀: El configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time se perciben como difíciles de usar, $H1_0 \Rightarrow \neg H1_1$.
- H2₀: El configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time no se perciben como útiles, $H2_0 \Rightarrow \neg H2_1$.
- H4₀: La facilidad de uso percibida no está determinada por la eficiencia actual, $H4_0 \Rightarrow \neg H4_1$.
- H5₀: La utilidad percibida no está determinada por la efectividad actual. $H5_0 \Rightarrow \neg H5_1$.

- H₆₀: La utilidad percibida no está determinada por la facilidad de uso percibida, H₆₀=¬H₆₁.

Mientras que, la segunda pregunta fue analizada por medio de las hipótesis:

- H₃₀: No existe la intención de utilizar el configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time en el futuro, H₃₀=¬H₃₁.
- H₇₀: La intención de uso no está determinada por la facilidad de uso percibida, H₇₀=¬H₇₁.
- H₈₀: La intención de uso no está determinada por la utilidad percibida, H₈₀=¬H₈₁.

El *contexto del cuasi experimento* estuvo determinado por:

- i) El configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time.
- ii) El escenario de IoT utilizado en la evaluación empírica, cuyo alcance fue implementar dos procesos de autoconsciencia en el subsistema SCA, uno pre-reflexivo y otro reflexivo que evalúen la concentración actual y promedio diario de CO en el ambiente de la cocina, respectivamente. Este escenario de IoT es parte del estudio de caso descrito en el Capítulo 6. Cabe recalcar que para desplegar de manera rápida las componentes de software de la infraestructura de soporte en cada una de las computadoras de los participantes del experimento, se implementaron e integraron estos componentes bajo la plataforma Docker, simulando una infraestructura IoT distribuida en varios nodos de computación, a nivel Edge, Fog y Cloud. Específicamente, los componentes de la infraestructura de soporte se dividieron en los siguientes contenedores de Docker: i) *configurador-autoconsciencia-1*, ejecuta el configurador web de capacidades de autoconsciencia; ii) *db-configurador-autoconsciencia-1*, base de datos en MySQL del configurador de autoconsciencia; iii) *app-supervisora-nodo-fog-1*, ejecuta la aplicación supervisora en el nodo Fog; iv) *app-implementadora-nodo-fog-1*, ejecuta la aplicación implementadora en el nodo Fog; v) *db-nodo-Fog*, base de datos en PostgreSQL que almacena los datos crudos de monitoreo y las métricas de autoconsciencia en el nodo Fog; vi) *visor-autoconsciencia-1*, ejecuta la aplicación web que visualiza las métricas resultantes de los procesos de autoconsciencia; vii) *app-implementadora-nodo-edge-1*, ejecuta la aplicación implementadora en el nodo Edge; viii) *app-implementadora-nodo-cloud-1*, ejecuta la aplicación implementadora en el nodo Cloud; y, ix) *db-nodo-cloud-1*, base de datos en PostgreSQL que almacena los datos de resumen diario de monitoreo en el nodo Cloud. Los dos últimos contenedores no son requeridos en este cuasi experimento; sin embargo, se encuentran desplegados como parte de la plataforma de Docker, ya que el cuasi experimento utiliza el mismo modelo de arquitectura de monitoreo de IoT del estudio de caso implementado en el Capítulo 6. En la Figura 7-6 se observan los contenedores de Docker en estado de ejecución durante el cuasi experimento. Con respecto a la obtención de datos desde el sensor de CO, se implementó una API que encapsula instrucciones que generan valores aleatorios entre 0 y 1200 partes por millón (PPM) de concentración de CO.
- iii) La participación de 42 estudiantes de los últimos semestres de la carrera de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca, con sólidos conocimientos sobre Arquitecturas de Software, Desarrollo Dirigido por Modelos e Internet de las Cosas. A pesar de ello, durante la etapa de entrenamiento del cuasi experimento, fue necesario capacitar a los participantes en Autoconsciencia Computacional (conceptos, tipos y niveles).

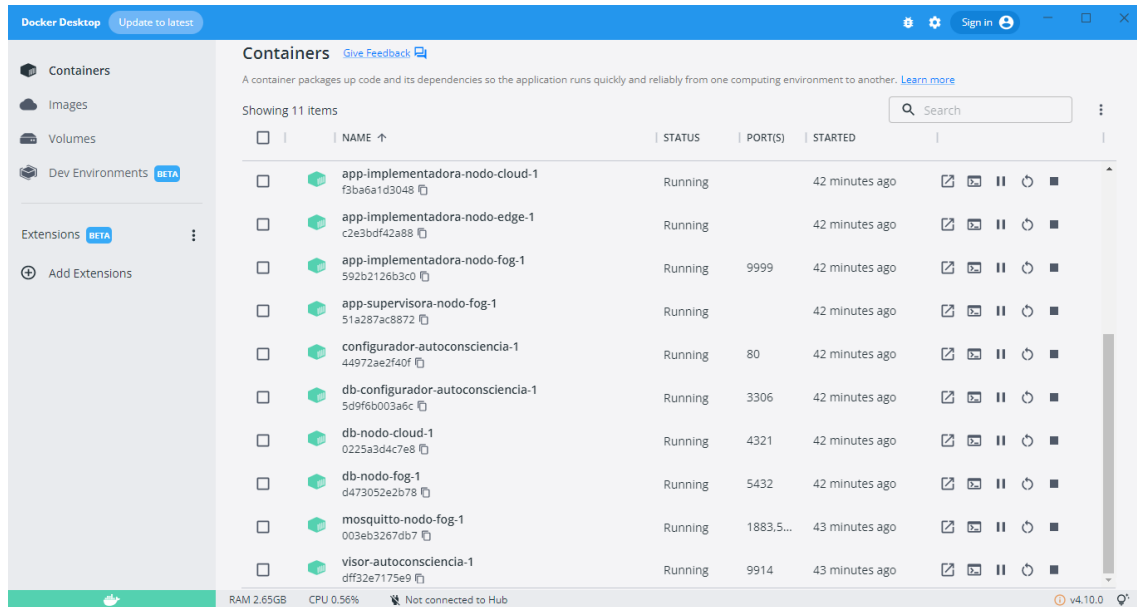


Figura 7-6. Plataforma de Docker desplegada durante el cuasi experimento Nro. 2.

Fuente: Elaboración propia.

Las *tareas experimentales del cuasi experimento* fueron definidas de acuerdo al proceso metodológico de Aware-IoT@Run.Time. El cuasi experimento se centró fundamentalmente en evaluar las tareas manuales de la Actividad 2 de la metodología Aware-IoT@Run.Time (ver Capítulo 4), relacionadas con la especificación de las capacidades de autoconsciencia de un sistema IoT, utilizando el configurador de autoconsciencia; así como la consecuente generación del archivo JSON con los modelos integrados de arquitectura y autoconsciencia. Además, como las actividades de la 3 a la 6 de la metodología Aware-IoT@Run.Time son transparentes al usuario, ya que se ejecutan automáticamente (sin la intervención del usuario) por medio del middleware de la infraestructura, una vez detectado el archivo JSON con los modelos en tiempo de ejecución; fue necesario incluir una tarea experimental final, con la intención de que los participantes del cuasi experimento, utilicen la consola del middleware y el visor de autoconsciencia para verificar que los procesos pre-reflexivos y reflexivos implementados automáticamente estén ejecutándose y generando las métricas y recomendaciones prescriptivas, conforme a las especificaciones establecidas en los modelos en tiempo de ejecución. Por lo tanto, las principales tareas del cuasi experimento fueron:

- **Tarea 1:** Configuración del dominio de autoconsciencia.
- **Tarea 2:** Definición de los recursos de soporte a los procesos de autoconsciencia.
- **Tarea 3:** Definición de los procesos y métodos de autoconsciencia (aprendizaje y razonamiento).
 - **Tarea 3.1:** Definición del proceso de autoconsciencia pre-reflexivo y su método de recolección y modelo de análisis.
 - **Tarea 3.2:** Definición del proceso de autoconsciencia reflexivo y su método de cálculo y modelo de análisis.
- **Tarea 4:** Generación del modelo de autoconsciencia del sistema IoT.
- **Tarea 5:** Monitorización de los procesos de autoconsciencia implementados y de las métricas y recomendaciones prescriptivas generadas para los aspectos de autoconsciencia.

Cabe mencionar que únicamente las tres primeras tareas experimentales fueron consideradas para evaluar las variables de rendimiento (eficiencia y efectividad). La tarea

4 no fue incluida debido a su simplicidad, mientras que, la tarea 5 se utilizó para que los participantes puedan verificar y comprender cómo los procesos de autoconsciencia operan automáticamente y los resultados que estos generan.

Las *variables dependientes del cuasi experimento* para evaluar el configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time fueron las variables de rendimiento (eficiencia y efectividad) y percepción de usuario (facilidad de uso, utilidad e intención de uso) definidas en el MEM. Estas variables permitieron probar las hipótesis y su forma de medir es presentada en la subsección 7.4.2.

El *material experimental del cuasi experimento* contempló la documentación, herramientas y artefactos necesarios para realizar y evaluar las tareas experimentales. De igual manera, todo el material experimental fue distribuido a los participantes a través de un sitio web. Por un lado, las herramientas y artefactos requeridos para el desarrollo del cuasi experimento fueron:

- i) La aplicación Docker Desktop utilizada para ejecutar los contenedores de Docker que contienen los componentes de la infraestructura de soporte a la metodología Aware-IoT@Run.Time.
- ii) El modelo de arquitectura de monitoreo del subsistema SCA en formato XMI (*ModeloArquitecturaMonitoreoSCA.MonitorIoT*), cuyas especificaciones son requeridas para configurar las capacidades de autoconsciencia del subsistema SCA.
- iii) Un archivo empaquetado *ContenedoresInfraestructura.zip* con la estructura de directorios (volúmenes) de la máquina host. Estos directorios son compartidos entre los contenedores de Docker y almacenan de manera persistente: el archivo JSON con los modelos de arquitectura y autoconsciencia en tiempo de ejecución; y, los archivos de las bases de datos de la infraestructura. Además, este empaquetado incluye el archivo *docker-compose.yml* con las configuraciones de los contenedores de Docker que definen cómo deben ejecutarse de manera orquestada.

Por otro lado, la documentación de soporte, incluyó:

- i) Una presentación utilizada por el investigador para capacitar a los participantes sobre los conceptos generales de Internet de las Cosas, Ingeniería Dirigida por Modelos (*models@run.time*) y Autoconsciencia Computacional. Además, la presentación incluye una explicación sobre el proceso metodológico Aware-IoT@Run.Time y su infraestructura de soporte (configurador de autoconsciencia, middleware y visor de autoconsciencia). Finalmente, la presentación describe cómo instalar las herramientas de la infraestructura de soporte bajo la plataforma Docker, los objetivos de la evaluación empírica; y, un ejercicio de entrenamiento que contempla la configuración e implementación de procesos de autoconsciencia pre-reflexivos y reflexivos para evaluar la carga de procesamiento de la CPU del nodo Fog. Este ejercicio es parte del estudio de caso presentado en el Capítulo 6.
- ii) Una guía metodológica con la descripción de las actividades, tareas y subtareas de Aware-IoT@Run.Time (ver Capítulo 4).
- iii) Un boletín y hoja de trabajo con la descripción e instrucciones del ejercicio experimental utilizado para evaluar las herramientas propuestas. Esta hoja de trabajo incluye varios campos para recolectar métricas sobre las actividades ejecutadas por los participantes y los productos obtenidos (ver Anexo E).
- iii) Un cuestionario en Google Forms para medir la percepción del usuario sobre la solución propuesta, una vez concluido el experimento.

7.4.2 Adaptación del MEM para la evaluación del configurador web de autoconsciencia y middleware de la infraestructura de soporte

De la misma forma que en el cuasi experimento anterior, se definieron los constructores del MEM en función de las variables de rendimiento y percepción relevantes para el configurador web de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time. A su vez, también se adaptaron los instrumentos de medición de estas variables bajo el contexto de la solución a evaluar.

En este sentido, la variable *Eficiencia Actual* se definió como la medición del esfuerzo requerido para configurar e implementar procesos de autoconsciencia pre-reflexivos y reflexivos para un sistema IoT, utilizando las herramientas y el proceso metodológico de Aware-IoT@Run.Time; mientras que, la variable *Efectividad Actual* se estableció como la medición de la calidad de las especificaciones del modelo de autoconsciencia en tiempo de ejecución resultante, mismas que sirven para implementar los procesos de autoconsciencia encargados de generar las métricas de evaluación de los aspectos autoconsciencia del sistema IoT. Las fórmulas utilizadas para evaluar la eficiencia actual y efectividad actual se encuentran definidas en las Ecuaciones 7-1 y 7-2, respectivamente; siendo las mismas fórmulas que se emplearon en el primer cuasi experimento.

Por otro lado, para la medición de las variables de percepción del MEM, en la Tabla 7-13 se presenta un cuestionario adaptado y conformado por trece preguntas: i) seis preguntas para *Facilidad de Uso Percibida (PEOU)*; ii) cuatro preguntas para *Utilidad Percibida (PU)*; iii) dos preguntas para *Intención de Uso (ITU)*; y, iv) una pregunta abierta para recopilar sugerencias sobre la solución tecnológica. Asimismo, las preguntas fueron formuladas utilizando la escala de Likert (5 puntos), asumiendo 3 como valor neutro, a partir del cual se puede aceptar o rechazar la solución propuesta.

Tabla 7-13. Cuestionario para medir las variables de percepción del MEM para el cuasi experimento Nro. 2.

Nro.	Pregunta
PEOU1	Las herramientas de la infraestructura de soporte a la metodología Aware-IoT@Run.Time son fáciles de instalar y poner en operación.
PEOU2	El configurador web de capacidades de autoconsciencia para sistemas IoT es intuitivo y fácil de usar.
PEOU3	El proceso metodológico Aware-IoT@Run.Time utilizado por el configurador de autoconsciencia y middleware para implementar procesos de autoconsciencia pre-reflexivos y reflexivos en sistemas IoT es claro y fácil de entender.
PEOU4	El proceso metodológico Aware-IoT@Run.Time utilizado por el configurador de autoconsciencia y middleware para implementar procesos de autoconsciencia pre-reflexivos y reflexivos en sistemas IoT es fácil de aplicar.
PEOU5	El configurador de autoconsciencia y middleware ayudan a abstraer y reducir los detalles de implementación (programación a bajo nivel) de un sistema IoT autoconsciente.
PEOU6	El visor de autoconsciencia para consultar las métricas de evaluación de los aspectos de autoconsciencia de sistemas IoT es intuitivo y fácil de usar.
PU1	El uso del configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time podrían reducir el tiempo y esfuerzo requerido en el desarrollo, despliegue y mantenimiento de sistemas IoT.
PU2	El configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time son útiles para apoyar las actividades de desarrollo, despliegue y mantenimiento de sistemas IoT.
PU3	El uso del configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time podrían mejorar su desempeño en el desarrollo, despliegue y mantenimiento de sistemas IoT.

<i>PU4</i>	La metodología Aware-IoT@Run.Time y las herramientas de su infraestructura de soporte (configurador de autoconsciencia, middleware) son útiles para construir sistemas IoT autónomos y resilientes frente a situaciones imprevistas.
<i>ITU1</i>	En caso de necesitar construir sistemas IoT autoconscientes en el futuro, utilizaría la metodología Aware-IoT@Run.Time y las herramientas de su infraestructura de soporte (configurador de autoconsciencia, middleware).
<i>ITU2</i>	Recomendaría el uso de la metodología Aware-IoT@Run.Time y las herramientas de su infraestructura de soporte (configurador de autoconsciencia, middleware) para construir, operar y mantener sistemas IoT autoconscientes.
<i>OQ1</i>	Tiene alguna sugerencia con respecto a la solución tecnológica propuesta para construir, operar y mantener sistemas IoT autoconscientes (<i>pregunta abierta</i>).

Igualmente, las variables del MEM (Eficiencia, Efectividad, PEOU, PU y ITU) fueron utilizadas para probar las hipótesis planteadas en la subsección anterior, con el objeto de predecir la probabilidad de que el configurador de capacidades de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time sean aceptados en la práctica. El modelo adaptado del MEM para evaluar el rendimiento y la percepción del usuario sobre la solución tecnológica propuesta en este cuasi experimento se presenta en la Figura 7-7.

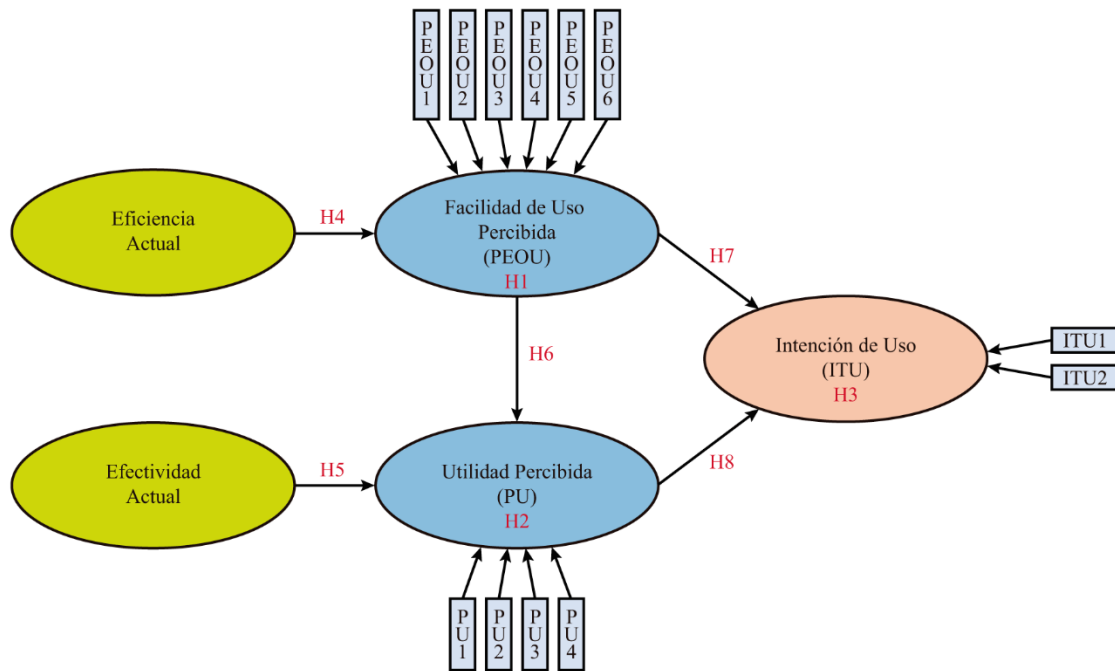


Figura 7-7. Modelo adaptado del MEM para evaluar el configurador de capacidad de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time. Fuente: Elaboración propia.

7.4.3 Ejecución del cuasi experimento

El cuasi experimento contó con la participación de 42 estudiantes de la carrera de Ingeniería de Sistemas, 27 pertenecientes a la Universidad del Azuay y 15 a la Universidad de Cuenca. A su vez, el cuasi experimento se realizó de manera presencial en los laboratorios de informática de las respectivas universidades con una duración de cuatro horas, divididas en dos sesiones.

Durante la primera sesión se instalaron los componentes de la infraestructura de soporte bajo la plataforma Docker y se capacitó a los participantes del experimento. Esta sesión tuvo una duración de dos horas e incluyó las siguientes actividades: i) presentación de un

marco conceptual sobre Internet de las Cosas, Ingeniería Dirigida por Modelos (models@run.time) y Autoconsciencia Computacional; ii) explicación del proceso metodológico de Aware-IoT@Run.Time; iii) revisión de las herramientas de la infraestructura de soporte, con énfasis en las funcionalidades del configurador de autoconsciencia, middleware y visor de autoconsciencia; y, iv) resolución (desarrollo guiado) del ejercicio de entrenamiento, a fin de que los participantes aprendan a construir un modelo de capacidades de autoconsciencia para un sistema IoT utilizando el configurador de autoconsciencia, así como a monitorear los procesos de autoconsciencia implementados y las correspondientes métricas generadas a través de la consola del middleware y el visor de autoconsciencia.

La segunda sesión se enfocó en el desarrollo del ejercicio experimental por parte de los participantes. Esta sesión tuvo una duración de dos horas, distribuidas de la siguiente manera: i) 20 minutos para presentar el caso práctico a resolver (ver Anexo E); ii) 90 minutos para configurar e implementar los procesos de autoconsciencia pre-reflexivos y reflexivos planteados en el ejercicio experimental; y, iii) 10 minutos para responder un cuestionario en Google Forms con las preguntas presentadas en la Tabla 7-13, así como para enviar la hoja de trabajo resuelta del ejercicio experimental y el archivo JSON resultante con los modelos de arquitectura y autoconsciencia. De igual modo, el investigador responsable del experimento aclaró cualquier duda durante el desarrollo del ejercicio experimental.

7.4.4 Análisis de los resultados del cuasi experimento

Tal como en el cuasi experimento anterior, se utilizó la estadística descriptiva, los diagramas de caja (box plots) y las pruebas de hipótesis con el apoyo de la herramienta SPSS V20 (con $\alpha = 0.05$). Los datos recogidos fueron analizados con el objetivo de probar las hipótesis establecidas en el experimento, mismas que fueron formuladas en términos de las variables de rendimiento y percepción del MEM (Eficiencia, Efectividad, PEOU, PU y ITU). Asimismo, para la aceptación o rechazo de las hipótesis se utilizaron los niveles significancia sugeridos por Moody [210] presentados en la Tabla 7-3.

A su vez, en el Anexo F se encuentran detallados los datos recogidos y calculados por participante para cada una de las variables de rendimiento y percepción del MEM.

7.4.4.1 Análisis de la percepción del usuario

La Figura 7.8 presenta los diagramas de caja de las variables de percepción, teniendo como común denominador que la media de cada variable es mayor al valor neutro (3) de la escala de Likert (ver también Tabla 7-14). En el diagrama de caja de la variable Intención de Uso (ITU) se observa un único dato atípico, perteneciente al participante con identificador 39, el cual ha sido excluido en los análisis posteriores, a fin de obtener resultados estadísticos representativos.

Como siguiente paso, se aplicó la prueba de Shapiro-Wilk para comprobar si las variables de percepción tienen una distribución normal y seleccionar el tipo de prueba a emplear para contrastar las hipótesis nulas H_{10} , H_{20} y H_{30} . La penúltima columna de la Tabla 7-14 presenta los resultados de la prueba de Shapiro-Wilk, demostrando que todas las variables de percepción (PEOU, PU y ITU) no tienen una distribución normal ($p < 0.05$). De ahí que, para probar las hipótesis se seleccionó la prueba de Wilcoxon one-tailed one-sample (prueba no paramétrica) con un valor de prueba igual a 3, correspondiente al valor neutral de la escala de Likert.

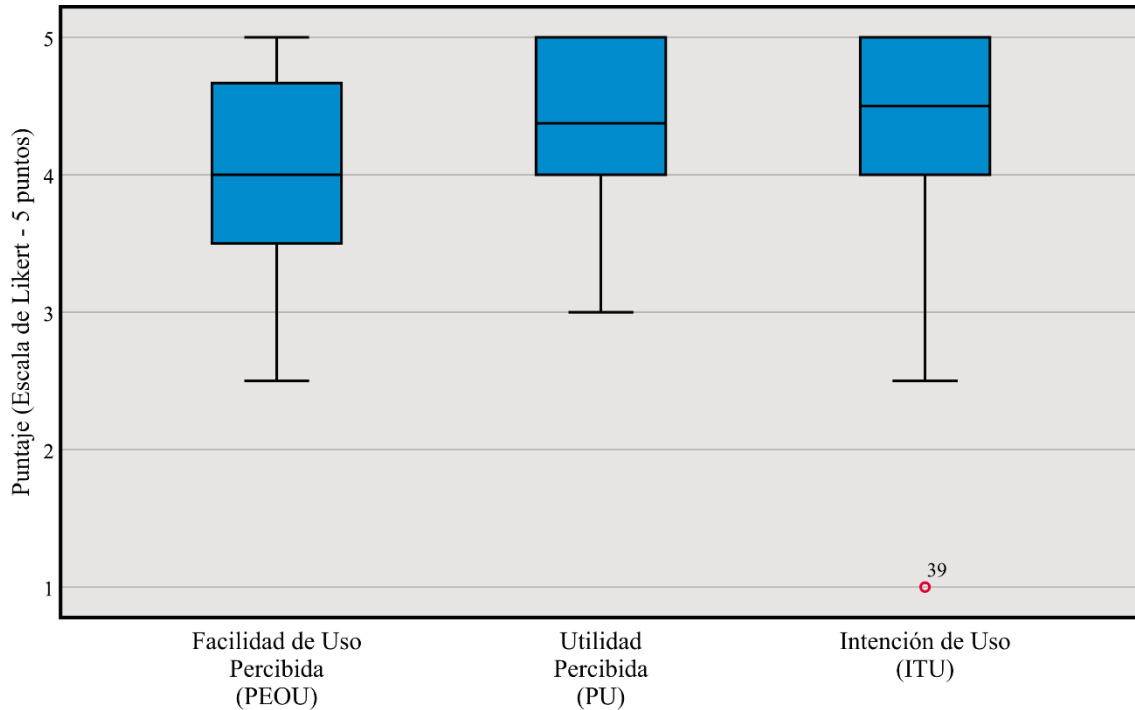


Figura 7-8. Diagramas de caja de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 2. Fuente: Elaboración propia.

La última columna de la Tabla 7-14 presenta los resultados de la prueba Wilcoxon one-tailed one-sample para cada una de las variables de percepción (PEOU, PU y ITU), evidenciando que todas las variables tienen un muy alto nivel de significancia ($p < 0.001$) con relación al valor de prueba 3. En consecuencia, estos resultados permiten rechazar las hipótesis nulas $H1_0$, $H2_0$ y $H3_0$, lo que significa que los participantes perciben al configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time como fáciles de usar y útiles; y, a su vez, tienen la intención de utilizarlas en proyectos futuros para construir sistemas IoT autoconscientes.

Tabla 7-14. Estadística descriptiva y pruebas de hipótesis de las variables de percepción (PEOU, PU y ITU) para el cuasi experimento Nro. 2.

Variable	Mín.	Máx.	Media	Desviación Estándar	Shapiro-Wilk test p-value	Wilcoxon one-tailed one-sample test p-value
PEOU	2.5000	5.0000	4.0813	0.7013	0.029 < 0.05	0.000 < 0.001
PU	3.0000	5.0000	4.3537	0.6446	0.000 < 0.05	0.000 < 0.001
ITU	2.5000	5.0000	4.3293	0.7212	0.000 < 0.05	0.000 < 0.001

7.4.4.2 Análisis del rendimiento del usuario

La Tabla 7-15 muestra los resultados de las medidas de estadística descriptiva para las variables de rendimiento (Efectividad y Eficiencia). La efectividad promedio del cuasi experimento fue de 91.87%, evidenciando que los participantes fueron capaces de cumplir en gran medida con las tareas experimentales relacionadas con la configuración e implementación de los procesos de autoconsciencia pre-reflexivos y reflexivos propuestos en el cuasi experimento. Particularmente, las tareas experimentales 1, 2 y 3 alcanzaron niveles de cumplimiento del 90.24%, 100.00% y 85.37%, respectivamente (ver Anexo F); siendo el mapeo incorrecto entre los parámetros de los recursos de implementación (funciones) y las métricas, el error más frecuente cometido por los participantes del experimento.

Por su parte, la eficiencia de los participantes osciló entre 40 y 83 minutos, con un tiempo de promedio de 57.71 minutos y una desviación estándar de 8.87 minutos. Esto representa un nivel de dispersión o variabilidad moderado, el cual igualmente puede ser consecuencia de la capacidad de discernimiento de los participantes, la prolijidad con la que realizan las tareas; y, la experiencia en el uso de las herramientas de software.

Tabla 7-15. Estadística descriptiva de las variables de rendimiento (Eficiencia y Efectividad) para el cuasi experimento Nro. 2.

Variable	Mín.	Máx.	Media	Desviación Estándar
<i>Efectividad</i>	0.7500	1.0000	0.9187	0.0941
<i>Eficiencia (minutos)</i>	40.0000	83.0000	57.7073	8.8748

Pese al limitado tiempo para la capacitación de los participantes en la metodología Aware-IoT@Run.Time y sus herramientas de soporte, los resultados de rendimiento fueron muy alentadores. Sin lugar a dudas, mediante una capacitación práctica más prolongada sobre el uso de la solución propuesta se puede alcanzar una reducción de los errores cometidos por los participantes.

Asimismo, estos resultados y las respuestas a la pregunta abierta del cuestionario de percepción permitieron obtener una retroalimentación sobre cómo los usuarios utilizan la solución; y, a partir de estos hallazgos, mejorar el proceso metodológico Aware-IoT@Run.Time y la usabilidad de las herramientas de soporte objeto de esta evaluación.

7.4.4.3 Análisis de las relaciones causales

Con respecto a la parte estructural del MEM de este cuasi experimento, a continuación, se describen los resultados de la aplicación del método de regresión lineal simple a las relaciones causales entre las variables Eficiencia, Efectividad, Facilidad de Uso Percibida (PEOU), Utilidad Percibida (PU) e Intención de Uso (ITU), como instrumento para probar las hipótesis H4₀, H5₀, H6₀, H7₀ y H8₀:

- **Eficiencia vs. Facilidad de Uso Percibida.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-8, donde la Eficiencia es la variable independiente y PEOU la variable dependiente. El modelo resultante no es significativo ($p > 0.1$) y el coeficiente de determinación (R^2) indica que la variable Eficiencia permite explicar solamente el 0.3% de la varianza de PEOU (ver Tabla 7-16). En consecuencia, los resultados no permiten rechazar la hipótesis nula H4₀ y aceptar su hipótesis alternativa, lo cual significa que la eficiencia actual de los participantes no influye en su percepción sobre la facilidad de uso.

$$PEOU = 4.311 - 0.004 * Eficiencia \quad (7-8)$$

Tabla 7-16. Modelo de regresión lineal simple entre Eficiencia y Facilidad de Uso Percibida para el cuasi experimento Nro. 2.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R ²
<i>Constante</i>	4.311	0.738	5.844	0.000		
<i>Eficiencia</i>	-0.004	0.013	-0.315	0.755 > 0.1	0.050	0.003

- **Efectividad vs Utilidad Percibida.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-9, donde la Efectividad es la variable independiente y PU la variable dependiente. El modelo resultante tiene una significancia media ($p < 0.05$) y el coeficiente de determinación (R^2) indica que la variable Efectividad permite explicar el 11.6% de la varianza de PU (ver Tabla 7-17). En consecuencia, los resultados

permiten rechazar la hipótesis nula H_{50} y aceptar su hipótesis alternativa, lo cual significa que la efectividad actual de los participantes incide en su percepción sobre la utilidad.

$$PU = 2.213 + 2.33 * Efectividad \quad (7-9)$$

Tabla 7-17. Modelo de regresión lineal simple entre Efectividad y Utilidad Percibida para el cuasi experimento Nro. 2.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R²
<i>Constante</i>	2.213	0.953	2.323	0.026		
<i>Efectividad</i>	2.330	1.032	2.258	0.030 < 0.05	0.340	0.116

- **Facilidad de Uso Percibida vs Utilidad Percibida.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-10, donde PEOU es la variable independiente y PU la variable dependiente. El modelo resultante tiene una muy alta significancia ($p < 0.001$) y el coeficiente de determinación (R^2) indica que la variable PEOU permite explicar el 49.6% de la varianza de PU (ver Tabla 7-18). En consecuencia, los resultados permiten rechazar la hipótesis nula H_{60} y aceptar su hipótesis alternativa, lo cual significa que la utilidad percibida sí está determinada en parte por la facilidad de uso percibida por los participantes.

$$PU = 1.711 + 0.647 * PEOU \quad (7-10)$$

Tabla 7-18. Modelo de regresión lineal simple entre Facilidad de Uso Percibida y Utilidad Percibida para el cuasi experimento Nro. 2.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R²
<i>Constante</i>	1.711	0.432	3.957	0.000		
<i>PEOU</i>	0.647	0.104	6.197	0.000 < 0.001	0.704	0.496

- **Facilidad de Uso Percibida vs Intención de Uso.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-11, donde PEOU es la variable independiente e ITU la variable dependiente. El modelo resultante tiene una muy alta significancia ($p < 0.001$) y el coeficiente de determinación (R^2) indica que la variable PEOU permite explicar el 56.7% de la varianza de ITU (ver Tabla 7-19). De ahí que, los resultados permiten rechazar la hipótesis nula H_{70} y aceptar su hipótesis alternativa, lo cual corrobora que la intención de uso sí está determinada en parte por la facilidad de uso percibida por los participantes.

$$ITU = 1.169 + 0.774 * PEOU \quad (7-11)$$

Tabla 7-19. Modelo de regresión lineal simple entre Facilidad de Uso Percibida e Intención de Uso para el cuasi experimento Nro. 2.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R²
<i>Constante</i>	1.169	0.448	2.606	0.013		
<i>PEOU</i>	0.774	0.108	7.148	0.000 < 0.001	0.753	0.567

- **Utilidad Percibida vs Intención de Uso.** El modelo de regresión lineal obtenido se presenta en la Ecuación 7-12, donde PU es la variable independiente e ITU la variable dependiente. El modelo resultante tiene una muy alta significancia ($p < 0.001$) y el coeficiente de determinación (R^2) indica que la variable PU permite explicar el 65.9% de la varianza de ITU (ver Tabla 7-20). Por lo tanto, los resultados permiten rechazar la hipótesis nula H_{80} y aceptar su hipótesis alternativa, lo cual significa que la intención de uso sí está determinada en parte por la utilidad percibida por los participantes.

$$ITU = 0.374 + 0.908 * PU \quad (7-12)$$

Tabla 7-20. Modelo de regresión lineal simple entre Utilidad Percibida e Intención de Uso para el cuasi experimento Nro. 2.

	Coefficiente	Err. Std.	t.	Sig. (p)	R	R ²
Constante	0.374	0.460	0.813	0.421		
PU	0.908	0.105	8.689	0.000 < 0.001	0.812	0.659

7.4.4.4 Síntesis de los resultados del cuasi experimento

La Figura 7-9 presenta una síntesis de los resultados obtenidos en la evaluación del configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time; mientras que, la Tabla 7-21 muestra un resumen de las hipótesis aceptadas y rechazadas durante el cuasi experimento. A partir de estos resultados, las principales conclusiones para cada una de las preguntas de investigación planteadas en el cuasi experimento son las siguientes:

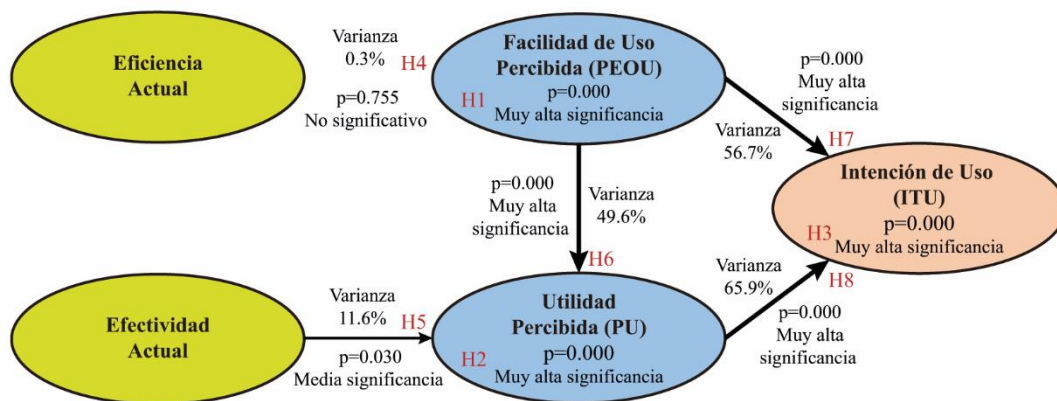


Figura 7-9. Síntesis de los resultados del cuasi experimento Nro. 2.
Fuente: Elaboración propia.

Tabla 7-21. Resumen de hipótesis aceptadas y rechazadas en el cuasi experimento Nro. 2.

Participantes	Número de participantes	Hipótesis aceptadas	Hipótesis rechazadas
Estudiantes de los últimos semestres de la carrera de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca.	42	H1 ₁ , H2 ₁ , H3 ₁ , H4 ₀ , H5 ₁ , H6 ₁ , H7 ₁ , H8 ₁ .	H1 ₀ , H2 ₀ , H3 ₀ , H4 ₁ , H5 ₀ , H6 ₀ , H7 ₀ , H8 ₀ .

RQ1. *¿El configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time son percibidos como fáciles de usar y útiles? De ser así ¿La percepción del usuario es el resultado de su rendimiento al utilizar esta solución tecnológica?*

Las hipótesis nulas H1₀ y H2₀ fueron rechazadas, demostrando que la mayoría de los participantes perciben al configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología Aware-IoT@Run.Time como fáciles de usar y útiles. Además, estos resultados son concordantes con la efectividad promedio de los participantes del 91,87%, evidenciando que gran parte de los mismos fueron capaces de cumplir correctamente con las tareas experimentales sobre la configuración e implementación de los procesos de autoconsciencia pre-reflexivos y reflexivos requeridos por el escenario de evaluación.

Al respecto de la eficiencia de los participantes, el tiempo promedio (57.71 minutos) invertido para implementar los recursos de software que operacionalizan los procesos de autoconsciencia fue mínimo, en comparación al tiempo que se necesitaría para implementar estos recursos con el apoyo de métodos y herramientas de ingeniería de software tradicionales, basados en enfoques de desarrollo a bajo nivel. A su vez, el nivel de dispersión de la eficiencia (tiempo) de los participantes para completar las tareas experimentales es moderado; inclusive, esta medida podría tender a disminuir conforme los usuarios se familiaricen con la solución metodológica propuesta y sus herramientas.

Un aspecto destacable es el puntaje promedio (4.34/5) alcanzado en la pregunta PEOU5, demostrando que la metodología *Aware-IoT@Run.Time* y sus herramientas son consideradas por la mayoría de los participantes como una solución que ayuda notablemente a abstraer la complejidad y reducir los detalles de implementación (programación a bajo nivel) de un sistema IoT autoconsciente.

En referencia a la influencia del rendimiento del usuario sobre su percepción, se analizaron las hipótesis nulas $H4_0$ y $H5_0$. Por un lado, la hipótesis $H4_0$ no pudo ser rechazada; de ahí que, la eficiencia actual de los participantes no incide en su percepción sobre la facilidad de uso de la solución propuesta, lo cual puede deberse igualmente a la variabilidad en la eficiencia de los participantes. Por otro lado, la hipótesis $H5_0$ fue rechazada, lo cual implica que la efectividad actual de los participantes repercute en su percepción sobre la utilidad de esta solución.

RQ2. ¿Existe la intención de usar el configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología *Aware-IoT@Run.Time*? De ser así ¿La intención de uso es el resultado de la percepción del usuario?

La hipótesis nula $H3_0$ fue rechazada, lo cual demuestra la intención favorable de los participantes a utilizar el configurador de autoconsciencia y middleware de la infraestructura de soporte a la metodología *Aware-IoT@Run.Time* para construir sistemas IoT autoconscientes.

Acerca de la influencia de la percepción del usuario sobre su intención de uso, se analizaron las hipótesis nulas $H6_0$, $H7_0$ y $H8_0$. Por su parte, la hipótesis $H6_0$ fue rechazada, comprobando que los participantes al percibir la solución evaluada como fácil de usar, también lo consideran útil. Asimismo, las hipótesis $H7_0$ y $H8_0$ fueron rechazadas, demostrando que la percepción de los participantes sobre la facilidad de uso y utilidad de la solución incide parcialmente en sus intenciones de usarla a futuro; sin embargo, pueden existir otras variables que influyen en la intención de uso de los participantes, las cuales serán objeto de análisis en trabajos futuros.

En términos generales, pese a que la solución propuesta es aún concebida como un prototipo, su evaluación cuantitativa ha sido muy satisfactoria, lo cual fue corroborado con las opiniones vertidas por los participantes en la pregunta abierta del cuestionario. De igual forma, los participantes han emitido recomendaciones importantes para la mejora de la solución, sobre todo relacionadas con la usabilidad del configurador web de autoconsciencia (p. ej., hacer más intuitivo el mapeo entre los parámetros de los recursos de implementación y las métricas), mismas que se están incorporando paulatinamente.

Adicionalmente, como trabajo futuro se tiene previsto ejecutar nuevos experimentos o estudios de caso que contemplen la evaluación de la solución mediante la implementación de procesos de autoconsciencia tanto individuales como colectivos en otros subdominios de IoT y con la participación de la industria.

7.5 Amenazas a la validez de los cuasi experimentos

En esta sección se explican las principales amenazas a la confiabilidad de los resultados de las evaluaciones, mismas que fueron mitigadas durante el desarrollo de los cuasi experimentos. Estas amenazas están organizadas según la clasificación propuesta por Cook y Campbell [211].

7.5.1 Validez interna

En este estudio, las amenazas a la validez interna tienen relevancia, debido a que los cuasi experimentos realizados intentan obtener conclusiones sobre relaciones causales. En este sentido, las principales amenazas identificadas dentro de esta categoría fueron: la experiencia de los participantes, la forma como se les trató a los participantes durante el experimento; y, los sesgos producidos por los investigadores, el material experimental y las herramientas de soporte de la metodología Aware-IoT@Run.Time.

En cuanto a la reducción de la amenaza sobre la experiencia de los participantes, se preparó una etapa de entrenamiento en cada cuasi experimento, enfocada principalmente en el desarrollo de casos prácticos que proporcionen a los participantes un claro entendimiento sobre el proceso metodológico Aware-IoT@Run.Time y sus herramientas de soporte. A su vez, para motivar a los participantes y evitar su ausencia a las sesiones de los experimentos se manifestó que recibirían un incentivo en el examen final de la asignatura correspondiente; debiendo recalcar que la realización de los experimentos no fue objeto de evaluaciones, a fin de evitar el temor de los participantes a ser evaluados. Asimismo, los ejercicios experimentales fueron planificados para una duración de dos horas en el caso del primer cuasi experimento; y, una hora y media para el segundo cuasi experimento, con el objeto de evitar el cansancio o abandono de los participantes. Sin embargo, se permitió a los participantes finalizar los experimentos, aunque sobrepasen el tiempo planificado, con el fin de mitigar el efecto techo. Finalmente, se prohibió la comunicación entre los participantes durante los experimentos para impedir casos de plagio.

Los sesgos producidos por los investigadores y el material experimental fueron reducidos mediante experimentos pilotos, en el cual participaron investigadores expertos en el área con el propósito de detectar y corregir errores durante el desarrollo de los experimentos, como consecuencia de la presencia de ambigüedades u omisiones en el material experimental. Adicionalmente, los sesgos derivados de los defectos y problemas de usabilidad de las herramientas de la infraestructura de soporte fueron mitigados a través de las respectivas pruebas de software (unitarias, integración) y los experimentos pilotos.

7.5.2 Validez externa

Las principales amenazas a la representatividad y generalización de los resultados de las evaluaciones fueron: el diseño experimental, el contexto de los participantes seleccionados y la complejidad de las tareas experimentales.

En primer lugar, para mitigar las amenazas relacionadas con el diseño experimental se propusieron escenarios de IoT lo suficientemente completos. En el caso del primer cuasi experimento se incluyeron diferentes tipos de entidades (físicas y digitales) y tipos de comunicación (síncrona, asíncrona); mientras que, el segundo cuasi experimento contempló procesos de autoconsciencia tanto pre-reflexivos como reflexivos enfocados en comprender el estado actual o la posible evolución (predicción) de entidades (objetos) físicas y digitales, utilizando una variedad de métricas (directas, indirectas, indicadores) y recursos de implementación (funciones, servicios web). Además, los escenarios experimentales abordaron varios subdominios populares de IoT (p. ej., hogar inteligente,

salud) a lo largo de las pruebas piloto, casos prácticos de entrenamiento y ejercicios experimentales.

A su vez, para reducir las amenazas relacionadas con el contexto de los participantes, los cuasi experimentos contaron con la participación de estudiantes de pregrado de los últimos semestres de la carrera de Ingeniería de Sistemas de varias universidades. Estos estudiantes pueden ser considerados la próxima generación de profesionales [202], ya que en varios estudios se ha demostrado que, bajo ciertas condiciones, no existe una diferencia significativa entre este tipo de estudiantes y profesionales [203][204]. Por lo tanto, la capacidad de estos estudiantes para desarrollar las tareas de los cuasi experimentos es comparable con la de profesionales junior, más aún cuando sólo se han seleccionado estudiantes que hayan aprobado asignaturas que incluyan temas sobre Arquitecturas de Software, Desarrollo Dirigido por Modelos e Internet de las Cosas. Además, los estudiantes fueron entrenados en el uso de la metodología Aware-IoT@Run.Time y sus herramientas de soporte. Puntualmente, en el segundo cuasi experimento, los estudiantes recibieron una capacitación sobre Autoconsciencia Computacional.

Finalmente, para minimizar las amenazas sobre la complejidad de las tareas experimentales, se propusieron un conjunto de tareas experimentales con un alcance y nivel de dificultad acorde al tiempo disponible para cada experimento.

Como complemento, será necesario en trabajos futuros realizar nuevos experimentos o estudios de caso con la participación de profesionales de la industria, considerando escenario de evaluación de mayor complejidad y tamaño, pudiendo incluso ser en otros subdominios de IoT.

7.5.3 Validez del constructo

La principal amenaza a la validez del constructo fue hasta dónde las variables permiten medir y probar correctamente las hipótesis establecidas en los cuasi experimentos. Para mitigar esta amenaza las variables subjetivas, relacionadas con el rendimiento y percepción de los usuarios, estuvieron basadas en el Modelo de Evaluación de Métodos (MEM) [49], el cual es un modelo teórico validado empíricamente y utilizado exitosamente en la evaluación de métodos y herramientas de desarrollo en el campo de la ingeniería de software [205].

En particular, las variables de rendimiento se basaron en medidas comúnmente utilizadas para medir la efectividad y eficiencia del usuario en la ejecución de las tareas experimentales. Mientras que, las variables de percepción fueron medidas a través de un cuestionario, siendo la confiabilidad de este instrumento una amenaza a la validez del constructo. Como respuesta se aplicó la prueba de confiabilidad del alfa de Cronbach [212] a cada grupo de preguntas relacionadas con las variables de percepción. Tanto para el primer como segundo cuasi experimento, los resultados de las pruebas superaron el umbral mínimo aceptado ($\alpha = 0.70$), obteniéndose un alfa de Cronbach para la Facilidad de Uso Percibida (PEOU) de 0.808 y 0.860, Utilidad Percibida (PU) de 0.778 y 0.894; e, Intención de Uso (ITU) de 0.855 y 0.929, respectivamente. Estos resultados demuestran una alta consistencia interna e interrelación entre las preguntas de los cuestionarios.

7.5.4 Validez de la conclusión

La principal amenaza a la validez estadística de las conclusiones de los cuasi experimentos fue el tamaño de la muestra, siendo para el primer cuasi experimento de 33 participantes y para el segundo de 42 participantes, debido a que puede afectar la causalidad entre las diferentes variables. Para controlar esta amenaza y en particular el riesgo de la variación

por las diferencias individuales entre los participantes, como se mencionó con anterioridad, se seleccionó un grupo de participantes lo más homogéneo posible, en cuanto a conocimientos y experiencia. Asimismo, por medio de la etapa de entrenamiento y capacitación se buscó minimizar las diferencias entre estos.

Otra amenaza potencial tuvo relación con la elección de los métodos estadísticos, por lo que, para mitigar esta amenaza, durante el análisis de los datos recolectados en los cuasi experimentos (subsecciones 7.3.4 y 7.4.4) se utilizaron las pruebas estadísticas apropiadas teniendo en consideración la naturaleza de los datos.

7.6 Conclusiones

En este capítulo se evaluó empíricamente el proceso metodológico y las herramientas de soporte de Aware-IoT@Run.time mediante la aplicación de dos cuasi experimentos, basados en el Modelo de Evaluación de Métodos (MEM) [49], con el propósito de comprender la eficacia percibida por el usuario al utilizar solución tecnológica propuesta.

A lo largo de todo el capítulo, se ha seguido exitosamente el proceso experimental (planificación, ejecución, análisis e interpretación) propuesto por Wohlin et al. [46], pero adaptando cada actividad a los objetivos de evaluación perseguidos en los cuasi experimentos. A su vez, los modelos teóricos utilizados en las evaluaciones, los cuales son una extensión del MEM, fueron instrumentos valiosos para probar correctamente las hipótesis establecidas en los cuasi experimentos; y, por ende, responder de manera adecuada a las interrogantes particulares formuladas en las evaluaciones. Por medio de la operacionalización del MEM se definieron las variables relevantes para comprender el rendimiento y la percepción del usuario sobre el uso y la posible adopción de la solución en prácticas futuras, así como el nivel de influencia entre estas variables. Específicamente, las variables de rendimiento Eficiencia y Efectividad fueron definidas como factores de influencia para las variables de percepción Facilidad de Uso, Utilidad e Intención de Uso. Asimismo, durante todo el proceso experimental se identificaron y mitigaron satisfactoriamente las posibles amenazas a la validez de los resultados de los cuasi experimentos.

Aunque las herramientas de soporte a la metodología Aware-IoT@Run.time se encuentran en estado de prototipos, los resultados obtenidos en los cuasi experimentos son satisfactorios, llegando a las siguientes conclusiones:

- i) La alta efectividad promedio de los participantes al utilizar las herramientas de Aware-IoT@Run.time; y, en consecuencia, el proceso metodológico en el que se basan; demostrando que en su gran mayoría los participantes fueron capaces de cumplir correctamente con las tareas experimentales encomendadas en los cuasi experimentos.
- ii) A pesar de que la variable eficiencia presentó el mayor nivel de dispersión, la totalidad de los participantes completaron las tareas experimentales en un tiempo mucho menor, en comparación al tiempo que se necesitaría para diseñar, configurar e implementar los artefactos de software de monitoreo y autoconsciencia propuestos en los escenarios de evaluación, con el apoyo de métodos y herramientas de ingeniería de software tradicionales, centrados en el dominio de implementación (programación a bajo nivel).
- iii) La efectividad actual de los participantes sí determina su percepción sobre la utilidad de la solución Aware-IoT@Run.Time; mientras que, la eficiencia actual de los participantes no incide en su percepción sobre la facilidad de uso de esta solución.

- iv) La intención favorable de los participantes de utilizar la metodología y las herramientas de Aware-IoT@Run.Time para la construcción de sistemas IoT autoconscientes en proyectos futuros. Además, se comprobó que esta intención de los participantes de usar la solución Aware-IoT@Run.Time es consecuencia de que la consideran fácil de usar y útil.
- v) Las recomendaciones proporcionadas por los participantes en la pregunta abierta de los cuestionarios de percepción han sido de gran utilidad para el refinamiento de la solución Aware-IoT@Run.Time.

Como se mencionó, si bien estos resultados son alentadores en cuanto a la utilidad de la solución Aware-IoT@Run.time y su posible adopción por parte de los stakeholders en proyecto futuros de construcción de sistemas IoT autoconscientes, deben ser considerados con cierta precaución, ya que representan una validación preliminar en el contexto de estudiantes de pregrado de los últimos ciclos de la carrera de Ingeniería de Sistemas y con un nivel de complejidad de los requerimientos de monitoreo y autoconsciencia entre bajo y medio. Por lo tanto, es necesario a futuro complementarlos con otras evaluaciones, mediante experimentos o estudios de caso en contexto industriales con la participación de profesionales experimentados; y, considerando escenarios de monitorización y autoconsciencia de mayor complejidad y tamaño, incluso en otros subdominios de IoT.

Adicionalmente, debido a que las herramientas de la infraestructura de soporte se fueron construyendo incrementalmente, la totalidad de las actividades y herramientas de la metodología Aware-IoT@Run.Time se evaluaron por etapas, divididas en los dos cuasi experimentos descritos en este capítulo. De ahí que, los nuevos experimentos o estudios de caso deberán abordar la evaluación de la solución Aware-IoT@Run.time como un todo, es decir, abarcar todas las actividades y herramientas en una sola evaluación.

Capítulo 8

Conclusiones y Trabajo Futuro

En este último capítulo se presentan las conclusiones que responden a los objetivos propuestos en la tesis doctoral, así como los principales hallazgos obtenidos de este trabajo. Específicamente, en la sección 8.1 se revisa el cumplimiento de los objetivos de la investigación. Luego, en la sección 8.2 se describen las contribuciones y productos (método, artefactos y herramientas de software) obtenidos como resultado de la ejecución de las actividades planificadas en la tesis doctoral. Además, en la sección 8.3 se detalla la producción científica en revistas y congresos. Finalmente, en la sección 8.3 se discuten las futuras líneas de investigación.

8.1 Conclusiones

El objetivo general de esta tesis doctoral consiste en *proponer una metodología e infraestructura de soporte basada en modelos en tiempo de ejecución para la construcción, operación y mantenimiento de sistemas IoT autoconscientes*. La consecución de este objetivo implica el cumplimiento de cada uno de los objetivos específicos propuestos en la tesis doctoral, mismos que se analizan en lo que resta de esta sección:

1. Analizar el estado del conocimiento tecnológico sobre la autoconsciencia computacional y la Ingeniería Dirigida por Modelos en IoT y otros dominios relacionados.

En primer lugar, en el capítulo 2 se construyó un marco conceptual integral que cubre todas las áreas tecnológicas (IoT, autoconsciencia computacional, MDE - models@run.time y medición de software) que contempla la tesis doctoral. Este marco conceptual fue fundamental para sustentar teóricamente y guiar formalmente el desarrollo de la tesis doctoral. Además, permitió identificar varias directrices e insumos que a su debido tiempo se consideraron en la construcción de la solución propuesta.

En segundo lugar, en el capítulo 3 se analizó el estado del conocimiento tecnológico sobre las principales áreas tecnológicas de interés para la tesis doctoral. Específicamente, este estudio cubrió el análisis de las más importantes propuestas existentes sobre DSLs para modelar arquitecturas de IoT, dado que la arquitectura de un sistema IoT es la base fundamental para soportar sus procesos de monitoreo y autoconsciencia; y, en general, para razonar sobre el sistema tanto al momento del diseño como en el tiempo de ejecución. Además, se analizaron los estudios previos más relevantes sobre autoconsciencia computacional en IoT y otros dominios relacionados, como: sistemas ciber físicos, ciudad inteligente, computación ubicua, entre otros; abarcando trabajos teóricos y soluciones prácticas basadas en enfoques genéricos y paradigmas de MDE (models@run.time). Como resultados de este análisis se hicieron explícitas las fortalezas y limitaciones del trabajo relacionado, así como las brechas, desafíos y oportunidades de investigación, producto de la conjunción de las áreas tecnológicas que aborda la tesis doctoral. Entre las principales conclusiones a las que se llegó en el estudio del estado del arte, se tienen:

- Si bien existen importantes avances en el campo de los DSLs para diseñar

arquitecturas de IoT, estas herramientas se basan en metamodelos con un lenguaje limitado para representar los conceptos claves (entidades físicas, entidades digitales, redes, propiedades de entidades, flujos de datos entre entidades, etc.) que pueden ser necesarios para soportar los procesos de monitoreo y autoconsciencia, a nivel de las capas Edge, Fog y Cloud de un sistema IoT. Además, ante la ausencia de un metamodelo de arquitectura de IoT globalmente aceptado, se evidenció la oportunidad de crear un nuevo metamodelo alineado con la arquitectura de referencia estandarizada para IoT ISO/IEC 30141:2018.

- Aunque la autoconsciencia computacional en IoT ha sido abordada desde diversas disciplinas y comunidades (inteligencia artificial, sistemas multiagentes, computación autónoma, computación orgánica, teoría de control, sistemas autoadaptativos, MDE, entre otros), esta área de investigación se encuentra en desarrollo. Es así que, un alto porcentaje de estudios son teóricos, los cuales se han preocupado de identificar los beneficios, desafíos y recomendaciones (hojas de ruta) sobre la incorporación de la autoconsciencia en sistemas IoT y otros similares. Por el contrario, los estudios de carácter práctico se han orientado únicamente a la construcción de arquitecturas, frameworks, middlewares y algoritmos de aprendizaje automático para apoyar la autoconsciencia en este tipo de sistemas. Incluso muchos de estos estudios prácticos han limitado sus esfuerzos a proponer soluciones de autoconsciencia a medida (ad hoc) para aplicaciones particulares de IoT. Por consiguiente, se evidenció la necesidad de reorientar los esfuerzos de investigación hacia la creación de enfoques metodológicos genéricos para la construcción de sistemas IoT autoconscientes, que sean aplicables a una amplia gama de subdominios y escenarios de IoT. Asimismo, estas soluciones metodológicas deben estar soportadas por una infraestructura integral, conformada por un conjunto de herramientas que cubran adecuadamente todo el ciclo de vida de un sistema IoT, desde el momento del diseño hasta la implementación y operación.
- Los modelos en tiempo de ejecución son un paradigma de MDE prometedor para apoyar la autoconsciencia computacional en los sistemas modernos y, particularmente en IoT. Sin embargo, mediante la revisión bibliográfica se corroboró que pocos estudios utilizan explícitamente MDE y los `models@run.time` como parte de sus soluciones autoconscientes. Por lo tanto, para contribuir a reducir esta brecha de investigación y como complemento a lo señalado en el párrafo anterior, se requiere promover investigaciones que incorporen a los modelos como artefactos centrales de los enfoques metodológicos y herramientas de ingeniería de software, de tal forma que estos artefactos sean el vínculo para gestionar con un alto grado de abstracción (independiente de la plataforma), automatización y flexibilidad, los requerimientos de autoconsciencia a lo largo de todas las etapas (diseño, implementación, operación) del ciclo de vida de un sistema IoT.
- De igual manera, se constató que las propuestas sobre DSLs y metamodelos para representar las preocupaciones de autoconsciencia de un sistema informático son escasas, orientándose preferentemente a la gestión autoconsciente del rendimiento de los recursos informáticos en infraestructuras de TI. Por tal razón, surge la necesidad de crear metamodelos y DSLs basados en un vocabulario amplio, expresivo y extensible para representar la diversidad de niveles y aspectos de autoconsciencia que un sistema IoT puede requerir, producto de las características

propias de estos sistemas; por ejemplo, autorreflexión sobre entidades físicas y digitales, autoconsciencia individual o colectiva (distribuida), autoconsciencia pre-reflexiva, reflexiva o meta-reflexiva, entre otros.

- En la literatura se pueden encontrar estudios relevantes sobre autoconsciencia computacional que proveen una base sólida de principios y lineamientos, a partir de los cuales los investigadores y profesionales pueden estructurar sus trabajos y proponer nuevas soluciones tecnológicas, cuidando su formalidad, rigor científico y calidad. Entre los estudios más destacados se tienen: el framework conceptual de niveles y aspectos de autoconsciencia de Lewis et al. [20] y la arquitectura de referencia para sistemas informáticos autoconscientes LRA-M Loop de Kounev et al. [19]. Asimismo, existen numerosos estudios sobre modelos de minería de datos y aprendizaje automático orientados a soportar la autoconsciencia en sistemas IoT y otros similares. De ahí que, los nuevos enfoques metodológicos y herramientas de ingeniería de software deben facilitar la reutilización de estos artefactos mediante mecanismos de abstracción de alto nivel, ocultando sus detalles de implementación.
 - Por último, aun cuando en el trabajo relacionado se han empleado una diversidad de métodos (pruebas de concepto, experimentos, estudios de caso) para evaluar empíricamente las soluciones propuestas, pocos son los estudios que llevado a cabo una evaluación rigurosa de la percepción del usuario sobre la facilidad de uso, utilidad e intención de uso futuro de las propuestas de DSLs para modelar arquitecturas de IoT; y, prácticamente nulos los estudios sobre autoconsciencia computacional en IoT y otros dominios relacionados que han empleado este tipo de evaluación. Sin embargo, orientar la evaluación hacia estos aspectos resulta fundamental para determinar la posible adopción de una propuesta tecnológica por parte de la industria de software.
2. ***Definir un proceso metodológico sistemático, disciplinado y repetible, basado en modelos en tiempo de ejecución, que integre las actividades, recursos, artefactos y herramientas necesarias para la construcción, operación y mantenimiento de sistemas IoT autoconscientes.***

En el capítulo 4 se ha definido la metodología Aware-IoT@Run.Time que utiliza la Ingeniería Dirigida por Modelos y, particularmente los modelos en tiempo de ejecución como artefactos centrales del proceso de construcción, operación y mantenimiento de un sistema IoT autoconsciente.

Este proceso metodológico ha sido descrito a profundidad y representado gráficamente bajo la notación SPEM, con el fin de facilitar su entendimiento y aplicación por parte de los stakeholders (expertos de dominio, desarrolladores, etc.). Es así que, la documentación de Aware-IoT@Run.Time contempla la descomposición jerárquica del proceso metodológico en actividades, tareas y subtareas, acompañado de la especificación de los roles de usuarios que deben participar en las actividades, las herramientas que soportan el proceso; y, los artefactos de entrada y salida de cada actividad y tarea.

Cabe recalcar que el proceso metodológico de Aware-IoT@Run.Time y sus actividades se fundamentan en el bucle de aprendizaje y razonamiento basado en modelos LRA-M loop [19], lo cual le da formalidad y rigor científico a esta propuesta. Las dos primeras actividades se centran en el diseño de los modelos de arquitectura y autoconsciencia del sistema IoT, mediante lenguajes de alto nivel basados en

conceptos del dominio del problema (p. ej. entidades físicas y digitales de IoT, flujos de datos, sujetos, objetos, aspectos y procesos de autoconsciencia, métricas de calidad). Mientras que, las restantes actividades mantienen la relación causal entre los modelos en tiempo de ejecución y el sistema IoT, y viceversa. Por un lado, conforme a las especificaciones de los modelos y sus cambios en el tiempo, se implementan y/o reconfiguran automáticamente a nivel de software, los correspondientes artefactos de monitoreo y autoconsciencia del sistema IoT. Por otro lado, una vez que estos artefactos entran en operación, se encargan de actualizar continuamente los parámetros (métricas, metadatos, etc.) especificados en los modelos, de manera que reflejen en todo momento el estado y la posible evolución del sistema IoT; y, a través de estos, el sistema pueda determinar las respectivas recomendaciones prescriptivas incluidas en los modelos.

El uso de los modelos a lo largo de todo el proceso metodológico de Aware-IoT@Run.Time, siendo el medio para gestionar los requerimientos de monitoreo y autoconsciencia tanto al momento del diseño como en el tiempo de ejecución de un sistema IoT, ha permitido definir un enfoque metodológico apropiado para hacer frente a los problemas que atañen al desarrollo de software para este tipo de sistemas, producto de las características intrínsecas en las que operan (distribuido, heterogéneo, dinámico, escalable). Entre las principales fortalezas de este enfoque se tienen: i) abstrae la complejidad e incrementa el nivel de automatización del proceso de desarrollo de sistemas IoT; ii) mejora la gestión de la evolución y cambio de los sistemas IoT; iii) mejora la sincronización entre los artefactos de diseño e implementación de los sistemas IoT; iv) utiliza modelos con especificaciones de alto nivel, independientes de cualquier dominio o plataforma tecnológica de IoT; v) minimiza los detalles de implementación (programación a bajo nivel), y, vi) mejora la capacidad autónoma y resiliente de los sistemas IoT frente a las situaciones imprevistas.

Finalmente, el método Aware-IoT@Run.Time ha pasado por un círculo de mejora continua a través de varias evaluaciones empíricas (estudios de caso, cuasi experimentos), en procura de alcanzar un proceso sistemático, ordenado, fácil de usar, eficaz, eficiente y repetible.

3. *Construir la infraestructura de soporte a la metodología, bajo una plataforma tecnológica que se adapte a las características intrínsecas de IoT (distribuido, heterogéneo, dinámico y escalable).*

En el capítulo 5 se ha diseñado la arquitectura general de la infraestructura de soporte a la metodología Aware-IoT@Run.Time, la cual contempla un conjunto de componentes (artefactos y herramientas de software) necesarios para cubrir todas las actividades y tareas de la metodología y, en sí integrar, automatizar y simplificar todo el proceso de construcción y mantenimiento de sistemas IoT autoconscientes con el apoyo de herramientas de ingeniería de software.

Específicamente, se han implementado las herramientas DSL Monitor-IoT y configurador web de capacidades de autoconsciencia que apoyan a los stakeholders en la construcción de modelos con especificaciones (conceptos) de alto nivel sobre las preocupaciones arquitectónicas, de monitoreo y autoconsciencia de un sistema IoT. Estas herramientas se basan en metamodelos construidos a partir de estándares, frameworks conceptuales y ontologías relevantes sobre arquitecturas de IoT, autoconsciencia computacional y medición de la calidad de software [2][20][106], lo

cual igualmente le otorga formalidad y rigor científico a la solución propuesta. Asimismo, estos metamodelos poseen un vocabulario amplio, expresivos y flexible, posibilitando que los stakeholders puedan modelar o especificar una diversidad de entidades IoT (físicas y digitales), flujos de datos de monitoreo; y, aspectos y procesos de autoconsciencia, a nivel de las capas Edge, Fog y Cloud de una infraestructura de IoT.

Como complemento, también se ha construido un prototipo avanzado de middleware para mantener la relación causal entre los modelos de arquitectura - autoconsciencia y el sistema IoT gestionado, y viceversa. Este middleware ante cualquier cambio en las especificaciones de los modelos, implementa y/o reconfigura de forma automática y distribuida los artefactos de software de monitoreo y autoconsciencia de un sistema IoT, a nivel de las capas Edge, Fog y Cloud. Por su parte, los artefactos generados por el middleware se encargan de proveer el soporte necesario para que un sistema IoT pueda aprender y razonar a partir de la actualización y calibración continua de los parámetros (métricas, metadatos, etc.) de los modelos, volviéndose capaz de comprender su estado y posible evolución para llegar a recomendaciones (acciones correctivas) de ser necesario.

En esta tesis doctoral, el middleware ha sido construido bajo Node.js, un entorno de tiempo de ejecución con características apropiadas (multiplataforma, metaprogramación reflexiva, programación asíncrona) para desplegar aplicaciones altamente distribuidas, heterogéneas, dinámicas y escalables, semejantes a los sistemas IoT. Sin embargo, el hecho de que la solución metodológica está basada en paradigmas de MDE (models@run.time), eleva su nivel de independencia del dominio de implementación, siendo factible replicarla rápidamente en cualquier otra plataforma, mediante la implementación de un nuevo middleware que transforme las especificaciones de los modelos en artefactos de software de monitoreo y autoconsciencia bajo la plataforma destino. Para facilitar este cometido, en el capítulo 5 se describió detalladamente la arquitectura del middleware y los requerimientos funcionales y no funcionales de sus componentes, consistiendo una importante guía para aquellos desarrolladores que deseen implementar la solución en otras plataformas. Incluso, la arquitectura del middleware es tan genérica que puede ser reutilizada para la implementación distribuida de artefactos de software que soporten otras incumbencias de los sistemas IoT (p. ej., actuación, seguridad).

Por otro lado, el middleware se basa en el patrón de diseño Pub/Sub, lo cual simplifica la comunicación y reduce el intercambio de datos requerido para mantener la relación causal entre los modelos y todos los nodos ubicados en las capas Edge, Fog y Cloud de un sistema IoT, sobre todo en sistemas altamente escalables, conformados por una gran cantidad de nodos IoT. Además, este patrón de diseño ha permitido implementar artefactos de software de monitoreo y autoconsciencia simples y ligeros, volviéndose viable el despliegue de una autoconsciencia distribuida y colectiva a lo largo de las capas Edge, Fog y Cloud de una infraestructura de IoT, en especial en aquellos nodos con capacidades limitadas de hardware, energía y ancho de banda.

Por último, como parte de la infraestructura también se ha construido una herramienta de visualización, a nivel de front-end, la cual presenta reportes con la información resultante de los procesos de monitoreo y autoconsciencia del sistema IoT, en lo referente, al estado actual o la posible evolución de sí mismo y su entorno, acompañado de las correspondientes recomendaciones prescriptivas.

4. *Utilizar la metodología y las herramientas de la infraestructura de soporte en un estudio de caso en IoT, a fin de demostrar la aplicabilidad de la solución tecnológica para la construcción, operación y mantenimiento de sistemas IoT autoconscientes.*

A lo largo del capítulo 6 se ha aplicado la metodología Aware-IoT@Run.Time y su infraestructura de soporte a un estudio de caso bajo el subdominio de ambientes de vida asistida, que tuvo como propósito implementar, poner en operación y mantener un sistema IoT con capacidades de monitoreo y autoconsciencia para gestionar las emergencias de los adultos mayores que viven solos en su hogar. En este capítulo, siguiendo paso a paso la guía metodológica y con el apoyo de las herramientas de soporte se ha demostrado que es factible construir sistemas IoT autoconscientes con un alto grado de abstracción, expresividad y flexibilidad.

Específicamente, el estudio de caso abarcó una amplia variedad de requerimientos de monitoreo y autoconsciencia relacionados con el control ambiental (temperatura, humedad, monóxido de carbono) de una casa. Estos requerimientos incluyeron diferentes tipos de entidades IoT (físicas, digitales), sensores (físicos, virtuales), flujos de datos (síncrono, asíncrono), procesos de autoconsciencia (pre-reflexivos, reflexivos), aspectos de autoconsciencia (estado, predicción), métricas; y, recursos de implementación (funciones, servicios web).

Haciendo un balance de la aplicación de la solución metodológica al estudio de caso, los resultados son muy favorables, entre los cuales se destacan:

- El proceso metodológico de Aware-IoT@Run.time ha orientado de manera formal, disciplinada y ágil al equipo de desarrollo en la implementación y mantenimiento de los requerimientos de monitoreo y autoconsciencia del estudio de caso.
- Las herramientas de la infraestructura han permitido que el equipo de desarrollo se concentre fundamentalmente en el dominio del problema (especificaciones de alto nivel), minimizando los detalles de implementación (programación a bajo nivel).
- Los metamodelos Monitor-IoT y Aware-IoT poseen un lenguaje completo para especificar la diversidad de requerimientos de monitoreo y autoconsciencia establecidos en el estudio de caso.
- El middleware de la infraestructura ha mantenido correctamente la relación causal entre los modelos y el sistema IoT observado, y viceversa. Es así que, por una parte, el middleware ha implementado y mantenido de manera exitosa los artefactos de software de monitoreo y autoconsciencia, conforme a las especificaciones dinámicas de los modelos; y, por otra parte, los artefactos de software han actualizado, estimado y calibrado correctamente las métricas, metadatos y otros parámetros especificados en los modelos, reflejando consistentemente en todo momento el estado y la posible evolución del sistema observado; y, obteniendo oportunamente las recomendaciones prescriptivas incluidas en los modelos.

5. *Evaluar empíricamente la metodología y su infraestructura de soporte en un subdominio de IoT, mediante el empleo de cuasi experimentos. Esta evaluación permitirá recolectar evidencia suficiente para comprender el rendimiento (eficiencia, efectividad) de los usuarios al utilizar la solución tecnológica en la*

construcción de sistemas IoT, así como su percepción sobre la facilidad de uso, utilidad e intención de uso futuro de la misma.

Como complemento, en el capítulo 7 se ha evaluado empíricamente la metodología Aware-IoT@Run.Time y sus herramientas de soporte mediante la ejecución de dos cuasi experimentos basados en el proceso experimental de Wohlin et al. [46] y el Modelo de Evaluación de Métodos (MEM) de Moody [49]. Tras la aplicación de estos cuasi experimentos se ha determinado la alta efectividad y eficiencia de los participantes al utilizar la solución Aware-IoT@Run.Time en la construcción de sistemas IoT autoconscientes. Además, se ha demostrado que los participantes perciben a Aware-IoT@Run.time como fácil de usar y útil, por lo que en su gran mayoría tienen la intención de usarla en proyectos futuros que requieran implementar sistemas IoT autoconscientes.

El contexto de los cuasi experimentos estuvo delimitado por: i) el proceso metodológico de Aware-IoT@Run.Time y sus herramientas de soporte; ii) el escenario de IoT utilizado para evaluar la solución, el cual se encuentra dentro del subdominio ambientes de vida asistida y consiste en la construcción de un sistema IoT con capacidades de monitoreo y autoconsciencia para gestionar las emergencias de los adultos mayores que viven solos en su hogar; y, iii) la participación de estudiantes de los últimos semestres de las carreras de Ingeniería de Sistemas de la Universidad del Azuay y la Universidad de Cuenca. Bajo este contexto, se presentaron varias amenazas a la validez de los resultados de la evaluación, las cuales fueron mitigadas oportunamente durante el diseño y ejecución de los cuasi experimentos, cuyos pormenores se presentan en el capítulo 7.

Como conclusión final, la aplicación del estudio de caso y la ejecución de los cuasi experimentos ha dejado resultados muy alentadores sobre la usabilidad, utilidad y futura adopción de la solución Aware-IoT@Run.Time para la construcción de sistemas IoT. Además, estas evaluaciones han servido para corroborar que un enfoque de ingeniería de software basado en la autoconsciencia computacional y los modelos en tiempo de ejecución resulta ser muy apropiado para abstraer la complejidad y elevar el nivel de automatización del desarrollo de los sistemas IoT, mejorar la gestión de su evolución y cambio; y, en definitiva, reducir el esfuerzo y tiempo requerido para su puesta en operación. Sin embargo, este es un primer estudio en el contexto de una validación académica que ha servido para instaurar un círculo de mejora continua de la solución. De ahí que, según el Modelo para la Transferencia de Tecnología de Gorschek et al. [37], en un futuro será necesario cubrir las dos últimas actividades de este modelo (validación realista y liberación de la solución), mediante la aplicación de estudios de caso y/o experimentos controlados en entornos industriales que contemplen proyectos con un mayor nivel de complejidad de los requerimientos de monitoreo y autoconsciencia, así como con la participación de profesionales y expertos de dominio. Para cerrar este apartado, las evaluaciones también han evidenciado una serie de nuevas oportunidades de investigación para perfeccionar la solución Aware-IoT@Run.Time, mismas que se tratan más adelante en la sección 8.3 (Trabajo futuro).

8.2 Resumen de las principales contribuciones

La presente tesis doctoral ha contribuido con una solución tecnológica que combina la autoconsciencia computacional y la Ingeniería Dirigida por Modelos (models@run.time) para hacer frente a los problemas y desafíos que atañen al desarrollo de software para

sistemas IoT, producto de los escenarios altamente distribuidos, heterogéneos, escalables, e inciertos en los que operan. Entre las principales contribuciones se destacan las siguientes:

1. **Estado del arte sobre la autoconsciencia computacional en IoT y otros dominios relacionados.** A pesar de que la investigación sobre autoconsciencia computacional se encuentra fragmentada en diferentes disciplinas y comunidades (inteligencia artificial, sistemas multiagentes, computación autónoma, computación orgánica, teoría de control, sistemas autoadaptativos, MDE, entre otros), se ha construido un estado del arte completo y actualizado sobre la autoconsciencia computacional en IoT y otros dominios relacionados, tales como: sistemas ciber físicos, ciudad inteligente y computación ubicua. El estado del arte sintetiza los más relevantes estudios teóricos y prácticos sobre esta área de investigación, haciendo énfasis sobre todo en aquellos estudios que emplean los paradigmas de MDE (models@run.time). Además, este estudio evidencia las fortalezas y limitaciones del trabajo relacionado, así como las brechas, desafíos y oportunidades de investigación, trazando una hoja de ruta para que la comunidad científica pueda direccionar apropiadamente sus proyectos de investigación futuros.
2. **Metodología Aware-IoT@Run.Time.** Se ha propuesto una metodología basada en modelos en tiempo de ejecución integrada por un conjunto de actividades, tareas, subtarear, guías, roles de usuario, artefactos y herramientas que indican cómo construir, operar y mantener un sistema IoT autoconsciente. En particular, los sistemas IoT construidos con esta metodología, utilizan los modelos en tiempo de ejecución para comprender su estado y posible evolución, así como la de su entorno; y, determinar las respectivas recomendaciones o acciones correctivas de ser necesario. Para ello, Aware-IoT@Run.Time está compuesta por seis actividades principales: i) Diseño de la arquitectura de monitoreo del sistema IoT; ii) Configuración de las capacidades de autoconsciencia del sistema IoT; iii) Sincronización entre los modelos de arquitectura - autoconsciencia y el sistema IoT observado (mantiene la relación causal); iv) Monitoreo; v) Aprendizaje; y, vi) Razonamiento. En definitiva, Aware-IoT@Run.Time es un enfoque metodológico genérico que simplifica y agiliza la construcción de sistemas autoconscientes para cualquier subdominio IoT.
3. **Infraestructura de soporte a la metodología Aware-IoT@Run.Time.** De igual manera, se ha diseñado la arquitectura general de la infraestructura de soporte y de cada uno de sus componentes. Además, se ha construido un prototipo avanzado de dicha infraestructura, que viabiliza la construcción, operación y mantenimiento de sistemas IoT autoconscientes, con un alto grado de abstracción (minimiza los detalles implementación), automatización y flexibilidad. La infraestructura de soporte está compuesta por varios artefactos y herramientas, mismos que se describen a continuación, haciendo énfasis en sus fortalezas:
 - 3.1. **DSL y metamodelo Monitor-IoT.** Lenguaje específico de dominio visual de alto nivel para diseñar arquitecturas de sistemas IoT que soporten el monitoreo, a través de las capas de Edge, Fog y Cloud. Monitor-IoT está conformado por un diseñador gráfico (sintaxis concreta) construido en las herramientas Obeo Designer Community y Eclipse Sirius, que simplifica y agiliza las tareas de creación y mantenimiento de modelos de arquitecturas de IoT. Además, Monitor-IoT está basado en un metamodelo (sintaxis abstracta) especificado en Ecore y alineado con la arquitectura de referencia para IoT ISO/IEC 30141:2018 [2], proporcionando un lenguaje más

completo capaz de modelar una amplia variedad de entidades digitales y flujos de datos (síncronos y asíncronos) entre estas, a nivel de las capas Edge, Fog y Cloud; y, de esta forma soportar el monitoreo en una diversidad de escenarios de IoT. Principalmente, Monitor-IoT provee un vocabulario para definir: i) las entidades físicas y digitales a ser monitoreadas; ii) las entidades digitales (nodos de computación y sus recursos) que soportan los procesos de monitoreo (recolección, transporte, procesamiento y almacenamiento de datos), a nivel de las capas Edge, Fog y Cloud; iii) las propiedades a monitorear para cada entidad IoT; iv) los flujos de datos entre las entidades digitales, basados en comunicaciones síncronas o asíncronas; y, v) las operaciones de agregación a ser aplicadas sobre los datos recolectados.

3.2. Configurador web de capacidades de autoconsciencia para sistemas IoT y metamodelo Aware-IoT. Aplicación web desarrollada en Node.js para construir modelos con especificaciones de alto nivel sobre una amplia variedad de niveles y aspectos de autoconsciencia para sistemas IoT. Estas especificaciones son expresadas mediante conceptos abstractos del dominio de la autoconsciencia computacional; y, complementados con conceptos sobre medición de la calidad de software. Para ello, el configurador está basado en el metamodelo Aware-IoT, construido a partir del framework conceptual de autoconsciencia de Lewis et al. [20] y la ontología de medición de software de García et al. [106]; e, integrado con el metamodelo Monitor-IoT. Específicamente, el configurador web permite definir: i) el dominio (sujeto, objeto) de autoconsciencia de un sistema IoT; ii) los aspectos de autoconsciencia a capturar de las entidades IoT, pudiendo ser individuales o colectivos; iii) los procesos de autoconsciencia pre-reflexivos y reflexivos, junto con sus métodos de autoconsciencia (recolección, cálculo y modelos de análisis) y recursos de implementación (fórmulas, funciones, servicios web); iv) las métricas (directas, indirectas e indicadores) para evaluar los aspectos de autoconsciencia; y, v) los criterios de decisión, umbrales y recomendaciones prescriptivas asociados a los modelos de análisis. Por último, el configurador web posibilita la integración y serialización de los modelos de arquitectura y autoconsciencia en un único archivo JSON, conforme a los metamodelos Monitor-IoT y Aware-IoT, pudiendo ser interpretados y procesados por un computador.

3.3. Middleware de la infraestructura de soporte. Componente de software esencial para mantener la relación causal entre los modelos de arquitectura - autoconsciencia y el sistema IoT gestionado, así como para elevar el nivel de automatización del proceso de desarrollo y mantenimiento de este tipo de sistemas. Este middleware se construyó en Node.js, un entorno de tiempo de ejecución ideal para desplegar aplicaciones IoT altamente distribuidas, heterogéneas, dinámicas y escalables. Además, Node.js es apropiado para soportar aplicaciones basadas en *models@run.time* que deben ser capaces de crear o modificar su propio código según las especificaciones dinámicas contenidas en los modelos. El middleware está conformado por los motores de sincronización de recursos de monitoreo y autoconsciencia. El primer motor implementa en tiempo de ejecución las APIs, aplicaciones, servicios web, tópicos de brokers y bases de datos que automatizan la recolección, transporte, procesamiento y almacenamiento de los datos crudos de los sensores u otras fuentes. Mientras que, el segundo motor implementa los demonios (aplicaciones que se ejecutan cada cierto intervalo de tiempo) que encapsulan y ejecutan de manera distribuida y en segundo plano los procesos pre-reflexivos y reflexivos, encargados de calcular las métricas de evaluación de los aspectos de autoconsciencia. Estos artefactos de software creados

por los motores hacen posible que un sistema IoT pueda aprender y razonar en tiempo de ejecución mediante la parametrización de modelos, es decir, la actualización y calibración permanente de los parámetros (métricas, metadatos, etc.) especificados en los modelos, de tal forma que estos modelos actualizados se conviertan en una autorrepresentación del sistema IoT observado.

3.4. Visor web de métricas de autoconsciencia. Aplicación web a nivel de front-end de la infraestructura de soporte que visualiza las métricas resultantes de los procesos de autoconsciencia pre-reflexivos y reflexivos ejecutados por el sistema IoT. En particular, el visor web presenta: i) reportes (tabular y gráfico) en tiempo real sobre el estado de los aspectos de autoconsciencia observados, incluyendo la valoración cuantitativa, cualitativa e interpretación de las mismas, así como las recomendaciones o acciones prescriptivas, en caso de existir; e, i) informes predictivos (simulación) sobre la posible evolución de un aspecto de autoconsciencia, considerando los escenarios, variables y valores de simulación disponibles en el modelo de autoconsciencia.

4. Evaluación empírica de la solución tecnológica. Se han diseñado dos cuasi experimentos para evaluar la metodología Aware-IoT@Run.time y sus herramientas de soporte, los cuales siguen el proceso experimental de Wohlin et al. [46]. A su vez, estos cuasi experimentos utilizan modelos teóricos adaptados del Modelo de Evaluación de Métodos (MEM) [49] que proporcionan un sustento conceptual sólido para comprobar el rendimiento (eficacia, eficiencia) del usuario al utilizar esta solución Aware-IoT@Run.time, así como su percepción sobre la usabilidad, utilidad y posible adopción de la misma en proyectos futuros de construcción de sistemas IoT autoconscientes. No obstante, los procedimientos, modelos teóricos, instrumentos y materiales de evaluación diseñados en la tesis doctoral pueden ser tomados como base para diseñar nuevos experimentos que permitan: i) evaluar otros enfoques metodológicos o herramientas de ingeniería de software para predecir su aceptación en la práctica; o, ii) comparar dos o más enfoques metodológicos o herramientas de ingeniería de software para determinar el más eficaz, eficiente, fácil de usar, útil y probable de ser adoptado en la práctica.

8.3 Difusión de resultados (publicaciones)

Durante el desarrollo de la tesis doctoral se han divulgado y comunicado los resultados parciales a través de varias publicaciones en revistas y conferencias con referato. Entre las publicaciones se destaca el artículo publicado en la Revista IEEE Access, catalogada con factor de impacto Q1, según el Scimago Journal & Country Rank (SJR). Otro artículo se publicó en el CIBSE 2022, considerada la conferencia más relevante de Iberoamérica en ingeniería de software. Asimismo, otros trabajos relacionados se han publicado en varias ediciones de conferencias internacionales organizadas por importantes universidades del Ecuador y la Corporación Ecuatoriana para el Desarrollo de la Investigación y Academia (CEDIA). En este punto se resalta una publicación que recibió el reconocimiento al *Mejor Artículo (Best Paper)* en el Track Científico de Desarrollo de Software del TICEC 2023, la conferencia emblemática de Tecnologías de Información y Comunicación del Ecuador. A continuación, se describen de forma detallada estas publicaciones y otras adicionales:

1. L. Erazo-Garzón, P. Cedillo, G. Rossi, and J. Moyano, "A Domain-Specific Language for Modeling IoT System Architectures That Support Monitoring," *IEEE Access*, vol. 10, pp. 61639-61665, Jun. 2022, **SJR Q1, Impact Factor 3.9 (JCR 2022)**, doi:

- <https://doi.org/10.1109/ACCESS.2022.3181166>.
2. L. Erazo-Garzón, "Metodología Basada en Modelos en Tiempo de Ejecución para la Construcción y Operación de Sistemas Autoconscientes de Internet de las Cosas," in *XXV Ibero-American Conference on Software Engineering (CIBSE 2022), Doctoral Symposium*, Córdoba, Argentina, Jun. 2022, pp. 392-399, doi: <https://doi.org/10.5753/cibse.2022.20989>.
 3. L. Erazo-Garzón, A. Román, J. Moyano, and P. Cedillo, "Models@runtime and Internet of Things: A Systematic Literature Review," in *2nd International Conference on Information Systems and Software Technologies (ICI2ST 2021)*, Quito, Ecuador, Mar. 2021, IEEE, pp. 128-134, doi: <https://doi.org/10.1109/ICI2ST51859.2021.00026>.

Publicaciones relacionadas:

4. L. Erazo-Garzón, K. Quinde, A. Bermeo, and P. Cedillo, "A Domain-Specific Language and Model-Based Engine for Implementing IoT Dashboard Web Applications," in *11th Ecuadorian Conference on Information and Communication Technologies (TICEC 2023)*, Cuenca, Ecuador, Oct. 2023, CCIS 1885, Springer, pp. 412-428, *Mejor Artículo (Best Paper) en el Track Científico de Desarrollo de Software*, doi: https://doi.org/10.1007/978-3-031-45438-7_28.
5. L. Erazo-Garzón, J. Avila, S. Pinos, and P. Cedillo, "A Systematic Review on the Use of Ontologies in the Internet of Things," in *3rd International Conference on Applied Technologies (ICAT 2021)*, Quito, Ecuador, Oct. 2021, CCIS 1535, Springer, pp. 509-524, doi: https://doi.org/10.1007/978-3-031-03884-6_37.
6. W. Valdez, P. Cedillo, K. Chávez-Z, S. Espinoza-A, A. Barzallo, L. Erazo-Garzón, "Towards the Evaluation of the Performance Efficiency of Fog Computing Applications," in *2021 Future of Information and Communication Conference (FICC 2021)*, Vancouver, Canada, Apr. 2021, AISC 1363, Springer, pp. 901-916, doi: https://doi.org/10.1007/978-3-030-73100-7_62.
7. L. Erazo-Garzón, L. Illescas-Peña, and P. Cedillo, "A Quality in Use Model for Ambient Assisted Living (AAL) Systems," in *2nd International Conference on Applied Technologies (ICAT 2020)*, Quito, Ecuador, Dec. 2020, CCIS 1388, Springer, pp. 643-660, doi: https://doi.org/10.1007/978-3-030-71503-8_50.
8. L. Erazo-Garzón, J. Erraez, L. Illescas-Peña, and P. Cedillo, "Quality Assessment Approaches for Ambient Assisted Living Systems: A Systematic Review," in *1st International Conference on Applied Technologies (ICAT 2019)*, Quito, Ecuador, Dec. 2019, CCIS 1193, Springer, pp. 421-439, doi: https://doi.org/10.1007/978-3-030-42517-3_32.
9. L. Erazo-Garzón, J. Erraez, L. Illescas-Peña, and P. Cedillo, "A Data Quality Model for AAL Systems," in *7th Ecuadorian Conference on Information and Communication Technologies (TICEC 2019)*, Cuenca, Ecuador, Nov. 2019, AISC 1099, Springer, pp. 137-152, doi: https://doi.org/10.1007/978-3-030-35740-5_10.
10. L. Erazo-Garzón, A. Patiño, P. Cedillo, and A. Bermeo, "CALMS: A Context-Aware Learning Mobile System Based on Ontologies," in *6th International Conference on eDemocracy & eGovernment (ICEDEG 2019)*, Quito, Ecuador, Apr. 2019, IEEE, pp. 84-91, doi: <https://doi.org/10.1109/ICEDEG.2019.8734423>.

8.4 Trabajo futuro

A partir del desarrollo de la tesis doctoral han surgido importantes oportunidades de investigación que, si bien están fuera de los objetivos y alcance de esta tesis, tienen como

finalidad mejorar y ampliar la solución metodológica Aware-IoT@Run.Time. A continuación, se resumen las principales líneas de trabajo que se plantean abordar en proyectos futuros.

- Continuar con las tareas de perfeccionamiento de la solución Aware-IoT@Run.Time, tendientes a optimizar el proceso metodológico y mejorar la usabilidad de las herramientas de soporte con base en los hallazgos y recomendaciones obtenidos de las evaluaciones.
- Completar la construcción de las funcionalidades y componentes de la infraestructura de soporte, especialmente en lo relacionado a los aspectos de autoconsciencia colectivos. En el configurador web de autoconsciencia implementar el formulario de definición de la estructura jerárquica de los aspectos colectivos. Mientras que, a nivel del middleware implementar el módulo LSP para evaluar los aspectos colectivos y el intérprete de fórmulas para operacionalizar los métodos de cálculo o modelos de análisis.
- Construir un Lenguaje de Modelado Específico de Dominio Visual (VDSML) basado en los metamodelos Aware-IoT y Monitor-IoT para diseñar gráficamente el dominio y las capacidades de autoconsciencia de un sistema IoT. Además, será necesario realizar un análisis comparativo entre este VDSML y el configurador web de autoconsciencia, a fin de determinar las fortalezas y debilidades de estas herramientas, en términos de usabilidad, funcionalidad, productividad y reutilización de componentes.
- La solución Aware-IoT@Run.Time se enmarca en el aprendizaje y razonamiento de los sistemas IoT mediante la parametrización de los modelos en tiempo de ejecución. Sin embargo, uno de los desafíos a explorar a futuro es el aprendizaje y razonamiento de estos sistemas fundamentado en la estructura de los modelos. Para ello, se requiere de sistemas con capacidades de automodelado, los cuales deben estar habilitados para construir y mantener el modelado de sí mismo y su entorno durante el tiempo de ejecución [193]. En particular, debido a la falta de herramientas de ingeniería de software en este ámbito [154], las nuevas investigaciones deberán concentrarse en construir e integrar nuevos componentes a la infraestructura de soporte de Aware-IoT@Run.Time, encargados principalmente de: i) descubrir en tiempo real nuevas entidades IoT (físicas, digitales) y sus propiedades, o detectar cambios en estas; y, ii) actuar como modeladores no humanos, actualizando en tiempo de ejecución los modelos a nivel estructural (p. ej., agregar nuevas entidades IoT, propiedades, metadatos, flujos de datos, procesos de autoconsciencia, métricas, entre otros), de tal forma que reflejen los cambios que ocurren en el propio sistema y su entorno.
- Ampliar el alcance de la solución tecnológica para que adicional a los procesos de monitoreo y autoconsciencia, soporte la implementación de los procesos de actuación de un sistema IoT. De esta forma, el conocimiento cuantitativo y cualitativo sobre el estado y la posible evolución de un sistema IoT y su entorno, obtenido a partir de sus procesos de monitoreo y autoconsciencia, más allá de servir para llegar a recomendaciones prescriptivas, permitirá disparar automáticamente procesos de adaptación en tiempo real, otorgándole al sistema IoT propiedades tanto de autoconsciencia como autoadaptación. En este punto, la funcionalidad genérica del middleware de la infraestructura será valiosa para implementar los artefactos de software que soportan los procesos de actuación del sistema IoT, conforme a modelos de autoadaptación que deberán ser construidos.

- Paralelo a la implementación de las funcionalidades y componentes antes citados se deberán proponer y ejecutar estudios de caso y experimentos ampliados, con un mayor alcance y complejidad, incorporando requerimientos, tales como: aspectos de autoconsciencia colectivos, recursos de implementación de tipo fórmula, sensores utilizando otras tecnologías o protocolos de comunicación (p. ej., Wifi, Bluetooth), capacidades de automodelado, entre otros. Asimismo, estas evaluaciones empíricas deberán desarrollarse bajo contextos industriales, con la participación de profesionales de la ingeniería de software y expertos del dominio de IoT, considerando inclusive otros subdominios de aplicación. Además, los nuevos estudios de caso o experimentos deberán ser integrales, abordando todas las actividades y herramientas de soporte de Aware-IoT@Run.Time en una sola evaluación.
- Por último, este trabajo de investigación ha generado productos innovadores, respaldados por una documentación científica - técnica completa y original que deberá seguir siendo difundida y comunicada a través de prestigiosas revistas y conferencias internacionales, a fin de contribuir a dinamizar la investigación en las áreas tecnológicas que cubre esta tesis doctoral.

Bibliografía

- [1] S. Madakam, R. Ramaswamy, and S. Tripathi, “Internet of Things (IoT): A literature review,” *Journal of Computer and Communications*, vol. 3, pp. 164–173, 2015.
- [2] *Internet of Things (IoT)—Reference Architecture, Standard ISO/IEC 30141:2018*. ISO, Aug. 2018. [Online]. Available: <https://www.iso.org/standard/65695.html>
- [3] *Overview of the Internet of Things, Recommendation ITU-T Y.2060*. ITU-T Study Group 13, Jun. 2012. [Online]. Available: <https://www.itu.int/rec/T-REC-Y.2060-201206-I>
- [4] Y. Perwej, K. Haq, F. Parwej, and M. M. Mohamed Hassan, “The Internet of Things (IoT) and its application domains,” *International Journal of Computer Applications*, vol. 182, no. 49, pp. 36–49, 2019.
- [5] A. A. Laghari, K. Wu, R. A. Laghari, M. Ali, and A. A. Khan, “A Review and state of art of Internet of Things (IoT),” *Archives of Computational Methods in Engineering*, vol. 29, pp. 1395–1413, 2022.
- [6] A. Khanna and S. Kaur, “Internet of Things (IoT), applications and challenges: A comprehensive review,” *Wireless Personal Communications*, vol. 114, pp. 1687–1762, 2020.
- [7] R. Hassan, F. Qamar, M. K. Hasan, A. H. M. Aman, and A. S. Ahmed, “Internet of things and its applications: A comprehensive survey,” *Symmetry*, vol. 12, no. 10, 2020.
- [8] R. van der Meulen. *Gartner Says 8.4 Billion Connected ‘Things’ Will be in Use in 2017, up 31 Percent From 2016*. Gartner, Egham, U.K. Accessed: Feb. 7, 2017. [Online]. Available: <https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016>
- [9] E. Estopace. *IDC Forecasts Connected IoT Devices to Generate 79.4 ZB of Data in 2025*. FutureIoT. Accessed: Jun. 22, 2019. [Online]. Available: <https://futureiot.tech/idc-forecasts-connected-iot-devices-to-generate-79-4zb-of-data-in-2025>
- [10] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of Things: Vision, applications and research challenges,” *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
- [11] P. R. Lewis, “Self-aware computing systems: From psychology to engineering,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, Switzerland, 2017, pp. 1044–1049.
- [12] C. M. Sosa-Reyna, E. Tello-Leal, and D. Lara-Alabazares, “Methodology for the model-driven development of service oriented IoT applications,” *Journal of Systems Architecture*, vol. 90, pp. 15–22, 2018.
- [13] P. R. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, *Self-aware computing systems: An Engineering Approach*. Natural Computing Series. Cham, Switzerland: Springer, 2016.
- [14] X. Chen, A. Li, X. Zeng, W. Guo, and G. Huang, “Runtime model based approach to IoT application development,” *Frontiers of Computer Science*, vol. 9, no. 4, pp. 540–553, 2015.
- [15] J. Mineraud, O. Mazhelisb, X. Suc, and S. Tarkoma, “A gap analysis of Internet-of-Things platforms,” *Computer Communications*, vols. 89–90, pp. 5–16, 2016.
- [16] L. Esterle and R. Grosu, “Cyber-physical systems: challenge of the 21st century,” *Elektrotechnik & Informationstechnik*, vol. 133, no. 7, pp. 299–303, 2016.
- [17] M. Möstl, J. Schlatow, R. Ernst, H. Hoffmann, A. Merchant, and A. Shraer, “Self-aware systems for the internet-of-things,” in *Proc. 11th IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis*, Pittsburgh, PA, USA, 2016, pp. 1–9.
- [18] P. Lalanda, S. Chollet, and C. Hamon, “Leveraging design and runtime architecture models to support self-awareness,” in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 669–686.
- [19] S. Kounev *et al.*, “The notion of self-aware computing,” in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 3–16.

- [20] P. R. Lewis *et al.*, “Towards a framework for the levels and aspects of self-aware computing systems,” in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 51–85.
- [21] J. Cámara *et al.*, “Self-aware computing systems: Related concepts and research areas,” in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 17–49.
- [22] A. Agarwal, J. Miller, J. Eastep, D. Wentziaff, and H. Kasture, “Self-aware computing,” MIT, Tech. Rep. AFRL-RI-RS-TR-2009-161, 2009.
- [23] A. Elhabbash, M. Salama, R. Bahsoon, and P. Tino, “Self-awareness in Software Engineering: A systematic literature review,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 14, no. 2, pp. 1–42, 2019.
- [24] K. Bellman *et al.*, “Self-aware Cyber-Physical Systems,” *ACM Transactions on Cyber-Physical Systems*, vol. 4, no. 4, pp.1–26, 2020.
- [25] K. Tammemäe, A. Jantsch, A. Kuusik, J. S. Preden, and E. Õunapuu, “Self-aware fog computing in private and secure spheres,” in *Fog Computing in the Internet of Things*, A. Rahmani, P. Liljeberg, J. S. Preden, and A. Jantsch, Eds. Cham, Switzerland: Springer, 2017, pp. 71–99.
- [26] L. Gurgun, O. Gunalp, Y. Benazzouz, and M. Gallissot, “Self-aware cyber-physical systems and applications in smart buildings and cities,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2013, pp. 1149–1154.
- [27] L. Esterle and J. N. A. Brown, “I Think Therefore You Are: Models for Interaction in Collectives of Self-Aware Cyber-Physical Systems,” *ACM Transactions on Cyber-Physical Systems*, vol. 4, no. 4, pp.1–24, 2020.
- [28] T. Stahl and M. Volter, *Model-driven software development: Technology, Engineering, Management*. New York, NY, USA: John Wiley and Sons, 2006.
- [29] J. Bézivin, “Model driven engineerings: An emerging technical space,” in *Generative and Transformational Techniques in Software Engineering (GTTSE). Lecture Notes in Computer Science*, vol. 4143, R. Lämmel, J. Saraiva, J. Visser, Eds. Berlin, Heidelberg: Springer, 2006, pp. 36–64.
- [30] M. Szvetits and U. Zdun, “Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime,” *Software & Systems Modeling*, vol. 15, pp. 31–69, 2016.
- [31] G. Blair, N. Bencomo, and R. B. France, “Models@run.time,” *Computer*, vol. 42, no. 10, pp. 22–27, 2009.
- [32] N. Bencomo, R. France, B. H. C. Cheng, and U. Aßmann, *Models@run.time: Foundations, applications, and roadmaps (vol. 8378)*. Cham, Switzerland: Springer, 2014.
- [33] N. Bencomo, S. Götz, and H. Song, “Models@run.time: a guided tour of the state of the art and research challenges,” *Software & Systems Modeling*, vol. 18, pp. 3049–3082, 2019.
- [34] P. Maes, “Concepts and experiments in computational reflection,” *ACM SIGPLAN Notices*, vol. 22, no. 12, pp 147–155, 1987.
- [35] N. Bencomo, G. Blair, S. Götz, B. Morin, and B. Rumpe, “Report on the 7th international workshop on models@run.time,” *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 1, pp. 27–30, 2013.
- [36] L. Erazo-Garzón, “Metodología Basada en Modelos en Tiempo de Ejecución para la Construcción y Operación de Sistemas Autoconscientes de Internet de las Cosas,” in *XXV Ibero-American Conference on Software Engineering (CIBSE), Doctoral Symposium*, Córdoba, Argentina, 2022, pp. 392–399.
- [37] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, “A model for technology transfer in practice,” *IEEE Software*, vol. 23, no. 6, pp. 88–95, 2006.
- [38] A. J. Salman, M. Al-Jawad, and W. Al Tameemi, “Domain-specific languages for IoT: Challenges and opportunities,” *IOP Conference Series: Materials Science and Engineering*, vol. 1067, no. 1, 2021.
- [39] L. Erazo-Garzon, A. Roman, J. Moyano-Dutan, and P. Cedillo, “Models@runtime and Internet of Things: A Systematic literature review,” in *Proc. 2nd Int. Conf. on Information Systems and Software Technologies (ICI2ST)*, Quito, Ecuador, 2021, pp. 128–134.

- [40] J. Pitt, *The computer after me: Awareness and self-awareness in autonomic systems*. London, UK: Imperial College Press, 2014.
- [41] S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, *Self-aware computing systems*. Cham, Switzerland: Springer, 2017.
- [42] R. S-Julián, I. Lacalle, R. Vaño, F. Boronat, and C. E. Palau, “Self-* capabilities of cloud-edge nodes: A research review,” *Sensors*, vol. 23, no. 6, p. 2931, 2023.
- [43] A. Jantsch, N. Dutt, and A. M. Rahmani, “Self-Awareness in Systems on Chip - A Survey,” *IEEE Design Test*, vol. 34, no. 6, pp 8–26, 2017.
- [44] B. Kitchenham and S. Charters, *Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE 2007-001*, Keele University and Durham University Joint Report, 2007.
- [45] K. Schwaber and J. Sutherland. *The Scrum Guide*. ScrumGuides.org, Nov. 2020. [Online]. Available: <https://scrumguides.org/scrum-guide.html>
- [46] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Germany: Springer, 2012.
- [47] M. Genero, J. Cruz, and M. Piattini, *Métodos de Investigación en Ingeniería del Software*. Madrid, Spain: Ra-Ma, 2014.
- [48] C. Wohlin, “Introduction to aggregation of case studies why aggregation,” Sweden, 2007.
- [49] D. L. Moody, “The method evaluation model: A theoretical model for validating information systems design methods,” in *Proc. 11th European Conference on Information Systems (ECIS)*, Naples, Italy, 2003.
- [50] K. Ashton, “That 'Internet of Things' thing,” *RFID Journal*, vol. 22 no. 7, pp. 97–114, 2009.
- [51] *The Internet of Things: From theory to reality*. Forbes, 2017. [Online]. Available: <https://www.forbes.com/forbes-insights/hitachi-vantara/internet-of-things/>
- [52] D. Newman. *The Top 8 IoT Trends For 2018*. Forbes. Accessed: Dec. 19, 2017. [Online]. Available: <https://www.forbes.com/sites/danielnewman/2017/12/19/the-top-8-iot-trends-for-2018/?sh=681c7cb067f7>
- [53] J. H. Nord, A. Koohang, and J. Paliszkievicz, “The Internet of Things: Review and theoretical framework,” *Expert Systems with Applications*, vol. 133, pp. 97–108, 2019.
- [54] C. Granell, A. Kamlaris, A. Kotsev, F. O. Ostermann, and S. Trilles, “Chapter 11: Internet of Things,” in *Manual of Digital Earth*, H. Guo, M. F. Goodchild, and A. Annoni, Eds. Singapore: Springer, 2020, pp. 387–423.
- [55] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
- [56] L. Erazo-Garzón, P. Cedillo, G. Rossi, and J. Moyano, “A Domain-Specific Language for Modeling IoT System Architectures That Support Monitoring,” *IEEE Access*, vol. 10, pp. 61639–61665, 2022.
- [57] H. Kopetz and W. Steiner, *Real-time systems: Design principles for distributed embedded applications*. Cham, Switzerland: Springer, 2022.
- [58] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A survey on enabling technologies, protocols, and applications,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
- [59] M. Chiang and T. Zhang, “Fog and IoT: An overview of research opportunities,” *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864 2016.
- [60] S. Dustdar, C. Avasalcá, and I. Murturi, “Invited paper: Edge and fog computing: Vision and research challenges,” in *Proc. IEEE Int. Conf. on Service-Oriented System Engineering (SOSE)*, San Francisco, CA, USA, 2019, pp. 96–105.
- [61] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi. *Fog Computing Conceptual Model, NIST Special Publication 500-325*. National Institute of Standards and Technology (NIST), Gaithersburg, MD, Mar, 2018. [Online]. Available: <https://www.itu.int/rec/T-REC-Y.2060-201206-I>
- [62] *OpenFog Reference Architecture for Fog Computing*. USA: OpenFog Consortium, 2017.
- [63] A. Munir, P. Kansakar, and S. U. Khan, “IFCIoT: Integrated fog cloud IoT: A novel architectural paradigm for the future Internet of Things,” *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 74–82, 2017.

- [64] N. Mohan and J. Kangasharju, "Edge-Fog cloud: A distributed cloud for Internet of Things computations," in *2016 Cloudification of the Internet of Things (CIoT)*, Paris, France, 2016, pp. 1–6.
- [65] A. A. Alli and M. M. Alam, "The fog cloud of things: A survey on concepts, architecture, standards, tools, and applications," *Internet of Things*, vol. 9, p. 100177, 2020.
- [66] G. Ortiz, M. Zouai, O. Kazar, A. Garcia-de-Prado, and J. Boubeta-Puig, "Atmosphere: Context and situational-aware collaborative IoT architecture for edge-fog-cloud computing," *Computer Standards & Interfaces*, vol. 79, p. 103550, 2022.
- [67] P. R. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, "Self-aware Computing: Introduction and Motivation," in *Self-Aware Computing Systems. Natural Computing Series*, P. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, Eds. Cham, Switzerland: Springer, 2016, pp. 1–5.
- [68] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
- [69] P. Horn, "Autonomic computing: IBMs perspective on the state of information technology," *International Business Machines Corporation*, Armonk, NY, USA, 2001.
- [70] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, "A concise introduction to autonomic computing," *Advanced Engineering Informatics*, vol. 19, no. 3, pp. 181–187, 2005.
- [71] M. Parashar and S. Hariri, "Autonomic computing: An overview," in *Unconventional Programming Paradigms (UUP). Lecture Notes in Computer Science*, vol. 3566, J. P. Banâtre, P. Fradet, J. L. Giavitto, and O. Michel, Eds. Berlin, Heidelberg: Springer, 2005, pp. 257–269.
- [72] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, "Fulfilling the vision of autonomic computing," *Computer*, vol. 43, no. 1, pp. 35–41, 2010.
- [73] E. Amir, M. L. Anderson, and V. K. Chaudhi, *Report on DARPA Workshop on Self-Aware Computer Systems, Tech. Rep UIUCDCS-R-2007-2810*, UIUC Computer Science, 2007.
- [74] P. R. Lewis, A. Chandra, and K. Glette, "Self-awareness and self-expression: Inspiration from psychology," in *Self-Aware Computing Systems. Natural Computing Series*, P. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, Eds. Cham, Switzerland: Springer, 2016, pp. 9–21.
- [75] S. Baker, "The identification of the self," *Psychological Review*, vol. 4, no. 3, pp. 272–284, 1897.
- [76] G. A. Tawney, "Feeling and self-awareness," *Psychological Review*, vol. 9, no. 6, pp. 570–596, 1902.
- [77] W. James, *The Principles of Psychology, Vol 1.*, Henry Holt & Co., 1890.
- [78] D. Hume, *A Treatise of Human Nature*, *Gutenberg eBook*, 1739. [Online]. Available: <http://www.gutenberg.org/ebooks/4705>
- [79] I. Kant, "The critique of pure reason," *Gutenberg eBook*, 1781. [Online]. Available: <http://www.gutenberg.org/ebooks/4280>
- [80] P. R. Lewis *et al.*, "A survey of self-awareness and its application in computing systems," in *Proc. 5th IEEE Conf. on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, Ann Arbor, MI, USA, 2011, pp. 102–107.
- [81] S. Kounev, X. Zhu, J. O. Kephart, and M. Kwiatkowska, "Model-driven Algorithms and Architectures for Self-Aware Computing Systems (Dagstuhl Seminar 15041)," *Dagstuhl Reports*, vol. 5, no. 1, pp. 164–196, 2015.
- [82] G. Lehmann, M. Blumendorf, F. Trollmann, and S. Albayrak, "Meta-modeling runtime models," in *Models in Software Engineering (MODELS). Lecture Notes in Computer Science*, vol. 6627, J. Dingel, and A. Solberg, Eds. Berlin, Heidelberg: Springer, 2010, pp. 209–223.
- [83] P. R. Lewis *et al.*, "Architectural aspects of self-Aware and self-expressive computing systems: From psychology to engineering," *Computer*, vol. 48, no. 8, pp. 62–70, 2015.
- [84] U. Neisser, "The roots of self-knowledge: Perceiving self, it, and thou," *Annals of the New York Academy of Sciences*, vol. 818, no. 1, pp. 19–33, 1997.
- [85] M. Mitchell, "Self-awareness and control in decentralized systems," in *AAAI Spring Symposium: Metacognition in computation*, 2005, pp. 80–85.

- [86] A. Diaconescu *et al.*, “Architectures for collective self-aware computing systems,” in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 191–235.
- [87] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2017.
- [88] *Model Driven Architecture (MDA), MDA Guide rev. 2.0*. Object Management Group-OMG. White Paper, Jun. 2014. [Online]. Available: <https://www.omg.org/cgi-bin/doc?ormsc/14-06-01>
- [89] J. García-Molina, F. García-Rubio, V. Pelechano, A. Vallecillo, J. M. Vera, and C. Vicente-Chicote, *Desarrollo de software dirigido por modelos: Conceptos, métodos y herramientas*. Madrid, Spain: Ra-Ma, 2012.
- [90] R. France and B. Rumpe, “Model-driven development of complex software: A research roadmap,” in *Proc. Future of Software Engineering (FOSE)*, Minneapolis, MN, USA, 2007, pp. 37–54.
- [91] G. Booch, “Growing the UML,” *Software and Systems Modeling*, vol. 1, no. 2, pp. 157–160, 2002.
- [92] P. Cedillo, “Monitorización de calidad de servicios cloud mediante modelos en tiempo de ejecución,” Ph. D. dissertation, Universitat Politècnica de València, 2017.
- [93] D. C. Schmidt, “Model-driven engineering,” *Computer*, vol. 39, no. 2, pp. 25–31, 2006.
- [94] B. Selic, “The pragmatics of model-driven development,” *IEEE Software*, vol. 20, no. 5, pp. 19–25, 2003.
- [95] B. Selic, “MDA manifestations,” *European Journal for the Informatics Professional*, vol. 9, no. 2, pp. 12–16, 2008.
- [96] J. Bézivin, M. Barbero, and F. Jouault, “On the applicability scope of model driven engineering,” in *Proc. Fourth Int. Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*, Braga, Portugal, 2007, pp. 3–7.
- [97] *Architecture-Driven Modernization*. Object Management Group-OMG. Accessed: Oct. 19, 2023. [Online]. Available: <https://www.omg.org/adm/>
- [98] *Meta-Object Facility Version 2.5.1*. Object Management Group-OMG. Accessed: Oct, 2016. [Online]. Available: <https://www.omg.org/spec/MOF/>
- [99] Obeo Designer Community. (11.4). *OBEO—Eclipse Sirius*. Accessed: Feb. 7, 2022. [Online]. Available: <https://www.obeodesigner.com/en/product/sirius>
- [100] *Eclipse Modeling Framework (EMF) Documentation*. Eclipse Foundation. Accessed: Feb. 7, 2022. [Online]. Available: <https://www.eclipse.org/modeling/emf/docs/>
- [101] C. Ghezzi, “The Fading Boundary between Development Time and Run Time,” in *IEEE Ninth European Conference on Web Services*, Lugano, Switzerland, 2011, pp. 11–11.
- [102] N. Bencomo, “On the use of software models during software execution,” in *ICSE Workshop on Modeling in Software Engineering*, Vancouver, BC, Canada, 2009, pp. 62–67.
- [103] B. Morin *et al.*, “An aspect-oriented and model-driven approach for managing dynamic variability,” in *Model Driven Engineering Languages and Systems (MODELS). Lecture Notes in Computer Science*, vol. 5301, K. Czarnecki, I. Ober, J. M. Bruel, A. Uhl, and M. Völter, Eds. Berlin, Heidelberg: Springer, 2008, pp. 782–796.
- [104] H. Giese and R. Wagner, “Incremental model synchronization with triple graph grammars,” in *Model Driven Engineering Languages and Systems (MODELS). Lecture Notes in Computer Science*, vol. 4199, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds. Berlin, Heidelberg: Springer, 2006, pp. 543–557.
- [105] M. Ferreira *et al.*, *Medición del Software Ontología y Metamodelo*. Departamento de Tecnologías y Sistemas de la Información, Universidad de Castilla-La Mancha, 2006.
- [106] F. García *et al.*, *Una Ontología de la Medición del Software*. Departamento de Informática, Universidad de Castilla-La Mancha, 2004.
- [107] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (3rd Edition)*. Westford, MA, USA: Addison-Wesley, 2012.
- [108] M. Bauer, N. Bui, J. D. Loof, and C. Magerkurth, “IoT reference model,” in *Enabling Things to Talk*, A. Bassi, Eds. Berlin, Germany: Springer, 2013, pp. 113–162.

- [109] S. Haller, A. Serbanati, M. Bauer, and F. Carrez, “A domain model for the Internet of Things,” in *Proc. IEEE Int. Conf. on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Beijing, China, 2013, pp. 411–417.
- [110] P. Patel, A. Pathak, T. Teixeira, and V. Issarny, “Towards application development for the Internet of Things,” in *Proc. 8th Middleware Doctoral Symposium (MDS)*, Lisbon, Portugal, 2011, pp. 1–6.
- [111] P. Patel and D. Cassou, “Enabling high-level application development for the Internet of Things,” *Journal of Systems and Software*, vol. 103, pp. 62–84, 2015.
- [112] T. Eterovic, E. Kaljic, D. Donko, A. Salihbegovic, and S. Ribic, “An Internet of Things visual domain specific modeling language based on UML,” in *Proc. XXV Int. Conf. on Information, Communication and Automation Technologies (ICAT)*, Sarajevo, Bosnia Herzegovina, 2015, pp. 1–5.
- [113] F. Ciccuzzi and R. Spalazzese, “MDE4IoT: Supporting the internet of things with model-driven engineering,” in *Intelligent Distributed Computing X. IDC. Studies in Computational Intelligence*, vol. 678, C. Badica *et al.*, Eds. Cham, Switzerland: Springer, 2017, pp. 67–76.
- [114] A. Salihbegovic, T. Eterovic, E. Kaljic, and S. Ribic, “Design of a domain specific language and IDE for Internet of Things applications,” in *Proc. 38th Int. Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 2015, pp. 996–1001.
- [115] N. Harrand, F. Fleurey, B. Morin, and K. E. Husa, “ThingML: A language and code generation framework for heterogeneous targets,” in *Proc. ACM/IEEE 19th Int. Conf. Model Driven Engineering Languages and Systems*, New York, NY, USA, 2016, pp. 125–135.
- [116] F. Pramudianto, M. Eisenhauer, C. A. Kamienski, D. Sadok, and E. J. Souto, “Connecting the Internet of Things rapidly through a model driven approach,” in *Proc. IEEE 3rd World Forum on Internet Things (WF-IoT)*, Reston, VA, USA, 2016, pp. 135–140.
- [117] B. Negash, T. Westerlund, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, “DoS-IL: A domain specific Internet of Things language for resource constrained devices,” *Procedia Computer Science*, vol. 109, pp. 416–423, 2017.
- [118] B. Costa, P. F. Pires, and F. C. Delicato, “Modeling SOA-based IoT applications with SoaML4IoT,” in *Proc. IEEE 5th World Forum on Internet Things (WF-IoT)*, Limerick, Ireland, 2019, pp. 496–501.
- [119] D. Alulema, J. Criado, and L. Iribarne, “A model-driven approach for the integration of hardware nodes in the IoT,” in *New Knowledge in Information Systems and Technologies (WorldCIST) (Advances in Intelligent Systems and Computing)*, vol. 930, Á. Rocha, H. Adeli, L. Reis, and S. Costanzo, Eds. Cham, Switzerland: Springer, 2019, pp. 801–811.
- [120] C. G. García, D. Meana-Llorián, V. García-Díaz, A. C. Jiménez, and J. P. Anzola, “Midgar: Creation of a graphic domain-specific language to generate smart objects for Internet of Things scenarios using model-driven engineering,” *IEEE Access*, vol. 8, pp. 141872–141894, 2020.
- [121] M. Sharaf, M. Abusair, R. Eleiwi, Y. Shana’a, I. Saleh, and H. Muccini, “Modeling and Code Generation Framework for IoT,” in *System Analysis and Modeling. Languages, Methods, and Tools for Industry 4.0. (SAM). Lecture Notes in Computer Science*, vol. 11753, P. Fonseca i Casas, M. R. Sancho, and E. Sherratt, Eds. Cham, Switzerland: Springer, 2019, pp. 99–115.
- [122] J. C. Kirchhof, B. Rumpe, D. Schmalzing, and A. Wortmann, “MontiThings: Model-Driven Development and Deployment of Reliable IoT Applications,” *Journal of Systems and Software*, vol. 183, p. 111087, 2022.
- [123] J. A. Barriga, P. J. Clemente, E. Sosa-Sánchez, and Á. E. Prieto, “SimulateIoT: Domain specific language to design, code generation and execute IoT simulation environments,” *IEEE Access*, vol. 9, pp. 92531–92552, 2021.
- [124] I. Alfonso, K. Garcés, H. Castro, and J. Cabot, “Modeling self-adaptative IoT architectures,” in *Proc. ACM/IEEE Int. Conf. on Model Driven Engineering Languages*

- and Systems Companion (MODELS-C)*, Fukuoka, Japan, Oct. 2021, pp.761–766.
- [125] I. Alfonso, K. Garcés, H. Castro, and J. Cabot, “A model-based infrastructure for the specification and runtime execution of self-adaptive IoT architectures,” *Computing*, vol. 105, pp. 1883–1906, 2023.
 - [126] A. Gomez-Cabrera, P. J. Escamilla-Ambrosio, and J. Happa, “ViLanIoT: A visual language for improving Internet of Things systems representation,” *Journal of Industrial Information Integration*, vol. 38, p. 100567, 2024.
 - [127] F. Ihrwe, D. Di Ruscio, S. Gianfranceschi, and A. Pierantonio, “CHESSIoT: A model-driven approach for engineering multi-layered IoT systems,” *Journal of Computer Languages*, vol. 78, p. 101254, 2024.
 - [128] S. Kohan, L. Johnstone, and D. Cetinkaya, “A survey on the model-centered approaches to conceptual modeling of IoT systems,” *Frontiers in Computer Science*, vol. 5, p. 1035225, 2023.
 - [129] L. Esterle *et al.*, “Self-awareness in cyber-physical systems: Recent developments and open challenges,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1–6, 2023.
 - [130] *Discover the world of Scientific Literature*. Litmaps. [Online]. Available: <https://www.litmaps.com/>
 - [131] *An Architectural Blueprint for Autonomic Computing*. IBM, White Paper, Jun. 2005. [Online]. Available: <https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf>
 - [132] P. R. Lewis and S. Sarkadi, “Reflective artificial intelligence,” *Minds & Machines*, vol. 34, no. 14, 2024.
 - [133] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (4th Edition)*, Boston, MA, USA: Pearson, 2020.
 - [134] M. Gotzinger *et al.*, “RoSA: A framework for modeling self-Awareness in cyber-physical systems,” *IEEE Access*, vol. 8, pp. 141373–141394, 2020.
 - [135] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, “A generalized software framework for accurate and efficient management of performance goals,” in *Proc. of the Int. Conf. on Embedded Software (EMSOFT)*, Montreal, QC, Canada, 2013, pp. 1–10.
 - [136] M. Mostl *et al.*, “Platform-centric self-Awareness as a key enabler for controlling changes in CPS,” in *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1543–1567, 2018.
 - [137] S. Sarma, N. Dutt, P. Gupta, N. Venkatasubramanian, and A. Nicolau, “CyberPhysical-System-On-Chip (CPSoC): A self-aware MPSoC paradigm with cross-layer virtual sensing and actuation,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2015, pp. 625–628.
 - [138] N. Dutt, A. Jantsch, and S. Sarma, “Self-aware Cyber-Physical Systems-on-Chip,” in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, Austin, TX, USA, 2015, pp. 46–50.
 - [139] H. Hoffmann, A. Jantsch, A. Jantsch, and N. D. Dutt, “Embodied Self-Aware Computing Systems,” in *Proceedings of the IEEE*, vol. 108, no. 7, pp. 1027–1046, 2020.
 - [140] S. J. Habib, P. N. Marimuthu, P. Renold, and B. G. Athi, “Development of self-aware and self-redesign framework for wireless sensor networks,” in *New Knowledge in Information Systems and Technologies. (WorldCIST). Advances in Intelligent Systems and Computing*, vol. 931, Á. Rocha, H. Adeli, L. Reis, and S. Costanzo, Eds. Cham, Switzerland: Springer, 2019, pp. 438–448.
 - [141] R. Angarita, “Responsible objects: Towards self-healing internet of things applications,” in *Proc. IEEE Int. Conf. on Autonomic Computing (ICAC)*, Grenoble, France, 2015, pp. 307–312.
 - [142] W. Ettazi and M. Nassar, “Towards a cognitive engineering of transactional services in IoT based systems,” *Journal of Systems and Software*, vol. 200, p. 111634, 2023.
 - [143] T. Zoppi, G. Merlino, A. Ceccarelli, A. Puliafito, and A. Bondavalli, “Anomaly Detectors for Self-Aware Edge and IoT Devices,” in *Proc. IEEE 23rd Int. Conf. on Software Quality, Reliability, and Security (QRS)*, Chiang Mai, Thailand, 2023, pp. 24–35.

- [144] N. Mohamudally, "Paving the Way Towards Collective Intelligence at the IoT Edge," *Procedia Computer Science*, vol. 203, pp. 8–15, 2022.
- [145] Y. Jin *et al.*, "Self-aware distributed deep learning framework for heterogeneous IoT edge devices," *Future Generation Computer Systems*, vol. 125, pp. 908–920, 2021.
- [146] J. S. Preden, K. Tammema, A. Jantsch, M. Leier, A. Riid, and E. Calis, "The Benefits of Self-Awareness and Attention in Fog and Mist Computing," *Computer*, vol. 48, no. 7, pp. 37–45, 2015.
- [147] L. Esterle and B. Rinner, "An architecture for self-aware IoT applications," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, Canada, 2018, pp. 6588–6592.
- [148] E. Gerbert-Gaillard, S. Chollet, and P. Lalanda, "Model-driven approach for self-aware pervasive systems," in *Proc. IEEE/ACIS 15th Int. Conf. on Computer and Information Science (ICIS)*, Okayama, Japan, 2016, pp. 1–6.
- [149] E. Gerbert-Gaillard, P. Lalanda, S. Chollet, and J. Demarchez, "A self-aware approach to context management in pervasive platforms," in *Proc. IEEE Int. Conf. on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kona, HI, USA, 2017, pp. 256–261.
- [150] P. Lalanda, E. Gerbert-Gaillard, and S. Chollet, "Self-Aware Context in Smart Home Pervasive Platforms," *IEEE Int. Conf. on Autonomic Computing (ICAC)*, Columbus, OH, USA, 2017, pp. 119–124.
- [151] S. Spinner, A. Filieri, S. Kounev, M. Maggio, and A. Robertsson, "Run-time models for online performance and resource management in data centers," in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 485–505.
- [152] S. Kounev, N. Huber, F. Brosig, and X. Zhu, "A model-based approach to designing self-aware IT systems and infrastructures," *Computer*, vol. 49, no. 7, pp. 53–61.
- [153] N. Huber, F. Brosig, S. Spinner, S. Kounev, and M. Bähr, "Model-based self-Aware performance and resource management using the Descartes modeling language," *IEEE Transactions on Software Engineering*, vol. 43, no. 5, pp. 432–452.
- [154] J. Walter, A. Di Marco, S. Spinner, P. Inverardi, and S. Kounev, "Online learning of run-time models for performance and resource management in data centers," in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 507–528.
- [155] E. Kasaeyan Naeini, S. Shahhosseini, A. Subramanian, T. Yin, A. M. Rahmani, and N. Dutt, "An Edge-Assisted and Smart System for Real-Time Pain Monitoring," in *Proc. IEEE/ACM Int. Conf. on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, Arlington, VA, USA, 2019, pp. 47–52.
- [156] F. Forooghifar, A. Aminifar, and D. Atienza Alonso, "Self-aware wearable systems in epileptic seizure detection," in *Proc. 21st Euromicro Conference on Digital System Design (DSD)*, Prague, Czech Republic, 2018, pp. 426–432.
- [157] F. Forooghifar, A. Aminifar, and D. Atienza, "Resource-Aware Distributed Epilepsy Monitoring Using Self-Awareness From Edge to Cloud," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1338–1350, 2019.
- [158] F. Dell’agnola, U. Pale, R. Marino, A. Arza, and D. Atienza, "MBioTracker: Multimodal Self-Aware Bio-Monitoring Wearable System for Online Workload Detection," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 5, pp. 994–1007, 2021.
- [159] A. Anzanpour *et al.*, "Self-awareness in remote health monitoring systems using wearable electronics," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, Switzerland, 2017, pp. 1056–1061.
- [160] J. Schladow *et al.*, "Self-awareness in autonomous automotive systems," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, Switzerland, 2017, pp. 1050–1055.
- [161] D. Kaiser *et al.*, "Towards self-aware multirotor formations," *Computers*, vol. 9, no. 1, p. 7, 2020.
- [162] A. Akbar and S. B. Belhaouarie, "SaVE: Self-aware Vehicular Edge Computing with

- Efficient Resource Allocation,” in *Proc. IEEE Int. Conf. on Autonomic Computing and Self-Organizing Systems (ACSOS)*, Toronto, ON, Canada, 2023, pp. 157–162.
- [163] B. Rinner *et al.*, “Self-Aware and Self-Expressive camera networks,” *Computer*, vol. 48, no. 7, pp. 21–28, 2015.
- [164] L. Esterle, J. Simonjan, G. Nebel, R. Pflugfelder, G. F. Domínguez, and B. Rinner, “Self-aware object tracking in multi-camera networks,” in *Self-Aware Computing Systems. Natural Computing Series*, P. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, Eds. Cham, Switzerland: Springer, 2016, pp. 261–277.
- [165] Z. Guettatfi, P. Hubner, M. Platzner, and B. Rinner, “Computational self-awareness as design approach for visual sensor nodes,” in *Proc. 12th Int. Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, Madrid, Spain, 2017, pp. 1–8.
- [166] A. Akbar and P. R. Lewis, “Self-adaptive and self-aware mobile-cloud hybrid robotics,” in *Proc. 5th Int. Conf. on Internet of Things: Systems, Management and Security (IoTSMS)*, Valencia, Spain, 2018, pp. 262–267.
- [167] A. Akbar, P. R. Lewis, and E. Wanner, “A Self-Aware and Scalable Solution for Efficient Mobile-Cloud Hybrid Robotics,” *Frontiers in Robotics and AI*, vol. 7, p. 102, 2020.
- [168] S. Kounev, F. Brosig, and N. Huber, *The Descartes Modeling Language*. University of Würzburg, 2014.
- [169] *Software & Systems Process Engineering Meta Model Specification Version 2.0*. Object Management Group-OMG. Apr. 2008. [Online]. Available: <https://www.omg.org/spec/SP-EM/2.0/PDF>
- [170] J. J. Dujmovic, “A method for evaluation and selection of complex hardware and software systems,” in *Proc. 22nd Int. for the Resource Management & Performance Evaluation of Enterprise Computing Systems, CMG. Part 1(of 2)*, pp. 368-378 1996.
- [171] J. Moyano and A. Roman, “Creación de una infraestructura auto adaptativa de monitoreo mediante un middleware de configuración de componentes sensibles al contexto en ambientes de vida asistida,” B.S. Thesis, Universidad de Cuenca, Cuenca, Ecuador, 2021. [Online]. Available: <http://dspace.ucuenca.edu.ec/handle/123456789/36011>
- [172] *About Node.js*. OpenJS Foundation. Accessed: Jun. 5, 2023. [Online]. Available: <https://nodejs.org/en/about>
- [173] *Introduction to Node.js*. OpenJS Foundation. Accessed: Jun. 5, 2023. [Online]. Available: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>
- [174] *MySQL*. Oracle. Accessed: Jun. 5, 2023. [Online]. Available: <https://www.mysql.com/>
- [175] D. Bonifaz, “Configurador web de capacidades de autoconsciencia basado en modelos en tiempo de ejecución para sistemas de Internet de las Cosas (Internet of Things - IoT),” B.S. Thesis, Universidad del Azuay, Cuenca, Ecuador, 2022. [Online]. Available: <https://dspace.uazuay.edu.ec/handle/datos/11663>
- [176] *Express Framework*. OpenJS Foundation. Accessed: Jun. 5, 2023. [Online]. Available: <https://expressjs.com/>
- [177] M. Eernisse. *Embedded JavaScript templating - EJS*. Accessed: Jun. 5, 2023. [Online]. Available: <https://ejs.co/>
- [178] *Routing*. OpenJS Foundation. Accessed: Jun. 5, 2023. [Online]. Available: <https://expressjs.com/en/guide/routing.html>
- [179] W. Larriva and C. Alvarado, “Motor de sincronización entre sistemas de IoT y modelos de arquitectura-autoconsciencia en tiempo de ejecución,” B.S. Thesis, Universidad del Azuay, Cuenca, Ecuador, 2021. [Online]. Available: <https://dspace.uazuay.edu.ec/handle/datos/11558>
- [180] S. Ochoa, “Motor de autoconsciencia basado en modelos en tiempo de ejecución para sistemas IoT,” B.S. Thesis, Universidad del Azuay, Cuenca, Ecuador, 2022. [Online]. Available: <https://dspace.uazuay.edu.ec/handle/datos/12391>
- [181] B. Llivisaca, “Prototipo de Middleware basado en models@run.time para implementación automática de artefactos de software en sistemas distribuidos. Estudio de caso: Monitoreo de sistemas IoT,” B.S. Thesis, Universidad del Azuay, Cuenca, Ecuador, 2024. [Online]. Available: <https://dspace.uazuay.edu.ec/handle/datos/14244>
- [182] A. Coro, “Prototipo de middleware basado en models@run.time de autoconsciencia

- distribuida y colectiva para sistemas IoT,” B.S. Thesis, Universidad del Azuay, Cuenca, Ecuador, 2024. [Online]. Available: <https://dspace.uazuay.edu.ec/handle/datos/14057>
- [183] The V8 JavaScript Engine. OpenJS Foundation. Accessed: Jun. 5, 2023. [Online]. Available: <https://nodejs.org/en/learn/getting-started/the-v8-javascript-engine>
- [184] A. Maltos *¿Qué es la Metaprogramación en JavaScript?* Accessed: Dec. 29, 2020. [Online]. Available: <https://www.freecodecamp.org/espanol/news/que-es-la-metaprogramacion-en-javascript-en-espanol-por-favor/>
- [185] H. Bhardwaj and R. Khatana “Challenges with Implementation of Node.Js,” *Int. Journal for Research in Applied Science & Engineering Technology (IJRASET)*, vol. 9, no. 8, pp. 1086-1090, 2021.
- [186] D. Ancona, L. Franceschini, G. Delzanno, M. Leotta, M. Ribaudo, and F. Ricca, “Towards runtime monitoring of node.js and its application to the Internet of Things,” *arXiv preprint arXiv:1802.01790*, 2018.
- [187] *eval*. MDN contributors. Accessed: Jun. 5, 2023. [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/eval
- [188] R. C. Sofia and P. M. Mendes, “An overview on push-based communication models for information-centric networking,” *Future Internet*, vol. 11, no. 3, p. 74, 2019.
- [189] *MQTT.js Version 5.7.0*. NPM. Accessed: Jun. 5, 2023. [Online]. Available: <https://www.npmjs.com/package/mqtt>
- [190] *chokidar Version 3.6.0*. NPM. Accessed: Jun. 5, 2023. [Online]. Available: <https://www.npmjs.com/package/chokidar>
- [191] *setInterval() global function*. MDN contributors Accessed: Jun. 5, 2023. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/setInterval>
- [192] R. Hebig, A. Seibel, and H. Giese, “On the unification of megamodels,” in *Electronic Communications of the EASST*, vol. 42, pp. 1-12, 2011.
- [193] K. L. Bellman *et al.*, “Self-modeling and self-awareness,” in *Self-Aware Computing Systems*, S. Kounev, J. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 279–304.
- [194] P. Cedillo, E. Insfran, and S. Abrahão, “Monitoring Cloud Services through Models at Runtime: A Case in an Ambient Assisted Living Environment,” *Journal of Object Technology*, vol. 21, no. 4, pp. 1–19.
- [195] *DHT11 Humidity & Temperature Sensor*. Alldatasheet. Accessed: May. 10, 2023. [Online]. Available: <https://www.alldatasheet.com/datasheet-pdf/pdf/1440068/ETC/DHT11.html>
- [196] *MQ-7 Semiconductor Sensor for Carbon Monoxide*. Henan Hanwei Electronics. Accessed: May. 10, 2023. [Online]. Available: <https://www.pololu.com/file/0J313/MQ7.pdf>
- [197] *GPIO and the 40-pin Header*. Raspberry Pi. Accessed: May. 10, 2023. [Online]. Available: <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#gpio-and-the-40-pin-header>
- [198] *PostgreSQL*. The PostgreSQL Global Development Group. Accessed: Jun. 5, 2023. [Online]. Available: <https://www.postgresql.org/>
- [199] *Eclipse Mosquitto*. Eclipse Foundation. Accessed: Jun. 5, 2023. [Online]. Available: <https://mosquitto.org/>
- [200] *CEDIA Cloud*. CEDIA. [Online]. May. 10, 2023. Available: <https://cedia.edu.ec/beneficio/cedia-cloud/>
- [201] V. R. Basili, “The experimental paradigm in software engineering,” in *Experimental Software Engineering Issues: Critical Assessment and Future Directions. Lecture Notes in Computer Science*, vol. 706, H. D. Rombach, V. R. Basili, and R. W. Selby, Eds. Berlin, Heidelberg: Springer, 1993, pp. 1–12.
- [202] B. A. Kitchenham, S. L. Pfleeger, and L. M. Pickard, “Preliminary guidelines for empirical research in software engineering,” *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002.
- [203] M. Höst, B. Regnell, and C. Wohlin, “Using students as subjects—A comparative study of students and professionals in lead-time impact assessment,” *Empirical Software Engineering*, vol. 5, no. 3, pp. 201–214, 2000.
- [204] V. R. Basili, F. Shull, and F. Lanubile, “Building knowledge through families of

- experiments,” *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 456–473, 1999.
- [205] S. Abrahão, E. Insfran, J. A. Carsí, and M. Genero, “Evaluating requirements modeling methods based on user perceptions: A family of experiments,” *Information Sciences*, vol. 181, no. 16, pp. 3356–3378, 2011.
 - [206] D. L. Moody, G. Sindre, T. Brasethvik, and A. Solvberg, “Evaluating the quality of information models: empirical testing of a conceptual model quality framework,” in *Proc. 25th Int. Conf. on Software Engineering*, Portland, OR, USA, 2003, pp. 295–305.
 - [207] N. Rescher, *Methodological pragmatism: A systems-theoretic approach to the theory of knowledge*. Oxford: Basil Blackwell, 1977.
 - [208] F. D. Davis, “A technology acceptance model for empirically testing new end-user information systems: Theory and results,” Ph.D. dissertation, Dept. Sloan School Manage., MIT, Boston, MA, USA, 1985. [Online]. Available: <https://dspace.mit.edu/handle/1721.1/15192>
 - [209] V. R. Basili, G. Caldiera, and H. D. Rombach, “The goal question metric approach,” in *Encyclopedia of Software Engineering*, vol. 1, J. J. Marciniak, Ed. New York, NY, USA: Wiley 1994, pp. 528–532.
 - [210] D. L. Moody, “Dealing with complexity: A practical method for representing large entity relationship models,” Ph.D. dissertation, Department of Information Systems, University of Melbourne, Melbourne, VIC, Australia, 2001.
 - [211] T. D. Cook and D. T. Campbell, *Quasi-experimentation: Design & Analysis Issues in Field Settings*. Boston, MA, USA: Houghton Mifflin, 1979.
 - [212] L. J. Cronbach, “Coefficient alpha and the internal structure of tests,” *Psychometrika*, vol. 16, no. 3, pp. 297–334, 1951.

Anexo A

Metamodelo Monitor-IoT

En este anexo se explica cada una de las metaclases del metamodelo Monitor-IoT. A su vez, las Figuras A-1, A-2, A-3 y A-4 muestran el metamodelo completo, incluyendo todas sus metaclases, relaciones, atributos y enumeraciones.

MonitoringArchitectureModel. Metaclase principal que describe y contiene el modelo de arquitectura de monitoreo multicapa de un sistema IoT (ver Fig. 5-2(1), A-1(1), y A-3(1)).

IoTSystem. Representa un sistema compuesto por un conjunto de entidades físicas (cosas) y digitales que interactúan y cooperan entre sí para recolectar, transmitir, procesar, analizar y almacenar datos del mundo físico y virtual, así como también para actuar sobre las entidades físicas a partir de instrucciones digitales, con la finalidad de brindar servicios a los usuarios finales en una diversidad de dominios (p. ej., hogar, salud, transporte, industria). El metamodelo permite la representación de ecosistemas IoT mediante la reutilización de las capacidades físicas y computacionales de los sistemas individuales (subsistemas) de IoT existentes en diferentes dominios, posibilitando la prestación de servicios emergentes más inteligentes con un uso eficiente de recursos. Por ejemplo, un ecosistema IoT emergente de recuperación de desastres puede estar compuesto por un sistema de transporte, un sistema de salud y un sistema de emergencias (ver Fig. 5-2(2) y A-1(2)).

EntityCategory. Define una agrupación de entidades del mismo tipo para especificar por una sola vez (empleando una plantilla) la información general sobre sus propiedades y APIs comunes. Esta abstracción reduce la carga de trabajo al momento de modelar la arquitectura de monitoreo de un sistema IoT. Por ejemplo, se puede crear una categoría que represente todas las habitaciones a ser monitoreadas en una casa (ver Fig. 5-2(3) y A-1(3)).

Entity. Concepto global que representa cualquier cosa (física o digital) con características distintivas e independiente (p. ej., lugares, personas, electrodomésticos, dispositivos electrónicos, servidores, redes de comunicación, componentes de software). En algunas de las propuestas estudiadas suelen llamarla “entidad de interés” para resaltar que es algo de interés del usuario para alcanzar sus objetivos [108] (ver Fig. 5-2(4) y A-1(4)).

PropertyTemplate. Define una plantilla con las especificaciones generales de una propiedad común para todas las entidades de una categoría. El metamodelo soporta tres tipos de propiedades: i) *metadatos*, contienen información que rara vez cambia sobre una entidad (p. ej., número de serie, marca, modelo, fecha de fabricación), pudiendo determinarse en tiempo de diseño; ii) *telemetría*, datos sobre una entidad física o el entorno recopilados a través de sensores (p. ej., temperatura o humedad de una habitación, presión arterial o nivel de glucosa en la sangre de un paciente); y iii) *estado*, información que describe el estado actual de una entidad digital (p. ej., nivel de energía de la batería de un sensor, cantidad de memoria o procesamiento utilizado por un nodo Edge, ancho de banda utilizado por una red). Una propiedad puede identificar de forma única a una entidad dentro de un sistema IoT (atributo *identifier*); además, posee un tipo de dato (atributo *dataType*) y una unidad de medida (metaclase *MeasurementUnit*) (ver Fig. 5-2(5), A-1(5, 6) y A-2(9)).

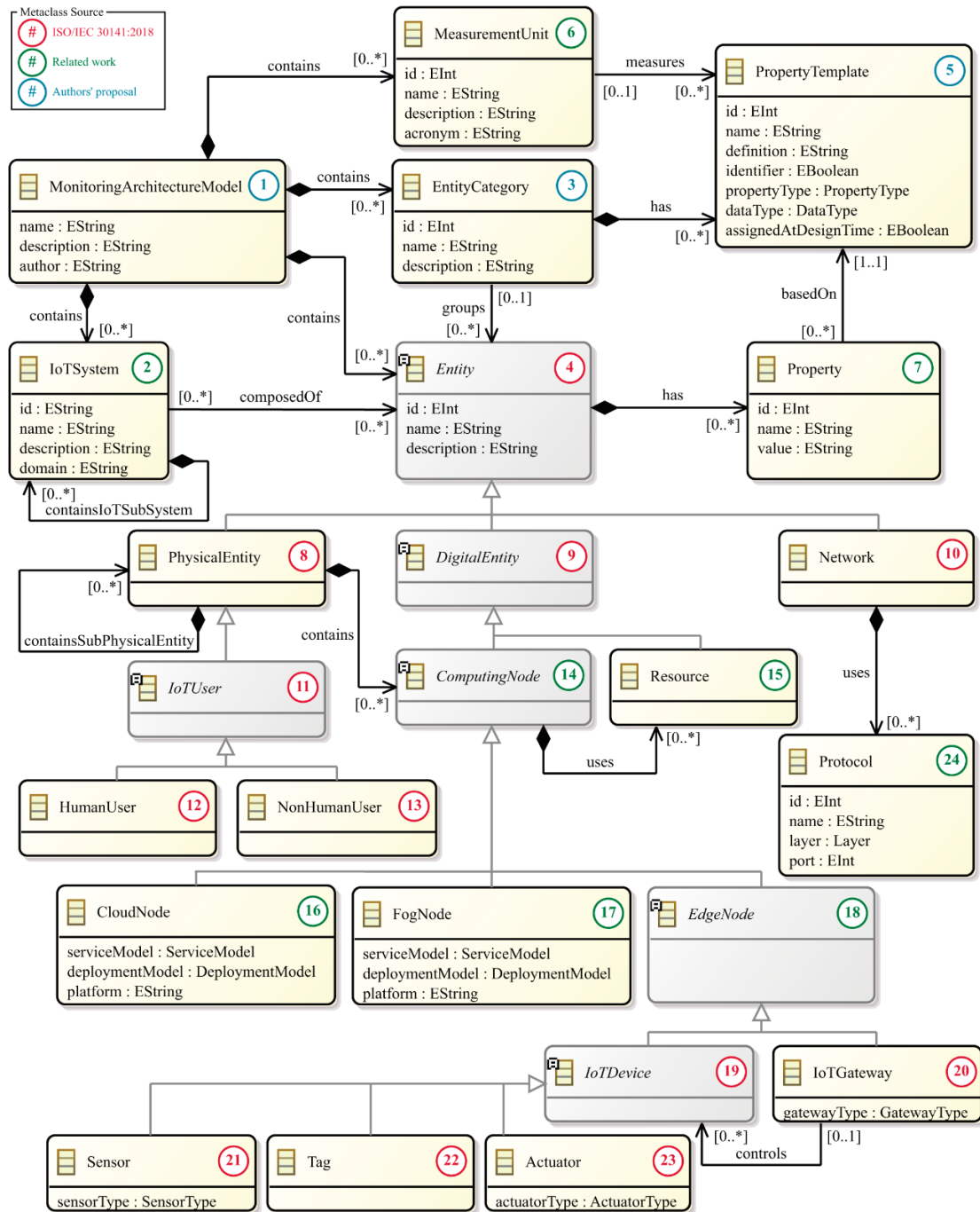


Figura A-1. Metamodelo Monitor-IoT (Parte 1), incluye metaclasses y relaciones para representar subsistemas, entidades físicas, entidades digitales (nodos de computación, dispositivos IoT) y redes de comunicación [56]. Fuente: Elaboración propia.

Property. Define un atributo que caracteriza a una determinada entidad física o digital en particular. Las propiedades permiten identificar o describir una entidad, determinar su estado en un momento dado, así como su evolución en el tiempo (ver Fig. 5-2(6), A-1(7), A-2(10) y A-3(7)). A su vez, las especificaciones generales de una propiedad están contenidas en una plantilla (metaclase *PropertyTemplate*).

PhysicalEntity. Representa una cosa del mundo real que es monitoreada por un sensor y/o controlada por un actuador. Las entidades físicas pueden ser: seres vivos, objetos o el entorno (p. ej., edificios, personas, animales, vehículos, electrodomésticos, artículos, ropa, medio ambiente). Una entidad física se extiende al mundo digital al incorporar, contener o llevar consigo uno o más dispositivos de tecnologías de información y comunicación que proporcionan una interfaz para obtener información o actuar sobre la entidad física. Además, una entidad física puede contener otras entidades físicas; por ejemplo, una casa puede contener habitaciones, y estas a su vez electrodomésticos (ver Fig. 5-2(7) y A-1(8)).

DigitalEntity. Representa un elemento computacional (a nivel de hardware o software) de un sistema IoT. Estos elementos pueden ser: nodos Cloud, nodos Fog, puertas de enlace, dispositivos IoT (sensores, tags, actuadores), middlewares, base de datos, servicios, APIs, aplicaciones, entre otros. Por lo tanto, las entidades digitales pueden especializarse en: nodos de cómputo y recursos de hardware o software (ver Fig. 5-2(8) y A-1(9)).

IoTUser. Representa un actor o beneficiario de un sistema IoT que puede especializarse en: i) *HumanUser*, persona que interactúa con un sistema IoT a través de una o más aplicaciones que se ejecutan en dispositivos de usuario (p. ej., computadora personal, tablet, smartphone u otros dispositivos especializados); y, ii) *NonHumanUser*, máquina, artefacto o dispositivo (p. ej., robot, vehículo) que actúa a nombre de los usuarios humanos y puede interactuar con uno o más servicios ofrecidos por un sistema IoT a través de una red de comunicación (ver Fig. 5-2(10-12) y A-1(11-13)).

ComputingNode. Representa una entidad digital con capacidades para recolectar, intercambiar, procesar y almacenar los datos en un sistema IoT. Los nodos de computación pueden ubicarse a nivel de las capas Cloud, Fog o Edge. Además, estos nodos pueden contener recursos de hardware y software, tales como: interfaces de red, middleware, agentes (broker), base de datos, servicios, APIs o aplicaciones (ver Fig. 5-2(13), A-1(14) y A-2(2)).

CloudNode. Nodo de computación centralizado con altas prestaciones para procesar, analizar y almacenar los datos obtenidos directamente de los dispositivos IoT o por medio de los nodos de computación intermedios Fog y Edge (ver Fig. 5-2(15) y A-1(16)).

FogNode. Nodo de computación descentralizado que actúa como una instancia mediadora entre el Cloud y los dispositivos Edge (sensores, actuadores y puertas de enlace). Los nodos Fog cumplen el rol de capa intermedia, deciden qué datos se almacenan y procesan en esta capa y qué datos se preparan y envían a la nube para su almacenamiento, procesamiento y análisis posterior. El objetivo de los nodos Fog es reducir los costos, la latencia y el tráfico de datos en la red, al implementar nodos de almacenamiento y procesamiento de datos más cerca de la fuente, es decir, de los dispositivos IoT (ver Fig. 5-2(16) y A-1(17)).

EdgeNode. Nodo de computación o dispositivo ubicado en el borde de la red de un sistema IoT. Los nodos Edge pueden ser: dispositivos IoT (sensores, tags, actuadores),

controladores o puertas de enlace IoT. Por lo tanto, pueden tener variadas capacidades de hardware, siendo en algunos casos capaces de almacenar y procesar datos en el mismo dispositivo, es decir, al borde de la red. Además, estos nodos pueden ubicarse cerca o estar incorporados en las entidades físicas para proporcionarles capacidades de detección, procesamiento, almacenamiento o actuación (ver Fig. 5-2(17) y A-1(18)).

IoTDevice. Dispositivo ubicado al borde de la red que asocia a las entidades físicas del mundo real con otras entidades digitales de un sistema IoT. Los dispositivos IoT interactúan a través de una red y dependiendo de sus capacidades de hardware y energía pueden comunicarse directamente o por intermedio de una puerta de enlace con las entidades digitales (nodos Cloud o Fog) ubicadas en redes de área amplia. Los sensores, tags y actuadores son un tipo de dispositivo IoT (ver Fig. 5-2(18), A-1(19) y A-2(3)).

Sensor. Dispositivo IoT que percibe o monitorea ciertas propiedades de una entidad física y las transforma en datos digitales para que puedan transmitirse a través de una red. Por ejemplo, un sensor de temperatura que envía las mediciones a una aplicación de un teléfono inteligente para monitorear la temperatura de una habitación (ver Fig. 5-2(20) y A-1(21)).

Tag. Objeto que se puede aplicar o incorporar a una entidad física con el fin de identificarla y rastrearla. El proceso de identificación se lleva a cabo mediante sensores (lectores) que leen las etiquetas (ver Fig. 5-2(21) y A-1(22)).

Actuator. Dispositivo IoT que actúa sobre (cambia) las propiedades de una entidad física mediante instrucciones digitales, es decir, permite modificar el estado de una entidad física. Por ejemplo, una aplicación móvil que se conecta vía Bluetooth con un aire acondicionado (actuador) para controlar la temperatura de una habitación (ver Fig. 5-2(22) y A-1(23)).

IoTGateway. Nodo de computación ubicado en el borde de la red. Este nodo actúa como intermediario para interconectar dispositivos IoT ubicados en redes de proximidad con otras entidades digitales (nodos Fog o Cloud) ubicadas en redes locales o de área amplia. A su vez, una puerta de enlace puede ofrecer una amplia gama de funcionalidades [2], tales como: i) agente de configuración, administración y control de los dispositivos IoT; ii) limpieza, filtrado y enrutamiento de datos, determinando qué datos deben enviarse a los nodos Fog o Cloud y qué datos deben procesarse localmente; iii) almacenamiento de datos recolectados por los dispositivos IoT asociados, como solución a problemas relacionados con redes de comunicación intermitentes o saturadas; iv) procesamiento local de datos para el control rápido y efectivo de los actuadores, en función de los datos de entrada de los sensores; v) conversión de protocolos y mapeo de direcciones; e, vi) implementación de seguridades a nivel de los dispositivos Edge. Además, las puertas de enlace pueden ser equipos independientes o estar integrados con otros dispositivos IoT de detección (sensores) y control (actuadores), por lo que pueden implementarse en: servidores, computadoras personales, microcontroladores, teléfonos inteligentes, enrutadores, máquinas virtuales, entre otros (ver Fig. 5-2(19) y A-1(20)).

Resource. Representa un componente de software o hardware utilizado por los nodos Cloud, Fog o Edge para detectar, recolectar, procesar y almacenar datos sobre las entidades físicas, así como para actuar sobre las mismas mediante instrucciones digitales. Los recursos pueden especializarse en: interfaces de red, APIs, aplicaciones, middleware, servicios, base de datos o archivos de configuración (ver Fig. 5-2(14), A-1(15) y A-2(1)).

NetworkInterface. Especifica una interfaz que conecta un nodo de computación o dispositivo IoT a una red de comunicación para compartir recursos y datos. Una interfaz de red utiliza una tecnología de comunicación (p. ej., Wifi, Ethernet, Bluetooth) y posee algún tipo de dirección de red que consiste en un identificador de nodo único por derecho propio (ver Fig. 5-2(24) y A-2(4)).

API (Application Programming Interface). Tipo de componente de software que encapsula funcionalidades para gestionar los dispositivos IoT (sensores, tags, actuadores). Específicamente, una API puede recuperar los datos sin procesar sobre las propiedades de las entidades físicas o actuar sobre ellas (cambiarlas). El metamodelo permite especificar los datos de entrada (metaclase *Parameter*) y la información de salida (metaclase *ReturnVariable*) de una API, así como encapsular la secuencia de instrucciones de la misma (atributo *instructions*) (ver Fig. 5-2(34, 35, 37) y A-2(6-8)).

Application. Componente de software que ofrece un conjunto de funcionalidades para realizar una tarea determinada. Una aplicación puede ser de escritorio, web, móvil o empotrada y ejecutarse en nodos Cloud, Fog y/o Edge. Además, una aplicación puede llamar a APIs para acceder a los datos observados por los sensores o controlar los actuadores. A su vez, una aplicación puede tanto consumir servicios para intercambiar datos con otras entidades digitales (comunicación síncrona), como publicar o recibir datos a través de un broker (comunicación asíncrona). Finalmente, una aplicación puede poseer una interfaz visual para interactuar con los usuarios humanos. Por ejemplo, en un sistema IoT para una casa inteligente, un usuario que utilice una aplicación instalada en un teléfono inteligente puede monitorear o ajustar la temperatura ambiente (ver Fig. 5-2(36) y A-2(11)).

Service. Componente de software que, a través de una interfaz abierta y estandarizada, brinda un conjunto de capacidades funcionales para intercambiar, procesar y almacenar datos, ocultando la heterogeneidad de las entidades involucradas. Los servicios están contenidos en middlewares y pueden ser accedidos a través de sus endpoints. Un servicio puede interactuar con otros servicios para proporcionar funcionalidades de alto nivel. Asimismo, los servicios pueden recibir datos de entrada (metaclase *Parameter*) y producir información de salida (metaclase *ReturnVariable*) representada en diferentes formatos (p. ej., JSON, XML, RDF). Adicionalmente, los servicios pueden contener cabeceras para enviar metadatos asociados a la solicitud o respuesta del servicio (metaclase *Header*) (ver Fig. 5-2(32-35) y A-2(7, 8, 12, 13)).

Middleware. Componente de software que soporta y provee servicios para la gestión (intercambio, procesamiento, almacenamiento) de datos de forma síncrona y asíncrona. Este recurso facilita la interoperabilidad entre las entidades digitales heterogéneas, ubicadas de manera distribuida en las capas Edge, Fog y Cloud de una plataforma IoT, mediante la implementación de servicios que ocultan los detalles de implementación particulares de cada entidad digital. Además, el metamodelo permite especificar uno o más protocolos y puertos de comunicación, a través de los cuales el middleware ofrece sus servicios (ver Fig. 5-2(29) y A-2(14)).

Broker. Es una especialización explícita de un middleware que soporta la mensajería o transferencia de datos asíncrona entre las entidades digitales de un sistema IoT. Un broker es el responsable de recibir, organizar y publicar los datos (payload) de las entidades IoT (editoras) en un determinado tópico; y, a su vez, distribuirlo a todas las entidades IoT (suscriptores) que se han suscrito a dicho tópico. Este modelo de transmisión de datos es recomendable para Internet de las Cosas, dado que los dispositivos IoT a menudo tienen

limitaciones de energía, procesamiento y ancho de banda. Los datos de las propiedades de una entidad se organizan e identifican mediante la metaclase *Topic* en un broker. Los tópicos pueden ser atómicos si incluyen datos de una única propiedad o compuestos si incluyen datos de varias propiedades. Además, pueden utilizar diferentes formatos para el intercambio de datos (p. ej., JSON, XML, RDF, HL7) (ver Fig. 5-2(30, 31) y A-2(16, 17)).

DataBase. Componente de software que almacena los datos de las propiedades monitoreadas en un sistema IoT. Los datos a almacenar pueden ser crudos (obtenidos directamente desde los dispositivos IoT) o ser el resultado de las operaciones de agregación sobre los datos recolectados, ejecutadas dentro de los servicios. El metamodelo permite especificar los datos de la cadena de conexión de las bases de datos empleadas. Una base de datos contiene tablas (metaclase *DataTable*) que pueden ser temporales o permanentes. Para el primer caso, se debe definir un intervalo de tiempo para gestionar la persistencia de los datos. A su vez, una tabla está conformada por columnas (metaclase *DataColumn*) que puede ser de dos tipos: i) *data*, contiene los datos recolectados o procesados para una propiedad; y, ii) *metadata*, contiene información que describe los datos recolectados o procesados, por ejemplo, qué dispositivo IoT recolectó el dato, a qué hora se recolectó el dato (marca de tiempo), la ubicación dónde se realizó la recolección, a qué entidad pertenece el dato, o qué nivel de calidad tienen los datos recolectados (precisión, completitud). Finalmente, la metaclase *DataInstance* representa los valores de las columnas, es decir, los valores de las propiedades monitoreadas y de sus metadatos asociados (ver Fig. 5-2(25-28), A-2(18-21) y A-3(8)).

Network. Representa una infraestructura de comunicación e intercambio de datos entre un conjunto de entidades digitales. El metamodelo soporta varias tecnologías (p. ej., Ethernet, Wifi, Bluetooth, Zigbee, Ultra Wideband, RFID) y protocolos de comunicación (metaclase *Protocol*) para interconectar nodos IoT heterogéneos con diversas capacidades de hardware. Los protocolos de comunicación de acuerdo con sus funciones se organizan en capas. Por ejemplo, a nivel de la capa de aplicación existen varios protocolos útiles para diferentes escenarios de IoT, tales como: CoAP (Constrained Application Protocol), MQTT (Message Queue Telemetry Transport) y HTTP/REST (Hypertext Transfer Protocol/ Representational State Transfer) (ver Fig. 5-2 (9, 23), A-1(10, 24) y A-2(5, 15)).

DataFlow. Representa con un alto grado de abstracción la trayectoria de los datos y como las entidades digitales (nodos de computación o sus recursos) interactúan dentro de un proceso de recolección, transferencia, procesamiento y almacenamiento de datos. En este sentido, un flujo de datos describe como los datos crudos recolectados por los dispositivos IoT son transportados a través de los diferentes nodos y recursos de computación hasta ser almacenados con diferentes niveles de agregación en las columnas y tablas correspondientes de una base de datos. Las propiedades *executionTimeInterval* y *flowExecutionTime* definen la frecuencia y hora de ejecución de un flujo de datos, respectivamente. El tipo de comunicación soportado por un flujos de datos pueden ser: i) *síncrona*, utiliza un modelo de solicitud/respuesta, en donde el envío – recepción de datos se realiza en tiempo real entre una entidad digital origen (cliente) que realiza la solicitud y una entidad digital destino (servidor) que recepta la solicitud; y, ii) *asíncrona*, emplea un modelo de publicación/suscripción de datos basado en eventos, es decir, los datos enviados por las entidades digitales emisoras (editoras) son publicados mediante tópicos en una entidad intermedia (broker) para que las entidades digitales receptoras (suscriptoras) puedan leerlos en otro momento. Un flujo de datos está compuesto por un conjunto de enlaces interconectados y ordenados (metaclase *Link*). Cada enlace permite la comunicación e intercambio de datos entre dos entidades digitales. Los tipos de enlace

admitidos por el metamodelo son: *API-IoTDevice*, *App-API*, *Service-API*, *App-Service*, *App-Broker*, *Broker-Service*, *Service-Service* y *Service-Database*. La relación *previousLink* define el orden de los enlaces que conforman un flujo de datos. Además, un flujo de datos contiene una o varias reglas de mapeo de datos (metaclase *DataMappingRule*) para relacionar las fuentes de datos (origen) con las ubicaciones de almacenamiento (destino) de un flujo de datos, mientras que los enlaces definen la forma como se transportaran los datos entre el origen y el destino (definen la trazabilidad de los datos). Las reglas de mapeo de datos pueden especializarse en las metaclases: i) *PropertyToDataColumn*, asocia una propiedad de una entidad con una columna de una tabla; y, ii) *DataColumnToDataColumn*, asocia dos columnas de datos, una columna origen a la que se aplican operaciones de agregación y su resultado se almacena en una columna destino. Cabe destacar que, si bien el alcance del metamodelo es soportar los flujos de recolección y agregación de datos, la diversidad de configuraciones, combinaciones y secuencias entre los enlaces que puede soportar un flujo de datos, proporciona flexibilidad y extensibilidad al metamodelo. De ahí que, adicional a los flujos de recolección y agregación de datos, el metamodelo podría soportar otros tipos de flujos de datos, por ejemplo: de actuación, generación de informes para la toma de decisiones, etc. (ver Fig. 5-2(38-42), A-2(22-30) y A-3(2-8)).

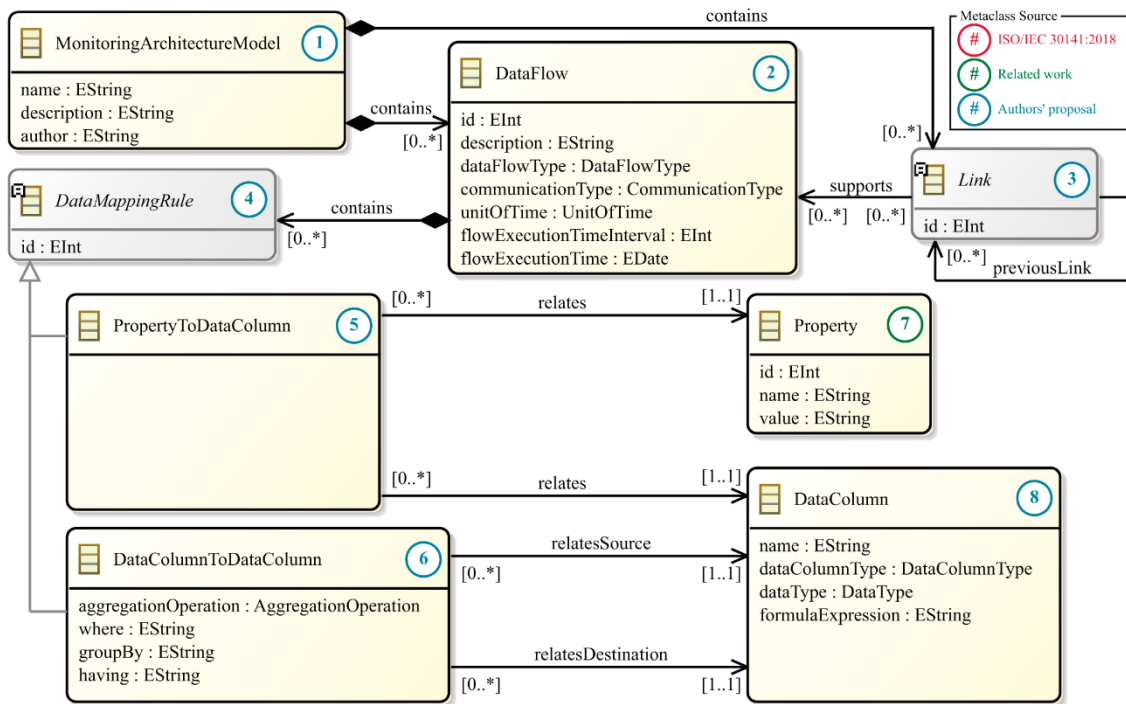


Figura A-3. Metamodelo Monitor-IoT (Parte 3), incluye metaclases y relaciones para representar los flujos de datos, los enlaces entre las entidades digitales y las reglas de mapeo [56]. Fuente: Elaboración propia.

Finalmente, la Figura A-4 presenta las enumeraciones del metamodelo Monitor-IoT. Algunas de las enumeraciones utilizadas por los atributos de las metaclases no tienen un carácter exhaustivo (p. ej., *SensorType*, *ActuatorType*, *AggregationOperation*, *CommunicationTechnology*); es decir, no reflejan todos los valores posibles. De ahí que pueden requerirse nuevos valores para las enumeraciones dependiendo de las necesidades de cada escenario de IoT.

 CommunicationType Asynchronous Synchronous	 TopicType Atomic Composed	 DataColumnType Data MetaData	 PersistenceType Temporary Permanent	 DataTableLinkType Select Insert
 BrokerLinkType Push Pull	 PropertyType Metadata Telemetry State	 ServiceModel SaaS PaaS IaaS	 DeploymentModel Public Private Community Híbrido	 DataFlowType DataCollectionFlow DataAggregationFlow ActuationFlow ReportGenerationFlow
 Method GET POST DELETE OPTIONS PUT	 ApplicationType Web Mobile Embedded Desktop Other	 DataFormatType JSON XML RDF Binary HL7 Other	 SensorType Infrared Proximity Pressure Ultrasonic Touch Other	 ActuatorType Electrical Mechanical Hydraulic Thermal Pneumatic Other
 Layer Application Presentation Session Transport Network DataLink Physical	 UnitOfTime Millisecond Second Minute Hour Day Week Month Year	 AggregationOperation Count Sum Mean Minimum Maximum StandardDeviation Variance Other	 DataType Byte Integer Long Float Double Char String Date Boolean Other	 GatewayType Server VirtualMachine Microcontroller PC Tablet Smartphone Router Switch SmartWatch Other
 CommunicationTechnology Wifi Bluetooth ZigBee Ethernet WiMax RFID OpticalFiber Other				

Figura A-4. Metamodelo Monitor-IoT (Parte 4), incluye las enumeraciones utilizadas por los atributos de las metaclasses [56]. Fuente: Elaboración propia.

Anexo B

Metamodelo Aware-IoT

En este anexo se explica cada una de las metaclases del metamodelo Aware-IoT. A su vez, las Figuras B-1, B-2, B-3 y B-4 presentan el metamodelo completo, incluyendo todas sus metaclases, relaciones, atributos y enumeraciones.

SelfAwarenessModel. Metaclase principal que describe y contiene el modelo de autoconsciencia de un sistema IoT (ver Fig. 5-6(1), B-1(1), B-2(1) y B-3(1)).

Span. Parte del dominio de autoconsciencia que representa el sujeto que es consciente de sí mismo y su entorno. El sujeto de autoconsciencia puede ser un sistema o subsistema IoT (metaclase *IoTSystem*) definido en el modelo de arquitectura del sistema IoT, mediante el lenguaje específico de dominio Monitor-IoT [56]. Los sujetos de autoconsciencia realizan procesos de pre-reflexión y reflexión para capturar el conocimiento sobre diversos aspectos de autoconsciencia de sus entidades físicas y digitales. Por ejemplo, un sistema de domótica es consciente del nivel de consumo promedio y tiempo de duración de la batería de sus sensores para una determinada frecuencia de lectura (ver Fig. 5-6(2, 3), B-1(2, 3)).

Scope. Parte del dominio de autoconsciencia que representan los objetos sobre los que se reflexiona y se es consciente. Dentro de un sistema IoT, existe una amplia gama de objetos de autoconsciencia que van desde entidades físicas hasta entidades digitales (metaclases *PhysicalEntity* y *DigitalEntity*), las cuales están definidas en el modelo de arquitectura del sistema IoT [56]. Ejemplos de estas entidades pueden ser: lugares, personas, electrodomésticos, dispositivos electrónicos, servidores, puertas de enlace, sensores, actuadores, redes de comunicación o componentes de software. Las entidades de un sistema IoT poseen propiedades (metaclase *Property*) que las caracterizan en el tiempo, siendo los valores de estas propiedades en distintos momentos, los datos crudos necesarios para evaluar los aspectos de autoconsciencia (capturar el conocimiento) mediante el uso de métricas (ver Fig. 5-6(4-6), B-1(4-6)).

SelfAwarenessAspect. Define un aspecto asociado a uno o varios objetos sobre el cual se requiere reflexionar y ser consciente. Para capturar el conocimiento sobre un determinado aspecto se aplican procesos y métodos de autoconsciencia. En términos generales, existen diversos tipos de aspectos (atributo *type*) que pueden ser capturados por un sistema autoconsciente, tales como: estado, comportamiento, causalidad, evolución en el tiempo, predicción, entre otros. Además, según el nivel de importancia que tiene el aspecto para el sistema IoT se le debe asignar un peso (atributo *weight*). Finalmente, los aspectos de autoconsciencia se pueden especializar en individual o colectivo (ver Fig. 5-6(11) y B-1(12)).

IndividualSelfAwarenessAspect. Especifica un aspecto de autoconsciencia en el ámbito de un único objeto y propiedad a evaluar, aislado de su entorno. Para la evaluación de los aspectos individuales se pueden utilizar procesos pre-reflexivos o reflexivos. En un sistema IoT, ejemplos de aspectos de autoconsciencia individuales pueden ser: conocer el estado de un sensor específico (encendido o apagado), el nivel de satisfacción de la temperatura de una habitación para un usuario, el nivel de riesgo de la concentración de monóxido de carbono (CO) en el ambiente de una casa para usuario; o, predecir el nivel de uso (carga) de la CPU de una puerta de enlace en particular para una cantidad

determinada de sensores y actuadores (ver Fig. 5-6(12) y B-1(13)).

CollectiveSelfawarenessAspect. Representa un aspecto de autoconsciencia con un nivel superior de abstracción. Este aspecto contempla una medida global (conocimiento) sobre el estado o evolución de un conjunto de objetos de autoconsciencia que forman parte o trabajan colectivamente en un sistema IoT. También, un aspecto colectivo puede contemplar la evaluación global de varias propiedades de un objeto de autoconsciencia. El metamodelo organiza jerárquicamente los aspectos de autoconsciencia individuales y colectivos en una estructura de árbol, ubicando los aspectos individuales a nivel de nodos hoja. A su vez, para la evaluación de los aspectos colectivos se utilizan procesos reflexivos basados en el Modelo de Agregación Lógica de Preferencias (Logic Scoring of Preference – LSP) [170]. Este modelo consiste en un mecanismo de agregación paso a paso,

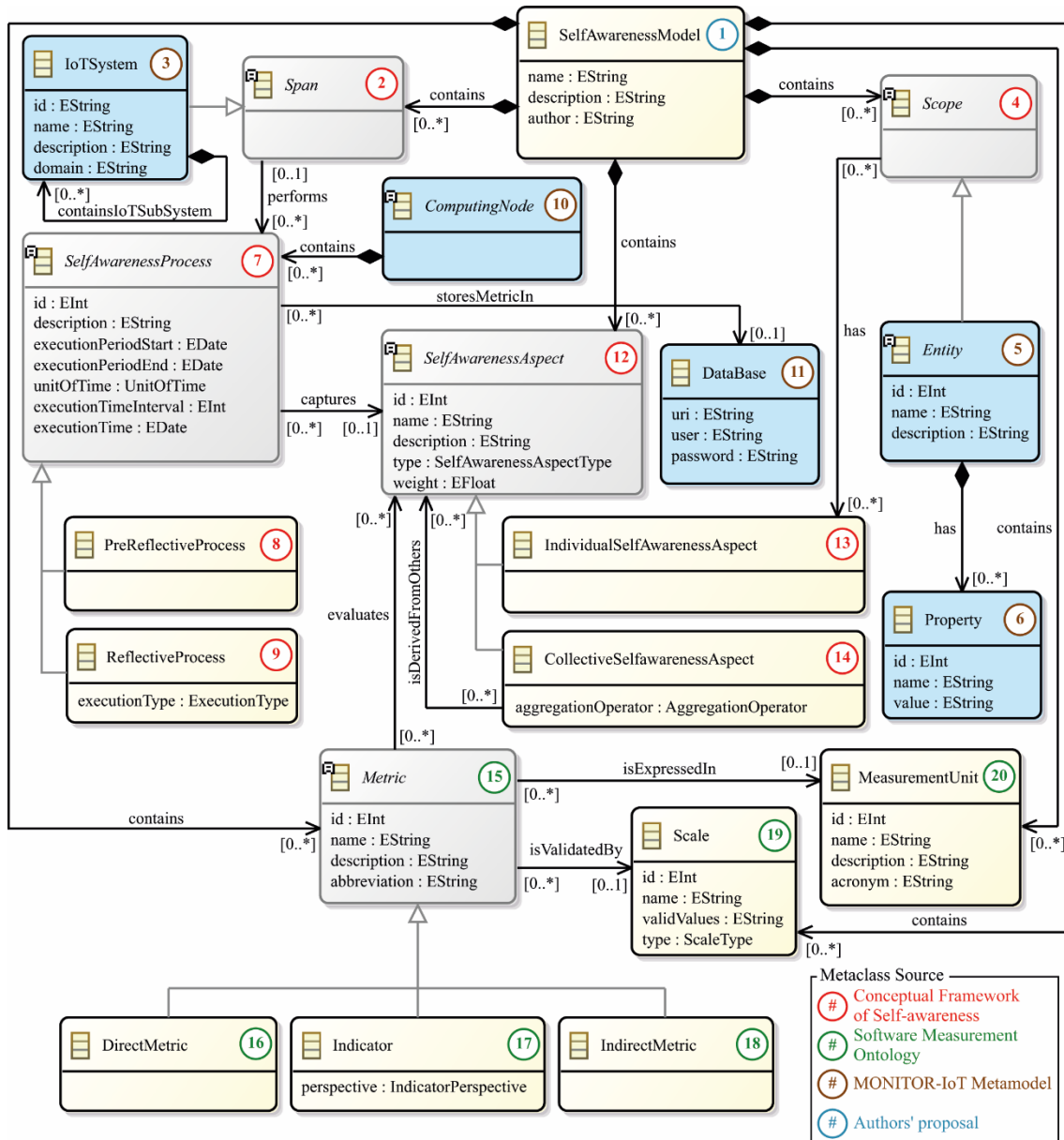


Figura B-1. Metamodelo Aware-IoT (Parte 1), incluye metaclasses y relaciones para representar el dominio (sujetos, objetos, aspectos) de autoconsciencia, los procesos de autoconsciencia y los tipos de métricas. Fuente: Elaboración propia.

de abajo hacia arriba (estrategia Botton – Up), donde los indicadores cuantitativos obtenidos para los aspectos de autoconsciencia (sean individuales o colectivos) de más bajo nivel, se agrupan convenientemente utilizando operadores de agregación del modelo LSP (atributo y enumeración *aggregationOperator*) para producir los indicadores de los aspectos de autoconsciencia colectivos de nivel superior. Además, para este cálculo el metamodelo utiliza el nivel de importancia de los aspectos de autoconsciencia (atributo *weight*). Ejemplos de aspectos de autoconsciencia colectivos pueden ser: conocer el nivel de corrección funcional de un sistema IoT con respecto al control ambiental de la temperatura, humedad, CO y humo de una casa de un adulto mayor; o, el nivel de consumo promedio de las baterías de todos los sensores de un sistema IoT (ver Fig. 5-6(13) y B-1(14)).

SelfAwarenessProcess. Representa un proceso ejecutado por un sujeto y que tiene como propósito capturar el conocimiento sobre un aspecto de autoconsciencia individual o colectivo, a través de la aplicación de métodos de autoconsciencia (recolección, cálculo y modelos de análisis). El metamodelo permite establecer el periodo de tiempo durante el cual se ejecutará un proceso de autoconsciencia (atributos *executionPeriodStart* y *executionPeriodEnd*), así como su frecuencia u hora de ejecución (atributos *unitOfTime*, *executionTimeInterval* y *executionTime*). Además, los procesos de autoconsciencia pueden ejecutarse de manera distribuida; de ahí que, a partir de la información contenida en el modelo de arquitectura [56], el metamodelo permite definir el nodo de computación del sujeto de autoconsciencia (metaclase *ComputingNode*), en donde se implementará y ejecutará periódicamente cada proceso de autoconsciencia en segundo plano (demonio). Asimismo, el metamodelo permite definir la base de datos (metaclase *DataBase*) en donde se almacenarán las métricas, metadatos y recomendaciones resultantes de la ejecución de cada proceso. Por último, los procesos de autoconsciencia se especializan en: pre-reflexivos y reflexivos (ver Fig. 5-6(7, 10, 14), B-1(7, 10, 11) y B-3(9)).

PreReflectiveProcess. Establece un proceso pre-reflexivo que desarrolla en el sujeto una consciencia subjetiva basada únicamente en información sensorial (observación inmediata). Por lo tanto, estos procesos utilizan métodos de recolección de datos empíricos desde sensores u otras fuentes para producir métricas directas; y, consecuentemente modelos de análisis basados en criterios de decisión, los cuales son aplicados a las métricas directas para producir indicadores cuantitativos (porcentual) y cualitativos sobre los aspectos de autoconsciencia, con las respectivas recomendaciones prescriptivas, en caso de existir las (ver Fig. 5-6(8) y B-1(8)).

ReflectiveProcess. Define un proceso reflexivo que construye y/o aplica en el sujeto un modelo conceptual interno, a partir de la información histórica (métricas directas, indirectas) generada por otros procesos pre-reflexivos o reflexivos, con el propósito de obtener nuevas métricas indirectas; y, sus correspondientes indicadores. En este sentido, los procesos reflexivos emplean métodos de cálculo para producir las métricas indirectas; y, de igual manera aplican modelos de análisis sobre las métricas indirectas obtenidas para generar indicadores cuantitativos (porcentual) y cualitativos sobre los aspectos de autoconsciencia, con las respectivas recomendaciones prescriptivas, en caso de existir las. En particular, los procesos reflexivos relacionados con aspectos de autoconsciencia colectivos utilizan únicamente los modelos de análisis, ya que deben calcular directamente sus indicadores cuantitativos a partir de los indicadores de los aspectos de más bajo nivel, mediante la aplicación del Modelo LSP [170]. De ahí que, estos procesos sobre aspectos colectivos se pueden ejecutar de dos formas (atributo *executionType*): i) automáticamente, ante cualquier cambio en los indicadores de los aspectos relacionados de más bajo nivel; o, ii) predefinida, según los valores fijados en el diseño del modelo

para los atributos frecuencia o fecha de ejecución (atributos *unitOfTime*, *executionTimeInterval* y *executionTime*) (ver Fig. 5-6(9) y B-1(9)).

Metric. Define una forma de medir y evaluar un aspecto de autoconsciencia, así como su escala (metaclase *escala*) y unidad de medida (metaclase *MeasurementUnit*). Las métricas pueden especializarse en: métricas directas, indirectas e indicadores (ver Fig. 5-6(17, 21, 22), B-1(15, 19, 20), B-2(4) y B-3(10)).

DirectMetric. Medida de una propiedad de un objeto de autoconsciencia, cuya medición no depende de otras métricas, sino más bien se obtiene directamente de los datos empíricos o crudos (en estado primario) recolectados desde los sensores u otras fuentes; es decir, las métricas directas se obtienen mediante la aplicación de los métodos de recolección. Las métricas directas son la base para medir las métricas indirectas y los indicadores. Ejemplos de métricas directas pueden ser: partes por millón de concentración de monóxido de carbono (CO) en el ambiente de una casa (propiedad de telemetría), el porcentaje de uso (carga) de la CPU de una puerta de enlace en un momento determinado (propiedad de estado), entre otros (ver Fig. 5-6(18), B-1(16) y B-2(5)).

IndirectMetric. Medida de una propiedad derivada de otras métricas directas e indirectas, mediante el empleo de un método de cálculo que construye y/o aplica un modelo conceptual - relacional entre métricas. Ejemplos de métricas indirectas pueden ser: partes por millón de concentración promedio diario de monóxido de carbono en el ambiente de una casa, porcentaje de uso (carga) estimado o predicho de la CPU de una puerta de enlace para una determinada cantidad de sensores o actuadores, entre otros (ver Fig. 5-6(20), B-1(18) y B-2(7)).

Indicator. Medida que evalúa o valora un aspecto de autoconsciencia individual o colectivo, a fin de determinar la situación actual o predecir la posible evolución futura ante ciertas condiciones de un sistema IoT. Para obtener un indicador se emplea un modelo de análisis que relaciona y compara los valores de otras métricas (directas, indirecta o indicadores) con los criterios de decisión (umbrales) asociados al modelo, a fin de obtener una valoración cuantitativa (porcentual) y cualitativa sobre un aspecto de autoconsciencia. Un indicador puede ser descrito en sentido positivo o negativo (atributo *perspective*). Ejemplos de indicadores pueden ser: el nivel de riesgo de la concentración de monóxido de carbono para un usuario, el nivel de uso estimado o predicho de la CPU de una puerta de enlace para una determinada cantidad de sensores o actuadores, entre otros (ver Fig. 5-6(19), B-1(17) y B-2(6)).

LearningReasoningMethod. Representa un método que define cómo un proceso de autoconsciencia debe ejecutarse para aprender y capturar el conocimiento sobre un aspecto; y, a su vez, razonar sobre el mismo. El metamodelo soporta métodos que van desde la recolección u observación de datos empíricos a partir de sensores u otras fuentes, pasando por métodos de cálculo para construir y/o aplicar modelos reflexivos sobre los aspectos de autoconsciencia, hasta modelos de análisis que en función de las métricas calculadas y criterios de decisión producen indicadores que determinan la situación actual o posible evolución futura de un sistema IoT (ver Fig. 5-6(23), B-2(8) y B-3(2)).

CollectionMethod. Define un método para recopilar los datos empíricos de una propiedad perteneciente a un objeto de autoconsciencia, produciendo como resultado una métrica directa. Los métodos de recolección dan soporte a los procesos pre-reflexivos, constituyéndose en el aparato sensorial capaz de realizar observaciones. Para operacionalizar los métodos de recolección se utilizan los flujos de datos instanciados a través del modelo de arquitectura de monitoreo del sistema IoT observado [56]. La

metaclase *DataFlow*, incluida en el metamodelo de arquitectura de monitoreo (Monitor-IoT), es una abstracción de alto nivel que describe cómo los datos crudos recolectados desde una fuente son transportados a través de diferentes nodos y recursos de computación hasta el almacén de datos. Como se ha mencionado, estos flujos de datos pueden ser síncronos o asíncronos. A manera de ejemplos, los métodos de recolección pueden ser: monitorear la concentración de monóxido de carbono (CO) en el ambiente de una casa o la carga de procesamiento de la CPU de una puerta de enlace (ver Fig. 5-6(24, 25) y B-2(9-11)).

CalculationMethod. Especifica un método conformado por un conjunto de operaciones e instrucciones de cálculo, cuyo objetivo es a partir de los datos recopilados históricamente, construir y/o aplicar un modelo conceptual reflexivo que obtenga como

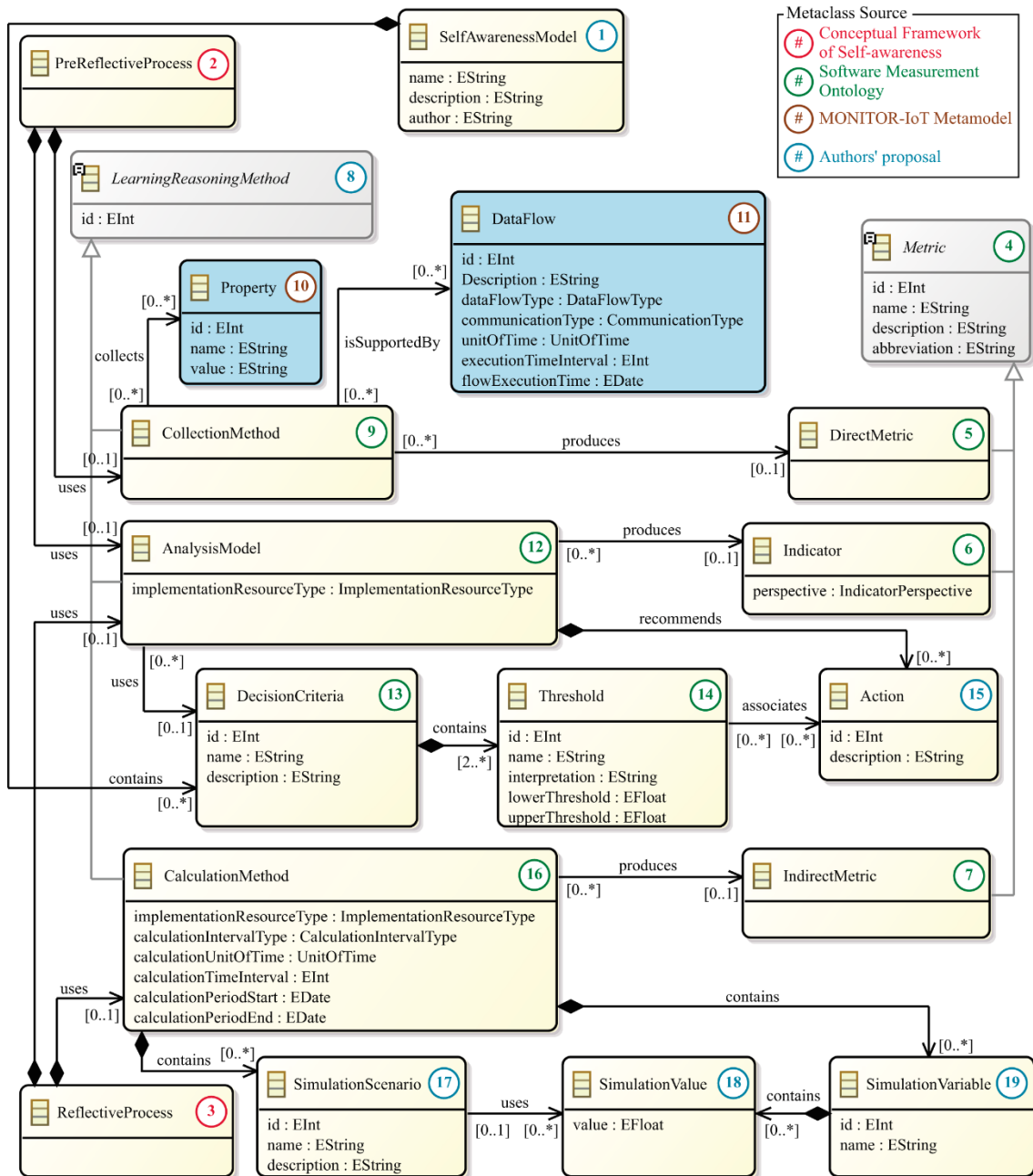


Figura B-2. Metamodelo Aware-IoT (Parte 2), incluye metaclases y relaciones para representar los métodos de autoconsciencia (aprendizaje y razonamiento). Fuente: Elaboración propia.

resultado una métrica indirecta para una propiedad de un objeto de autoconsciencia; es decir, dan soporte a los procesos reflexivos. Los modelos reflexivos pueden ser contruidos en tiempo de ejecución o especificados explícitamente durante el tiempo de diseño. Los datos requeridos por un método de cálculo para crear y/o ejecutar un modelo reflexivo son definidos en el metamodelo mediante argumentos, los cuales pueden ser: métricas (directas o indirectas) calculadas anteriormente, metadatos que contiene información para describir o contextualizar las métricas; y, variables de simulación. Para operacionalizar los métodos de cálculo se reutilizan los recursos de implementación genéricos (fórmulas, funciones o servicios web), los cuales encapsulan las operaciones e instrucciones de cálculo. Los argumentos definidos en los métodos de cálculo representan los valores que pasan como parámetros a los recursos de implementación genéricos para ser utilizados durante la ejecución de las operaciones o instrucciones encapsuladas. Adicionalmente, los métodos de cálculo pueden soportar la definición de escenarios de simulación que establecen valores de análisis para una o más variables dependientes del modelo reflexivo (metaclases *SimulationScenario*, *SimulationVariable* y *SimulationValue*). Los escenarios de simulación permiten predecir a través del modelo reflexivo el estado de un objeto de autoconsciencia ante determinados cambios (variables de simulación). Por ejemplo, mediante un modelo de regresión de mínimos cuadrados se puede predecir el nivel de uso (carga) de la CPU de una puerta de enlace, en caso de que la cantidad de sensores a controlar (variable de simulación) se requiera incrementar al doble (ver Fig. 5-6(30-33), B-2(16-19) y B-3(4)).

AnalysisModel. Define un método conformado por un conjunto de operaciones o instrucciones, cuyo propósito es relacionar y comparar una o más métricas previamente calculadas (directas, indirectas e indicadores) con un criterio de decisión (compuesto de umbrales) asociado, obteniéndose como resultado un indicador. Por lo tanto, los modelos de análisis son utilizados tanto por los procesos pre-reflexivos como reflexivos. Al igual que los métodos de cálculo, para operacionalizar los modelos de análisis se invocan a los recursos de implementación genéricos (fórmulas, funciones o servicios web), pasando como parámetros los argumentos (métricas) establecidos en el modelo de análisis. En particular, los modelos de análisis de procesos reflexivos relacionados con aspectos de autoconsciencia colectivos se operacionalizan mediante la llamada a un método disponible en la infraestructura, el cual ejecuta el modelo LSP [170] para obtener los indicadores cuantitativos de este tipo de aspectos. El algoritmo LSP calcula los indicadores en función de la estructura jerárquica entre los aspectos de autoconsciencia que soporta el metamodelo; y, los operadores de agregación y pesos (atributos *aggregationOperator*, *weight*) que permite asignar el metamodelo a los aspectos de autoconsciencia. Los modelos de análisis utilizan un criterio de decisión compuesto por umbrales (metaclases *DecisionCriteria* y *Threshold*) para valorar cualitativamente un aspecto de autoconsciencia, a fin de determinar su situación actual o posible evolución futura, así como la necesidad de una acción posterior. En este sentido, a los umbrales se pueden asociar recomendaciones o acciones (metaclase *Action*), en caso de ser necesario. Como ejemplos, se pueden obtener indicadores para determinar si la concentración de monóxido de carbono (CO) en el ambiente de una casa representa un riesgo alto, normal o bajo para un usuario; si el nivel de uso de la CPU de una puerta de enlace se encuentra en un estado normal o de sobrecarga; o, si el nivel de consumo promedio de la batería de todos los sensores de un sistema IoT es satisfactorio, marginal o insatisfactorio (ver Fig. 5-6(26-29), B-2(12-15) y B-3(3)).

ImplementationResource. Define un recurso genérico que encapsula e implementa una operación o algoritmo de cálculo para ser reutilizado por los métodos de cálculo o

modelos de análisis. Los recursos de implementación genéricos pueden ser: fórmulas, funciones o servicios web (metaclases *Formula*, *Function* y *Service*). Con respecto a las fórmulas, el atributo *expresión* contiene su formulación matemática. Mientras que para las funciones y servicios web, el atributo *instructions* encapsula la secuencia ordenada de instrucciones a ejecutar. Los servicios web son publicados en los middlewares (metaclase *middleware*) de los nodos de computación del sistema IoT y pueden ser accedidos a través de sus endpoints (atributo *endPoint*). Además, el metamodelo permite especificar los parámetros (metaclase *Parameter*) o datos de entrada que son utilizados por los recursos

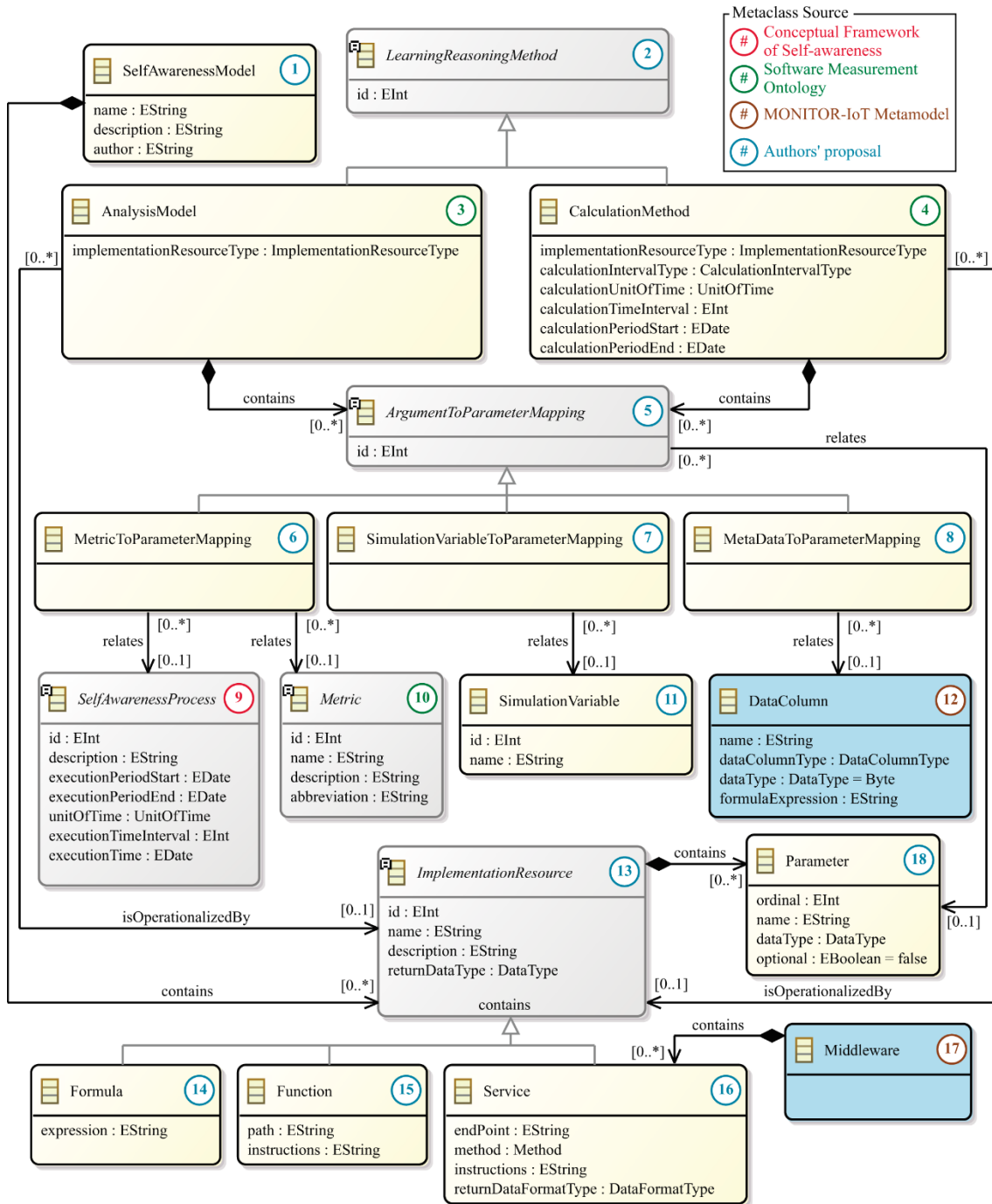


Figura B-3. Metamodelo Aware-IoT (Parte 3), incluye metaclases y relaciones para representar los métodos de autoconsciencia, los recursos de implementación y el mapeo de parámetros entre estos. Fuente: Elaboración propia.

de implementación para producir la información de salida. Finalmente, el metamodelo soporta la definición de tipos y formatos de datos, tanto para los parámetros de entrada como para la información de salida. Ejemplos de recursos de implementación genéricos pueden ser: una fórmula para calcular el promedio de un conjunto de valores históricos, una función o servicio web que construye un modelo de regresión por mínimos cuadrados para la predicción en nuevos escenarios, entre otros (ver Fig. 5-6(34-39) y B-3(13-18)).

ArgumentToParameterMapping. Metaclase abstracta que mapea los argumentos de los métodos de cálculo o modelos de análisis con los parámetros de los recursos de implementación genéricos. Esta metaclase se puede especializar en: i) *MetricToParameterMapping*, mapea una métrica (directa, indirecta, indicador) producida por un proceso de autoconsciencia con un parámetro; ii) *SimulationVariableToParameterMapping*, mapea una variable de simulación con un parámetro, con el propósito de realizar predicciones; y, iii) *MetaDataToParameterMapping*, mapea un metadato con un parámetro, a fin de suministrar información adicional sobre las métricas, por ejemplo, la fecha y hora en la que obtuvo la métrica, el aspecto u objeto de autoconsciencia relacionado con la métrica (ver Fig. 5-6(40-43) y B-3(5-12)).

Finalmente, la Figura B-4 presenta las enumeraciones utilizadas por los atributos de las metaclases de Aware-IoT.

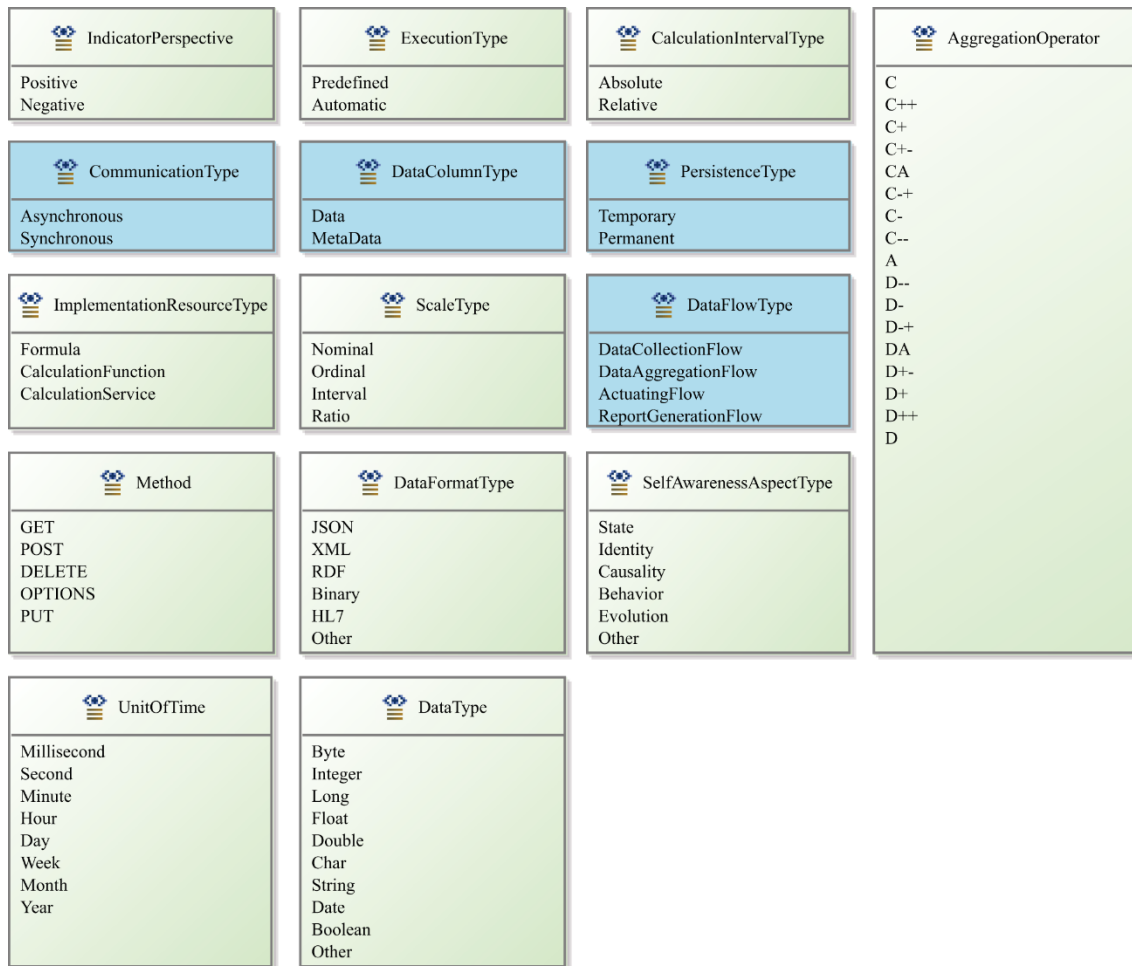


Figura B-4. Metamodelo Aware-IoT (Parte 4), incluye las enumeraciones utilizadas por los atributos de las metaclases. Fuente: Elaboración propia.

Anexo C

Modelo de Arquitectura de Monitoreo del Estudio de Caso

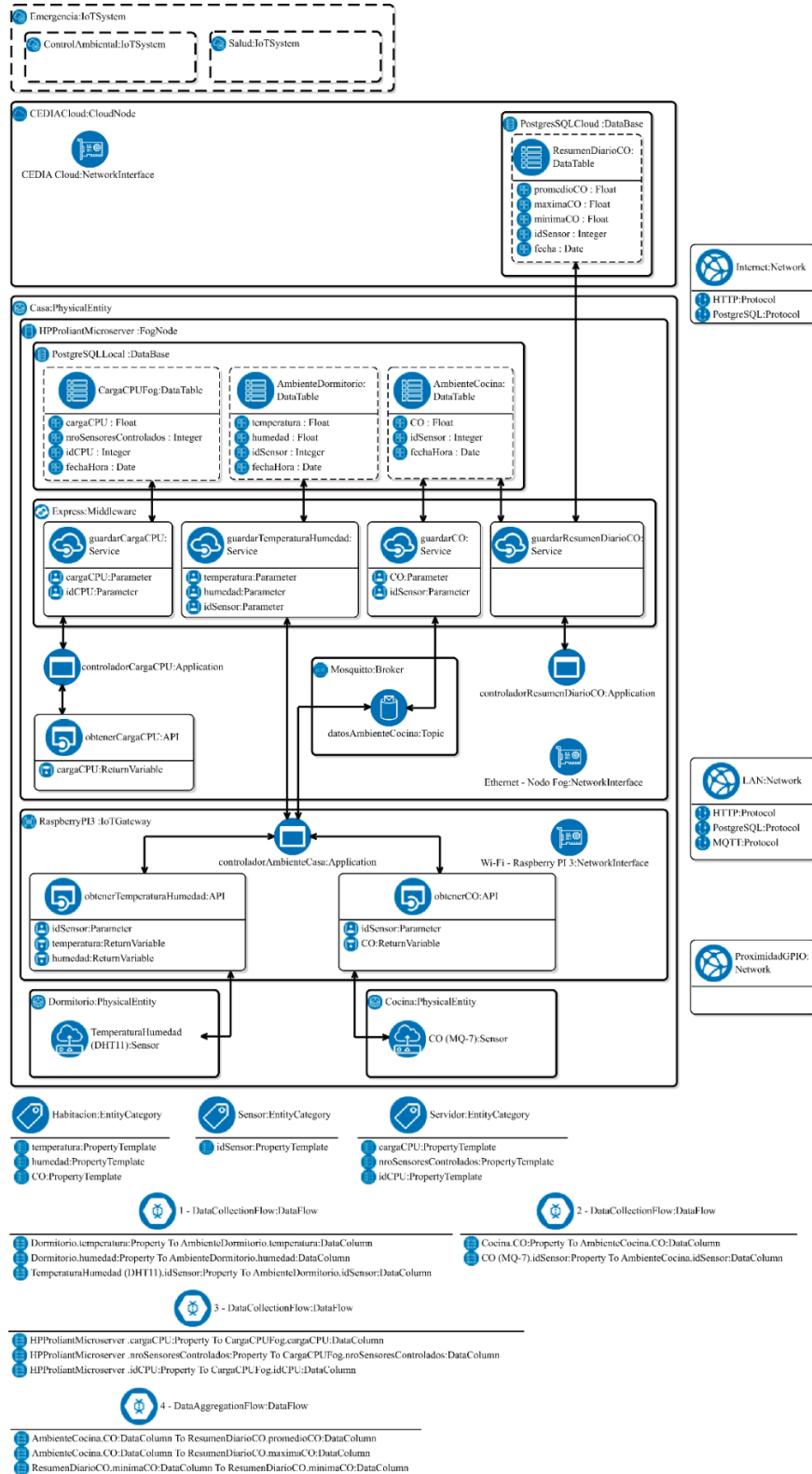


Figura C-1. Modelo de arquitectura de monitoreo resultante del estudio de caso.
Fuente: Elaboración propia.

Anexo D

Boletín y Hoja de Trabajo del Cuasi Experimento Nro. 1

1. Objetivo de la evaluación empírica (cuasi experimento)

El presente cuasi experimento se enmarca dentro una tesis doctoral que tiene como objetivo proponer una metodología (Aware-IoT@Run.Time) basada en modelos en tiempo de ejecución para la construcción, operación y mantenimiento de sistemas IoT autoconscientes. Específicamente, el propósito de este cuasi experimento es evaluar la primera actividad de la metodología Aware-IoT@Run.Time, la cual tiene como alcance el diseño visual de la arquitectura de monitoreo de un sistema IoT, con el apoyo de la herramienta DSL Monitor-IoT. Los resultados del cuasi experimento permitirán determinar el rendimiento (efectividad y eficiencia) de los participantes en el uso del DSL Monitor-IoT y su proceso metodológico, así como comprender la percepción de los mismos sobre la facilidad de uso, utilidad e intención de uso futuro de esta solución.

2. Especificación de requerimientos del escenario de IoT para el ejercicio experimental

En un ámbito general, el escenario de IoT propuesto se orienta al subdominio de ambientes de vida asistida y tiene como objetivo **diseñar una arquitectura de monitoreo para un Sistema de Gestión de Emergencias (SGE) orientado a adultos mayores que viven solos en su hogar**. El sistema se encuentra conformado por dos subsistemas: i) un **Subsistema de Control Ambiental (SCA)**, encargado de monitorear la temperatura, humedad y la presencia de monóxido de carbono (CO) en el ambiente de una casa; y, ii) un **Subsistema de Salud (SS)** para monitorear la frecuencia cardíaca de un adulto mayor.

En particular, este **ejercicio experimental** tiene como alcance únicamente el **diseño de la arquitectura de monitoreo del Subsistema de Salud**. Para ello, el escenario incluye la utilización de varios dispositivos y nodos de computación a nivel de las capas Edge, Fog y Cloud (ver Fig. D-1), los cuales se describen en las siguientes subsecciones:

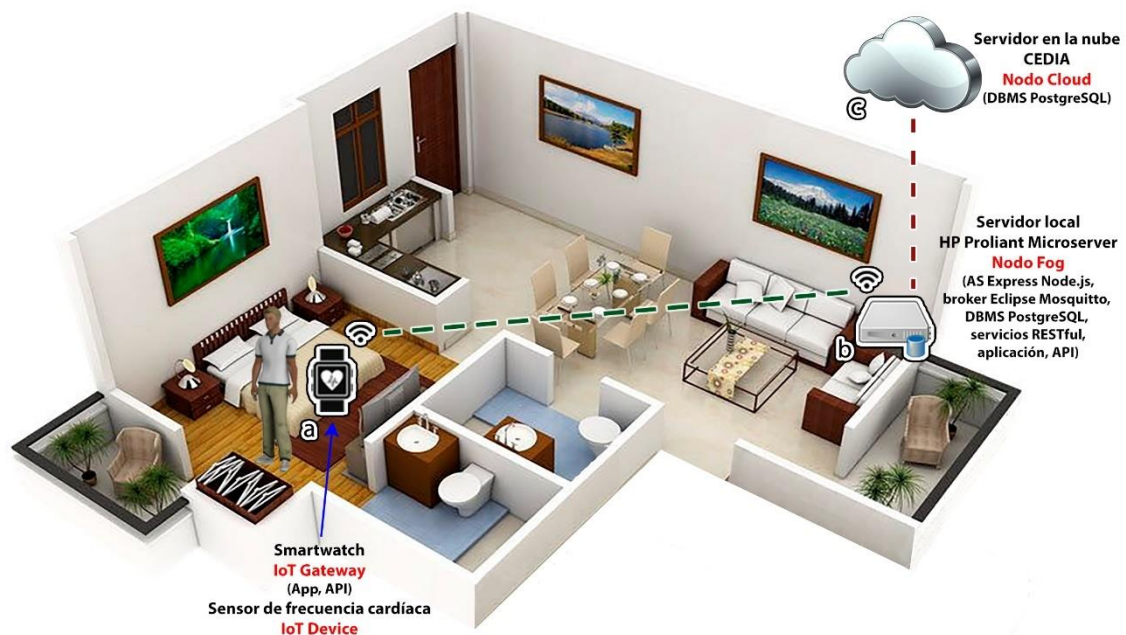


Figura D-1. Escenario de IoT para el ejercicio experimental.

Fuente: Elaboración propia.

2.1. Capa Edge

Para cumplir con los requerimientos de monitoreo, la plataforma IoT incluye los siguientes dispositivos IoT:

- i) **Un sensor de frecuencia cardíaca** (ver Fig. D-1 (a)) incorporado en un **smartwatch** que utiliza el adulto mayor, el cual mide el número de pulsaciones del corazón del usuario cada **30 minutos**.
- ii) Además, el **smartwatch** (ver Fig. D-1 (a)) actúa como una **puerta de enlace (IoT Gateway)** que ejecuta una **App (ControladorSalud)** que llama a una **API (obtenerFrecuenciaCardiaca)** para recuperar los datos del **sensor de frecuencia cardíaca**. Adicionalmente, la App envía de **forma asíncrona** los datos recolectados a un servidor local ubicado en la casa del usuario (**nodo Fog**), utilizando como intermediario un **broker Eclipse Mosquitto**.

La **puerta de enlace (smartwatch)** y el **nodo Fog** se comunican mediante una **red de área local Wifi** empleando el **protocolo de aplicación MQTT (comunicación asíncrona)**.

2.2. Capa Fog

Como ya se mencionó, esta capa contiene un **servidor (HP Proliant Microserver, IP: 192.168.1.11)** ubicado en la casa del adulto mayor, con capacidades de procesamiento y almacenamiento local (ver Fig. D-1 (b)). El nodo Fog incluye los siguientes recursos:

- i) Un **sistema de gestión de base de datos PostgreSQL (puerto 5432)**, cuyo objetivo es almacenar los datos crudos provenientes del sensor de frecuencia cardíaca. Esta base de datos contiene una **tabla (Salud)** para almacenar la frecuencia cardíaca del adulto mayor (**frecuenciaCardiaca**), la fecha y hora en que se monitorea el dato (**fechaHora**); y, el identificador del usuario que es monitoreado (**idUsuario**).
- ii) Un **broker Eclipse Mosquitto (puerto 1883)** que implementa un modelo de transferencia **asíncrona** de datos (modelo publicación / suscripción) entre el **smartwatch** y el **nodo Fog**. El **broker** contiene un tópico (**datosSalud**) en el que se publican los datos recopilados.
- iii) Un **servidor de aplicaciones creado mediante el framework Express de Node.js (puerto 9999)** que contiene **servicios RESTful** con funcionalidades de intercambio, agregación y almacenamiento de datos.
- iv) Una **aplicación de software (controladorResumenDiarioSalud)** que invoca una vez al día (23h55) a un **servicio RESTful (guardarResumenDiarioSalud)** para almacenar el resumen (promedio, máximo, mínimo) diario de la frecuencia cardíaca del adulto mayor en una base de datos del nodo Cloud.
- v) Un **servicio RESTful (guardarFrecuenciaCardiaca)** que es consumido desde un evento asociado al broker, con el propósito de que los datos de la frecuencia cardíaca publicados en el tópico respectivo (**datosSalud**) sean almacenados en la **base de datos ubicada en el nodo Fog** (soporta flujos de recolección de datos asíncronos).
- vi) Un **servicio RESTful (guardarResumenDiarioSalud)** que calcule el valor promedio, máximo y mínimo diario de la frecuencia cardíaca del adulto mayor; y, los datos resultantes los almacena en la **base de datos del nodo Cloud** (soporta flujos de agregación de datos).

2.3. Capa Cloud

El escenario de IoT propuesto requiere de un **nodo Cloud (CEDIACloud, IP: 172.16.1.10)** implementado en la **plataforma de CEDIA** (ver Fig. D-1 (c)) que incluye:

- i) Un **sistema de gestión de base de datos PostgreSQL (puerto 5432)** para almacenar los datos resultantes de las operaciones de agregación ejecutadas en el nodo Fog. Esta base de datos contiene una tabla (**ResumenDiarioSalud**) para almacenar la frecuencia cardíaca promedio, máxima y mínima diaria del adulto mayor (**promedioFrecuenciaCardiaca**, **maximaFrecuenciaCardiaca**, **minimaFrecuenciaCardiaca**), el identificador del usuario (**IdUsuario**) que es monitoreado; y, la fecha de la medición (**fecha**).

La comunicación entre los nodos Fog y Cloud es a través de la **red de Internet**.

3. Prerrequisitos para la ejecución del ejercicio experimental

- Descargue e instale Java SE Development Kit 8, el cual se puede obtener de la siguiente URL: <https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>
- Descargue, configure y ejecute el DSL Monitor-IoT bajo la plataforma Obeo Designer y Eclipse Sirius, el cual se puede obtener de la siguiente URL: <https://drive.google.com/file/d/11ME7pEz0x1HilqJtw0oBXqF-W5thU6iJ/view?usp=sharing>

4. Hoja de trabajo del ejercicio experimental

a. Datos del participante

Nombres y apellidos:

Institución:

b. Tareas a completar para el diseño de la arquitectura de monitoreo del Subsistema de Salud (SS)

El proceso a seguir en este cuasi experimento para diseñar la arquitectura de monitoreo del subsistema SS, comprende la Actividad 1 de la metodología Aware-IoT@Run.Time, la cual se subdivide en tres tareas principales: **i) identificación de subsistemas, entidades, propiedades y flujos de datos; ii) modelado de subsistemas, entidades y propiedades; y, iii) modelado de flujos de datos**. En el sitio web del experimento se encuentra: i) una guía con la descripción del proceso metodológico Aware-IoT@Run.Time para diseñar arquitecturas de monitoreo para IoT con el apoyo del DSL Monitor IoT, incluyendo el detalle de sus actividades, tareas y artefactos; y, ii) un documento con la descripción de la paleta de herramientas del DSL Monitor-IoT.

Tarea 1. Identificación de subsistemas, entidades, propiedades y flujos de datos

El propósito de esta tarea es analizar los requerimientos del escenario de IoT y preparar la documentación requerida (subsistemas, entidades físicas y digitales, propiedades, flujos de datos) por la herramienta DSL Monitor-IoT para el modelado de la arquitectura de monitoreo. En este sentido, para identificar y organizar adecuadamente esta documentación se dispone de un conjunto de plantillas (ver Tablas de la D-1 a la D-17). Tenga en consideración que, según el alcance de este ejercicio experimental, solo se requiere llenar los campos de color rojo de las plantillas. Finalmente, esta tarea se divide en cuatro subtareas que se deben ejecutar secuencialmente:

Registre la hora de inicio de la Tarea 1:

Subtarea 1.1. Descomposición del sistema IoT

Defina la estructura jerárquica del sistema IoT, descomponiéndole en subsistemas. A su vez, especifique los siguientes datos para cada sistema o subsistema: sigla (identificador), nombre, descripción y dominio de aplicación al que pertenece.

Estructura jerárquica del sistema IoT

Tabla D-1. Plantilla para registrar los sistemas y subsistemas del escenario de IoT.

Id. (Sigla)	Nombre	Descripción	Dominio

Subtarea 1.2. Identificación de entidades

Especifique las entidades a monitorear, así como las entidades digitales (nodos de computación y dispositivos IoT), los recursos de hardware / software (p. ej., interfaces de red, middleware, servicios, APIs, aplicaciones, base de datos, tópicos de broker) y las redes de comunicación que son necesarios para soportar los procesos de monitorización (recolección, transporte, procesamiento y almacenamiento), a nivel de las capas Cloud, Fog y Edge.

Tabla D-2. Plantilla para registrar las especificaciones de las entidades físicas.

Entidad física	Descripción	Atributos	
		Sistema o Subsistema que lo utiliza	Tipo de entidad (metaclass)
			Elija un elemento.

Tabla D-3. Plantilla para registrar las especificaciones de los dispositivos IoT (sensores).

Dispositivo IoT	Descripción	Atributos	
		Sistema o subsistema que lo utiliza	Tipo de sensor
			Elija un elemento.

Tabla D-4. Plantilla para registrar las especificaciones de las puertas de enlace IoT.

Puerta de enlace (Gateway) de IoT	Descripción	Atributos	
		Sistema o subsistema que lo utiliza	Tipo de Gateway
			Elija un elemento.

Tabla D-5. Plantilla para registrar las especificaciones de los nodos Fog.

Nodo Fog	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Modelo de servicio	Modelo de despliegue	Plataforma
			Elija un elemento.	Elija un elemento.	

Tabla D-6. Plantilla para registrar las especificaciones de los nodos Cloud.

Nodo Cloud	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Modelo de servicio	Modelo de despliegue	Plataforma
			Elija un elemento.	Elija un elemento.	

Tabla D-7. Plantilla para registrar las especificaciones de las APIs.

Tabla 5. Formulario para registrar las especificaciones de las APIs.		
API	Descripción	Atributos
		Sistema o subsistema que lo utiliza:
		Nodo de computación en el que opera:
		Parámetros de entrada:
		Instrucciones que encapsula:
		Variables de salida:
		Tipo de formato de los datos de salida: Elija un elemento.

Tabla D-8. Plantilla para registrar las especificaciones de las aplicaciones.

Aplicación	Descripción	Atributos		
		Sistema o subsistema que lo utiliza	Nodo de computación en el que opera	Tipo de aplicación
				Elija un elemento.

Tabla D-9. Plantilla para registrar las especificaciones de los middlewares.

Middleware	Descripción	Atributos		
		Sistema o subsistema que lo utiliza	Nodo de computación en el que opera	Protocolo que usa

Tabla D-10. Plantilla para registrar las especificaciones de los servicios web.

Servicio web	Descripción	Atributos
		Sistema o subsistema que lo utiliza:
		Nodo de computación en el que opera:
		Endpoint:
		Método: Elija un elemento.
		Tipo de formato de los datos de salida: Elija un elemento.
		Parámetros de entrada:
		Variables de salida:

Tabla D-11. Plantilla para registrar las especificaciones de las bases de datos.

Base de datos	Descripción	Atributos
		Sistema o subsistema que lo utiliza:
		Nodo de computación en el que opera:
		Uri:
		User:
		Password:
		Protocolo que usa:

Tabla D-12. Plantilla para registrar las especificaciones de las tablas de base de datos.

Tabla:			
Descripción		Propiedades de tabla	
		Sistema o subsistema que lo utiliza:	
		Base de datos que almacena:	
		Tipo de persistencia: Elija un elemento.	
		Unidad de tiempo persistencia: Elija un elemento.	
		Tiempo de persistencia:	
Columnas			
Nombre	Tipo de columna	Tipo de dato	Fórmula/Instrucciones
	Elija un elemento.	Elija un elemento.	

Tabla D-13. Plantilla para registrar las especificaciones de los tópicos de broker.

Tópico	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Broker en el que se publica	Tipo de tópico	Tipo de formado de dato
				Elija un elemento.	Elija un elemento.

Tabla D-14. Plantilla para registrar las especificaciones de las redes de comunicación y protocolos.

Red de comunicación	Descripción	Atributos		
		Sistema o subsistema que lo utiliza	Protocolo	Puerto

Tabla D-15. Plantilla para registrar las especificaciones de las interfaces de red.

Interfaz de red	Descripción	Atributos			
		Sistema o subsistema que lo utiliza	Red a la que se conecta	Network Address	Mac Address

Subtarea 1.3. Identificación de propiedades

Especifique las propiedades a ser monitoreadas o utilizadas como metadatos para cada entidad identificada.

Tabla D-16. Plantilla para registrar las especificaciones de las propiedades.

Propiedad	Definición	Atributos					
		Entidad física o digital relacionada	Identificador	Tipo de propiedad	Unidad de medida	Tipo de dato	Valor asignado en tiempo de diseño
				Elija un elemento.	Elija un elemento.	Elija un elemento.	

Subtarea 1.4. Identificación de flujos de datos

Especifique los flujos de datos a utilizar para transportar los datos desde las fuentes (sensores) hasta su ubicación de almacenamiento con diferentes niveles de agregación (bases de datos).

Tabla D-17. Plantilla para registrar las especificaciones de los flujos de datos.

Flujo de dato	Atributos				
	Tipo de flujo de dato	Tipo de comunicación	Unidad de tiempo	Frecuencia de ejecución del flujo de datos	Hora de ejecución del flujo de datos (opcional)
	Elija un elemento.	Elija un elemento.	Elija un elemento.		

Registre la hora de fin de la Tarea 1:

Tarea 2. Modelado de subsistemas, entidades y propiedades

Primera tarea de modelado de la arquitectura de monitoreo, la cual utiliza la herramienta DSL Monitor-IoT y la documentación generada en la Tarea 1. Particularmente, se enfoca en modelar la estructura jerárquica entre el sistema y sus subsistemas, diseñar la estructura entre las entidades físicas y digitales (nodos de computación y sus recursos) y configurar las propiedades a monitorear para cada entidad. Esta tarea se divide en siete subtareas, varias de las cuales se ejecutan en paralelo:

Registre la hora de inicio de la Tarea 2:

Subtarea 2.1. Modelado de subsistemas

Diseñe la estructura jerárquica entre el sistema IoT y sus subsistemas. A su vez, asigne cada entidad física o digital al subsistema respectivo.

Subtarea 2.2. Modelado de entidades físicas

Diseñe la estructura jerárquica entre las entidades físicas identificadas en la Tarea 1.

Subtarea 2.3. Modelado de nodos de computación a nivel Edge, Fog y Cloud

Agregue al modelo los nodos Cloud, nodos Fog, puertas de enlace y/o dispositivos IoT (sensores) especificados en la Tarea 1. Los nodos de computación pueden estar contenidos dentro de las entidades físicas para dotarles de capacidades de detección, procesamiento o almacenamiento.

Subtarea 2.4. Modelado de recursos utilizados por los nodos de computación

Incorpore al modelo los recursos de software y hardware (p. ej., middleware, base de datos, aplicaciones, APIs, servicios, tópicos de broker, interfaces de red) requeridos por los nodos de computación y que fueron especificados en la Tarea 1.

Subtarea 2.5. Modelado de redes y protocolos de comunicación

Agregue al modelo las redes y protocolos que soportan la comunicación síncrona o asíncrona entre las entidades digitales del sistema IoT. Además, asigna y configura las redes y protocolos a utilizar por cada uno de los recursos de software (p. ej., middlewares, bases de datos).

Tarea 2.6. Configuración de categorías de entidades

Agrupe en categorías, aquellas entidades del mismo tipo que comparten propiedades comunes a ser monitoreadas o utilizadas como metadatos.

Tarea 2.7. Configuración de propiedades de entidades

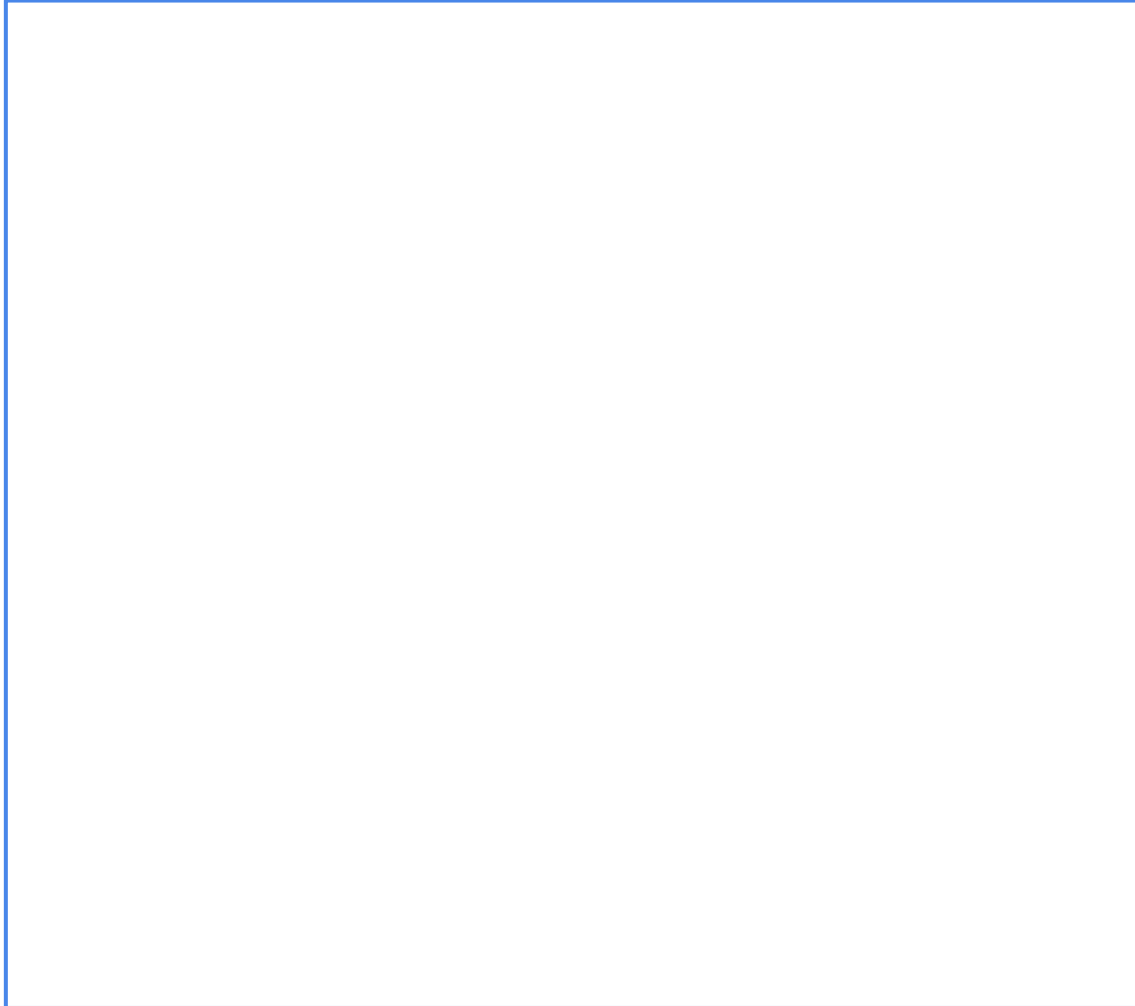
En primer lugar, cree las plantillas con la información general de las propiedades y APIs que son comunes para todas las entidades de una categoría. Luego, con base en las plantillas cree las

propiedades particulares a monitorear o ser utilizadas como metadatos para cada una de las entidades (físicas o digitales) modeladas previamente.

Registre la hora de fin de la Tarea 2:

**Modelo de arquitectura de monitoreo del Subsistema de Salud
(incluye subsistemas, entidades y propiedades)**

Una vez terminadas todas las subtareas de la Tarea 2, capture el modelo resultante desarrollado en el DSL Monitor-IoT y pegue en este recuadro.



Tarea 3. Modelado de flujo de datos

En esta última tarea se diseña la manera como interactúan las entidades digitales para soportar los flujos de datos a través de las capas Edge, Fog y Cloud. Para alcanzar este objetivo, esta tarea se subdivide en dos subtareas:

Registre la hora de inicio de la Tarea 3:

Subtarea 3.1. Mapeo de datos

Defina las reglas de mapeo, es decir, para cada flujo de datos, mapee las fuentes de datos (origen) con las ubicaciones de almacenamiento (destino).

Subtarea 3.2. Modelado de enlaces entre entidades digitales

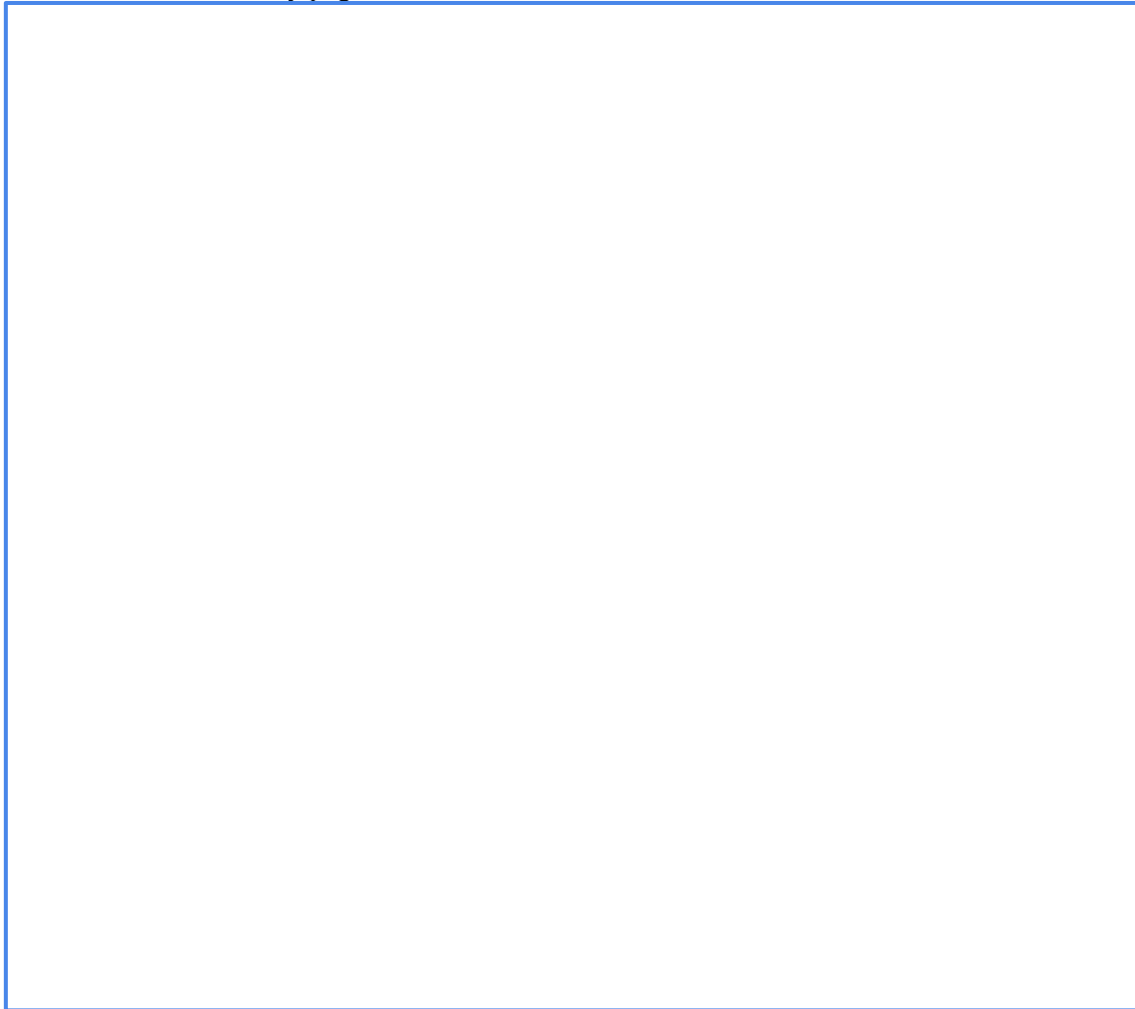
Diseñe una secuencia ordenada de enlaces entre entidades digitales para cada uno de los flujos identificados en la Tarea 1. La secuencia ordenada de enlaces determina la trayectoria de los

datos y la forma como interactúan los nodos de computación y los recursos que intervienen en un flujo de datos, con el objetivo de transportar los datos desde la fuente hasta su ubicación de almacenamiento lógico.

Registre la hora de fin de la Tarea 3:

**Modelo de arquitectura de monitoreo del Subsistema de Salud
(incluye reglas de mapeo, enlaces y flujos de datos)**

Una vez terminadas todas las subtareas de la Tarea 3, capture el modelo resultante desarrollado en el DSL Monitor-IoT y pegue en este recuadro.



c. Tareas finales del cuasi experimento

1. Complete la encuesta sobre el uso de Monitor-IoT para diseñar arquitecturas de monitoreo para sistemas IoT. La encuesta se encuentra en el siguiente enlace:
https://docs.google.com/forms/d/e/1FAIpQLScKjAymANajQw0oq_68fv-0g6hwlnV2LC8yB_7f48KHHvu0jQ/viewform?usp=sf_link
2. Entregue el **Formulario llenado del Ejercicio Experimental** y el **archivo resultante "EjercicioExperimental.MonitorIoT"** con el modelo de arquitectura de monitoreo del Subsistema de Salud al correo lerazo@uazuay.edu.ec.

¡¡GRACIAS POR SU COLABORACIÓN!!

Anexo E

Boletín y Hoja de Trabajo del Cuasi Experimento Nro. 2

1. Objetivo de la evaluación empírica (cuasi experimento)

El presente cuasi experimento forma parte de una tesis doctoral que tiene como objetivo proponer una metodología (Aware-IoT@Run.Time) basada en modelos en tiempo de ejecución para la construcción, operación y mantenimiento de sistemas IoT autoconscientes. Particularmente, este cuasi experimento se orienta a evaluar las actividades de la 2 a la 6 de la metodología Aware-IoT@Run.Time. La actividad 2 consiste en utilizar una aplicación web (configurador de autoconsciencia) para configurar las capacidades de autoconsciencia de un sistema IoT y generar un archivo JSON que integra los modelos de arquitectura y autoconsciencia de dicho sistema. Por su parte, las actividades de la 3 a la 6 se ejecutan automáticamente por medio de un middleware, el cual detecta el archivo JSON, analiza las especificaciones de los modelos de arquitectura y autoconsciencia en tiempo de ejecución; y, consecuentemente, implementa y pone en operación los recursos de monitoreo y autoconsciencia, encargados de producir las métricas directas, indirectas e indicadores sobre el estado y evolución del sistema IoT observado. Además, en este cuasi experimento se utiliza un visor de autoconsciencia, el cual publica los resultados de las métricas, junto con sus interpretaciones y recomendaciones prescriptivas.

Los resultados del cuasi experimento permitirán comprender el rendimiento (eficiencia y efectividad) de los participantes con el uso del configurador web de autoconsciencia, el middleware de la infraestructura y el proceso metodológico Aware-IoT@Run.Time en el que se basan, así como su percepción sobre la facilidad de uso, utilidad e intención de uso futuro de esta solución.

2. Especificación de requerimientos del escenario de IoT para el ejercicio experimental

El escenario de IoT propuesto se enmarca dentro del subdominio de ambientes de vida asistida y tiene como objetivo **dotar de capacidades de autoconsciencia a un Sistema de Control Ambiental, encargado de monitorear la temperatura, humedad y presencia de monóxido de carbono (CO) en el ambiente de una casa en la que vive solo un adulto mayor.**

En la Figura E-1 se presenta el **modelo de arquitectura de monitoreo del Subsistema SCA**, el cual será utilizado como insumo durante las actividades de configuración y despliegue de los procesos de autoconsciencia de este subsistema. En el modelo de arquitectura, se puede observar que la casa tiene un dormitorio y una cocina. A su vez, la arquitectura incluye la utilización de dispositivos y nodos de computación a nivel de las capas Edge, Fog y Cloud. A nivel de la capa Edge, el escenario de IoT utiliza un sensor de temperatura (*DHT11*) en el dormitorio; y, un sensor de CO (*MQ-7*) en la cocina. Estos sensores recolectan los datos del ambiente cada 30 segundos. Además, en la capa Edge se encuentra un microcontrolador (*Raspberry Pi 3*) que actúa como un IoT Gateway para recuperar los datos de los sensores y enviarlos a un servidor (HP Proliant Microserver) ubicado en la capa Fog. Este servidor (nodo Fog) se encuentra en la casa del usuario y tiene como función almacenar localmente los datos crudos de los sensores en una base de datos en PostgreSQL. El IoT Gateway envía los datos de temperatura y humedad al nodo Fog consumiendo directamente un servicio web (comunicación síncrona); mientras que, los datos de CO son enviados por intermedio de un broker (comunicación asíncrona). Finalmente, la capa Cloud, dispone de un nodo de computación implementado bajo la plataforma de CEDIA para almacenar los datos resultantes de las operaciones de agregación ejecutadas en el nodo Fog.

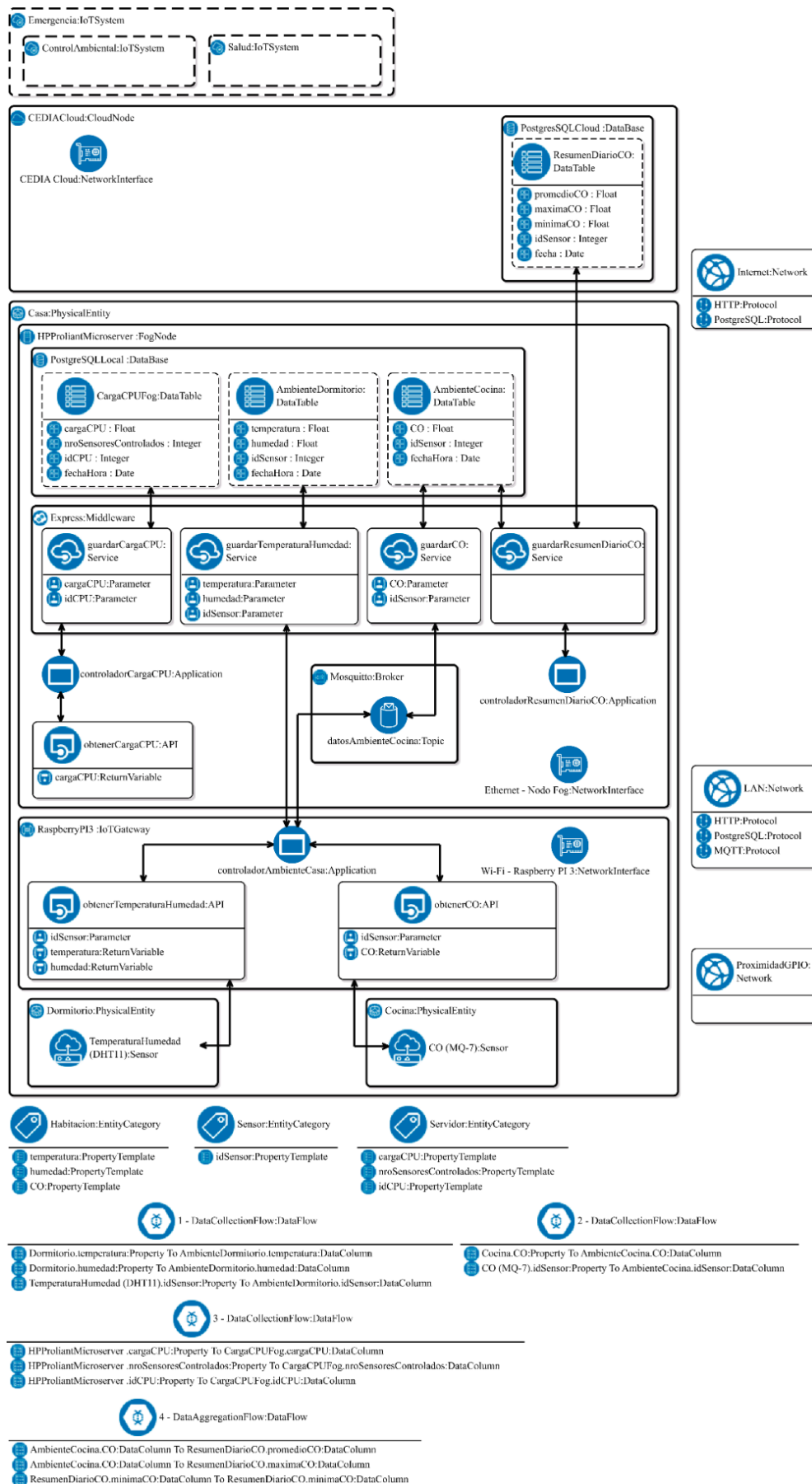


Figura E-1. Modelo de arquitectura de monitoreo del Sistema de Control Ambiental (SCA) a utilizar en el ejercicio experimental. Fuente: Elaboración propia.

Bajo este contexto, el ejercicio experimental tiene como alcance **configurar e implementar dos procesos de autoconsciencia: i) uno pre-reflexivo**, que **captura el conocimiento sobre la concentración y nivel de riesgo actual (en tiempo real) de CO en el ambiente de la cocina; y, ii) otro reflexivo**, que **captura el conocimiento sobre la concentración y nivel de riesgo promedio diario de CO en el ambiente de la cocina**. Los procesos tendrán un tipo de ejecución **predefinido** y se ejecutarán **entre el 1 de enero y el 31 de diciembre del 2023**. El proceso **pre-reflexivo deberá ejecutarse cada 30 segundos, mientras que el reflexivo cada 45 segundos**. Además, estos procesos deberán ejecutarse en el nodo Fog y almacenar las métricas resultantes y su información relacionada en la base de datos ubicada en este mismo nodo. En la Tabla E-1, se describen en detalle los dos procesos de autoconsciencia a implementar en el cuasi experimento.

A su vez, en las Tablas E-2 y E-3 se enlistan las métricas (directa, indirecta, indicadores) y recursos de implementación que deberán ser utilizados para evaluar los aspectos de autoconsciencia del ejercicio experimental.

3. Prerrequisitos para la ejecución del ejercicio experimental

Pre-requisito 1: Instalación e inicialización de los contenedores de docker con los componentes de software de la infraestructura de autoconsciencia.

1. Descargue, instale y ejecute el programa **Docker Desktop**, el cual se puede obtener de la siguiente URL:
https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe?utm_source=docker&utm_medium=webreferral&utm_campaign=dd-smartbutton&utm_location=module
2. Descargue el archivo empaquetado *ContenedoresInfraestructura.zip* con la estructura de directorios (volúmenes) de la máquina host y el archivo *docker-compose.yml* de la siguiente URL:
https://drive.google.com/file/d/10A6WjXEDVbv7Jfk_rVplbisdTqd6vdVb/view?usp=sharing
3. Descomprima el archivo empaquetado en la carpeta *c:\ContenedoresInfraestructura*.
4. Abra la ventana de comandos (ejecute el comando *cmd*) y ubíquese en la carpeta *c:\ContenedoresInfraestructura*.
5. Verifique que los puertos: 80, 9999, 3306, 5432, 4321, 1883 y 9914 estén disponibles (utilice el comando *netstat -ab* para verificar).
6. En la ventana de comando, ejecute el comando *docker-compose up -d* para inicializar los contenedores de Docker con los componentes de la infraestructura.

Pre-requisito 2: Obtención del modelo de arquitectura de monitoreo del Subsistema de Control Ambiental (SCA).

7. Descargue el modelo de arquitectura de monitoreo del subsistema SCA, denominado *ModeloArquitecturaMonitoreoSCA.MonitorIoT*, de la siguiente URL:
<https://drive.google.com/file/d/1KuV9uWYHhIW2LH07I57ZeKBhHsMKrvVi/view?usp=sharing>
A su vez, guárdelo en el directorio *c:\ContenedoresInfraestructura*.

Pre-requisito 3: Creación del usuario para acceder al configurador web de capacidades de autoconsciencia.

8. Abra un navegador (usar de preferencia Google Chrome) e ingrese la url: <http://localhost:80>.
9. En el formulario *Login*, haga clic en el enlace *Crear usuario*.
10. En el formulario *Registro*, complete los campos: nombres y apellidos, correo electrónico, contraseña, confirmar contraseña.
11. Haga clic en el botón *Crear cuenta*.
12. En la ventana *Login*, ingrese el correo electrónico y contraseña.
13. Finalmente, haga clic en el botón *Login*.

Pre-requisito 4: Creación de un nuevo modelo de trabajo de autoconsciencia.

14. En el formulario que aparece, sección *Crear un nuevo modelo (área de trabajo) de autoconsciencia*, ingrese los siguientes campos:

Tabla E-1. Descripción de los procesos pre-reflexivos y reflexivos a implementar en el ejercicio experimental.

Id Proceso	Tipo de proceso de autoconsciencia	Frecuencia de ejecución del proceso	Aspectos de Autoconsciencia a capturar	Criterios de decisión (umbrales) ¹	Interpretación	Acciones (recomendaciones prescriptivas)
1	Pre-reflexivo	30 segundos	Concentración en tiempo real de monóxido de carbono (CO) en el ambiente de la cocina. Tipo de Aspecto: Estado Peso: 0.6	>=0 ppm Y <=35 ppm >35 ppm Y <= 200 ppm >200 ppm	Ausencia de riesgo , el usuario no sufre ningún síntoma ante una exposición prolongada.	No es necesario implementar acciones.
4	Reflexivo	45 segundos	Concentración promedio diario de monóxido de carbono (CO) en el ambiente de la cocina. Tipo de Aspecto: Estado Peso: 0.6		Riesgo medio , el usuario puede experimentar dolores de cabeza, pecho, fatiga o náuseas. Riesgo alto , el usuario puede experimentar desorientación, pérdida del conocimiento e incluso la muerte.	El subsistema SCA debe encender una alerta auditiva intermitente y notificar a la central de monitoreo. El sistema SCA debe encender la alerta auditiva permanente y notificar a la central de monitoreo.

¹ ppm: Partes por millón de concentración de monóxido de carbono (CO).

Tabla E-2: Listado de métricas para evaluar los aspectos de autoconsciencia del ejercicio experimental.

Métrica	Descripción	Atributos				
		Abreviatura	Escala	Unidad de medida	Tipo	Perspectiva
Partes por millón de concentración actual de CO.	Partes por millón de concentración actual de monóxido de carbono (CO) en el ambiente.	PPMCACO	Cuantitativa, continua, ratio	Partes por millón	Directa	
Partes por millón de concentración promedio diario de CO.	Partes por millón de concentración promedio diario de monóxido de carbono (CO) en el ambiente.	PPMCPDCO	Cuantitativa, continua, ratio	Partes por millón	Indirecta	
Nivel de riesgo de la concentración de CO.	Nivel de riesgo de la concentración de monóxido de carbono (CO) para el usuario, a fin de determinar si es necesario alguna acción.	NRCCO	Cuantitativa, continua, ratio	Porcentaje	Indicador	Negativa

Tabla E-3: Listado de recursos de implementación genéricos a utilizar en el ejercicio experimental.

Recurso de implementación	Descripción	Atributos			
obtenerPromedio	Obtiene la media o promedio de un conjunto de datos.	Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	datos	Float	Falso
		Instrucciones: const math = require('mathjs'); return math.mean(datos);			
obtenerNivelRiesgoCO	Obtiene el nivel de riesgo porcentual de la concentración de monóxido de carbono (CO).	Tipo de dato de salida: Float			
		Tipo de recurso: Función			
		Parámetros de entrada:			
		Ordinal	Nombre	Tipo de dato	Opcional
		1	monoxidoCarbono	Float	Falso
		Instrucciones: return (monoxidoCarbono/1000)*100;			
		Tipo de dato de salida: Float			

- Nombre: *Modelo Autoconsciencia SCA*.
 - Descripción: *Modelo de capacidades de autoconsciencia del Subsistema de Control Ambiental (SCA)*.
 - Autor: Nombre y apellido del participante del experimento.
15. Haga clic en el botón *Seleccionar archivo* y escoja el archivo *ModeloArquitecturaMonitoreoSCA.MonitorIoT* que incluye el modelo de arquitectura de monitoreo del Subsistema SCA.
16. Finalmente, haga clic en el botón *Guardar* y comience la ejecución del experimento.

4. Hoja de trabajo del ejercicio experimental

a. Datos del participante

Nombres y apellidos:

Institución:

b. Tareas a completar para configurar e implementar los procesos de autoconsciencia del Subsistema de Control Ambiental (SCA)

El proceso a seguir en este cuasi experimento para configurar e implementar los procesos de autoconsciencia del subsistema SCA se basa en la metodología *Aware-IoT@Run.Time* y contempla las siguientes tareas principales:

- **Tarea 1.** Configuración del dominio de autoconsciencia.
- **Tarea 2.** Definición de los recursos de soporte a los procesos de autoconsciencia.
- **Tarea 3.** Definición de los procesos y métodos de autoconsciencia (aprendizaje y razonamiento).
 - **Tarea 3.1.** Definición del proceso de autoconsciencia pre-reflexivo y su método de recolección y modelo de análisis.
 - **Tarea 3.2.** Definición del proceso de autoconsciencia reflexivo y su método de cálculo y modelo de análisis.
- **Tarea 4.** Generación del modelo de autoconsciencia del sistema IoT.
- **Tarea 5.** Monitorización de los procesos de autoconsciencia implementados y de las métricas y recomendaciones prescriptivas generadas para los aspectos de autoconsciencia.

En el sitio web del experimento se encuentra una guía metodológica con la descripción de las actividades, tareas y subtareas de *Aware-IoT@Run.Time* para construir sistemas IoT autoconscientes con el apoyo del configurador web y middleware de la infraestructura.

Tarea 1. Configuración del dominio de autoconsciencia

Defina y registre los sujetos, objetos y aspectos de autoconsciencia en el configurador web, según los requerimientos del cuasi experimento. Para completar esta tarea se ejecutan tres subtareas:

Registre la hora de inicio de la Tarea 1:

Subtarea 1.1. Definición de sujetos de autoconsciencia

Seleccione los sujetos (sistema o subsistema IoT) de autoconsciencia del ejercicio experimental.

Subtarea 1.2. Definición de objetos de autoconsciencia

Seleccione los objetos (entidad IoT) de autoconsciencia del ejercicio experimental.

Subtarea 1.3. Definición de aspectos de autoconsciencia

Registre los aspectos de autoconsciencia sobre los cuales el sistema IoT será consciente.

Registre la hora de fin de la Tarea 1:

Tarea 2. Definición de los recursos de soporte a los procesos de autoconsciencia

En esta tarea, se registran las métricas, criterios de decisión, umbrales y recursos de implementación (funciones o servicios web) necesarios para la operacionalización de los procesos de autoconsciencia pre-reflexivos y reflexivos requeridos por el sistema IoT en el ejercicio experimental. Para este cometido se ejecutan tres subtareas:

Registre la hora de inicio de la Tarea 2: hh:mm

Subtarea 2.1. Definición de escalas, unidades de medida y métricas

En este ejercicio experimental, las escalas y unidades de medida se encuentran previamente ingresadas en el configurador web de autoconsciencia. Por lo tanto, según la Tabla E-2 registre las métricas a ser utilizadas por los procesos de autoconsciencia a implementar en el cuasi experimento.

Subtarea 2.2. Definición de criterios de decisión y umbrales

Según la Tabla E-1 registre los criterios de decisión y sus umbrales, los cuales serán utilizados por los modelos de análisis de los procesos de autoconsciencia para transformar los indicadores cuantitativos (porcentual) en valoraciones cualitativas sobre el estado o la evolución de los objetos de autoconsciencia del sistema IoT.

Subtarea 2.3. Definición de recursos de implementación

Según la Tabla E-3 agregue en el configurador, las fórmulas, funciones o servicios web necesarios para operacionalizar los métodos de cálculo o modelos de análisis de los procesos de autoconsciencia a implementar en el ejercicio experimental.

Registre la hora de fin de la Tarea 2: hh:mm

Tarea 3. Definición de procesos y métodos de autoconsciencia (aprendizaje y razonamiento)

Ejecute las siguientes subtareas para crear y configurar los procesos de autoconsciencia pre-reflexivos y reflexivos requeridos por el escenario experimental.

Subtarea 3.1. Definición del proceso de autoconsciencia pre-reflexivo y su método de recolección y modelo de análisis

Registre la hora de inicio de la Subtarea 3.1: hh:mm

Subtarea 3.1.1. Registro de la información general del proceso de autoconsciencia pre-reflexivo

Según las especificaciones descritas en el boletín experimental, registre en el configurador la información general sobre el proceso pre-reflexivo: descripción, tipo de ejecución, periodo de tiempo durante el cual se ejecuta, frecuencia u hora de ejecución; sujeto y nodo de computación en el que se ejecuta; y, aspecto de autoconsciencia que captura.

Subtarea 3.1.2. Definición del método de recolección

Configure el método de recolección asociado al proceso pre-reflexivo. Este método es el encargado de recopilar desde un sensor u otra fuente, los datos crudos o empíricos de una propiedad determinada de un objeto de autoconsciencia (entidad IoT); y, producir como resultado una métrica directa.

Subtarea 3.1.3. Definición del modelo de análisis

Configure el modelo de análisis asociado al proceso pre-reflexivo. Este modelo transforma una métrica directa previamente calculada en un indicador cuantitativo (porcentual), con el apoyo de un recurso de implementación. A su vez, como parte del modelo de análisis, el indicador

cuantitativo es comparado con un criterio de decisión (compuesto por umbrales), a fin de obtener un indicador cualitativo que determine el estado o la posible evolución de un objeto de autoconsciencia de un sistema IoT, así como las correspondientes recomendaciones prescriptivas, en caso de ser necesario.

Registre la hora de fin de la Subtarea 3.1: hh:mm

Subtarea 3.2. Definición del proceso de autoconsciencia reflexivo y su método de cálculo y modelo de análisis

Registre la hora de inicio de la Subtarea 3.2: hh:mm

Tarea 3.2.1. Registro de la información general del proceso de autoconsciencia reflexivo

Según las especificaciones descritas en el boletín experimental, registre en el configurador la información general sobre el proceso reflexivo: descripción, tipo de ejecución, periodo de tiempo durante el cual se ejecuta, frecuencia u hora de ejecución; sujeto y nodo de computación en el que se ejecuta; alcance de la autoconsciencia; y, aspecto de autoconsciencia que captura.

Tarea 3.2.2. Definición del método de cálculo

Configure el método de cálculo asociado al proceso reflexivo. Este método construye y/o aplica un modelo conceptual o causal que, a partir de los datos recopilados históricamente, produce una métrica indirecta con el soporte de un recurso de implementación.

Tarea 3.2.3. Definición del modelo de análisis

Configure el modelo de análisis asociado al proceso reflexivo. Este modelo transforma una métrica indirecta previamente calculada en un indicador cuantitativo (porcentual), con el apoyo de un recurso de implementación. A su vez, como parte del modelo de análisis, el indicador cuantitativo es comparado con un criterio de decisión (compuesto por umbrales), a fin de obtener un indicador cualitativo que determine el estado o la posible evolución de un objeto de autoconsciencia de un sistema IoT, así como las correspondientes recomendaciones prescriptivas, en caso de ser necesario.

Registre la hora de fin de la Subtarea 3.2: hh:mm

Tarea 4. Generación del modelo de autoconsciencia del sistema IoT

Con el apoyo del configurador web de autoconsciencia, genere el archivo JSON (*ModelosArquitecturaAutoconsciencia.json*) con los modelos de arquitectura y autoconsciencia en tiempo de ejecución del ejercicio experimental. Para ello, realice las siguientes subtareas:

- 1) En el menú ubicado a la izquierda del configurador web de autoconsciencia, seleccione la opción **“Generar modelo de autoconsciencia”**.
- 2) Haga clic en el botón **“Descargar archivo JSON”** y guárdelo en alguna carpeta del disco duro con el nombre *ModeloArquitecturaAutoconsciencia.json*, con el propósito de remitirlo por correo electrónico al final del experimento.

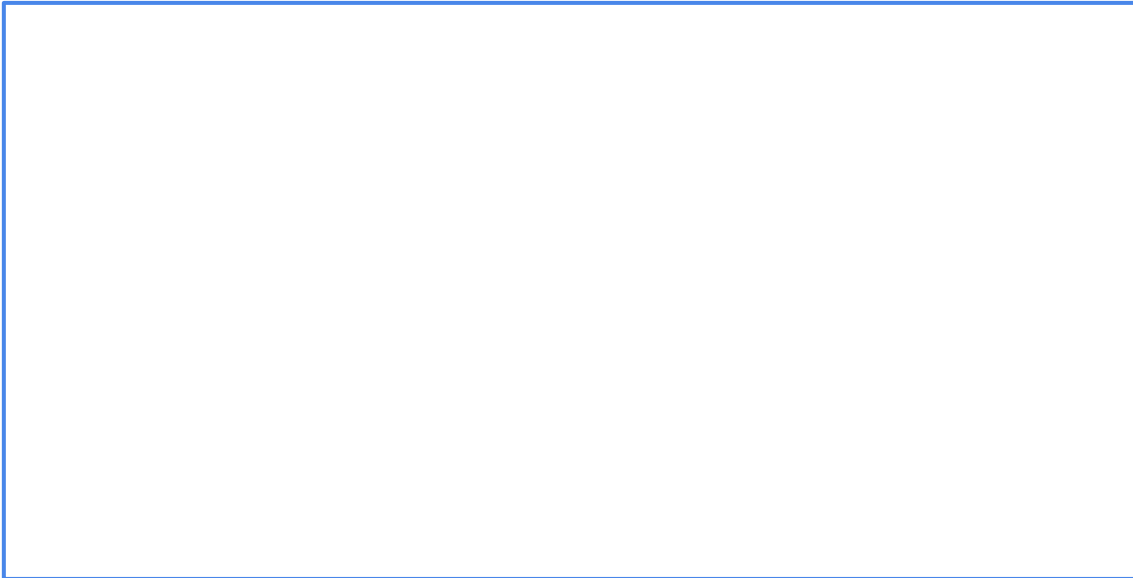
Tarea 5. Monitorización de los procesos de autoconsciencia implementados

En esta tarea, verifique que los procesos de autoconsciencia se encuentran en operación y generando las métricas directas, indirectas e indicadores correspondientes; así como las interpretaciones y recomendaciones prescriptivas, conforme a lo requerido en el escenario experimental. Para ello, se utilice la consola del contenedor de Docker de la aplicación implementadora del nodo Fog y el visor web de autoconsciencia.

Tarea 5.1: Monitorear los procesos de autoconsciencia en la consola de la aplicación implementador del nodo Fog

Realice las siguientes subtareas:

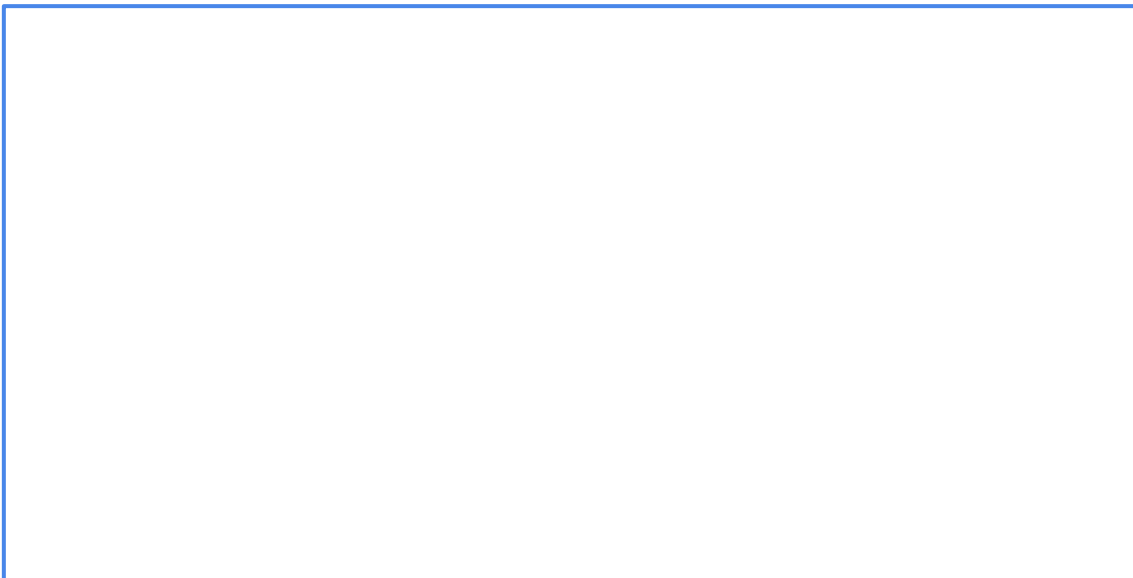
- 1) Abra el programa Docker Desktop.
- 2) En el menú izquierdo, seleccione la opción *Containers*.
- 3) Haga clic en el contenedor de Docker de la aplicación implementadora del nodo Fog.
- 4) A continuación, se presenta la consola de la aplicación implementadora, en la cual se publica periódicamente la información sobre la ejecución de los procesos de autoconsciencia, en lo referente a las métricas producidas, su interpretación y las recomendaciones prescriptivas correspondientes.
- 5) Capture (Print Screen) la consola con los resultados obtenidos por los procesos de autoconsciencia y pegue en el siguiente recuadro:



Tarea 5.2: Monitorear los procesos de autoconsciencia en el Visor web de autoconsciencia

Realizar las siguientes subtareas:

- 1) Abra un navegador (usar de preferencia Google Chrome) e ingrese la URL: <http://localhost:9914>.
- 2) Consulte el indicador del *nivel de riesgo actual (en tiempo real) de la concentración de monóxido de carbono (CO)* producido por el proceso de autoconsciencia pre-reflexivo *PIDI*. Capture (Print Screen) la pantalla del visor de autoconsciencia con los resultados obtenidos y pegue en el siguiente recuadro.



- 3) Consulte el indicador del *nivel de riesgo promedio diario de la concentración de monóxido de carbono (CO)* producido por el proceso de autoconsciencia reflexivo *PID2*. Capture (Print Screen) la pantalla del visor de autoconsciencia con los resultados obtenidos y pegue en el siguiente recuadro.



c. Tareas finales del cuasi experimento

1. Complete la encuesta sobre el uso de la metodología Aware-IoT@Run.Time y sus herramientas de soporte. La encuesta se encuentra en el siguiente enlace:
https://docs.google.com/forms/d/e/1FAIpQLSftnehxqDhvIHx-2ZcD0AhYRAKHDom1gyN_Nnp3ciOw9r9tKA/viewform
2. Entregue la **Hoja de Trabajo del Ejercicio Experimental** y el **archivo JSON** resultante con los modelos de arquitectura y autoconsciencia del Subsistema de Control Ambiental (*ModelosArquitecturaAutoconsciencia.json*) al correo lerazo@uazuay.edu.ec.

¡¡GRACIAS POR SU COLABORACIÓN!!

Anexo F

Resultados de los Cuasi Experimentos

Tabla F-1. Resultados de las variables de rendimiento y percepción por participante del cuasi experimento Nro. 1.

ID	PEOU							PU					ITU			Efectividad				Eficiencia (minutos)			
	P1	P2	P3	P4	P5	$\bar{x}$		P1	P2	P3	P4	$\bar{x}$	P1	P2	$\bar{x}$	T1	T2	T3	$\bar{x}$	T1	T2	T3	Σ
1	2	4	3	3	4	3,20		5	4	4	3	4,00	2	1	1,50	1,00	1,00	1,00	1,00	22	33	7	62
2	4	4	4	4	4	4,00		5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	24	45	6	75
3	5	5	4	4	4	4,40		5	5	4	4	4,50	5	5	5,00	1,00	1,00	1,00	1,00	26	40	15	81
4	5	5	5	5	5	5,00		5	5	5	5	5,00	5	5	5,00	1,00	0,50	1,00	0,83	32	39	27	98
5	5	5	3	4	5	4,40		3	4	4	4	3,75	5	3	4,00	1,00	0,50	0,50	0,67	31	27	15	73
6	5	4	5	4	5	4,60		5	5	5	4	4,75	5	5	5,00	1,00	1,00	1,00	1,00	35	52	16	103
7	5	5	5	4	5	4,80		5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	45	30	15	90
8	4	4	4	3	4	3,80		5	5	5	5	5,00	4	4	4,00	1,00	1,00	0,50	0,83	31	37	13	81
9	5	5	5	5	5	5,00		5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	31	75	12	118
10	4	4	4	4	4	4,00		4	5	4	4	4,25	4	5	4,50	1,00	1,00	0,50	0,83	46	50	9	105
11	5	5	4	4	4	4,40		5	4	5	5	4,75	4	4	4,00	1,00	1,00	1,00	1,00	40	34	21	95
12	3	3	4	4	4	3,60		4	4	4	4	4,00	4	3	3,50	1,00	1,00	1,00	1,00	45	59	22	126
13	5	4	5	4	3	4,20		3	4	5	4	4,00	5	5	5,00	1,00	1,00	1,00	1,00	21	25	10	56
14	4	4	4	5	5	4,40		5	5	4	4	4,50	5	4	4,50	1,00	1,00	1,00	1,00	31	12	12	55
15	5	5	5	4	5	4,80		3	4	3	4	3,50	4	4	4,00	0,50	1,00	0,00	0,50	22	19	11	52
16	2	4	4	4	2	3,20		5	4	4	4	4,25	4	4	4,00	1,00	1,00	1,00	1,00	30	35	15	80
17	4	4	5	3	3	3,80		3	3	4	5	3,75	3	4	3,50	0,50	1,00	0,00	0,50	25	23	16	64
18	5	4	4	4	5	4,40		5	4	5	5	4,75	4	5	4,50	0,50	1,00	1,00	0,83	31	35	9	75
19	5	5	5	5	5	5,00		5	5	5	5	5,00	5	5	5,00	0,50	1,00	1,00	0,83	41	31	8	80
20	4	4	3	4	4	3,80		5	4	5	4	4,50	4	4	4,00	1,00	1,00	1,00	1,00	35	27	14	76
21	5	5	5	5	5	5,00		5	5	5	5	5,00	5	5	5,00	0,50	1,00	1,00	0,83	43	47	4	94
22	5	5	5	5	5	5,00		5	5	5	5	5,00	5	5	5,00	0,50	1,00	1,00	0,83	30	49	11	90
23	4	5	4	4	4	4,20		4	4	4	4	4,00	4	3	3,50	0,50	0,50	0,50	0,50	30	40	20	90
24	5	5	5	4	5	4,80		5	5	4	5	4,75	5	5	5,00	1,00	1,00	0,50	0,83	44	23	17	84
25	5	5	5	5	5	5,00		5	5	5	5	5,00	5	5	5,00	1,00	1,00	0,50	0,83	31	32	39	102
26	4	3	3	3	4	3,40		3	4	3	4	3,50	4	4	4,00	0,50	1,00	0,50	0,67	36	34	20	90
27	5	5	5	4	5	4,80		5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	13	35	34	82
28	4	4	5	5	5	4,60		5	4	4	5	4,50	5	5	5,00	1,00	1,00	1,00	1,00	27	56	9	92
29	5	5	5	4	4	4,60		5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	55	25	24	104
30	5	5	5	5	5	5,00		5	5	5	5	5,00	5	5	5,00	1,00	1,00	0,50	0,83	49	38	15	102
31	4	5	3	5	4	4,20		5	4	4	4	4,25	4	4	4,00	1,00	1,00	0,50	0,83	47	21	17	85
32	4	5	5	5	5	4,80		5	5	4	5	4,75	5	5	5,00	1,00	1,00	0,50	0,83	37	47	10	94
33	5	4	4	5	5	4,60		4	5	4	5	4,50	5	5	5,00	1,00	1,00	1,00	1,00	18	55	12	85

Tabla F-2. Resultados de las variables de rendimiento y percepción por participante del cuasi experimento Nro. 2.

ID	PEOU							PU					ITU			Efectividad							Eficiencia (minutos)			
	P1	P2	P3	P4	P5	P6	x̄	P1	P2	P3	P4	x̄	P1	P2	x̄	T1	T2	T3			x̄	T1	T2	T3	Σ	
																		T3.1	T3.2	x̄						
1	5	5	4	5	5	5	4,83	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	0,50	0,75	0,92	12	9	19	40	
2	5	5	5	5	5	5	5,00	5	5	5	5	5,00	5	5	5,00	0,50	1,00	1,00	0,50	0,75	0,75	18	15	26	59	
3	5	5	5	5	5	5	5,00	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	0,50	0,75	0,92	21	10	31	62	
4	4	5	5	5	4	5	4,67	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	0,50	0,75	0,92	13	10	31	54	
5	5	5	5	5	5	5	5,00	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	0,50	0,75	0,92	20	10	31	61	
6	5	5	5	5	5	5	5,00	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	1,00	1,00	11	8	24	43	
7	3	3	4	4	4	3	3,50	4	4	3	5	4,00	4	4	4,00	1,00	1,00	1,00	0,50	0,75	0,92	16	11	31	58	
8	5	5	5	4	5	5	4,83	5	5	5	5	5,00	5	5	5,00	0,50	1,00	1,00	0,50	0,75	0,75	24	16	29	69	
9	3	3	4	3	4	4	3,50	3	4	3	4	3,50	4	3	3,50	1,00	1,00	1,00	1,00	1,00	1,00	22	8	23	53	
10	2	4	4	4	4	4	3,67	4	4	4	4	4,00	4	4	4,00	1,00	1,00	1,00	1,00	1,00	1,00	14	8	23	45	
11	3	4	4	4	4	4	3,83	4	4	4	4	4,00	4	4	4,00	1,00	1,00	1,00	0,50	0,75	0,92	28	9	23	60	
12	3	2	2	4	3	4	3,00	4	4	4	4	4,00	4	4	4,00	1,00	1,00	1,00	0,50	0,75	0,92	21	10	31	62	
13	4	5	4	4	5	5	4,50	5	5	4	4	4,50	4	4	4,00	1,00	1,00	1,00	1,00	1,00	1,00	19	13	33	65	
14	5	5	5	4	4	4	4,50	4	3	4	3	3,50	4	4	4,00	1,00	1,00	1,00	1,00	1,00	1,00	19	10	32	61	
15	4	5	5	5	4	5	4,67	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	1,00	1,00	26	12	25	63	
16	4	4	5	5	5	5	4,67	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	1,00	1,00	28	15	18	61	
17	3	4	4	3	5	4	3,83	4	5	4	5	4,50	5	5	5,00	1,00	1,00	1,00	1,00	1,00	1,00	17	7	18	42	
18	5	5	5	5	5	5	5,00	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	0,50	0,75	0,92	26	12	28	66	
19	5	5	5	5	5	5	5,00	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	0,50	0,75	0,92	22	13	21	56	
20	4	5	3	4	5	4	4,17	4	3	4	4	3,75	4	5	4,50	1,00	1,00	1,00	1,00	1,00	1,00	20	7	22	49	
21	3	3	4	4	3	4	3,50	4	4	4	5	4,25	4	5	4,50	1,00	1,00	0,50	0,50	0,50	0,83	20	15	35	70	
22	4	4	4	4	5	4	4,17	4	5	4	4	4,25	5	4	4,50	1,00	1,00	1,00	0,50	0,75	0,92	35	10	20	65	
23	5	3	2	3	5	4	3,67	4	4	3	4	3,75	4	3	3,50	1,00	1,00	1,00	1,00	1,00	1,00	12	9	23	44	
24	4	4	4	4	4	4	4,00	4	4	4	4	4,00	4	4	4,00	1,00	1,00	1,00	1,00	1,00	1,00	18	9	30	57	
25	3	3	3	3	3	3	3,00	3	3	3	3	3,00	3	3	3,00	0,50	1,00	1,00	0,50	0,75	0,75	30	7	21	58	
26	4	3	3	3	4	3	3,33	5	5	5	5	5,00	4	5	4,50	1,00	1,00	1,00	1,00	1,00	1,00	16	10	29	55	
27	3	4	3	3	4	3	3,33	4	3	4	3	3,50	3	3	3,00	0,50	1,00	1,00	0,50	0,75	0,75	19	12	36	67	
28	4	4	4	2	4	5	3,83	5	3	4	5	4,25	5	4	4,50	1,00	1,00	1,00	1,00	1,00	1,00	12	9	30	51	
29	4	4	1	1	4	5	3,17	4	5	5	5	4,75	3	3	3,00	1,00	1,00	1,00	1,00	1,00	1,00	17	9	35	61	
30	3	2	2	2	2	4	2,50	4	4	4	5	4,25	4	4	4,00	1,00	1,00	1,00	1,00	1,00	1,00	15	7	33	55	
31	3	3	4	3	5	4	3,67	4	4	4	4	4,00	4	4	4,00	0,50	1,00	1,00	0,50	0,75	0,75	16	14	38	68	
32	5	2	3	3	3	3	3,17	3	3	3	3	3,00	3	2	2,50	0,50	1,00	1,00	0,50	0,75	0,75	25	10	21	56	
33	4	4	3	3	5	5	4,00	3	4	5	4	4,00	4	5	4,50	1,00	1,00	1,00	0,50	0,75	0,92	16	7	29	52	
34	3	3	3	3	3	3	3,00	3	3	3	3	3,00	3	3	3,00	0,50	1,00	1,00	0,50	0,75	0,75	22	13	35	70	
35	4	4	4	4	4	4	4,00	4	4	4	4	4,00	4	4	4,00	0,50	1,00	1,00	0,50	0,75	0,75	31	18	34	83	
36	5	5	5	5	5	5	5,00	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	1,00	1,00	12	8	40	60	
37	4	5	4	3	5	4	4,17	5	5	5	5	5,00	5	5	5,00	1,00	1,00	1,00	1,00	1,00	1,00	18	9	27	54	
38	4	5	4	4	5	5	4,50	5	5	5	5	5,00	5	5	5,00	1,00	1,00	0,50	1,00	0,75	0,92	12	8	29	49	
39	3	5	5	1	3	2	3,17	5	3	5	5	4,50	1	1	1,00	1,00	1,00	0,50	0,50	0,50	0,83	19	12	41	72	
40	4	3	4	4	5	4	4,00	4	4	3	5	4,00	5	5	5,00	1,00	1,00	0,50	1,00	0,75	0,92	10	7	31	48	
41	5	5	4	4	4	5	4,50	5	5	5	5	5,00	5	5	5,00	1,00	1,00	0,50	1,00	0,75	0,92	9	8	34	51	
42	4	5	5	4	5	5	4,67	4	5	5	5	4,75	4	4	4,00	1,00	1,00	1,00	1,00	1,00	1,00	26	12	25	63	