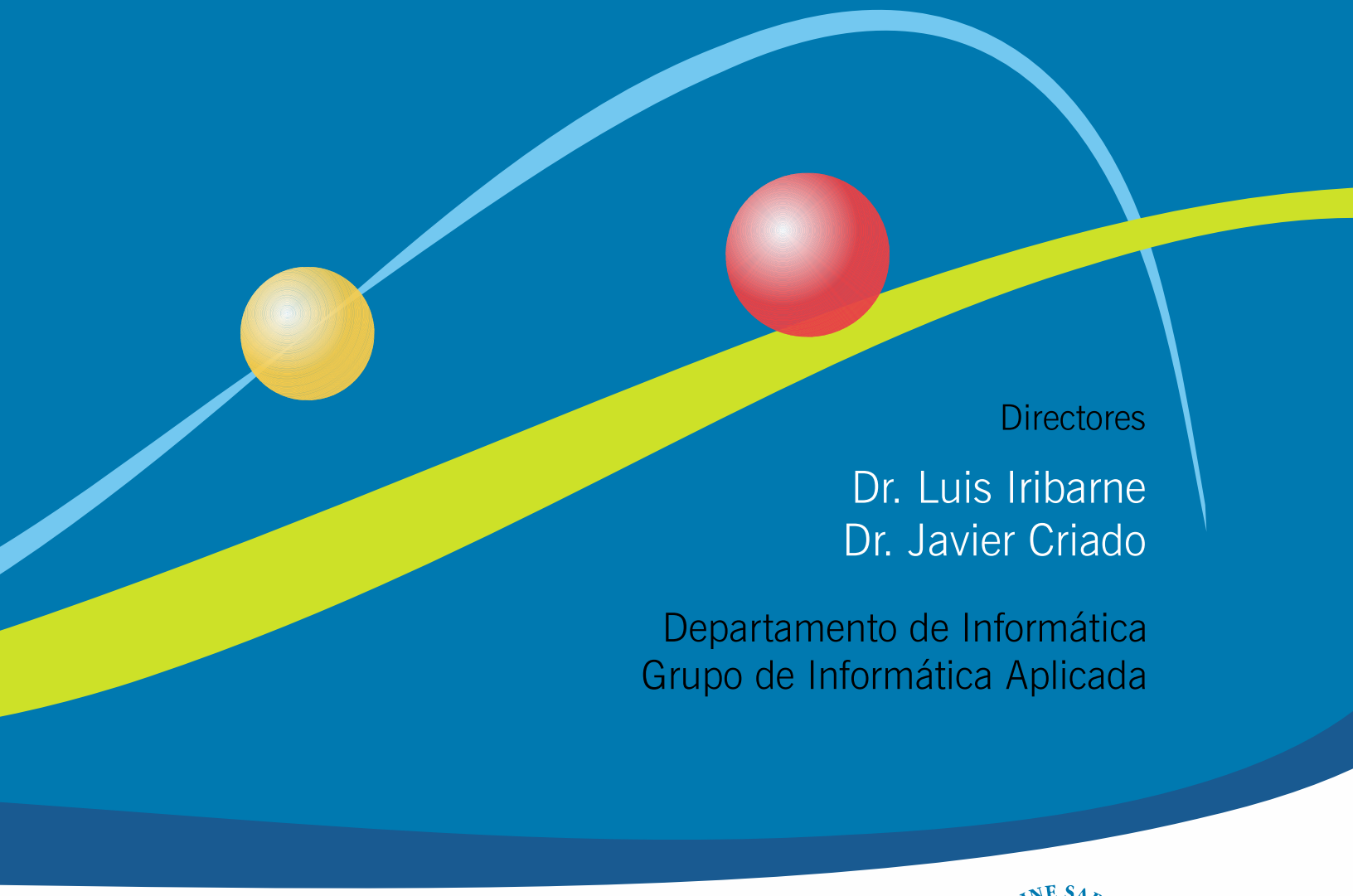
Darwin Omar Alulema Flores

Una Metodología basada en Modelos y Servicios para la Integración de sistemas IoT

Tesis Doctoral



Directores

Dr. Luis Iribarne
Dr. Javier Criado

Departamento de Informática
Grupo de Informática Aplicada

UNIVERSIDAD DE ALMERÍA

Marzo 2021

Departamento de Informática
Universidad de Almería

Marzo de 2021

Una metodología basada en modelos y servicios para la integración de sistemas IoT

TESIS DOCTORAL

Presentada por

Darwin Omar Alulema Flores

para optar al grado de
Doctor Ingeniero en Informática

Dirigida por

Dr. Luis Iribarne
Profesor Titular de Universidad
Departamento de Informática
Universidad de Almería

Dr. Javier Criado
Profesor Ayudante Doctor
Departamento de Informática
Universidad de Almería

Escrito y editado por: Darwin Omar Alulema Flores

Marzo de 2021

Department of Informatics
University of Almería

March 2021

A methodology based on models and services for the integration of IoT systems

THESIS

Presented by

Darwin Omar Alulema Flores

for the degree of
PhD in Computer Science

Thesis supervised by

Dr. Luis Iribarne
Associate Professor
Department of Informatics
University of Almería

Dr. Javier Criado
Assistant Professor
Department of Informatics
University of Almería

Written and edited by: Darwin Omar Alulema Flores

March 2021

Este documento ha sido generado con LaTeX.

Una metodología basada en modelos y servicios para la integración de sistemas IoT

A methodology based on models and services for the integration of IoT systems

Darwin Omar Alulema Flores
Departamento de Informática
Grupo de Investigación de Informática Aplicada (TIC-211)
Universidad de Almería
Almería, Marzo de 2021

http://acg.ual.es

A Dios, a mis Padres Carlos y Marcia, a mi hermana Verónica y a mi esposa Mayerly...

Agradecimientos

A Dios por guiar mis pasos y llevarme a buen puerto.
A mis padres, Carlos y Marcia, por ser el soporte inquebrantable e incondicional durante toda mi vida.
A mi hermana Verónica por ser mi cómplice en cada instante de mi vida.
A mis hermanos, Carlos y Alexis, por el aliento a seguir adelante.
A mi esposa Mayerly, por su apoyo y comprensión.
A Luis y Javi por su guía durante el desarrollo de este trabajo y principalmente por su amistad y calidad humana.
A Rosa y Antonio Jesús, por permitirme conocer un poco más de la hermosa ciudad de Almería, que quedará grabada en mi corazón.
A Nicolás, Antonio Leopoldo, José Antonio, Manel, Juan Jesús y a todo el Grupo de Investigación de Informática Aplicada que he tenido el gusto y el privilegio de conocer; y con quienes espero tener la oportunidad de seguir trabajando.

La presente tesis se finaliza al amparo de los objetivos del proyecto TIN2017-83964-R "Estudio de un enfoque holístico para la interoperabilidad y coexistencia de sistemas dinámicos: Implicación en modelos de Smart Cities".

Darwin Omar Alulema Flores
Universidad de Almería
Almería, 2021

Contenido

Índice de Figuras

Índice de Tablas

Resumen

La tecnología está cada vez más presente en la vida cotidiana, con nuevos productos con mayores prestaciones y menores costes. Una de estas tecnologías es el Internet de las Cosas (IoT, Internet of Things), el cual ha capitalizado el desarrollo de las tecnologías de telecomunicaciones, electrónica e informática. Permite disponer de aplicaciones en áreas tan diversas como la industria, agricultura, transporte, hogar, entretenimiento, gobierno, entre otras. Sin embargo, este amplio espectro de aplicaciones tiene el problema de la diversidad de productos, la falta de estandarización y la dificultad de integración. Estos problemas limitan la creación de aplicaciones complejas, debido a la mayor exigencia de conocimiento de múltiples tecnologías a un desarrollador. En este sentido, la Web de las Cosas (WoT, Web of Things) es una alternativa alentadora que permite el desarrollo de aplicaciones con tecnologías maduras y conocidas, empleadas en el diseño de servicios web. Además, mediante patrones de integración es posible incorporar estos servicios en aplicaciones que requieren de una mayor interoperabilidad, y aplicar técnicas de procesamiento de eventos complejos en datos provenientes de objetos inteligentes.

En esta tesis se propone una metodología basada en técnicas de Ingeniería Dirigida por Modelos (MDE, Model-Driven Engineering) que permite solventar las dificultades presentes en el diseño de sistemas IoT. Uno de los objetivos de esta metodología es facilitar el proceso de desarrollo, al establecer modelos que abstraen la complejidad tecnológica implícita a cada plataforma de despliegue. Como consecuencia, los desarrolladores pueden centrarse en la lógica de negocio de la aplicación. Como parte de la metodología, se han desarrollado las siguientes tareas: (a) definición de una arquitectura de integración para sistemas IoT con componentes heterogéneos; (b) definición de una metodología de implementación basada en MDE para la construcción de aplicaciones IoT; y (c) definición de un Lenguaje Específico del Dominio (DSL, Domain-Specific Language), para el modelado de Sistemas IoT.

Para la definición de la arquitectura se ha planteado tres capas: Física, Lógica y Aplicación. La capa Física hace referencia a los componentes hardware, específicamente al controlador y los transductores (sensores y actuadores), con los cuales se crean los objetos inteligentes que interactúan con el entorno. La capa Lógica se corresponde con los componentes software que permiten la interacción con los objetos inteligentes en Internet, y la definición de la lógica de negocio de la aplicación. Por último, la capa de Aplicación define la interfaz gráfica con la cual el sistema IoT interactúa con los usuarios, permitiendo acceder al sistema desde interfaces tradicionales, como las presentes en la televisión digital (DTV) o los dispositivos móviles.

En la definición de la metodología de implementación, se consideran técnicas de MDE con el propósito de construir modelos para el diseño de aplicaciones IoT conformes a la arquitectura de integración propuesta previamente. Para cumplir con este propósito, se establecieron dos etapas: Especificación y Desarrollo. En la primera, un experto del dominio define un DSL, compuesto por una sintaxis abstracta y concreta, y una transformación modelo a texto (M2T, Model-To-Text). En la etapa de Desarrollo, el usuario final emplea el DSL para el diseño de las aplicaciones IoT, creando instancias específicas de la sintaxis abstracta que sirve de entrada para la transformación M2T. Como resultado de la transformación M2T, se crean los artefactos software con código funcional para las plataformas de software y hardware en las que se despliega la aplicación final.

Con respecto a la definición del DSL para la Integración de Sistemas IoT, se ha considerado el ecosistema Eclipse para la construcción de cada una de sus partes: (a) para el caso de la sintaxis abstracta se emplea EMF, con el cual se construye el metamodelo; (b) para el caso de la sintaxis concreta se emplea Sirius, con el cual se construye el editor gráfico; y (c) para el caso de la transformación de modelos se emplea Acceleo, con el cual se genera código fuente funcional.

El metamodelo para la integración de los sistemas IoT consta de cinco niveles: (a) infraestructura, que permite modelar las características que deben tener los servidores que ejecutarán las aplicaciones y la red a la cual se conectarán los nodos de hardware; (b) hardware, el cual permite modelar el controlador y los transductores, los cuales componen los objetos inteligentes; (c) control, permite modelar la lógica de control de los objetos por medio de la orquestación de los servicios REST y Bridge de cada uno de los transductores; (d) DTV, el cual permite modelar la interfaz de usuario para la DTV; y (e) móvil, que permite modelar la interfaz de usuario para los móviles.

El editor gráfico permite a los desarrolladores crear instancias del metamodelo a partir de una interfaz gráfica al arrastrar iconos representativos, configurar sus características e interconectarlos con otros componentes, con el propósito de establecer la lógica de negocio del sistema. Por último, la transformación M2T permite crear código fuente de Arduino, Ballerina, Node-Red, Android y NCL-Lua, de acuerdo con el modelo creado en el editor gráfico.

Para la validación de la propuesta se usaron cinco instrumentos: (a) escala PANAS (Positive and Negative Affect Schedule); (b) escala SUS (System Usability Scale); (c) pruebas de funcionamiento; (d) costes de desarrollo; y (e) pruebas de rendimiento. Las escalas PANAS y SUS, están dentro del grupo de pruebas de usabilidad; la primera analiza las emociones experimentadas por los usuarios con el DSL, mientras que la segunda determina el grado de usabilidad del editor gráfico. Con respecto a las pruebas de funcionamiento y de costes de desarrollo, estas son pruebas funcionales que permiten, en el primer caso, comprobar el funcionamiento del DSL para crear una aplicación IoT funcional, mientras que el segundo, compara de forma teórica el proceso de desarrollo con respecto al tiempo que le tomaría a un desarrollador crear la misma aplicación con lenguajes de propósito general. Por último, para las pruebas de rendimiento se empleó Gatling para evaluar los servicios REST de los transductores, lo cual permite determinar el estrés máximo que pueden soportar los servicios antes de funcionar de forma incorrecta. Estas pruebas permitieron validar la utilidad de la metodología al facilitar el desarrollo de aplicaciones IoT con gran capacidad de gestionar nodos heterogéneos.

Como resultado de la investigación realizada en la presente tesis, se ha obtenido un total de 8 contribuciones científicas, de las cuales 2 han sido en congreso nacional (Jornadas de Ingeniería del Software y Bases de Datos, España), 4 en congresos internacionales con publicaciones en las series Springer *Advances in Intelligent Systems and Computing* y *Advances in Intelligent Systems and Computing*, y otras 2 contribuciones en las revistas internacionales de impacto *Cluster Computing* (Elsevier, JCR Q1, Computer Science) y *Journal of Universal Computer Science* (JCR Q3, Computer Science).

Finalmente, se han identificado algunas áreas de investigación que aún están abiertas y que pueden ser el inicio de investigaciones futuras: (a) incorporar servicios de indexación y descubrimiento con el propósito de solventar los casos en los cuales un servicio no está disponible en tiempo real con otros servicios equivalentes usando mecanismos de mediación; (b) extender la evaluación de la metodología propuesta con el fin de verificar los niveles de aceptación y satisfacción de expertos comerciales, y ampliar las pruebas para integración y calidad de software; (c) aplicar estrategias de balanceo de carga para manejar un gran número de usuarios concurrentes; (d) extender la propuesta para emplear buses empresariales como Mule ESB, lo que permitiría ampliar la posibilidad de patrones de integración más robustos y complejos; y (e) incorporar otros mecanismos de control a la propuesta, como controles diferencial, integral, o difuso, para permitir el diseño de aplicaciones críticas, donde el tiempo de respuesta requerido sea mucho menor.

Summary

Technology is increasingly present in everyday life, with new products with higher performance and lower costs. One of these technologies is the Internet of Things (IoT), which has capitalised on the development of telecommunications, electronics and computing technologies. It enables applications in areas as diverse as industry, agriculture, transport, home, entertainment, government, among others. However, this broad spectrum of applications has the problem of product diversity, lack of standardisation and difficulty of integration. These problems limit the creation of complex applications, due to the greater demand on a developer's knowledge of multiple technologies. In this sense, the Web of Things (WoT) is an encouraging alternative that allows the development of applications with established and well-known technologies used in the design of web services. Moreover, by means of integration patterns it is possible to incorporate these services in applications that require greater interoperability, and to apply complex event processing techniques to data coming from intelligent objects.

In this thesis, a methodology based on Model-Driven Engineering (MDE) techniques is proposed to overcome the difficulties present in the design of IoT systems. One of the objectives of this methodology is to facilitate the development process by establishing models that abstract the technological complexity implicit in each deployment platform. As a result, developers can focus on the business logic of the application. As part of the methodology, the following tasks have been developed: (a) definition of an integration architecture for IoT systems with heterogeneous components; (b) definition of an implementation methodology based on MDE for the construction of IoT applications; and (c) definition of a Domain-Specific Language (DSL), for the modelling of IoT Systems.

For the definition of the architecture, three layers have been proposed: Physical, Logic and Application. The Physical layer refers to the hardware components, specifically the controller and transducers (sensors and actuators), with which the smart objects that interact with the environment are created. The Logic layer corresponds to the software components that enable the interaction with the smart objects on the Internet, and the definition of the business logic of the application. Finally, the Application layer defines the graphical interface with which the IoT system interacts with users, allowing access to the system from traditional interfaces, such as those present in digital television (DTV) or mobile devices.

In the definition of the implementation methodology, DEM techniques are considered in order to build models for the design of IoT applications conforming to the previously proposed integration architecture. To fulfil this purpose, two stages were established: Specification and Development. In the first one, a domain expert defines a DSL, com-

posed of an abstract and concrete syntax, and a Model-to-Text transformation (M2T). In the Development stage, the end-user uses the DSL for the design of IoT applications, creating specific instances of the abstract syntax that serve as input for the M2T transformation. As a result of the M2T transformation, software artifacts are created with functional code for the software and hardware platforms on which the final application is deployed.

Regarding the definition of the DSL for IoT Systems Integration, the Eclipse ecosystem has been considered for the construction of each of its parts: (a) for the case of the abstract syntax, EMF is used, with which the metamodel is built; (b) for the case of the concrete syntax, Sirius is used, with which the graphical editor is built; and (c) for the case of the model transformation, Acceleo is used, with which functional source code is generated.

The metamodel for IoT systems integration consists of five levels: (a) infrastructure, which allows modelling the characteristics that the servers that will run the applications and the network to which the hardware nodes will be connected must have; (b) hardware, which allows modelling the controller and transducers, which compose the smart objects; (c) control, which allows modelling the control logic of the objects by orchestrating the REST and Bridge services of each of the transducers; (d) DTV, which allows modelling the user interface for the DTV; and (e) mobile, which allows modelling the user interface for the mobiles.

The graphical editor allows developers to create instances of the metamodel from a graphical interface by dragging representative icons, configuring their characteristics and interconnecting them with other components, in order to establish the business logic of the system. Finally, the M2T transformation allows the creation of source code for Arduino, Ballerina, Node-Red, Android and NCL-Lua, according to the model created in the graphical editor.

Five instruments were used to validate the proposal: (a) Positive and Negative Affect Schedule (PANAS); (b) System Usability Scale (SUS); (c) functional testing; (d) development costs; and (e) performance testing. The PANAS and SUS scales fall into the usability testing group; the former analyses the emotions experienced by users with the DSL, while the latter determines the degree of usability of the graphical editor. With regard to the performance and development cost tests, these are functional tests that allow, in the first case, to check the performance of the DSL to create a functional IoT application, while the second one compares in a theoretical way the development process with respect to the time it would take a developer to create the same application with general-purpose languages. Finally, for the performance tests, Gatling was used to evaluate the REST services of the transducers, which allows determining the maximum stress that the services can withstand before malfunctioning. These tests validated the usefulness of the methodology in facilitating the development of IoT applications with high capacity to manage heterogeneous nodes.

As a result of the research carried out in this thesis, a total of 8 scientific contributions have been obtained, of which 2 have been in national conferences (Jornadas de Ingeniería del Software y Bases de Datos, Spain), 4 in international conferences with publications in the Springer series *Advances in Intelligent Systems and Computing* and *Advances in Intelligent Systems and Computing*, and another 2 contributions in the in-

ternational journals of impact *Cluster Computing* (Elsevier, JCR Q1, Computer Science) and *Journal of Universal Computer Science* (JCR Q3, Computer Science).

Finally, some research areas have been identified which are still open and may be the start of future research: (a) incorporating indexing and discovery services with the purpose of solving cases in which a service is not available in real time with other equivalent services using mediation mechanisms; (b) extending the evaluation of the proposed methodology in order to verify the levels of acceptance and satisfaction of commercial experts, and extending the tests for software integration and quality; (c) apply load balancing strategies to handle a large number of concurrent users; (d) extend the proposal to employ enterprise buses such as Mule ESB, which would extend the possibility of more robust and complex integration patterns; and (e) incorporate other control mechanisms into the proposal, such as differential, integral, or fuzzy controls, to enable the design of critical applications, where the required response time is much lower.

Capítulo 1

Introducción

Capítulo 1

Introducción

Contenidos

El gran desarrollo que está teniendo Internet de las Cosas (IoT, Internet of Things) implica la adaptación de las aplicaciones existentes a las nuevas tecnologías de hardware y de software que se popularizan cada vez más. En este sentido, la necesidad de desarrollar aplicaciones IoT cobra mayor relevancia, ya que al no estar establecido un estándar para el desarrollo de las tecnologías se genera una gran heterogeneidad. Es así, que el trabajo de investigación desarrollado en esta tesis doctoral presenta una metodología para la construcción semiautomática de los artefactos de software que participan en una aplicación IoT. Para lo cual se propone una metodología basada en modelos y servicios, en la que se construye un Lenguaje Específico de Dominio (DSL, Domain-Specific Language), que permite desarrollar modelos para las aplicaciones IoT; y se definen las reglas de transformación Modelo a Texto (M2T, Model-to-Text), con lo cual se generan los artefactos de software a partir de los modelos desarrollados.

En el presente capítulo, se resumen los aspectos generales que motivan este trabajo y se estructura en siete secciones. En primer lugar, la Sección 1.1 presenta el planteamiento del problema y que se pretende resolver. La Sección 1.2 presenta las preguntas de investigación que motivan el trabajo. La Sección 1.3 presenta un estudio sistemático de la literatura relacionada con este trabajo. La Sección 1.4 describe brevemente la metodología propuesta para la creación de aplicaciones IoT. La Sección 1.5 describe las consideraciones generales de la propuesta y presenta resultados derivados del trabajo de investigación. Para finalizar, la Sección 1.6 describe la organización de la tesis.

1.1. Planteamiento del problema

Uno de los retos que plantea el rápido crecimiento de los dispositivos conectados al Internet es permitir que los usuarios consuman y combinen las funcionalidades que los recursos de software y hardware pueden ofrecer en cualquier lugar y en cualquier momento [Urbieta et al., 2017]. Para lograr la necesaria integración e inmediatez en los escenarios actuales del IoT, los objetos deben estar disponibles para otras aplicaciones de Internet o para el proceso de negocio, como entidades autónomas con comportamiento proactivo, con conocimiento del entorno, y capacidad para comunicarse y colaborar entre sí, así como entre usuarios y sistemas [Teixeira et al., 2017]. Es así que en el IoT se dispone de diferentes formas para acceder a las cosas y a sus datos, lo que permite a los desarrolladores crear aplicaciones multidominio, multiplataforma y globales. En consecuencia, surgen nuevas oportunidades comerciales, especialmente para las pequeñas empresas innovadoras [Broring et al., 2017]. Sin embargo, el IoT se enfrenta a la dificultad de integrar aplicaciones en un ecosistema flexible de modo que los dispositivos se puedan reutilizar para diferentes escenarios [Guinard et al., 2010].

De la amplia diversidad de dispositivos y plataformas han surgido dos desafíos principales. En primer lugar, la naturaleza heterogénea de la información hace que la tarea de interpretar esa información y detectar eventos del mundo real sea más compleja. En segundo lugar, la entrega de información sensorial genera algunos problemas, como el consumo limitado de recursos de los dispositivos IoT [Al-Osta et al., 2019]. Uno de los

escenarios más atractivos donde se pueden apreciar realmente esta diversidad es el Hogar Inteligente [Jie et al., 2013], porque visualiza un entorno en el cual pueden coexistir una gran variedad de sensores y actuadores que están configurados y controlados de forma remota y autónoma. Sin embargo, estos sistemas a menudo no son interoperables ni están interconectados [Miori and Russo, 2014]. Por ejemplo, en caso de que el usuario necesite interconectar dos o más objetos de diferentes fabricantes, se deben desarrollar diferentes aplicaciones, con formatos de información diferentes, con un código diferente, valores diferentes para los objetos, con protocolos de comunicación diferentes y quizás con un lenguaje de programación diferente [González-García et al., 2017]. El escenario propuesto presenta varios desafíos adicionales. En primer lugar, el diseño y posterior implementación de aplicaciones capaces de utilizar información de dispositivos IoT. Por ejemplo, si un desarrollador desea agregar un dispositivo que informe la intensidad de la luz a su aplicación, ¿cómo pueden integrar su funcionalidad en los servicios web tradicionales? En segundo lugar, las aplicaciones de IoT están sujetas a eventos asíncronos, es decir que el sistema debe ser reactivo ante la presencia de diferentes eventos generados en el entorno de IoT.

Todos los desafíos mencionados ponen de manifiesto la necesidad de abordar la interoperabilidad y la coordinación de los componentes de software y hardware del sistema, debido a que cada componente de la aplicación está diseñado para una plataforma determinada. Como tal, los desarrolladores deben conocer los requisitos específicos de cada plataforma de hardware y software, lo cual crea un alto grado de dependencia entre el software de control y la plataforma, que resulta en poca flexibilidad y gran dificultad en los procesos de mantenimiento [Teixeira et al., 2017]. Un punto adicional a considerar es la interacción del usuario con los sistemas de IoT [Brambilla et al., 2018], el cual no ha tenido el mismo desarrollo que las propuestas en torno al hardware y el software de control. De hecho, la visión actual de IoT se centra en la infraestructura, gestión y análisis de la gran cantidad de datos generados [González-García et al., 2017].

En consecuencia, se hace notorio la dificultad que tienen los desarrolladores para lidiar con muchos aspectos específicos de las plataformas de software y hardware, además de la empinada curva de aprendizaje relacionada con los lenguajes de programación y los recursos de hardware. Una estrategia para abordar los desafíos anteriores es promover la separación de cada componente del sistema en un conjunto de características específicas en cada paso del diseño. Por lo tanto, se puede considerar dos perspectivas [Teixeira et al., 2017]: (a) los detalles de programación, configuraciones y protocolos de comunicación de bajo nivel; y (b) las reglas de alto nivel y software de control, junto con eventos de dominio.

Una estrategia que actualmente está en auge para el desarrollo de las aplicaciones IoT es emplear la Web, ya que parece ser el mejor candidato para una plataforma de integración universal que aborda los aspectos de la comunicación y las reglas de control, empleando tecnologías propias de la Web. Esta iniciativa ha sido promovida por la W3C Web of Things (WoT) [W3C, 2020], que propone una capa de abstracción basada en las tecnologías utilizadas en la Web para la representación e interacción con dispositivos IoT. En esta los objetos deben ser directamente accesibles como recursos web normales [Mainetti et al., 2015], donde los servicios exponen recursos del mundo real, y así pueden integrarse y coordinarse fácilmente en el mundo virtual. Para abordar el diseño de una

posible arquitectura de integración, Mainetti [Mainetti et al., 2015] plantea que la arquitectura debe ser capaz de abstraer dispositivos físicos, virtualizándolos y exponiéndolos a través de una interfaz común; y debe permitir el desarrollo de aplicaciones IoT para usuarios con diferentes capacidades y plataformas heterogéneas. Sin embargo, crear una arquitectura de referencia que integre los servicios IoT con los servicios tradicionales sigue siendo un desafío de investigación abierto [Dar et al., 2015].

La presente investigación está destinada a posibilitar la convivencia de tecnologías heterogéneas. En este sentido, un enfoque basado en la Ingeniería Dirigida por Modelos (MDE, Model-Driven Engineering) puede aportar una solución ya que permite utilizar modelos para abstraer las características de cada plataforma, lo que significa que este paradigma facilita la adopción de nuevas tecnologías y permite automatizar algunas de las fases de desarrollo, por ejemplo, mediante la generación del código. Además, la propuesta debe proporcionar un mecanismo de comunicación flexible que permita enviar información desde los objetos y servicios, para que puedan ser procesados según la lógica de negocio de la aplicación. Por ello, una Arquitectura combinada de SOA, (Service-Oriented Architecture) y EDA (Event-Driven Architecture) parece ser el diseño más adecuado para asegurar un entorno adaptable que permita la conexión dinámica de dispositivos, nodos de hardware, aplicaciones y otro tipo de elementos en el sistema.

El planteamiento inicial surge de la necesidad de desarrollar metodologías y herramientas que faciliten el trabajo de los programadores. De esta forma, en este documento se propone una metodología basada en modelos y servicios para el desarrollo de software que permite la interoperabilidad de las aplicaciones IoT. En este sentido, es necesario identificar las fortalezas, oportunidades, debilidades y amenazas que puede presentar esta metodología. En la Tabla 1.1 se listan algunos factores.

FORTALEZAS	**OPORTUNIDADES**
Rapidez en el desarrollo de aplicaciones.	Ampliación a nuevas tecnologías.
Estandarización de tecnologías.	Ampliación a nuevas características.
Interoperabilidad garantiza.	Aplicación en la docencia, investigación e industria.
Tendencia en las investigaciones actuales.	Aumento en el interés de nuevas tecnologías.
DEBILIDADES	**AMENAZAS**
Poca versatilidad.	Fallos en la seguridad.
Requiere especialización para las modificaciones.	Cambios en licencias de herramientas de desarrollo.

Tabla 1.1: Análisis DAFO del Planteamiento de partida.

Como se puede observar en la Tabla 1.1 existen más fortalezas y oportunidades, comparado con las debilidades y amenazas, que inciden en la metodología que contesta la hipótesis de partida. La razón principal es el empleo de MDE que permite abstraer las características generales de los componentes de un sistema IoT sin entrar en los detalles, y obtener un diseño modular que permite actualizar la propuesta para que esta se mantenga vigente en el tiempo. Además, al ser hoy en día el IoT una tecnología con gran auge y que se espera crezca aún más con el devenir de la Industria 4.0, la propuesta cada vez cobrará más importancia. Por último, al disponer de una herramienta que permite a los desarrolladores crear aplicaciones en un tiempo reducido promueve el interés al poder obtener resultados de manera casi inmediata. Estos y otros detalles de la propuesta se desarrollan con mayor profundidad en los capítulos siguientes.

1.2. Preguntas de investigación

En esta tesis doctoral se describe una metodología basada en modelos que permite la automatización de diversas tareas en la fase de desarrollo de aplicaciones IoT. Este enfoque utiliza EDA para procesar los eventos generados por los sensores y actuadores mediante el uso de una cola de mensajes. Los procesos de negocio de la arquitectura se implementan como microservicios permitiendo que cada dispositivo cuente con una API RESTful que le permita comunicarse con el resto de los componentes de la arquitectura, logrando así la integración de sus servicios. Para la integración, se utiliza el patrón Saga basado en orquestación encargado de coordinar las transacciones que deben ser ejecutadas por cada uno de los microservicios. Los principales objetivos de la investigación desarrollada han sido los siguientes:

— Establecer las bases del conocimiento sobre MDE, SOA, EDA e IoT, identificando y categorizando la investigación disponible sobre el tema,

— Definir una arquitectura de integración para el dominio de IoT, con múltiples plataformas de software y de hardware,

— Crear una sintaxis abstracta para modelar los elementos de una aplicación de IoT,

— Crear una sintaxis concreta que permita la construcción de aplicaciones de IoT,

— Definir reglas de transformación de modelos para crear los artefactos de software utilizados en una aplicación de IoT y

— Evaluar los artefactos de software que se generan a partir de la transformación del modelo.

Para lograr los objetivos de investigación, se deben responder las siguientes preguntas de investigación:

RQ1: *¿Existen tecnologías en la industria para la integración de sistemas IoT? Y si es así, ¿cuáles son y qué grado de madurez presentan?*

Debido a la necesidad de integrar todas las aplicaciones y tecnologías existentes, se utilizan múltiples tecnologías que abordan los sistemas de IoT con diferentes enfoques: (a) objetos (por ejemplo, Raspberry, mangOH, Arduino, Raspberry Pi, Jetson, Google Coral, Sierra Inalámbrico); (b) redes (por ejemplo, NB-IoT, LoRa, LTE-M); (c) plataformas (por ejemplo, Amazon AWS, Google Cloud, Fiware, Microsoft Azure); y (d) análisis (por ejemplo, Keras, Tensofflow). Por su grado de madurez, estas tecnologías se han utilizado en muchos proyectos. Por ejemplo, en el campo de los objetos, Raspberry y Arduino, fueron los primeros en ofrecer una alternativa de bajo costo. Sin embargo, tienen algunas limitaciones en sus capacidades de procesamiento y comunicación, por lo que empresas como Google han comenzado a incursionar en este entorno con tarjetas de desarrollo orientadas a aplicaciones de Inteligencia Artificial. Además, han utilizado sus plataformas como Google Cloud para brindar servicios adicionales que se integran más fácilmente.

Si bien han surgido iniciativas como la de Google, todavía quedan algunos campos sin considerar, como la electrónica de aplicaciones y la integración de estos servicios. Para ello, la metodología propuesta ofrece una solución que abarca la electrónica de los nodos de IoT, la comunicación a través de WiFi, el transporte de información a través de MQTT, la gestión de la lógica empresarial a través de patrones de integración basados en la orquestación de microservicios y la monitorización de la información a través de interfaces de usuario como DTV (Digital TV) y móvil. Sin embargo, no se han considerado otras colas de mensajes como las que se utilizan en los servicios empresariales (por ejemplo, Kafka, AMQ). Para remediar esto, la propuesta proporciona a los desarrolladores un mecanismo para crear aplicaciones que es completamente independiente de las plataformas comerciales existentes.

RQ2: *¿Qué, cómo y cuánto podría contribuir una metodología MDE al desarrollo de sistemas IoT en comparación con las metodologías tradicionales?*

MDE permite la reutilización de componentes para simplificar y acelerar el trabajo de los desarrolladores, ya que se abstraen las características de los elementos de los sistemas IoT, lo que significa que se pueden utilizar en múltiples escenarios. Además, permite generar código de forma automática o semiautomática mediante transformaciones M2T, sin necesidad de ampliar el conocimiento de las plataformas hardware y software que intervienen en los sistemas IoT, y así los desarrolladores pueden centrarse en la lógica de los sistemas. Finalmente, MDE permite agilizar el proceso de creación de artefactos software para sistemas IoT, ya que genera código fuente funcional, con lo que los desarrolladores solo tienen que desplegar los artefactos en cada una de las plataformas correspondientes. De esta manera la propuesta utiliza las características de MDE para el desarrollo de aplicaciones IoT, para lo cual se desarrolla un metamodelo, un Editor Gráfico y un generador de código que a través de transformaciones M2T (Model-to-Text), con lo que es posible crear artefactos de software. Si bien la propuesta se limita a crear código para Arduino, Node-Red, Ballerina, NCL-Lua y Android, se puede extender a otras plataformas agregando más reglas de transformación, de acuerdo con las que se requieran.

RQ3: *¿Se puede aplicar la metodología propuesta a diferentes escenarios de IoT?*

Aunque la mayoría de las aplicaciones experimentales se han desarrollado en el campo del Hogar Inteligente, el enfoque no se limita solo a este dominio. Esta versatilidad se debe al DSL que se ha creado, ya que considera las características funcionales básicas de los elementos presentes en los sistemas IoT, divididos en las tres capas propuestas, que hace que no dependa de un escenario en particular. En este sentido, la propuesta se puede aplicar a áreas como Smart Health, Smart Agro y Smart Cities, entre otras. Esto se debe a que los sensores y actuadores son componentes simples y pueden ser tratados como un servicio web tradicional. Además, la propuesta permite la integración de servicios, por lo que se puede procesar escenarios complejos, ya que se utilizan señales binarias (ON/OFF) o señales continuas de tipo analógico. Finalmente, la propuesta permite integrar diferentes interfaces al consumir las API REST para cada sensor o actuador.

RQ4: *¿Qué técnicas o herramientas se pueden utilizar para evaluar la metodología?*

Para la evaluación de la propuesta se optó por realizar tres tipos de pruebas, que cubren los aspectos del hardware y software e interacción con el usuario:

— Pruebas de usabilidad: Con estas pruebas evaluamos la interacción de los usuarios en un entorno académico, con el propósito de registrar las percepciones sobre la usabilidad de la herramienta y su influencia en las emociones que experimentan los usuarios.

— Pruebas funcionales: Con estas pruebas se pretende validar el funcionamiento de las herramientas de acuerdo con los requisitos para generar aplicaciones IoT. Por lo tanto, esta prueba asegura que un dispositivo IoT sea capaz de proporcionar un rendimiento de referencia y puede funcionar sin errores.

— Pruebas de rendimiento: Con este tipo de pruebas se pretende comprobar que las aplicaciones IoT generadas con la herramienta funcionen correctamente dentro de una determinada carga de trabajo.

En este caso, como lo evidencian las pruebas aplicadas, se obtienen resultados favorables con respecto a la usabilidad de la herramienta. A través de un mecanismo teórico se verifica el beneficio de utilizar MDE para las aplicaciones de IoT, entre los cuales se destaca el agilizar el proceso de desarrollo reduciendo el trabajo a unas pocas horas. Para ello, el desarrollador debe centrarse en el diseño arquitectónico y la lógica operativa, y no en los detalles de codificación, ya que estos son semiautomatizados. Por último, las pruebas de rendimiento muestran un buen comportamiento del sistema ante una gran cantidad de solicitudes, lo cual es deseable en aplicaciones IoT, donde los nodos que intervienen pueden llegar a ser muchos.

Las preguntas de investigación han permitido guiar el desarrollo del trabajo, pero también han permitido validar que este tipo de enfoques podría ser muy útil en varios campos, desde el mundo académico hasta el industrial [Mainetti et al., 2015]. Desde un punto de vista académico, simplificaría y aceleraría la implementación y prueba de nuevas aplicaciones de IoT, al verificar el comportamiento de los nodos en entornos reales. Desde un punto de vista industrial, las empresas podrían aprovechar este enfoque para monitorizar y actualizar continuamente sus productos.

1.3. Estudio sistemático de la Literatura

Con el propósito de identificar y analizar los trabajos relacionados se realizó el Mapeo Sistemático de la Literatura (SMS, Systematic Mapping Study) [Teixeira et al., 2017] y una Revisión Sistemática de la Literatura para (SLR, Systematic Literature Review) [Monguel, 2018]: (1) proporcionar una descripción general amplia de esta área de investigación, (2) encontrar y mostrar evidencia dentro de un tema de interés y (3) utilizar una metodología bien definida para identificar, analizar e interpretar toda la evidencia disponible relacionada con preguntas de investigación específicas de manera imparcial.

El SMS proporciona una amplia descripción de un área de investigación, para establecer si existe evidencia de investigación sobre un tema y proporcionar una indicación de la cantidad de evidencia. Los resultados de un estudio de mapeo pueden identificar áreas adecuadas para realizar revisiones sistemáticas de literatura y también áreas donde un estudio primario es más apropiado. Mientras que la SLR es un medio de identificar, evaluar e interpretar toda la investigación disponible relevante para una pregunta de investigación, área temática o fenómeno de interés en particular.

Las principales diferencias entre SMS y SLR son [Kitchenham, 2007]:

— Los estudios de mapeo generalmente tienen preguntas de investigación más amplias que los impulsan y, a menudo, plantean múltiples preguntas de investigación.

— Los términos de búsqueda para estudios de mapeo están menos enfocados que para las revisiones sistemáticas y es probable que devuelvan una gran cantidad de estudios.

— El proceso de extracción de datos para estudios de mapeo también es mucho más amplio que el proceso de extracción de datos para revisiones sistemáticas y se puede denominar con mayor precisión una etapa de clasificación o categorización.

— La etapa de análisis de un estudio de mapeo consiste en resumir los datos para responder a las preguntas de investigación planteadas. Es poco probable que incluya técnicas de análisis en profundidad como el meta-análisis y la síntesis narrativa, sino totales y resúmenes.

Con el propósito de identificar la literatura relevante se procede primero con un mapeo bibliográfico que consta de los siguientes pasos [Castillo-Vergara et al., 2018]:

(a) *Definición del campo de estudio*: para la revisión de la literatura el campo de estudio es una "Metodología basada en modelos y servicios para la integración de sistemas IoT".

(b) *Selección de la base de datos*: la información se recuperó de la base de datos Scopus, debido a que incluye más revistas indexadas que otras bases de datos como WoS (Web of Science), y permite extraer más información. Este estudio analiza publicaciones desde 2002 hasta la fecha actual del análisis.

(c) *Ajuste de los criterios de investigación*: las palabras clave incluidas en la búsqueda fueron las siguientes:

```
(ALL (iot OR wot) AND ALL (orchestrator OR choreography OR eda OR soa OR roa
 OR rest OR tv OR test OR tools) AND ALL (mde OR dsl OR model*))
```

Los resultados de esta búsqueda arrojaron un total de 2889 publicaciones, las cuales fueron filtradas por el campo de "Computer Science" y "Engineering", ya que son las áreas de interés para el análisis, arrojando 2711 documentos. La base de datos fue filtrada por "artículos", obteniendo un total de 922 resultados, que se utilizaron para desarrollar el presente estudio. La búsqueda se realizó en dos fases, principios 2017 y posteriormente durante el segundo semestre del 2020, con objeto de actualizar el estado actual de la investigación.

Estos resultados permiten establecer los principales autores, instituciones, países y revistas en este campo. Además, a través de un análisis de co-ocurrencia de palabras clave, este estudio identifica algunos de los temas que han recibido más interés, así como algunas tendencias de investigación para el futuro. En este caso el número mínimo de ocurrencia de las palabras clave se consideró de 5, con lo cual se obtuvo 395 palabras claves que se ilustran en la Figura 1.1. Estos resultados sirven de guía para que en los capítulos siguientes se aborden los contenidos teóricos y metodológicos que guían esta tesis doctoral.

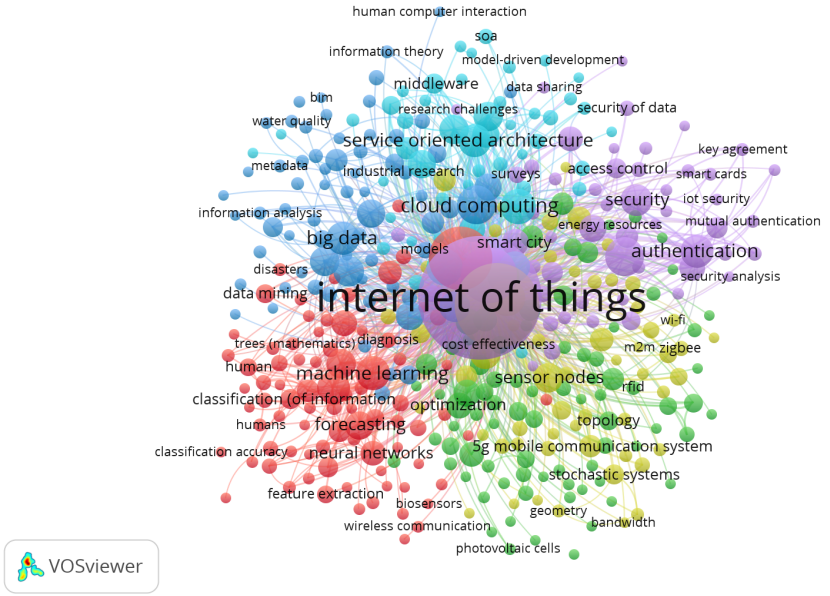


Figura 1.1: Mapa bibliométrico con las palabras claves.

(d) *Codificación del material recuperado*: una vez realizada la búsqueda, se creó una base de datos única en un archivo que contiene todo el registro con las variables de autor, idioma, año de publicación, tipo de investigación, país, campo de investigación, palabras clave y referencias citadas en cada una de las publicaciones incluidas en la búsqueda. También se definieron los intervalos de tiempo de estudio para realizar un análisis de contenido.

(e) *Análisis de la información*: para el análisis de la información se ha empleado VOSviewer [University-Leiden, 2020] para la construcción y visualización de mapas bibliométricos. La principal ventaja de este programa sobre la mayoría de los programas de tecnología de la información disponibles para el mapeo bibliométrico es que se centra en las representaciones gráficas de los mapas [Castillo-Vergara et al., 2018].

Realizando un análisis bibliométrico, en la Figura 1.2 se muestran las 20 palabras clave más utilizadas durante el período de 2002 a 2020. Este aspecto es importante ya que permite visualizar las temáticas más abordas, así como el período en el que más fueron empleadas, lo que permite orientar la investigación hacia trabajos futuros.

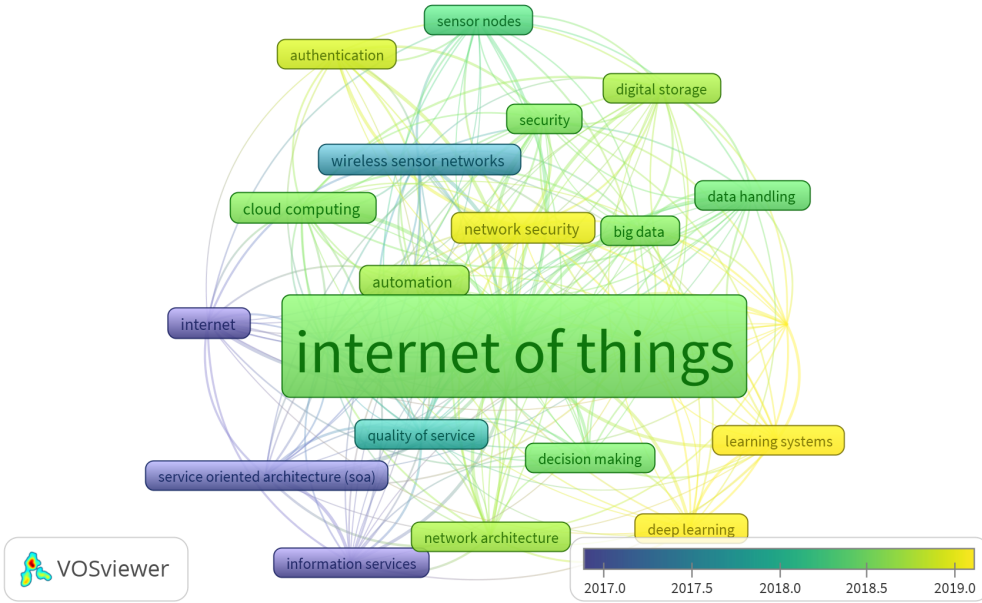


Figura 1.2: Palabras claves más empleadas.

En la Figura 1.3 se muestra la evolución de los artículos relacionados con la búsqueda. Como se observa, el número de manuscritos ha pasado de 1 en 2002 a 316 en 2020. También cabe destacar que el 75.49 % de los artículos han sido publicados en los últimos tres años (2018-2020), lo que denota el reciente interés en este campo de la investigación.

En la Figura 1.4 se muestra la evolución de los artículos en función de la fuente. Como se puede observar, la "IEEE Access" es la que agrupa el mayor número de publicaciones con 71, seguido de "IEEE Internet of Things Journal" y "Sensors Switzerland" con 68 publicaciones cada una, y luego "Future Generation Computer Systems" con 28 publicaciones. El 25.49 % de los artículos se han publicado en estas tres fuentes.

En la Figura 1.5 se muestra la evolución de los artículos relacionados por país. Como se puede observar, China tiene 195 publicaciones, India 158 publicaciones, Estados Unidos 119 y Corea del Sur 109 publicaciones. Lo que muestra que estos cuatro países concentran el 62.25 % de los artículos que se han publicado.

Como se observa en la Figura 1.6.a, hay dos autores más activos, Das A.K., con 12 publicaciones, y Kumari S., con 10. Sin embargo, destaca Das A.K., ya que posee 304 citas y un índice H de 40. Por otra parte se puede observar en la Figura 1.6.b, que Catarinucci I. es el autor más citado con 455 citas, seguido de He W. con 329 citas.

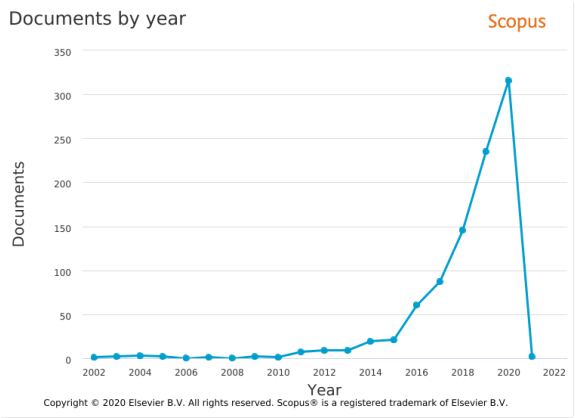


Figura 1.3: Documentos publicados por año.

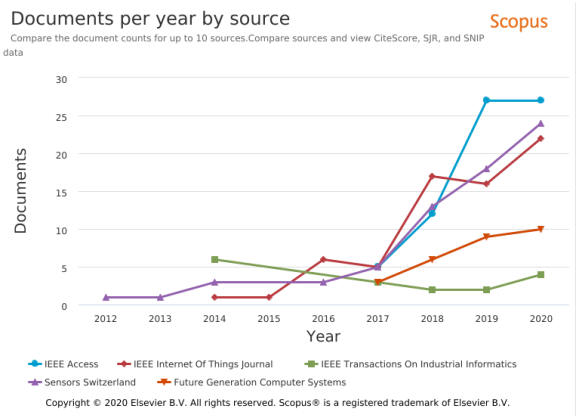


Figura 1.4: Documentos publicados por fuente.

Otro indicador importante del análisis bibliográfico son las redes de colaboración que se forman para realizar las investigaciones. En este caso existen 2 clústers con mayor productividad y visibilidad:

(a) Clúster 1: Está el Institute of computing tecnology, Chinese academy of sciences (China), con 5 documentos y 548 citas; y e Old domain university – USA, con 5 docuementos y 694 citas.

(b) Clúster 2: Está el Center for security, theory and algorithmic, International institute of information technol (India), con 7 documentos y 197 citas; y el School of electronics engineering, Kyungpook national university (Corea del sur), con 5 documentos y 54 citas.

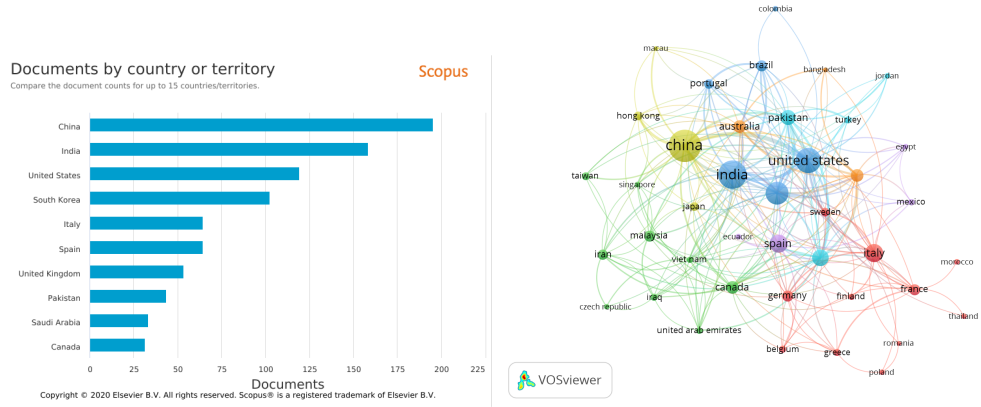


Figura 1.5: Publicaciones/país (izquierda) y países que más publican (derecha).

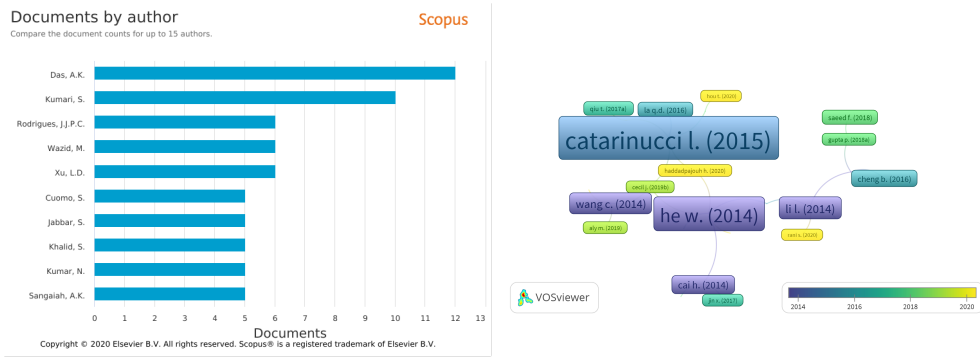


Figura 1.6: Publicaciones por autor (izquierda) y autores con más citas (derecha).

1.4. Metodología propuesta

El IoT permite unir el mundo de los objetos con el mundo de las personas. Sin embargo, estos objetos deben colaborar con diferentes plataformas por lo que se requieren sistemas que puedan adaptarse y modificarse a lo largo del ciclo de vida de una aplicación. Esta integración presenta varias dificultades debido al surgimiento de nuevas tecnologías y tendencias, la modernización de los servicios y los recursos, o la adecuación a las preferencias de los usuarios. En este sentido cobra importancia un mecanismo que permita automatizar los procesos de desarrollo para los sistemas IoT. En esta investigación se presenta una metodología basada en modelos y servicios para la integración de sistemas IoT. Se propone una formalización de los componentes y sus relaciones en los sistemas IoT y además se construyen unas herramientas que permiten automatizar algunas etapas del desarrollo de sistemas IoT. La metodología ofrece una arquitectura para la integración (Capítulo 3), compuesto por tres capas, véase Figura 1.7:

— Capa física, corresponde a los sensores, actuadores y controladores; además especifica el protocolo de comunicación. Estos dispositivos constituyen los nodos *Edge* de una arquitectura *Cloud*.

— Capa lógica, junto con la capa física son el Back-End del sistema y permite coordinar la información proveniente de las capas física y de aplicación, para lo cual consta de: (i) Broker, recibe todos los mensajes de los clientes y luego enruta los mensajes a los clientes de destino adecuados, (ii) Bridge, enruta los mensajes provenientes del Broker hacia la API RESTful o el Orquestador, (iii) API RESTful, despliega los métodos HTTP (GET, POST, PUT, DELETE) con comunicación a una Base de Datos, para que cada nodo físico al cual se encuentran asociados puedan ser consumidos por el orquestador, la Capa de Aplicación o por Servicios Remotos; (iv) Orquestador, coordina las secuencias de llamado a los servicios REST o Bridge, para la ejecución de la lógica de negocio. Estos servicios pueden ejecutarse en una nube pública o privada. Además, se puede formar una arquitectura *Cloud* con los orquestadores como parte de la computación *Fog* y para coordinaciones más complejas se establecen coreografías de orquestadores, que se corrresponde con la computación *Cloud*.

— Capa de aplicación, corresponde al Front-End del sistema y permite la interacción de los usuarios en interfaces no web.

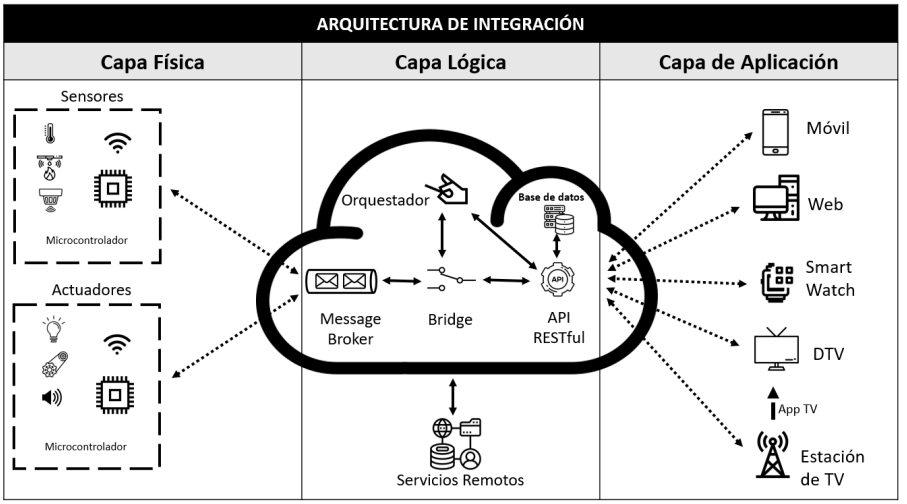


Figura 1.7: Arquitectura de integración.

En base a la arquitectura propuesta para la integración de sistemas IoT se establece una etapa de especificación que está constituida por tres operaciones principales (Capítulo 4): (a) Diseño del metamodelo, (b) Desarrollo del editor gráfico y (c) Generación de

la transformación M2T. Las dos primeras acciones se realizan en los dos niveles de representación: el nivel abstracto y el nivel concreto. En el nivel abstracto, los modelos de arquitectura describen los componentes de hardware, los servicios de integración y las interfaces de usuarios; y en el nivel concreto se crea una herramienta gráfica para diseñar los modelos de las aplicaciones IoT, con los cuales por medio de una transformación M2T se genera código fuente funcional para cada plataforma del sistema.

A su vez las capas de la arquitectura se dividen en niveles que agrupan las metaclases de acuerdo con el ámbito que describen. En la Tabla 1.2 se muestra cada una de las capas de la arquitectura con sus niveles y principales metaclases.

CAPAS DE LA ARQUITECTURA	NIVEL
Capa Física	Infraestructura
	Hardware
Capa Lógica	Control
Capa de Aplicación	DTV
	Móvil

Tabla 1.2: Sistemas de la arquitectura de integración.

Para crear el Editor Gráfico (Sintaxis Concreta) se utilizó Eclipse Sirius [Sirius, 2020]. El proceso de especificación del editor gráfico consideró las siguientes actividades: (a) creación de un proyecto de especificación, (b) especificación de un VSM (Viewpoint Specification Model), (c) especificación del tipo de representación, (d) mapeo entre los elementos gráficos del diagrama con los elementos del metamodelo y (e) especificación de los elementos de la barra de herramientas del editor (Paleta).

Para automatizar el proceso de desarrollo de las aplicaciones, se utilizó transformación Modelo a Texto con la herramienta Eclipse Acceleo [Acceleo, 2020]. Acceleo permite generar ficheros de código ejecutable o documentación. Con lo cual se consigue ahorrar esfuerzos y reducir errores, automatizando la construcción de nuevas aplicaciones IoT.

1.5. Consideraciones generales

Las aplicaciones IoT son sistemas intrínsecamente heterogéneos que consisten en muchos sensores, actuadores, protocolos de comunicación y sistemas operativos diferentes. Esto hace que la prueba de los sistemas sea una tarea desafiante [Bosmans et al., 2019]. En este sentido y una vez definidas las tres etapas principales de la metodología propuesta para el desarrollo de aplicaciones IoT, es necesario realizar una implementación de pruebas que permitan evaluar y validar las herramientas construidas. Dado que existe una fuerte cohesión entre el hardware y el software en los proyectos de IoT, el enfoque de prueba utilizado por lo general se basa en las mejores prácticas utilizadas en el desarrollo clásico de software o hardware [Bosmans et al., 2019]. Para evaluar las herramientas generadas se optó por realizar tres tipos de pruebas: Pruebas de usabilidad, Pruebas funcionales y Pruebas de rendimiento (Capítulo 5).

Para finalizar, como resultado de esta tesis doctoral, se han obtenido 8 publicaciones, dos en revistas JCR, 4 en congresos internacionales y 2 en congresos nacionales (Capítulo

6). A continuación, se listan las publicaciones en orden cronológico y los aportes que hacen a la metodología:

— *A DSL for the Development of Heterogeneous Applications* (FiCloudW'2017). En esta publicación se planteó el primer metamodelo en el cual se incluye la DTV.

— *Una propuesta de editor gráfico para el desarrollo de aplicaciones multiplaforma* (JISBD'2018). Se crea el primer editor gráfico para la propuesta.

— *A Cross-Device Architecture for Modelling Authentication Features in IoT Application* (J.UCS, 2018). Se plantea una primera arquitectura de integración para las plataformas de una aplicación IoT.

— *Merging DTV and MDE Technologies on the Internet of Things* (ICITS'2019). Se completa el editor gráfico y la generación de código para aplicaciones IoT con interacción en la DTV.

— *A model-driven approach for the integration of hardware nodes in the IoT* (WorldCist'2019). Se incorporan los nodos de hardware y se genera código para el controlador propuesto.

— *RESTIoT: A model-based approach for building RESTful web services in IoT systems* (JISBD'2019). Se genera código para los servicios REST asociados a los sensores y actuadores de la aplicación IoT.

— *A model-driven engineering approach for the service integration of IoT systems* (Cluster Computing, 2020). Se completa con un metamodelo, un editor gráfico, una transformación M2T y pruebas de rendimiento de los servicios REST.

— *An approach to integrate IoT systems with no-web interfaces* (ICITS'21). Se amplía la propuesta con la interfaz móvil.

1.6. Organización de la tesis

El presente documento está compuesto por seis capítulos, un anexo, un listado de acrónimos y un listado con las referencias bibliográficas utilizadas.

— El Capítulo 2 ofrece una revisión del estado del arte del conjunto de tecnologías relacionadas con la investigación. Se aborda el desarrollo dirigido por modelos, las técnicas de integración, las arquitecturas SOA y EDA. Además, se revisa el IoT y las tecnologías empleadas para el despliegue de las aplicaciones.

— El Capítulo 3 describe de forma global la metodología propuesta para la creación de aplicaciones IoT basado en modelos y servicios. Para lo cual se proponen una arquitectura de tres capas (Física, Lógica y Aplicación). Además, se introduce como caso de estudio el Hogar Inteligente, que será utilizado a lo largo del trabajo como escenario de ejemplo para la explicación de determinadas partes de la metodología y se propone una metodología para la implementación en la que se diferencian dos roles principales el especificador y el desarrollador.

— En el Capítulo 4 se presenta la metodología para el modelado y construcción de aplicaciones IoT. Para ello, primero se presenta la sintaxis abstracta y concreta que guían el proceso de desarrollo de las aplicaciones basado en la arquitectura propuesta del Capítulo 3. Luego se ofrecen detalles del proceso de transformación M2T con el cual se crean los artefactos de software específicos para cada plataforma de la aplicación.

— El Capítulo 5 describe el proceso de evaluación y validación de la metodología a partir del escenario de Hogar Inteligente y se aplican tres tipos de pruebas: usabilidad, funcional y rendimiento. El caso de las pruebas de usabilidad se emplea la escala de PANAS y SUS, para analizar el comportamiento de los usuarios ante las herramientas construidas en la propuesta. Para las pruebas funcionales se implementó una aplicación para el control de varios sensores y actuadores controlados por el controlador Node MCU, además se estableció una lógica de negocio implementada en un orquestador que coordina la integración de los servicios, y una interfaz DTV y móvil con la que el usuario interactúa con el sistema. Para la evaluación de la funcionalidad también se analizó el tiempo de desarrollo de la aplicación, y se realizó una estimación basada en puntos de función. Finalmente, se realizaron pruebas de rendimiento con Gatling con el fin de probar la fiabilidad de los servicios REST de la aplicación frente a distintos escenarios.

— El Capítulo 6 contiene los resultados y las conclusiones del trabajo de investigación desarrollado. Para ello, se describen cuáles son las aportaciones realizadas, cuáles son las limitaciones y qué líneas de investigación se proponen como trabajo futuro.

— El Anexo contiene el metamodelo desarrollado y los instrumentos empleados para las pruebas de usabilidad.

— El listado de acrónimos organiza todos los acrónimos utilizados.

— El listado con las referencias bibliográficas que se han utilizado en el documento.

La estructura de cada capítulo es la siguiente: comienza con la sección *Introducción y conceptos relacionados*, y luego incluye partes específicas de cada capítulo. Al final de cada capítulo existe una sección *Trabajos relacionados*, seguida de la última sección que contiene el capítulo *Resumen y Conclusiones*.

Capítulo 2

Revisión de la Tecnología

Capítulo 2

Revisión de la Tecnología

Contenidos

El objetivo principal del trabajo de investigación que aquí se presenta es el desarrollo de una metodología para la integración de sistemas IoT basada en modelos y servicios. La cual debe considerar varias tecnologías de hardware y software que permitan el diseño y descripción de los componentes que intervienen en un sistema IoT. Por esta razón, en este capítulo se realiza una revisión de los mecanismos de integración y las tecnologías que permiten el modelado de estos sistemas.

El capítulo está estructurado en ocho secciones. Se comienza con la introducción del trabajo de investigación y se presenta las secciones principales sobre las que se sustenta. A continuación, se revisa cada una de las secciones. En primer lugar, se presentan las técnicas de Ingeniería Dirigida por Modelos (MDE, Model-Driven Engineering). Además, se introduce el concepto de Arquitectura Orientada a Servicios (SOA, Service-Oriented Architecture) y la integración de los procesos por medio de patrones Saga basados en Orquestación y Coreografía. Posteriormente se revisan las Arquitecturas Dirigidas por Eventos (EDA, Event-Driven Architecture), que permiten la gestión de los nodos IoT. A continuación se describen las características del IoT, considerando los aspectos de software y hardware, así como las herramientas de prueba. También se describen algunas de las tecnologías relacionadas con las plataformas que se emplean durante el desarrollo de la propuesta. El capítulo finaliza con un resumen breve y con las principales conclusiones extraídas de la integración de sistemas IoT.

2.1. Desarrollo dirigido por modelos

En el Desarrollo de Software Dirigido por Modelos (MDSD, Model-Driven Software Development), los modelos no constituyen documentación, pero se consideran iguales al código, ya que su implementación es automatizada. Por ejemplo, si se considera una línea de producción automotriz: el pedido de un automóvil que incluye características personalizadas se convierte en realidad. En este ejemplo se muestra que el dominio es fundamental para los modelos, al igual que para los procesos de producción automatizados. Sin embargo, el lenguaje de modelado orientado al cliente para la fabricación de automóviles no puede construir automóviles. Por lo tanto, MDSD tiene como objetivo encontrar abstracciones específicas de dominio y hacerlas accesibles a través de modelos formales. Para aplicar el concepto de 'modelo específico de dominio', se deben cumplir tres requisitos [Sthal et al., 2006]:

— Se requieren lenguajes específicos de dominio para permitir la formulación real de modelos.

— Se necesitan lenguajes que puedan expresar las transformaciones necesarias de modelo a código.

— Se requieren compiladores, generadores o transformadores que puedan ejecutar las transformaciones para generar código ejecutable en las plataformas disponibles.

El desarrollo de software dirigido por modelos ofrece un enfoque en el que los modelos son abstractos y formales al mismo tiempo. La abstracción significa compacidad y reducción a la esencia. Los modelos MDSD tienen el significado exacto de código de programa ya que la mayor parte de la implementación final pueden generarse a partir de ellos. En este caso, los modelos ya no son solo documentación, sino partes del software, lo que constituye un factor decisivo para aumentar la velocidad y la calidad del desarrollo del software.

Los medios de expresión utilizados por los modelos están orientados hacia el espacio de problemas del dominio respectivo, lo que permite la abstracción del nivel del lenguaje de programación y permite la compacidad correspondiente. Para formalizar estos modelos, se requiere un lenguaje de modelado específico de dominio (DSL, Domain-Specific Language) de nivel superior. Además de los modelos formales y abstractos, las plataformas específicas de dominio constituyen el segundo pilar fundamental: los componentes y marcos prefabricados y reutilizables ofrecen una base mucho más poderosa que un lenguaje de programación. La Figura 2.1 muestra las relaciones en el desarrollo de aplicaciones con MDSD.

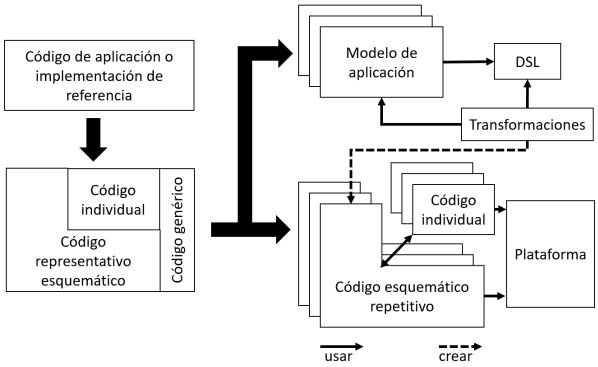


Figura 2.1: Esquema del proceso de desarrollo de software dirigido por modelos (Fuente: adaptado de [Sthal et al., 2006]).

2.1.1. Ingeniería dirigida por modelos (MDE)

La ingeniería dirigida por modelos (MDE) es un enfoque para el desarrollo de software donde los artefactos principales son los modelos. El objetivo principal es reducir la complejidad del software causada por la tecnología, los métodos y los lenguajes de programación utilizados para desarrollar software. El principio básico detrás de MDE es que todo es un modelo. Como tal, proporciona un enfoque genérico para tratar todos los artefactos de software posibles utilizados y producidos durante el ciclo de vida de desarrollo de software [Schlegel et al., 2013]. Incluso los lenguajes utilizados para especificar los modelos también pueden considerarse modelos, a los que se hace referencia como metamodelos [Babau et al., 2009].

MDE se ha aplicado en varios campos de la industria para desarrollar nuevas funciones de software. El diseño de estos componentes de software involucra a diseñadores desde varios puntos de vista, tales como teoría de control, ingeniería de software, seguridad, etc. En la práctica, mientras que un diseñador de una disciplina se enfoca en los aspectos centrales de su campo (por ejemplo un ingeniero de control se concentra en diseñar un controlador estable), descuida o considera menos importante los otros aspectos de ingeniería (por ejemplo ingeniería de software en tiempo real o eficiencia energética). Esto puede causar que algunos de los requisitos funcionales y no funcionales no se cumplan satisfactoriamente [Sundharam et al., 2018]. En este sentido MDE organiza el desarrollo de software en torno a modelos. Estos recopilan la mayor parte de la información requerida para producir sistemas y sus artefactos relacionados. El proceso de desarrollo se concibe principalmente como un refinamiento iterativo de modelos donde se les agrega y modifica información [Fernández-Isabel and Fuentes-Fernández, 2015]. Los principios básicos del MDE son [García-Molina et al., 2012]:

— Modelo: Representa total o parcialmente una parte de un sistema software. El OMG (Object Management Group) define los modelos como: "un modelo de un sistema es una descripción o especificación de ese sistema y su entorno para un propósito determinado" [Raibulet et al., 2017].

— Lenguaje Específico del Dominio (DSL, Domain-Specific Language): los modelos son representados por medio de un DSL, el cual consta de tres elementos principales: 1) la sintaxis abstracta que define los conceptos del lenguaje y las relaciones entre ellos; 2) la sintaxis concreta que establece la notación, y 3) la semántica que normalmente es definida a través de la traducción a conceptos de otro lenguaje destino cuya semántica se conoce.

— Metamodelo: La sintaxis abstracta de un DSL se define mediante un metamodelo, el cual es un modelo conceptual del DSL expresado con un lenguaje de metamodelado con un conjunto de reglas que definen restricciones adicionales para que un modelo se considere bien formado.

— La transformación de modelos: La automatización es conseguida a través de la transformación de los modelos a código mediante sucesivas transformaciones.

2.1.2. Metamodelado

Los metamodelos son modelos que hacen declaraciones sobre modelado. Un metamodelo describe la posible estructura de los modelos, además, define las construcciones de un lenguaje de modelado y sus relaciones, así como las restricciones y las reglas de modelado, pero no la sintaxis concreta del lenguaje. Los metamodelos y los modelos tienen una relación clase-instancia: cada modelo es una instancia de un metamodelo. Para definir un metamodelo, se requiere un lenguaje de metamodelado que a su vez lo describa un metamodelo [Sthal et al., 2006]. En la Figura 2.2 se presenta un ejemplo en el que se relaciona el mundo real con su representación abstracta. En este caso se han considerado dos animales de compañía que se agrupan y se clasifican según se especie. Estos conceptos son representados mediante un metamodelo que permite su representación abstracta.

Los modelos se pueden describir en un lenguaje de modelado arbitrario. La selección del idioma debe hacerse en función de la idoneidad del idioma para el dominio que se describirá. Esta decisión a menudo está determinada por la pregunta de si las herramientas están disponibles o no para el lenguaje de modelado, lo que significa que el Lenguaje Unificado de Modelado (UML, Unified Modeling Language) se usa para modelar en muchos casos. El OMG define cuatro meta-niveles que se pueden ver en la Figura 2.2: (a) M0: Instancia, (b) M1: Modelo, (c) M2: Metamodelo, y (d) Meta-modelo. Cada modelo se escribe en el lenguaje que define su metamodelo, quedando establecida la relación entre el modelo y su metamodelo por una relación de "conformidad" [Durán et al., 2013].

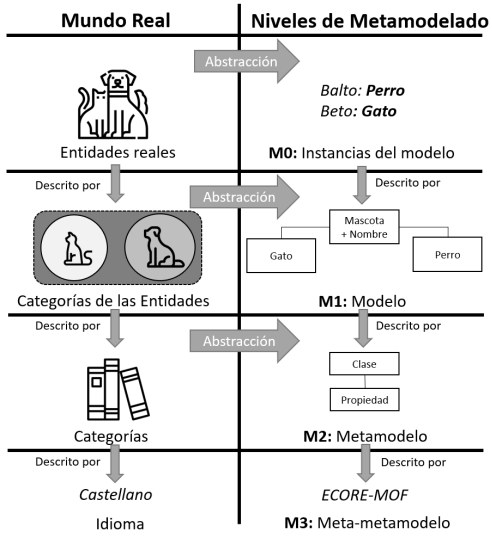


Figura 2.2: Niveles de metamodelado.

La definición de un lenguaje de modelado incluye los aspectos expuestos en la Tabla 2.1. En la cual se observa la separación que existe entre la semántica y la sintaxis de un lenguaje. La semántica se refiere a la relación que tienen los modelos con otros modelos y su ámbito de aplicación, mientras que la sintaxis se refiere a la estructura lógica y la representación gráfica. Estos aspectos no son completamente independientes, por una parte, la sintaxis concreta depende de la sintaxis abstracta, ya que deben definir un símbolo gráfico para cada concepto definido en la sintaxis abstracta, y el símbolo debe expresar adecuadamente las propiedades del concepto. La notación o sintaxis concreta puede ser textual o gráfica y en el contexto de MDE ha aparecido un buen número de herramientas basadas en el metamodelado que permiten crear [Fernández-Isabel and Fuentes-Fernández, 2015]: (1) DSL textuales (por ejemplo EMFText y Xtext) y (2) gráficos (por ejemplo MetaEdit+ y DSL Tools). La tendencia es disponer de herramientas que permitan crear DSL con una naturaleza híbrida que combine texto y gráficos.

Semántica	Interpretación (correspondencia semántica)	Relación de correspondencia con la realidad representada en el modelo.
	Derivación (teoría deductiva)	Relación con otros modelos derivables por medio de reglas deductivas de transformación
Sintaxis	Concreta (notación gráfica)	Aspecto visual: conjunto de símbolos gráficos utilizados para dibujar los diagramas.
	Abstracta (estructura lógica)	Estructura lógica: reglas que especifican las expresiones bien formadas de símbolos.

Tabla 2.1: Los cuatro aspectos de la definición de un lenguaje de modelado.

2.1.2.1. Lenguajes específicos de dominio (DSL)

Los lenguajes específicos del dominio (DSL, Domain-Specific Language), son lenguajes de programación o lenguajes de especificación que se dirigen a un dominio de problemas específico. No están destinados a proporcionar funciones para resolver todo tipo de problemas dentro del dominio para el que se ha creado. Probablemente no se pueda implementar todos los programas que se podrían implementar con Java o C, estos conocidos como lenguajes de propósito general (GPL, General-purpose Programming Language). Por otra parte, si el dominio del problema está cubierto por un DSL en particular, podrá resolver ese problema más fácil y rápidamente utilizando ese DSL en lugar de una GPL [Bettini, 2016]. Algunos ejemplos de DSL son SQL (para consultar bases de datos relacionales), Mathematica (para matemáticas simbólicas), HTML entre otros. Un programa o una especificación escrita en una DSL se puede interpretar o compilar en una GPL. En otros casos, la especificación puede representar datos simples que serán procesados por otros sistemas.

La estructura de un DSL se captura mediante su metamodelo, comúnmente conocido como su sintaxis abstracta. Un metamodelo es realmente un modelo que proporciona la base para construir otro modelo a partir de él. Por tanto, aunque ambos son modelos, uno se expresa en términos del otro. Es decir, un modelo es una instancia del otro o se ajusta al mismo. La sintaxis abstracta de un DSL se define utilizando el modelo Ecore de EMF (Eclipse Modeling Framework), que es, por tanto, su metamodelo y el modelo utilizado para definir todos los DSL. Un modelo creado en términos de la sintaxis abstracta del DSL se conoce comúnmente como una instancia del modelo; siendo el DSL, entonces, el metamodelo, y lo que convierte a Ecore en el meta-metamodelo. Ecore se expresa en términos de sí mismo [Dai et al., 2009].

El término sintaxis abstracta se refiere a un metamodelo, por lo que a menudo tiene una sintaxis concreta correspondiente en forma de notación de texto o diagrama. Estos se conocen como sintaxis concreta textual y sintaxis concreta gráfica, respectivamente. Una sintaxis textual permite a los usuarios trabajar con instancias de modelos tal como lo harían con otros lenguajes de programación basados en texto. Una sintaxis gráfica permite a los usuarios trabajar con las instancias de modelos utilizando mediante representaciones gráficas en forma de diagrama; el más popular es el lenguaje unificado UML de la OMG. Las sintaxis abstractas se definen mediante modelos Ecore, que a su vez se conservan en formato XMI (XML Metadata Interchange); técnicamente, esto podría considerarse una sintaxis concreta, aunque a veces se la denomina sintaxis de serialización.

2.1.2.2. El modelo Ecore

El modelo utilizado para representar modelos en EMF se llama Ecore. Ecore es un modelo EMF y, por lo tanto, es su propio metamodelo. Se podría decir que esto lo convierte en un meta-metamodelo. Un metamodelo es simplemente el modelo de un modelo, y si ese modelo es en sí mismo un metamodelo, entonces el metamodelo es de hecho un meta-metamodelo [Steinberg et al., 2008] [Khononov, 2019]. En la Figura 2.3 se muestra un subconjunto simplificado del metamodelo Ecore. Por ejemplo, en el metamodelo de Ecore real, las clases `EClass`, `EAttribute` y `EReference` comparten una clase base común, `ENamedElement`, que define el atributo de nombre que aquí se muestra explícitamente en las propias clases.

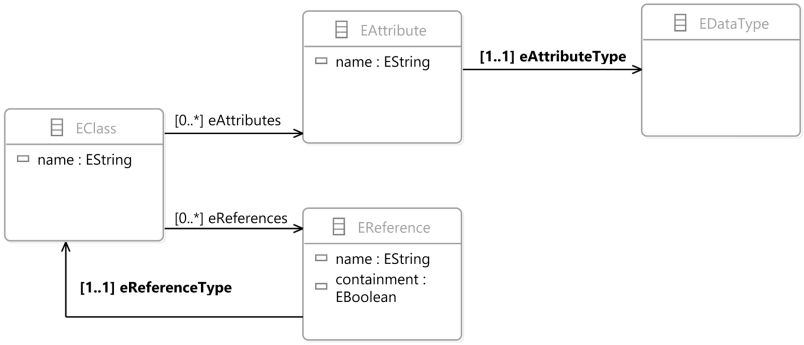


Figura 2.3: Conjunto simplificado del metamodelo Ecore.

Cuando se crea un modelo Ecore, se define la estructura de una serie de instancias de ese modelo. Es decir, se está especificando los tipos de objetos que componen las instancias de ese modelo, los datos que contienen y las relaciones entre ellos. El metamodelo Ecore define la estructura de los objetos en un modelo Ecore. Los modeladores usan el término "metamodelo" para este tipo de modelo, que, define un lenguaje que sus instancias usan para describir otras cosas .

Ecore tiene sus raíces en MOF (Meta-Object Facility) y UML, ambos soportados por la OMG, y fue diseñado para mapear limpiamente las implementaciones de Java. Ecore admite una serie de conceptos de nivel superior que no se incluyen directamente en Java. Por ejemplo, los modelos Ecore pueden incluir relaciones bidireccionales y de contención. Parte del valor de EMF es su capacidad para generar implementaciones de Java correctas y eficientes para estas y otras construcciones, lo que ahorra tiempo y esfuerzo al programador.

Como una instancia de cualquier otro modelo en EMF, un modelo Ecore se puede construir mediante programación o cargar desde un formulario serializado. Generalmente se usa en dos contextos diferentes: durante el desarrollo de la aplicación y cuando la aplicación se está ejecutando. Durante el desarrollo, el modelo Ecore es la principal fuente de información para el generador EMF, cuando produce código para ser utilizado en la aplicación [Steinberg et al., 2008].

2.1.3. Transformación de modelos

Una transformación es la generación automática de un modelo objetivo a partir de un modelo fuente de acuerdo con una definición de transformación, por medio de un conjunto de reglas de transformación que, juntas, describen cómo un modelo en el idioma origen puede transformarse en otro modelo en el idioma destino. Una regla de transformación es una descripción de cómo una o más construcciones en el idioma de origen pueden transformarse en una o más construcciones en el idioma de destino [Babau et al., 2009]. Esta definición cubre una amplia gama de actividades para las cuales se puede utilizar la transformación del modelo: generación automática de código, síntesis del modelo, evolución del modelo, simulación del modelo, ejecución del modelo, mejora de la calidad del modelo, traducción del modelo, pruebas basadas en modelos, verificación de modelos, verificación de modelos y muchos más.

MDE proporciona dos enfoques principales para transformar modelos: el primero Modelo a Modelo (M2M, Model-to-Model) y el segundo Modelo a Texto (M2T, Model-to-Text) (Figura 2.4). La transformación M2M ofrece la posibilidad de transformar modelos o una parte de un modelo en otro. Mientras que la transformación M2T proporciona una forma de transformar un modelo principalmente en código fuente. La transformación toma un modelo de entrada que se ajusta al metamodelo de origen y produce un modelo de salida, que se ajusta al metamodelo de destino. El metamodelo es el modelo del modelo, por lo que se utiliza el prefijo"meta" para indicar este concepto [Bouquet et al., 2014].

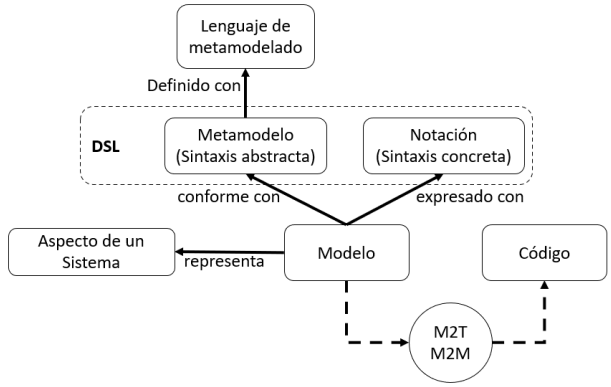


Figura 2.4: Relaciones entre modelo, metamodelo y meta-metamodelo (Fuente: adaptado de [Bouquet et al., 2014]).

La herramienta para transformar los modelos en el código que implementa el sistema son los lenguajes de generación de código que permiten trasladar la especificación del sistema, recogida en varios modelos, a un conjunto de ejecutables entendibles por la plataforma tecnológica elegida. Estos lenguajes son también lenguajes de transformación, que consumen varios modelos de entrada para producir otros modelos de salida. La particularidad está en que los modelos de salida tienen un nivel de abstracción muy bajo, pues se trata del código fuente que implementa el sistema [García-Molina et al., 2012].

En general, de las aproximaciones para el desarrollo de transformaciones, dos son las más frecuentemente adoptadas por los generadores de código. En primer lugar, la generación basada en plantillas es muy similar a la programación de páginas web dinámicas, como el desarrollo de páginas JSP (Java Server Pages) o ASP (Active Server Pages). Cada plantilla combina bloques de texto del código fuente que se quiere generar con sentencias y estructuras de control del lenguaje de generación utilizado. Las segundas se ocupan de combinar los bloques de texto con la información recuperada de los modelos de entrada [Bouquet et al., 2014].

Para ilustrar todos los conceptos mencionados en las secciones anteriores en la Figura 2.5 se ilustra un ejemplo de modelado de los animales de compañía. En este ejemplo se plantean cuatro etapas para el desarrollo de un DSL con generación de código:

— En la etapa 1 se desarrolla un metamodelo que describe el dominio de las mascotas de compañía y se definen dos clases (Perro y Gato).

— En la etapa 2 se crea una instancia del metamodelo. En este caso se ha creado un diagrama de árbol con el nombre de las mascotas.

— En la etapa 3 se crea el código en Acceleo para generar una clase con el nombre de la mascota en código Java.

— En la etapa 4 se observa el código generado acorde a la instancia definida en la etapa 2. En este caso se ha creado una clase con el nombre “Balto” que es el nombre de la instancia de la clase Perro.

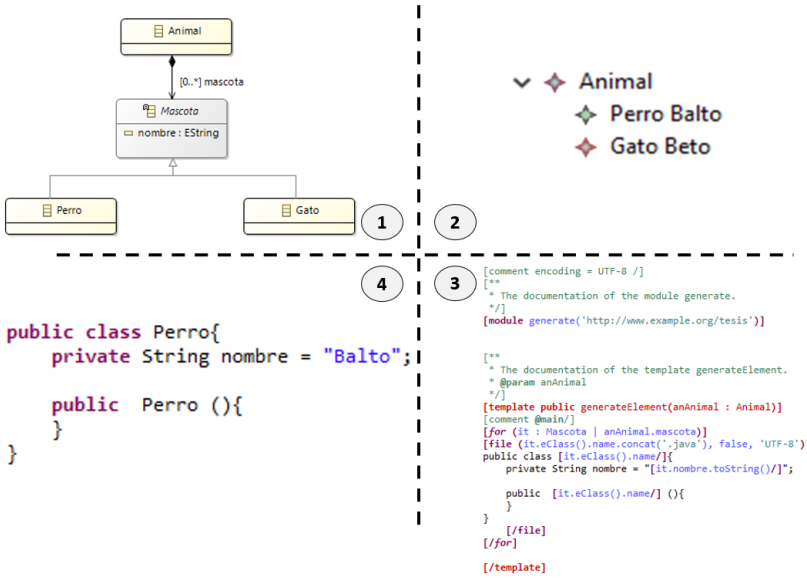


Figura 2.5: Ejemplo de transformación de modelos.

2.1.4. Lenguajes de transformación de modelos

Los lenguajes de transformación de modelos sirven para especificar la sintaxis y la semántica de las transformaciones de modelos, y son esenciales si se quiere proporcionar soporte automatizado para la transformación de modelos. Existe una amplia variedad de lenguajes de transformación de modelos. Muchos de ellos han surgido de la comunidad académica, mientras que otros se originan en la industria. En esta última categoría se encuentra, por ejemplo, la especificación QVT de OMG, que es compatible con el enfoque MDA basado en MOF y UML. Los lenguajes académicos incluyen, sin intentar ser completos [Babau et al., 2009]: ATL, Kermeta, Tefkat, SiTra y muchos lenguajes que se basan en el enfoque subyacente de la transformación gráfica (por ejemplo, ATOM3, AGG, Fujaba, GReAT, MOFLON, VIATRA2).

2.1.5. Herramientas de desarrollo con modelos

Actualmente existen varias alternativas de herramientas que se encuentran maduras para el desarrollo de software bajo el paradigma de MDE. Sin embargo una de las propuestas que ha tenido gran acogida es la de Eclipse Foundation, ya que es una alternativa de Software Libre. Algunas de las herramientas disponibles son Sirius y Acceleo.

Sirius. Es un proyecto de Eclipse que permite crear herramientas de modelado gráfico. Emplea las tecnologías de modelado EMF para la gestión de modelos y GMF para la representación gráfica. Sirius es el resultado de una colaboración entre Thales Group y Obeo iniciada en 2007. El objetivo de Thales era obtener un framework de trabajo genérico para la ingeniería de arquitectura basada en modelos que pudiera adaptarse a las necesidades específicas de cada proyecto. La tecnología Sirius fue empaquetada y distribuida por Obeo como parte del producto Obeo Designer [Sirius, 2020].

Acceleo. Es un generador de código de código libre que permite utilizar un enfoque basado en modelos para crear aplicaciones. Es una implementación del estándar MOFM2T de OMG para realizar la transformación de modelo a texto. El proyecto Acceleo nace en 2006 con la licencia GNU Public License (GPL) compatible con Eclipse y varios modeladores basados en EMF y UML. Acceleo proporciona herramientas para la generación de código a partir de modelos basados en EMF. Por ejemplo, la generación incremental. La generación incremental brinda a las personas la capacidad de generar un fragmento de código y luego modificar el código generado y finalmente regenerar el código una vez más sin perder las modificaciones anteriores (Acceleo, 2020).

2.2. Estilos de integración

Si las necesidades de integración fueran siempre las mismas, solo habría un estilo de integración. Sin embargo, como cualquier esfuerzo tecnológico complejo, la integración de aplicaciones implica una serie de consideraciones y consecuencias que deben tenerse en cuenta para cualquier oportunidad de integración. Los siguientes son algunos criterios para la decisión de cómo realizar la integración [Hohpe, 2003]:

— *Acoplamiento de aplicaciones:* las aplicaciones integradas deben minimizar sus dependencias entre sí para que cada una pueda evolucionar sin causar problemas.

— *Intrusión:* al integrar una aplicación, los desarrolladores deben esforzarse por minimizar los cambios en la aplicación y la cantidad de código de integración.

— *Selección de tecnología:* las diferentes técnicas de integración requieren distintas cantidades de software y hardware especializados. Dichas herramientas pueden ser costosas, pueden generar un bloqueo del proveedor y pueden aumentar la curva de aprendizaje para los desarrolladores.

— *Formato de datos:* las aplicaciones integradas deben acordar el formato de los datos que intercambian.

— *Datos o funcionalidad:* muchas soluciones de integración permiten que las aplicaciones compartan no solo datos, sino también funcionalidad, porque el intercambio de funcionalidad puede proporcionar una mejor abstracción de las aplicaciones.

— *Comunicación remota:* el procesamiento suele ser sincrónico, es decir, un procedimiento espera mientras se ejecuta su subprocedimiento. Sin embargo, llamar a un subprocedimiento remoto es mucho más lento que a uno local, por lo que es posible que un procedimiento no desee esperar a que se complete el subprocedimiento; en su lugar, puede querer invocar el subprocedimiento de forma asíncrona, es decir, iniciar el subprocedimiento, pero continuar con su propio procesamiento simultáneamente. La sincronicidad puede generar una solución mucho más eficiente, pero dicha solución también es más compleja de diseñar, desarrollar y depurar.

— *Fiabilidad:* las conexiones remotas no solo son lentas, sino que son mucho menos fiables que una llamada de función local. Cuando un procedimiento llama a un subprocedimiento dentro de una sola aplicación, es un hecho que el subprocedimiento está disponible.

Como se puede ver hay varios criterios diferentes que se deben considerar al elegir y diseñar un enfoque de integración. No existe un enfoque de integración que aborde todos los criterios por igual. Por lo tanto, con el tiempo han evolucionado múltiples enfoques para integrar aplicaciones.

2.2.1. Transferencia de archivos

La gran ventaja de los archivos es que los integradores no necesitan conocer los aspectos internos de una aplicación. El propio equipo de la aplicación suele proporcionar el archivo. El contenido y el formato del archivo se negocian con los integradores. Luego, los integradores se ocupan de las transformaciones necesarias para otras aplicaciones, o dejan que las aplicaciones consumidoras decidan cómo quieren manipular y leer el archivo. Como resultado, las diferentes aplicaciones están muy bien desacopladas entre sí. Cada aplicación puede realizar cambios internos libremente sin afectar a otras aplicaciones, siempre que sigan produciendo los mismos datos en los archivos en el mismo formato. Los archivos se convierten en la interfaz pública de cada aplicación.

Figura 2.6: Transferencia de archivos (Fuente: adaptado de [Hohpe, 2003]).

Uno de los problemas más obvios con la transferencia de archivos es que las actualizaciones tienden a producirse con poca frecuencia y, como resultado, los sistemas pueden perder la sincronización. De hecho, el mayor problema con la desactualización suele estar en el propio personal de desarrollo de software, que con frecuencia debe lidiar con datos que no son del todo correctos. Esto puede dar lugar a incoherencias difíciles de resolver. Sin embargo, no hay ninguna razón por la que no pueda producir archivos con más frecuencia. El problema es administrar todos los archivos que se producen, asegurando que se lean todos y que ninguno se pierda. Esto va más allá de lo que pueden hacer los enfoques basados en el sistema de archivos, especialmente porque hay costosos recursos asociados con el procesamiento de un archivo, que pueden resultar prohibitivos si desea producir muchos archivos rápidamente.

2.2.2. Base de datos compartida

La base de datos compartida se hace mucho más fácil debido al uso generalizado de bases de datos relacionales basadas en SQL (Structured Query Language). Prácticamente todas las plataformas de desarrollo de aplicaciones pueden trabajar con SQL. Dado que todas las aplicaciones utilizan la misma base de datos, esto elimina los problemas de disonancia semántica. En lugar de dejar que estos problemas se agraven hasta que sean difíciles de resolver con transformaciones, el desarrollador se ve obligado a enfrentarlos y lidiar con ellos antes de que el software entre en funcionamiento y recopile grandes cantidades de datos incompatibles.

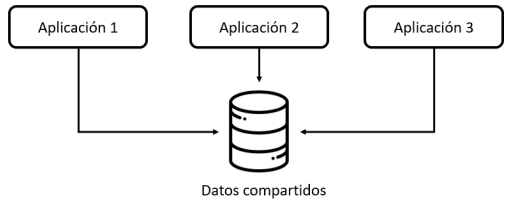


Figura 2.7: Base de datos compartida (Fuente: adaptado de [Hohpe, 2003]).

Una de las mayores dificultades con la base de datos compartida es encontrar un diseño adecuado para la base de datos compartida. Otro límite más difícil para la base de datos compartida son los paquetes externos, que agrupan procedimientos y funciones adicionales, para la casuística asociada a un determinado tipo de tarea. Por ejemplo, si un conjunto de procedimientos y funciones para realizar cálculos matemáticos complejos.

La mayoría de las aplicaciones empaquetadas no funcionarán con un esquema que no sea el suyo. Además, varias aplicaciones que usan una base de datos compartida para leer y modificar con frecuencia los mismos datos pueden convertir la base de datos en un cuello de botella de rendimiento y pueden causar interbloqueos ya que cada aplicación bloquea a otras fuera de los datos. Cuando las aplicaciones se distribuyen en varias ubicaciones, el acceso a una única base de datos compartida en una red de área amplia suele ser demasiado lento para ser práctico.

2.2.3. Invocación de procedimientos remoto

La invocación de procedimiento remoto aplica el principio de encapsulación para integrar aplicaciones. Si una aplicación necesita información que es propiedad de otra aplicación, la solicita directamente. Si una aplicación necesita modificar los datos de otra, lo hace haciendo una llamada a la otra aplicación. Esto permite que cada aplicación mantenga la integridad de los datos que posee. Además, cada aplicación puede alterar el formato de sus datos internos sin afectar a todas las demás aplicaciones.

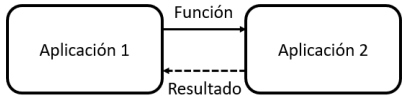


Figura 2.8: Invocación de procedimientos remoto (Fuente: adaptado de [Hohpe, 2003]).

El hecho de que haya métodos que envuelvan los datos hace que sea más fácil lidiar con la disonancia semántica. Las aplicaciones pueden proporcionar múltiples interfaces a los mismos datos, permitiendo que algunos clientes vean un estilo y otros uno diferente. Incluso las actualizaciones pueden utilizar múltiples interfaces. Esto proporciona mucha más capacidad para respaldar múltiples puntos de vista que la que se puede lograr con las vistas relacionales. Sin embargo, es incómodo para los integradores agregar componentes de transformación, por lo que cada aplicación tiene que negociar su interfaz con las aplicaciones cercanas.

2.2.4. Mensajería

La mensajería asíncrona es fundamentalmente una reacción a los problemas de los sistemas distribuidos. Enviar un mensaje no requiere que ambos sistemas estén activos y listos al mismo tiempo. Además, pensar en la comunicación de forma asíncrona obliga a los desarrolladores a reconocer que trabajar con una aplicación remota es más lento, lo que fomenta el diseño de componentes con alta cohesión y baja adhesión.

Los sistemas de mensajería también permiten gran parte del desacoplamiento que se obtiene al utilizar transferencia de archivos. Los mensajes se pueden transformar en tránsito sin que el remitente ni el receptor sepan de la transformación. El desacoplamiento permite a los integradores elegir entre transmitir mensajes a múltiples receptores, enrutar un mensaje a uno de muchos receptores u otras topologías. Esto separa las decisiones de integración del desarrollo de las aplicaciones.

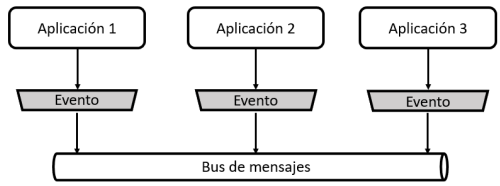


Figura 2.9: Mensajería (Fuente: adaptado de [Hohpe, 2003]).

2.3. Orientación al servicio

La orientación al servicio es un paradigma de diseño destinado a la creación de unidades lógicas que se configuran individualmente para que puedan utilizarse de manera colectiva y reiterada. La lógica de acuerdo con la orientación al servicio puede calificarse como Orientada a Servicio, y las unidades de lógica se denominan "servicios" [Erl et al., 2012]. El paradigma del diseño de orientación al servicio se compone principalmente de ocho principios de diseño específicos:

— *Contrato de servicio estandarizado*: Los servicios dentro del mismo inventario de servicios cumplen con los mismos estándares de diseño del contrato.

— *Acoplamiento ligero del servicio*: Los contratos de servicio imponen requisitos bajos de acoplamiento del consumidor y están desacoplados de su entorno.

— *Abstracción de servicio*: Los contratos de servicio solo contienen información esencial y la información sobre los servicios se limita a lo que se publica en los contratos de servicio.

— *Reutilización del servicio*: Los servicios contienen y expresan lógica agnóstica y pueden posicionarse como recursos empresariales reutilizables.

— *Autonomía de servicio*: Los servicios ejercen un alto nivel de control sobre su entorno de ejecución de tiempo de ejecución subyacente.

— *Sin estado*: Los servicios minimizan el consumo de recursos al diferir la gestión de la información del estado cuando sea necesario.

— *Descubrimiento del servicio*: Los servicios se complementan con metadatos comunicativos por los cuales se pueden descubrir e interpretar de manera efectiva.

— *Composición del servicio*: Los servicios son participantes efectivos de la composición, independientemente del tamaño y la complejidad de la composición.

2.3.1. Arquitectura Orientada a Servicios (SOA)

Una implementación de la Arquitectura Orientada a Servicios (SOA, Service Oriented Architecture) puede consistir en una combinación de tecnologías, productos, API, extensiones de infraestructura de soporte y varias otras partes. La complexión real de una

arquitectura orientada a servicios desplegada es única dentro de cada empresa; sin embargo, se caracteriza por la introducción de nuevas tecnologías y plataformas que apoyan específicamente la creación, ejecución y evolución de soluciones orientadas a servicios. La construcción de una arquitectura tecnológica con el modelo SOA establece un entorno adecuado para la lógica de la solución que se ha diseñado de conformidad con los principios de diseño de orientación al servicio [Erl et al., 2012]. Después del modelado conceptual del servicio, el diseño orientado al servicio y las etapas de desarrollo implementan un servicio como un programa físicamente independiente con características de diseño específicas. A cada servicio se le asigna su propio contexto funcional distinto y se compone de un conjunto de capacidades relacionadas con el contexto.

La orientación al servicio es neutral para cualquier plataforma tecnológica. Por lo que cualquier tecnología de implementación puede usarse para crear un sistema distribuido para la aplicación de la orientación al servicio. Al diseñar servicios, hay diferentes niveles de granularidad que deben tenerse en cuenta de la siguiente manera [Erl et al., 2012]:

— *Granularidad del servicio*: representa el alcance funcional de un servicio. Por ejemplo, la granularidad fina del servicio indica que hay una pequeña cantidad de lógica asociada con el contexto funcional general del servicio.

— *Granularidad de capacidad*: el alcance funcional de las capacidades de servicio individuales está representado por este nivel de granularidad.

— *Granularidad de restricción*: el nivel de detalle de la lógica de validación se mide por la granularidad de restricción. Por ejemplo, cuanto más gruesa sea la granularidad de la restricción, menos restricciones tendrá una capacidad determinada.

— *Granularidad de datos*: este nivel de granularidad representa la cantidad de datos procesados. Por ejemplo, un nivel fino de granularidad de datos es equivalente a una pequeña cantidad de datos.

Debido a que el nivel de granularidad del servicio determina el alcance funcional de un servicio, generalmente se determina durante las etapas de análisis y modelado que preceden al diseño del contrato de servicio. Una vez que se ha establecido el alcance funcional de un servicio, entran en juego los otros tipos de granularidad y afectan tanto el modelado como el diseño físico de un contrato de servicio.

2.3.2. Arquitectura de Microservicios (MSA)

El patrón arquitectónico orientado a servicios ha evolucionado durante décadas para expresar y exponer cualquier aplicación heredada, monolítica y masiva como una colección dinámica de servicios interdependientes. Los servicios están dotados de interfaces e implementaciones. Las interfaces son el mecanismo de contacto para cualquier aplicación habilitada para el servicio.

El estilo SOA se basa principalmente en un modelo de datos compartidos con múltiples jerarquías. El intercambio de datos en SOA crea un acoplamiento de datos fuerte entre los servicios y otros componentes del sistema, y esto afecta a la forma de producir

los cambios deseados en la base de datos. Otro desafío es que los servicios SOA son típicamente de grano grueso y, por lo tanto, el aspecto de la reutilización es un asunto bastante difícil. La mayoría de los componentes de las aplicaciones heredadas están equipados con interfaces orientadas a servicios para permitir el descubrimiento, la integración y las interacciones. A nivel de infraestructura, no hay problema de dependencia porque estos componentes de aplicaciones habilitados para servicios pueden literalmente ejecutarse en cualquier lugar. SOA utiliza una especie de arquitectura organizativa escalonada que contiene un middleware de mensajería centralizado para la invocación, intermediación y coordinación de servicios. Pero el problema es que los componentes de la aplicación necesitan conocer detalles para iniciar y establecer la colaboración [Raj et al., 2017].

La arquitectura de microservicios (MSA, MicroServices Architecture) es un patrón arquitectónico para definir, diseñar, desarrollar, implementar y desplegar aplicaciones de software distribuidas y de nivel empresarial. Este patrón de rápida aparición y evolución permite lograr de manera fácil y rápida los requisitos no funcionales, como la escalabilidad, la disponibilidad y la confiabilidad para cualquier aplicación de software. El patrón MSA es útil para producir servicios detallados, débilmente acoplados, escalables horizontalmente, implementables de forma independiente, interoperables, públicamente reconocibles, accesibles en red, fácilmente administrables y componibles.

Los microservicios se basan en un concepto conocido como contexto acotado, que permite la asociación autónoma entre un solo servicio y sus datos. No existe una restricción de tecnología o proveedor en lo que respecta a las aplicaciones inspiradas en MSA. Los microservicios se basan únicamente en la comunicación entre servicios. Cada microservicio llama a otro microservicio según sea necesario para completar su función. Además, los microservicios llamados pueden llamar a otros servicios según sea necesario en un proceso conocido como encadenamiento de servicios. Los microservicios utilizan una capa de API no coordinada sobre los servicios que componen una aplicación.

Al comparar los microservicios con SOA, ambos se basan en los servicios como componente principal, pero la principal distinción entre los dos enfoques se reduce al alcance. La arquitectura orientada a servicios es esencialmente una colección de servicios. Estos servicios se comunican entre sí. La comunicación puede implicar el paso de datos simples o dos o más servicios que coordinan alguna actividad. Una arquitectura de microservicios es un estilo arquitectónico que estructura una aplicación como una colección de pequeños servicios autónomos modelados en torno a un dominio particular [IBM, 2017a]. En la Tabla 2.2 y la Figura 2.10 se muestran algunas diferencias entre SOA y microservicios.

	SOA	**Microservicios**
Comunicación entre servicios	Buses inteligentes, como Enterprise Service Bus, con protocolos pesados, como SOAP y otros estándares WS	Buses tradicionales, como un agente de mensajes, o comunicación directa de servicio a servicio, usa protocolos ligeros como REST o gRPC
Datos	Modelo de datos global y bases de datos compartidas.	Modelo de datos y base de datos por servicio.
Servicio típico	Aplicación monolítica más grande.	Servicio más pequeño.

Tabla 2.2: Comparación de SOA con microservicios.

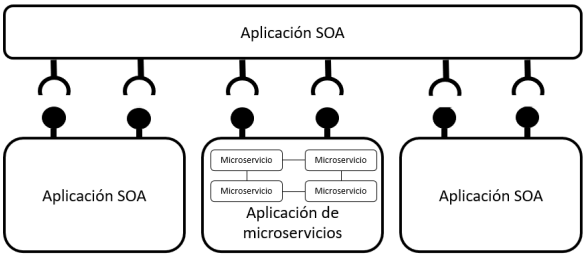


Figura 2.10: Diferencia entre SOA y Microservicios.

2.3.3. Arquitecturas Orientadas a los Recursos (ROA)

Un recurso se puede definir como cualquier entidad relevante en el dominio de una aplicación que está expuesta en la red (una página web, un vídeo o un pedido en un sitio web de comercio electrónico pueden considerarse recursos web). Por lo que, un recurso es cualquier cosa con la que un usuario interactúa mientras progresa hacia algún objetivo. Una alternativa a una SOA es una arquitectura orientada a servicios (ROA, Resource-Oriented Architecture). Una implementación basada en SOA se refiere a una arquitectura donde dos puntos finales se comunican a través de un conjunto predefinido de contratos de mensajería. Sin embargo, esto podría suponer un problema, porque si, por ejemplo, un servidor cambia sus servicios, el cliente debería necesitar obtener acceso al nuevo servicios web o se invalidan sus funcionalidades. En una ROA, por otro lado, no hay servicios de exposición de punto final; solo hay recursos que pueden ser manipulados, como ejemplo REST (REpresentational State Transfer) se basa en el concepto de un recurso [Cirani et al., 2018].

2.3.4. Representational State Transfer (REST)

El estilo REST (Representational State Transfer) es un conjunto de prácticas de ingeniería de software que contiene restricciones que deben usarse para crear servicios web en sistemas hipermedia distribuidos. REST no es una herramienta ni es un lenguaje; de hecho, REST es independiente de protocolos, componentes y lenguajes. Es importante decir que REST es un estilo arquitectónico y no un conjunto de herramientas. REST proporciona un conjunto de reglas de diseño para crear servicios sin estado que se muestran como recursos y, en algunos casos, fuentes de información específica como datos y funcionalidad. La identificación de cada recurso se realiza mediante su identificador uniforme de recursos (URI) único . REST describe interfaces simples que transmiten datos a través de una interfaz estandarizada como HTTP y HTTPS sin ninguna capa de mensajería adicional, como el protocolo SOAP [Nikaj et al., 2019]. El estilo arquitectónico REST describe seis limitaciones. Estas limitaciones fueron descritas originalmente por Roy Fielding en su tesis doctotal [Fielding, 2020]:

— *Interfaz uniforme*: la interfaz uniforme es una restricción que describe un contrato entre clientes y servidores (Figura 2.11). Una de las razones para crear una in-

terfaz entre ellos es permitir que cada parte evolucione independientemente de la otra. Una vez que existe un contrato alineado con las partes cliente y servidor, pueden comenzar sus trabajos de forma independiente porque, la forma en que se comunicarán se basa firmemente en la interfaz.

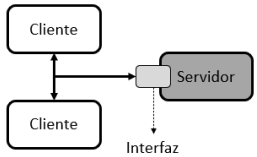


Figura 2.11: Esquema de Interfaz uniforme (Fuente: adaptado de [Muniz, 2019]).

— *Stateless*: esencialmente, stateless significa que el estado necesario durante la solicitud está contenido dentro de la solicitud (Figura 2.12). Básicamente, el URI es el identificador único del destinatario y el cuerpo contiene el estado o el recurso. Stateless permite una alta escalabilidad ya que el servidor no mantendrá sesiones. Otro punto interesante a tener en cuenta es que al equilibrador de carga no le importan las sesiones en los sistemas sin estado.

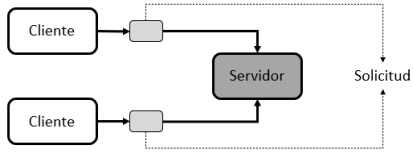


Figura 2.12: Esquema Stateless (Fuente: adaptado de [Muniz, 2019]).

— *Cacheable*: el almacenamiento en caché permite no tener que generar la misma respuesta más de una vez (Figura 2.13). Los beneficios de usar esta estrategia son un aumento en la velocidad y una reducción en el procesamiento del servidor.

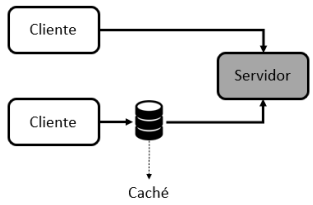


Figura 2.13: Esquema Cacheable (Fuente: adaptado de [Muniz, 2019]).

— *Arquitectura cliente-servidor*: el estilo REST separa a los clientes de un servidor (Figura 2.14). Siempre que sea necesario reemplazar el lado del servidor o del

cliente, las cosas deberían fluir de forma natural ya que no hay ningún acoplamiento entre ellos. El lado del cliente no debería preocuparse por el almacenamiento de datos y el lado del servidor no debería preocuparse por la interfaz.

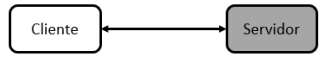


Figura 2.14: Esquema Cliente-Servidor (Fuente: adaptado de [Muniz, 2019]).

— *Sistema en capas*: cada capa debe funcionar de forma independiente e interactuar solo con las capas directamente conectadas a ella (Figura 2.15). Esta estrategia permite pasar la solicitud sin pasar por alto otras capas. Por ejemplo, cuando se desea escalar un servicio, puede usar un proxy que funcione como un equilibrador de carga; de esa manera, las solicitudes entrantes se pueden entregar a la instancia de servidor adecuada. Siendo ese el caso, el lado del cliente no necesita entender cómo va a funcionar el servidor; solo realiza solicitudes al mismo URI.

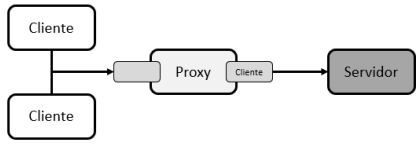


Figura 2.15: Esquema del Sistema en capas (Fuente: adaptado de [Muniz, 2019]).

— *Código bajo demanda*: este patrón permite al cliente descargar y ejecutar código desde el servidor en el lado cliente. Esta estrategia mejora la escalabilidad ya que el código se puede ejecutar independientemente del servidor en el lado cliente.

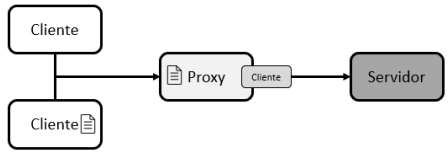


Figura 2.16: Esquema de Código bajo demanda (Fuente: adaptado de [Muniz, 2019]).

2.3.5. Servicios Web RESTful

Los servicios web RESTful son servicios web basados en la arquitectura REST. Los servicios web RESTful se basan en recursos. Un recurso es una entidad, que se almacena principalmente en el servidor, el cliente solicita el recurso utilizando los servicios web RESTful. Aunque los términos REST y RESTful se utilizan a menudo como sinónimos, no lo son. REST es un modelo de arquitectura web basado en el protocolo HTTP que

puede mejorar la comunicación entre cliente y servidor, mientras que Restful Web Service o Restful API es un programa basado en REST. Las características principales de los servicios web RESTful son las siguientes (Muniz, 2019):

— Dispone de cinco operaciones: consultar, crear, leer, actualizar y eliminar.

— Cada operación requiere dos elementos: método URI y HTTP.

— URI es un sustantivo que contiene el nombre del recurso.

— El método HTTP es un verbo.

El uso de verbos HTTP permite una comprensión clara de lo que va a hacer una operación. En general, los verbos HTTP primarias o más comúnmente usados son POST, GET, PUT, PATCH, y DELETE (Tabla 2.3), que representan crear, leer, actualizar (PATCH y PUT), y eliminar, respectivamente. Por supuesto, también hay muchos otros verbos, pero no se usan con tanta frecuencia [Muniz, 2019].

MÉTODO	DESCRIPCIÓN
`GET`	Su función es recuperar datos de un servidor en el recurso especificado.
`HEAD`	Similar al método GET excepto que el servidor responde sin el cuerpo del mensaje.
`POST`	Permite crear un nuevo recurso.
`PATCH`	Se utiliza para aplicar modificaciones parciales a un recurso.
`PUT`	A diferencia del método PATCH, este método reemplaza todo el recurso.
`DELETE`	Elimina un recurso.
`CONNECT`	Convierte la solicitud de conexión en un túnel de transporte TCP/IP, para facilitar la comunicación cifrada.
`OPTIONS`	Retorna los métodos HTTp admitidos por el servidor para la URL especificada.
`TRACE`	Retorna la misma solicitud que se envía para determinar si existieron cambios por servidores intermediarios.

Tabla 2.3: Métodos HTTP para servicios RESTful.

2.4. Integración de procesos con microservicios

Los servicios pueden utilizar mecanismos de comunicación síncronos basados en solicitudes/respuestas, como REST basado en HTTP. Alternativamente, pueden usar mecanismos de comunicación asíncronos basados en mensajes como AMQP (Advanced Message Queuing Protocol). Se debe considerar el que primero el estilo de interacción entre un servicio y sus clientes antes de seleccionar un mecanismo de IPC (Inter-Process Communication) para la API de un servicio. Pensar primero en el estilo de interacción permite enfocarse en los requisitos y evitar enredarse en los detalles de una tecnología IPC particular. Además, la elección del estilo de interacción afecta la disponibilidad de la aplicación. Además, ayuda a seleccionar la estrategia de prueba de integración adecuada [Richardson, 2018]. La interacción cliente/servidor puede darse de cuatro formas:

— *Uno a uno*: cada solicitud de cliente es procesada por exactamente un servicio. Tipos de interacciones uno a uno:

  - Solicitud/respuesta: un cliente del servicio realiza una solicitud a un servicio y espera una respuesta. A continuación, el cliente espera que la respuesta llegue de manera oportuna. Puede suceder que se bloquee mientras espera. Este es un estilo de interacción generalmente da como resultado servicios estrechamente acoplados.
  - Solicitud/respuesta asíncrona: un cliente de servicio envía una solicitud a un servicio, que responde de forma asíncrona. El cliente no bloquea mientras espera, porque el servicio podría no enviar la respuesta durante mucho tiempo.
  - Notificaciones en un solo sentido: Un cliente de servicio envía una solicitud a un servicio, pero no se espera ni se envía respuesta.

— *Uno a muchos*: cada solicitud es procesada por múltiples servicios. Tipos de interacciones uno a muchos:

  - Publicar/suscribirse: un cliente publica un mensaje de notificación, que es consumido por cero o más servicios interesados.
  - Publicar/respuestas asíncronas: un cliente publica un mensaje de solicitud y espera un cierto tiempo para recibir respuestas de los servicios interesados.

— *Síncrono*: el cliente espera una respuesta oportuna del servicio e incluso puede bloquear mientras espera.

— *Asíncrono*: el cliente no bloquea y la respuesta, si la hay, no se envía necesariamente de inmediato.

2.4.1. El patrón Sagas: Transacciones en microservicios

La gestión de transacciones es sencilla en una aplicación monolítica que accede a una única base de datos. La gestión de transacciones es más desafiante en una aplicación monolítica compleja que utiliza múltiples bases de datos y mensajes. En una arquitectura de microservicio, las transacciones abarcan múltiples servicios, cada uno de los cuales tiene su propia base de datos. En esta situación, la aplicación debe usar un mecanismo más elaborado para administrar las transacciones.

Desde la perspectiva del desarrollador, las transacciones distribuidas tienen el mismo modelo de programación que las transacciones locales. Para resolver el problema de mantener la consistencia de los datos en una arquitectura de microservicio, una aplicación debe usar un mecanismo diferente que se base en el concepto de servicios asíncronos y poco acoplados. Para ello se usa el patrón Saga como mecanismo para mantener la consistencia de datos en una arquitectura de microservicio sin tener que usar transacciones distribuidas. En consecuencia una Saga es una secuencia de transacciones locales, en la que cada transacción local actualiza los datos dentro de un solo servicio utilizando los frameworks y las bibliotecas de transacciones ACID (Atomicity, Consistency, Isolation, Durability) [Richardson, 2018].

Saga mantiene la coherencia de los datos en todos los servicios mediante una secuencia de transacciones locales que se coordinan mediante mensajes asíncronos. La operación del sistema inicia el primer paso de la saga. La finalización de una transacción local desencadena la ejecución de la siguiente transacción local. Un beneficio importante de la mensajería asíncrona es que garantiza que se ejecuten todos los pasos de una saga, incluso si uno o más de los participantes de la saga no están disponibles temporalmente. Las sagas difieren de las transacciones ACID en que carecen de la propiedad de aislamiento y debido a que cada transacción local confirma sus cambios, una saga debe revertirse utilizando transacciones compensatorias.

2.4.2. Coordinación de SAGAS

La implementación de una Saga consiste en una lógica que coordina los pasos de la Saga. Cuando un comando del sistema inicia una Saga, la lógica de coordinación debe seleccionar y decirle al primer participante de la Saga que ejecute una transacción local. Una vez que se completa esa transacción, la coordinación de secuencia de la Saga selecciona e invoca al siguiente participante de la Saga. Este proceso continúa hasta que la saga haya ejecutado todos los pasos (Figura 2.17). Si alguna transacción local falla, la Saga debe ejecutar las transacciones compensatorias en orden inverso. Aún cuando Saga presenta ventajas es necesario identificar los escenario en los cuales es más o menos conveniente su uso. La Tabla 2.4 muestra un resumen de cuando usar el patrón Saga.

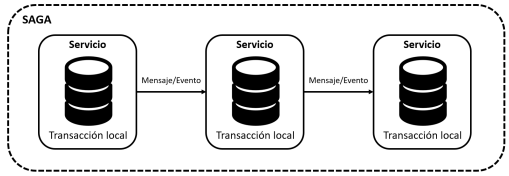


Figura 2.17: Patrón Saga. (Fuente: adaptado de [Microsoft, 2020])

Más adecuado	Menos adecuado
Para garantizar la coherencia de los datos en un sistema distribuido sin un acoplamiento estrecho. Para revertir o compensar si falla una de las operaciones de la secuencia.	En transacciones estrechamente acopladas. Cuando exista compensación de transacciones que ocurren en participantes anteriores. Cuando existan dependencias cíclicas.

Tabla 2.4: Cuando usar el patrón Saga.

Para la implementación del patrón Saga existen dos formas diferentes de estructurar la lógica de coordinación [Malyuga et al., 2020]: *(a)* Coreografía de servicios y *(b)* Orquestación de servicios.

— *Coreografía*: en este caso, los microservicios están vinculados a la cadena de llamadas secuenciales (Figura 2.18). Cada servicio realizará cambios en el evento o solicitud, emite un evento exitoso y espera a que toda la saga continúe. En caso

de falla durante los cambios locales, el servicio debe emitir un evento de falla al servicio anterior. Este servicio anterior debe aplicar la lógica de compensación y enviar un evento de falla a un servicio anterior. A continuación se listan algunas de las ventajas e inconvenientes de la Coreografía [Microsoft, 2020].

Ventajas:

- Bueno para flujos de trabajo simples que requieren pocos participantes y no necesitan una lógica de coordinación.
- No requiere implementación ni mantenimiento de servicios adicionales.
- No introduce un solo punto de falla, ya que las responsabilidades se distribuyen entre los participantes de la saga.

Inconvenientes:

- El flujo de trabajo puede volverse confuso al agregar nuevos pasos, ya que es difícil rastrear qué participantes de la saga escuchan qué comandos.
- Existe el riesgo de dependencia cíclica entre los participantes de la saga porque tienen que consumir los comandos de los demás.
- Las pruebas de integración son difíciles porque todos los servicios deben estar ejecutándose para simular una transacción.

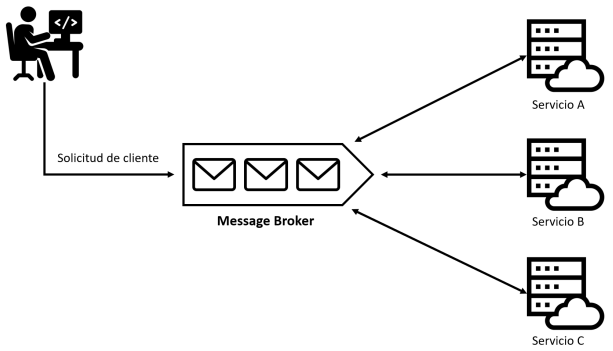


Figura 2.18: Coreografía de servicios. (Fuente: adaptado de [Microsoft, 2020])

— *Orquestación*: este tipo de sagas requieren la implementación del orquestador que controla las llamadas de sagas secuenciales (Figura 2.19). El Orquestador envía comandos a los servicios en la cadena uno por uno, esperando respuesta sucesiva. En caso de falla, envía llamadas de compensación a los servicios donde ya se realizaron cambios para devolver dichos servicios al estado anterior. A continuación se listan algunos de las ventajas e inconvenientes de la Orquestación [Microsoft, 2020].

Ventajas:

- Bueno para flujos de trabajo complejos que involucran a muchos participantes o nuevos participantes agregados con el tiempo.

- Adecuado cuando hay control sobre todos los participantes en el proceso y control sobre el flujo de actividades.
- No introduce dependencias cíclicas, porque el orquestador depende unilateralmente de los participantes de la saga.
- Los participantes no necesitan conocer los comandos de otros participantes. La clara separación de preocupaciones simplifica la lógica de negocio.

Inconvenientes:

- La complejidad adicional del diseño requiere la implementación de una lógica de coordinación.
- Hay un punto adicional de falla, porque el orquestador administra el flujo de trabajo completo.

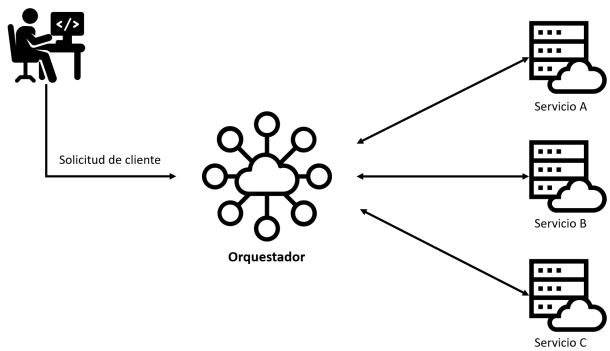


Figura 2.19: Orquestación de servicios. (Fuente: adaptado de [Microsoft, 2020])

2.5. Procesamiento de eventos

Las arquitecturas basadas en servicios, como microservicios o SOA, se suelen construir con protocolos de solicitud-respuesta síncronos. Este enfoque es muy natural. Después de todo, es la forma en que se escribe programas: se hace llamadas a otros módulos de código, se espera una respuesta y se continua. Pero cuando se entra en un mundo de muchos servicios independientes, las cosas comienzan a cambiar. A medida que el número de servicios crece gradualmente, la red de interacciones sincrónicas crece con ellos. Los problemas de disponibilidad previamente benignos comienzan a desencadenar interrupciones mucho más generalizadas. Este es un problema en el cual se puede simplemente romper los lazos síncronos que unen los servicios mediante asincronía [Stopford, 2018].

2.5.1. Eventos

Un evento es cualquier cosa que se puede observar que ocurre en un punto particular en el tiempo [Crettaz and Dean, 2019]. La Figura 2.20 presenta cuatro ejemplos de eventos

de cuatro sectores empresariales diferentes. Contrario al significado de un evento, se considera que ninguno de los siguientes ejemplos es un evento:

— Una descripción del estado actual de algo: el día era cálido; el auto estaba negro; El cliente API estaba roto. Pero "el cliente API se rompió al mediodía del martes" es un evento.

— Una ocurrencia recurrente - la bolsa de valores abrió a las 09:30 todos los días en 2020. Pero cada apertura individual de la bolsa de valores en 2020 es un evento.

— Una colección de eventos individuales: la guerra del Cenepa involucró a Ecuador y Perú. Pero la guerra fue declarada el 26 de enero de 1995, lo cual es un evento.

— Un suceso que abarca un período de tiempo: la semana Santa del 2020 se realizó del domingo 5 de abril al sábado 11 de abril del 2020. Pero el comienzo y el son eventos individuales.

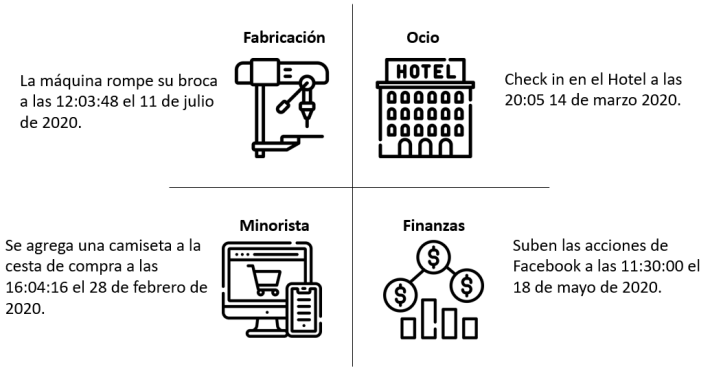


Figura 2.20: Ejemplos de eventos.

2.5.2. Productor de eventos

Cuando se habla de un productor de eventos, es importante identificar que se puede usar el concepto para referirse a tres cosas ligeramente diferentes [Etzion and Niblett, 2010]:

— Un tipo abstracto de productor de eventos, por ejemplo, es un sensor de GPS (Global Positioning System).

— Una colección de instancias de productor de eventos, todas del mismo tipo, que aparecen en una aplicación determinada, por ejemplo, todos los sensores GPS.

— Una instancia única de un productor de eventos, por ejemplo, el sensor GPS dentro del vehículo de un conductor específico.

Los productores reales constituyen una parte importante de la aplicación de procesamiento de eventos. Existen tres formas de producción de eventos: aquellos generados por dispositivos hardware, los generados mediante software y los producidos por la interacción humana. Veamos cada uno por separado.

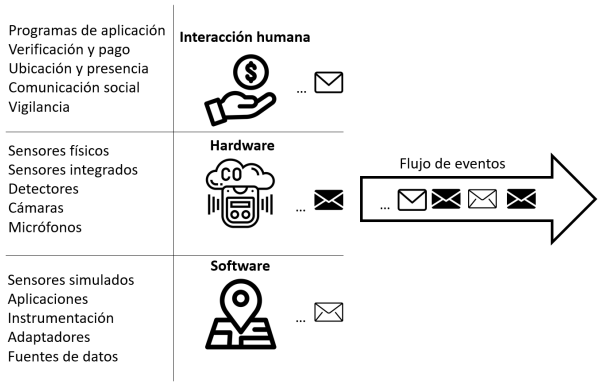


Figura 2.21: Tipos de productores de eventos encontrados en aplicaciones de procesamiento de eventos.

— *Interacción humana*: algunos eventos se generan directamente por interacción humana, aunque con un poco de asistencia de software y hardware. La interacción humana puede ser facilitada por un programa de aplicación con una interfaz de usuario que, de hecho, permite al usuario ingresar al evento. Los eventos también se pueden generar usando un dispositivo de verificación o pago, por ejemplo, la confirmación de entrega producida por el dispositivo portátil del conductor en la aplicación, o un evento de compra generado por una caja registradora en una tienda minorista.

— *Productores de eventos de hardware*: el prototipo de productor de eventos de hardware es un sensor que genera eventos que informan sobre uno o más aspectos del entorno físico en el que se encuentra, por ejemplo, un detector de humo. Un sensor se puede empaquetar como una pieza discreta de hardware, como el ejemplo del detector de humo; o puede integrarse en otro equipo, por ejemplo, un sensor que detecta la velocidad del ventilador en la placa base de una computadora.

— *Productores de eventos de software*: aunque el software a menudo se asocia con un productor de hardware, algunos productores de eventos están formados únicamente por software. Los sensores simulados son simulaciones de software de los tipos de productores de hardware. Los sensores simulados se usan cuando todo el sistema externo es, en sí mismo, una simulación, por ejemplo, un simulador de entrenamiento de vuelo o un juego de realidad virtual; también se pueden usar para reemplazar una pieza real de hardware cuando se prueba una aplicación de procesamiento de eventos.

2.5.3. Consumidor de eventos

El consumidor del evento es el complemento lógico del productor del evento. Un consumidor de eventos acepta objetos de eventos de entidades en una red de procesamiento de eventos y los procesa. El elemento de definición de consumidor de eventos se puede usar de una de tres maneras [Etzion and Niblett, 2010]:

— Para definir un tipo de consumidor de evento abstracto.

— Para representar una clase de instancias de consumidores de eventos concretos

— Para representar una instancia de consumidor de evento concreto único.

El elemento de definición de consumidor de eventos solo se refiere a la interacción entre el consumidor o consumidores que representa y el resto de la red de procesamiento de eventos; no describe la lógica interna de los consumidores de eventos. El elemento de definición consta de tres partes: detalles del consumidor, especificaciones del terminal de entrada y relaciones con otros consumidores. En la Figura 2.22 se muestra tres amplias familias de consumidores de eventos: hardware, software e interacción humana.

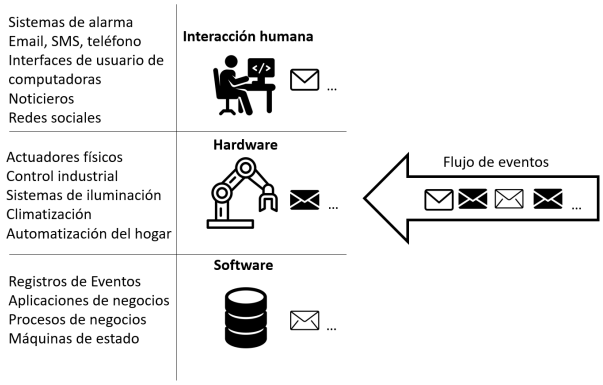


Figura 2.22: Tipos de consumidores de eventos encontrados en aplicaciones de procesamiento de eventos.

— *Interacción humana*: los actuadores descritos reaccionan a los eventos controlando directamente algo en el mundo físico. Hay otro estilo de consumidor de eventos cuyo trabajo es interactuar con los seres humanos. Esto podría ser para alertar a alguien sobre un hecho grave que requiere atención inmediata, para informar algo menos urgente que, sin embargo, podría ser de interés, o para actualizar la información mostrada por una pantalla visual de algún tipo. Los consumidores frecuentemente transmiten información sobre eventos a través de la interfaz de usuario de una computadora. Esta interfaz se implementa mediante una aplicación cliente especialmente escrita o, cada vez con más frecuencia, a través de una aplicación web que permite ver los datos del evento en un navegador web sin necesidad de instalar un software especializado.

— *Consumidores de eventos de hardware*: un hardware que consume eventos a menudo se conoce como un actuador, y es la contraparte del sensor que produce eventos. Un actuador toma un evento entrante y reacciona a él realizando una acción física, a menudo para controlar algo en el mundo físico. Esto podría implicar movimiento físico (si el actuador incluye algún tipo de motor), cambiar un campo magnético o producir una señal eléctrica o de radio. Un actuador podría estar físicamente empaquetado junto con un sensor en la misma pieza de hardware, pero cuando esto sucede, todavía se modela el sensor y el actuador como un productor y consumidor por separado. En las aplicaciones de control industrial, los actuadores se utilizan para encender y apagar los equipos, para controlar el funcionamiento de la maquinaria y para controlar el flujo de líquidos. En aplicaciones de edificios inteligentes, los actuadores se pueden usar para controlar sistemas de iluminación, calefacción y aire acondicionado. Con el surgimiento de la automatización del hogar, los actuadores se pueden utilizar para realizar tareas como abrir o cerrar puertas de garaje o persianas de ventanas, así como para controlar la iluminación y la calefacción.

— *Consumidores de eventos de software*: la tercera familia consiste en consumidores de eventos que comprenden solo software y que no ofrecen una interfaz de usuario. Los consumidores de software pueden mantener un registro de los eventos que reciben, ya sea en un archivo plano o en una base de datos. Los consumidores se denominan registros de eventos. Un registro de eventos puede ser útil en aplicaciones de diagnóstico activas. Al analizar un problema, con frecuencia es útil poder retroceder en el tiempo y observar los eventos que tuvieron lugar en el período previo a la aparición del problema en sí, pero que en ese momento parecían irrelevantes. Otro uso de un registro de eventos es proporcionar una pista de auditoría [Etzion and Niblett, 2010]. Al diseñar un registro de eventos, se debe tener en cuenta qué datos de eventos se deben registrar y qué tipos de búsquedas se realizarán con respecto a esos datos. Si el registro se lee ocasionalmente, se debe implementar para que optimice la escritura de los datos en lugar de la lectura.

2.5.4. Arquitectura Dirigida por Eventos (EDA)

La arquitectura dirigida por eventos (EDA, Event-Driven Architecture) se basa típicamente en un modelo de comunicación asíncrona basado en mensajes para propagar información en toda la organización. Apoya una alineación más natural con el modelo operativo de una organización al describir las actividades comerciales como una serie de eventos. EDA conduce a sistemas altamente desacoplados lo que conlleva a que los problemas comunes que introducen las dependencias del sistema se están eliminando mediante la adopción de la EDA. El estilo EDA se basa en los aspectos fundamentales de las notificaciones de eventos para facilitar la difusión inmediata de información y la ejecución reactiva de la lógica de negocio. En un entorno EDA, la información se puede propagar a todos los servicios y aplicaciones en tiempo real. El patrón EDA permite aplicaciones empresariales altamente reactivas ya que los componentes de un sistema solo se ejecutan al ocurrir el evento específico que están esperando. Esta arquitectura es capaz de producir sistemas altamente escalables. La arquitectura consta de compo-

nentes de procesamiento de eventos de un solo propósito que escuchan los eventos y los procesan de forma asíncrona. Hay dos topologías principales en la arquitectura dirigida por eventos [Raj et al., 2017]: la mediadora y la intermediaria.

— *Topología del mediador*: esta topología tiene una única cola de eventos donde un mediador dirige cada uno de los eventos a los procesadores de eventos relevantes. Por lo general, los eventos alimentan a los procesadores de eventos que pasan por un canal de eventos para filtrar o preprocesar eventos. La implementación de la cola de eventos podría ser en forma de una cola de mensajes simple o mediante un mediador de mensajes que aproveche un gran sistema distribuido, que intrínsecamente involucra protocolos de mensajería complejos. En la Figura 2.24 se muestra la implementación arquitectónica de la topología del mediador.

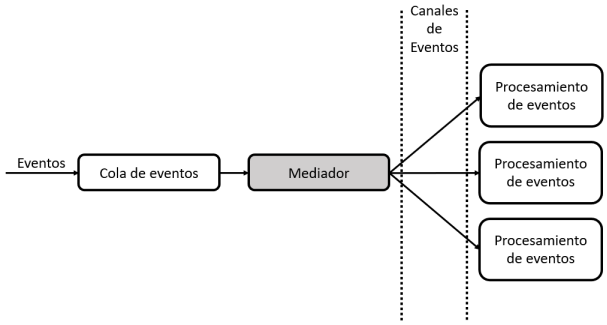


Figura 2.23: Topología del mediador (Fuente: adaptado de [Raj et al., 2017]).

— *Topología de intermediario*: esta topología no implica ninguna cola de eventos. Los procesadores de eventos son responsables de obtener los eventos, procesar y publicar otro evento que indique el final. Los procesadores de eventos actúan como intermediarios para encadenar eventos. Una vez que un procesador procesa un evento, se publica otro evento para que otro procesador pueda continuar.

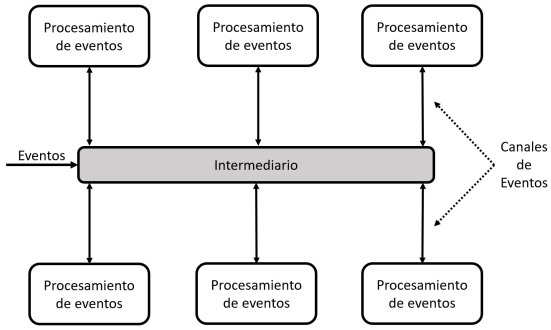


Figura 2.24: Topología de intermediario (Fuente: adaptado de [Raj et al., 2017]).

2.5.5. Procesamiento de eventos y eventos complejos (CEP)

Las operaciones comunes de procesamiento de eventos incluyen leer, crear, transformar y eliminar eventos. En este sentido existen dos actividades principales dentro del procesamiento [Etzion and Niblett, 2010]:

— El diseño, codificación y operación de aplicaciones que usan eventos, ya sea directa o indirectamente.

— Las operaciones de procesamiento que se puede realizar en eventos. Estos incluyen filtrar ciertos eventos, cambiar una instancia de evento de un formulario a otro y examinar una colección de eventos para encontrar un patrón particular. Estas operaciones pueden generar nuevas instancias de eventos.

Es posible escribir programas basados en eventos sin usar operaciones explícitas de procesamiento de eventos y las personas han estado haciendo programación basada en eventos durante muchos años. Las tres cosas que distinguen el procesamiento de eventos de la programación simple basada en eventos son las siguientes [Etzion and Niblett, 2010]:

— *Abstracción*: las operaciones que forman la lógica de procesamiento de eventos se pueden separar de la lógica de la aplicación, lo que permite modificarlas sin tener que cambiar las aplicaciones productoras y consumidoras.

— *Desacoplamiento*: los eventos detectados y producidos por una aplicación en particular pueden ser consumidos y aplicados por aplicaciones completamente diferentes. No es necesario que las aplicaciones de producción y consumo sean conscientes de la existencia de los demás, y pueden distribuirse en diferentes localidades. Muchas aplicaciones consumidoras pueden actuar sobre un evento emitido por una sola aplicación productora.

— *Enfoque en el mundo real*: el procesamiento de eventos con frecuencia trata con eventos que ocurren, o podrían ocurrir, en el mundo real.

El procesamiento de eventos complejos (CEP, Complex Event Processing) es una tecnología para agregar, procesar y analizar flujos masivos de datos para obtener información en tiempo real de los eventos a medida que ocurren [Confluent, 2020]. CEP es una generalización del procesamiento de flujo tradicional. El procesamiento de flujo tradicional se ocupa de encontrar patrones de bajo nivel en los datos, como el número de clics del ratón dentro de una ventana de quince minutos. Utilizando modelos de causalidad y jerarquías conceptuales, CEP puede hacer inferencias de alto nivel sobre eventos complejos dentro del dominio.

Una de las características distintivas de CEP es el uso de jerarquías conceptuales. Los os eventos del mundo real se producen en diferentes niveles de abstracción. En un nivel, se encuentran las intenciones y sentimientos de una persona. En un nivel inferior, se encuentra su rastro GPS o las acciones de su ratón. Cuando se trata dos o más niveles a la vez, no se puede esperar procesar una secuencia de eventos simple ordenada en el tiempo. En su lugar, se debe poder consumir “una nube de eventos”. La diferencia entre una nube de eventos y un flujo de eventos es que la nube de eventos contiene datos de

muchos flujos en diferentes niveles de abstracción. El conjunto de herramientas CEP puede identificar patrones complejos en un sistema multinivel.

Otra característica distintiva de CEP es el uso de relaciones causales. Por ejemplo, si se busca un patrón en el que se espera que una cierta combinación de movimientos del GPS y clics del ratón provoque un evento comercial, como una compra o una cancelación. Con CEP, se puede optar por marcar la combinación como un evento complejo. Sin embargo, no se debe asumir que los datos siempre llegarán en la secuencia de tiempo correcta. El beneficio clave de CEP es que las acciones pueden desencadenarse por una combinación de eventos que ocurren en diferentes momentos y en diferentes contextos.

CEP brinda la capacidad de sintetizar información significativa a partir de datos sin procesar y conocimiento del dominio. CEP tiene la capacidad de organizar la información en conceptos de alto nivel al considerar diferentes marcos de tiempo, contextos y relaciones causales dentro de los datos.

2.5.6. Procesamiento EDA y SOA

EDA y SOA no son excluyentes, ya que es perfectamente posible utilizar el procesamiento de eventos dentro de un SOA. El término SOA basado en eventos es utilizado para denotar la combinación de EDA y SOA. Si se adopta una definición limitada de SOA, que establece que cada aplicación debe construirse a partir de servicios orientados a solicitud-respuesta y solo a partir de servicios orientados a solicitud-respuesta, entonces estaría limitado a las interacciones de solicitud-respuesta y no podrá beneficiarse de todas las ventajas del procesamiento de eventos. Sin embargo, no hay nada fundamental para SOA que dicte el uso exclusivo de solicitud-respuesta general [Etzion and Niblett, 2010]. Un enfoque de programación basado en eventos se puede mezclar con componentes de solicitud-respuesta en un SOA de dos maneras:

— Es posible que un componente implemente ambos enfoques. En otras palabras, puede proporcionar o consumir una interfaz de solicitud-respuesta y también ser un productor o consumidor de eventos. Como ejemplo, se puede considerar un servicio que realiza pedidos a través de una interfaz de solicitud-respuesta y produce un evento cuando detecta niveles bajos de *stock*.

— La infraestructura SOA que aloja los componentes SOA puede proporcionar instrumentación que produce eventos en nombre de los servicios de estilo de solicitud-respuesta. Por ejemplo, puede implementar el servicio de cumplimiento de pedidos anterior para que se produzca un evento cada vez que se realiza un pedido. Estos eventos son procesados por componentes de procesamiento de eventos.

Los componentes de procesamiento de eventos pueden ser tratados como una extensión del repertorio de componentes utilizados dentro de un SOA, y una red de procesamiento de eventos como un tipo de servicio compuesto SOA. El ciclo de vida y los aspectos de gobernanza de SOA se aplican igualmente bien a estos componentes, aunque, por supuesto, sus interfaces se definen de manera ligeramente diferente: un productor de eventos se define en términos de los eventos que produce.

2.6. Internet de las cosas (IoT)

En el mundo de la tecnología Informática, dos de los hitos históricos que tienen un significado especial son la invención de ARPANET (Advanced Research Projects Agency Network), y el surgimiento de Internet de las cosas (IoT, Internet of Things). ARPANET surge en 1967 del trabajo conjunto de los equipos del Laboratorio Nacional de Física (Reino Unido) y Rand Corporation, para permitir a las computadoras intercambiar datos incluso cuando están geográficamente separadas. Sin embargo, el IoT fue un proceso en evolución en lugar de un solo evento. Las primeras implementaciones del concepto de IoT ocurrieron en 1982 cuando Mike Kazar e Ivor Durhamun, estudiantes de la Universidad Carnegie Mellon, encontraron una manera de monitorizar la cantidad de latas que quedaban en una máquina expendedora. Lo hicieron agregando un fotosensor y un microcontrolador dentro de la máquina dispensadora para contar cada vez que una lata salía de la máquina. El microprocesador se conectó al PDP-10 que entonces era la computadora principal de la Universidad en el cual se procesaba la información. El programa para controlar el estado de la máquina dispensadora incluía el tiempo que cada botella había estado en la máquina y el número de latas restantes [Gupta, 2019]. En este sentido, Kevin Aston, mencionó por primera vez el término Internet de las cosas. Aston explicó que "En el siglo XX, las computadoras eran cerebros sin sentidos, solo sabían lo que se les contaba". Las computadoras dependían de los humanos para ingresar datos y conocimientos a través de la escritura, códigos de barras, etc. En el siglo XXI, las computadoras están sintiendo cosas por sí mismas [Hames et al., 2017], lo que permitirá administrar y monitorizar objetos inteligentes utilizando conectividad en tiempo real de lo que ocurre en su entorno, permitiendo un nivel completamente nuevo de toma de decisiones basada en datos.

En el diseño de sistemas IoT se pueden reconocer cuatro elementos indispensables [AVSystem, 2020]:

(a) *Objetos Inteligentes*: es prácticamente cualquier objeto que se pueda imaginar. Comenzando con sensores, microcontroladores y actuadores especializados y adaptados a su propósito, pasando por objetos utilitarios cotidianos como las cafeteras o autos inteligentes, y terminando con aplicaciones poco usuales como basureros u ostras. Las tareas de los objetos inteligentes incluyen no solo la recopilación de datos sino, sobre todo, comunicarse entre sí y con el servidor remoto o la nube.

(b) *Conectividad*: si los objetos conectados comunican su ubicación, estado u otra información, deben disponer de un lenguaje en el que puedan hacerlo y un canal a través del cual puedan transmitir su información. Mientras que el lenguaje está representado por los protocolos específicos de Internet utilizados en una implementación determinada, el canal de comunicación está definido por los protocolos de conectividad, como Wi-Fi o Bluetooth, o tecnologías menos conocidas, como Sigfox o Narrowband IoT.

(c) *Software*: esta es la parte de IoT que generalmente se oculta a los ojos del usuario final habitual de IoT, pero aunque se encuentra un poco al margen del escenario principal, es el campo real de acción del IoT. Con el software, los datos recopilados

por los objetos inteligentes se pueden estructurar, analizar y gestionar; En base a eso, decide si se deben tomar acciones o si se debe notificar al usuario.

(d) *Aplicación*: esta es la parte en la que el usuario final interviene y toma el control. En la aplicación, los datos recopilados y analizados se visualizan y el usuario obtiene información sobre el funcionamiento de todo el sistema.

2.6.1. Arquitecturas IoT

Si bien las arquitecturas de red tradicionales para IT (Information Technology) han servido bien durante muchos años, no se adaptan bien a los complejos requisitos de IoT. La diferencia clave entre IT e IoT son los datos. Si bien los sistemas de IT se enfocan principalmente por el soporte confiable y continuo de aplicaciones comerciales como correo electrónico, web, bases de datos, etc., IoT se centra en los datos generados por los sensores y cómo se usan esos datos. La esencia de las arquitecturas de IoT implica, por lo tanto, cómo se transportan, recopilan, analizan y, en última instancia, actúan los datos [Hames et al., 2017]. La Tabla 2.5 analiza de forma más detallada las diferencias entre las redes de IT y de IoT, enfocándose en los requisitos de IoT que están impulsando las nuevas arquitecturas de red, y considera qué ajustes son necesarios.

Como se puede observa en la Tabla 2.5 las arquitecturas de referencia para ser aplicados en el IoT deben tener algunas particularidades. Es así que disponer de una arquitectura de referencia permite aplicar estrategias que se pueden aplicar al planificar una aplicación IoT. Estas arquitecturas pueden ayudar a simplificar el desarrollo, administrar la complejidad y garantizar que las soluciones de IoT sigan siendo escalables, flexibles y sólidas [IBM, 2017b]. A continuación, se comentan las arquitecturas de tres y de siete capas [Ramgir, 2019].

2.6.1.1. Modelo de referencia de tres capas

Esta arquitectura describe la estructura de las soluciones IoT, incluidos los aspectos físicos y los aspectos virtuales. Este enfoque modular puede ayudar al desarrollador a administrar la complejidad de las soluciones de IoT. En este sentido, para una aplicación IoT basada en datos que involucran análisis de borde, la arquitectura de tres niveles puede ser una buena alternativa [Sittón-Candanedo et al., 2019]:

— *Capa 1: Dispositivo*: los sensores recopilan y acumulan información relevante de un objeto o del entorno, después de lo cual se convierten en datos útiles. Por ejemplo, al mover el teléfono en el aire, se detecta el movimiento y esta información se convierte en datos útiles. Por otro lado, los actuadores pueden alterar el estado físico de un objeto. Por ejemplo, puede modificar el flujo de aire de una válvula o abrir/cerrar una fuente de alimentación eléctrica. Los datos del sensor se registran principalmente en formato analógico, por lo que debe existir una estrategia que pueda agregar y convertir estos datos en un formato digital para procesarlos. Para ello, los sistemas de adquisición de datos establecen una conexión con la red del sensor, agregan información y ejecutan la conversión de formato analógico a digital. Por ejemplo, un sistema de riego se compone de varios actuadores y sensores que

DESAFÍO	DESCRIPCIÓN	CAMBIO ARQUITECTÓNICO REQUERIDO PARA IOT
Escala	La escala masiva de los puntos finales de IoT (sensores) está mucho más allá de la de las redes de TI típicas.	El espacio de direcciones IPv4 se ha agotado y no puede cumplir con los requisitos de escalabilidad de IoT. La escala solo se puede alcanzar mediante el uso de IPv6
Seguridad	Los dispositivos IoT, especialmente aquellos en redes de sensores inalámbricos (WSN, Wireless Sensor Networks), a menudo están físicamente expuestos al mundo.	Se requiere seguridad en todos los niveles de la red IoT. Cada nodo final IoT en la red debe ser parte de la estrategia de seguridad general y debe admitir la autenticación a nivel de dispositivo y el cifrado de enlaces.
Dispositivos y redes limitados por el poder, la memoria de la CPU y la velocidad del enlace	Debido a la escala masiva y la mayor distancia, las redes a menudo son limitadas, con pérdidas y capaces de soportar solo velocidades de datos mínimas (decenas de bps a cientos de kbps).	Se necesitan nuevas tecnologías inalámbricas de última milla para soportar dispositivos IoT restringidos a largas distancias. La red también está restringida, lo que significa que es necesario realizar modificaciones en los mecanismos de transporte de capa de red tradicionales.
El volumen masivo de datos generados	Los sensores generan una cantidad masiva de datos a diario, causando cuellos de botella en la red y análisis lento en la nube.	Las capacidades de análisis de datos deben distribuirse en toda la red de IoT, desde el borde hasta la nube. En las redes de TI tradicionales, los análisis y las aplicaciones generalmente se ejecutan solo en la nube.
Soporte para dispositivos heredados	Una red de IoT a menudo consta de una colección de puntos finales con capacidad IP, así como de dispositivos heredados no IP basados en protocolos seriales o patentados.	La transformación digital es un proceso largo que puede llevar muchos años, y las redes IoT deben admitir la traducción de protocolos y/o mecanismos de túnel para admitir protocolos heredados sobre protocolos basados en estándares, como Ethernet e IP.
La necesidad de analizar los datos en tiempo real	Mientras que las redes de TI tradicionales realizan un procesamiento de datos por lotes programado, los datos de IoT deben analizarse y responderse en tiempo real.	El software de análisis debe ubicarse más cerca del borde y debe admitir análisis de transmisión en tiempo real. El software de análisis de TI tradicional (como las bases de datos relacionales o incluso Hadoop) se adapta mejor a los análisis de nivel de lote que se producen.

Tabla 2.5: Controladores arquitectónicos de IoT.

envían información a un sistema de adquisición de datos vinculado a un Gateway para enrutarlos por Internet para su procesamiento.

— *Capa 2: Borde (*Edge*)*: los datos de un ecosistema de IoT consumen los recursos del centro de datos, como el ancho de banda de la red. Por lo que es recomendable utilizar sistemas de borde, que puedan realizar un primer procesamiento de la información para reducir la carga de trabajo de la infraestructura central del sistema. Es así que se pueden reducir los problemas relacionados con el almacenamiento, la seguridad y los retrasos en el procesamiento. Por ejemplo, al utilizar el aprendizaje automático en el borde para poder detectar irregularidades en un sistema de transporte, lo que puede ayudar a los vehículos a descubrir rutas alternas y reducir la congestión.

— *Capa 3: Nube (*Cloud*)*: cuando los datos requieren un mayor grado de procesamiento y cuando no es necesario que la retroalimentación sea inmediata, estos datos se enrutan a un sistema basado en la nube o un centro de datos físico. Donde se

pueden administrar, analizar y almacenar de una forma segura. Sin embargo, es posible que el retardo para obtener una respuesta sea mayor pero el análisis es más detallado. Además, se puede combinar los datos de los sensores e integrarlos con otras fuentes de datos para realizar un procesamiento más complejo.

2.6.1.2. Modelo de referencia de siete capas

En 2014, el comité de arquitectura IoTWF (IoT World Forum) [IoTWF, 2020] publicó un modelo de referencia arquitectónica IoT de siete capas. Si bien existen varios modelos de referencia de IoT, el presentado por el Foro Mundial de IoT ofrece una perspectiva limpia y simplificada sobre IoT e incluye computación de vanguardia, almacenamiento de datos y acceso. Proporciona una forma simple de visualizar IoT desde una perspectiva técnica. Cada una de las siete capas se divide en funciones específicas, y la seguridad abarca todo el modelo. La Tabla 2.6 detalla las capas del modelo de referencia IoTWF.

CAPA DE MODELO DE REFERENCIA DE IOT	DESCRIPCIÓN
Capa 1: Dispositivos físicos y controladores	Es el hogar de las "cosas" en el Internet de las cosas, incluidos los diversos dispositivos de punto final y sensores que envían y reciben información. Su función principal es generar datos y ser capaz de ser consultado y/o controlado a través de una red.
Capa 2: Conectividad	Transmite confiable y de forma oportuna los datos. Más específicamente, esto incluye transmisiones entre dispositivos de Capa 1 y la red y entre la red y el procesamiento de información que ocurre en la Capa 3 (la capa de cómputo Edge). La capa de conectividad abarca todos los elementos de red de IoT y realmente no distingue entre la red de última milla, puerta de enlace y red de retorno.
Capa 3: Computación Edge	Reduce los datos y la conversión de flujos de datos de red en información que está lista para el almacenamiento y el procesamiento por capas superiores. Uno de los principios básicos de este modelo de referencia es que el procesamiento de la información se inicia lo antes posible y lo más cerca posible del borde de la red.
Capa 4: Acumulación de datos	Captura datos y los almacena para que las aplicaciones puedan utilizarlos cuando sea necesario. Convierte datos basados en eventos en procesamiento basado en consultas.
Capa 5: Abstracción de datos	Concilia múltiples formatos de datos y asegura una semántica consistente de varias fuentes. Confirma que el conjunto de datos está completo y consolida los datos en un solo lugar o en múltiples almacenes de datos mediante la virtualización.
Capa 6: Aplicaciones	Interpreta datos utilizando aplicaciones de software. Las aplicaciones pueden monitorizar, controlar y proporcionar informes basados en el análisis de los datos.
Capa 7: Colaboración y procesos	Consume y comparte la información de la aplicación. Colaborar y comunicar información de IoT a menudo requiere múltiples pasos, y es lo que hace que IoT sea útil. Esta capa puede cambiar el proceso de negocio y ofrece los beneficios de IoT.

Tabla 2.6: Capas del modelo de referencia IoTWF.

2.6.2. Computación Edge, Fog y Cloud

La computación en la Nube (Cloud Computing) ha sido una tecnología habilitadora para el IoT, un pequeño aumento en el porcentaje de objetos conectados o ciberfísicos representa un cambio dramático en la computación y la conectividad. Los sistemas de control distribuido a gran escala, las aplicaciones geodistribuidas, las aplicaciones móviles dependientes del tiempo y las aplicaciones que requieren una latencia o interoperabilidad muy baja son solo algunas de las aplicaciones del IoT para las que las infraestructuras de nube existentes no están bien equipadas. Las arquitecturas tradicionales de la computación en la nube no fueron diseñadas para el IoT, caracterizado por una distribución geográfica extrema, heterogeneidad y dinamismo. Por lo que, se requiere un enfoque diferente para cumplir con los requisitos IoT, como son la escalabilidad, interoperabilidad, flexibilidad, confiabilidad, eficiencia, disponibilidad y seguridad [Buyya and Srirama, 2019]. Sin embargo, las organizaciones que dependen en gran medida de los datos cada vez más han tenido que utilizar otras infraestructuras de computación como Edge y *Fog* (Figura 2.25), lo cual ha permitido aprovechar una variedad de recursos informáticos y de almacenamiento de datos [WINSYSTEMS, 2017].

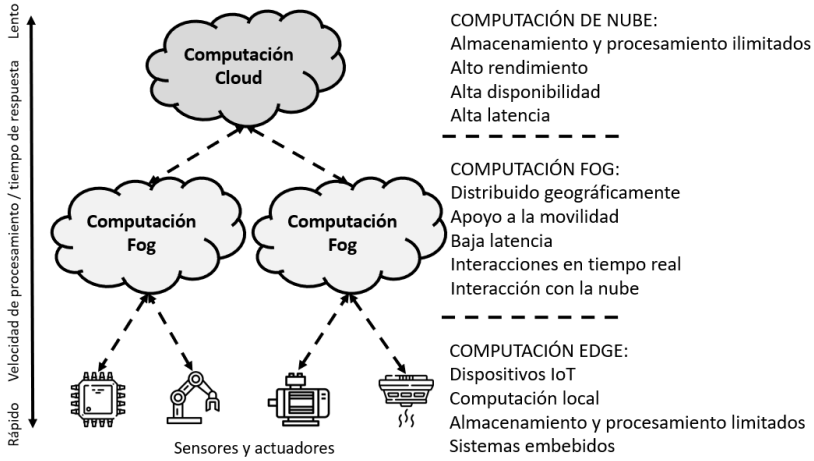


Figura 2.25: Integración de dispositivos IoT con computación Fog y Cloud.

2.6.2.1. Computación Edge

El IoT ha introducido una cantidad prácticamente infinita de puntos finales en las redes comerciales. Esta tendencia ha hecho que sea más difícil consolidar los datos y el procesamiento en un solo centro de datos, lo que ha dado lugar al uso de la informática de borde. Esta arquitectura realiza cálculos cerca del borde de la red, que está más cerca de la fuente de datos [WINSYSTEMS, 2017]. Hay dos métodos con respecto a cómo funcionarán y se comunicarán con Internet los dispositivos y sensores *Edge* [Lea, 2020]: (a) Los sensores y dispositivos de nivel *Edge* proporcionarán una ruta directa a la nu-

be. Esto implica que estos sensores y nodos de nivel *Edge* tienen suficientes recursos, hardware, software y recursos para transmitir datos a través de la red directamente; y (b) Los sensores de nivel *Edge* forman agregaciones y agrupaciones alrededor de puertas de enlace y enrutadores para proporcionar áreas de preparación, conversiones de protocolo y capacidades de procesamiento de *Edge*/*Fog*, y administrarán la seguridad y la autenticación entre los sensores y la WAN.

2.6.2.2. Computación Fog

La computación *Fog* surgió como un paradigma informático situado entre la nube y los dispositivos finales conectados o inteligentes en los que los elementos informáticos intermedios (nodos *Fog*) proporcionan servicios de gestión de datos y/o comunicaciones para facilitar la ejecución de aplicaciones IoT. El propósito de la computación *Fog* es un mayor soporte para la interoperabilidad entre proveedores de servicios, procesamiento y análisis en tiempo real, movilidad y distribución geográfica. A pesar de estas ventajas, la computación *Fog* agrega una capa de complejidad que se debe tener en cuenta, sobre todo la orquestación y gestión de recursos [Buyya and Srirama, 2019]. La computación *Fog* abarca desde sistemas de nube de centros de datos centrales hasta dispositivos *Edge* cercano y extremo. Además, representa una abstracción de un conjunto geográficamente dispar de computadoras *Cloud* y *Edge* para comportarse y actuar como una sola entidad o sistema [Lea, 2020].

La computación *Fog* y la computación *Edge* parecen similares, ya que ambas implican acercar la inteligencia y el procesamiento a la fuente de los datos. Sin embargo, la diferencia clave entre los dos radica en dónde se ubica la inteligencia y el poder de cómputo. Un entorno *Fog* coloca la inteligencia en la red de área local (LAN). La computación *Edge* coloca la inteligencia y la potencia de procesamiento en dispositivos como los controladores [WINSYSTEMS, 2017].

2.6.2.3. Computación en la Nube

La computación en la nube es la forma de computación donde los datos se almacenan en múltiples servidores y se puede acceder a ellos en línea desde cualquier dispositivo. En lugar de guardar información en el disco duro local en una sola computadora, los usuarios la almacenan en servidores en línea de terceros. Para acceder a los datos, un usuario debe ingresar una cuenta asociada con el servicio en la nube. Los datos se someten a un cifrado de extremo a extremo, por lo que incluso los proveedores de servicios no tienen acceso a los contenidos del usuario. La computación en la nube ofrece elasticidad y escalabilidad de recursos y aplicaciones, el servicio y los recursos son fácilmente accesibles y están disponibles. Por lo tanto, la convergencia de la nube y la IoT puede brindar grandes oportunidades para ambas tecnologías [Biswas and Giaffreda, 2014]. Para el IoT, esto significa almacenar y administrar de forma segura una gran cantidad de datos y tener acceso inmediato a ellos desde múltiples dispositivos, en cualquier momento y en cualquier lugar. La principal ventaja de los sistemas basados en la nube es que permiten recopilar datos de varios sitios y dispositivos, a los que se puede acceder en cualquier parte del mundo [WINSYSTEMS, 2017].

2.6.3. La Web de las Cosas (WoT)

Un enfoque prometedor que se está incorporando al IoT es la idea de que debería construirse de manera similar a Internet, lo que se ha denominado WoT (Web of Things) [W3C, 2020]. Hay varias razones para usar un enfoque basado en la web en el IoT. La web ha existido durante décadas y se ha ganado mucha experiencia. Desde su lanzamiento público en 1991, la World Wide Web ha evolucionado enormemente y se ha convertido en una infraestructura sobre la cual almacenar documentos, recursos y construir aplicaciones distribuidas [Cirani et al., 2018]. Los aspectos más importantes de la introducción de la web fueron la referencia de recursos a través de identificadores uniformes de recursos (URI, Uniform Resource Identifier) y la introducción del Protocolo de transferencia de hipertexto (HTTP, Hypertext Transfer Protocol) como el protocolo de la capa de aplicación para sistemas hipermedia. Junto con estos dos pilares principales, se han desarrollado otros estándares y tecnologías esenciales, como el Lenguaje de Marcado de Hipertexto (HTML, HyperText Markup Language) para documentos web, navegadores web y servidores web. A medida que la web se hizo más y más popular, los navegadores integraron el comportamiento dinámico a través de Javascript y CSS (Cascading Stylesheets). HTTP es el protocolo de capa de aplicación más común y las bibliotecas de software que implementan protocolos web están disponibles para cualquier lenguaje de programación. Las formas de crear aplicaciones web son ampliamente conocidas y utilizadas: la adopción de enfoques similares para el IoT, por lo tanto, podría aprovechar la experiencia de los desarrolladores existentes. Además, la web ha demostrado escalar extremadamente bien, esto es importante para el IoT, donde se espera que funcionen miles de millones de dispositivos conectados. El IoT puede beneficiarse enormemente de toda la experiencia obtenida en el desarrollo de la web y, por lo tanto, el uso de una arquitectura similar parece ser una buena elección de diseño.

En este sentido la WoT, es la siguiente etapa en esta evolución. En el que se emplean y adaptan protocolos Web para conectar cualquier objeto en el mundo físico y darle presencia en la Web. En este sentido la WoT es un refinamiento del IoT al integrar objetos inteligentes no solo en Internet, sino también en la Arquitectura Web [Guinard et al., 2010]. Un objeto es la abstracción de una entidad física o virtual (por ejemplo, un dispositivo o una habitación) y se describe mediante metadatos estandarizados. Para lo cual los metadatos de descripción deben ser una descripción del objeto.

Una entidad virtual es la composición de una o más objetos (por ejemplo, una habitación que consta de varios sensores y actuadores). Una opción para la composición es proporcionar una descripción única y consolidada que contenga el conjunto de capacidades para la entidad virtual. En los casos en los que la composición es bastante compleja, su descripción puede vincularse a sub-objetos jerárquicos dentro de la composición. La vinculación no solo se aplica a los objetos jerárquicos, sino a las relaciones entre las cosas y otros recursos en general. Los tipos de relación expresan cómo se relacionan los objetos, por ejemplo, un interruptor que controla una luz o una habitación supervisada por un sensor de movimiento. Otros recursos relacionados con un objeto pueden ser manuales, catálogos de repuestos, archivos CAD, una interfaz de usuario gráfica o cualquier otro documento en la Web. En general, la vinculación web entre los objetos hace que la WoT sea navegable, tanto para humanos como para máquinas [W3C, 2020].

2.6.3.1. Arquitectura de la WoT

Al igual que las capas de modelo OSI organizan los protocolos y estándares de Internet, la arquitectura WoT es un intento de estructurar la gran diversidad de protocolos y herramientas web para conectar cualquier dispositivo u objeto a la web. La arquitectura de la WoT no está compuesta por capas en sentido estricto, sino por niveles que agregan funcionalidad adicional. Cada capa ayuda a integrar las Cosas a la Web de manera aún más íntima y, por lo tanto, hace que esos dispositivos sean más accesibles para las aplicaciones y las personas [Guinard et al., 2010]:

— *Capa 1: Acceso*: esta capa es responsable de convertir cualquier objeto en un objeto web con el que se pueda interactuar mediante solicitudes HTTP como cualquier otro recurso en la web. Es decir que un objeto web es una API REST que permite interactuar con algo en el mundo real, como abrir una puerta o leer un sensor de temperatura. Sin embargo, muchos escenarios de IoT son en tiempo real y/o impulsados por eventos. En el cual en lugar de que la aplicación solicite continuamente datos, se establece que le notifique cuando algo sucede, por ejemplo, cuando un sistema de seguridad detecta que la humedad alcanza un cierto umbral o se detecta ruido durante la noche.

— *Capa 2: Búsqueda*: esta capa garantiza que un objeto no solo pueda ser utilizado fácilmente por otros clientes HTTP, sino que también pueda ser encontrado y utilizado automáticamente por otras aplicaciones de la WoT. El enfoque aquí es reutilizar los estándares semánticos web para describir cosas y sus servicios. Esto permite buscar cosas a través de motores de búsqueda y otros índices web, así como la generación automática de interfaces de usuario o herramientas para interactuar con los objetos.

— *Capa 3: Compartición*: esta capa es la responsable de Compartir los datos generados por los objetos de manera eficiente y segura a través de la web. Por ejemplo, el protocolo cuando un sistema biométrico emplea TLS, para asegurar las transacciones en la Web.

— *Capa 4: Composición*: una vez que las cosas estén en la Web (capa 1), donde las personas y las máquinas las pueden encontrar (capa 2) y sus recursos se pueden compartir de forma segura con otros (capa 3), es posible construir aplicaciones para la WoT. Las herramientas web en la capa de Composición van desde kits de herramientas web, por ejemplo, JavaScript o Node-RED.

2.6.3.2. Thing Description

La Thing Description (TD) describe los metadatos y las interfaces de los objetos, donde un objeto es una abstracción de una entidad física o virtual que proporciona interacciones y participa en la WoT. Además, proporciona un conjunto de interacciones basadas en un pequeño vocabulario que hace posible integrar diversos dispositivos y permite la interoperabilidad de diversas aplicaciones. Esta descripción está codificada en un formato JSON que proporciona una representación de los dispositivos de forma comprensible

para las máquinas. Una instancia TD tiene cuatro componentes principales: metadatos textuales sobre el objeto en sí, un conjunto instrucciones que indican cómo se puede usar el objeto, esquemas para los datos intercambiados con el objeto y, finalmente, enlaces web para expresar cualquier relación formal o informal con otros objetos o documentos en la Web [W3C, 2020].

En el Listado 2.1 se muestra un ejemplo de una instancia TD que ilustra el modelo de interacción con propiedades, acciones y eventos al describir una bombilla denominada MiBombilla.

```
{
    "@context": "https://...",
    "id": "...",
    "title": "MiBombilla",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"}
    },
    "security": ["basic_sc"],
    "properties": {
        "status" : {
            "type": "string",
            "forms": [{"href": "https://.../status"}]
        }
    },
    "actions": {
        "toggle" : {
            "forms": [{"href": "https://.../toggle"}]
        }
    },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{
                "href": "https://.../oh",
     "subprotocol": "longpoll"
            }]
        }
    }
}
```

Listado 2.1: Ejemplo de Thing Description para una bombilla.

2.6.4. La web como plataforma

Los costos, el tamaño limitado y el consumo mínimo de energía son algunas de las razones por las que los dispositivos IoT tienen capacidades computacionales limitadas. Debido a estos requisitos funcionales y económicos, los objetos inteligentes, especialmente aquellos que funcionan con baterías, no pueden permitirse tener grandes cargas de procesamiento y utilizar protocolos de comunicación costosos. Sin embargo, las capacidades de procesamiento limitadas significan que es difícil procesar mensajes grandes. Por otro lado, menos procesamiento significa menor consumo de energía. Como resultado, los dispositivos IoT generalmente necesitan minimizar la cantidad de datos transmitidos [Cirani et al., 2018]. En este sentido, la web nació para ser un sistema fácil de usar, distribuido y poco acoplado para compartir documentos, es lo suficientemente simple como para facilitar la creación de aplicaciones y la administración de contenido. La web se basa en un pequeño conjunto de principios, sin embargo, ha demostrado escalar y evolucionar rápidamente. Debido a estos principios, la web ha evolucionado para convertirse en una plataforma para construir sistemas distribuidos utilizando HTTP.

En este sentido, modelar el IoT utilizando principios RESTful orientados a la web puede ser una forma de comenzar a desarrollar una infraestructura global de objetos interconectados y fomentar el desarrollo de aplicaciones IoT escalables y robustas. La idea

básica es considerar los objetos inteligentes como pequeños servidores que implementan aplicaciones IoT utilizando hipermedia. Construir el IoT en torno al paradigma REST y modelarlo de acuerdo con los conceptos web permite reutilizar toda la experiencia adquirida en las décadas de construcción de la web. De esta manera la WoT proporciona una capa de aplicación que simplifica la creación de IoT. Al llevar los patrones de la web al IoT, será posible crear aplicaciones robustas a largo plazo y construir una infraestructura diseñada para escalar indefinidamente en el tiempo.

2.6.5. Objetos Inteligentes

Para proporcionar una definición general de las plataformas de hardware, la Figura 2.26 muestra una vista de alto nivel de los principales componentes de hardware en un objeto inteligente. Los módulos ilustrados son [Cirani et al., 2018]:

— *Módulo de comunicación*: esto le da al objeto inteligente sus capacidades de comunicación. Por lo general, es un transceptor de radio con una antena o una conexión por cable.

— *Microcontrolador*: Esto le da al objeto inteligente su comportamiento. Es un pequeño microprocesador que ejecuta el software del objeto inteligente.

— *Sensores o actuadores*: estos le dan al objeto inteligente una forma de sentir e interactuar con el mundo físico.

— *Fuente de alimentación*: esto es necesario porque el objeto inteligente contiene circuitos eléctricos. La fuente de energía más común es una batería, pero también hay otros ejemplos, como las fuentes de energía piezoeléctricas, que proporcionan energía cuando se aplica una fuerza física, o pequeñas células solares que proporcionan energía cuando la luz brilla sobre ellas.

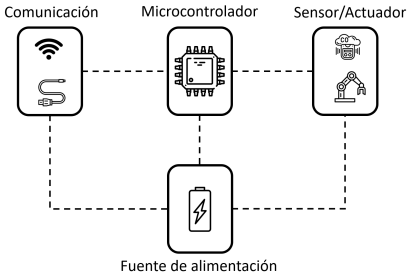


Figura 2.26: Hardware de objetos.

Un objeto inteligente es impulsado por la electrónica, y la electrónica necesita energía. Por lo tanto, cada objeto inteligente necesita una fuente de energía. Hoy en día, la fuente de energía más común es una batería, pero hay varias otras posibilidades de energía, como las células solares, piezoelectricidad, energía transmitida por radio y otras

formas de captación de energía. Las baterías de celda de litio son actualmente las más comunes. Con un hardware de baja potencia y un software de administración de energía adecuado, un objeto inteligente puede tener una vida útil de años en baterías de celda de litio estándar. A diferencia de los teléfonos celulares y las computadoras portátiles, que son operados por humanos, la mayoría de los objetos inteligentes están diseñados para funcionar sin control ni supervisión humanos. Además, muchos objetos inteligentes están ubicados en lugares difíciles de alcanzar y muchos están incrustados en otros objetos. Por lo tanto, en la mayoría de los casos no es práctico recargar sus baterías. A continuación se revisan las características que tiene estos objetos y su clasificación.

2.6.5.1. Clases de dispositivos restringidos

Internet es cada vez más usado en dispositivos pequeños con severas limitaciones de energía, memoria y recursos de procesamiento, creando redes de nodos restringidos. En este sentido el RFC7228 [RFC-Editor, 2014] proporciona una estandarización para la definición de las principales clases y características de los objetos inteligentes IoT y dispositivos restringidos, existiendo tres clases de dispositivos [Bormann, 2014]:

(a) *Dispositivos de Clase 0*: son los dispositivos severamente restringidos en la memoria y en las capacidades de procesamiento donde lo más probable es que no tengan los recursos requeridos para comunicarse directamente con Internet de manera segura. Estos dispositivos participarán en las comunicaciones de Internet con la ayuda de dispositivos más grandes que actúen como servidores *proxy*, puertas de enlace o servidores. Los dispositivos de clase 0 generalmente no se pueden asegurar o administrar de manera integral en el sentido tradicional. Lo más probable es que estén preconfigurados con un conjunto de datos muy pequeño. Para fines de gestión, podrían responder señales de encendido/apagado.

(b) *Dispositivos de Clase 1*: los dispositivos están bastante limitados en las capacidades de procesamiento, de modo que no pueden comunicarse fácilmente con otros nodos de Internet que emplean una pila de protocolos completa como HTTP, Transport Layer Security y protocolos de seguridad relacionados y representaciones de datos basadas en XML. Pueden usar una pila de protocolos específicamente diseñada para nodos restringidos y participar en conversaciones significativas sin la ayuda de un nodo de puerta de enlace. Pueden proporcionar soporte para las funciones de seguridad requeridas en una red grande. Por lo tanto, se pueden integrar como dispositivos completamente desarrollados en una red IP.

(c) *Dispositivos de Clase 2*: los dispositivos están menos restringidos y son fundamentalmente capaces de admitir la mayoría de las mismas pilas de protocolos que se usan en computadoras portátiles o servidores. Pueden beneficiarse de protocolos livianos y eficientes energéticamente y de consumir menos ancho de banda. El uso de menos recursos para la creación de redes deja más recursos disponibles para las aplicaciones.

2.6.5.2. Sensores

Un sensor mide cierta cantidad física y convierte esa lectura de medición en una representación digital. Esa representación digital generalmente se pasa a otro dispositivo para su transformación en datos útiles que pueden ser consumidos por dispositivos inteligentes o humanos. Los sensores no se limitan a datos sensoriales similares a los humanos. De hecho, pueden proporcionar un espectro extremadamente amplio de datos de medición ricos y diversos con una precisión mucho mayor que los sentidos humanos. Esta dimensión adicional de datos hace que el mundo físico sea una fuente de información increíblemente valiosa.

Los sensores pueden integrarse fácilmente en cualquier objeto físico que se conecte fácilmente a Internet mediante redes cableadas o inalámbricas. Debido a que estos objetos físicos *host* conectados con capacidades de detección multidimensional se comunican entre sí y con sistemas externos, pueden interpretar su entorno y tomar decisiones inteligentes. La conexión de dispositivos de detección de esta manera ha introducido el mundo de IoT y un paradigma completamente nuevo de inteligencia. Hay diferentes categorías para agrupar sensores, incluidas las siguientes [Hames et al., 2017]:

— *Activo o pasivo*: los sensores se pueden clasificar en función de si producen una salida de energía y, por lo general, requieren una fuente de alimentación externa (activa) o si simplemente reciben energía y normalmente no requieren una fuente de alimentación externa (pasiva).

— *Invasivo o no invasivo*: los sensores se pueden clasificar en función de si un sensor es parte del entorno que está midiendo (invasivo) o externo (no invasivo).

— *Contacto o sin contacto*: los sensores se pueden clasificar en función de si requieren contacto físico con lo que están midiendo (contacto) o no (sin contacto).

— *Absoluto o relativo*: los sensores se pueden clasificar en función de si miden en una escala absoluta (absoluta) o en función de una diferencia con un valor de referencia fijo o variable (relativo).

— *Área de aplicación*: los sensores se pueden clasificar en función de la industria específica o vertical donde se utilizan.

— *Cómo miden los sensores*: los sensores se pueden clasificar en función del mecanismo físico utilizado para medir la información sensorial (por ejemplo, termoeléctrica, electroquímica, piezoresistiva, óptica, eléctrica, mecánica de fluidos, fotoelástica).

— *Qué miden los sensores*: los sensores se pueden clasificar según sus aplicaciones o qué variables físicas miden.

El esquema de clasificación más útil para la aplicación pragmática de sensores en una red IoT, es simplemente clasificar según el fenómeno físico que mide un sensor. Este tipo de categorización se muestra en la Tabla 2.7.

DISPOSITIVO	TIPO	EJEMPLOS
Sensor	Posición	Potenciómetro, inclinómetro, sensor de proximidad.
	Presencia y movimiento	Ojo eléctrico, radar.
	Velocidad y aceleración	Acelerómetro, giroscopio.
	Fuerza	Medidor de fuerza, viscosímetro, sensor táctil, sensor de contacto.
	Presión	Barómetro, medidor Bourdon, piezómetro.
	Caudal	Anemómetro, sensor de flujo másico, medidor de agua.
	Acústico	Micrófono, geófono, hidrófono.
	Humedad	Higrómetro, humistor, sensor de humedad del suelo.
	Luz	Sensor infrarrojo, fotodetector, detector de llama.
	Radiación	Contador Geiger-Muller, centelleador, detector de neutrones.
	Temperatura	Termómetro, calorímetro, medidor de temperatura.
	Químico	Alcoholímetro, olfatómetro, detector de humo.
	Biosensores	Biosensor de glucosa en sangre, oximetría de pulso, electrocardiógrafo.
Actuador	Mecánico	Palanca, gato de tornillo, manivela.
	Eléctrico	Tiristor, transistor bipolar, diodo.
	Electromecánico	Motor AC, motor DC, motor paso a paso.
	Electromagnético	Electroimán, solenoide lineal.
	Hidráulico y neumático	Cilindro hidráulico, cilindro neumático, pistón, válvulas de control de presión, motores neumáticos.
	Material inteligente	Aleación con memoria de forma, fluido de intercambio iónico, material magnetorestrictivo, tira bimetálica, bimorfo piezoeléctrico.
	Micro y nano actuadores	Motor electrostático, microválvula, accionamiento de peine.

Tabla 2.7: Tipos de sensores y actuadores.

2.6.5.3. Actuadores

Los actuadores son complementos naturales de los sensores. Los actuadores, reciben algún tipo de señal de control (comúnmente una señal eléctrica o comando digital) que desencadena un efecto físico, generalmente algún tipo de movimiento, fuerza, etc. Algunas formas comunes en que pueden clasificarse incluyen las siguientes [Hames et al., 2017]:

— *Tipo de movimiento*: los actuadores se pueden clasificar según el tipo de movimiento que producen (por ejemplo, lineal, giratorio, uno/dos/tres ejes).

— *Potencia*: los actuadores se pueden clasificar según su potencia de salida (por ejemplo, alta potencia, baja potencia, micro potencia).

— *Binario o continuo*: los actuadores se pueden clasificar en función del número de salidas de estado estable.

— *Área de aplicación*: los actuadores se pueden clasificar en función de la industria específica o vertical donde se utilizan.

— *Tipo de energía*: los actuadores se pueden clasificar según su tipo de energía.

La clasificación más utilizada se basa en el tipo de energía. La Tabla 2.7 muestra los actuadores clasificados por tipo de energía y algunos ejemplos para cada tipo.

2.6.5.4. Principales tipos de sensores y actuadores en IoT

El IoT se compone de varias capas de tecnología que permiten que los objetos comunes compartan los datos que recopilan a través de Internet para brindar inteligencia y acciones autónomas que dependen de la calidad de los datos. Por lo tanto, los sensores y actuadores son una parte indispensable del IoT. Como habilitadores indispensables de IoT, los sensores y actuadores ayudan a monitorizar, controlar y optimizar las operaciones en casi todos los tipos de sectores, desde vehículos inteligentes hasta el turismo. En la Figura 2.27 se ilustran los símbolos de algunos de los sensores y actuadores empleados en aplicaciones IoT.

Tipos de sensores de IoT. Los sensores pueden ser dispositivos independientes o dispositivos integrados en objetos o máquinas ordinarios para hacerlos inteligentes, y se pueden dividir en categorías en términos del fenómeno físico que pretenden medir. La siguiente lista es una descripción general de algunos de los tipos de sensores más aplicados en IoT (AVSystem, 2020):

(a) *Sensores de temperatura*: Este tipo de sensor, el más básico, sirve para realizar un seguimiento de las condiciones térmicas del aire, el entorno de trabajo, las máquinas u otros objetos. Los sensores de temperatura son particularmente útiles en plantas de fabricación, almacenes, sistemas de informes meteorológicos y agricultura, donde la temperatura del suelo se controla para proporcionar un crecimiento equilibrado y maximizado.

- Termistor: un tipo de resistencia, cuya resistencia (resistencia) depende significativamente de la temperatura. Los termistores se utilizan ampliamente como sensores en electrónica, por ejemplo, como sensores de temperatura en termómetros electrónicos o en sistemas que evitan un aumento excesivo de la corriente.
- Detectores de temperatura por resistencia: son instrumentos para medir la temperatura basados en un cambio en la resistencia que está asociado con los cambios de temperatura.

(b) *Sensores de humedad*: su uso más obvio y generalizado es en estaciones meteorológicas para informar y pronosticar el clima. Sin embargo también se emplean en agricultura, monitoreo ambiental, cadena de suministro de alimentos, y monitoreo de la salud.

- Sensor de humedad de tensión de cabello: es un sensor de humedad tradicional y el más antiguo. El diseño del dispositivo se basa en las propiedades específicas del cabello humano, aunque actualmente también se utilizan fibras sintéticas o de algodón. Estas fibras cambian de longitud al entrar en contacto con la humedad.
- Psicrómetro: la construcción de este tipo de dispositivo se basa en dos termómetros (el denominado "seco" y el "húmedo"). El termómetro húmedo tiene un recipiente de mercurio revestido con material que absorbe la humedad. El

termómetro seco muestra la temperatura ambiente y la humedad se puede calcular a partir de la diferencia de temperatura.

(c) *Sensores de Luz*: estos sensores dependiendo de la intensidad de la luz ambiental, los televisores inteligentes, los teléfonos móviles o las pantallas de las computadoras pueden ajustar su brillo gracias. También se emplean en aplicaciones de ciudades inteligentes. Se utilizan para adaptar el alumbrado público o los niveles de iluminación urbana para una mayor economía.

- Fotorresistor: es un elemento fotosensible, cuya resistencia cambia a través de la radiación. Se puede conectar fácilmente, por ejemplo, a un Arduino como sensor de luz analógico.
- Fotodiodo: un diodo que funciona basándose en el efecto fotoeléctrico. Cuando los fotones alcanzan la unión de un fotodiodo, son absorbidos, lo que hace que el electrón se transfiera a la banda de conductividad para crear un par electrón-hueco. Los fotodiodos se utilizan ampliamente en la automatización industrial (sistemas de señalización y control), telecomunicaciones (optoacopladores, enlaces optoelectrónicos) y muchas más industrias.

(d) *Sensores se ruido y vibración*: los sensores acústicos permiten controlar el nivel de ruido en un entorno determinado. Al poder medir y proporcionar datos para ayudar a prevenir la contaminación acústica, los cuales son empleados en las de ciudades inteligentes.

- Hidrófono: es un micrófono que se utiliza para captar los sonidos que se propagan en el agua u otros líquidos. Los hidrófonos son un elemento estructural básico de los sonares pasivos que se utilizan, por ejemplo, para detectar peces en diversos entornos acuáticos.
- Geófono: es un sensor que convierte las vibraciones del suelo (frecuencia y amplitud) en voltaje eléctrico. Se puede considerar un tipo de sismómetro.

(e) *Sensores de nivel de agua*: para prevenir desastres naturales, los datos recopilados por los sensores de monitoreo del nivel del agua se pueden usar en sistemas de alerta de inundaciones para análisis y predicción. Además de la protección del medio ambiente, este sensor se emplea en aplicaciones industriales para controlar y optimizar los procesos de fabricación.

- Sensor de presión hidrostática: se utiliza para medir el nivel de llenado de líquido. Estos sensores funcionan basándose en la presión hidrostática medida en el punto de medición del tanque independientemente de la forma y el volumen del tanque.
- Sensor óptico: este sensor detecta el nivel de agua que implica la refracción de la luz en un prisma después del contacto con el líquido. Los sensores ópticos tienen una cierta ventaja sobre los sensores de nivel de agua típicos debido al hecho de que están limitados por partes mecánicas y móviles que pueden romperse o simplemente desgastarse.

(f) *Sensores de presencia y proximidad*: al emitir un haz de radiación electromagnética, este tipo de sensor es capaz de detectar la presencia de su objetivo y determinar la distancia de separación. Por su alta confiabilidad y larga vida, se emplean en automóviles inteligentes, robótica, fabricación, máquinas, aviación e incluso soluciones de estacionamiento inteligente.

- Radar Doppler: es un radar que utiliza el efecto Doppler, es decir, la diferencia en la frecuencia de la onda enviada por su fuente y la frecuencia de la onda registrada por el observador que se mueve en relación con esta fuente. El radar no solo se utiliza para detectar objetos y determinar su ubicación, sino también para conocer su dirección y velocidad.
- Sensor de presencia: es un tipo de sensor que utiliza luz infrarroja o altas frecuencias para detectar movimiento en oficinas, instalaciones sanitarias, pasillos, pasillos, almacenes, etc.

(g) *Sensores de movimiento*: estos sensores se emplean para monitorizar espacios públicos o privados contra intrusos y robos, el uso de sensores de movimiento se está extendiendo a soluciones de administración de energía, cámaras inteligentes, dispositivos automatizados y muchos otros.

- Sensor de movimiento ultrasónico activo: este sensor envía y recibe ondas pasivas ultrasónicas.
- Sensor de movimiento infrarrojo pasivo: este sensor detecta cambios en la radiación infrarroja.
- Sensor de radar activo: este sensor emite y recibe ondas electromagnéticas.
- Sensor de infrarrojos pasivo (PIR): es un sensor electrónico utilizado para la detección de movimiento, que se utiliza comúnmente en sistemas de alarma, iluminación automática y sistemas de ventilación, etc.

(h) *Sensor giroscópico*: la función de este tipo de sensor es detectar la rotación y velocidad angular, lo que es útil en sistemas de navegación, robótica, electrónica de consumo y procesos de fabricación que implican rotación.

- Acelerómetro: este sensor no mantiene una dirección constante, pero indica la velocidad angular del objeto sobre el que se ubica. En este grupo se incluyen los giroscopios mecánicos, que tienen una libertad de rotación limitada (generalmente en uno de los ejes del sistema de coordenadas cartesianas), los giroscopios ópticos (láser y fibra óptica) y finalmente los giroscopios que utilizan el efecto Coriolis.
- Indicador de rumbo: el giroscopio direccional o indicador de rumbo permite observar la rotación del cuerpo al que está unido. El giroscopio se fabrica con mayor frecuencia como un objeto rígido de rotación rápida (generalmente un disco) suspendido en una estructura que permite su rotación libre con respecto al sistema de referencia.

(i) *Sensores químicos*: los sensores capaces de detectar compuestos químicos (sólidos, líquidos y gases) son elementos indispensables en los sistemas de seguridad industrial, protección ambiental, investigación científica, monitorización de la calidad del aire que ayuda a las ciudades y los estados a combatir el impacto dañino de la contaminación del aire y el agua.

— Alcoholímetro electroquímico: sensor sencillo que se utiliza para determinar el contenido de alcohol en sangre.

— Nariz electrónica: conjunto de detectores, que reacciona a diferentes tipos de partículas contenidas en el ambiente o sus diferentes características (como la presencia de enlaces químicos específicos, acidez, alcalinidad, la capacidad de estabilizar dipolos vecinos, etc.). Estos sensores se emplean en sistemas de seguridad de aeropuertos para el control de explosivos y narcóticos.

(j) *Sensores de imagen*: al convertir los datos ópticos en impulsos eléctricos, un sensor de imagen permite ver el entorno que lo rodea. Los sensores de imagen se utilizan siempre que sea necesario que el dispositivo inteligente 'vea' su entorno inmediato, que incluye vehículos inteligentes, sistemas de seguridad, equipos militares como radares y sonares, dispositivos de imágenes médicas así como cámaras digitales.

— Sensor de píxeles activos: en estos sensores se disponen muchos elementos fotosensibles realizados en tecnología CMOS, y se encuentran en dispositivos, como cámaras web, cámaras digitales compactas, cámaras DSLR (Digital Single Lens Reflex), cámaras digitales de rayos X entre otros.

— Dispositivo de carga acoplada: es un sistema de muchos elementos fotosensibles que registran y luego permiten leer una señal eléctrica proporcional a la cantidad de luz que incide sobre él.

Tipos de actuadores de IoT. Los actuadores interactúan con su entorno inmediato para permitir el funcionamiento de máquinas o dispositivos en los que están integrados. Estos dispositivos se pueden encontrar en vehículos, máquinas industriales o cualquier otro equipo electrónico que involucre tecnologías de automatización. Se pueden separar en cuatro categorías principales según su patrón de construcción y la función que desempeñan en un entorno de IoT específico [AVSystem, 2020]:

(a) *Actuadores lineales*: se utilizan para permitir el movimiento de objetos o elementos en línea recta.

(b) *Motores*: permiten movimientos de rotación precisos de componentes de dispositivos u objetos completos.

(c) *Relés*: esta categoría incluye actuadores basados en electroimán para operar interruptores de energía en lámparas, calentadores o incluso vehículos inteligentes.

(d) *Solenoides*: son utilizados en electrodomésticos como parte de mecanismos de bloqueo o activación, también actúan como controladores en sistemas de monitoreo de fugas de agua y gas.

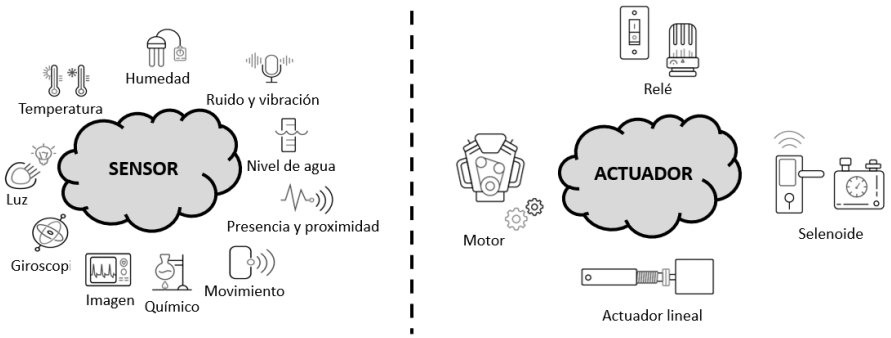


Figura 2.27: Tipos de sensores y actuadores utilizados en IoT.

2.6.5.5. Microcontrolador

Un microcontrolador es un circuito integrado compacto diseñado para gobernar una operación específica en un sistema integrado. Un microcontrolador típico incluye un procesador, memoria y periféricos de entrada/salida en un solo chip, también denominado unidad de microcontrolador (MCU, Micro-Controller Unit). En esencia, son computadoras personales (PC) en miniatura simples diseñadas para controlar pequeñas funciones de un componente más grande, sin un Sistema Operativo (SO) de interfaz de usuario complejo. Los elementos centrales de un microcontrolador son [Chin, 2020]:

— El procesador (CPU): Procesa y responde a varias instrucciones que dirigen la función del microcontrolador. Esto implica realizar operaciones básicas de aritmética, lógica y E/S. También realiza operaciones de transferencia de datos, que comunican comandos a otros componentes en el sistema integrado más grande.

– Memoria: la memoria de un microcontrolador se utiliza para almacenar los datos que el procesador recibe y utiliza para responder a las instrucciones que ha sido programado para llevar a cabo. Un microcontrolador tiene dos tipos de memoria principales: *(a)* Memoria de programa, que almacena información a largo plazo sobre las instrucciones que ejecuta la CPU. La memoria del programa es una memoria no volátil, lo que significa que retiene información a lo largo del tiempo sin necesidad de una fuente de alimentación; y *(b)* Memoria de datos, que es necesaria para el almacenamiento temporal de datos mientras se ejecutan las instrucciones. La memoria de datos es volátil, lo que significa que los datos que contiene son temporales y solo se mantienen si el dispositivo está conectado a una fuente de alimentación.

— Periféricos de E/S: los dispositivos de entrada y salida son la interfaz del procesador con el mundo exterior. Los puertos de entrada reciben información y la envían al procesador en forma de datos binarios. El procesador recibe esos datos y envía las instrucciones necesarias a los dispositivos de salida que ejecutan tareas externas al microcontrolador.

Uno de los microntroladores más populares para el desarrollo de aplicaciones IoT es el NodeMCU que usa el módulo ESP-12E basado en el chip ESP8266. Es un dispositivo independiente que no necesita otro microcontrolador para funcionar. Tiene pines expuestos para operaciones generales de entrada y salida. Tiene un puerto USB que se puede usar para alimentar NodeMCU y depurar el programa que se ejecuta en el dispositivo. También tiene un pin *Vin* donde puede ingresar voltaje desde una fuente de alimentación, como una batería, para alimentar el dispositivo [Chin, 2020]. La Figura 4.2 muestra el NodeMCU.

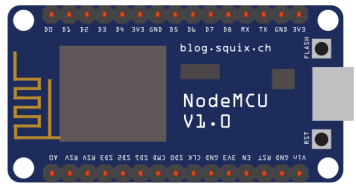


Figura 2.28: Esquemático de controlador NodeMCU.

2.6.6. Conectividad para IoT

Para establecer la conectividad de los objetos en los sistemas IoT se debe considerar la distancia que los separa. Para ello, el desarrollador debe seleccionar un protocolo de conectividad para corto, mediano y largo alcance (AVSystem, 2020). En la Figura 2.29) se muestran algunos de los protocolos de conectividad y rango de alcance.

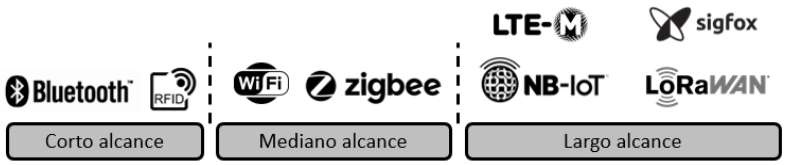


Figura 2.29: Tecnologías para conectividad IoT.

(a) Protocolos de conectividad IoT de corto alcance:

— *Bluetooth*: es una tecnología de conectividad de corto alcance empleada especialmente para el mercado de la electrónica portátil, como los auriculares inalámbricos o los sensores de geolocalización. Esta tecnología tiene un consumo reducido de energía y una baja tasa de transferencia.

— *RFID (Radio Frequency Identification)*: es una de las primeras tecnologías para IoT implementadas, y permite aplicaciones de gestión y logística de cadenas de suministros, que requieren la capacidad de determinar la posición de los objetos dentro de edificios.

(b) Protocolos de conectividad IoT de mediano alcance:

— *WiFi*: desarrollado en base al protocolo IEEE 802.11, es uno de los protocolos de comunicaciones inalámbricas más extendido y conocido. Una de sus principales desventajas es el consumo de energía superior al promedio que resulta de la necesidad de mantener una alta intensidad de señal y una rápida transferencia de datos para una mejor conectividad y confiabilidad.

— *ZigBee*: es un estándar de redes de malla inalámbrica aplicado en sistemas de gestión del tráfico, electrónica del hogar e industria. Se basa en el estándar IEEE 802.15.4 y admite tasas de intercambio de datos bajas, operación de bajo consumo, seguridad y confiabilidad.

(c) Protocolos de conectividad IoT de largo alcance:

— *Narrowband IoT (ND-IoT)*: es un producto de las tecnologías 3GPP (The 3rd Generation Partnership Project) existentes, ND-IoT es un estándar de tecnología de radio que garantiza un consumo de energía extremadamente bajo (10 años de funcionamiento con batería) y proporciona conectividad con una intensidad de señal de aproximadamente 23 dB más bajo que en el caso de la telefonía móvil 2G. Además, utiliza la infraestructura de red LTE existente, lo que permite una cobertura global y una buena calidad de señal. En muchos casos, este hecho permite implementar NB-IoT en lugar de soluciones que requerían la construcción de redes locales, como LoRa o Sigfox.

— *LTE-CAT M1*: es un estándar de conectividad de área amplia de baja potencia que conecta dispositivos IoT y M2M (Machine to Machine) con requisitos de velocidad de datos media. Permite un ciclo de vida más prolongado de la batería y ofrece un rango interno de cobertura en comparación con tecnologías móviles como 2G, 3G o LTE-Cat 1. Al ser compatible con la red LTE, CAT M1 no requiere que los operadores construyan una nueva infraestructura para su implementación. En comparación con NB-IoT, LTE Cat M1 es mejor para casos de uso móvil, ya que su manejo de transferencia entre celdas celulares es significativamente mejor y es muy similar al LTE de alta velocidad.

— *LoraWan*: es un protocolo de red de área amplia de largo alcance y bajo consumo optimizado para un bajo consumo de energía y que admite grandes redes de dispositivos. LoRaWAN está diseñado para proporcionar redes de área amplia de bajo consumo bidireccionales seguras, móviles y de bajo costo para IoT, M2M, ciudades inteligentes y aplicaciones industriales.

— *SigFox*: este protocolo permite conectividad para aplicaciones M2M de baja potencia que requieren bajos niveles de transferencia de datos para las cuales el rango de WiFi es demasiado corto y el rango de celular es demasiado caro y consume mucha energía. Sigfox emplea una tecnología que permite manejar bajas velocidades de transferencia de datos de 10 a 1,000 bits por segundo. Consumiendo hasta 100 veces menos energía en comparación con las soluciones de comunicación celular. Permite un ciclo de funcionamiento de 20 años con baterías de 2,5 Amperios.

2.6.7. Protocolos para IoT

Los protocolos de IoT son una parte crucial; sin ellos, el hardware se volvería inútil ya que los protocolos de IoT permiten intercambiar datos de una manera estructurada. Lo más visible en una aplicación IoT es la interacción entre sensores, dispositivos, puertas de enlace, servidores y aplicaciones de usuario. Pero lo que permite que todas estos objetos inteligentes hablen e interactúen son los protocolos de IoT. Los siguientes son algunos de los protocolos empleados en el despliegue de aplicaciones [AVSystem, 2020]:

(a) *Protocolo de aplicación restringida (CoAP)*: este protocolo fue creado por el grupo de trabajo IETF Constrained RESTful Environments y lanzado en 2013. CoAP (Constrained Application Protocol) fue diseñado para traducir el protocolo HTTP de modo que pudiera usarse en dispositivos restrictivos. Se basa en el Protocolo UDP (User Datagram Protocol) para establecer una comunicación segura entre los puntos finales. Al permitir la transmisión y la multidifusión, UDP puede transmitir datos a múltiples hosts mientras conserva la velocidad de comunicación y el uso de ancho de banda bajo. Además, CoAP comparte con HTTP la arquitectura RESTful que admite un modelo de interacción de solicitud/respuesta entre los puntos finales de la aplicación y permite los métodos básicos HTTP.

(b) *Transporte de telemetría de Message Queue Server (MQTT)*: MQTT (Message Queue Server Telemetry Transport) es un protocolo de mensajería ligera de tipo publicación/suscripción. Diseñado para dispositivos que funcionan con baterías, la arquitectura de MQTT es simple y liviana, lo que proporciona un bajo consumo de energía para los dispositivos. Trabaja sobre el protocolo TCP/IP para ser empleado en redes de comunicación poco fiables. MQTT puede resultar problemático para algunos dispositivos muy restrictivos, debido al hecho de la transmisión de mensajes sobre TCP y la gestión de nombres largos de temas. Esto se solventa con la variante MQTT-SN que usa UDP y admite la indexación de nombres de temas.

(c) *Protocolo de cola de mensajes avanzado (AMQP)*: este es un protocolo de tipo publicación/suscripción de estándar abierto que se originó en 2003. Su uso aún es limitado en la industria del IoT. La especificación AMQP (Advanced Message Queuing Protocol) describe características tales como orientación de mensajes, cola, enrutamiento, confiabilidad y seguridad. Probablemente el mayor beneficio de AMQP es su robusto modelo de comunicaciones.

2.6.8. Software para IoT

En las aplicaciones IoT el hardware y software trabajan juntos. Es por esto que existen variadas herramientas software que permiten configurar el comportamiento que tendrán los objetos. Algunas de ellas son los siguientes [Ramgir, 2019]:

(a) *OpenWSN*: es un proyecto de código abierto de una pila de protocolos basada en estándares para redes de IoT. Se basó en el nuevo estándar IEEE802.15.4e junto con los estándares de IoT como 6LoWPAN, RPL y CoAP, permite redes de malla de muy baja potencia y alta confiabilidad que son compatibles con Internet.

OpenWSN se puede implementar en microcontroladores de 16 bits hasta arquitecturas Cortex-M de 32 bits de última generación.

(b) *TinyOS*: es un sistema operativo gratuito, de código abierto y con licencia BSD (Berkeley Software Distribution) diseñado para dispositivos inalámbricos de bajo consumo que se utilizan en redes de sensores. Fue desarrollado por la Universidad de California, Berkeley, Intel Research y Crossbow Technology. Está escrito en el lenguaje de programación nesC (Network Embedded Systems C), que es una versión de C optimizada para admitir componentes y simultaneidad.

(c) *FreeRTOS*: es un kernel de sistema operativo en tiempo real para dispositivos integrados diseñado para ser pequeño y simple. Ha sido compilado para 35 microcontroladores y se distribuye bajo la GPL con una excepción opcional. La excepción permite que el código propietario de los usuarios permanezca de código cerrado mientras se mantiene el núcleo como fuente abierta, lo que permite el uso de FreeRTOS en aplicaciones propietarias. El código está escrito principalmente en C (pero se han incluido algunas funciones de ensamblador para admitir las rutinas específicas de la arquitectura). Proporciona métodos para múltiples hilos o tareas, mutex, semáforos y temporizadores de software.

(d) *RIOT*: es un sistema operativo de microkernel de código abierto para IoT, con licencia LGPL (Lesser General Public License). Permite programación de aplicaciones C y C ++, y proporciona capacidades completas de subprocesos múltiples y en tiempo real. RIOT se ejecuta en hardware de 8 bits (por ejemplo, AVR Atmega), 16 bits (por ejemplo, TI MSP430) y de 32 bits (por ejemplo, ARM Cortex). Un puerto nativo también permite que RIOT se ejecute como un proceso de Linux o MacOS, lo que permite el uso de herramientas de depuración y desarrollo estándar como GNU Compiler Collection, GNU Debugger, Valgrind, Wireshark, etc. RIOT es parcialmente compatible con POSIX y proporciona múltiples pilas de red, incluidos IPv6, 6LoWPAN y protocolos estándar como UDP, TCP y CoAP.

(e) *Contiki*: es un sistema operativo para sistemas en red con limitaciones de memoria, que apunta a dispositivos IoT inalámbricos de baja potencia. Por ejemplo, en términos de memoria, a pesar de proporcionar multitarea y una pila TCP/IP incorporada, Contiki solo necesita alrededor de 10 kB de RAM y 30 kB de ROM. Un sistema Contiki típico tiene una memoria del orden de kilobytes, un requerimiento de energía del orden de milivatios, una velocidad de procesamiento en megahercios y un ancho de banda del orden de cientos de kilobits/segundo.

(f) *Node-Red*: es una plataforma de programación para una integración entre varias API, dispositivos inteligentes y servicios en línea. Node-RED es un motor de desarrollo basado en flujos visuales accesibles desde un navegador web. Crea funciones de JavaScript y almacena los flujos creados en el formato de archivo JSON.

(g) *Arduino*: es una plataforma de creación de prototipos de hardware y software de código abierto, para programar microcontroladores de una sola placa capaces de detectar y ejercer control sobre el mundo físico. Arduino emplea un conjunto de especificaciones de software y hardware aplicadas a la electrónica.

2.6.9. Plataformas de IoT

Internet de las cosas es una de las industrias de más rápido crecimiento. Los dispositivos conectados están en todas partes: hogares inteligentes, automóviles, dispositivos portátiles, etc. La predicción de expertos como Gartner muestra que 20 mil millones de cosas estarán conectadas a Internet para 2020. Estos objetos no son dispositivos de uso general, como teléfonos inteligentes y PC, sino objetos con funciones dedicadas, como máquinas expendedoras, motores a reacción, automóviles conectados y una miríada de otros ejemplos [Gartner, 2017]. Por esta razón el IoT cada vez tiene mayor en la economía al transformar muchas empresas en negocios digitales y facilitar nuevos modelos comerciales, mejorar la eficiencia y aumentar el compromiso de los empleados y clientes. En consecuencia, al realizar proyectos con dispositivos basados en la nube y conectados, es necesario plataformas que permitan administrar la información generada. Las plataformas IoT tienen un rol importante en el desarrollo del IoT, debido a que permiten a las aplicaciones estar disponibles en la nube brindan servicios como: Inserción de datos, Conversión de datos, Cree una pared o interfaz para mostrar datos, Gestión de reglas, Gestión de equipos, Servicio de seguridad e Integración de plataformas IoT. A continuación, se listan algunas plataformas IoT.

2.6.9.1. FIWARE

FIWARE es un proyecto patrocinado por la Comisión Europea para desarrollar tecnologías para la Internet de las cosas y la Internet del futuro, en colaboración con participantes del sector de las TIC. La iniciativa FIWARE proporciona un conjunto de interfaces de programación de aplicaciones (API) para desarrollar aplicaciones inteligentes, basadas en especificaciones abiertas y libres de regalías [Alonso et al., 2019].

La plataforma FIWARE se basa en habilitadores genéricos que ofrecen funciones reutilizables y comúnmente compartidas. Los habilitadores genéricos de FIWARE cubren aspectos que incluyen [Wiki, 2020]:

— Alojamiento en la nube.

— Gestión de datos/contexto.

— Arquitectura del ecosistema de aplicaciones / servicios y marco de prestación.

— Habilitación de servicios de Internet de las cosas (IoT).

— Seguridad.

— Interfaz para redes y dispositivos (I2ND).

FIWARE define una Arquitectura de habilitación de servicios de IoT para los escenarios que necesitan conectar dispositivos u otros puntos de datos a componentes en la plataforma. Especifica dos habilitadores genéricos y proporciona implementaciones de código abierto de ellos. El primer componente es el habilitador genérico de administración de dispositivos *backend,* que se utiliza para conectar dispositivos IoT a la plataforma. El segundo componente es el habilitador genérico de *Context Broker* (Orion), que se utiliza

para el modelado de datos IoT como entidades de contexto y ponerlos a disposición de otros servicios [Araujo et al., 2019].

2.6.9.2. oneM2M

OneM2M es el estándar global que cubre requisitos, arquitectura, protocolo, soluciones de seguridad para la interoperabilidad de tecnologías M2M e IoT. oneM2M se formó en 2012 y consta de ocho de las organizaciones de desarrollo de estándares mundiales, incluidas ETSI, TTA, TIA, TTC, etc. El estándar oneM2M proporciona un conjunto de funciones básicas de servicio común (CSF), incluida la función de registro de aplicaciones y dispositivos , búsqueda de información, seguridad, gestión de grupos (GMR), función de análisis de gestión y almacenamiento de datos (DMR), función de notificación y suscripción de información (SUB), función de gestión de dispositivos y aplicaciones, función de facturación de servicios [Choi et al., 2019].

OneM2M define una arquitectura horizontal que proporciona funciones de servicios comunes que habilitan aplicaciones en múltiples dominios, utilizando un marco común y API uniformes. El uso de estas API estandarizadas hace que sea mucho más sencillo desarrollar aplicaciones M2M/IoT comparado con otras opciones de conectividad complejas y heterogéneas al abstraer los detalles del uso de tecnologías de red subyacentes, protocolos de transporte subyacentes y serialización de datos. Todo esto lo maneja la capa de servicio oneM2M sin necesidad de que el programador se convierta en un experto en cada una de estas capas. Por lo tanto, el desarrollador de aplicaciones puede enfocarse en el proceso/lógica comercial del caso de uso que se va a implementar y no necesita preocuparse por cómo funcionan exactamente las capas subyacentes. Esto es muy parecido a escribir un archivo en un sistema de archivos sin preocuparse de cómo funcionan realmente los discos duros y sus interfaces. La capa de servicio de IoT especificada en oneM2M puede entenderse como un sistema operativo distribuido para IoT que proporciona API uniformes para aplicaciones de IoT de manera similar a como lo hace un sistema operativo móvil para el ecosistema de teléfonos inteligentes [OneM2M, 2020].

2.6.10. Pruebas para IoT

Dado que existe una fuerte cohesión entre el hardware y el software en los proyectos de IoT, el enfoque de prueba utilizado en los proyectos de Internet de las cosas a menudo se basa en las mejores prácticas utilizadas en el desarrollo clásico de software o hardware.

2.6.10.1. Pruebas de software

Las pruebas de software se utilizan principalmente para la validación de aspectos funcionales y no funcionales del software de middleware de IoT o de código más bajo, como la lógica del software implementada en los sensores y actuadores de IoT. Se describen dos tipos principales de pruebas [Bosmans et al., 2019]:

(a) *Pruebas de caja blanca*: Las pruebas tienen total transparencia para la estructura interna del software. Las pruebas unitarias a menudo se consideran un enfoque de prueba de caja blanca. En el contexto de las pruebas de IoT, este método se utiliza

para probar la funcionalidad de piezas de código de bajo nivel, como métodos y clases individuales.

(b) *Pruebas de caja negra*: Con este método de prueba, el sistema de software se considera una caja negra completa. Estas pruebas no tienen conocimiento de la estructura interna del sistema. Las pruebas de caja negra se refieren a aspectos más de alto nivel del software, generalmente a nivel del sistema. En la práctica, este tipo de pruebas son difíciles de ejecutar, porque los sistemas de IoT generalmente contienen muchos sensores, actuadores y partes de software diferentes que interactúan entre sí.

2.6.10.2. Pruebas de usabilidad

Hay tantos dispositivos de diferentes formas y los usuarios utilizan factores de forma. Además, la percepción también varía de un usuario a otro. Es por eso que verificar la usabilidad del sistema es muy importante en las pruebas de IoT [Murad et al., 2018].

Dado que existe una amplia gama de dispositivos, las pruebas de usabilidad se basan en todos los usuarios finales específicos que pueden usarlos en la práctica. Para comenzar con las pruebas de usabilidad de IoT, todos los miembros del equipo deben estar alineados con la misma visión de cómo deben realizarse los procedimientos de prueba. En su mayoría, esto puede incluir evaluar un producto en términos de tres aspectos [Bosmans et al., 2019]: ¿Cuál es el propósito del dispositivo de IoT?, ¿Quién va a usar el dispositivo de IoT? y ¿Cómo se va a usar el dispositivo de IoT?.

Cuando se pueda explicar claramente la respuesta a las preguntas antes mencionadas, entonces se podrá decidir cómo debe llevarse a cabo la estrategia de prueba y cómo seleccionar a los evaluadores. Se debe evaluar si el dispositivo IoT sirve como una solución personalizada a un problema o es más un dispositivo "divertido". Es decir, el propósito del dispositivo es crucial para trabajar con pruebas de usabilidad. Ejemplos de usuarios sobre los que se pueden realizar pruebas IoT son: un usuario promedio estándar, personas que conocen la tecnología y están entusiasmadas con probar el dispositivo, personas que planean comprar el dispositivo con fines de entretenimiento, estudiantes de escuelas, colegios y universidades que deseen utilizar el dispositivo con fines de investigación, usuarios de los departamentos gubernamentales, trabajadores de los sectores comercial, minorista e industrial, pacientes para quienes el dispositivo es parte de MIoT (Internet médico de las cosas), entre otros ejemplos de usuarios.

2.6.10.3. Pruebas funcionales

Las pruebas funcionales son un tipo de prueba en el que todas las funciones de la infraestructura de IoT funcionan de acuerdo con los requisitos. Por lo tanto, esta prueba asegura que un dispositivo IoT sea capaz de proporcionar un rendimiento de referencia y pueda funcionar sin errores. Un evaluador debe tener en cuenta los siguientes aspectos durante las pruebas funcionales [Ramgir, 2019]:

— Se necesitan muchos recursos para la replicación de un entorno de IoT. A veces, simplemente no es posible crear un entorno de este tipo, especialmente si se requiere acceso para aquellos componentes que aún no se han implementado.

— Los grupos de terceros y los proveedores poseen diferentes servicios, subcomponentes y subsistemas que están interconectados entre sí. Cuando existe una restricción de acceso para un subcomponente de un proveedor diferente, los procedimientos de prueba funcional pueden interrumpirse.

— Se requiere una gran coordinación y cooperación entre varios equipos de departamento para obtener los datos de prueba requeridos; luego se trabaja sobre toda la infraestructura.

— La disponibilidad del dispositivo también es un factor crítico; si no está disponible en el momento adecuado, es posible que el consumidor tenga que enfrentarse a situaciones no deseadas.

— La prueba funcional también se ocupa de los problemas de responsabilidad. Por ejemplo, en una infraestructura de IoT, si un dispositivo de IoT que pertenece a un vehículo comienza a liberar información incorrecta, puede forzar al conector de diagnóstico, que se encuentra en la placa del dispositivo, a publicar eventos incorrectos; lo que podría provocar accidentes.

2.6.10.4. Pruebas de escalabilidad

Las pruebas de escalabilidad son aquellas en las que se puede evaluar la capacidad de un proceso, sistema o red en una infraestructura de dispositivo IoT mientras se producen modificaciones en el volumen o tamaño de sus datos para satisfacer la creciente demanda. Se clasifica en un tipo de prueba no funcional. Con las pruebas de escalabilidad, las aplicaciones de IoT pueden gestionar el aumento estimado del tráfico de usuarios, la frecuencia de los recuentos de transacciones y el volumen de datos.

Las pruebas de escalabilidad miden las bases de datos, el sistema y los procesos de la infraestructura de IoT. En esencia, ayuda a determinar el punto por el cual las aplicaciones de IoT no se pueden escalar más y arroja luz sobre la razón detrás de esto. Dichas pruebas se realizan para los siguientes propósitos [Ramgir, 2019]:

— Evalúa cómo está la aplicación después de la carga agregada.

— Evalúa la experiencia del usuario final y la degradación del lado del cliente mientras se ocupa de la carga de trabajo.

— Evalúa la degradación y solidez en el lado del servidor.

— Evalúa el límite para los consumidores de una aplicación IoT.

2.6.10.5. Pruebas de compatibilidad

La compatibilidad se refiere a tener la funcionalidad que puede ayudar en la coexistencia. Las pruebas de compatibilidad son un tipo de prueba de IoT que puede ayudar a evaluar la capacidad de IoT y de firmware que se ejecutan con diversas aplicaciones, sistemas operativos, hardware, dispositivos de IoT y entornos de red. Dichas pruebas entran en el rango de las pruebas no funcionales. Además, se divide en lo siguiente [Ramgir, 2019]:

— *Software*: evalúa el software diseñado y asegura que funciona bien con otro software en la infraestructura de IoT.

— *Sistemas operativos*: determina si el software funciona bien o no con el sistema operativo de la infraestructura de IoT.

— *Hardware*: evalúa la compatibilidad del software con cambios en las configuraciones del hardware.

— *Red*: realiza un análisis del desempeño del sistema con una red mientras prueba múltiples parámetros como capacidad, velocidad de operación y ancho de banda. Además, también evalúa varias redes y sus aplicaciones.

— *Navegador*: evalúa la compatibilidad del sitio web del sistema IoT utilizando navegadores web como Chrome, Firefox e IE.

— *Dispositivos*: realiza una prueba de compatibilidad de software para diferentes dispositivos IoT.

— *Móvil*: evalúa la compatibilidad con las plataformas móviles como Android e iOS.

— *Versiones de software*: se utiliza para la verificación de la compatibilidad entre las distintas versiones de software y la aplicación de software de IoT. Las pruebas de versión se dividen en pruebas hacia adelante y hacia atrás. El primero se utiliza para verificar el comportamiento del software o hardware de IoT diseñado mediante el uso de las últimas versiones de software o hardware. Por otro lado, la compatibilidad con versiones anteriores se utiliza para verificar el comportamiento del software o hardware de IoT diseñado mediante el uso de versiones anteriores de software o hardware.

2.6.10.6. Pruebas de rendimiento

Las pruebas de rendimiento aseguran que las aplicaciones en la infraestructura de IoT funcionen correctamente dentro de la carga de trabajo estimada. Principalmente, la verificación del rendimiento se utiliza para evaluar tres aspectos del software de IoT.

— ¿Cuál es la velocidad de la aplicación de IoT, es decir, con qué rapidez creará una respuesta?.

— ¿Cuál es la carga de trabajo extrema que puede gestionar el software de IoT?.

— ¿Qué tan estable es la aplicación de IoT con cargas de usuarios en constante cambio?.

Las pruebas de rendimiento se ejecutan para ofrecer información significativa a las partes interesadas en términos de escalabilidad, estabilidad y velocidad de la aplicación de IoT. Además, puede ofrecer información sobre cuáles son los pasos que pueden mejorar antes de su lanzamiento público. La falta de pruebas de rendimiento puede hacer que el software de IoT funcione considerablemente más lento cuando muchos usuarios acceden

al sistema de IoT al mismo tiempo. En una nota similar, ayuda a identificar la mala usabilidad y las irregularidades entre diferentes sistemas operativos. Las pruebas de rendimiento se dividen en los siguientes tipos [Ramgir, 2019]:

— *Pruebas de estrés*: en las pruebas de estrés, una aplicación se prueba para cargas de trabajo máximas con el fin de determinar su gestión del procesamiento de datos y el alto tráfico. El objetivo de la prueba de estrés es averiguar el punto de ruptura de la aplicación.

— *Prueba de carga*: evalúa la capacidad de la aplicación para funcionar mientras trabaja con la carga de trabajo esperada. El objetivo de esta prueba es identificar el cuello de botella en el rendimiento antes del lanzamiento de la aplicación IoT.

— *Prueba de picos*: se utiliza para probar la reacción del software de IoT cuando tiene que enfrentarse repentinamente a cantidades voluminosas de datos.

— *Prueba de resistencia*: se utiliza para garantizar que el software de IoT maneje permanentemente la carga de trabajo anticipada.

— *Pruebas de volumen*: se utiliza para probar grandes cantidades de datos que se almacenan en una base de datos para determinar el comportamiento completo de un sistema. El objetivo de las pruebas de volumen es evaluar el rendimiento de una aplicación de IoT mientras se trabaja con volúmenes en constante cambio.

2.6.11. Herramientas de prueba de IoT

A la hora de construir arquitecturas basadas en IoT es necesario realizar pruebas de validación para el despliegue y correcto funcionamiento de su estructura y componentes. Existen diferentes tipos de herramientas de prueba. Algunos de ellos son los siguientes.

2.6.11.1. Gatling

Gatling es una de las herramientas de prueba de carga que tiene soporte para el protocolo HTTP, por lo que es una alternativa para probar la carga en aplicaciones de IoT que utilizan un servidor HTTP. El motor central de Gatling es independiente del protocolo; lo que lo hace fácil aplicar la implementación de cualquier protocolo de su elección. Algunos de los conceptos de Gatling son los siguientes [Ramgir, 2019]:

— *Usuarios Virtuales*: Gatling, puede abordar usuarios virtuales que almacenan sus propios datos y posiblemente utilicen una ruta distinguible para navegar. Estos usuarios virtuales también pueden implementarse como hilos mediante algunas herramientas.

— *Guión*: para la evaluación se puede representar el comportamiento de los usuarios en forma de scripts. Se utiliza un DSL para escribir los escenarios en forma de script. Por ejemplo, considere el siguiente ejemplo.

```
scenario(\Usuario 1")
    .exec(http(\Acceso a la página prinicpal").get(\https://www.darwinalulema.com"))
    .pause(3, 4)
    .exec(http(\Consulta al sensor S1").get(\http://167.86.81.93:7071/restS1/S1/all"))
    .pause(2)
```

Como puede ver, este escenario tiene un usuario con el nombre de "Usuario 1" mientras que tiene dos solicitudes de HTTP y dos pausas. Las pausas simulan un tiempo de reflexión para un usuario. Si un usuario real puede hacer clic en el sitio web, el navegador carga la página web, después de lo cual lo recorre y decide el siguiente curso de acción. La aplicación recibe solicitudes HTTP cada vez que un usuario hace clic en un enlace o botón. Aparte de los recursos de la página web, es fácil captar una solicitud HTTP. En este ejemplo, "Acceso a la página principal" y "Consulta al sensor S1" son solicitudes GET.

— *Simulación*: la descripción de la prueba de carga también se conoce como simulación. Contiene el detalle de las estimaciones sobre las poblaciones de usuarios, la ejecución de escenarios y la inyección de usuarios virtuales.

2.6.11.2. IoTIFY

IoTIFY es una herramienta para pruebas de carga de plataformas IoT que presenta funcionalidades de Inteligencia Artificial basadas en la nube [IoTIFY, 2020]. Permite el desarrollo de aplicaciones IoT a través de la simulación de dispositivos virtuales en la nube. Para crear simulaciones de dispositivos IoT, IoTIFY crea una aplicación siguiendo esta estrategia [Ramgir, 2019]:

— Crear un modelo virtual.

— Iterar y hacer prototipos.

— Ejecutar una simulación a nivel de sistema.

Para desarrollar y probar un dispositivo de IoT en este entorno, se requiere algunos conocimientos básicos de JavaScript, los conceptos básicos de conectividad de IoT e información sobre los protocolos de IoT.

2.6.11.3. LoadRunner

LoadRunner se utiliza para pruebas de rendimiento de sistemas IoT [LoadRunner, 2020]. Admite una amplia gama de tecnologías, protocolos de comunicación y herramientas de desarrollo. Algunas de las tecnologías que admite incluyen HTTP, Ajax, Silverlight, HTTP, SAP, Mobile, Oracle, SQL Server, RTE, Mail, Citrix y RIA. También ofrece integración con ALM (Application Lifecycle Management) y UFT (Unified Functional Test) para los procesos de prueba. LoadRunner realiza una simulación de VUsers (usuarios virtuales) mientras trabaja con una aplicación de IoT. Estos usuarios virtuales realizan la replicación de las solicitudes de los clientes y requieren una respuesta adecuada que puede ayudar a aprobar una transacción.

2.6.12. El Hogar inteligente

Cuando se instala una serie de dispositivos inteligentes dentro del hogar y se conectan y comunican entre sí, se obtiene un hogar inteligente, porque los dispositivos funcionan en conjunto para automatizar una variedad de tareas y operaciones domésticas. La automatización del hogar ha existido durante algunas décadas, en términos de automatización de iluminación, calefacción y refrigeración, y similares. Al automatizar las operaciones básicas se consigue: una respuesta más rápida a los cambios en el entorno o las brechas en la seguridad, un funcionamiento más eficiente que conduce a ahorros de energía, y similares. Actualmente ha habido un mayor interés en la automatización del hogar, ya que cada vez hay más dispositivos inteligentes disponibles a costos más asequibles. Con le fin de determinar el grado de desarrollo de un hogar inteligente, se han definido seis niveles. Siendo el nivel seis el mayor posible y deseable. [Miller, 2015]:

— *Nivel 1: Comunicaciones básicas.* El primer paso para crear un hogar inteligente es permitir que los miembros del hogar se comuniquen con otras personas fuera del hogar. La conectividad a Internet permite la comunicación de datos además de la comunicación de voz, por lo que es particularmente importante en términos de las tareas futuras.

— *Nivel 2: Comandos simples.* Se refiere la capacidad de emitir algún tipo de comando para realizar tareas básicas, como bloquear o desbloquear una puerta, encender o apagar las luces, verificar el correo, entre otros. En este nivel, el hogar también responderá a los comandos del exterior. Como cuando alguien pasa junto a un sensor de movimiento exterior, suena una alarma.

— *Nivel 3: Automatización de funciones básicas.* En este nivel, el control manual da paso al control automático, mediante instrucciones programadas. Estamos hablando de automatizar funciones tales como controlar la temperatura de la habitación, encender o apagar las luces en momentos específicos, activar o desactivar el sistema de alarma en un horario determinado, ejecutar el sistema de rociadores exteriores de acuerdo con un programa establecido, y similares. Se trata de utilizar tecnología, normalmente en forma de temporizadores programables, para hacer lo que de otro modo tendría que hacer manualmente. En lugar de encender manualmente esta luz o presionar el botón de inicio, la tarea comienza tanto si una persona está allí como si no.

— *Nivel 4: Seguimiento y acción.* Gracias al desarrollo de la tecnología de sensores, el hogar ahora puede rastrear lo que hace (y dónde) para determinar patrones de actividad, patrones de sueño e incluso el estado de salud. En esta etapa el hogar se convierte en un sistema de monitoreo gigante. Por ejemplo, si el hogar sabe que siempre se levanta a las 6:00 am durante la semana, sabe que debe subir la calefacción, encender la cafetera, etc. Si exhibe un comportamiento contrario a su norma, su hogar sabe alertar a las autoridades que puede estar enfermo o lesionado o lo que sea.

— *Nivel 5: Actividades de incitación y respuesta a preguntas.* En este nivel, el hogar inteligente sabe más sobre el usuario que el mismo. Por ejemplo, le indicará que

tome la medicación diaria, monte en bicicleta estática, alimente al perro y prepare comida para la cena. Dada la inteligencia inherente (y el acceso no solo a la información de su hogar, sino también al amplio mundo de información disponible en Internet), podrá hacer preguntas básicas y obtener respuestas precisas.

— *Nivel 6: Automatización de tareas.* Cuanto más sepa el hogar sobre el usuario y sus actividades, más podrá hacer. El hogar programará automáticamente el mantenimiento y las reparaciones necesarias para todos sus componentes y piezas, volverá a pedir los medicamentos antes de que se acaben, preparará listas de la compra, ejecutará la aspiradora robótica y hará casi todo lo posible. cualquier otra cosa que pueda hacer de forma autónoma.

En el Hogar Inteligente hay una serie de elementos que son necesarios, la mayoría de los cuales se listan a continuación [Miller, 2015] [Kumar et al., 2019]:

— *Sensores*: Como ocurre en Internet de las cosas, es necesario sensores para determinar el estado del entorno y de varios dispositivos. Estos sensores pueden ser independientes o integrados en otros dispositivos. En un hogar inteligente, se necesitan sensores para detectar temperatura, humedad, luz, ruido y movimiento. Los sensores especializados detectan los niveles de humo y monóxido de carbono; Los sensores de proximidad detectan si las puertas y ventanas están abiertas o cerradas. Los sensores también pueden detectar el estado de un dispositivo determinado (encendido, apagado, etc.) y la ubicación de dispositivos y personas.

— *Controladores*: Los controladores son necesarios para enviar señales a otros dispositivos para iniciar algún tipo de operación. Un controlador puede dedicarse a una operación específica o ser parte de un dispositivo más grande (como un teléfono inteligente o una computadora doméstica) para controlar múltiples dispositivos.

— *Actuadores*: Un actuador es típicamente un dispositivo mecánico o eléctrico que activa una actividad determinada. Estamos hablando de motores e interruptores, como los de los interruptores de luz electrónicos, válvulas motorizadas y similares. Sin actuadores, no se hace nada.

— *Buses*: Un bus es el sistema de comunicación que transfiere datos entre dispositivos en un hogar inteligente. Los diferentes tipos de buses transfieren datos de acuerdo con diferentes protocolos; para que los dispositivos dentro del hogar se comuniquen entre sí, todos deben ser compatibles con el mismo tipo de bus.

— *Interfaces*: Una interfaz permite la comunicación entre diferentes dispositivos o entre humanos y dispositivos. En la comunicación de dispositivo a dispositivo, la interfaz es en realidad solo un protocolo digital. En el caso de la comunicación de persona a dispositivo, la interfaz generalmente incluye algún tipo de controlador y pantalla, de modo que la persona que controla pueda ver lo que está haciendo.

— *Redes*: Toda la comunicación dentro del hogar tiene lugar a través de algún tipo de red, ya sea por cable o inalámbrica. La mayor parte de la automatización del hogar actual se realiza de forma inalámbrica, utilizando Wi-Fi, Bluetooth o tecnologías de red similares.

2.7. Otras tecnologías relacionadas

En esta sección se describen algunas tecnologías adicionales que se han empleado para la implementación de la metodología, como son la televisión digital y el estándar ISDB-T.

La Televisión Digital: La televisión digital es un grupo de tecnologías que transmiten imágenes y sonido a través de señales digitales. A diferencia de la televisión analógica, la televisión digital codifica la señal en forma binaria. De esta forma, debido a la existencia de diferentes formatos de compresión de señales existentes, es posible transmitir múltiples señales en un mismo canal y crear aplicaciones interactivas a través del canal de retorno entre consumidores y productores de contenido. Actualmente, es posible acceder a la Televisión Digital mediante las siguientes tecnologías de acceso [SETID, 2020]: Ondas Terrestres (TDT), Cable, Satélite, ADSL y Dispositivos Móviles.

Hay cuatro estándares TDT internacionales: norteamericano (ATSC, Advanced Television Systems Committee), europeo (DVB-T, Digital Video Broadcasting – Terrestrial), japonés (ISDB-T, Integrated Services Digital Broadcasting) y chino (DMB-T, Digital Multimedia Broadcast-Terrestrial). Estos estándares no son compatibles entre sí y las transmisiones de un sistema no pueden ser recibidas por televisores en otro diferente. Esta situación ha obligado a los países a elegir la norma que más se adapta a sus objetivos, y también ha provocado que los promotores de normas compitan para atraer al mayor número posible de países [Angulo et al., 2011]. Debido al potencial que puede tener la televisión digital en América Latina donde está terminando su implementación, unos de los estándares más prometedores es el ISDB-T.

El estándar ISDB-T: El Sistema Brasileño de Televisión Digital Terrestre (SBTVD-T) fue definido el 26 de noviembre de 2003 y adoptó el estándar ISDB-T (Servicios Integrados de Radiodifusión Digital Terrestre), que permite la transmisión de datos digitales en alta definición (HDTV, High Definition Television) o en definición estándar (SDTV, Standard Definition Television). Así, el modelo adoptado en Brasil se conoce como ISDB-TB. Una de las principales diferencias entre el sistema ISDB-T y el adoptado en Brasil es el formato de compresión de imágenes, que reemplazó MPEG-2 por MPEG-4/H.264 [Gomes and Junqueira, 2012].

El middleware especificado en ISDB-TB, se denomina Ginga [Ginga, 2020] y realiza la intermediación entre el sistema operativo y las aplicaciones desarrolladas para TV Digital. Ginga permite crear aplicaciones interactivas de televisión digital con dos funciones principales: una es hacer que las aplicaciones sean independientes del sistema operativo de la plataforma de hardware utilizada. El otro es ofrecer un mejor soporte para el desarrollo de aplicaciones. Estas aplicaciones permiten, por ejemplo, el acceso a Internet, operaciones IoT, envío de mensajes al canal de la televisión que se está viendo, entre otras operaciones.

El sistema Ginga se divide en dos subsistemas principales de interconexión: el núcleo común (Ginga-CC) y el entorno de ejecución de la aplicación. El entorno de ejecución de la aplicación consiste en el entorno de ejecución de la aplicación NCL (Ginga-NCL) integrado con el entorno de ejecución de la aplicación Java (Ginga-J). Para terminales fijos, ambos entornos son necesarios en el sistema brasileño de televisión digital terrestre,

mientras que en terminales portátiles, el entorno Ginga-J es opcional. De forma resumida, las tres tecnologías Ginga son:

— *Ginga-CC*: Ginga-CC (Ginga Common Core) proporciona soporte básico para entornos declarativos (Ginga-NCL) e imperativos (Ginga-J). Su función principal es controlar la visualización de los objetos multimedia (como JPEG, MPEG-4, MP3, GIF, etc.)

— *Ginga-NCL*: proporciona una infraestructura de presentación para aplicaciones declarativas escritas en NCL (Nested Context Language) [NCL, 2020], que es una aplicación XML con propiedades declarativas para interactividad. Para tareas que requieren programación de algoritmos, NCL usa Lua [PUC-Rio, 2020] como su lenguaje de programación.

— *Ginga-J*: proporciona una infraestructura de ejecución de aplicaciones basada en el lenguaje Java y tiene funciones específicas para el entorno de TV digital. Estas funciones son proporcionadas por la especificación Global Executable MHP (GEM), que luego fueron reemplazadas por la especificación open Java DTV.

2.8. Resumen y conclusiones

En este capítulo se han presentado las tecnologías principales sobre las que se sustenta este trabajo de investigación: la Ingeniería Dirigida por Modelos, las Arquitecturas Orientadas a Servicios y Eventos, los Servicios REST, los patrones de Integración y el Internet de las Cosas. La combinación de estas tecnologías posibilita el desarrollo de la metodología para la integración de tecnologías heterogéneas, que se presenta en los siguientes capítulos de esta memoria.

La primera sección ha estado dedicada a presentar la Ingeniería Dirigida por Modelos. En esta sección se han revisado las bases y los conceptos de esta tecnología, con el fin de exponer las ventajas que ofrece en el desarrollo de aplicaciones al automatizar los procesos de generación de código. A continuación, se ha introducido el concepto Arquitectura Orientada a Servicios (SOA) con el fin de emplearla como una aproximación a la estandarización para la interconexión de tecnologías heterogéneas en el dominio del IoT. Además, se ha revisado la Arquitectura Dirigida por Eventos (EDA) con el propósito de ofrecer un mecanismo de comunicación y gestión de los nodos de hardware (sensores y actuadores). Para la coordinación de todos los componentes se revisó los patrones basados en orquestación y Coreografía, con los cuales se puede coordinar cada uno de los servicios y componentes de la aplicación. Se han visto las tecnologías de hardware para la construcción de los nodos IoT, y las características físicas de los sensores, actuadores y controladores, empleados en el diseño de los componentes de hardware del sistema IoT. Por último, se ha revisado brevemente los estándares de DTV, que en esta tesis doctoral fue empleada como una interfaz adicional para el IoT.

A lo largo de este capítulo se ha revisado varias y diversa tecnologías que van desde la ingeniería de software aplicada al desarrollo de aplicaciones, a la electrónica de los sensores y actuadores. Pasando por los servicios web y las técnicas de pruebas para aplicaciones. Además se ha mencionado la TDT ya es una interfaz que puede ser empleada

en el despliegue de las aplicaciones IoT. En este sentido se ha logrado tener una visión completa del IoT lo que avizora el potencial que va a seguir teniendo esta tecnología y la multitud de caminos que se puede seguir para llegar. Sin embargo también se ha visto que si no se adopta por ley o *de facto* un marco para el desarrollo este puede agregar más dificultad en la adopción del IoT ya como ocurrió con las Tecnologías Operativas, cada fabricante seguirá generando productos que serán difíciles de integrar y poner en producción en entornos tan grandes como las Smart Cities o en la Industria 4.0.

Capítulo 3

Arquitectura de Integración IoT

Capítulo 3

Arquitectura de Integración IoT

Contenidos

El IoT permite unir el mundo de los objetos con el mundo de las personas. Sin embargo, estos objetos deben colaborar con diferentes plataformas por lo que se requieren sistemas que puedan adaptarse y modificarse a lo largo del ciclo de vida de una aplicación. Esta integración presenta varias dificultades debido al surgimiento de nuevas tecnologías y tendencias, la modernización de los servicios y los recursos, o la adecuación a las preferencias de los usuarios. En este sentido cobra importancia un mecanismo que permita automatizar los procesos de desarrollo para los sistemas IoT, considerando los siguientes aspectos: (a) integración con otras plataformas; (b) diseño del hardware para el control de los sensores y actuadores; (c) interfaz de interacción con los usuarios y (d) mecanismos de coordinación. Es así que en este capítulo se presenta una formalización de los componentes y sus relaciones en los sistemas IoT, además de una metodología propuesta para automatizar el desarrollo e integración de sistemas IoT.

El presente capítulo se estructura en cinco secciones. En la Sección 3.1 se realiza una introducción de la arquitectura propuesta, poniendo de manifiesto la motivación de la integración de sistemas heterogéneos para IoT. La Sección 3.2 se ofrece una definición de la arquitectura para la integración. Para ello, se considera el escenario de Hogar Inteligente que ha sido escogido para la aplicación y validación de la metodología. Además, en esta sección se presentan las capas en las que se estructura la arquitectura y se describen las operaciones que se realizan en cada una de estas. La Sección 3.3 detalla la metodología que se ha seguido para la construcción de las herramientas propuestas para el diseño de aplicaciones IoT. Para finalizar, en la Sección 3.4 se discuten algunos de los trabajos relacionados con la arquitectura de integración propuesta. El capítulo concluye con un breve resumen y con algunas de las conclusiones extraídas.

3.1. Introducción

Los sistemas IoT suelen estar compuestos por cientos, si no miles, de nodos, interfaces de usuario, servicios de terceros, servicios de integración o una combinación de todos ellos, los cuales operan en diferentes niveles, plataformas y sistemas operativos. Sin embargo, surgen una serie de desafíos debido a que existe mayor complejidad en la integración de estos componentes, así como dificultades en la gestión de transacciones distribuidas. En este sentido, la integración permite que aplicaciones dispares trabajen coordinadas para producir un conjunto unificado de funcionalidades, y que se puedan desarrollar de forma personalizada o por proveedores externos. Además, es probable que se ejecuten en varios nodos y plataformas que pueden estar dispersos geográficamente. Es así que existen diferentes estrategias de integración como los menciona Hohpe en su obra [Hohpe, 2003]:

— *Transferencia de archivos:* cada aplicación produce archivos de datos compartidos para que las aplicaciones consuman archivos que otras han producido.

— *Base de datos compartida:* las aplicaciones almacenan los datos que desean compartir en una base de datos común.

— *Invocación de procedimiento remoto:* cada aplicación expone algunos de sus procedimientos para que se puedan invocar de forma remota y que las aplicaciones los invoquen para iniciar el comportamiento e intercambiar datos.

— *Mensajería:* cada aplicación se conecta a un sistema de mensajería común para intercambiar datos e invocar comportamientos mediante mensajes.

Además, existen dos patrones para la integración de servicios [Adamko, 2014]:

— Mediación (intracomunicación): existe un módulo que actúa como intermediario entre las aplicaciones. Siempre que ocurre un evento relevante (por ejemplo, un sensor de temperatura detecta un incremento en la temperatura) el módulo propaga los cambios a otras aplicaciones.

— Federación (intercomunicación): existe un módulo que actúa como fachada general para las aplicaciones. La aplicación está configurada para exponer solo la información y las interfaces relevantes de sus aplicaciones al mundo exterior.

Cada estrategia de integración se basa en la anterior, para buscar un enfoque más sofisticado y abordar las deficiencias de sus predecesoras. Por tanto, el orden de los estrategias refleja un orden creciente de sofisticación, pero también de complejidad. Pero por lo general ambos patrones se utilizan a menudo al mismo tiempo [Crowd-Machine, 2020]. El módulo integrador podría mantener sincronizadas varias aplicaciones (mediación), mientras atiende las solicitudes de usuarios externos contra estas aplicaciones (federación). Es así que las aplicaciones pueden integrarse utilizando múltiples estrategias y patrones para que cada punto de integración aproveche lo que más le convenga. Como resultado, muchos enfoques de integración pueden verse mejor como un híbrido de múltiples estilos y patrones de integración.

De acuerdo con los beneficios que tiene cada estrategia y debido a la heterogeneidad de los componentes de los sistemas IoT se optó por una estrategia híbrida para la arquitectura de integración propuesta y un patrón de mediación. Esta emplea: (a) Mensajería para el ámbito de los nodos de hardware y (b) Invocación de procedimiento remoto para la coordinación de los servicios web y las interfaces de usuario. Es así que esta sección se centra en la descripción de la estrategia de integración de los componentes de hardware y software de los sistemas IoT, por medio de una descripción de cómo publicar y consumir la información de los componentes del sistema.

3.2. Escenarios de integración IoT

En los últimos años, se está produciendo la cuarta revolución industrial, y una de las principales tecnologías habilitadoras de esta revolución es el IoT, que consiste en un conjunto de *Things* interconectadas (seres humanos, etiquetas, sensores, actuadores, etc.) a través de Internet, que tienen la capacidad de medir, comunicarse y actuar en todo el mundo. En este escenario, la idea clave del IoT es obtener información sobre el entorno para comprenderlo, controlarlo y actuar sobre él [Díaz et al., 2016]. Sin embargo, la falta de un estándar común y reglas claras de interoperabilidad para el IoT puede resultar

en mayores costos de mantenimiento y desarrollo, pudiendo provocar que los problemas de escalabilidad del IoT se agraven si se mantiene la tasa de crecimiento actual en un futuro próximo [Cai et al., 2018].

En el IoT existen diferentes formas de acceder a las *Things* y sus datos, lo que permite a los desarrollares crear aplicaciones globales y multiplataforma [Broring et al., 2017]. No obstante, uno de los desafíos que presenta este gran crecimiento es permitir que los usuarios consuman y combinen las funcionalidades que ofrecen los recursos de software y hardware en cualquier lugar y en cualquier momento [Urbieta et al., 2017]. Para conseguir la integración e inmediatez necesaria en los escenarios IoT, los objetos deben ser vistos por otras aplicaciones de Internet o por el proceso de negocio como entidades autónomas con comportamiento proactivo, conocimiento del entorno, y la capacidad de comunicarse y colaborar entre usuarios, objetos y sistemas [Teixeira et al., 2017].

El IoT es uno de los campos de investigación más interesantes y los desarrollos que presentan más actividad son el transporte, hogar inteligente, cirugía robótica, aviación, defensa, infraestructura crítica, etc. [Wang et al., 2015]. En el caso de los hogares, cada vez hay más objetos inteligentes y su número seguirá creciendo, por lo que, una adecuada orquestación entre todos estos objetos podría ahorrar dinero a los usuarios finales. Al automatizar el hogar se permitirá no solo conocer la temperatura promedio del hogar o el consumo de energía y agua, sino también la calidad del aire que se respira al poder automatizar la ventilación del hogar, la predicción de agua o gas, fugas o cualquier falla estructural [Díaz et al., 2016]. Pese a todo, a menudo estos sistemas no son interoperables ni están interconectados [Miori and Russo, 2014]. Por ejemplo, en el caso de que el usuario necesite interconectar dos o más objetos de diferentes fabricantes, el usuario debe desarrollar diferentes aplicaciones, con código diferente, diferente información en los objetos y tal vez con lenguaje de programación diferente [González-García et al., 2017].

Supongamos el siguiente escenario de Hogar Inteligente, escenario que usaremos a lo largo de este documento: "*Un usuario llega a su casa y desea una temperatura agradable tanto en verano como en invierno. Sin embargo, la temperatura ambiente puede cambiar cuando aumenta el número de personas o cuando se enciende el horno.*" En este contexto, el escenario propuesto (Figura 3.1) está compuesto por múltiples objetos inteligentes (sensores y actuadores) conectados localmente y con la opción de ser controlados desde Internet, en los cuales, a través de una interfaz móvil, el propietario puede controlar el estado de su vivienda o incluso activar algunos de los actuadores desde su lugar de trabajo. Además, mientras se llevan a cabo otras tareas o se encuentra fuera del hogar, la interfaz DTV (TV Digital) o la interfaz móvil también están disponibles para que pueda controlar todos los objetos inteligentes de su hogar. Este sistema también mantiene comunicación con los servicios remotos de Salud y puede actuar ante los protocolos recomendados. Por ejemplo, cuando hay una concentración de personas en un ambiente cerrado y puede activar un ventilador o abrir las ventanas para mantener un ambiente fresco y saludable.

El problema principal en el escenario anterior es la interoperabilidad y la coordinación del sistema de los componentes de hardware y software, ya que cada componente de la aplicación está diseñada en una plataforma específica. Por lo tanto, los desarrolladores deben conocer los requisitos específicos de cada una de las plataformas de hardware y de software. Este hecho crea un alto grado de dependencia entre la lógica

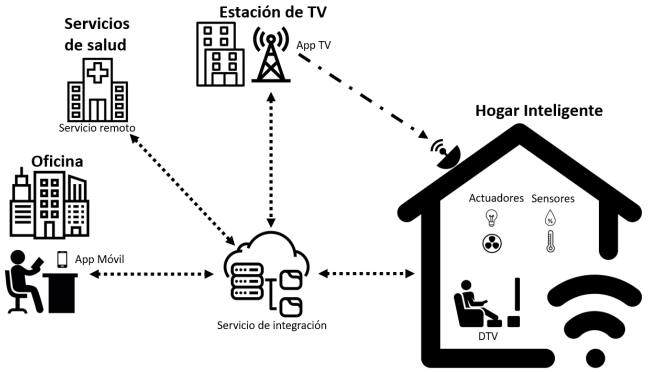


Figura 3.1: Escenario de ejemplo para integración.

de control y la plataforma, lo que resulta en poca flexibilidad y gran dificultad en los procesos de mantenimiento [Teixeira et al., 2017]. Un aspecto adicional en el escenario es la interacción del usuario con los sistemas IoT. De hecho, la visión actual de IoT se centra en la infraestructura, gestión y análisis de la gran cantidad de datos generados [González-García et al., 2017].

3.3. Arquitectura de integración

El IoT enfrenta la dificultad de integrar las aplicaciones en un ecosistema flexible para que los dispositivos puedan reutilizarse [Guinard et al., 2010]. Estas consideraciones han promovido el paso del concepto de IoT al concepto de Web de las Cosas (WoT, Web of Things), el cual es un enfoque reciente en el dominio del IoT que se centra en resolver el problema de la interoperabilidad del IoT mediante el uso de la Web como capa de integración. La Web, como uno de los pilares de Internet, ofrece la posibilidad de llegar a un consenso para alcanzar la interoperabilidad del IoT [García and Suárez, 2019]. La iniciativa de la W3C para la especificación de la WoT, tiene como finalidad abordar la fragmentación del IoT mediante el uso de estándares web existentes [Iglesias-Urkia et al., 2020]. En este sentido, los datos producidos por los dispositivos inteligentes deberían ser accesibles directamente como recursos web normales [Mainetti et al., 2015]. De esta manera, los servicios exponen los recursos del mundo real para que se puedan integrar fácilmente en el mundo virtual. Sin embargo, una arquitectura de referencia que integre los servicios de IoT con los servicios tradicionales sigue siendo un desafío de investigación abierto [Dar et al., 2015].

Se debe tener presente que antes de emprender la tarea del diseño de una Arquitectura de Integración se debe considerar que existen varias dificultades como las que plantea Teixeira en [Teixeira et al., 2017]:

— Interoperabilidad entre sistemas.

— Costos de desarrollo.

Una estrategia para abordar las dificultades anteriores es promover la separación de cada componente del sistema en un conjunto limitado de características específicas en cada paso del diseño, lo cual permite la especialización en áreas específicas. En este caso, se consideró dos ámbitos de separación: a) los detalles de programación de bajo nivel, configuraciones del hardware y protocolos de comunicación; y b) las reglas de alto nivel y la lógica de control, junto con las interfaces humano máquina.

Adicionalmente, una posible arquitectura de integración debe considerar los requisitos que se menciona en [Mainetti et al., 2015]:

— La arquitectura debe ser capaz de abstraer los dispositivos físicos, virtualizándolos y exponiéndolos a través de una interfaz común.

— La arquitectura debe permitir el desarrollo de aplicaciones IoT, para usuarios con diferentes habilidades y plataformas heterogéneas.

En este sentido el concepto de la WoT permite exponer a los dispositivos físicos a través de una interfaz web, con lo cual es más fácil su integración con otros sistemas, ya que se maneja protocolos estandarizados.

Por último, se debe considerar los desafíos comunes a los que se enfrentan los Sistemas Ciberfísicos como el IoT, y debido a su escala y variedad en sus dispositivos se deben tener en cuenta los siguientes [Kathiravelu et al., 2018]:

— Impredecibilidad de los entornos de ejecución.

— Orquestación de la comunicación y coordinación dentro del sistema.

— Seguridad, tolerancia distribuida a fallos y recuperación ante fallos del sistema y de la red.

— Toma de decisiones en entornos de ejecución geodistribuidos a gran escala.

— Modelado y diseño de entornos complejos.

Para solventar estos desafíos, se considera una estrategia de integración híbrida que permite unir el ámbito de la impredecibilidad de los eventos de los nodos de IoT con la lógica de negocio propia de los servicios web. Para lo cual se propone un mecanismo de coordinación de los componentes del sistema basado en el patrón Saga, junto con la definición de los servicios RESTful asociados a los sensores y actuadores. Además, para la integración de los eventos de los sensores y actuadores con los servicios web se establece un componente denominado “Bridge” que enruta los mensajes hacia los servicios RESTful y el Orquestador. La arquitectura propuesta consta de tres capas; véase Figura 3.2 que se describen a continuación:

— *Capa física*: corresponde a los sensores, actuadores y controladores; además especifica el protocolo de comunicación. Por ejemplo, en un hogar inteligente se pueden encontrar desplegados sensores de temperatura, de movimiento de apertura puertas, de $CO2$; o actuadores como motores para las persianas, aire acondicionado, ventilador o bombillas; los cuales pueden ser controlador por una Raspberry o un NodeMCU ESP8266.

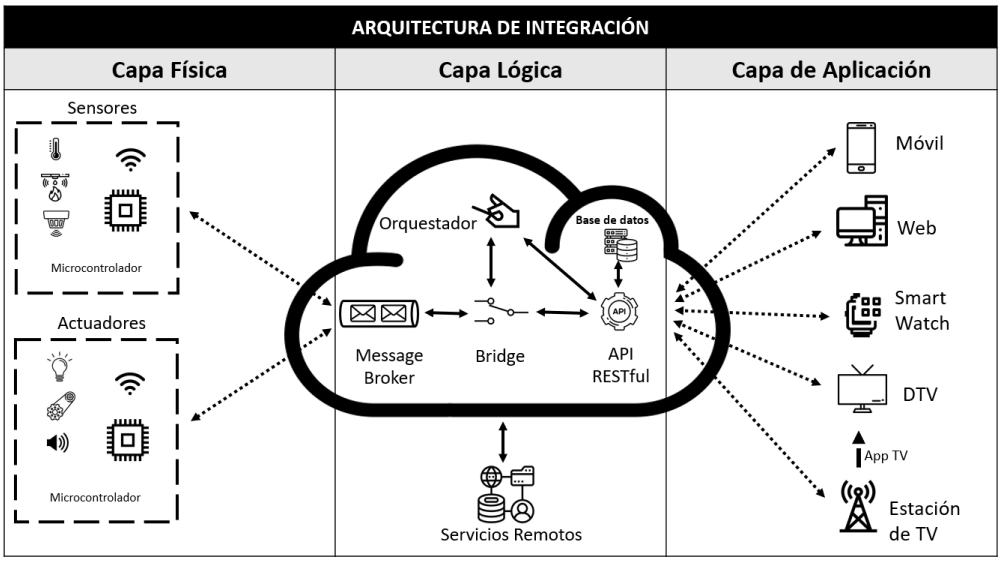


Figura 3.2: Arquitectura de integración IoT propuesta.

— *Capa lógica*: junto con la Capa física son el *back-end* del sistema y permite coordinar la información proveniente de las capas física y de aplicación, para lo cual consta de:

- Broker, recibe todos los mensajes de los clientes y luego enruta los mensajes a los clientes de destino adecuados. Por ejemplo, el *Broker* puede admitir mensajes MQTT, AMQ o Kafka
- Bridge, enruta los mensajes provenientes del Broker hacia la API RESTful o el Orquestador. Por ejemplo el *Bridge* puede recibir los mensajes MQTT y enviar la información a un servicio web.
- API RESTful, despliega los métodos HTTP (GET, POST, PUT, DELETE) para cada sensor y actuador. Por ejemplo, un servicio RESTful permite gestionar los recursos con la información proveniente de los sensores y actuadores por otros servicios que lo requieran.
- Orquestador, coordina las secuencias de llamado a los servicios REST o Bridge, para la ejecución de la lógica de negocio. Por ejemplo, un orquestador puede activarse cuando un sensor de movimiento detecte la presencia de una persona, en ese caso puede consultar los servicios RESTful de lo otros sensores como un de apertura de ventanas o un servicio remoto para saber la fecha y la hora; y decidir si se activa una sirena y se encienden las luces.

— *Capa de aplicación*: corresponde al *front-end* del sistema y permite la interacción de los usuarios en interfaces no web. A pesar de que existen varias alternativas de

interfaz con el usuario se ha optado por la DTV y móvil, con el fin de validar la aplicabilidad no solo en entornos web. Por ejemplo, el usuario de un sistema IoT cuando se encuentra fuera de su casa, puede consultar el estado de los sensores de su hogar y saber si su familia ha llegado y enviar un mensaje para que lo avise mediante una alarma o un aviso.

Los principales componentes de la arquitectura son los **productores de eventos**, y los **consumidores de eventos**. Los productores de eventos (por ejemplo, sensor, orquestador, TV app, Mobile app) generan los eventos sujetos a ser coordinados por el orquestador para procesar situaciones relevantes. Los consumidores de eventos (por ejemplo, orquestador) reciben los eventos y desencadenan tipos concretos de **acción**, los cuales se encuentran en la capa Lógica. El orquestador emplea la información de los productores de eventos para *procesar, transformar y enriquecer* la información. Con esta nueva información, el orquestador envía mensajes a los consumidores de eventos para establecer su nuevo estado.

En la arquitectura, además, se determinó el formato para la información que se intercambia entre los productores y consumidores de eventos, el cual es una propuesta de estandarización inspirada de la Thing Description de la WoT [W3C, 2020] y del trabajo de Zhou [Zhou et al., 2018]. La propuesta es una versión reducida de la descripción de un componente, con el fin de que sea procesado por dispositivos que disponen de menos recursos de cómputo; además, evita saturar el ancho de banda de la red si se despliegan muchos nodos. El formato para la estandarización de la información propuesto es por medio de objetos JSON, con la siguiente estructura:

— *id*, identificador único del evento.

— *date*, fecha de ocurrencia del evento.

— *time*, hora de ocurrencia del evento.

— *location*, ubicación física del objeto.

— *attribute*, valor del evento.

— *artefact*, nombre del objeto.

— *property*, identificador del tipo de dato del valor del atributo.

3.3.1. Capa Física

En la arquitectura propuesta, los objetos inteligentes envían la información captada por un sensor, o reciben información del estado al cual debe cambiar un actuador. No obstante, estos objetos, al interactuar con las personas, presentan el problema de la impredecibilidad, como por ejemplo, un sensor de movimiento que detecta cuando una persona se mueve. En este caso, se produce un evento asíncrono. Para esta particularidad, la **Capa Lógica** debe admitir una interacción asíncrona, lo cual entra en el ámbito de una arquitectura EDA. Es así que en la arquitectura propuesta se considera emplear mensajería ligera, que es más fácil de procesar por muchos dispositivos IoT, ya que

requiere de menor costo computacional, lo que a su vez se ve reflejado en menor consumo de energía y ancho de banda de la red.

Por último, la capa física describe la configuración de hardware y software del controlador que se encarga de gestionar las señales físicas que usan los sensores y los actuadores; y establecer la conectividad a Internet. En el caso de los sensores y actuadores, el controlador debe considerar que existen señales analógicas y digitales, que son procesadas en puertos diferentes con electrónica diferente, y en el caso de las comunicaciones, estas pueden ser inalámbricas (ZigBee, Lorawan, Zig Fox, Bluetooth, WiFi, etc) o cableadas (802.3, serial), pero en cualquier caso debe tener conectividad con internet y manejo del protocolo HTTP. Es así que esta capa aportará con la información del mundo físico que será procesada por la capa Lógica. Por ejemplo, el dueño de un almacén puede desplegar un sistema de seguridad con sensores de movimiento, sensores de apertura de puertas, relés para el control del encendido de la iluminación y una alarma. Con lo cual, si el sistema detecta movimiento en la madrugada puede activar automáticamente la alarma, encender la luces y enviar una notificación al *smart phone* del dueño.

3.3.2. Capa Lógica

En la capa lógica el componente *Bridge*, permite realizar una comunicación entre EDA, por medio de mensajería ligera (MQTT, Message Queuing Telemetry Transport) y los microservicios. Para ilustrar la utilidad de este artefacto, considere un cliente software que desea cambiar el estado de una bombilla. Para ello, dicho cliente debe ejecutar un método POST con el nuevo estado para el dispositivo que se envía al tópico específico del dispositivo. Por el contrario, si el cliente quiere obtener la información del estado de un sensor en tiempo real, debe ejecutar un método GET, de la API REST específica del dispositivo, que extrae el último mensaje del tópico del dispositivo. Además, cada servicio tiene un conjunto de operaciones (GET, POST, PUT, DELETE) para acceder a la información de cada nodo de hardware. La descripción del servicio consta de un nombre comprensible para los humanos y una URI, que contiene el nombre de host y el número de puerto del recurso. Estas especificaciones con los verbos HTTP se detallan a continuación:

— `GET port/device/`, devuelve todos los valores históricos del sensor o actuador.

— `GET port/device/last`, devuelve el último valor del sensor o actuador.

— `GET port/device/{n}`, devuelve un valor en particular del sensor o actuador, de acuerdo al código identificador del evento.

— `PUT port/device/{n}`, actualiza un valor en particular del sensor o actuador, de acuerdo al código identificador del evento.

— `POST port/device/`, inserta un nuevo valor para un sensor o actuador.

— `DELETE port/device/{n}`, elimina un valor para un sensor o actuador, de acuerdo al código identificador del evento.

Es así que el componente *Bridge* permite la integración de los objetos físicos con los componentes web. De esta manera, la coordinación de todos los servicios del sistema se realiza en base el patrón Saga [Richardson, 2018], que es una secuencia de transacciones locales donde cada transacción actualiza la información dentro de un solo servicio. En la arquitectura propuesta se contempla el uso de los dos conceptos de Saga (orquestación y coreografía), al establecer la posibilidad de coreografías de orquestadores, con el fin de alcanzar acciones más complejas. Es decir, un orquestador puede ser parte de un sistema más grande, en el cual actúa como una señal de *trigger* para iniciar una cadena de procesos más complejos.

Para ilustrar la utilidad de este artefacto considérese una persona que está en la sala de su casa una película en su SmartTV y automáticamente la persiana se cierra para evitar reflejos en la pantalla; pero como todavía hay luz solar, la bombilla se apaga. Como se aprecia en el ejemplo, existen algunos eventos simples que se han coordinado para que la experiencia de mirar una película sea más confortable. Aún cuando el ejemplo solo ha considerado una situación particular con la televisión, la arquitectura propuesta permite integrar más componentes de hardware como otros sensores y actuadores, como sistemas de seguridad o control de agua y gas, o servicios remotos, como los servicios meteorológicos que pueden vincularse con actuadores locales para activar la calefacción o el aire acondicionado. Por último, estos eventos simples pueden combinarse para ofrecer eventos más complejos, como en el caso del usuario mire la televisión y con una notificación de lluvia del servicio remoto, el sistema automáticamente encienda la cafetera y realice un pedido a un restaurante para solicitar una "tapa de migas" (algo muy típico en la ciudad de Almería). Evidentemente, estas acciones, ligadas a los eventos, dependerán de situaciones de contexto particulares, bien a los usuarios o de la propia zona.

3.3.3. Capa de Aplicación

La visión actual de IoT se centra en la infraestructura, gestión y análisis de la gran cantidad de datos generados [González-García et al., 2017] y solo un conjunto limitado de investigaciones [Brambilla et al., 2018], [Broil et al., 2009], [Nazari and Semsar, 2017] abordan el diseño de los interfaces para la interacción con el usuario. Por este motivo, la capa de Aplicación de la arquitectura propuesta se centra en el Front-End de los sistemas IoT.

En esta capa de la arquitectura se establece el medio de interacción del usuario con el sistema. Para lo cual el IoT permite diversas alternativas como la web, los *smart phone* o los *smart watch*. Sin embargo, no son las únicas plataformas, también es posible encontrar aplicaciones en los *smart TV* o en la TV digital. Es por esto que la arquitectura propuesta permite que estas plataformas interactúen con el sistema como clientes REST de los recursos expuestos por la Capa Lógica. Con lo cual los usuarios del sistemas IoT pueden pueden monitorizar el estado de los sensores y actuadores, por medio de los servicios RESTful [Cheng et al., 2018]; o generar eventos que inicien una cadena de procesos en el Orquestador [Cheng et al., 2017]. Por ejemplo, el dueño de un hogar inteligente se encuentra de viaje y tiene sus plantas con un sistema de riego automático. Este caso desde su *smart phone* puede activar la secuencia de apertura de la válvula de agua, encendido del bomba de succión de agua y la apertura de la persianas para que ingrese luz.

3.4. Metodología de implementación

Una tecnología que podría facilitar el desarrollo de aplicaciones de acuerdo a la arquitectura propuesta es la Ingeniería Dirigida por Modelos (MDE) [Zolotas et al., 2017], debido a que permite a los desarrolladores estandarizar y automatizar el proceso de desarrollo de software. De esta manera, es posible continuar expandiendo los sistemas para cubrir una gran parte de las plataformas mediante el uso de modelos y transformaciones para la especificación y generación de aplicaciones de forma automática o semiautomática [Bruneliere et al., 2017].

La metodología propuesta para el desarrollo de aplicaciones IoT requiere seis procesos divididos en dos etapas:

(a) **Especificación**, en la que se crean las herramientas para la creación automática de aplicaciones IoT;

(b) **Desarrollo**, en la cual se definen los escenarios particulares para las aplicaciones IoT y se generan los artefactos de software específicos.

Además, se definen dos roles inspirados en la propuesta de Boubeta-Puig en su trabajo [Boubeta-Puig et al., 2015], que intervienen en cada una de las etapas:

(a) **Experto en dominios**: son las personas que tienen un vasto conocimiento de un área específica, los cuales realizan la especificación de los dominios de aplicación y establecen las reglas de transformación de los modelos.

(b) **Usuario final**: son las personas que tienen el conocimiento requerido para desarrollar las aplicaciones específicas en un dominio particular, empleando las herramientas creadas en la etapa de especificación. El usuario final, a su vez, también puede ser un experto en el dominio.

3.4.1. Especificación (Experto en dominios)

En la etapa de especificación, se diseña el metamodelo y se implementa un proceso de transformación M2T. Además, se crean las herramientas que podrá utilizar el usuario final para el desarrollo de las aplicaciones IoT. Esta etapa consta de tres procesos que se ilustran en la Figura 3.3, y se detallan a continuación:

— *Generación del modelo* (paso 1). En este paso se define un metamodelo que es una abstracción del funcionamiento de los sistemas IoT, enfocado en el diseño del hardware de los sensores y actuadores, así como el software de control y de interacción con las personas.

— *Generación de reglas y el editor gráfico* (pasos 1 y 2). Este paso consta de dos fases que no tienen un orden secuencial:

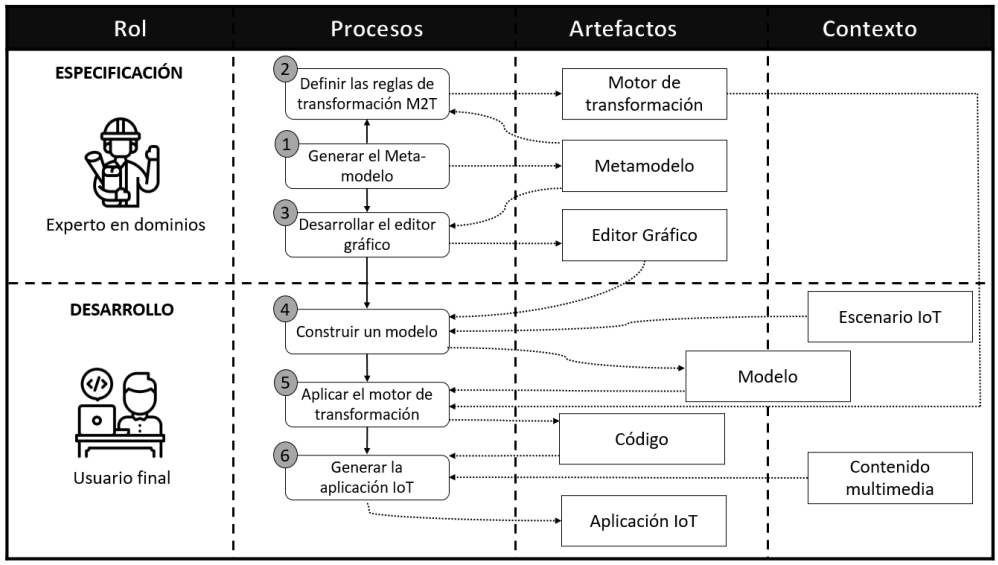


Figura 3.3: Metodología basada en modelos para la integración.

— *Definir las reglas de transformación M2T* (paso 2): en este paso se establecen las reglas de transformación que emplea el motor de transformación para que, en base a una instancia particular del metamodelo, se generen instancias de código fuente ejecutable específicas de cada plataforma.

— *Desarrollar el editor gráfico* (paso 3): en este paso se construye el editor gráfico que en base al Metamodelo definido en el paso 1, genere un modelo específico para una aplicación IoT.

Los pasos 2 y 3 no siguen un orden concreto en la secuencia descrita y se pueden presentar uno antes que el otro; por este motivo, en la figura, el flujo operacional sale en los dos sentidos tras generar el metamodelo (paso 1).

3.4.2. Desarrollo (Usuario final)

En la etapa de desarrollo, el usuario final diseña el modelo de una aplicación y genera los artefactos de software que se desplegarán en cada una de las plataformas específicas de la aplicación IoT. Esta etapa consta de tres procesos consecutivos, ilustrados en la Figura 3.3, y que se detallan a continuación:

— *Construcción de un modelo* (paso 4). En este paso se crea un modelo particular para una aplicación IoT empleando el editor gráfico creado en la etapa de Especificación. En este paso el usuario final de las herramientas puede desarrollar aplicaciones IoT aun sin disponer de un conocimiento profundo de las plataformas. Este es un paso semiautomático y requiere que el usuario configure las propiedades específicas,

como el tipo de interfaz, conexiones entre los componentes, nombres de los archivos multimedia, entre otros.

— *Aplicación del motor de transformación* (paso 5). En este paso se realiza el proceso de transformación M2T, el cual toma como entrada el modelo específico de la aplicación y genera los artefactos de software específicos de cada plataforma.

— *Generación de la aplicación IoT* (paso 6). En este paso se despliegan los artefactos de software específicos (por ejemplo `.ino`, `.json`, `.java`, `.xml`, `.ncl`, `.lua`, `.bal`) en cada una de las plataformas y se incorpora los recursos multimedia específicos que emplea la interfaz de usuario.

3.5. Trabajos relacionados

Para determinar las investigaciones relevantes se siguió la metodología descrita en el Capítulo 1, para lo cual se estableció una lista con los términos principales de búsqueda: Comunicación, Arquitectura, Integración, SOA, MDE y EDA. Con los trabajos seleccionados se realizó un resumen con el aporte de cada uno y una Tabla 3.1 para visualizar de forma rápida los aportes.

En [Jazayeri et al., 2015], los autores evalúan cuatro estándares abiertos (OGC PUCK sobre Bluetooth, TinySOS, SOS sobre CoAP, y API OGC SensorThings) para dispositivos IoT con recursos limitados, con el propósito de abordar la interoperabilidad en dispositivos IoT. Para la evaluación, se analiza el consumo de memoria, el tamaño del mensaje de solicitud, el tamaño del mensaje de respuesta y la latencia de respuesta para comparar la eficiencia de los protocolos implementados. La principal contribución de esta investigación es explorar los posibles enfoques para lograr la interoperabilidad entre dispositivos IoT de clase 1 (dispositivos con aproximadamente 10 KB de RAM y 100 KB de espacio de código). Los autores afirman que al implementar estos protocolos los dispositivos no solo se vuelven autodescriptibles, autónomos e interoperables, sino que también se pueden desarrollar aplicaciones con interfaces estandarizadas. Sin embargo con respecto a la propuesta de esta tesis solo se contempla una integración a nivel físico y no se aborda la lógica de negocio ni la interfaz de usuario.

	IoT	**Comunic.**	**Arquitec.**	**Integración**	**SOA**	**MDE**	**EDA**
[Jazayeri et al., 2015]	✓	✓					
[Darabseh and Freris, 2019]	✓		✓	✓			
[Kathiravelu et al., 2018]	✓			✓	✓		
[Thramboulidis et al., 2019]	✓				✓		
[Grace et al., 2016]	✓				✓	✓	
[Dar et al., 2015]	✓		✓		✓		
[Guinard et al., 2010]	✓		✓		✓		
[Zhang et al., 2014]	✓		✓		✓	✓	✓
[Cedeno et al., 2013]	✓		✓	✓	✓	✓	
[Chen and Englund, 2017]	✓			✓		✓	
Propuesta	✓	✓	✓	✓	✓	✓	✓

Tabla 3.1: Resumen de las principales características cubiertas.

En la obra de Darabseh and Freris [Darabseh and Freris, 2019], los autores proponen una arquitectura holística para sistemas ciberfísicos (CPS, Cyber-Physical System) e IoT. Para ello, se caracteriza el flujo de datos, el flujo de comunicación y el flujo de control de sensores, actuadores, controladores y coordinadores. Además, se proponen un mecanismo para la distribución de la etapa de control por medio de varias capas jerárquicas especializadas. Por último, los autores proponen una capa de middleware para encapsular unidades y servicios para operaciones de tiempo crítico en entornos altamente dinámicos. La arquitectura comprende tres dominios principales: el espacio físico, el ciberespacio y el espacio de control estructurado. Con respecta a la presente propuesta no se aborda los mecanismos de comunicación ni la interfaz de usuario.

Los autores en [Kathiravelu et al., 2018] describen los CPS definidos por software y la adaptación de los principios de diseño de redes definidas por software (SDN, Software Defined Networks) en CPS. La propuesta coordina cada paso de ejecución de CPS, realizado por un microservicio, a través de una arquitectura de controlador SDN extendida. La propuesta reduce la complejidad del cálculo y la limitación de los recursos al desacoplar y descomponer la ejecución de CPS en flujos de trabajo de microservicios y descargar los flujos de trabajo a entornos de borde. Sin embargo, el descubrimiento de la disponibilidad de recursos y la implementación del flujo de trabajo son invocaciones que los nodos deben realizar para aprovechar los beneficios del perímetro en el cual se encuentra el controlador de borde. En este trabajo a diferencia de la presente propuesta no se aborda los aspectos de hardware pero existe una coincidencia en el empleo de microservicios para la coordinación de los sistemas.

En [Thramboulidis et al., 2019], los autores afirman que el paradigma de microservicios tendrá un impacto significativo en la forma en que se desarrollarán los sistemas de fabricación futuros. Proponen la integración de tecnologías IoT con la arquitectura de microservicios y examinan escenarios alternativos para su explotación en sistemas de fabricación. Los microservicios se describen mediante tecnologías web y están disponibles para su descubrimiento y uso durante el tiempo de desarrollo del sistema de ensamblaje. También están disponibles, durante la operación del sistema, lo que conduce a un sistema de ensamblaje flexible. Sin embargo, a diferencia de esta propuesta no se aborda la construcción de los objetos de hardware ni se describen otras interfaces de usuarios más que la web en sí.

La propuesta presentada por los autores en [Grace et al., 2016], sugiere emplear ingeniería basados en modelos para reducir el esfuerzo de desarrollo y asegurar que los sistemas de software complejos interoperen entre sí. En el trabajo se presenta un editor de modelos gráficos y una herramienta de prueba para resaltar cómo un modelo visual mejora las especificaciones textuales. Los modelos de interoperabilidad propuestos son artefactos de software visual reutilizables que modelan el comportamiento de los servicios de una manera liviana e independiente de la tecnología. Los modelos son una combinación de especificación de arquitectura y especificación de comportamiento. Estos modelos se basan en máquinas de estados finitos (FSM, Finite-State Machine); hay una serie de soluciones de prueba activas basadas en FSM. A diferencia de esta tesis no se abordan las tecnologías web y la lógica de control difiere con la presente propuesta ya que emplea FSM. Sin embargo hay coincidencias en los criterios al emplear modelos para la construcción de los sistemas IoT.

Los autores Dar *et al.* en [Dar et al., 2015] introducen un enfoque orientado a los recursos para proporcionar una arquitectura para la integración de un extremo a otro de los dispositivos de IoT (*front-end*) con las aplicaciones de procesos comerciales (*back-end*). La arquitectura propuesta presenta un entorno amigable para el desarrollador de servicios de IoT, un mecanismo de gestión de eventos para propagar información de contexto desde los dispositivos de IoT, una instalación de reemplazo de servicios en caso de falla del servicio y la ejecución descentralizada de los procesos comerciales conscientes de IoT. Para ello, los autores adoptan el enfoque ROA para diseñar e implementar un modelo arquitectónico genérico que cumpla con los estándares para la integración de dispositivos con recursos limitados en anotaciones de modelado (BPMN, Business Process Model and Notation). Sin embargo, el enfoque aquí no explota la coordinación de servicios en la capa de control ni la plataforma móvil en la capa de usuario y tampoco explota ninguna plataforma en la capa de usuario.

El trabajo de Guinard *et al.* en [Guinard et al., 2010] analiza un enfoque para reutilizar y adaptar los patrones utilizados para la Web e introducir una arquitectura para la Web de las cosas. Los autores emplean el estilo arquitectónico REST para integrarse con el mundo físico. Como algunos dispositivos no pueden conectarse a Internet, proponen el uso de Smart Gateways, que son servidores web integrados que abstraen las comunicaciones y los servicios de los dispositivos no habilitados para la web detrás de una API RESTful. A diferencia de esta tesis, no explota las arquitecturas controladas por eventos en la capa física ni ninguna plataforma en la capa de usuario y tampoco aborda el tema de los modelos en su solución.

Zang *et al.* en [Zhang et al., 2014] presentan una arquitectura orientada a servicios y orientada a eventos que se basa en tres partes: (a) un grupo de recursos distribuidos, (b) un sistema distribuido basado en eventos y (c) eventos de sinergia de relación de servicios públicos. Esta arquitectura permite el acceso a datos en tiempo real y aborda la heterogeneidad de escalabilidad. Además, ejecuta servicios de IoT, manejando el intercambio de mensajes utilizando el mecanismo de eventos distribuidos. Se propone un mecanismo de sesión centrado en la información para describir el comportamiento del servicio que trabaja en eventos distribuidos, denominado sesión de evento. Analiza cómo construir una infraestructura SOA basada en eventos, donde la información de recursos se puede utilizar para crear servicios de IoT. Sin embargo, el enfoque propuesto no explota la coordinación de servicios en la capa de control, o cualquier plataforma en la capa de usuario, ni aborda el tema de MDE como parte de su propuesta.

En [Cedeno et al., 2013], los autores discuten una arquitectura colaborativa de servicios móviles. Esta arquitectura involucra un orquestador de servicios que es responsable del proceso de orquestación y también administra la interacción entre los componentes. La arquitectura también se compone de una API de coordinación que el orquestador de servicios usa siempre que detecta dependencias en los datos. Como tal, la API de coordinación invoca los métodos para recuperar los datos o pasarlos a otras actividades. La propuesta describe un modelo de servicio móvil colaborativo basado en sistemas de comunicación impulsados por eventos, como la publicación/suscripción, e identifica los requisitos y características necesarios para funcionar en un escenario de Internet de las cosas. En este trabajo aun cuando contempla la orquestación de los componentes de los sistemas IoT no explota ninguna plataforma en la capa de usuario.

La propuesta presentada por los autores en [Chen and Englund, 2017] sugiere una plataforma de composición de coreografías de servicios que permite la creación de prototipos de aplicaciones IoT. La herramienta de software de este trabajo permite la síntesis automática, la ejecución y la adaptación dinámica de la coreografía del servicio. Además, aborda las dificultades de orquestar los servicios tradicionales cuando se enfrentan a una serie de servicios y cosas heterogéneas, como se anticipa en el futuro del Internet. La plataforma aprovecha BPMN y lo adapta a una plataforma integrada para desarrollar aplicaciones basadas en coreografías. Libera a los desarrolladores de lidiar con la heterogeneidad, la seguridad y la dinámica del servicio, lo que les permite concentrarse en sus tareas comerciales profesionales. A diferencia de esta tesis este trabajo no explota las arquitecturas controladas por eventos en la capa física ni ninguna plataforma en la capa de usuario. Tampoco aborda el problema de MDE pero en cambio, aborda el diseño de *front-end* en la plataforma DTV.

3.6. Resumen y conclusiones

Aunque muchas soluciones proporcionan interfaces de alto nivel para simplificar el desarrollo de aplicaciones de IoT, a menudo se basan en HTTP, que es demasiado costoso para los escasos recursos de los dispositivos integrados. Por el contrario, las soluciones basadas en SOA más eficaces adolecen de un acceso ineficaz a los recursos físicos: algunas de ellas requieren interacción directa con dispositivos integrados; otros exponen los recursos, pero solo a través de una base de datos intermedia. Obviamente, en el primer caso, los requisitos de cálculo y almacenamiento son demasiado pesados para los dispositivos integrados, mientras que, en el segundo caso, la información proporcionada a la aplicación del usuario puede ser inconsistente y desactualizada. En última instancia, ninguna solución proporciona herramientas verdaderamente simplificadas que permitan el desarrollo de aplicaciones de IoT tanto para usuarios no expertos como para expertos, que abarquen hardware, lógica de negocio y una interfaz de usuario. Es por esto que se ha considerado una estrategia híbrida para la integración de los sistemas IoT, permitiendo así incorporar a los dispositivos de escasos recursos como son los nodos de hardware con los sistemas web tradicionales.

Capítulo 4

Metodología para la Integración de Sistemas IoT

Capítulo 4

Metodología para la Integración de Sistemas IoT

Contenidos

En el capítulo anterior se describió una arquitectura para la integración de aplicaciones IoT y una metodología basada en modelos para automatizar la creación de nuevas aplicaciones, lo cual permite definir un lenguaje específico del dominio (DSL, Domain-Specific Language) que se componga de: (a) una sintaxis abstracta, que permite representar los componentes hardware, los servicios de integración y las interfaces de usuario como metamodelos; (b) una sintaxis concreta, para crear modelos conforme al metamodelo a través de una herramienta gráfica; y (c) una transformación de modelos, para generar código fuente funcional para cada plataforma del sistema en base al modelo generado con el editor gráfico.

En este capítulo se describe la metodología propuesta para el desarrollo de aplicaciones IoT, en la que se crean tanto la sintaxis abstracta como la concreta de un DSL junto con una transformación de los modelos M2T (Model-to-Text). La Sección 4.1 presenta una introducción y se hace la descripción de los conceptos principales que se emplean. Posteriormente, la Sección 4.2 describe las tecnologías descritas en la arquitectura de integración que se plasmarán en la sintaxis abstracta, además de las tecnologías de desarrollo. En la Sección 4.3 se describe el DSL propuesto para la generación de aplicaciones IoT, para lo cual se describe la sintaxis abstracta (metamodelo), la sintaxis concreta (editor gráfico) y la transformación M2T para la generación de código fuente de cada una de las plataformas del sistema IoT. La Sección 4.4 revisa algunos de los trabajos relacionados con la implementación de la metodología propuesta. Al final del capítulo, la Sección 4.5 concluye con un breve resumen de los contenidos presentados y con algunas de las conclusiones extraídas.

4.1. Introducción

Hoy en día, el Internet de las cosas (IoT) está transformando la forma en que las personas se comunican, colaboran y coordinan su vida diaria. Este cambio se produce por el aumento de dispositivos conectados a Internet, que ofrece un ecosistema de integración tecnológica que permite: a) extraer datos de la vida cotidiana, b) analizar datos en un entorno virtual, y c) dar un valor agregado a través de análisis y algoritmos que permiten la toma de decisiones y respuestas rápidas [Zheng and Carter, 2015]. Sin embargo, los desarrolladores se enfrentan al problema de la interoperabilidad, porque los dispositivos IoT pueden ser de diversos tipos, utilizar diferentes protocolos de comunicación (HTTP, MQTT, DDS y CoA) y administrar diferentes formatos de datos (binarios, XML, JSON y GIOP) [Grace et al., 2016]. Todo esto crea un alto grado de dependencia entre el software de control y la plataforma, resultando en sistemas poco flexibles [Teixeira et al., 2017]. Para solucionar este problema, proyectos como Fiware, Nimbits o Amazon Web Services, han propuesto plataformas de integración. Sin embargo, estas propuestas no cubren todos los posibles casos, ya que existen dispositivos de diferentes fabricantes y con diferentes características. Por este motivo, una tecnología que podría facilitar el desarrollo de aplicaciones es la Ingeniería dirigida por modelos (MDE, Model-Driven Engineering) [Zolotas et al., 2017], la cual permite a los desarrolladores separar

de cada componente del sistema un conjunto de características específicas, y promueve la estandarización y automatización del proceso de desarrollo de software. De esta forma, es posible seguir expandiendo los sistemas para cubrir gran parte de plataformas mediante el uso de modelos y transformaciones para la especificación y generación de aplicaciones semiautomáticas o automáticas [Bruneliere et al., 2017].

La Ingeniería Dirigida por Modelos es un enfoque para el desarrollo de software en el que los principales artefactos son los modelos y su objetivo principal es reducir la complejidad intrínseca del software, provocada por la tecnología, los métodos y los lenguajes de programación utilizados para desarrollar software [Di Rocco et al., 2020]. Para la creación de los modelos se emplean los lenguajes específicos del dominio o DSLs (Domain-Specific Language), los cuales son lenguajes de programación que se dirigen a un dominio de problema específico. No están destinados a proporcionar funciones para resolver todo tipo de problemas y no es posible implementar todos los programas que se puede implementar con Java o C, a los cuales se conoce como Lenguajes de Propósito General (GPL, General Purpose Languages) [Babau et al., 2009]. Si el dominio del problema está cubierto por un DSL, se podrá resolver ese problema más fácil y rápidamente utilizando ese DSL en lugar de un GPL [Bettini, 2016]. La estructura de un DSL se especifica en su metamodelo, conocido como sintaxis abstracta, el cual proporciona la base para construir otro modelo; y su sintaxis concreta, que corresponde al formato de representación que puede ser textual o visual. La sintaxis abstracta de un DSL se puede definir utilizando diversas herramientas como Eclipse Modeling Framework EMF, MS Visual Studio, o Meta Edit+, lo que las convierte en el meta-metamodelo. Por otra parte, la sintaxis concreta se puede definir empleando herramientas como Sirius, EuGENia, Graphiti para entornos representaciones visuales y Xtext o TCS para representaciones textuales [Dai et al., 2009]. Además, para completar la construcción del DSL se emplean transformaciones de modelo como M2T (Model-to-Text) que permite generar código de algún lenguaje de programación a partir de las instancias del modelo creadas con la sintaxis concreta. Estas transformaciones se pueden realizar empleando herramientas como Acceleo, MOFScript, JET, Xpand o Xtend.

Para abordar el diseño de una solución a los problemas de interoperabilidad empleando técnicas de MDE, se ha diseñado un DSL, compuesto por un metamodelo (sintaxis abstracta), un editor gráfico (sintaxis concreta) y una transformación M2T. Para lo cual se eligieron las herramientas de desarrollo: (a) EMF para la sintaxis abstracta del DSL; (b) Sirius para la sintaxis concreta del DSL; y (c) Acceleo para la transformación M2T de las instancias de los modelos. Estas herramientas se eligieron por ser de código abierto, estar bien integradas y existe documentación tanto a nivel de libros como de artículos. Las herramientas creadas se basan en la arquitectura descrita en el Capítulo 3, compuesta por tres capas (Física, Lógica y de Aplicación), que permiten la interacción de tecnologías heterogéneas.

En la Figura 4.1 se ilustra el proceso seguido en la construcción de las herramientas para la generación de aplicaciones IoT. En primer lugar, se realiza el diseño del metamodelo que permite abstraer las características de los objetos inteligentes (Capa Física), Plataformas de interacción con el usuario (Capa de Aplicación) y Lógica de Control (Capa Lógica). Posterior al diseño del metamodelo, se desarrolla el editor gráfico el cual permite modelar las aplicaciones y con el cual se pueden incorporar características es-

pecíficas a cada modelo. Al final se define el proceso de generación de código a través de una transformación M2T, que permite crear código ejecutable para los componentes de software y hardware de la aplicación IoT.

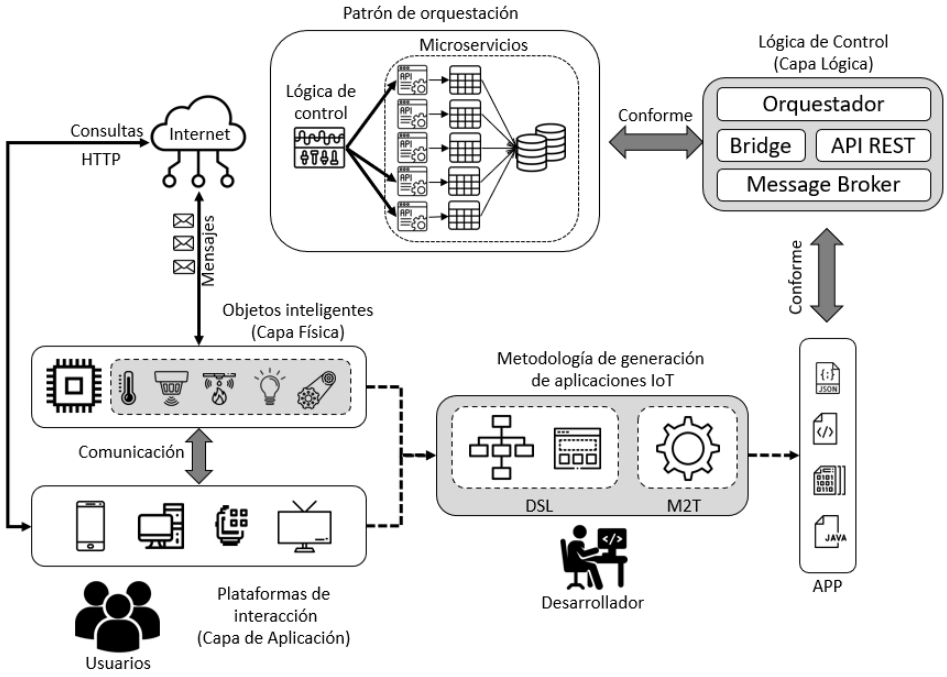


Figura 4.1: Metodología basada en modelos para la construcción de aplicaciones IoT.

4.2. Descripción de la tecnología

El IoT ha facilitado la interacción de las personas y los objetos, lo que permite el desarrollo de una gran cantidad de aplicaciones, en dominios tan diversos como la industria, la agricultura, el transporte, el comercio, el hogar, entre otros. Sin embargo, cada uno de estos dominios tiene una complejidad inherente, por lo que es necesario establecer mecanismos que permitan desarrollar aplicaciones en base a principios generales a todos los dominios. Es así, que en esta tesis doctoral se plantea una metodología basada en MDE para el desarrollo de aplicaciones IoT, que además permita la orquestación de los componentes software y hardware del sistema. Para el desarrollo de la metodología se eligió como campo de prueba el dominio del hogar inteligente, ya que es un entorno en el cual no se afectaría aspectos del giro de negocio de ninguna industria; no requiere respuestas en tiempo real como ocurre en el campo de la salud o en el ámbito militar; los componentes hardware son relativamente más baratos; y los aspectos de seguridad no afectan ningún sistema crítico.

Para comenzar, se propuso una arquitectura de integración, descrita en el Capítulo 3, que permite la interacción de los objetos inteligentes, los servicios de control y las plataformas de interacción con los usuarios. De acuerdo con la arquitectura propuesta, se creó un metamodelo, un editor gráfico y una transformación M2T, para lo cual se emplearon herramientas de código abierto como EMF, Sirius y Acceleo. Por último, para la validación de la metodología se han empleado varias tecnologías de despliegue: (a) para el caso del hardware, se consideró el controlador Node MCU ESP8266, junto con varios sensores y actuadores; y (b) para el caso del software, se empleó Ballerina para los servicios REST y el Orquestador, Node-Red para el Bridge, Java para los teléfonos inteligentes, NCL-Lua para la TV digital y Arduino para el controlador. En las siguientes subsecciones se amplían los detalles de las tecnologías empleadas.

4.2.1. Herramientas software

Como se observa en la Figura 4.1 existen dos ámbitos en la metodología propuesta, el primero es la etapa de desarrollo, en la cual se construyen las herramientas para la generación de aplicaciones IoT; y el segundo es la etapa de despliegue, en la cual se ejecuta el código creado por las herramientas. En el caso de las plataformas de desarrollo se consideró aquellas que cumplan los siguientes requisitos:

— Permiten al menos un lenguaje de programación textual.

— Disponen de librerías para conectividad a Internet.

— Permiten el desarrollo de aplicaciones con reducidas líneas de código.

— Existe una comunidad de desarrollo activa.

En base a los requisitos que deben cumplir las herramientas de software la Tabla 4.1 resume las herramientas seleccionadas y algunas de sus características, las cuales se abordarán con mayor detalle en las secciones siguientes.

HERRAMIENTA	PLATAFORMA	ESTADO	CÓDIGO	LOGO
Desarrollo	Eclipse Modeling Framework	Maduro	Abierto	
	Eclipse Sirius	Maduro	Abierto	
	Eclipse Acceleo	Maduro	Abierto	
Despliegue	Node-Red (.json)	Maduro	Abierto	
	Ballerina (.bal)	Incubación	Abierto	
	Arduino (.ino)	Maduro	Abierto	
	Android (.java)	Maduro	Abierto	
	NCL-Lua (.ncl/.lua)	Maduro	Abierto	

Tabla 4.1: Resumen de las herramientas software empleadas.

4.2.1.1. Eclipse Modeling Framewok

Eclipse Modeling Framework (EMF), proporciona facilidades de generación de código para construir herramientas y aplicaciones basadas en modelos de datos estructurados. La mayoría de los proyectos de Eclipse que se ocupan del modelado se basan en EMF ya que simplifica el desarrollo de software. El modelo se puede describir en XMI, XML Schema, UML, Rational Rose o Java. Siendo también posible especificar el modelo mediante programación utilizando Xcore. Un modelo se define en el Formato Ecore, que es básicamente una implementación de un subconjunto de diagramas de clases UML [Bettini, 2016]. Después de especificar el modelo en EMF, el generador puede crear un conjunto de clases correspondiente. El uso de EMF para crear aplicaciones también proporciona otros beneficios, como la notificación de modificación del modelo, la serialización basada en el esquema XML, el framework de validación del modelo y la interfaz de operación de objetos EMF eficaz. El núcleo básico de EMF consiste en el subproyecto EMF Core, que contiene [Fernández, 2009]:

— El metamodelo desde el cual se puede describir el modelo.

— El entorno de edición, con clases reutilizables para construir el editor del modelo.

— Herramientas para generar código a partir de modelos creados por los usuarios.

Esta herramienta fue elegida para el desarrollo de la sintaxis abstracta del DSL, debido a que es la base de muchas otras herramientas como GMF (Graphical Modeling Framework), Eugenia, Obeo Designer, entre otras.

4.2.1.2. Eclipse Sirius

Para permitir el uso práctico de un DSL es necesario disponer de herramientas para trabajar con él. En el caso específico de los DSL visuales, lo más importante puede ser un editor de gráfico, ya que permite especificar el modelo de una forma más comprensible. Actualmente existen algunos frameworks que soportan la generación de editores de gráficos a partir de metamodelos, como GMF (Graphical Modeling Framework), Diagen, Eugenia, Graffiti, Obeo Designer, Papyrus, Poseidon o Sirius. Uno de los frameworks más importantes es GMF, el cual permite crear editores propios. El resultado es un complemento que se instala en cualquier proyecto basado en la arquitectura Eclipse, permitiendo la edición gráfica de modelos que se ajusten al DSL. GMF se divide básicamente en dos partes [Fernández, 2009]: (a) El componente de herramientas GMF, que describe los símbolos, aspectos funcionales del editor gráfico, y parametriza el generador encargado de construir código y generar el plugin; y (b) El complemento que será ejecutado por otro componente GMF en tiempo de ejecución, permitiendo el uso de editores basados en GMF. Aunque GMF es extremadamente potente y permite generar complejos editores gráficos a partir de la definición de la sintaxis abstracta del DSL, el desarrollo de un editor en GMF es un proceso tedioso y propenso a errores que requiere de amplios conocimientos previos [Granada et al., 2015]. Por lo cual se consideró el framework de Sirius para el diseño del editor gráfico, el cual se basa en GMF con algunas mejoras [Vujovic et al., 2014] [Cooper and Kolovos, 2019]:

— Ambos frameworks se pueden utilizar para crear un editor gráfico fácil de usar basado en un DSL propuesto.

— Ambos editores proporcionan muchas de las funciones necesarias, pero el editor basado en Sirius proporciona mejores funciones.

— Sirius utiliza un modelo para describir los elementos del editor en lugar del código Java, por lo que el desarrollo es más rápido y menos propenso a errores.

— Sirius es más personalizable.

— Sirius permite la creación de diagramas de secuencia, tablas, matrices y árboles con un conocimiento mínimo de los aspectos internos de GMF.

— Sirus permite utilizar Java para implementar servicios personalizados.

4.2.1.3. Eclipse Acceleo

Las transformaciones M2T constituyen una tecnología fundamental en los diversos enfoques del desarrollo de software dirigido por modelos. Esta tecnología permite caracterizar artefactos capaces de traducir automáticamente modelos en formatos textuales. En la actualidad existen numerosas tecnologías, JET, MOFScript o Acceleo, las cuales permiten realizar transformaciones de modelo a texto. Sin embargo, la manera de realizar el proceso es más o menos el mismo en todos los casos [García-Molina et al., 2012]. Java Emitter Template (JET) es un proyecto de EMF, que se centra en el proceso de generar código automáticamente a partir de cualquier modelo basado en MOF (Meta Object Facility). JET utiliza plantillas muy similares a JSP (JavaServer Pages), estas plantillas se convierten en clases Java y luego se ejecutan mediante clases generadoras creadas por el usuario [Pérez, 2012]. MOFScript es una herramienta para la transformación de modelo a texto, por ejemplo, para soportar la generación de código de implementación o documentación a partir de modelos. Además admite la funcionalidad de transformación de modelo a modelo simple. Proporciona un lenguaje independiente del metamodelo que permite utilizar cualquier tipo de metamodelo y sus instancias para la generación de texto u otro modelo. La herramienta MOFScript se basa en EMF y Ecore como framework de metamodelo. MOFScript se distribuye bajo la licencia EPL (Eclipse Public License) y se proporciona como un complemento de Eclipse, pero también se puede ejecutar como una aplicación Java independiente. Las características más notables son [Pérez, 2012]:

— Genera texto de cualquier modelo basado en MOF.

— Capacidad para especificar mecanismos de control básicos.

— Manipula cadenas de texto.

— Editor con finalización automática, resaltado de sintaxis y detección de errores.

— Llamadas a funciones Java externas.

Acceleo es una herramienta para crear generadores de código que implementa la especificación de lenguaje estándar para la conversión de modelo a texto desarrollada por OMG (Object Management Group), denominada MOFM2T. Se proporciona como un complemento de Eclipse con licencia EPL y se desarrolla en lenguaje Java bajo EMF. El lenguaje MOFM2T utiliza un enfoque basado en plantillas, la cual contiene una sección dedicada en la que el texto es generado por componentes del modelo de entrada. Algunas de las características más importantes que tiene Acceleo se listan a continuación [Kahani and Cordy, 2015]:

— Genera código a través de cualquier tipo de metamodelo compatible con EMF.

— Creación de código de forma modular a través de plantillas MOFM2T.

— Editor con finalización automática, detección y reconstrucción de errores en tiempo real y resaltado de sintaxis.

— Permite llamar al sistema y bibliotecas externas a través del lenguaje Java.

Para esta investigación se optó por Acceleo para la generación de código, ya que permite un lenguaje estandarizado por la OMG y debido a que el tiempo de compilación es menor comprado a otras herramientas [Goubet, 2020].

4.2.1.4. Node-Red

Node-RED es una herramienta de desarrollo basada en flujo de código abierto para la integración de dispositivos de hardware de IoT y servicios en línea. Es una herramienta gratuita basada en JavaScript, construida sobre la plataforma Node-JS, que proporciona un editor de flujo visual basado en navegador [Lekic and Gardasevic, 2018]. Node-RED admite diferentes tipos de hardware como Raspberry Pi, Arduino, Beagle Bone Black, entre otros, y es compatible con sistemas operativos como Windows y Mac [Krishnamurthi, 2018]. En Node-Red los flujos creados se almacenan utilizando JSON (JavaScript Object Notation) y la interfaz tiene tres componentes básicos: (a) Panel de nodo, (b) Panel de flujo, y (c) Panel de información y depuración.

Node-Red tiene dos factores que lo hacen muy interesante para su aplicación en el IoT [Tobar, 2016]: (a) modelo de programación basado en procesos, el cual es muy adecuado para aplicaciones de IoT, debido a que los eventos reales son característicos y pueden desencadenar eventos de procesamiento y acción de la vida real; y (b) bloques de construcción, los cuales son nodos de entrada y salida que ocultan gran parte de la complejidad operativa, lo que significa que los desarrolladores pueden centrarse en el proceso y la integración del hardware y las funciones en lugar de los detalles de programación. Estos factores son las razones por lo cual se optó por Node-Red para la implementación del “Bridge” de la Arquitectura propuesta.

4.2.1.5. Ballerina

Ballerina se convirtió en un proyecto público de código abierto a principios de 2017, pero aún cuando es bastante nuevo tiene muchas características que pueden tener acogida en

la industria. Es un lenguaje de propósito general pero también tiene características para escribir aplicaciones distribuidas [Krishnamurthi, 2018]. Ballerina permite el intercambio de datos al proporcionar dos formatos, JSON y XML, como tipos de datos nativos. Además, los datos de valores separados por comas (CSV) son fáciles de analizar como cadenas de texto. Algunas de las funciones que incorpora Ballerina para trabajar en la red son las siguientes [Oram, 2019]:

— Creación de clientes y servidores RESTful.

— Acceso a servicios de terceros.

— Consultas bases de datos.

— Microservicios y contenedores.

— Completar llamadas asincrónicas.

— Producir y consumir datos de transmisión.

— Intercambiar datos en formatos comunes como JSON.

— Paso de mensajes.

— Métricas de rendimiento y confiabilidad.

Ballerina podría convertirse en un lenguaje de programación de referencia porque facilita la conexión entre aplicaciones y servicios en todo tipo de escenarios de integración, debido a las siguientes características [Chakray, 2020]:

— *Es accesible para todos*: requiere un conocimiento mínimo a diferencia de otros lenguajes de programación.

— *Diseño llamativo*: se basa en una representación visual comprensible para todos.

— *Integración*: ofrece un lenguaje de programación integrado, con el menor código y con el uso de diagramas.

— *Flexible*: permite utilizar complementos codificados en Ballerina.

— *Mejora constante*: se ha presentado como un producto, pero también es un proyecto en curso.

Aun cuando existen otras tecnologías para el desarrollo de aplicaciones web como JavaScript y PHP, en esta investigación se optó por Ballerina para la construcción de los servicios REST y los servicios de integración, debido a la reducida cantidad de líneas de código necesarias y a su funcionalidad para llamar servicios en forma paralela. La Tabla 4.2 resume una comparación general de los lenguajes mencionados.

LENGUAJE	IMPERATIVO	ORIENTADO A OBJETOS	PROCESAL	EVENTO CONDUCIDO
Ballerina	Si	Si	Si	Si
JavaScript	Si	Si	Si	Si
PHP	Si	Si	Si	No

Tabla 4.2: Comparación de las herramientas para el intercambio de datos de servicios web.

4.2.1.6. Arduino

Arduino fue diseñado para ser fácil de usar por principiantes que no tienen experiencia en software o electrónica. Con Arduino, se puede construir objetos que pueden responder y/o controlar diferentes señales como por ejemplo la luz, el sonido, el tacto y el movimiento. Esta plataforma es mejor conocido por su hardware, pero también necesita software para programar ese hardware. Tanto el hardware como el software se denominan "Arduino" [Margolis et al., 2020]. Todo el hardware y el software que componen la plataforma Arduino se distribuyen como código abierto y cuentan con la licencia GNU Lesser General Public License (LGPL) o la GNU General Public License (GPL). Esto permite la fabricación y distribución de placas Arduino por cualquier persona y ha permitido el desarrollo de numerosas placas compatibles con Arduino genéricas y de menor costo [Blum, 2019].

El software de Arduino es simplemente una abstracción del lenguaje C/C++ que proviene de `avr-libc` y que provee una librería de C para usar con GCC (GNU Compiler Collection) en los microcontroladores AVR de Atmel [Dunbar, 2020]. Aún, cuando existen otras plataformas como FPGA, Raspberry o microcontroladores PIC, Arduino sobresale por las siguientes ventajas:

— Arduino simplifica el proceso de uso de un microcontrolador.

— El software Arduino se puede ejecutar en sistemas operativos Windows, Macintosh OSX y Linux.

— El entorno de programación Arduino es fácil de usar para principiantes y muy flexible para usuarios avanzados.

— El software Arduino se distribuye con licencia libre.

Debido a que esta investigación tiene un componente grande de electrónica se optó por Arduino sobre otras plataformas, ya que dispone de puertos analógicos/digitales y permite la comunicación inalámbrica de una forma sencilla y embebida en la tarjeta. En esta plataforma se desarrolla el código de los controladores, encargados de acondicionar las señales para los sensores y actuadores.

4.2.1.7. Android

James Gosling de Sun Microsystems creó el lenguaje de programación Java a mediados de la década de 1990, el cual tuvo un aumento en su uso debido a la elegancia del lenguaje y la arquitectura bien concebida. Java se convirtió en un lenguaje de uso general

con especial fortaleza en servidores y en los procesadores integrados. Para vincular Java con Android, los fundadores de Android agregaron la Máquina virtual Dalvik y las bibliotecas Harmony. En el 2013 Google comenzó el proceso de reemplazar Dalvik con una nueva máquina virtual llamada Android Runtime (ART), en la cual los programas se ejecutan más rápido, consumen menos memoria y energía [Burd and Mueller, 2020]. Android se ha expandido más allá de los teléfonos móviles para proporcionar una plataforma de desarrollo para una gama cada vez más amplia de hardware, que incluye tabletas, televisores, relojes, automóviles y dispositivos de Internet de las cosas (IoT). Es así que Android permite crear aplicaciones con algunas de las siguientes características [Meier and Lake, 2018]:

— Acceso a recursos de telefonía e Internet a través de GSM, EDGE, 3G, 4G, LTE y soporte de red Wi-Fi.

— APIs para servicios basados en la ubicación, como GPS y detección de ubicación basada en la red.

— Soporte completo para integrar mapas dentro de la interfaz de usuario.

— Control completo de hardware multimedia, incluido la reproducción y grabación con la cámara y el micrófono.

— Bibliotecas de medios para reproducir y grabar una variedad de formatos de audio/video o imágenes fijas.

— APIs para usar hardware incluidos como: acelerómetros, brújulas, barómetros, sensores de huellas dactilares, WiFi, Bluetooth, NFC, entre otros.

— Servicios en segundo plano y un sistema de notificación avanzado.

— Gráficos acelerados por hardware optimizados para dispositivos móviles.

Con el propósito de permitir el acceso al ecosistema de usuarios, la barrera de entrada para los nuevos desarrolladores de Android es mínima, como se detalla a continuación [Burd and Mueller, 2020]:

— No se requiere certificación para convertirse en desarrollador de Android.

— Play Store ofrece opciones gratuitas de compra por adelantado, facturación en la aplicación y suscripción para la distribución y monetización de sus aplicaciones.

— No existe un proceso de aprobación para la distribución de solicitudes.

— Los desarrolladores tienen un control total sobre sus marcas.

Desde una perspectiva comercial, Android representa el sistema operativo de teléfonos inteligentes más común y brinda acceso a más de 2 mil millones de dispositivos activos mensualmente a nivel mundial, lo que ofrece un alcance incomparable para que las aplicaciones estén disponibles para usuarios de todo el mundo [DiMarzio, 2016]. Además,

la principal ventaja de adoptar Android es que ofrece un enfoque unificado para el desarrollo de aplicaciones. Los desarrolladores solo necesitan desarrollar para Android en general, y sus aplicaciones deberían poder ejecutarse en numerosos dispositivos diferentes, siempre que los dispositivos se alimenten con Android. Aun cuando han surgido nuevos lenguajes de programación la máquina virtual Java puede ejecutar códigos de diferentes lenguajes de programación como Kotlin, sin embargo, Java es el más común. Esta es la principal razón por la que se optó por Java para la construcción de la interfaz de usuario que se despliega en los teléfonos inteligentes.

4.2.1.8. NCL-Lua

A diferencia de la televisión analógica que codifica imágenes y sonidos de forma análoga, la televisión digital (TVD) codifica imágenes y sonidos de forma digital, permitiendo una mejor calidad de imagen, acceso a múltiples canales y servicios interactivos, por lo que la televisión digital brinda la oportunidad de crear aplicaciones interactivas. Con la TVD el consumidor puede pasar de ser un espectador pasivo para convertirse en un participante activo, ya que cambia de ser un mero difusor de contenidos a permitir el acceso a contenidos. A través de la televisión digital es posible acceder a un conjunto de servicios públicos o privados que cubren diversos campos como el comercio, la gestión administrativa, el entretenimiento, el aprendizaje, telemedicina, juegos, o el correo electrónico, entre otros campos [Soares et al., 2010].

El Foro del Sistema Brasileño de Televisión Digital (SBTVD) propuso el middleware Ginga para la creación de aplicaciones multimedia interactivas para el estándar ISDB-T de TVD. Ginga tiene dos subsistemas: Ginga-NCL, para la presentación de las aplicaciones basado en NCL/Lua; y Ginga-J para aplicaciones basado en Java [Costa et al., 2016]. NCL (Nested Context Language) se basa en XML y consta de un lenguaje específico de dominio para la creación multimedia que se centra en especificar aplicaciones multimedia con medios audiovisuales sincronizados e interacciones de usuario [Leitao et al., 2019]. Además, NCL es una Recomendación de la UIT para sistemas de IPTV y se utiliza en sistemas de TV en muchos países de América del Sur y África [Soares Neto et al., 2010]. Por su parte, Lua es un lenguaje de programación de scripting que ocupa poco espacio, funciona en varias plataformas y es bastante flexible [Szauer, 2018]. Lua fue desarrollado en 1993 por Roberto Ierusalimschy, Luiz Henrique de Figueiredo y Waldemar Celes en la Pontificia Universidad Católica de Río de Janeiro en Brasil, bajo una licencia MIT [Varna, 2012]. Las ventajas de Lua son su extensibilidad, simplicidad, eficiencia y portabilidad. Además, su curva de entrada suave lo hace bastante adecuado como primer lenguaje de programación [Jung, 2007]. La máquina virtual y el intérprete de Lua están escritos en C y contiene 21 palabras clave, por lo que es un lenguaje pequeño. Lua proporciona paradigmas imperativos, funcionales y orientados a objetos para escribir sus scripts [Emmerich, 2009].

Para esta investigación se eligió el estándar ISDB-T para televisión digital con el middleware Ginga, ya que permite ejecutar las aplicaciones en la TVD las cuales son compatibles con la recomendación para IPTV. Además, Ginga-NCL se eligió sobre Ginga-J ya que la gran mayoría de los decodificadores lo incorporan sobre Ginga-J, debido a que el hardware requerido para ejecutar las aplicaciones requiere menos recursos.

4.2.2. Componentes hardware

Además del software, también es necesario el hardware, por este motivo para el diseño de los objetos inteligentes se consideró los siguientes componentes físicos, como controladores, sensores y actuadores.

4.2.2.1. Controladores

Un controlador es un circuito integrado compacto diseñado para gobernar una operación específica en un sistema integrado. Un microcontrolador típico incluye un procesador, memoria y periféricos de entrada/salida en un solo chip. Son esencialmente computadoras personales en miniatura diseñadas para controlar pequeñas funciones de un componente más grande, sin un complejo sistema operativo (SO) de interfaz de usuario. Uno de ellos es el Node MCU, el cual es una plataforma de desarrollo similar a Arduino orientada al IoT. Tiene como núcleo un SoM (System on Module) que a su vez está basado en el SoC (System on Chip) ESP8266 [Pasika and Gandla, 2020]; un microcontrolador Tensilica L106 con arquitectura de 32 bits [Kuncoro and Saputra, 2017]; un conversor USB-Serial TTL (Transistor-Transistor Logic) CP2102; y un conector micro-USB para la programación y comunicación. Posee un regulador de voltaje de 3.3V, que permite alimentar la placa directamente desde el puerto micro-USB o por los pines *Vin* de 5V y GND. Los pines de entrada/salida (GPIO) trabajan a 3.3V por lo que para conexión a sistemas de 5V es necesario utilizar conversores de nivel. El Node MCU se puede programar en Arduino IDE usando los lenguajes C/C ++ y Lua [Sangsanit and Techapanupreeda, 2019]. La Figura 4.2 muestra el esquema del NodeMCU.

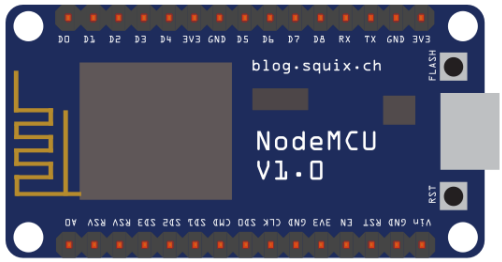
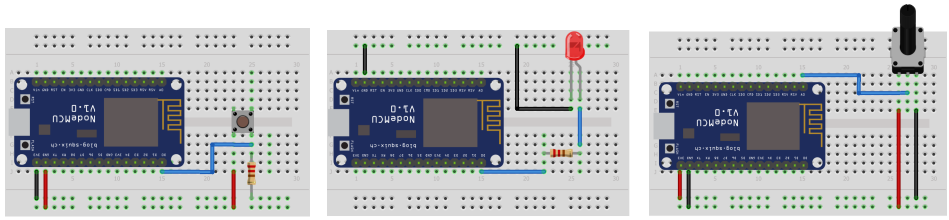


Figura 4.2: Esquemático de controlador Node MCU.

Entradas y salidas digitales. Una señal digital es una señal generada por un fenómeno electromagnético, donde cada símbolo que codifica su contenido se puede analizar en función de una cantidad que representa un valor discreto, en lugar de un valor dentro de un cierto rango. Los sistemas digitales (como los microcontroladores) utilizan lógica de dos estados representados por dos niveles de voltaje, un nivel alto (High) y el otro nivel bajo (Low). Mediante la abstracción, estos estados se reemplazan por ceros y unos, lo que facilita la aplicación de la lógica y la aritmética Booleana. La Figura 4.3.a ilustra un circuito básico para la entrada de una señal digital y la Figura 4.3.b ilustra un circuito básico para la salida de una señal digital.



(a) Entrada digital. (b) Salida digital. (c) Entrada analógica.

Figura 4.3: Puertos digitales y analógicos.

Entradas y salidas analógicas. Una señal eléctrica analógica es una señal cuyo voltaje cambia constantemente y puede tomar cualquier valor. En el caso de la corriente alterna, la señal analógica aumenta su valor con un signo positivo (+) en un medio ciclo y luego disminuye con un signo negativo (–) en el siguiente medio ciclo. Los módulos de control (como los microcontroladores) no pueden utilizar señales analógicas, por lo que deben convertirse en señales digitales antes de poder utilizarlas. En el caso del Node MCU solo se dispone de un puerto analógico que permite únicamente la entrada de datos y para el caso de la salida de datos se emplea una modulación PWM (Pulse-width modulation) que emula una salida analógica. La Figura 4.3.c ilustra un circuito básico con el Node MCU para la entrada de una señal analógica en el puerto A0.

4.2.2.2. Sensores

Con el propósito de que un controlador pueda tener información del exterior existen una gran variedad de sensores como se mencionó en el Capítulo 2. Sin embargo para la validación de la metodología se eligieron los siguientes:

— *LM-35: Sensor de temperatura.* Se utiliza para medir la temperatura ambiental y se puede utilizar en dos métodos de configuración. En el primer caso, este sensor puede medir solo temperaturas positivas, mientras que el segundo caso puede usarse para medir el rango de escala completa (tanto temperaturas positivas como negativas) [Savaridass et al., 2020]. La Figura 4.4.a muestra el circuito de conexión de un sensor de temperatura LM35 en el Node MCU.

— *DHT-11: Módulo Sensor de temperatura y humedad relativa.* El DHT11 se utiliza para medir los valores de temperatura y humedad de la atmósfera circundante. El sensor para la medición de la temperatura tiene una precisión de 8 bits. Además, los valores de temperatura y humedad enviados al microcontrolador son datos en serie. El sensor puede calcular temperaturas de entre 0 a 50 grados centígrados y niveles de humedad entre 20 % y 90 %, con una exactitud de +/– 1 % de error [Pasika and Gandla, 2020]. La Figura 4.4.b ilustra el circuito de conexión de un sensor de temperatura y humedad ambiental en el Node MCU.

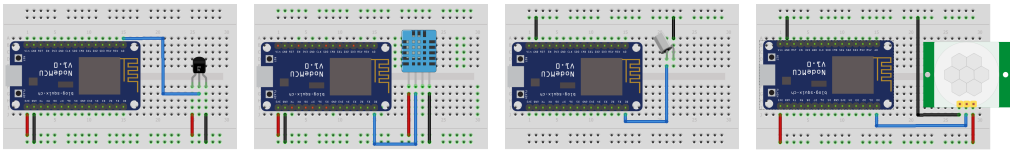
— *SW-520D: Sensor de inclinación.* El sensor permite detectar la orientación o inclinación debido a que contiene dos esferas metálicas en su interior que conmutarán

los dos pines del dispositivo de encendido a apagado en función de su inclinación respecto al vector de campo gravitacional de la Tierra [Chakraborty et al., 2018]. La Figura 4.4.c ilustra el circuito de conexión de un sensor de inclinación Node MCU.

— *HC-SR501: Módulo detector de movimiento.* Es un dispositivo de detección electrónico que estima las señales infrarrojas emitidas por los objetos en su rango de visión, utilizado principalmente en detectores de movimiento. El sensor PIR (Passive Infrared) generalmente consta de un sensor piroeléctrico, que puede distinguir varios grados de radiación infrarroja [Mahbub, 2020]. La Figura 4.4.d ilustra el circuito de conexión de un sensor de movimiento PIR en el Node MCU.

— *MQ-7: Módulo sensor Gas.* Este sensor se utiliza en dispositivos de control de calidad del aire, el cual puede medir los gases, donde la unidad de medida es PPM [Mahbub, 2020]. El pin de datos del sensor se conecta en un puerto analógico del microcontrolador y su voltaje de funcionamiento es de 5 Voltios y 40 mA (mili Amperios) de corriente [Kharade et al., 2020]. La Figura 4.4.e ilustra el circuito de conexión de un sensor de gas en el Node MCU.

— *YL-69: Módulo Sensor de humedad del suelo.* El sensor humedad del suelo dispone de dos almohadillas de detección descubiertas, las cuales funcionan como una resistencia variable. La presencia de más agua en el suelo significa que mejor será la conductividad entre las almohadillas de detección y causará una menor resistencia y una mayor salida de señal [Mahbub, 2020]. La Figura 4.4.f muestra el circuito de conexión de un sensor de humedad de suelo en el Node MCU.

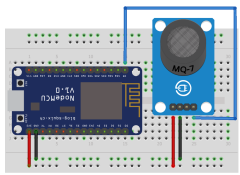


(a) Sensor de temperatura.

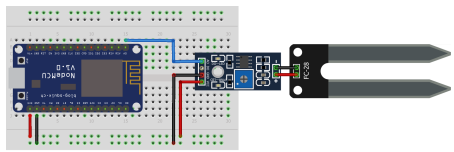
(b) Sensor de temperatura y humedad.

(c) Sensor de inclinación.

(d) Sensor de movimiento.



(e) Sensor de gas.



(f) Sensor de humedad de suelo.

Figura 4.4: Conexión eléctrica de los sensores considerados.

4.2.2.3. Actuadores

Con el propósito de que un controlador pueda interactuar con el mundo exterior existen una gran variedad de actuadores como se mencionó en el Capítulo 2. Sin embargo para la validación de la metodología se eligieron los siguientes (ver Figura 4.5:

— *Módulo relé de 1 canal.* El módulo de relé es una especie de interruptor operado por un electroimán. El electroimán necesita una pequeña cantidad de voltaje que se proporcionará a través del microcontrolador y, una vez que se active, moverá el contacto para hacer el circuito de alto voltaje [Mahbub, 2020]. La Figura 4.5.a ilustra el circuito de conexión del módulo de relé en el Node MCU.

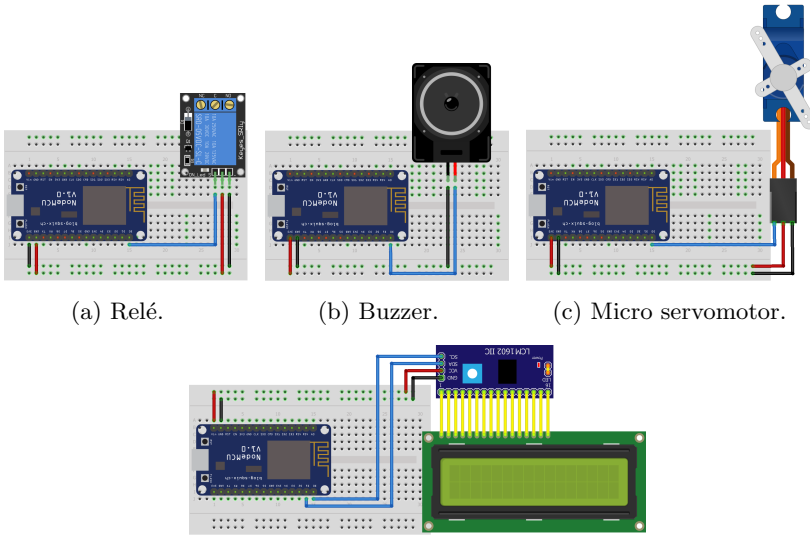
— *Buzzer activo electromagnético.* Un Buzzer es un dispositivo de señalización de audio que se puede clasificar en tres categorías: mecánico, electromecánico o piezoeléctrico y en base a la resonancia de la cavidad acústica genera un sonido audible [Mahbub, 2020]. La Figura 4.5.b ilustra el circuito de conexión de un Buzzer en el Node MCU.

— *SG90: Micro servomotor.* La mayoría de los micro servomotores operan de 4.8 a 6.5 Voltios, cuanto mayor es el voltaje mayor es el par que se puede lograr, pero lo más común es que operen a 5 Voltios y pueden girar solo de 0 a 180 grados debido a su disposición de engranajes [Chakraborty et al., 2018]. La Figura 4.5.c ilustra el circuito de conexión de un micro servomotor en el Node MCU.

— *Pantalla LCD I2C.* La pantalla de cristal líquido (LCD, Liquid-Crystal Display) es una combinación de líquido y sólido que se utiliza para producir una imagen que es visible. Se compone de dos láminas polarizadoras y un cristal líquido [Dhatri et al., 2019]. La pantalla LCD dispone de 2 líneas con un bus de interfaz I2C y funciona con una tensión de alimentación de 5 Voltios [Lapshina et al., 2019]. La Figura 4.5.d ilustra el circuito de conexión de una pantalla LCD con conexión I2C en el Node MCU.

4.3. Lenguaje Específico de Dominio

Como ya se ha comentado, IoT cubre una gran cantidad de plataformas tanto de software como hardware interconectadas. Sin embargo, cada vez existen nuevos productos y requerimientos de los usuarios. Por lo que surge la necesidad de disponer de una metodología, como la propuesta en el Capítulo 3, que permita integrar todas estas tecnologías. De ahí que en este capítulo se presenta una propuesta de un lenguaje específico de dominio (DSL, Domain-Specific Language) compuesto por un metamodelo, un editor gráfico que ayuda a definir instancias de modelo conforme al metamodelo, y una transformación modelo a texto (M2T) que genera de forma automática código ejecutable conforme al modelo creado con el editor gráfico. Se ha empezado describiendo las herramientas de software y hardware empleadas para validar la aplicabilidad de la metodología, continuando en esta sección con la descripción formal del lenguaje.



(a) Relé. (b) Buzzer. (c) Micro servomotor.

(d) Pantalla LCD.

Figura 4.5: Conexión eléctrica de los actuadores considerados.

La Figura 4.6 muestra las tres partes que constituyen el DSL propuesto para la generación de aplicaciones IoT:

(a) **Sintaxis abstracta**. Corresponde al metamodelo en el cual se hace una representación abstracta de las posibles interacciones que pueden surgir entre los componentes software y hardware, conforme a la arquitectura de integración propuesta. El metamodelo está constituido por las metaclases que modelan las capas Física, Lógica y de Aplicación implementadas en Eclipse Modeling Framework.

(b) **Sintaxis concreta**. Corresponde al editor gráfico el cual está construido en función al metamodelo e implementado en Eclipse Sirius. El editor gráfico permite construir nuevos modelos de aplicaciones conforme al metamodelo, por medio de una interfaz gráfica con la cual el desarrollador interactúa para arrastrar los componentes de la aplicación que desea, e interconectarlos entre sí. Para la construcción del editor gráfico se consideraron las herramientas de software de despliegue y los componentes hardware descriptos anteriormente enfocados al dominio de aplicación del hogar inteligente.

(c) **Transformación M2T**. Corresponde a la generación de los artefactos de software por medio de una transformación M2T implementada en Eclipse Acceleo. Los artefactos contienen código que se genera conforme a los modelos creados por el desarrollador con el editor gráfico, el cual puede ejecutarse en las herramientas de software de despliegue.

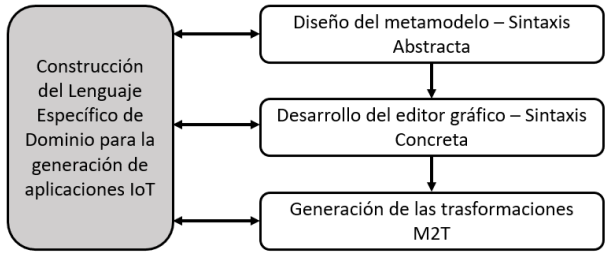


Figura 4.6: Metodología basada en modelos para la construcción de aplicaciones IoT.

En la Figura 4.7 se ilustra un escenario de hogar inteligente, que sirve de guía para la implementación del DSL, en el cual se cuenta con una App para el móvil y para la DTV, con las que los usuarios pueden interactuar con los objetos inteligentes desplegados dentro del hogar, que contiene sensores y actuadores. Además, para gestionar la interacción de los componentes de la aplicación se implementa el Orquestador, los servicios REST, el Bridge y el Message Broker, que reciben las consultas y los mensajes provenientes de los componentes, con lo cual se establece la lógica de negocio del sistema. Por último, se observa que al sistema se pueden integrar otras interfaces ya que, al implementar los criterios de la WoT, cada componente es accesible como un servicio web, con lo cual se pueden desarrollar nuevas aplicaciones que se acoplen con facilidad. Así también, se ilustra la opción de nutrir a los servicios remotos con información proveniente de otros hogares con los cuales se puede hacer un procesamiento complejo de eventos. Esta información es accesible por cada hogar para ejecutar acciones, como apertura de ventanas en caso de contaminación o activación de alarmas en caso de desastres naturales.

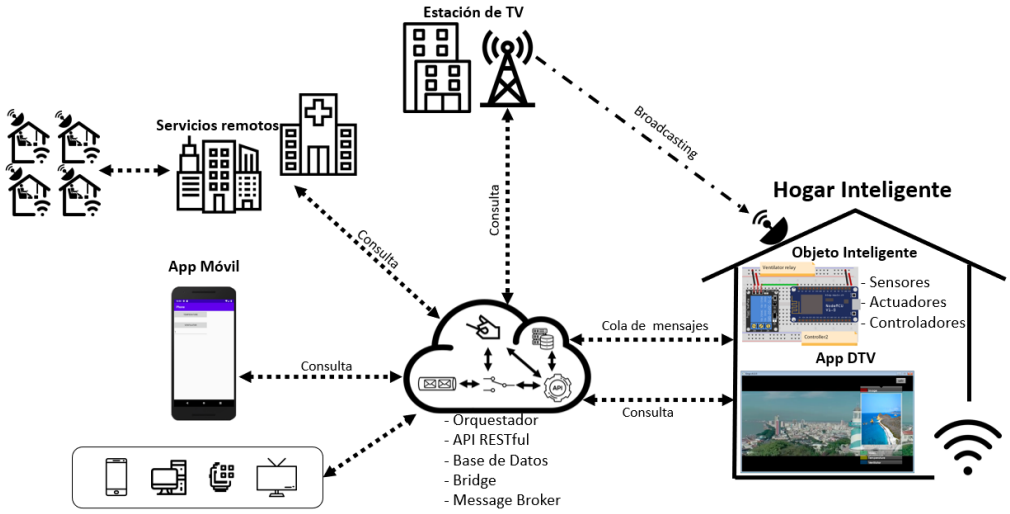


Figura 4.7: Escenario de ejemplo para el diseño del metamodelo.

4.3.1. Sintaxis Abstracta del Lenguaje

En esta subsección se describe el metamodelo para la integración de sistemas IoT siguiendo un enfoque basado en modelos y servicios. Este metamodelo permite definir los escenarios como modelos que ignoran la complejidad de la sintaxis específica de las plataformas de software y hardware que se utilizan para implementarlos. En la página web del proyecto[1] se muestra el metamodelo completo para la integración de sistemas IoT, desarrollado en EMF (véase también Anexo A).

Como se comentó en el capítulo anterior, la arquitectura de integración consta de tres capas: Física, Lógica y de Aplicación, donde se separan los ámbitos de acción de cada componente. En la capa Física se identifican los componentes hardware, como los controladores, los sensores y los actuadores. Además, se especifica el proceso de comunicación del hardware con los componentes software y las características de la infraestructura sobre la cual se despliega la aplicación. La capa Lógica identifica los procesos de integración de los componentes software y hardware del sistema IoT. Por último, en la capa de Aplicación se identifican los mecanismos de interacción del usuario con el sistema. En este caso, la propuesta implementa dos interfaces distintas a la Web, como son la DTV y los Smart Phone, no limitándose exclusivamente a estas plataformas ya que todos los recursos son accesibles como servicios web, permitiendo que se puedan desarrollar nuevos componentes para integrar en el sistema. En la Figura 4.8 se ilustra un metamodelo simplificado con los principales componentes de la arquitectura de integración; en el cual las metaclases se han agrupado de acuerdo con las tres capas de la arquitectura.

Como se observa en la Figura 4.8, existen varias metaclases que describen distintos aspectos de cada capa de la arquitectura, como los tipos de servidores, los protocolos finales del enlace de comunicación *last mile* o los mecanismos de coordinación de mensajes. La razón por la que se definen varios niveles dentro de cada capa es que esta división por niveles permite organizar los componentes del sistema, lo que a su vez facilita que el metamodelo pueda seguir creciendo, ya que se facilita el proceso de agregar nuevas funcionalidades como un bloque conceptual. Esto se puede observar en el caso de la capa de aplicación, en la cual se han descrito dos plataformas que tienen componentes en común, pero también presentan sus particularidades, como es el caso de DTV. Esta tiene la característica de la temporalidad en la que se puede utilizar, ya que la aplicación solo es accesible desde el canal de bajada de la señal de TV y para la interactividad emplea una conexión a Internet. En la Tabla 4.3 se muestra cada una de las capas de la arquitectura con sus niveles y principales metaclases. En las siguientes secciones se muestra el metamodelo en detalle de cada uno de los niveles de la arquitectura que representan las aplicaciones IoT, describiendo cada una de las metaclases que lo forma.

4.3.1.1. Nivel de Infraestructura

En este apartado se describe el metamodelo utilizado para el nivel de Infraestructura de la capa Física. Para el modelado, se utilizan técnicas de diseño de MDE con el objetivo de construir un metamodelo como muestra la Figura 4.9. Esta representación ayuda a comprender las diferentes partes de la infraestructura necesaria para el despliegue de

[1]Framework SI4IOT (Grupo ACG, UAL): `http://acg.ual.es/projects/cosmart/si4iot/`

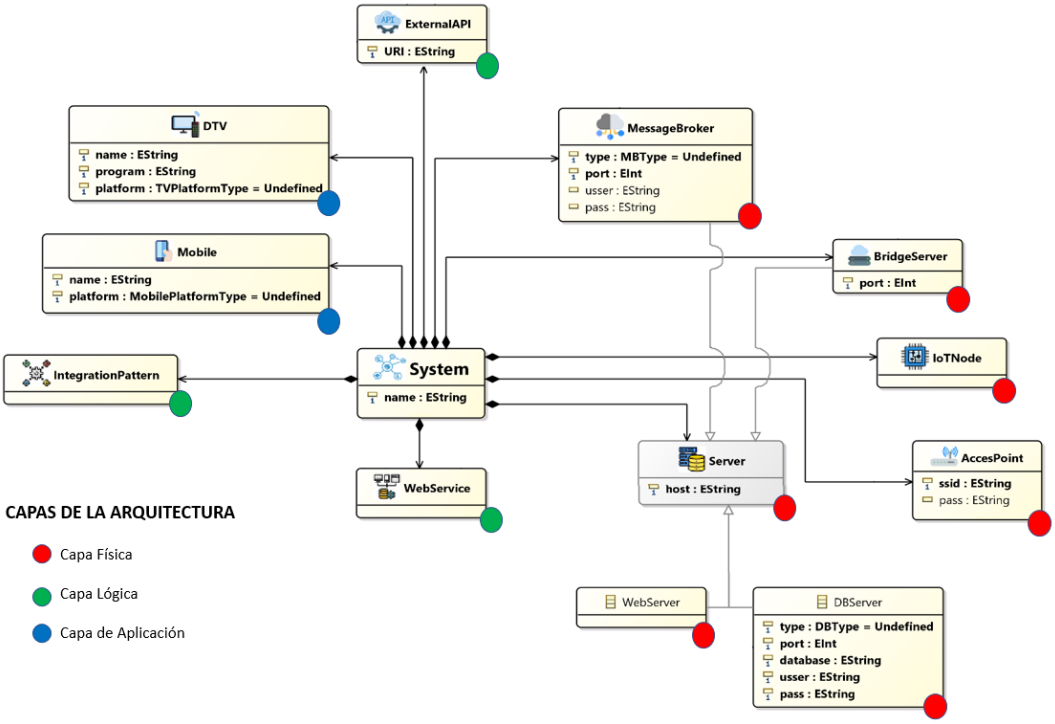


Figura 4.8: Metamodelo simplificado para la arquitectura de integración.

CAPAS DE LA ARQUITECTURA	NIVEL	DESCRIPCIÓN	METACLASES
Capa Física	Infraestructura	Agrupa las metaclases que describen toda la infraestructura de red requerida para el funcionamiento de los objetos inteligentes y para el despliegue de los servicios web.	`MessageBroker` `BridgeServer` `Server` `WebServer` `DBServer` `AccesPoint`
	Hardware	Agrupa las metaclases para la construcción de los objetos inteligentes	`IoTNode`
Capa Lógica	Control	Agrupa las metaclases que permiten la integración de los servicios y establecer la lógica de negocio.	`IntegrationPatterns` `WebService` `ExternalAPI`
Capa de Aplicación	DTV	Agrupa las metaclases para el diseño de la interfaz DTV.	`DTV`
	Móvil	Agrupa las metaclases para el diseño de la interfaz móvil.	`Mobile`

Tabla 4.3: División de las capas de la arquitectura de Integración.

la aplicación IoT. El metamodelo permite definir cinco tipos de componentes: `Message Broker`, `Web Server`, `Data Base Server`, `Bridge Server` y `Acces point`. Los cuatro primeros representan los servidores en los cuales se despliegan los servicios de la aplicación. Para lo cual disponen de atributos que identifican el puerto en el cual se ejecutan y de ser necesario las credenciales de acceso. El último componente representa la infraestructura de red (en este caso WiFi) a la cual se conectan los objetos inteligentes. No obstante, al tratar a los componentes de forma abstracta se pueden incorporar otras tecnologías, como es el caso de la telefonía móvil 5G o protocolos propietarios como SigFox. A continuación, se describen cada una de las metaclases que componen el metamodelo:

— `MessageBroker`: Modela el intermediario de los mensajes provenientes de los componentes hardware de una Arquitectura Dirigida por Eventos (EDA). Este componente se ejecuta en la red y es accesible a través de su host y puerto lógico. Por ejemplo, uno de los más conocidos es Eclipse Mosquitto, el cual acepta mensajes MQTT para tópicos de publicación y suscripción.

— `Bridge`: Es una metaclase abstracta que modela un componte creado para la integración de una arquitectura EDA con una arquitectura de Microservicios.

— `OutputBridge`: Especializa la función de `Bridge`, para traducir los mensajes enrutados por el `MessageBroker` provenientes de los sensores a consultas de los servicios web de la capa de Control. Por ejemplo, considere un artefacto creado en Node-Red, el cual al inicio del flujo tiene un nodo MQTT suscrito a un tópico, que al recibir la notificación del Messages Broker que existe una nueva publicación de un sensor de temperatura, toma esa información y en su nodo final realiza una consulta con un método POST a un servicio web.

— `InputBridge`: Especializa la función de `Bridge`, que a diferencia de `OutputBridge` trabaja con los actuadores para realizar las consultas de los servicios web de la capa de Control. Por ejemplo, considere un artefacto creado en Node-Red, el cual al inicio del flujo tiene un nodo de servicio web con un método POST, que al recibir una consulta con información del estado de encendido o apagado de una bombilla, publica esa información en un tópico MQTT del relé asociado al actuador.

— `Server`: Es una metaclase abstracta que define un servidor.

— `WebServer`: Especializa la función de `Server` para modelar un servidor de aplicaciones web, que es empleado para el despliegue de los servicios generados en la capa de Control. Por ejemplo, considere un servidor virtualizado en la nube de Google, en el cual se pueden desplegar los servicios REST de los sensores y actuadores de la aplicación IoT.

— `DBServer`: Especializa la función de `Server` para modelar un servidor de base de datos, que es empleado por los servicios de la capa de Control, para proveer almacenamiento de la información generada por los objetos inteligentes. Por ejemplo, considere un servidor de base de datos como MySql que se ejecuta en el servidor virtualizado antes mencionado, donde se almacenará la información de los sensores y actuadores de una aplicación IoT.

— `BridgeServer`: Especializa la función de `Server` para modelar el servidor en el cual se ejecutan los componentes `OutputBridge` o `InputBridge`. Por ejemplo, considere una instancia de Node-Red desplegada en el servidor virtual, la cual ejecuta los flujos de un componente `OutputBridge`, de un sensor de temperatura conectado a un controlador, el cual envía por MQTT los valores medidos por el sensor a un servicio REST.

— `AccesPoint`: Modela las características del punto de acceso a Internet al cual se conectan los objetos inteligentes. Para conectarse los objetos deben conocer el SSID y si es necesario la contraseña. Por ejemplo, considere que se están desplegando varios sensores en su hogar, los cuales deben conectarse a la red local para poder acceder a Internet. Se deberá configurar en el controlador las credenciales para conectarse a la red.

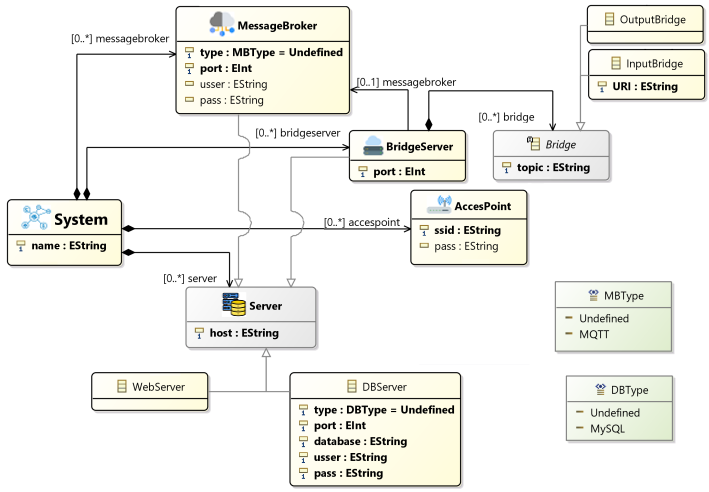


Figura 4.9: Fragmento del Metamodelo con las metaclases del nivel de Infraestructura.

4.3.1.2. Nivel de Hardware

En este apartado se describe el metamodelo utilizado para el nivel de Hardware de la capa Física. El metamodelo de este nivel se muestra en la Figura 4.10, el cual define la estructura de un objeto inteligente que se puede construir y adaptar a diferentes aplicaciones, como puede ser la industria, el hogar, la agricultura, el transporte, etc. Los objetos tienen dos componentes fundamentales: un transductor y un controlador. El primero es un dispositivo capaz de transformar una determinada manifestación de energía de entrada en otra diferente de salida, pudiendo ser un sensor o un actuador. El segundo es un circuito integrado programable, compuesto de varios bloques funcionales, entradas y salidas digitales o analógicas y mecanismos de comunicación. Una de las plataformas más común en las aplicaciones de IoT es el controlador Node MCU, el cual

es compatible con Arduino e incorpora un módulo WiFi. A continuación, se describen cada una de las metaclases que componen el metamodelo:

— `IoTNode`: Modela el concepto de un nodo IoT como una unidad que contiene el transductor y el controlador. Por ejemplo, considere la analogía de un *protoboard*, en el cual se montan todos los componentes y se conectan entre sí. De forma similar `IoTNode` es el contenedor donde se montan y conectan los componentes de forma virtual.

— `Device`: Es una metaclase abstracta que define un dispositivo electrónico.

— `Controller`: Especializa la metaclase `Device`, la cual describe un controlador encargado de acondicionar la información de los sensores y actuadores. Además, se encarga de gestionar los protocolos de comunicación del controlador con otros dispositivos. Por ejemplo, considere el controlador Node MCU, el cual tiene puertos de entrada y salidas así como comunicación WiFi, y que pueden ser programados de forma similar a un Arduino.

— `Communication`: Modela los protocolos de comunicación que dispone el controlador. Los cuales pueden ser cableados o inalámbricos. Por ejemplo, uno de los protocolos de comunicación de red más difundidos en los hogares es el WiFi, el cual se encuentra embebido en tarjetas de desarrollo como Raspberry Pi o Node MCU.

— `PropertyComm`: Modela las características particulares del protocolo de comunicación. Por ejemplo, considere la comunicación serial, la cual necesita el nombre del puerto físico (COMM) que se emplea o en el caso del WiFi el canal en el que opera la red.

— `Port`: Modela el concepto de un puerto del controlador.

— `InputPort`: Especializa la metaclase `Port`, la cual describe un puerto de entrada del controlador que puede ser analógico o digital y que por lo general es al que se conectan los sensores. Por ejemplo, considere una tarjeta Node MCU que tiene 12 puertos digitales y 1 puerto analógico para entrada de información.

— `OutputPort`: Especializa la metaclase `Port`, la cual describe un puerto de salida del controlador que puede ser analógico o digital y que por lo general es al que se conectan los actuadores. Por ejemplo, considere una tarjeta Node MCU que tiene 12 puertos digitales para salida de información. Sin embargo, aun cuando dispone de un puerto analógico este no puede ser empleado para salida de información.

— `Sensor`: Especializa la metaclase `Device`, la cual describe un sensor que entrega información de su entorno en formato analógico o digital al controlador. Por ejemplo, considere un sensor de temperatura, el cual produce una diferencia de potencial eléctrico en función de la temperatura ambiente. Esta información es recibida por un puerto analógico del controlador y en un módulo interno ADC (Analog to Digital Converter) convierte esta variación de tensión en un número binario, el cual puede ser procesado.

— `Actuator`: Especializa la metaclase `Device`, la cual describe un actuador que transforma la información generada por el controlador en una magnitud física. Por ejemplo, un relé es un actuador de tipo digital, el cual requiere de una señal binaria para controlar el cierre o la apertura de los contactos de la etapa de potencia, y deberá conectarse a un puerto digital de salida del controlador.

— `DeviceData`: Modela el formato con el cual se estructura la información que emplean el controlador y el resto de los componentes de la aplicación IoT para comunicarse.

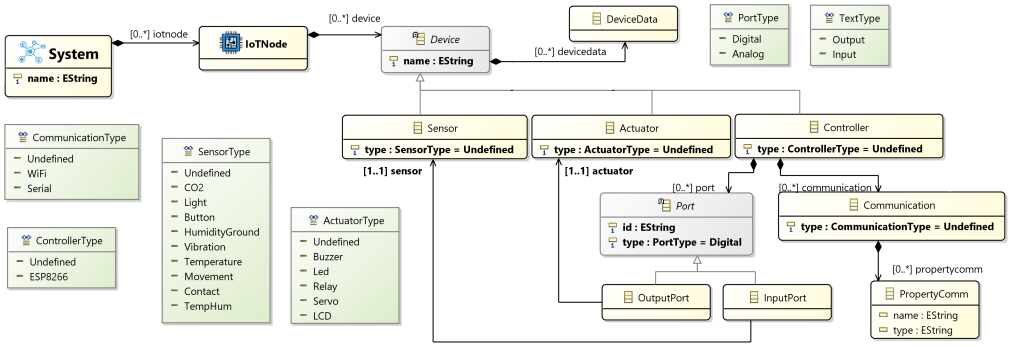


Figura 4.10: Fragmento del Metamodelo con las metaclases del nivel de Hardware.

4.3.1.3. Nivel de Control

En este apartado se describe el metamodelo utilizado para el nivel de Control de la capa de Control. El metamodelo de este nivel se muestra en la Figura 4.12, el cual define el proceso de integración de los componentes de un sistema IoT. Se ha tomado como base la arquitectura de la WoT y Thing Description, con lo cual se crean servicios REST para cada uno de los componentes de hardware, lo que hace posible emplear tecnologías web para su integración. En este caso se empleó el patrón Saga por medio de orquestación de servicios para establecer la lógica de negocio del sistema, la cual se implementa en dos formas: (a) por medio de funciones lógicas de álgebra booleana, en el caso de querer controlar información de sensores y actuadores digitales y (b) por medio del condicional IF para el control de señales analógicas. Un ejemplo que puede servir para ilustrar la funcionalidad de este nivel es un semáforo inteligente (ver Figura 4.11) el cual dispone de sensores de movimiento para personas en las aceras y para vehículos en la calle. Con la información binaria que generan estos sensores se coordina el tiempo que deben funcionar las luces del semáforo ya que de no existir tráfico en un sentido puede mantener un fluido más rápido de la vía con más vehículos. Además, en este escenario se puede ver que cuando se incrementa el número de semáforos también se incrementa la complejidad del procesamiento de eventos. De esta manera se puede ver que la propuesta es una solución viable con el uso de una coreografía de orquestadores. A continuación, se describen cada una de las metaclases que componen el metamodelo:

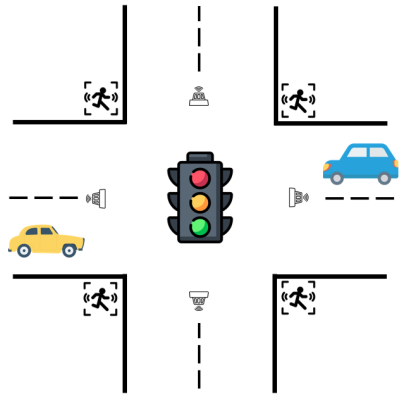


Figura 4.11: Escenario de un semáforo inteligente.

— `WebService`: Modela el concepto de los servicios web que están relacionados a cada uno de los sensores y actuadores de la aplicación.

— `REST`: Modela la implementación de los servicios web con la arquitectura REST e implementa los métodos GET, POST, PUT y DELETE. Por ejemplo, un servicio programado en Spring Boot, el cual permite consultar la información de un sensor almacenada en una base de datos.

— `IntegrationPattern`: Modela el concepto del patrón Saga para la integración de los servicios web de la aplicación

— `Orchestrator`: Modela la implementación de la lógica de negocio con el patrón Saga, basado en la Orquestación de los servicios web. Por ejemplo, considere el caso del semáforo antes mencionado donde cada sensor de movimiento y actuador (luces del semáforo) tienen asociados un servicios REST, los cuales son llamados por el orquestador en función de la lógica de negocio.

— `InputOrchestrator`: Complementa la definición de `Orchestrator` al modelar el End-Point, con el cual los servicios web de los sensores pueden iniciar el patrón de integración con la lógica de negocio. Por ejemplo, considere un sensor de movimiento que envía la información de la temperatura a un orquestador por medio de un `OutputBridge`.

— `OutputOrchestrator`: Complementa la definición de `Orchestrator` al modelar el mecanismo para la ejecución de los servicios web de los actuadores. Por ejemplo, considere un relé que controla el encendido o apagado de una bombilla, a partir de la información resultante del procesamiento de la información de un sensor de movimiento en un orquestador, la cual se comunica por medio de un `InputBridge` al relé de la bombilla.

— `Operation`: Es una metaclase abstracta que modela la lógica de negocio para los componentes de hardware empleando información binaria.

— `Function`: Especializa la metaclase `Operation`, al describir la función lógica expresada en álgebra Booleana que determina la lógica de control de los actuadores en función de la información de los sensores. Por ejemplo, considere el caso del semáforo antes mencionado en el cual todas sus entradas y salidas son binarias. En este escenario, aunque se puede implementar la lógica de negocio a partir de estructuras de control, cuando el número de variables crece, como cuando se controla múltiples semáforos, se vuelve demasiado complejo y susceptible de errores si se emplea estas estructuras. Es por eso que el álgebra Booleana empleada en la electrónica digital reduce a una expresión algebraica el control de todo el sistema.

— `Status`: Especializa la metaclase `Operation`, la cual describe el complemento de la función lógica, con el fin de enviar información a los actuadores si existe un cambio en el estado, como resultado de procesar la función lógica. Por ejemplo, considere el mismo caso del semáforo en el cual se está controlando el sentido de la vía que permita más trafico y se detecta movimiento de peatones que cruzan de una calzada a otra en el mismo sentido del tráfico. En este escenario la función `Status` aun cuando recibe información de los sensores de las personas, no debe cambiar el estado del semáforo, ya que el estado actual es el deseado y es irrelevante enviar nuevamente la señal para ejecutar ese estado.

— `Condition`: Modela el segundo esquema para implementar la lógica de negocio basada en un condicional `If`. Esta es empleada para la información proveniente de sensores analógicos, que no puede ser procesada con álgebra Booleana, ya que sus valores son continuos y no discretos.

— `Structure`: Es una metaclase abstracta que modela la estructura de la sentencia condicional, para la implementación de la lógica de control.

— `Conditional`: Especializa la metaclase `Structure`, la cual clasifica las posibles estructuras condicionales que se pueden implementar.

— `If`: Especializa la metaclase `Conditional`, la cual describe la estructura condicional `If`. Esto posibilita implementar condicionales especialmente cuando se trata de valores distintos a un sistema binario. Por ejemplo, considere un sensor de temperatura que monitoriza la temperatura ambiental, el cual está asociado al control de encendido o apagado de un calefactor cuando se alcanza una temperatura mínima.

— `InputIf`: Esta metaclase complementa la definición de la estructura `If`, para definir la información de entrada que serán analizadas. Por ejemplo, en el caso anterior son la información del sensor de temperatura y el valor de la temperatura deseada para activar el calefactor.

— `OutputIf`: Esta metaclase complementa la definición de la estructura `If`, para definir la información que resulta de la evaluación de la condición. Por ejemplo, en el caso anterior es el valor de activación o de encendido para el calefactor.

— `Else`: Especializa la metaclase `OutputIf`, la cual describe el resultado cuando es verdadero la evaluación de la condición.

— `Then`: Especializa la metaclase `OutputIf`, la cual describe el resultado cuando es falso la evaluación de la condición.

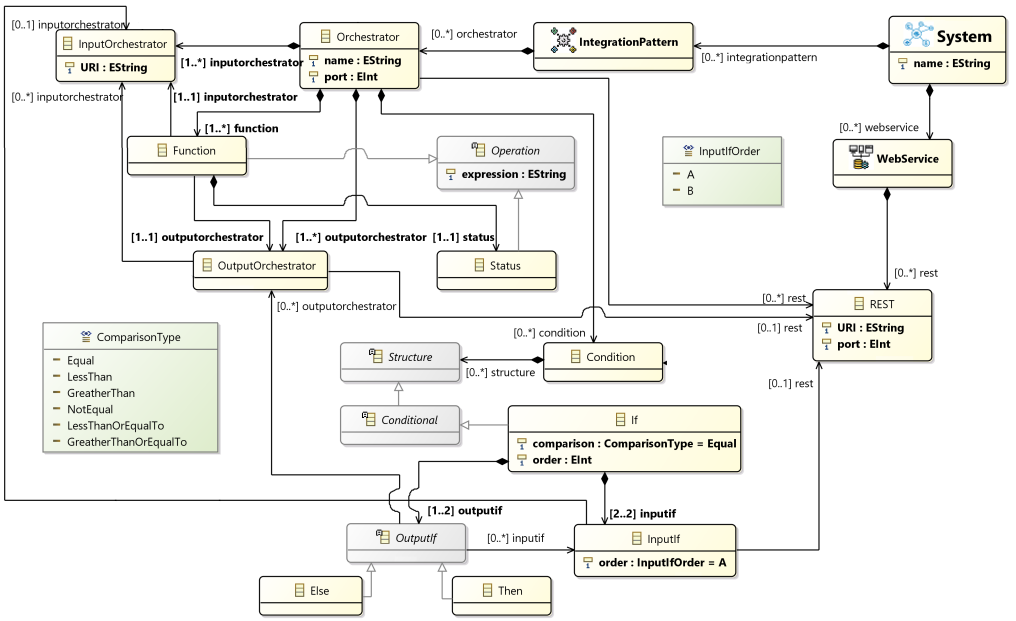


Figura 4.12: Fragmento del Metamodelo con las metaclases del nivel de Control.

4.3.1.4. Nivel de DTV

En este apartado se describe el metamodelo utilizado para el nivel de DTV de la capa de Aplicación. El metamodelo de este nivel se muestra en la Figura 4.14, el cual define el proceso de creación de la interfaz de usuario para DTV, con capacidades de interactividad con los componentes de hardware de una aplicación IoT. Se ha tomado como base la Televisión Digital Terrestre con el estándar ISDB-T, con lo cual se define una interfaz con componentes multimedia (imagen y vídeo), componentes de interacción (botones y campos de texto), y consultas a servicios web. Para el caso de los botones se considera los cuatro botones representativos del mando de DTV (Rojo, Verde, Amarillo y Azul) que sirven para ejecutar los eventos para cargar los recursos multimedia o para realizar consultas a los servicios web del sistema IoT. Para ilustrar este funcionamiento considere la Figura 4.13 en la que el mando a distancia al ser presionado el botón Azul realiza una consulta al servicio REST de un sensor de temperatura, donde se retorna el valor de la temperatura en grados para que se visualice en un campo de texto en la pantalla. A continuación, se describen cada una de las metaclases que componen el metamodelo:

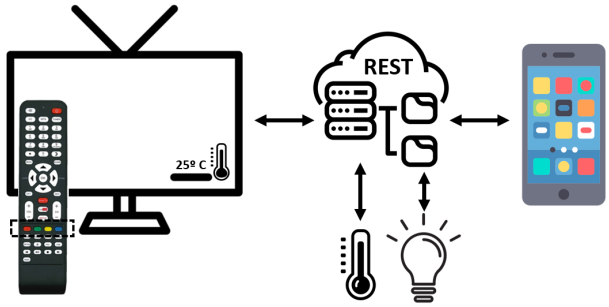


Figura 4.13: Escenario IoT con una interfaz DTV y móvil.

— `DTV`: Esta metaclase describe la interfaz de usuario de una aplicación para DTV.

— `Interface`: Es una metaclase abstracta que modela la descripción de la interfaz de la aplicación y sus componentes.

— `Banner`: Especializa la metaclase `Interface`, la cual describe una plantilla que despliega contenido textual asociado al evento de presionar los botones.

— `Window`: Es una metaclase abstracta que describe una plantilla en la que se despliega contenido textual y multimedia asociado al evento de presionar los botones.

— `Frame`: Especializa la metaclase `Window`, la cual describe una plantilla que despliega contenido textual y multimedia asociado al evento de presionar los botones, sobre el video principal ocultando algunas áreas.

— `Accordion`: Especializa la metaclase `Window`, la cual describe una plantilla que despliega contenido textual y multimedia asociado al evento de presionar los botones y que a diferencia de la metaclase `Frame`, reduce el video principal a un 25 % de su tamaño original y deja el resto de la pantalla para visualizar los componentes de la plantilla.

— `Button`: es una metaclase abstracta que modela los 4 botones (Rojo, Verde, Amarillo y Azul) representativos de la DTV. Además, su evento de ejecución está asociado a los contenidos textuales y multimedia.

— `Red`: Especializa la metaclase `Button`, la cual modela el botón Rojo.

— `Green`: Especializa la metaclase `Button`, la cual modela el botón Verde.

— `Yellow`: Especializa la metaclase `Button`, la cual modela el botón Amarillo.

— `Blue`: Especializa la metaclase `Button`, la cual modela el botón Azul.

— `Content`: Es una metaclase abstracta que modela el contenido multimedia que se presenta en la aplicación.

— `Video`: Especializa la metaclase `Content`, la cual describe un vídeo que se puede visualizar en la plantilla `Frame` o `Accordion`.

— `Image`: Especializa la metaclase `Content`, la cual describe una imagen que se puede visualizar en la plantilla `Frame` o `Accordion`.

— `Text`: Especializa la metaclase `Content`, la cual describe un campo de texto que se puede visualizar en la plantilla textttBanner, `Frame` o `Accordion`. En estos componentes se pueden desplegar un texto estático, definido al momento del diseño de la aplicación o puede ser un texto dinámico proveniente de una consulta a un servicio web de la aplicación.

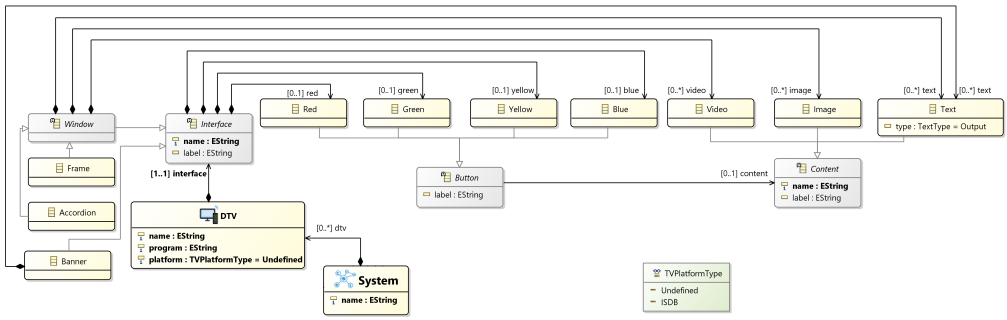


Figura 4.14: Fragmento del Metamodelo con las metaclases del nivel de DTV.

4.3.1.5. Nivel Móvil

En este apartado se describe el metamodelo utilizado para el nivel de Móvil de la capa de Aplicación. El metamodelo de este nivel se muestra en la Figura 4.15, el cual define el proceso de creación de la interfaz de usuario para dispositivos móviles con capacidades de interactividad con los componentes de hardware de una aplicación IoT. Para ilustrar la utilidad este nivel, considere el ejemplo de la Figura 4.13, en el que además de la DTV un usuario puede controlar los sensores y actuadores de una aplicación IoT por medio de una aplicación móvil, al consultar los servicios REST de los componentes de hardware del sistema. De esta forma se le permite al usuario tener una interfaz que puede consultar en cualquier lugar con conectividad a internet. A continuación, se describen cada una de las metaclases que componen el metamodelo:

— `Mobile`: Esta metaclase describe la interfaz de usuario de una aplicación para dispositivos móviles.

— `Activity`: Esta metaclase agrupa todos los componentes de interactividad (botones y campos de texto) que se despliegan en la interfaz de la aplicación.

— `View`: Esta metaclase representa una de las múltiples vistas en la que se pueden organizar los componentes de la aplicación. Por ejemplo, considere una aplicación móvil con la información de múltiples sensores y actuadores, que se agrupan en vistas diferentes de la aplicación, con el propósito de organizar la información.

— `ContentM`: Es una metaclase abstracta que describe los componentes de interactividad que se pueden desplegar en la aplicación.

— `ButtonM`: Especializa de `ContentM` para describir los botones de la aplicación.

— `TextView`: Se usa para describir los campos de texto de aplicación.

— `Order`: Esta metaclase que complementa las características de `ButtonM` y `TextView` al permitir indicar la posición en la cual aparecen en el `View` de la aplicación. Por ejemplo considere una aplicación móvil en la cual los componentes se van ubicando uno a continuación de otro en función del orden que se les asigna.

— `Connection`: Modela los End Point de los servicios web del sistema IoT a los cuales la aplicación puede realizar consultas. Su ejecución depende del evento del botón y la respuesta de la consulta se visualiza en un campo de texto específico. Por ejemplo, considere la Figura 4.14 en la cual un botón de la aplicación móvil pude cambiar el estado de la bombilla al realizar una consulta con el nuevo estado (on/off) escrito por el usuario en un campo de texto.

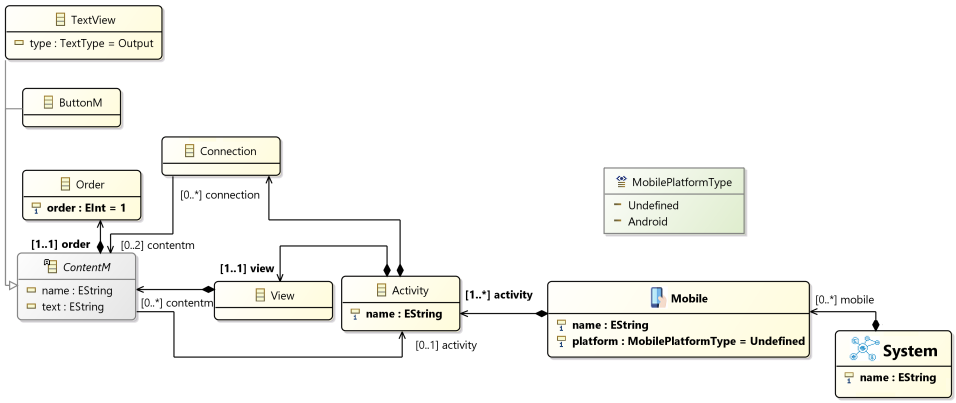


Figura 4.15: Fragmento del Metamodelo con las metaclases del nivel Móvil.

4.3.2. Sintaxis Concreta del Lenguaje

El desarrollo y masificación del Internet, junto con la reducción de costos y mejor rendimiento del hardware, ha impulsado el desarrollo del IoT. Este desarrollo ha permitido el surgimiento de tecnologías con las cuales los desarrolladores novatos puedan crear rápidamente aplicaciones, pero el escenario se complica cuando la aplicación tiene: (a) gran

cantidad de objetos inteligentes, (b) incorpora procesamiento de eventos o (c) permite la interacción de las personas por múltiples interfaces. Es ahí cuando un desarrollador novato encuentra una barrera muy difícil de sobre pasar, ya que debe conocer diversas tecnologías a un nivel más elevado.

Por este motivo, se describe una herramienta para el modelado de aplicaciones IoT, la cual se basa en tecnologías MDE, que permiten desarrollar modelos de las aplicaciones IoT sin la necesidad de abordar los detalles técnicos específicos de las plataformas. Por esta razón se ha propuesto un DSL que permite la construcción de aplicaciones IoT por medio de un editor gráfico, el cual es más rápido de aprende a usar en comparación que la sintaxis de un editor textual. Esto se consideró debido a que los posibles usuarios podrían ser de distintos ámbitos de la ingeniería, como las telecomunicaciones, electrónica, informática, entre otras similares. Además, el editor gráfico al momento de configurar los componentes visuales solo debería requerir información específica al despliegue de la aplicación. Evitando así que el desarrollador deba ingresar de forma textual más detalles del comportamiento del sistema ya que estos son definidos por medio de las conexiones entre los componentes. Por último, el editor gráfico es capaz de crear los artefactos de software necesarios para desplegar la aplicación de acuerdo con el modelo, sin que requiera ninguna línea de código adicional.

En la especificación del editor gráfico se empleó Eclipse Sirius [Sirius, 2020], para lo cual en la Figura 4.16, se presentan las etapas que se siguieron para la construcción del editor en el nivel de Infraestructura, además se ilustra la relación que tienen cada una de estas configuraciones con el metamodelo y con la paleta resultante que desplegará el editor. A continuación, se describen cada una de las etapas:

(a) Creación de un Proyecto de Especificación. Una vez creado el metamodelo del DSL se debe ejecutar una nueva instancia de Eclipse en la cual se crea un proyecto de tipo "Viewpoint Specification Project" para realizar la configuración del editor gráfico.

(b) Especificación de un VSM (Viewpoint Specification Model). En esta etapa se especifica el nombre del metamodelo al cual está asociado el editor gráfico.

(c) Especificación del tipo de representación. Una vez especificado el metamodelo se selecciona el tipo de representación que emplea el editor gráfico para los componentes del metamodelo. En este caso se optó por el tipo "Diagram Description" con el propósito de que sea más visual las interconexiones entre los componentes. Además, se especifica la metaclase raíz del metamodelo.

(d) Mapeo entre los componentes gráficos del diagrama con los componentes del metamodelo. En esta etapa se asigna a cada metaclase y relación un "Diagram Element" que corresponde a la representación que tendrá la metaclase en el editor.

(e) Especificación de los componentes de la barra de herramientas del editor (Paleta). Por último, se establece las características que tendrán cada componente en la paleta del editor como es la creación, ingreso de parámetros de configuración y eliminación.

Una vez terminada la especificación del editor gráfico, en la Figura 4.17 muestra el resultado con un ejemplo de aplicación IoT. Se han especificado: dos controladores, cada

uno con un sensor y un actuador; una aplicación de DTV, para la lectura del sensor y la activación del actuador; una aplicación móvil que permite consultar el valor del sensor y enviar un nuevo estado para el controlador; los parámetros de la red WiFi, del servidor de base de datos, del servidor web y del Broker MQTT; y por último se establece la lógica de negocio de la aplicación para la integración del sensor de temperatura analógico con el actuador que controla un ventilador, que se activa una vez que se sobrepase un umbral determinado.

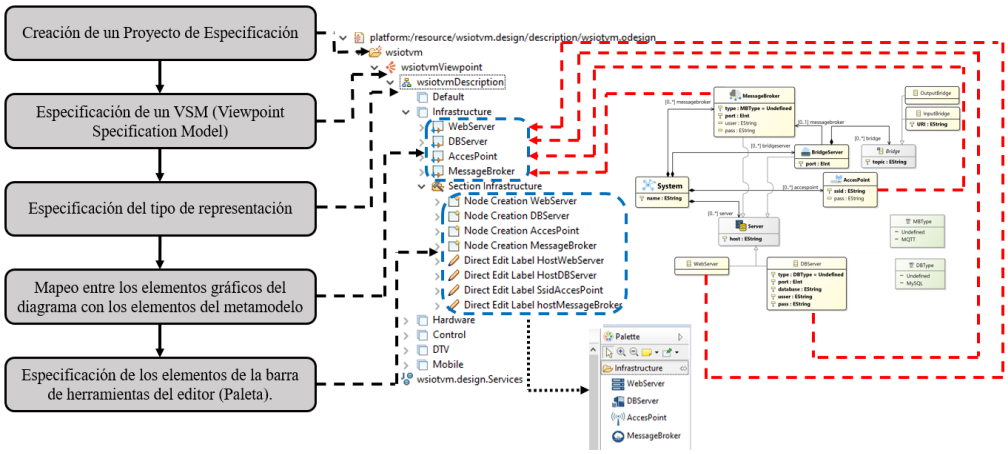


Figura 4.16: Captura de pantalla del VSM del editor gráfico

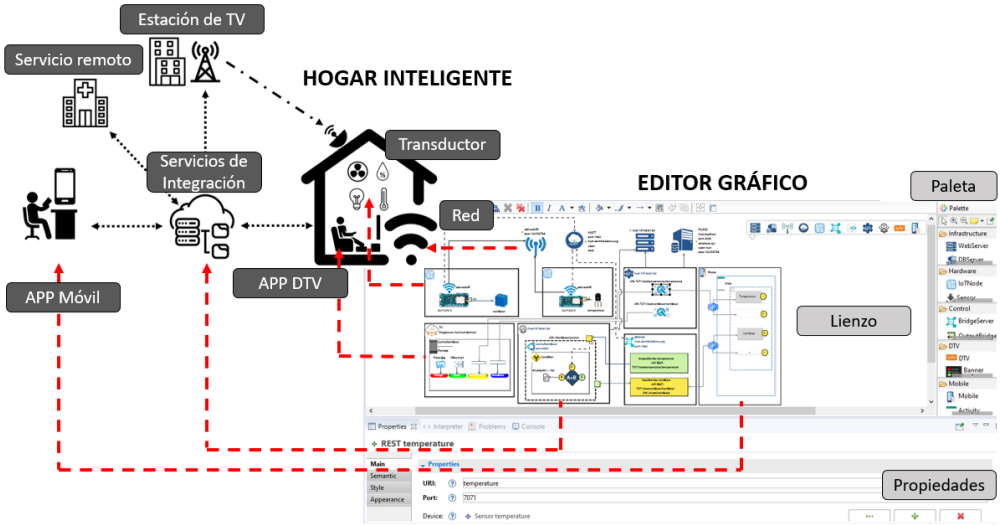


Figura 4.17: Captura de pantalla del Editor Gráfico.

Además, en la misma figura (Figura 4.17) se puede ver que el editor gráfico se divide en 3 áreas principales que se describen a continuación:

— Paleta de herramientas (panel derecho) permite modelar la Infraestructura de Red, el hardware de los nodos IoT, los patrones de integración de los servicios y la interfaz de control para las interfaces Móvil y DTV. La herramienta dispone de iconos para los componentes que el desarrollador los puede arrastrar al Lienzo.

— Lienzo (panel central) permite agregar de forma visual las instancias de los componentes de los sistemas IoT y definir sus relaciones con los demás componentes. En el lienzo, el desarrollador puede mover y ubicar los componentes con el fin de diseñar el modelo de la aplicación IoT.

— Propiedades (panel inferior) permite especificar características adicionales para los componentes de la aplicación de acuerdo a los atributos de las metaclases y sus relaciones con otras metaclases.

4.3.2.1. Nivel de Infraestructura

El nivel de Infraestructura se muestra en la Figura 4.18, el cual permite al desarrollador incluir en el modelo de una aplicación IoT, los nodos `WebServer`, `DBServer`, `AccesPoint`, y `MessageBrober`, los cuales se corresponden a las metaclases con mismo nombre del metamodelo. En el diseño del editor se define el estilo para estos nodos con una imagen que corresponde al concepto del componente, la cual aparecerá sobre el lienzo y en la Paleta de herramientas. Además, se definen las acciones específicas que tendrán los nodos en el editor gráfico, en este caso la creación de los nodos y la edición de sus parámetros más representativos.

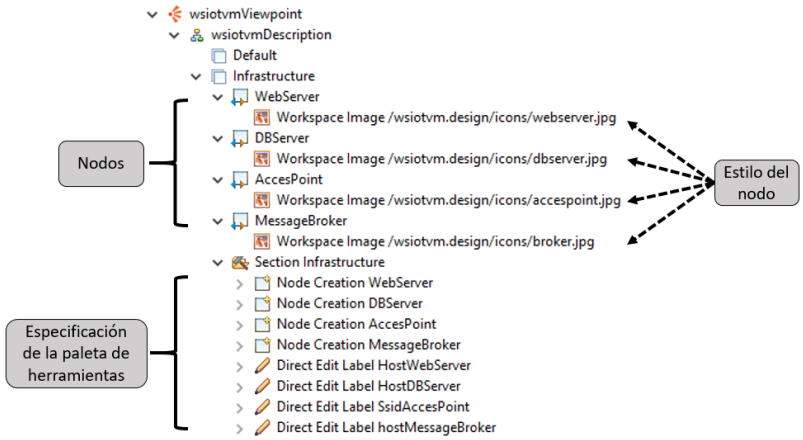


Figura 4.18: Captura de pantalla de la sección del VSM del nivel de Infraestructura.

4.3.2.2. Nivel de Hardware

Permite al desarrollador incluir en el modelo los componentes para la construcción de los nodos de hardware de la aplicación IoT (Figura 4.19). Para el diseño de este nivel en el editor gráfico se crea un contenedor `IoTNode` que agrupa los nodos `Sensor`, `Actuator` y `Controller`, junto con las relaciones que tienen estos componentes. En el caso de los nodos `Sensor` y `Actuator`, se han definido estilos condicionales para cada tipo de sensor y actuador que permite utilizar el editor. En el caso del nodo `Controller`, además, se han definido los nodos de borde `OutputPort`, `InputPort` y `Communication`, que permiten especificar el modo de operación del controlador al definir si los puertos son de entrada o salida, y si son digitales o analógicos. A estos puertos se conectan los sensores y actuadores de la aplicación IoT. `Communication` establece el protocolo de comunicación y especifica la conexión a un punto de acceso a Internet. Al final se definen las acciones específicas que tendrán los nodos y el contenedor en el editor gráfico, en este caso la creación de los nodos y la edición de sus parámetros más representativos. Las relaciones que tienen los componentes hardware con otros de los niveles del DSL, se han establecido por medio de recursos "Relation Based Edge" de Sirius. Para la relación `Communication` y `AccesPoint`, se ha creado `Communication2AccesPoint`, que describe la conexión del controlador a la red WiFi. El segundo caso de conexión considerado es entre `Communication` y `MessageBroker`, la cual especifica la conexión que tiene el controlador con el Broker MQTT. Para los componentes `InputPort` y `OutputPort` se creó `In2Sensor` y `Out2Actuator`, las cuales permiten especificar conexión entre los puertos de entrada con los sensores y los puertos de salida con los actuadores respectivamente.

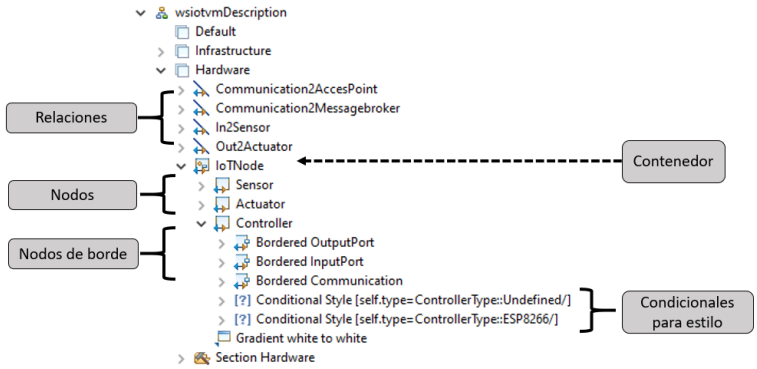


Figura 4.19: Captura de pantalla de la sección del VSM del nivel de Hardware.

4.3.2.3. Nivel de Control

El nivel de Control se muestra en la Figura 4.20, el cual permite al desarrollador incluir en el modelo todos los componentes para la lógica de negocio de la aplicación IoT. El contenedor `WebService` agrupa el nodo `REST`, con el cual se definen los servicios REST que permite consultar y almacenar la información de cada uno de los sensores y actua-

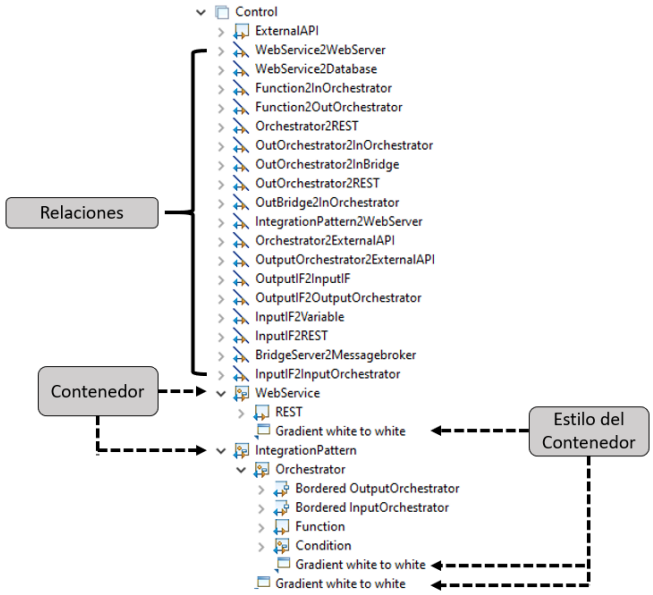


Figura 4.20: Captura de pantalla de la sección del VSM del nivel de Control.

dores. El contenedor `IntegrationPattern` contiene `Orchestrator`, el cual permite por medio de `Function`, para información binaria y `Condition`, para información decimal, establecer la lógica de negocio de la aplicación. También se definen las acciones específicas que tendrán los nodos y los contenedores en el editor gráfico, en este caso la creación de los nodos, contenedores, establecimiento de relaciones y la edición de sus parámetros más representativos.

Las relaciones que tienen los componentes del nivel de Control con otros de los niveles del DSL, se han establecido por medio de relaciones. Este nivel es el que tiene la mayor cantidad de relaciones, ya que es el encargado de coordinar los recursos del sistema. Existen 17 relaciones que se describen a continuación:

— `WebService2WebServer`: Establece las características que tiene el servidor de aplicaciones en el cual se ejecutan los servicios REST. Estas características se incorporan en el código cuando se generan los artefactos de software.

— `WebService2DataBase`: Establece en el servicio web REST cuál es la base de datos que emplea para gestionar la información proveniente de los sensores y actuadores.

— `Funtion2InOrchestrator`: Esta relación pasa la información recibida en la entrada del orquestador a la función lógica para que se ejecute la operación Booleana. La función evalúa las entradas/salidas que definen la lógica de negocio de la aplicación.

— `Fuction2Outorchestrtor`: Esta relación pasa la información resultante de evaluar la función lógica a la salida del orquestador para que pueda ser enviada al exterior.

— `Orchestrator2REST`: Esta relación le permite al orquestador realizar las consultas a los servicios REST de los sensores o actuadores, para que la información recuperada sea evaluada.

— `OutOrchestrator2InOrchestrator`: Relación que permite pasar la información resultante de evaluar los estados de los actuadores y sensores a la entrada de otro orquestador. Esta funcionalidad permite diseñar modelos de sistemas IoT más complejos al integrar varios orquestadores por medio de coreografías.

— `OutOrchestrtor2InBridge`: Relación que permite pasar la información resultante de evaluar los estados de los actuadores y sensores a la entrada de un `Bridge`. Esta funcionalidad permite publicar la información en el tópico del actuador que controla el orquestador.

— `OutBridge2InOrchestrator`: Relación que permite pasar la información proveniente del `Bridge` de un sensor a la entrada de un orquestador, con el fin de que la información sea evaluada.

— `IntegrationPattern2WebServer`: Establece las características que tiene el servidor de aplicaciones donde se ejecutan los servicios de orquestación. Estas características se incorporan en el código cuando se generan los artefactos software.

— `Orchestrator2ExternalAPI`: Esta relación le permite al orquestador realizar las consultas a los servicios remotos, para que la información recuperada sea evaluada.

— `OutputOrchestor2ExternalAPI`: Relación que permite pasar la información resultante de evaluar los estados de los actuadores y sensores a un servicio remoto.

— `OutputIF2InputIF`: Relación que permite pasar la información resultante de evaluar la condición IF a otra función IF. Esta funcionalidad permite implementar condiciones más complejas.

— `OutputIF2OutputOrchestrator`: Relación de la salida de la función IF con la salida de información del orquestador. Relación que permite pasar la información resultante de evaluar la condición IF a la salida del orquestador. Esta información se podrá enviar a un `Bridge`, un servicio REST o incluso a otro orquestador.

— `InputIF2Variable`: Relación entre la entrada de información de la función IF con una variable de comparación. Esta relación le permite al condicional tener en una de sus entradas un valor fijo para su evaluación.

— `InputIF2REST`: Entrada de información de la función IF con la información proveniente de un servicio REST. Esta relación le permite al condicional tener en una de sus entradas un valor variable proveniente de un sensor o de un actuador.

— `BridgeServer2MessageBroker`: Especifica el Broker de mensajes donde se encuentran los tópicos que emplean los `Bridge` de los sensores y actuadores.

— `InputIF2InputOrchestrator`: Relación entre la entrada de información de la función IF con la información proveniente del orquestador. Esta información es la que inicia el orquestador para que ejecute la lógica de negocio de la aplicación.

4.3.2.4. Nivel de DTV

El nivel de DTV se muestra en la Figura 4.21, el cual permite incluir en el modelo los componentes para la construcción de una interfaz para DTV. El contenedor `DTV`, representa la aplicación de DTV en su conjunto, la cual puede tener tres posibles plantillas `Banner`, `Accordion` y `Frame`. La primera es una plantilla que solo admite los compoenentes `Button` y `Text`, mientras que las otras dos plantillas incorporan recursos multimedia `Image` y `Video` con un diseño diferente. Al final se definen las acciones específicas que tendrán los nodos y el contendor en el editor gráfico, en este caso la creación de los nodos, establecimiento de relaciones y la edición de sus parámetros más representativos, y se muestran las relaciones que tienen estos elementos con otros del DSL. En este nivel existen 14 relaciones que se describen a continuación:

— Las relaciones `TextBanner2InputBridge`, `TextFrame2InputBridge` y `TextAccordion2InputBridge` permiten indicar el `Bridge` de un actuador al cual se desee enviar nueva información para cambiar su estado de encendido o apagado desde los capos de texto de las plantillas `Banner`, `Frame` y `Accordion`. Estas relaciones se reflejan en el código generado en Lua con los end-point y el método POST para realizar la consulta al `Bridge`.

— Las relaciones `ButtonBanner2Text`, `ButtonFrame2Text` y`ButtonAccordion2Text` permiten establecer la relación que tienen los eventos de los botones del mando a distancia para desplegar en pantalla mensaje preestablecidos en las plantillas `Banner`, `Frame` y `Accordion`.

— Las relaciones `TextBanner2Rest`, `TextFrame2Rest` y `TextAccordion2Rest` permiten establecer el servicio REST de un sesnor o actuador del cual se requiera realizar una consulta desde los campos de texto de las plantillas `Banner`, `Frame` y `Accordion`. Estas relaciones se ejecutan cuando tienen lugar los eventos de los botones del mando a distancia (Rojo, Verde, Amarillo y Azul) asociados a los campos de texto correspondientes.

— Las relaciones `ButtonFrame2Video` y `ButtonAccordion2Video` permiten establecer la relación que tienen los eventos de los botones del mando a distancia (Rojo, Verde, Amarillo y Azul) para desplegar en pantalla vídeos secundarios preestablecidos en las plantillas `Frame` y `Accordion`. Esta opción no está disponible para la plantilla `Banner` ya que en diseño no se contempla el despliegue de vídeos.

— Las relaciones `ButtonFrame2Image` y `ButtonAccordion2Image` permiten establecer la relación que tienen los eventos de los botones del mando a distancia (Rojo, Verde, Amarillo y Azul) para desplegar en pantalla imágenes preestablecidas en las plantillas `Frame` y `Accordion`. Esta opción no está disponible para la plantilla `Banner` ya que en diseño no se contempla el despliegue de imágenes.

— `Text2ExternalAPI`, permite establecer la consulta a un servicio remoto desde los campos de texto de las plantillas `Banner`, `Frame` y `Accordion`. Esta relación se ejecuta cuando ocurren los eventos de los botones (Rojo, Verde, Amarillo y Azul) del mando a distancia asociados a los campos de texto.

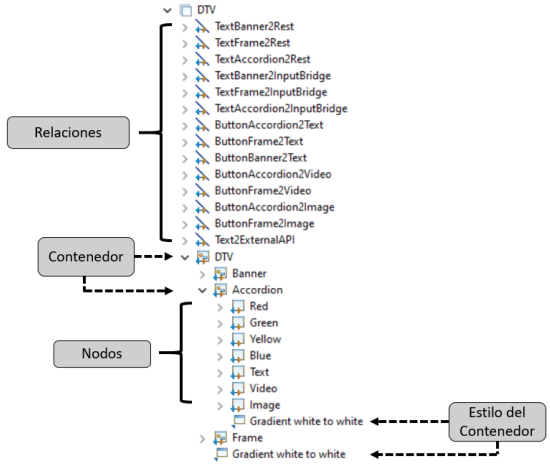


Figura 4.21: Captura de pantalla de la sección del VSM del nivel de DTV.

4.3.2.5. Nivel Móvil

El nivel Móvil se muestra en la Figura 4.22, el cual permite incluir en el modelo los componentes para la construcción de una interfaz para dispositivos móviles. El contenedor `Mobile`, representa una aplicación móvil, la cual está compuesta por una o múltiples vistas o pantallas definidas por los contenedores `Activity` y `View`. Además, para diseñar la interfaz de la aplicación se dispone de componentes `Button` y `TextView`, con el propósito de enviar o recibir información de los sensores y actuadores. También se definen las acciones específicas que tendrán los nodos y los contenedores en el editor gráfico, en este caso la creación de los nodos, contenedores, establecimiento de relaciones y la edición de sus parámetros más representativos. Además, se definen las relaciones que tienen estos elementos con otros del DSL:

— `Connection2ContentM`, permite definir la asociación de `TextView` con los eventos de `Button` para ejecutar una acción.

— `Connection2REST`, esta relación complementa a `Connection2ContentM`, al permitir que `TextView` y `Button` realicen la consulta a los servicios REST de los sensores o actuadores.

— `Connection2ExternalAPI`, esta relación a diferencia de `Connection2REST`, permite realizar consultas a servicios remotos.

— `Connection2InputBridge`, esta relación complementa a `Connection2ContentM`, al permitir enviar información al `Bridge` de un actuador para cambiar el estado encendido o apagado.

— `Connection2InputOrchestrator`, esta relación complementa a `Connection2ContentM`, al permitir enviar información a `Orchestrator` para que la información del `TextView` sea empleada en la lógica de negocio de la aplicación.

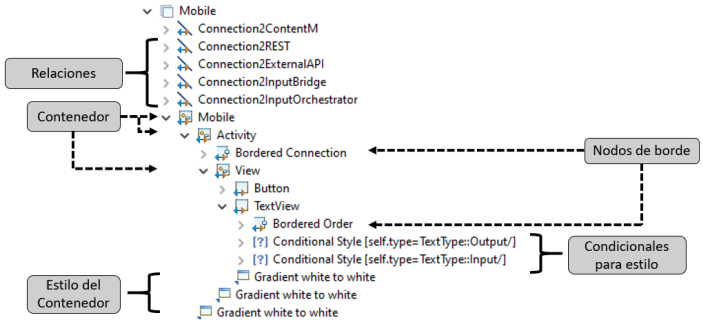


Figura 4.22: Captura de pantalla de la sección del VSM del nivel Móvil.

4.3.3. Transformaciones de Modelos

Para completar el DSL propuesto, en la presente sección se describe un mecanismo de transformación de los modelos creados con el editor gráfico a código funcional. Para lo cual se ha diseñado una transformación M2T (Model-to-Text) implementada en Eclipse Acceleo [Acceleo, 2020]. Es así que en la Figura 4.23 se muestra un fragmento de código Acceleo relacionados con los niveles de la Arquitectura de Integración. Cada uno de los archivos generados corresponde a los niveles de las capas de la arquitectura de integración. Al nivel de Hardware le corresponde el apartado (c), al nivel de Control los apartados (a, b y d), al de móvil Móvil (f) y al nivel de DTV (e).

En el código de la Figura 4.23 se muestra la generación de los archivos específicos para cada plataforma seleccionada, en este caso (a) presenta la creación de los servicios REST de todos los sensores y actuadores del sistema, para lo cual se crea código Ballerina con extensión `.bal`; en el caso de (b) se presenta la creación de todos los `InputBridge` de los sensores y actuadores, para lo cual se crea código Node-Red con extensión `.json`; en el caso de (c) se observa la creación de los archivos de configuración del controlador que gestiona la información de los sensores y actuadores, para lo cual se crea código Arduino con extensión `.ino` que se ejecuta en un Node MCU; para el caso (d) se presenta la creación de los servicios de integración basados en orquestación para la implementación de la Lógica de Negocio, para lo cual se crea código Ballerina con extensión `.bal`; en (e) se presenta la creación de la interfaz de usuario de DTV, para lo cual se crea código Ginga NCL con extensión `.ncl` y `.lua`; y por último (f) presenta la creación de la interfaz de usuario para dispositivos móviles, para lo cual se crea código Android con extensión `.java` y `.xml`. El código completo que se menciona en la Figura 4.23 está disponible en la página web del proyecto[2].

4.3.3.1. Nivel Hardware

En el nivel de Hardware se genera un archivo con código Arduino que ejecuta el controlador Node MCU, el cual se encarga de procesar la información de los sensores y

[2]Framework SI4IOT (Grupo ACG, UAL): `http://acg.ual.es/projects/cosmart/si4iot/`

```
[template public generateElement(aSystem : System)]
[comment API RESTFULL/]
[for (webservice : WebService | aSystem.webservice)]
  [for (rest : REST | aSystem.webservice.rest)]
    [file (rest.device.name.toString().concat('.bal'), false, 'UTF-8')]    ...    [/file]
  [/for]
[/for]
[comment BRIDGE  MQTT/]
[for (broker : BridgeServer | aSystem.bridgeserver)]
  [for (out : OutputBridge | broker.bridge)]
    [file (out.sensor.name.toString().concat('.json'), false, 'UTF-8')]    ...    [/file]
  [/for]
  [for (input : InputBridge | broker.bridge)]
    [file (input.actuator.name.toString().concat('.json'), false, 'UTF-8')]    ...    [/file]
  [/for]
[/for]
[comment HARDWARE/]
[for (nodo : IoTNode | aSystem.iotnode)]
  [for (controller : Controller | nodo.device)]
    [file (controller.name.toString().concat('.ino'), false, 'UTF-8')]    ...    [/file]
  [/for]
[/for]
[comment ORCHESTRATOR/]
[for (integration : IntegrationPattern | aSystem.integrationpattern)]
  [for (orchestrator : Orchestrator | integration.orchestrator)]
    [file (orchestrator.name.toString().concat('.bal'), false, 'UTF-8')]    ...    [/file]
  [/for]
[/for]
[comment TV APP/]
[for (tv : DTV | aSystem.dtv)]
  [file (tv.name.concat('.ncl'), false)]    ...    [/file]
  [file ('tcp.lua', false)]    ...    [/file]
  [file ('ConnectorBase.ncl', false)]    ...    [/file]
[comment MOBILE APP/]
[for (movil : Mobile | aSystem.mobile)]
  [for (actividad : Activity | movil.activity)]
    [file (actividad.name.concat('Activity.java'), false, 'UTF-8')]    ...    [/file]
    [file ('activity_'.concat(actividad.name.toLower().concat('.xml')), false, 'UTF-8')]    ...    [/file]
  [/for]
  [file ('AndroidManifest.xml', false, 'UTF-8')]    ...    [/file]
[/for]
[/template]
```

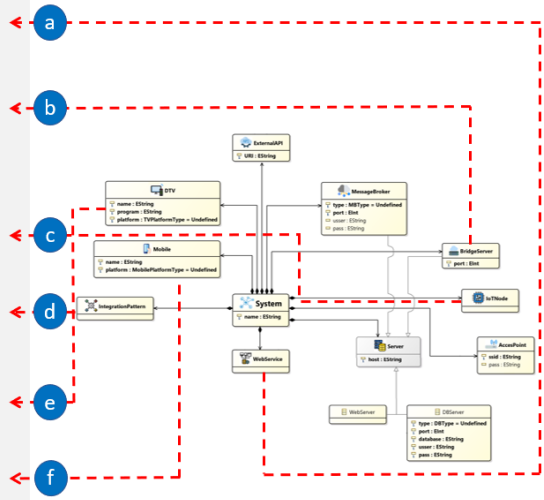


Figura 4.23: Esquema de transformación M2T de los artefactos de software.

actuadores. Para la construcción del citado código se empleó una transformación modelo a texto (M2T) a partir del modelo generado con el editor gráfico mediante el desarrollo de los siguientes pasos: (1) Se realiza la inclusión de las librerías para la comunicación WiFi y MQTT y librerías específicas de los sensores y actuadores; (2) Se lleva a cabo la definición de los pines del controlador Input/Output y Digital/Analog a los cuales se conectarán los sensores y actuadores; (3) Se realiza la definición de las credenciales para la conexión en la red WiFi y con el MessageBroker; (4) Se establece la definición de las variables para contener la información de cada sensor y actuador; (5) Se hace la configuración de la conexión WiFi; (6) Se realiza la configuración de la conexión MQTT; (7) Se hace la lectura de los tópicos con los valores de los actuadores; y (8) se realiza la publicación de la información de los sensores en sus tópicos.

En el Listado 4.1 se muestran los fragmentos de código que ejecuta el controlador, para un sensor de CO2 y un actuador Relé. El primero (mostrado entre las líneas 1 a 14) muestra el proceso de determinación del sensor de CO2 definido en el modelo, para lo cual se crean variables auxiliares que permiten guardar la información proveniente de la lectura del puerto analógico al cual se debe conectar este tipo de sensor. A continuación, se determina si hay cambios en las mediaciones en el sensor, y de ser el caso, se construye un objeto JSON con la información en el formato definido en la arquitectura, la cual es publicada en un tópico MQTT. En el segundo caso (mostrado entre las líneas 15 a 22), el fragmento de código muestra de la función `callBack` que lee los tópicos MQTT que se publican en el MessageBroker. En el caso que exista una publicación para uno de los actuadores se llama a la función `actuators` con la información del objeto JSON

recibido. A continuación, la función `actuators` (mostrada en las líneas 23 a 42) extrae el valor del atributo y el nombre del objeto para determinar a qué actuador le corresponde la información, que en este caso es un Relé. Esta información se almacena en variables auxiliares, y la información del atributo del objeto JSON se escribe en el puerto digital al cual está conectado el actuador.

```
1  void sensors(){
2   [for (sensor : Sensor | nodo.device)]
3    [if (sensor.type.toString()='CO2')]
4     int aux[sensor.name/] = analogRead([sensor.name/]);
5     int PPM[sensor.name/]=(aux[sensor.name/]/10);
6     String auxPPM[sensor.name/] = String(PPM[sensor.name/]);
7     if ( flag[sensor.name/] != auxPPM[sensor.name/]) {
8      flag[sensor.name/] = auxPPM[sensor.name/];
9      String JSON= "{\"date\":\"...\", \"time\":\"...\", \"location\":\"...\", \"attribute
        \":"+auxPPM[sensor.name/]+",\"artefact\":\"[sensor.name/]\"}";
10     client.publish("[sensor.name/]",String(JSON).c_str(),true);
11    }
12   [/if]
13  [/for]
14 }
15 void callback(char* topic, byte* payload, unsigned int length) {
16  String topico = String(topic);
17  String aux;
18  for(int i = 0; i < length; i++) {
19   aux=aux+(char)payload['['/]i[']'/];
20   }
21   actuators(aux);
22 }
23 void actuators(String aux){
24  int aux1 = aux.indexOf("attribute\":", 0);
25  int aux2 = aux.indexOf(",\"ar", 0);
26  String atributo=aux.substring(aux1+11, aux2);
27  int aux11 = aux.indexOf("artefact\":", 0);
28  int aux22 = aux.indexOf("}", 0);
29  String artefacto=aux.substring(aux11+11, aux22-1);
30  [for (actuator : Actuator | nodo.device)]
31   [if (actuator.type.toString()='Relay')]
32    if(artefacto=="[actuator.name/]") {
33     int numero = String(atributo).toInt();
34     if(numero==1) {
35      digitalWrite([actuator.name/],HIGH);
36     } else {
37      digitalWrite([actuator.name/],LOW);
38     }
39    }
40   [/if]
41  [/for]
42 }
```

Listado 4.1: Transformación M2T para la generación de código Arduino para implementar los nodos de IoT.

4.3.3.2. Nivel de Control

En este nivel se generan tres tipos de artefactos software: Servicios REST, Bridge, y Orquestador. Estos implementan la lógica de negocio de la aplicación IoT. Para la construcción del código se empleó una transformación M2T con el modelo generado con el editor gráfico. Para los servicios REST se genera código Ballerina siguiendo los pasos: (1) Inclusión de las librerías para el protocolo HTTP y para las sintaxis SQL; (2) Definición de las credenciales para la conexión en la Base de Datos; (3) Definición del puerto para el servicio REST: (4) Definición de la URI del servicio REST; y (5) Definición de métodos para registrar información (`POST`), buscar por ID (`GET/{id}`), buscar todos los registros (`GET/all`), devolver a último registro (`GET/last`), actualizar un registro por su ID (`PUT/{id}`) y eliminar un registro por su ID (`DELETE/{id}`).

En el Listado 4.2 se muestra un fragmento de código para generar los servicios REST en Ballerina. En este se analiza cuantos servicios tiene el modelo y se crea un archivo por

cada servicio asociado a un sensor o un actuador. A continuación, se establece el puerto del servidor en el que se ejecutará el servicio REST (línea 6), luego se define un objeto `Device` en el que se almacenará la información procedente de los sensores o actuadores (mostrado entre las líneas 7 a 13) y por último se define la URI con la que el servicio será accesible (mostrado entre las líneas 15 a 17).

```
1  [comment SERVICIOS API RESTFULL/]
2  [for (webservice : WebService | aSystem.webservice)]
3   [for (rest : REST | aSystem.webservice.rest)]
4    [file (rest.device.name.toString().concat('.bal'), false, 'UTF-8')]
5     ...
6     listener http:Listener httpListener = new([rest.port/]);
7     type Device record {
8      string date;
9      string time;
10     string location;
11     float attribute;
12     string artefact;
13    };
14    ...
15    @http:ServiceConfig {
16     basePath: "/rest[rest.URI/]"
17    }
18    ...
19   [/file]
20  [/for]
21 [/for]
```

Listado 4.2: Transformación M2T para la generación de código Ballerina para implementar los servicios REST.

En la construcción de los `Bridge`, se consideró la plataforma Node-Red en la cual se definen el `OutputBridge` y `InputBridge`. Para lo cual se siguió los siguientes pasos: (1) Definición del flujo para conexión con MQTT; (2) Definición de la URI para conexión con el servicio REST; (3) Definición de la URI para conexión con el Orquestador; y (4) Definición de las credenciales para la conexión con el Message Broker.

En el Listado 4.3 se muestra un fragmento de código para generar el `Bridge` en Node-Red. En el que se muestran dos nodos, el primero con el tópico MQTT específico del sensor o actuador al que se suscribe o publica información (mostrado entre las líneas 6 a 10), y el nodo con las credenciales para conectarse al Message Broker que coordina los tópicos (mostrado entre las líneas 11 a 16).

```
1  [for (broker : BridgeServer | aSystem.bridgeserver)]
2   [for (out : OutputBridge | broker.bridge)]
3    [file (out.sensor.name.toString().concat('.json'), false, 'UTF-8')]
4     ['['/]
5      ...
6     {
7      "id":"5395ff74.042a3", "type":"mqtt in", "z":"2fd02feb.4b81c", "name":"MQTT IN",
8      "topic":"[out.topic/]", "qos":"2", "datatype":"auto", "broker":"a7d68513.f26bd8",
9      "x":180, "y":80, "wires":['['/]['['/]"3ace6424.a7cbfc""[']'/][']'/]
10    }, ... ,{
11     "id":"a7d68513.f26bd1", "type":"mqtt-broker", "z":"", "name":"[aSystem.name/]Broker",
12     "broker":"[broker.messagebroker.host/]", "port":"[broker.messagebroker.port/]", "
       clientid":"",
13     "usetls":false, "compatmode":true, "keepalive":"60", "cleansession":true, "birthTopic
       ":"",
14     "birthQos":"0", "birthPayload":"", "closeTopic":"", "closeQos":"0", "closePayload":""
       ,
15     "willTopic":"", "willQos":"0", "willPayload":""
16    }
17    [']'/]
18   [/file]
19  [/for]
20  ...
21 [/for]
```

Listado 4.3: Transformación M2T para la generación de código Node-Red para implementar los Bridge.

Para la construcción del Orquestador, se consideró la plataforma Ballerina y para obtener código funcional se siguieron los siguientes pasos: (1) Inclusión de las librerías para el protocolo HTTP; (2) Definición de las URI de los servicios REST que aportan información para el procesamiento de los eventos; (3) Definición de los eventos que provocan que se ejecute el Orquestador; (4) Procesamiento de la información recibida en el Orquestador para ejecutar la lógica de negocio; (5) Definición de la respuesta con el resultado del procesamiento de los eventos; Y (6) Definición de la URI del servicio al que se envía información para los actuadores o para otro Orquestador.

En el Listado 4.4 se muestra un fragmento de código para generar el Orquestador en Ballerina. Aquí se define el puerto que empleará el Orquestador para ejecutarse en el servidor (línea 6), el end-point de los servicios REST que consultará el Orquestador cuando realice el procesamiento de los eventos (mostrado entre las líneas 7 a 13) y la URI que tendrá el Orquestador para que pueda ser consultado, con lo cual empieza el proceso para la coordinación de los servicios de acuerdo a la lógica de negocio de la aplicación (línea 19).

```
1  [comment Patron de integracion/]
2  [for (integration : IntegrationPattern | aSystem.integrationpattern)]
3   [for (orchestrator : Orchestrator | integration.orchestrator)]
4    [file (orchestrator.name.toString().concat('.bal'), false, 'UTF-8')]
5     ...
6     listener http:Listener asyncServiceEP = new([orchestrator.port/]);
7     [for (rest : REST | orchestrator.rest)]
8      [for (ws : WebService | aSystem.webservice)]
9       [for (rest1 : REST | ws.rest)]
10       [if (rest1.device.name = rest.device.name)]
11        http:Client [rest.device.name/]EP = new("http://[ws.webserver.host/]:[rest.port/]/
          rest[rest.URI/]/[rest.device.name/]");
12       [/if]
13      [/for]
14     [/for]
15    [/for]
16    ...
17    http:Response finalResponse = new;
18    json responseJson = ();
19    @http:ServiceConfig { basePath: "/[orchestrator.name/]" }
20    ...
21   [/file]
22  [/for]
23 [/for]
```

Listado 4.4: Transformación M2T para la generación de código Ballerina para implementar los Orquestadores.

4.3.3.3. Nivel DTV

En el nivel DTV se generan archivos con código NCL para el diseño visual y código Lua para el procedimiento de comunicación con los servicios del sistema. Estos archivos se ejecutan en los TV con el estándar ISDB-T o en los sistemas de IPTV. Para la construcción del código se empleó una transformación M2T con el modelo generado en el editor gráfico siguiendo los siguientes pasos: (1) Definición las regiones y descriptores de los componentes visuales; (2) Definición del canal de video en el que se multiplexa la aplicación; (3) Selección de la plantilla de la interfaz propuesta: Accordion, Banner o Frame; (4) Inclusión de las librerías para el protocolo TCP; (5) Definición de la URI del servicio; y (6) Extracción de la información devuelta por el servidor.

En el Listado 4.5 se muestra un fragmento de código para el consumo de los servicios web de la App de DTV, en Lua. En este se muestra la creación de uno de los archivos Lua

asociado al botón amarillo del mando a distancia. En este caso el programa realiza la configuración de los parámetros del servicio (REST u Orquestador) al cual va a realizar una consulta (mostrado entre las líneas 11 a 22), la respuesta a la consulta se gurda en una variable auxiliar (línea 23), del cual se extrae la información y se la guarda en texto claro en un archivo para que pueda accedido para los componentes visuales que requieran la información (mostrado entre las líneas 26 a 40).

```
1  [for (y : Yellow | aSystem.dtv.interface.yellow)]
2   [for (yt : Text | y.content)]
3    [for (yto : REST | yt.rest)]
4     [file ('media/'.concat(yto.device.name.concat('.txt')), false)][/file]
5     [file (yto.device.name.concat('.lua'), false)]
6      ...
7      [for (w : WebService | aSystem.webservice)]
8       [for (wr : REST | w.rest)]
9        [for (wrd : Device | wr.device)]
10        [if (wrd.name = yto.device.name)]
11         hostIP = '[w.webserver.host/]'
12         tcp.connect(hostIP, [wr.port/])
13         local url = "GET /rest[yto.device.name/]/[yto.device.name/]/last HTTP/1.1\r\n\r\n"
14         tcp.send(url)
15         tcp.send('Host: [w.webserver.host/]:[wr.port/]\r\n')
16         tcp.send('User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:65.0) Gecko
       /20100101 Firefox/65.0\r\n')
17         tcp.send('Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
       ,*/*;q=0.8\r\n')
18         tcp.send('Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n')
19         tcp.send('Accept-Encoding: gzip, deflate\r\n')
20         tcp.send('Connection: keep-alive\r\n')
21         tcp.send('Upgrade-Insecure-Requests: 1\r\n')
22         tcp.send('\r\n')
23         result = tcp.receive()
24         print("Dato recibido [yto.device.name/]")
25         print(result)
26         file = io.open("media/aux[yto.device.name/].txt","w")
27         [file ('media/aux'.concat(yto.device.name).concat('.txt'), false)][/file]
28         file:write("[yto.device.name/]: ")
29         file:write(result)
30         file:write("\n")
31         file:close()
32         var3=result
33         var5=string.find(var3, '"attribute":')
34         var51=string.find(var3, ', "artefact"')
35         var6=string.sub(var3,var5+12,var51-1)
36         file = io.open("media/[yto.device.name/].txt","w")
37         file:write("[yto.device.name/]: ")
38         file:write(var6)
39         file:write("\n")
40         file:close()
41         ...
42        [/if]
43       [/for]
44      [/for]
45     [/for]
46    [/file]
47   [/for]
48  [/for]
49 [/for]
```

Listado 4.5: Transformación M2T para la generación de código NCL-Lua para implementar las aplicaciones de DTV.

4.3.3.4. Nivel Móvil

Por último, en el caso del nivel Móvil se generan archivos con código Java y XML para el diseño la interfaz en Android de la aplicación IoT. Para la construcción del código se empleó una transformación M2T a partir del modelo generado con el editor gráfico. Para ello, se realizaron los siguientes pasos de proceso: (1) Inclusión de las librerías para el protocolo HTTP y para los componentes de la interfaz; (2) Declaración de los componentes; (3) Definición de la URI del servicio; (4) Extracción de la información

devuelta por el servidor; (5) Definición de la ubicación de los componentes en la interfaz; y (6) Definición de los permisos para el acceso a Internet.

En el Listado 4.6 se muestra un fragmento de código para generar la interfaz móvil. En el cual se observa código del archivo XML de una aplicación Android, que describe las características de los componentes `Button` con el evento que se ejecuta cuando este es presionado (mostrado entre las líneas 14 a 23); los componentes `TextView` y `EditText`, que son empelados para cuando la aplicación recibe información de un servicio REST y para cuando se va a enviar información a un servicio REST, un Bridge o un Orquestador (mostrado entre las líneas 30 a 44).

```
1  [for (movil : Mobile | aSystem.mobile)]
2   [for (actividad : Activity | movil.activity)]
3    ...
4    [file ('activity_'.concat(actividad.name.toLower().concat('.xml')), false, 'UTF-8')]
5     <?xml version="1.0" encoding="utf-8"?>
6     <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.
        android.com/apk/res/android"
7       ...
8      [for (vistas : View | actividad.view)]
9       [for (contenido : ContentM | vistas.contentm)]
10       [let x : Integer = i]
11        [for (contenido1 : ContentM | vistas.contentm)]
12         [if (x=contenido1.order.order)]
13          [if (contenido1.eClass().name.toString()='ButtonM')]
14           <Button
15            android:id="@+id/[contenido1.name/]"
16            android:layout_width="match_parent"
17            android:layout_height="wrap_content"
18        [if (contenido1.activity.name.toString() <> 'invalid' )]
19             android:onClick="[contenido1.name/]Met"
20        [else]
21         android:onClick="data[contenido1.name/]"
22        [/if]
23              android:text="[contenido1.text/]" />
24          [/if]
25         [/if]
26        [/for]
27        [for (contenido1 : TextView | vistas.contentm)]
28         [if (x=contenido1.order.order)]
29          [if (contenido1.type.toString()='Output')]
30           <TextView
31            android:id="@+id/[contenido1.name/]"
32            android:layout_width="match_parent"
33            android:layout_height="wrap_content"
34            android:text="[contenido1.text/]"
35            android:textSize="18sp" />
36          [/if]
37          [if (contenido1.type.toString()='Input')]
38           <EditText
39            android:id="@+id/[contenido1.name/]"
40            android:layout_width="match_parent"
41            android:layout_height="wrap_content"
42            android:ems="10"
43            android:inputType="textPersonName"
44            android:text="[contenido1.text/]" />
45          [/if]
46         [/if]
47        [/for]
48       [/let]
49      [/for]
50     [/for]
51    </LinearLayout>
52    </androidx.constraintlayout.widget.ConstraintLayout>
53   [/file]
54  [/for]
55  ...
56 [/for]
```

Listado 4.6: Transformación M2T para la generación de código Java y XML para implementar las aplicaciones Móviles.

4.4. Trabajos relacionados

Para determinar las investigaciones relevantes se siguió la metodología descrita en el Capítulo 1, para lo cual se estableció una lista con los términos principales de búsqueda: DSL, MDE, IoT, Orquestación, EDA, SOA, DTV y Mobile. Con los trabajos seleccionados se realizó un resumen con el aporte de cada uno y una Tabla 4.4 para visualizar de forma rápida los aportes.

	IoT	**MDE**	**Generación de código**	**Editor Gráfico**	**Interfaz no web**
[Schachinger and Kastner, 2015]	✓	✓	✓		
[Cai et al., 2018]	✓	✓	✓		
[Boubacar et al., 2016]	✓	✓			
[Thramboulidis et al., 2019]	✓	✓	✓		
[Boubeta-Puig et al., 2015]		✓	✓	✓	
[Grace et al., 2016]	✓	✓	✓	✓	
[García and Suárez, 2019]	✓	✓	✓	✓	✓
[Sánchez et al., 2017]	✓				✓
[Sneps-Sneppe and Namiot, 2015]	✓	✓			
[Salihbegovic et al., 2015]	✓	✓	✓		
[Brambilla et al., 2018]	✓	✓	✓	✓	
Propuesta	✓	✓	✓	✓	✓

Tabla 4.4: Resumen de las principales características cubiertas.

En [Schachinger and Kastner, 2015] los autores centran su trabajo en sistemas de hogar inteligentes, a través de interfaces estandarizadas definidas en un DSL. Desde una perspectiva basada en modelos, centrada en el dominio de los sistemas de automatización de edificios (BAS, Building Intelligent System), se generan varios tipos de artefactos de texto para el estándar OBIX (Building Intelligent System), con acceso a tecnologías de comunicación como BACnet, KNX, EnOcean o M-Bus (inalámbrico). A diferencia del presente trabajo, no se aborda la conectividad con Internet, ni se considera el diseño de interfaces para interactuar con los usuarios.

En la propuesta [Cai et al., 2018] los autores proponen un metamodelo que integra CIM (Computation Independent Model) con PIM (Platform Independent Model) como estructura de referencia para encapsular y administrar recursos comerciales para el desarrollo de aplicaciones para Cloud of Things. El metamodelo conecta los requisitos comerciales con los componentes ejecutables. Además, los autores proporcionan tres patrones de desarrollo: basado en roles, datos y procesos para la configuración del servicio. Estos patrones permiten el rápido desarrollo de servicios móviles basados en recursos de servicio. Sin embargo, esta propuesta no contempla el uso de interfaces no-web ni tampoco la coordinación de servicios.

En el trabajo [Boubacar et al., 2016] los autores de la propuesta presentan una herramienta de telemedicina en la que, mediante el uso de SOA y MDA, se configuran aplicaciones de software médico para la plataforma e-health, de tal forma que se pueda

conectar con cualquier dispositivo externo en cualquier área remota. Este enfoque utiliza metamodelos para resolver el problema de la heterogeneidad y complejidad en la gestión del ciclo de vida de los componentes de los equipos de telemedicina. Sin embargo, el enfoque establecido en la propuesta no explota las arquitecturas dirigidas por eventos en la capa física, la coordinación de servicios en la capa de control e incluso DTV y plataformas móviles en la capa de usuario.

En la propuesta [Thramboulidis et al., 2019], los autores discuten el concepto de un microservicio ciberfísico en el ámbito de la fabricación, y presentan la herramienta de desarrollo CPuS-IoT. Los autores emplean ingeniería dirigida por modelos para semi-automatizar el desarrollo, la evolución y operación de sistemas ciberfísicos, así como para establecer un vocabulario común para expertos en sistemas de ensamblaje e IoT. Para ello, los autores combinan MDE con IoT y el paradigma arquitectónico de microservicios. Sin embargo, el enfoque no explota las arquitecturas dirigidas por eventos en la capa física, ni tampoco la coordinación de servicios en la capa de control, ni hacen uso o desarrollan una interfaz en la capa de usuario.

En [Boubeta-Puig et al., 2015] los autores integran CEP (Complex Event Processing) con SOA 2.0 en un editor gráfico que permite la generación de código. Proponen una solución basada en modelos para la toma de decisiones en tiempo real. Esta solución permite detectar tanto los patrones de eventos como definir las alertas para notificaciones en tiempo real, ocultando todos los detalles de su implementación. El editor propuesto se reconfigura para diferentes dominios, porque el editor puede reconfigurar dinámicamente la paleta de herramientas para diferentes modelos de dominio. Este trabajo no explota ninguna plataforma en la capa de usuario, por el contrario, utiliza CEP que es un tema que no hemos abordado en esta investigación.

En [Grace et al., 2016] los autores proponen métodos de ingeniería basada en modelos para reducir el esfuerzo de desarrollo que asegura que los sistemas de software complejos interactúen entre sí. Se pueden especificar modelos de interoperabilidad ligeros para monitorizar y probar la ejecución del software en ejecución, de modo que esos problemas de interoperabilidad se puedan identificar rápidamente y a su vez encontrar soluciones. La propuesta también incluye un editor de modelos gráficos y una herramienta de prueba para resaltar cómo un modelo visual mejora las especificaciones textuales. El enfoque propuesto explora modelos que se centran únicamente en la interoperabilidad; es decir, concretar los intercambios entre los servicios de IoT con reglas que definan el comportamiento requerido para garantizar la interoperabilidad. Hay dos tipos de modelo: (a) el modelo de interoperabilidad utilizado por los desarrolladores de aplicaciones y los probadores de interoperabilidad, y (b) los modelos de especificación y los probadores de cumplimiento. El enfoque de este trabajo no emplea las arquitecturas controladas por eventos en la capa física, ni ninguna plataforma en la capa de usuario, aunque sí hace uso de modelos.

La propuesta [García and Suárez, 2019] sugiere usar MDE para diseñar un DSL para el dominio de IoT. Tiene un editor gráfico para crear objetos que se pueden interconectar con una plataforma IoT. Los usuarios eligen los sensores que tienen sus smartphones o los elementos que quieren conectar a su microcontrolador. El DSL permite la creación del software necesario para que un objeto envíe y reciba datos hacia y desde otros objetos. Los autores crean objetos para diferentes plataformas, por ejemplo, teléfonos

inteligentes, computadoras o microcontroladores. En caso de que el usuario necesite interconectar dos o más objetos diferentes, debe desarrollar diferentes aplicaciones con (a) código diferente, (b) valores diferentes en los objetos (por ejemplo, para sus sensores) y (c) un lenguaje de programación diferente; por ejemplo, si un objeto es un teléfono inteligente Android (Java), si es una computadora (Java, C, Python, etc.), o si es un microcontrolador Arduino (C). El enfoque presentado se centra en el desarrollo a nivel de usuario, con una propuesta que utiliza modelos para el dominio de IoT.

Por otro lado, en [Sánchez et al., 2017] los autores idearon un enfoque que incluía e integraba una red de sensores inalámbricos, una plataforma de IoT y una aplicación de TV interactiva real. La propuesta abarca el despliegue y la comunicación de la red de sensores inalámbricos mediante la interoperabilidad de los datos, hasta el consumo final a través de una aplicación de televisión interactiva real. El trabajo fue probado en una comunidad residencial que ofrecía información en tiempo real, con el fin de mejorar la calidad de vida de sus habitantes. Además, incorporaba la posibilidad de analizar esta información para establecer procesos con el objetivo de reducir el consumo de energía, y así mejorar la sostenibilidad y contribuir al uso eficiente de los recursos existentes. El marco propuesto sirve como base para cualquier despliegue de características similares. Sin embargo, esta propuesta centra su principal investigación en tratar de vincular la DTV con el dominio de IoT, sin llegar a cubrir el resto de temas que abarca nuestra propuesta (véase de nuevo la Tabla 4.4).

En el trabajo [Sneps-Sneppe and Namiot, 2015] los autores analizan los desafíos para la detección y recopilación de datos de diversas fuentes en el desarrollo de aplicaciones para ciudades inteligentes. En el trabajo, los autores proponen un DSL basado en páginas de servidor Java. Mediante el DSL propuesto, las operaciones permiten la comunicación IoT entre instancias de proceso y sensores. Sin embargo, este enfoque no genera código automáticamente ni tiene un editor gráfico. Además, no considera un mecanismo de interacción con los usuarios.

En [Salihbegovic et al., 2015] los autores proponen un lenguaje específico de dominio denominado DSL-4-IoT, el cual utiliza representaciones formales y un metamodelo. La interfaz visual del Editor se ha desarrollado en lenguaje JavaScript para diseñar aplicaciones IoT. La propuesta genera los archivos de configuración de aplicaciones de IoT que se ejecutan en la plataforma "OpenHAB". Para la validación de la propuesta los autores implementan un escenario E-Health basado en la placa Arduino Uno, y 10 sensores biomédicos diferentes. Sin embargo, en esta propuesta no se hace uso de ningún mecanismo de coordinación de servicios.

El trabajo de Brambilla en [Brambilla et al., 2018] analiza los requisitos y escenarios de uso que cubren aspectos de los sistemas de IoT y presenta un enfoque basado en modelos para el diseño de interfaces a través de componentes específicos y patrones de diseño utilizando un lenguaje de modelado visual para aplicaciones de IoT. La propuesta permite el diseño de interfaces de usuario para sistemas IoT, mediante la definición de patrones de diseño y componentes GUI específicos de IoT. Sin embargo, no aprovecha las arquitecturas controladas por eventos en la capa física. En cambio, la propuesta se centra en el uso de modelos para la coordinación de servicios para el dominio de IoT y utiliza una plataforma móvil en la capa de usuario.

4.5. Resumen y conclusiones

El Internet de los cosas (IoT) está transformando la forma en que se desarrollarán y operarán los sistemas modernos. Sin embargo, la adopción de IoT impone un cambio de paradigma para el desarrollo de sistemas y complica el proceso de diseño, por lo que se requieren enfoques efectivos para manejar la complejidad que introduce esta transición. Aunque muchas soluciones proporcionan interfaces para simplificar el desarrollo de aplicaciones de IoT, ninguna propone una solución verdaderamente simplificada que permita el desarrollo de aplicaciones de IoT tanto para usuarios no expertos como para expertos, que abarquen hardware, lógica de negocio y una interfaz de usuario. Es por esto que se ha propuesto e implementado una solución basada en modelos y servicios que permite la integración de plataformas de software y hardware. Así, se evita que los desarrolladores tengan que ahondar en los lenguajes de programación de todas las tecnologías que intervienen en el sistema. Las herramientas creadas son: (a) un editor gráfico para el diseño de aplicaciones de IoT y; (b) una transformación M2T para la generación de código para cada plataforma. Estas herramientas permiten a los desarrolladores facilitar el mantenimiento de los sistemas, la integración de nuevas tecnologías y aumentar la productividad y la eficiencia reduciendo errores.

Como resultado, se obtuvo un proceso semiautomático para la generación de código Ballerina para servicios RESTful y orchestrator, código Arduino para el despliegue de nodos IoT, código NCL-Lua para la interfaz DTV y código Android para Smart Phones. La propuesta incluye un paso manual antes de ejecutar la aplicación que involucra al desarrollador, el cual debe disponer de los componentes multimedia para la aplicación DTV (imágenes y videos) y debe copiar los archivos de Android (`.java` y `.xml`) en un proyecto de Android Studio para su compilación. Finalmente, las conexiones eléctricas de los componentes (sensores y actuadores) deben realizarse en el controlador, el cual debe estar dentro del área de cobertura de la red WiFi especificada en el modelo.

Capítulo 5

Escenarios Experimentales

Capítulo 5

Escenarios Experimentales

Contenidos

Las aplicaciones IoT son sistemas intrínsecamente heterogéneos, compuestos por múltiples sensores, actuadores, protocolos de comunicación y sistemas operativos diferentes, lo cual hace que la prueba de los sistemas sea una tarea verdaderamente desafiante [Bosmans et al., 2019]. En este sentido, una vez ya definidas las tres etapas principales de la metodología para el desarrollo de aplicaciones IoT, es necesario realizar una implementación de las pruebas que permitan evaluar y validar las herramientas construidas. En esta sección, se proporciona una descripción general de las técnicas de prueba utilizadas para evaluar los proyectos de IoT.

Este capítulo se organiza en cinco secciones. La Sección 5.1 presenta una introducción y una descripción de los conceptos principales que se emplean para el desarrollo de las pruebas. La Sección 5.2 describe el escenario de prueba en el que se aplicarán las herramientas creadas. En la Sección 5.3 se presentan las actividades realizadas para acometer con la evaluación y validación. La Sección 5.4 revisa algunos trabajos relacionados con la evaluación de las aplicaciones IoT generadas. El capítulo termina con la Sección 5.5, la cual presenta un breve resumen de los contenidos presentados así como algunas de las conclusiones extraídas tras el análisis experimental.

5.1. Introducción

Con el paso del tiempo, las expectativas de velocidad, rendimiento y confiabilidad de los dispositivos están aumentando desde el punto de vista del consumidor de los sistemas IoT. Por lo tanto, no es de extrañar que las personas quieran más control e información de sus dispositivos, por lo que probar la infraestructura IoT es absolutamente crucial. En este sentido, las pruebas de IoT se refieren a la analítica, las redes, las plataformas, los estándares, los sistemas operativos, los procesadores y la seguridad de toda la infraestructura IoT [Ramgir, 2019].

Dado que existe una fuerte cohesión entre el hardware y el software en los proyectos de IoT, el enfoque de prueba utilizado por lo general se basa en las mejores prácticas utilizadas en el desarrollo clásico de software o hardware [Bosmans et al., 2019]. Es así que se pueden ejecutar los siguientes tipos de prueba:

(a) Pruebas de software.

(b) Pruebas de usabilidad.

(c) Pruebas funcionales.

(d) Pruebas de escalabilidad.

(e) Pruebas de rendimiento.

(f) Pruebas de seguridad.

(g) Pruebas de integridad.

Cada uno de estos tipos de prueba tiene el objetivo de encontrar errores en el sistema, con lo cual se logra reducir los errores y mejorar la calidad del sistema para hacerlo más resistente a fallas. En este sentido para evaluar las herramientas generadas en esta propuesta, se optó por realizar tres tipos de pruebas: las de usabilidad, las funcionales y las de rendimiento.

Pruebas de usabilidad. Se pretende medir la *experiencia del usuario*. Mediante estas pruebas se pretende conocer las opiniones que tienen los usuarios al emplear las herramientas. En este contexto se ha optado por emplear dos instrumentos:

— Escala PANAS (Positive and Negative Affect Schedule): Este instrumento permite determinar las emociones positivas o negativas que los usuarios experimentan durante el uso de las herramientas.

— Escala SUS (System Usability Scale): Este instrumento permite medir la efectividad, eficiencia y satisfacción de los usuarios con las herramientas.

Los experimentos de uso se realizaron con estudiantes de la Universidad de las Fuerzas Armadas, en Quito, Ecuador, durante el segundo semestre de 2019, sobre dos grupos de 12 y 21 estudiantes de dos carreras de ingeniería.

Pruebas funcionales. Con estas pruebas se pretende validar el funcionamiento de las herramientas de acuerdo con los requisitos para generar aplicaciones IoT. Por lo tanto, esta prueba asegura que un dispositivo IoT sea capaz de proporcionar un rendimiento de referencia y puede funcionar sin errores. Además, se ha empleado el análisis de Puntos de Función, con el fin de validar la eficiencia de las herramientas con respecto al tiempo necesario que le toma a un desarrollador generar las aplicaciones.

Pruebas de rendimiento. Con este tipo de pruebas se pretende comprobar que las aplicaciones IoT generadas con la herramienta funcionen correctamente dentro de una determinada carga de trabajo. Para las pruebas de rendimiento se utilizó el framework Gatling [Maila-Maila et al., 2019] en 4 escenarios distintos con 1000, 2000, 5000 y 10000 usuarios concurrentes.

5.2. Escenarios de prueba y experimentación

Tras el desarrollo de las herramientas descritas en el capítulo anterior, en esta sección se demuestra la utilidad que estas presentan a través de un caso de uso específico. Para lo cual, en esta sección se plantea diseñar una aplicación IoT en el dominio del Hogar Inteligente. La Figura 5.1 muestra un escenario en el cual los usuarios interactúan con su hogar, especificando rutinas en base a los datos que proporcionan los sensores y los estados de los actuadores. En este escenario hay dos bombillas, un calentador, un ventilador, un sensor para detectar la apertura de una puerta, un sensor de temperatura, un sensor de movimiento y un sensor de CO_2. Además, hay una televisión inteligente y una aplicación móvil, que permiten a los usuarios interactuar con algunos de los sensores y actuadores del hogar previamente definidos.

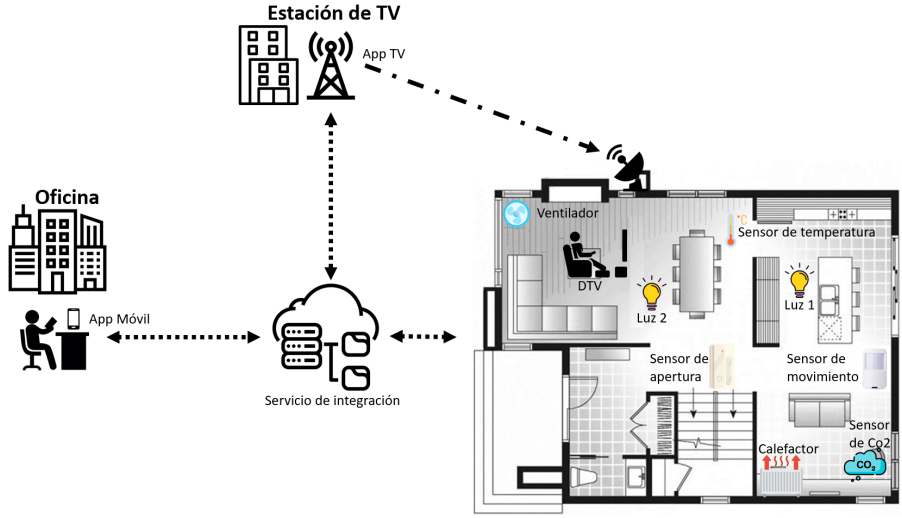


Figura 5.1: Escenario de prueba.

5.2.1. Modelo del Escenario de Hogar Inteligente

En el escenario propuesto se pretende detectar la presencia de personas en un hogar y determinar las acciones en tiempo real de los actuadores desplegados. Para lo cual se crea un modelo en el editor gráfico y se implementan los niveles: (a) Infraestructura, (b) Control, (c) Hardware, (d) DTV y (e) Móvil de la arquitectura de integración propuesta en este trabajo. En la Figura 5.2 se ilustra el modelo de hogar inteligente implementado con el editor gráfico generado, en el cual, además de los componentes del sistema, se muestran las relaciones que tienen.

Con el objetivo de validar la potencialidad de las herramientas desarrolladas, se optó por implementar en el modelo la lógica de control de una bombilla y del ventilador, los cuales implementan los dos mecanismos propuestos para la integración y la coordinación de los componentes del sistema. Además, se implementan una interfaz para DTV y una para Móvil, las cuales interactúan con los sensores y actuadores. Por último, se detalla la configuración de hardware necesaria para el despliegue de los sensores y actuadores, así como las características de la infraestructura de red.

5.2.1.1. Nivel de Infraestructura

En el caso del nivel de Infraestructura (Figura 5.3) las características que se ingresaron en los componentes son las siguientes:

— `Web Server`: En este caso se ha empleado el Framework Ballerina para el despliegue de los servicios REST y los Orquestadores. Este Framework se ejecuta en un servidor virtual privado (VPS, Virtual Private Serve) con Windows Server 2012 en la dirección IP `167.86.81.93`.

— `Data Base Server`: En este caso se empleó el servidor de Bases de Datos MySQL, el cual se ejecuta en el VPS antes mencionado.

— `Acces Point`: Es el router inalámbrico que provee la conexión de Internet, a la que se conectan los nodos de hardware con la información del SSID y password.

— `Message Broker`: Es el servidor de mensajería Eclipse Mosquitto, el cual se ejecuta en el VPS antes mencionado.

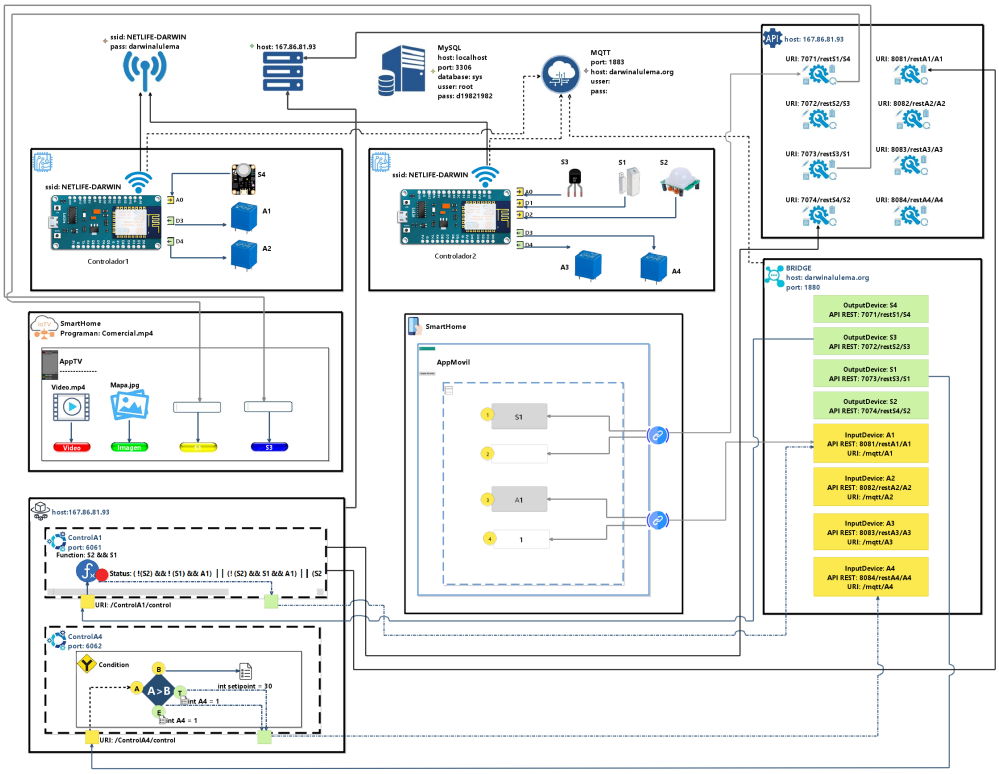


Figura 5.2: Representación gráfica del modelo para el escenario de prueba de un sistema IoT.

Figura 5.3: Nivel de Infraestructura del escenario de prueba de un sistema IoT.

5.2.1.2. Nivel Hardware

En la Figura 5.4 se muestra un fragmento del modelo del escenario propuesto con el nivel de Hardware. Además, a continuación, se detalla las características que se ingresaron en los componentes:

— `Controller`: Para el diseño del controlador se empleó una tarjeta Node MCU y debido a que solo dispone un puerto analógico se emplearon dos tarjetas. El Controlador 1 para la conexión del sensor de CO_2 y el Controlador 2 para el sensor de temperatura.

— `Sensor`: En este caso se dispone de cuatro sensores:

- *Apertura, MC 38 (S1)*: Se conecta al pin 2 del puerto digital del Controlador 2.
- *Movimiento, HC SR 501 (S2)*: Pin 1 del puerto digital del Controlador 2.
- *Temperatura, LM 35 (S3)*: Pin 0 del puerto analógico del Controlador 2.
- CO_2 MQ7 (S4): se conecta al pin 0 del puerto analógico del Controlador 1.

— `Actuator`: En este caso se dispone de cuatro actuadores:

- *Relé de control de Luz 1 (A1)*: Pin 3 del puerto digital del Controlador 1.
- *Relé de control de Luz 2 (A2)*: Pin 4 del puerto digital del Controlador 1.
- *Relé de control de Calefactor (A3)*: Pin 3 del puerto digital del Controlador 2.
- *Relé de control de Ventilador (A4)*: Pin 4 del puerto digital del Controlador 2.

— `Communication`: Establece la configuración del módulo ESP8266 del controlador NODE MCU para la conexión a la red provista por el `Acces Point`.

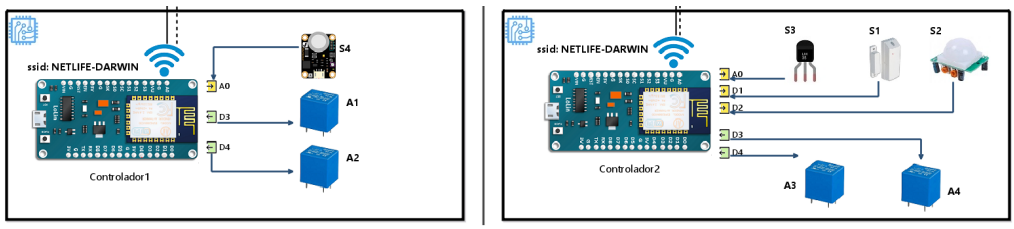
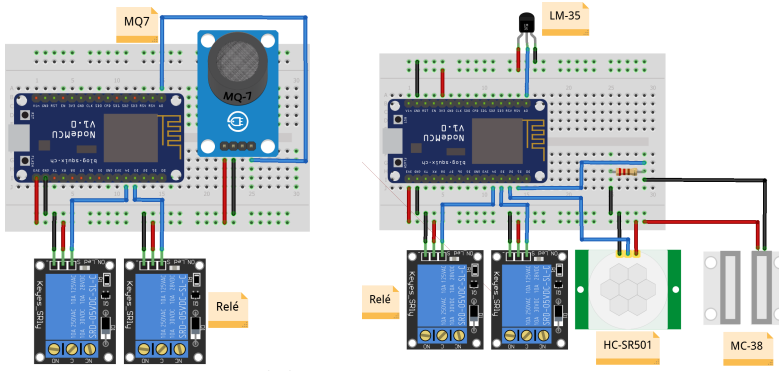


Figura 5.4: Nivel de Hardware del escenario de prueba de un sistema IoT.

Además del modelado de los artefactos software que se realiza en el editor gráfico, también es necesario establecer el esquema electrónico para la conexión de los sensores y actuadores para la implementación del circuito de prueba en una placa de pruebas. En la Figura 5.5 se muestra su esquema eléctrico.



(a) Controlador 1 (b) Controlador 2

Figura 5.5: Esquema eléctrico del nivel Hardware para los Controladores 1 y 2.

5.2.1.3. Nivel de Control

El nivel de Control está compuesto por tres componentes principales: (a) el Bridge, que permite integrar los mensajes provenientes de los nodos de hardware con los servicios; (b) los servicios RESTful, que almacenan y proveen recursos para consultar los estados de los sensores y actuadores; y (c) el patrón de integración, que permite implementar la lógica de control de la aplicación.

Componente `Bridge`. La Figura 5.6 ofrece un fragmento del modelo del escenario propuesto con el nivel de Control. En él se muestran los ocho artefactos Bridge que se despliegan en el servidor Node-Red que está alojado en el VPS y se ejecutan en el puerto 1880. Los primeros cuatro corresponden a los sensores y los siguientes a los actuadores.

Servicios RESTful. La Figura 5.7 ofrece un fragmento del modelo del escenario con el nivel de Control. Se observan ocho servicios REST que se despliegan en el Servidor Web alojado en el VPS. Para el despliegue de los servicios se consideró el rango de puertos 707x para los sensores y el rango de puertos 808x para los actuadores. Además, cada servicio implementa los métodos POST, GET, PUT y DELETE.

Patrón de Integración. En la Figura 5.8 se muestra un fragmento del modelo del escenario propuesto con el nivel de Control, en cual se muestran los patrones de integración de los servicios que permiten el control de la bombilla (A1) y del ventilador (A4). Para el primer caso se implementó un patrón basado en una función lógica y para el segundo caso un patrón basado en un condicional.

Para establecer el *patrón de integración* basado en función (parte superior de la Figura 5.8), se asumió que la bombilla (A1) debe activarse cuando el usuario llega a casa, abre la puerta y se detecta movimiento en el interior. De esta manera el sensor de movimiento actúa como un *Trigger* para iniciar la operación del orquestador.

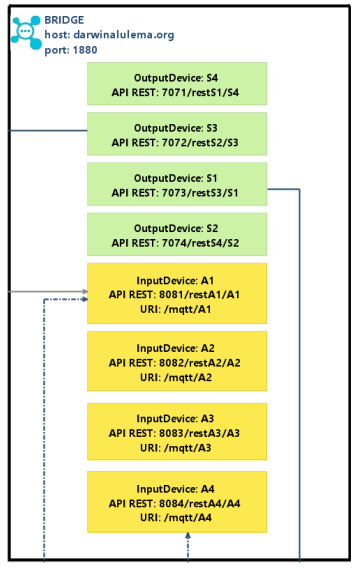


Figura 5.6: Servicio Bridge del nivel de Control.

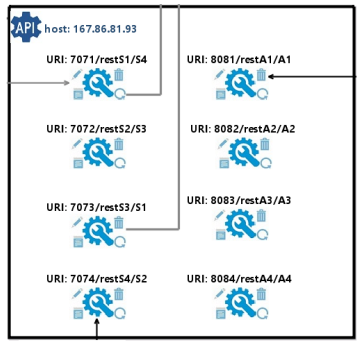


Figura 5.7: Servicios RESTful del nivel de Control.

Para construir la función lógica encargada del control de la bombilla se utilizó un diagrama BPMN [Valderas et al., 2019] [Torres et al., 2020], ya que es una notación gráfica que permite representar la secuencia de actividades que ocurren durante la ejecución de un proceso específico. Esta notación está diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. La Figura 5.9 ilustra las condiciones para el control de la bombilla, en la que la función lógica de la bombilla depende del estado actual del actuador y de los últimos estados de los sensores de movimiento y contacto. Además, una característica esencial en el modelo es la función *Status*, que representa el último estado cuando la información no debe enviarse al actuador, por ser datos redundantes, evitando así la saturación de la red.

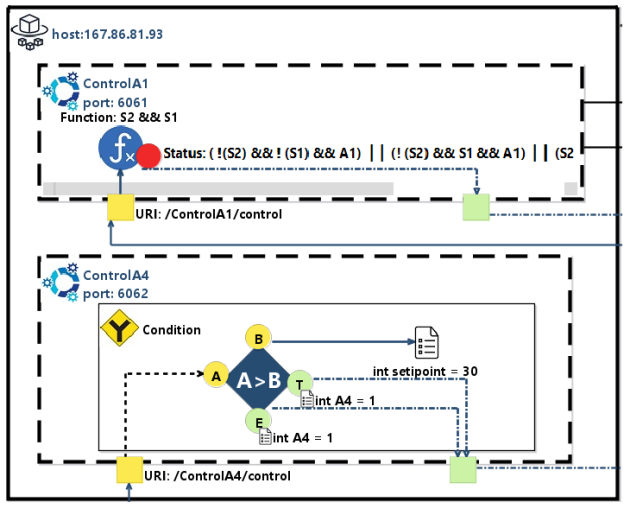


Figura 5.8: Patrón de Integración del nivel de Control.

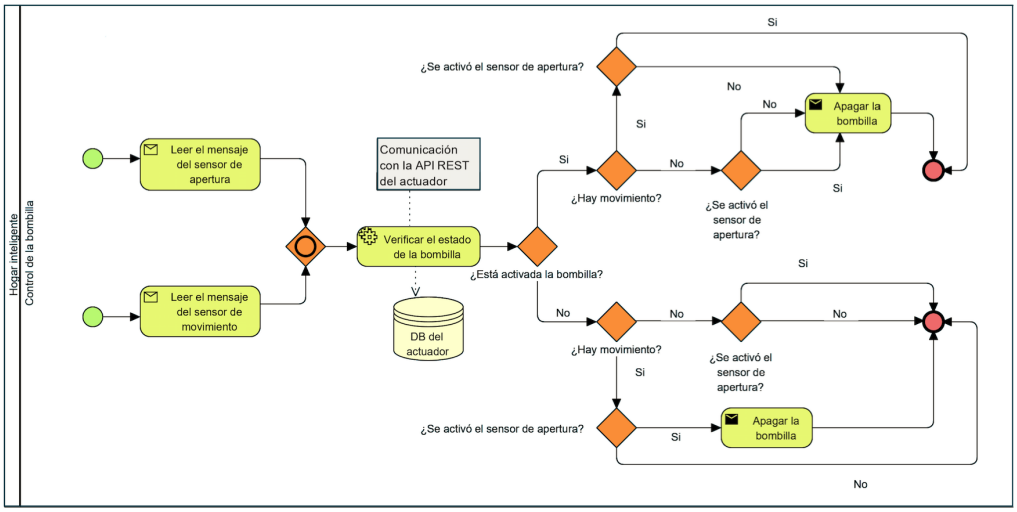


Figura 5.9: Diagrama BPMN del proceso de control la bombilla (A1).

Para representar de forma algebraica todos los casos descritos en el diagrama BPMN, se crea una tabla de verdad que muestra el comportamiento de los actuadores cuando están encendidos o apagados y de los sensores cuando detecta cambios en el entorno. La Tabla 5.1 ilustra las condiciones para el control de la bombilla (A1), en función del sensor de movimiento (S3)(S2) y el sensor de apertura (S4)(S1). Además, la Tabla 5.2 ilustra las condiciones en las cueles el nivel de control no debe enviar información redundante.

Sensor de movimiento (S2)	Sensor de apertura (S1)	Luz1(A1)
OFF	OFF	OFF
OFF	ON	OFF
ON	OFF	OFF
ON	ON	ON

Tabla 5.1: Tabla lógica para la función de control de A1.

Sensor de movimiento (S2)	Sensor de apertura (S1)	Luz1(A1)	Status
OFF	OFF	OFF	OFF
OFF	OFF	ON	ON
OFF	ON	OFF	OFF
OFF	ON	ON	ON
ON	OFF	OFF	OFF
ON	OFF	ON	ON
ON	ON	OFF	ON
ON	ON	ON	OFF

Tabla 5.2: Tabla lógica para la función de status de A1.

La función resultante para el control de la bombilla (A1) surge del análisis de la Tabla 5.1; y la función para *Status* se determina de acuerdo con la Tabla 5.2. Estas funciones lógicas tienen su expresión Canónica basada en mintérminos, lo que corresponde a la suma (OR) de los productos (AND) de los estados de los sensores. Estas funciones permiten determinar si el orquestador debe enviar la información resultante de la evaluación de la función lógica al actuador. A continuación, se detallan las dos funciones se ingresan como parámetros al modelo creado en el Editor gráfico:

$$Function = S2 * S1 \tag{5.1}$$

$$Status = (\overline{S2} * \overline{S1} * A1) + (\overline{S2} * S1 * A1) + (S2 * \overline{S2} * A1) + (S2 * S1 * \overline{A1}) \tag{5.2}$$

Para establecer el patrón de integración basado en condicional del ventilador (A4) (parte inferior de la Figura 5.8), se considera la condición If con el fin de analizar los cambios de temperatura detectados por el sensor con respecto al valor de referencia (por ejemplo 36 grados centígrados). En este caso, la temperatura es el *Trigger* del orquestador que analiza si el valor enviado por el sensor es mayor al valor de referencia. El resultado de esta evaluación construye un objeto JSON con información para que el ventilador cambien de estado a activo o a apagado.

5.2.1.4. Nivel de DTV

En la Figura 5.10 se muestra un fragmento del modelo del escenario propuesto con el nivel de DTV y se detalla las características que se ingresaron en los componentes. Para la aplicación de DTV se escogió la interfaz "Accordion", en la cual se despliegan los cuatro botones básicos: (a) Rojo, para controlar el despliegue de un vídeo; (b) Verde, para el despliegue de una imagen; (c) Amarillo, para la consulta del sensor S1 (Apertura); y (d) Azul: realiza la consulta del sensor S3 (Temperatura).

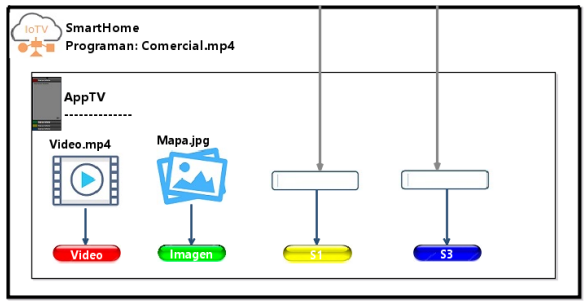


Figura 5.10: Modelo de la interfaz DTV del escenario de prueba.

5.2.1.5. Nivel Móvil

En la Figura 5.11 se muestra un fragmento del modelo del escenario propuesto con el nivel Móvil, en el cual se muestra la interfaz que despliega dos botones para realizar la consulta al servicio REST del sensor S1 (Apertura) y un campo de texto vinculado en el cual se presenta la información devuelta por el servicio REST. Para el caso del actuador A1 (Luz1), la información ingresada en el campo de texto vinculado al botón es enviada al Bridge del actuador con la finalidad de cambiar su estado actual.

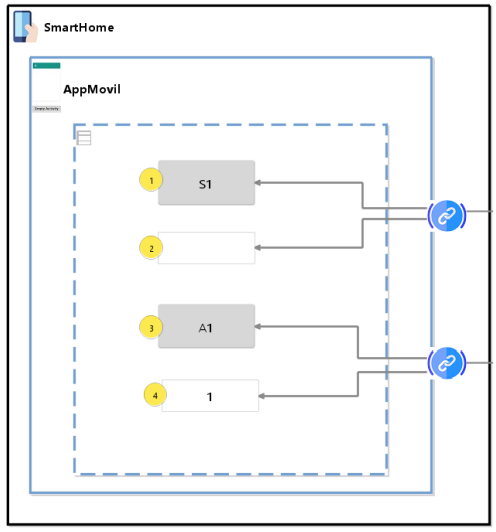


Figura 5.11: Modelo de la interfaz móvil del escenario de prueba.

5.3. Evaluación y validación

Para evaluar y validar la herramienta y los artefactos de software se consideraron las siguientes pruebas:

— *Pruebas de usabilidad*: Se consideró dos acciones: (a) analizar el comportamiento de los usuarios por medio de la escala PANAS, y (b) evaluar la usabilidad del Editor Gráfico por medio de la escala SUS.

— *Pruebas funcionales*: Se consideró dos acciones: a) validar el funcionamiento de los artefactos de software generados, y b) analizar mediante Puntos de Función los artefactos generados y el tiempo necesario para crearlos.

— *Pruebas de rendimiento*: En este caso se realiza pruebas de carga empleado Gatling con el fin de evaluar el comportamiento del sistema.

5.3.1. Pruebas de usabilidad

Para las pruebas de usabilidad se consideraron dos acciones: (a) analizar el comportamiento delos usuarios por medio de la escala PANAS, con el propósito de conocer las emociones positivas o negativas que la herramienta despierta en los usuarios; y (b) evaluar la usabilidad del Editor Gráfico por medio de la escala SUS, con el propósito de determinar la percepción que tienen los usuarios al emplear las herramientas.

5.3.1.1. Escala de Afectividad PANAS

Con el propósito de conocer el comportamiento de los usuarios de las herramientas creadas se planteó una experiencia con estudiantes de la Universidad de las Fuerzas Armadas, en Quito, Ecuador, del segundo semestre de 2019. Con estos estudiantes se conformador dos grupos donde realizar los experimentos:

— *Grupo Experimental 1* (EG1/Nivel Bajo): Conformado por 6 hombres y 6 mujeres, estudiantes de primer año del grado de Ingeniería en Tecnologías de la Información.

— *Grupo Experimental 2* (EG2/Nivel Medio): Conformado por 18 hombres y 3 mujeres, estudiantes de tercer año de la carrera de Ingeniería Mecatrónica.

La edad media de los estudiantes del grupo EG1 era de 19 años y para los del grupo EG2 de 23 años. Los estudiante del grupo EG1 poseían poco conocimiento en desarrollo de software como de electrónica, mientras que los del grupo EG2 presentaban un nivel medio de conocimiento en aspectos de mecánica, electrónica y software. Los dos grupos realizaron un pre-test al inicio de la experiencia y un post-test al final.

La Tabla 5.3 muestra la escala de emociones PANAS que emplea una escala de Likert (del 1 al 5), la cual es una herramienta de medición diferente a las preguntas dicotómicas con respuesta afirmativa/negativa, que permite medir actitudes y comprender el grado en que el encuestado está de acuerdo con cualquiera de las afirmaciones de la escala

	TERMINOS DE EMOCIONES POSITIVAS		TERMINOS DE EMOCIONES NEGATIVAS
1	Listo	*1*	Asustado
2	Interesado	*2*	Nervioso
3	Enérgico	*3*	Tenso
4	Animado	*4*	Irritado
5	Entusiasta	*5*	Enojado
6	Orgulloso	*6*	Temeroso
7	Inspirado	*7*	Intranquilo
8	Atento	*8*	Avergonzado
9	Activo	*9*	Disgustado
10	Decidido	*10*	Culpable

Tabla 5.3: Emociones de la escala PANAS.

PANAS. En este sentido, la categoría de respuesta ayudará a captar la intensidad de los sentimientos del encuestado sobre la afirmación.

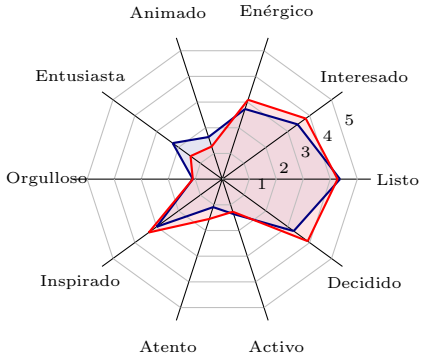
En el caso del grupo EG1 la Figura 5.12.a y 5.12.b muestran los resultados del pre-test (azul) y el post-test(rojo) con los resultados de las emociones positivas y negativas respectivamente. Se observa que durante el experimento existió un aumento de las emociones positivas y una disminución de las negativas. Para el grupo EG2 la Figura 5.12.c y 5.12.d muestran los resultados del pre-test (azul) y el post-test(rojo) con los resultados de las emociones positivas y negativas respectivamente. Se observa que durante el experimento tanto las emociones positivas como las negativas permanecieron casi iguales, con un ligero aumento en las positivas.

5.3.1.2. Escala de usabilidad (SUS)

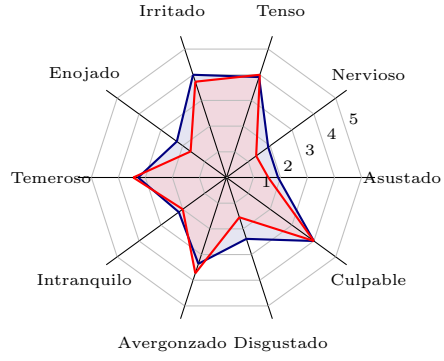
Para evaluar la usabilidad de las herramientas construidas para el modelado de aplicaciones IoT se empleó la escala SUS y se realizó un experimento con estudiantes en el aula. Los estudiantes que participaron en la experiencia son los descritos en el apartado anterior y realizaron un post-test al final de las actividades. La Tabla 5.4 muestra el cuestionario compuesto por diez preguntas, que se puntúan mediante una escala Likert (del 1 al 5). La Figura 5.13 muestra los resultados de la aplicación SUS.

	LISTA DE ASEVERACIONES
Q1	Creo que me gustará usar el sistema frecuentemente.
Q2	Encuentro el sistema innecesariamente complejo.
Q3	Pensé que el sistema era fácil de usar.
Q4	Creo que necesitaré la ayuda de un técnico para poder usar el sistema.
Q5	Encontré que las distintas funciones del sistema estaban bien integradas.
Q6	Pensé que había mucha inconsistencia en el sistema.
Q7	Me imagino que la gente aprenderá a usar el sistema rápidamente.
Q8	Encuentro el sistema engorroso de usar.
Q9	Me sentí muy seguro usando el sistema.
Q10	Necesito aprender muchas cosas antes de poder utilizar el sistema.

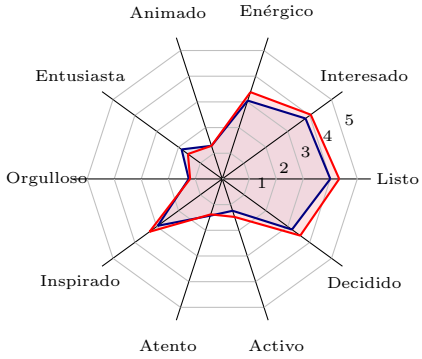
Tabla 5.4: Preguntas consultadas para la prueba de usabilidad.



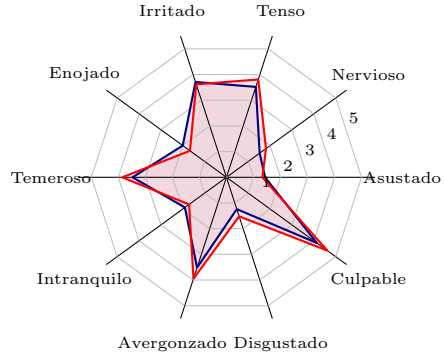
(a) Emociones positivas en EG1



(b) Emociones negativas en EG1



(c) Emociones positivas en EG2



(d) Emociones negativas en EG2

Figura 5.12: Resultados de las pruebas PANAS (azul = Pre-Test, rojo = Post-Test).

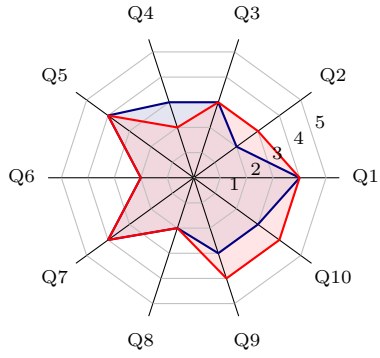


Figura 5.13: Resultados de SUS en EG1 (rojo) y EG2 (azul).

En este caso, las afirmaciones Q1, Q5, Q7 son las que mayor puntuación obtuvieron en los grupos EG1 y EG2. Estos resultados indican que los encuestados *están de acuerdo* en afirmar que la herramienta es fácil de usar, está bien integrada y que su uso es fácil de aprender. En contra parte, las preguntas Q6 y Q8 fueron las que menor puntuación obtuvieron en los dos grupos, estos resultados indican que los encuestados consideran que *no corresponden* las afirmaciones de inconsistencia del programa o dificultad de uso. Para finalizar se observa que el grupo EG1 se siente más seguro al manejar este tipo de herramientas, como se observa en las preguntas Q9 y Q10, aun cuando su nivel de formación todavía se encuentra en un nivel bajo, lo cual puede ocurrir debido al propio ámbito de su formación.

5.3.2. Pruebas funcionales

Para las pruebas funcionales se consideró dos acciones: (a) validar el funcionamiento de los artefactos de software generados, al ejecutarse sobre las plataformas de hardware y software del sistema IoT; y (b) analizar mediante Puntos de Función los artefactos generados y el tiempo necesario para crearlos, con le fin de comprara la mejora en el tiempo de desarrollo que implica emplear herramientas basadas en modelos para la generación automática o semi automática de código.

5.3.2.1. Funcionamiento

La Figura 5.14 ilustra la ejecución de la aplicación de Hogar Inteligente, en la que se muestra la lectura del sensor de CO_2 y de temperatura.

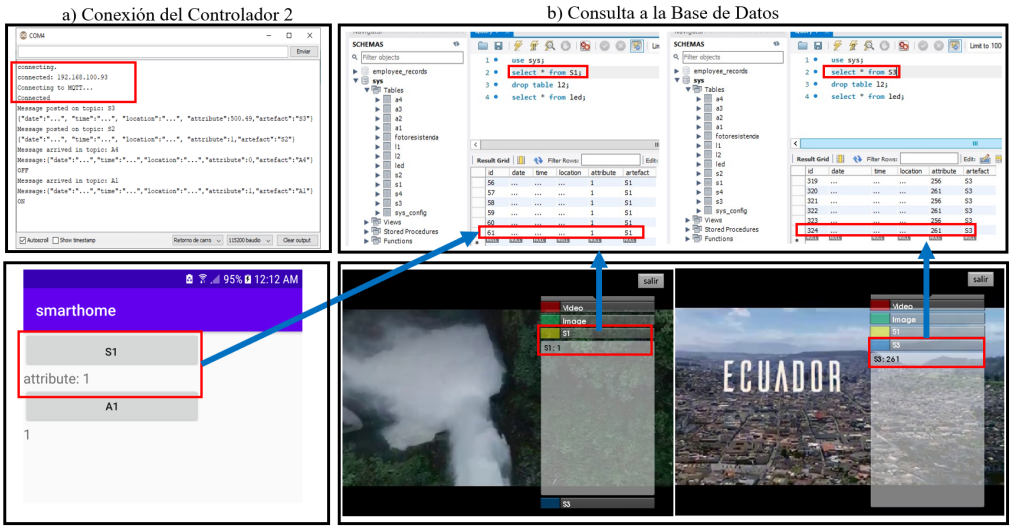


Figura 5.14: Prueba de funcionamiento del sistema IoT.

Como se observa en la Figura 5.14, esta consta de cuatro partes:

— En el cuadro superior izquierdo de la figura (a) se muestra una captura de pantalla de la consola del Controlador 2, donde se puede observar la conexión al punto de acceso y al Broker MQTT.

— En el cuadro superior derecho (b) se muestran las consultas realizadas a la base de datos para visualizar que los valores consultados por las interfaces sean los últimos.

— En el cuadro inferior izquierdo (c) se muestra una pantalla de la aplicación móvil, con el valor de consulta del sensor de apertura (S1).

— Finalmente, en el cuadro inferior derecho (d) se muestra dos capturas de pantalla de la ejecución de la aplicación DTV, la cual permite consultar los valores de los sensores (S1) y (S3).

5.3.2.2. Costes del desarrollo (Puntos de Función)

Para evaluar los costes del desarrollo de los artefactos de software generados, se consideró el Análisis de Puntos de Función (FPA, Function Point Analisys), con el cual se puede estimar las horas o días laborales [Ferrucci et al., 2014], que se requerirían para el desarrollo de la aplicación.

La Tabla 5.5 muestra las consideraciones realizadas para el cálculo total de los puntos de función no ajustados para el escenario propuesto, en base a los valores estándar IFPUG (International Function Point Users Grup). Para determinar el factor de ajuste se debe considerar los niveles de influencia que tienen las características de la Tabla 5.6 en el programa IoT.

Tipo	**Complejidad**	**Puntos de Función (PF)**	**Aplicación**	**Número**	**PF sin ajustar**
EI Entrada Externa	Baja	3	Lectura del sensor	4	12
EO Salida Externa	Baja	4	Escritura en actuadores	4	16
EQ Consulta Externa	Media	4	Consulta de la base de datos con servicios web	8	32
ILF Archivo Lógico Interno	Alta	15	Archivo de lógica de negocio del patrón de integración	2	30
EIF Archivo de Interfaz externo	Media	7	Interfaz DTV y Móvil	2	14
				PFSA	**104**

Tabla 5.5: Análisis de Puntos de Función para el escenario IoT.

Factor de Ajuste	**Puntuación**	**Descripción**
Comunicación de datos	5	La aplicación es más que una entrada on-line y soporta más de un protocolo.
Procesamiento distribuido	3	Procesamiento distribuido y a transferencia de datos son on-line, en apenas una dirección.
Objetivos de rendimiento	3	El tiempo de respuesta y volumen de procesamiento son items críticos durante todo el horario comercial.Ninguna determinación especial para la utilización del proceso fue establecida.
Configuración del equipamiento	1	Existen restricciones operacionales leves. No es necesario un esfuerzo especial para resolver estas restricciones.
Tasa de transacciones	0	No están previstos períodos pico de volumen de transacción.
Entrada de datos en línea	5	Más del 30 % de las transacciones son entradas de datos online.
Interface con el usuario	1	Las funciones online del sistema hacen énfasis en la amigabilidad del sistema y su facilidad de uso, buscando aumentar la eficiencia del usuario final.
Actualizaciones en línea	0	La aplicación posibilita la actualización online de los archivos lógicos internos.
Procesamiento complejo	1	El procesamiento complejo es una de las características de la aplicación.
Reusabilidad del código	5	La aplicación fue específicamente proyectada y/o documentada para tener su código fácilmente reutilizable por otra aplicación y la aplicación es configurada para uso a través de parámetros que pueden ser alterados por el usuario.
Facilidad de implementación	1	Ninguna consideración especial fue establecida por el usuario, más procedimientos especiales son requeridos en la implementación.
Facilidad de operación	0	Ninguna consideración especial de operación, además del proceso normal de respaldo establecido por el usuario.
Instalaciones múltiples	3	La necesidad de múltiples locales fue considerada en el proyecto y la aplicación está separada para trabajar sobre diferentes ambientes de hardware y/o software.
Facilidad de cambios	0	La aplicación fue específicamente proyectada y diseñada con vistas a facilitar su mantenimiento.
FA	**28**	

Tabla 5.6: Análisis de Puntos de Función ajustados para el escenario IoT.

Para calcular los puntos de función ajustados se realizó siguiente expresión:

$$PFA = PFSA * (0,65 + (0,01 * FA)) \tag{5.3}$$

donde:

FA: Factor de Ajuste
PFSA: Punto de Función sin Ajustar
PFA: Punto de Función Ajustado

Reemplazando los valores se obtiene:

$$PFA = 104 * (0,65 + (0,01 * 28)) = 96,72 \equiv 97 \tag{5.4}$$

Para realizar la estimación del esfuerzo requerido se consideró los valores de la Tabla 5.7, con los cuales se realizó los cálculos de la estimación.

Lenguaje	**Horas PF promedio**	**Líneas/PF**
Lenguaje de 4ta generación	8	20

Tabla 5.7: Parámetros para los lenguajes de programación del escenario IoT.

Finalmente, para calcular la duración (cantidad estimada de horas/hombre) para el desarrollo de la aplicación se considera la siguiente expresión:

$$D = PFA * HPFP \tag{5.5}$$

donde:

PFA: Punto de Función Ajustado
HPFP: Horas PF promedio
D: Horas hombre

Reemplazando los valores se obtiene:

$$D = 97 * 8 = 776 \tag{5.6}$$

Para calcular la cantidad estimada de líneas de código de la aplicación se considera la siguiente expresión:

$$LC = PFA * LFP \tag{5.7}$$

donde:

PFA: Punto de Función Ajustado
LFP: Líneas de código por PF
LC: Líneas de código

Reemplazando los valores se obtiene:

$$LC = 97 * 20 = 1940 \tag{5.8}$$

Para comparar el valor estimado de líneas de código se determinó la cantidad de líneas de código generadas sin considerar las librerías `tcp.Lua` y `ConnectorBase.ncl` ni los archivos multimedia que se muestran en Figura 5.15 y se resume en la Tabla 5.8. Se determinó que el total de líneas de código fue de 5058 para todos los artefactos de software generados; lo cual, comparado con el valor estimado de líneas de código, es un 260.7 % superior, debido a la heterogeneidad de las plataformas de desarrollo empleadas.

Para comparar el valor estimado de horas hombre se trabajó con los grupos experimentales EG1 y EG2. En el caso del grupo EG1, el promedio de tiempo que dedicaron los estudiantes para el modelado de un escenario de similares características con las herramientas fue de 6 horas. En comparación, el tiempo promedio para el grupo EG2 que

Artefactos software	Lenguaje de programación	Nombre de los artefactos	LDC
2	Arduino (Hardware)	`Controlador1.ino`, `Controlador2.ino`	361
8	Node-Red (Bridge)	`A1.json`, `A2.json`, `A3.json`, `A4.json`, `S1.json`, `S2.json`, `S3.json`, `S4.json`	716
8	Ballerina (API RESTful)	`A1.bal`, `A2.bal`, `A3.bal`, `A4.bal`, `S1.bal`, `S2.bal`, `S3.bal`, `S4.bal`	2832
1	NCL (DTV)	`SmartHome.ncl`	534
2	Lua (DTV)	`S1.lua`, `S2.lua`	133
1	Java (Móvil)	`AppMovilActivity.java`	66
2	XML (Movil)	`activity_appmovil.xml`, `AndroidManifest.xml`	77
2	Ballerina (Orquestador)	`ControlA1.bal`,`ControlA2.bal`	339
		TOTAL	**5058**

Tabla 5.8: Resumen de los artefactos software generados.

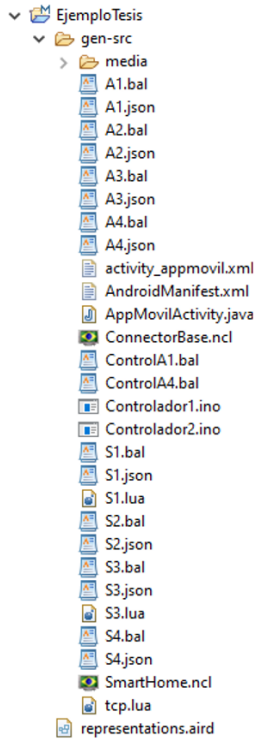


Figura 5.15: Artefactos de software generados para el escenario IoT.

fue de 4'5 horas. Cabe mencionar que no se ha considerado el tiempo que se destinó a la instalación de la herramienta ni a la explicación del funcionamiento. Como se observa al comparar con el tiempo estimado que le tomaría a un desarrollador implementar de forma manual la aplicación IoT existe una reducción notoria al reducirse de las 776 horas estimadas a 6 con el grupo EG1 y a 4'5 horas con el grupo EG2.

5.3.3. Pruebas de rendimiento

Para determinar el rendimiento de los servicios RESTful creados, se ha utilizado Gatling, que es un framework de pruebas de carga y rendimiento [Maila-Maila et al., 2019]. Para la evaluación se han propuesto 4 escenarios con 1000, 2000, 5000 y 10000 usuarios concurrentes. En cada escenario, cada usuario realiza 6 consultas, una para cada uno de los métodos implementados (*ie*, `GET /uri/last`, `GET /uri/{id}`, `GET /uri/all`, `POST /uri`, `PUT /uri/{id}`, `DELETE /uri/{id}`) y cada consulta separada por un intervalo de 5 segundos. A continuación, el Listing 5.1 muestra la estructura del código empleado para los escenarios evaluados.

```
import scala.concurrent.duration._
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._
class Test1 extends Simulation {
  val httpProtocol = http
    .baseUrl("...[host]...")
    .acceptHeader("*/*")
    .acceptEncodingHeader("gzip, deflate")
    .acceptLanguageHeader("es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3")
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101
     Firefox/73.0")
 val headers_1 = Map("Content-Type" -> "text/plain;charset=UTF-8")
 val scn = scenario("Test1")
    .exec(http("request_0")
      .get("/...[uri].../last"))
    .pause(5)
    .exec(http("request_1")
      .post("/... [uri]...")
      .headers(headers_1)
      .body(RawFileBody("/test1/0001_request.txt")))
    .pause(5)
    .exec(http("request_2")
      .get("/...[uri].../all"))
    .pause(5)
    .exec(http("request_3")
      .put("/...[uri].../66")
      .headers(headers_1)
      .body(RawFileBody("/test1/0003_request.txt")))
    .pause(5)
    .exec(http("request_4")
      .get("/...[uri].../66"))
  .pause(5)
  .exec(http("request_5")
      .delete("/...[uri].../66")
      .headers(headers_1)
      .body(RawFileBody("/test1/0005_request.txt")))
 setUp(scn.inject(atOnceUsers(...[#]...))).protocols(httpProtocol)
}
```

Listado 5.1: Fragmento de código de Gatling para las pruebas de carga.

La Figura 5.16 ilustra el resultado en cada escenario, en términos de respuestas fallidas y tiempos de respuesta:

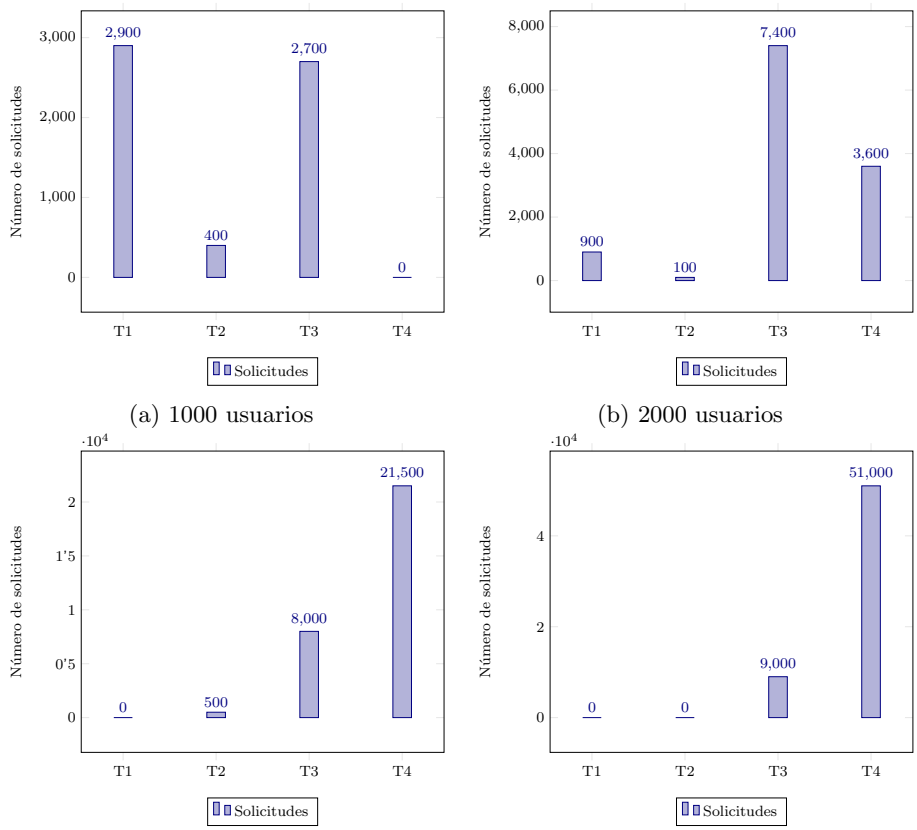
— Escenario con 1000 usuarios: El servicio es capaz de manejar todas las consultas sin perder ninguna de las 6000 solicitud. El tiempo de respuesta oscila entre 0'175 a 36'269 segundos.

— Escenario con 2000 usuarios: El servicio pierde 3600 solicitudes de las 12000 que se envían. Los tiempos de respuestas oscilan entre 0'173 a 60'027 segundos.

— Escenario con 5000 usuarios: El servicio pierde 21500 solicitudes de las 30000 que se envían. Los tiempos de respuestas oscilan entre 0'771 a 60'817 segundos.

— Escenario con 10000 usuarios: El servicio pierde 51000 solicitudes de las 60000 que se envían. Los tiempos de respuestas oscilan entre 0'507 a 118'344 segundos.

Como se observa incluso cuando aumentan las pérdidas de las solicitudes, la comunicación no se pierde por completo. Esto es importante en escenarios que tienen muchas conexiones como ocurre en el IoT, en los que hay una gran cantidad de objetos (sensores y actuadores) y servicios interconectados.



(a) 1000 usuarios (b) 2000 usuarios

(c) 5000 usuarios (d) 10000 usuarios

Figura 5.16: Resumen de resultados en todos los escenarios (T1: t < 800ms, T2: 800ms< t <1200ms, T3: t > 1200ms, T4: Perdido).

5.4. Trabajos relacionados

Para determinar las investigaciones relevantes se siguió la metodología descrita en el Capítulo 1, para lo cual se estableció una lista con los términos principales de búsqueda: IoT, Test, hardware, software, usuarios y herramientas. Con los trabajos seleccionados se realizó un resumen con el aporte de cada uno y una Tabla 5.9 para visualizar de forma rápida los aportes.

	IoT	**Test**	**Hardware**	**Software**	**Usuarios**	**Herramientas**
[Bosmans et al., 2019]	✓	✓	✓		✓	
[Hagar, 2018]	✓	✓		✓		
[Medhat et al., 2019]	✓	✓	✓	✓		
[Bures, 2017]	✓	✓		✓		
[Murad et al., 2018]	✓	✓	✓	✓		
[Patel et al., 2019]	✓	✓		✓		✓
[Boubeta-Puig et al., 2015]				✓		✓
Propuesta	✓	✓	✓	✓	✓	✓

Tabla 5.9: Resumen de las principales características cubiertas.

En [Bosmans et al., 2019] los autores presentan una descripción de los desafíos que surgen al probar aplicaciones IoT a nivel de sistema. Para lo cual, analizan el comportamiento de los dispositivos IoT y personas que interactúan con el sistema. Las interacciones de estas entidades conducen a un comportamiento emergente, tanto el comportamiento emergente como el comportamiento local deben tenerse en cuenta al probar los sistemas de IoT. Los autores plantean un enfoque de prueba basado en simulación híbrida para facilitar las interacciones de las entidades locales. Aun cuando el trabajo aborda la interacción con las personas, no se aborda las emociones de las personas ni el rendimiento de los servicios web del sistema.

Hagar et al. [Hagar, 2018] en su trabajo describen los desafíos, las implicaciones para las pruebas y los riesgos de IoT. Además, proporciona algunas definiciones, ejemplos de arquitectura y entornos de prueba de IoT a gran escala. Los autores abordan la necesidad de realizar test en los sistemas IoT, sin embargo, solo lo realizan de forma teórica por lo que no se aborda ninguna técnica específica.

Los autores en [Medhat et al., 2019] presentan un estudio de las principales técnicas y herramientas de prueba que se han considerado para los sistemas basados en IoT. En este documento, los autores abordan las principales técnicas de prueba, niveles, tipos y herramientas que se han considerado para los sistemas basados en IoT en los tres dominios de aplicación de IoT dominantes; salud y medicina, ciudades inteligentes y agricultura de precisión. Se llevan a cabo una comparación detallada y una evaluación analítica, de los diferentes tipos de pruebas que se han aplicado para los principales dominios de aplicación.

En el trabajo [Bures, 2017] los autores abordan los problemas de interoperabilidad e integración, mediante un enfoque híbrido de simulación de infraestructura de IoT y prueba de integración. En la solución propuesta, los autores combinan las ideas de simulación, integración de unidades y pruebas de integración E2E (End-toEnd). El Framework de

prueba de integración se basa técnicamente en JUnit. Debido a la amplia base de usuarios y el tamaño de la comunidad de desarrollo de JUnit. Sin embargo, en este trabajo, no se aborda la interacción con los usuarios.

En [Murad et al., 2018] los autores analizan las pruebas de software y herramientas para dispositivos loT, y proporcionan detalles de las pruebas de casos de uso para el entorno de IoT, usabilidad, seguridad y conectividad. Los autores describen las herramientas y los tipos de pruebas para sistemas IoT, sin embargo, solo lo realizan de forma teórica por lo que no se aborda ninguna técnica específica.

Los autores en [Patel et al., 2019] comparan dieciséis simuladores (IOTSim, IoTIFY, Bevywise-IoT, etc.), cuatro emuladores (Cooja, NetSim, MAMMotH, NCTUns 6.0) y seis bancos de pruebas (MBTAAS, FIT IoT-LAB, etc.) para IoT, considerando la base del alcance, tipo, lenguaje de programación, capas de IoT, escala de operación, estándares de IoT, integración de API, ciber-resiliencia, dominio de servicio y medidas de seguridad. Los autores abordan las herramientas y técnicas que permiten la validación de sistemas IoT, sin embargo, no abordan la interacción con el usuario.

En [Boubeta-Puig et al., 2015] los autores proponen una solución dirigida por modelos para la toma de decisiones en tiempo real de servicios dirigidos por eventos. Este enfoque permite la integración de CEP con este tipo de arquitectura, así como definir el dominio CEP y el patrón de eventos a través de un editor gráfico e intuitivo, que también permite la generación automática de código. Además, se evalúa la usabilidad de la solución propuesta. Aun cuando el trabajo no aborda el IoT directamente, CEP es una temática que puede complementar la funcionalidad del IoT.

5.5. Resumen y conclusiones

En este capítulo se proporciona los resultados del análisis de la experiencia de los usuarios, usabilidad y rendimiento. En él se muestra cómo los usuarios que han experimentado con las herramientas creadas aumentan su interés después de usarla y confirman que la herramienta es fácil de usar y que cumple con las expectativas. A partir de la experiencia realizadas sobre dos grupos de estudiantes de dos titulaciones de ingenierías, se observó que, en promedio, los mejores resultados se obtuvieron en uno de los dos grupos por la mayor cercanía de estos estudiantes con el área de la Informática.

Se observa que el tiempo estimado para la generación de los artefactos software se reduce a 6 horas para desarrolladores con bajo nivel de conocimiento en informática y electrónica en uno de los grupos, y a 4'5 horas para desarrolladores con nivel medio de conocimiento en el otro, frente a las 776 horas estimadas que podría necesitar un desarrollador convencional que realice la aplicación de forma manual sin usar la metodología propuesta. Esto pone de manifiesto la principal ventaja de utilizar técnicas MDE, la de acortar los tiempos de desarrollo, ya que estas técnicas permiten abstraer las características específicas de los lenguajes de programación, sin requerir que el desarrollador conozca en profundidad todas las plataformas.

Además, se pudo comprobar que los servicios tienen un buen rendimiento cuando aumenta la carga de solicitudes, lo que es muy importante en los sistemas IoT que pueden llegar a tener grandes cantidades de nodos.

Capítulo 6

Resultados

Capítulo 6

Resultados

Contenidos

El constante avance de la tecnología para la integración hardware y software amplía la posibilidad de nuevas aplicaciones. Sin embargo, los desarrolladores y los usuarios no solo deben adaptarse a las tecnologías emergentes, sino también adoptar nuevas tendencias de desarrollo. En este sentido, los sistemas IoT se ven afectados por este continuo cambio que vincula cada vez más al hardware y el software. Por ello, en el presente trabajo de investigación se propone una forma de generar aplicaciones IoT basadas en modelos y servicios que facilite esta labor a los desarrolladores. Para lo cual, se propone una metodología que establece dos fases de desarrollo: un proceso de especificación basado en modelos y servicios, en la cual se construyen las herramientas por parte de un especificador, y un proceso de desarrollo en el cual se crean los modelos para las aplicaciones IoT con las herramientas construidas.

En el presente capítulo se resumen los resultados obtenidos durante el desarrollo de este trabajo, el cual se divide en cuatro partes. En primer lugar, la Sección 6.1 describe la contribución de la investigación realizada a la comunidad científica. Posteriormente, la Sección 6.2 determina las limitaciones de la metodología propuesta. La Sección 6.3 presenta las líneas abiertas. Finalmente, en la Sección 6.4 se resume el listado de publicaciones derivadas de la investigación realizada.

6.1. Aportaciones a la comunidad científica

La principal contribución del trabajo de investigación desarrollado es una metodología basada en modelos y servicios para el diseño de aplicaciones IoT. Para lo cual se aborda los desafíos que enfrentan los desarrolladores en la construcción de aplicaciones interoperables en un contexto de plataformas de software y hardware, protocolos de comunicación y datos altamente heterogéneos. A continuación, se describen las aportaciones específicas derivadas de este trabajo de investigación:

(a) Se definió una metodología basada en técnicas de MDE, para el modelado de aplicaciones IoT. La metodología propone una arquitectura de integración que consta de tres capas (Física, Lógica y Aplicación) que permite la interconexión de los nodos de hardware y los servicios empresariales de las aplicaciones. Esta interconexión se realiza por medio de un componente que vincula la mensajería que emplean los nodos de hardware (para enviar y recibir información de los eventos de los sensores y actuadores); con los servicios RESTful y los servicios de integración (que proporcionan la lógica de negocio a la aplicación). Para lo cual, la metodología propone un formato de información simplificado que emplean todos los componentes del sistema y se crean funciones de control para la coordinación de todos los servicios. Por último, esta arquitectura permite vincular interfaces distintas a la web, como son la DTV y la Móvil, con las cuales el usuario final puede interactuar con el sistema IoT [Alulema et al., 2017] [Alulema et al., 2021] [Alulema et al., 2018a] (Capítulo 3).

(b) Se ha construido un DSL para la definición de los componentes principales de la arquitectura propuesta. Para lo cual se construye un metamodelo (sintaxis abstrac-

ta), un editor gráfico (sintaxis concreta) y una transformación M2T. Se usó EMF, para el desarrollo del metamodelo; Sirius, para la construcción del editor gráfico; y Acceleo, para las transformaciones M2T. Con estas herramientas se consigue un proceso semi-automático que permite a los desarrolladores crear aplicaciones IoT sin ser expertos en todas las plataformas que intervienen en la aplicación. Como resultado, se consigue el conjunto de artefactos software que se deberán posteriormente desplegar en cada una de las plataformas de hardware y software [Alulema et al., 2018b] [Alulema et al., 2019a] [Alulema et al., 2019b] [Alulema et al., 2019c] (Capitulo 4).

(c) Para la validación de la metodología propuesta, se ha implementado un escenario en el dominio del Hogar Inteligente, sobre el cual se realizaron tres tipos de pruebas: pruebas de usabilidad, pruebas funcionales y pruebas de rendimiento. Para el caso de las pruebas de usabilidad se trabajó con estudiantes de grado con los cuales se probaron las herramientas. En el caso de las pruebas funcionales, los artefactos software generados para el escenario de Hogar Inteligente se desplegaron sobre las plataformas Arduino, Ballerina, Node-Red, Android y Ginga NCL-Lua. Además, se comparó el tiempo que tarda un desarrollador en crear la aplicación con las herramientas generadas, respecto al tiempo estimado que le llevaría realizar la misma aplicación de forma manual. Por último, en el caso de las pruebas de rendimiento se realizaron pruebas de carga con Gatling a los servicios REST de sensores y actuadores, con el propósito de determinar la cantidad de solicitudes que pueden soportar estos servicios bajo una alta carga de demanda [Alulema et al., 2020] (Capítulo 5).

6.2. Limitaciones

A pesar de los aportes de la investigación, también hay una serie de características que pueden cuestionar la efectividad del trabajo propuesto, las cuales deben ser comentadas. De manera general, las limitaciones asociadas a la metodología propuesta tienen implicaciones en el modelo abstracto para regenerar una aplicación IoT:

(a) Aunque el metamodelo propuesto permite crear nodos hardware basados en un controlador con conectividad WiFi, no se ha considerado ninguna otra tecnología de conectividad. Esto se debe a que otros protocolos como SigFox, Lora, GSM, entre otros, requieren de licencias para usar sus infraestructuras de comunicación.

(b) Con respecto a los componentes hardware que se puede conectar al sistema, solo se consideró la opción de crear hardware *ad-hoc* para cada aplicación, por lo que ningún otro componente comercial, como los asistentes de Google o Alexa o sensores meteorológicos como de Netatmo, no podrían participar en el sistema sin un componente intermediario. Esto se debe a que la investigación pretende dar realce a la construcción del hardware, lo que permite realizar aplicaciones en dominios de aplicación diversos, como la agricultura, el hogar, el transporte, entro otros.

(c) Aun cuando el protocolo MQTT no es el único esquema para implementar mensajería, en esta investigación no se implementó otros esquemas. Esto se debe a que en la actualidad MQTT está siendo empleado en plataformas Cloud como Google,

AWS, Azure, además de plataformas industriales como Siemens en sus PLC (Programmable Logic Controller) Simatic IoT.

(d) Para la implementación de la lógica de control solo se ha considerado un esquema de control ON/OFF basado en condicionales o en funciones combinacionales. Sin embargo, en aplicaciones de procesos en tiempo real se emplean otro tipo de esquemas de control, como el control diferencial, integral, o difuso, entre otros. Esto se debe a que los escenarios en los cuales se ha centrado la investigación no son críticos y no requieren respuestas inmediatas o robustas, como ocurre el área médica o militar.

(e) Para el caso de la base de datos que emplean los servicios REST, solo se ha considerado una base de datos relacional, aun cuando debido al ámbito del IoT, actualmente se emplean bases de datos no relacionales. Esto se debe a que, de acuerdo con la metodología propuesta, se ha definido una estructura de información simplificada, con el fin de emplear menor capacidad de memoria, a diferencia de otras propuestas como la del W3C para la Thing Description. Esto se debe a que los nodos hardware poseen memoria limitada y si el sistema funciona mucho tiempo, se transmite información constantemente, o si existen demasiados nodos en la red, el controlador del nodo puede llegar a saturarse e inhibirse.

(f) Para el caso de las interfaces de usuario, la metodología propuesta no permite mayor versatilidad en los diseños, ya que esta se centra en el uso de interfaces sencillas tipo WIMP (Windows, Icons, Menus and Pointers) [Almendros and Iribarne, 2008] basada principalmente en botones y campos de texto que se comunican con los servicios REST, Orquestador o Bridge del sistema. Esta limitación se debe a que, conceptualmente, se puede considerar a un botón como un sensor, y un campo de texto como un actuador en sus formas más simples. Esta representación permite que la información enviada sea procesada como un valor discreto, con lo cual se puedan aplicar los esquemas para la lógica de control.

Aunque se han comentado algunas limitaciones, es necesario enfatizar que el propósito del análisis es delimitar el campo de aplicación y validez de la metodología desarrollada, además de encontrar las áreas que pueden mejorarse en futuros trabajos de investigación.

6.3. Líneas abiertas

A partir de la investigación que se ha realizado y del análisis de algunas de las limitaciones descritas, se han identificado una serie de áreas de investigación que aún están abiertas y pueden ser adoptadas como trabajos de investigación futuros:

(a) Ampliar el metamodelo para incorporar servicios de indexación y descubrimiento que permitan el uso de múltiples redes remotas para implementar aplicaciones más complejas e inteligentes. De esta forma, si determinados servicios no están disponibles en tiempo real, se puede solventar su caída con otros servicios equivalentes usando mecanismos de mediación [Iribarne et al., 2004] [Criado et al., 2021].

(b) La solución basada en modelos solo ha sido evaluada en el ámbito académico y de investigación. Puede ser de interés su extensión para aplicar la solución a la industria, con el fin de verificar los niveles de aceptación y satisfacción de los expertos comerciales. Esto permitirá incorporar aspectos relevantes, como la seguridad y disponibilidad de los servicios, los cuales son fundamentales en entornos de producción. Además, se podrían ampliar las pruebas de software para integración y calidad de software, con lo cual se podría mejorar el rendimiento de las aplicaciones y mejorar la seguridad del sistema.

(c) Con el propósito de que el sistema pueda soportar mayores cargas de tráfico, se pueden aplicar estrategias de balanceo de carga para manejar un gran número de usuarios concurrentes.

(d) También puede ser de interés, extender la propuesta para ir más allá de REST y servicios web, e incluir AMQ y Kafka. Esto permitiría aumentar la aplicabilidad de las herramientas a otros dispositivos IoT más diversos, ya que, al emplear buses empresariales como MULE permitiría ampliar la posibilidad de patrones de integración más robustos y complejos.

(e) Finalmente, se puede considerar la incorporación de otros mecanismos de control a la propuesta, como controles diferencial, integral, o difuso, entre otros mecanismos. Esto permitiría a la propuesta que pudiera ser empleada en aplicaciones críticas, donde el tiempo de respuesta requerido sea mucho menor, como ocurre en escenarios de control de vuelo de Drones, aplicaciones biomédicas o en el ámbito militar, entre otros muchos ejemplos.

6.4. Publicaciones derivadas de la investigación

En esta sección se presentan los artículos en revista, capítulos de libro y contribuciones en actas de congreso publicados durante la redacción de esta tesis doctoral. La siguiente lista muestra las publicaciones en orden cronológico:

(1) D. Alulema, L. Iribarne, J. Criado. (2017): A DSL for the Development of Heterogeneous Applications. 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW). Prague, Czech Republic, 21-23 August 2017. IEEE, pp., 251–257. DOI: `https://doi.org/10.1109/FiCloudW.2017.108`

(2) D. Alulema, J. Criado, L. Iribarne. (2018): Una propuesta de editor gráfico para el desarrollo de aplicaciones multiplaforma. Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2018), Sevilla, 17-20 septiembre 2018. DOI: `http://hdl.handle.net/11705/JISBD/2018/042`

(3) D. Alulema, J. Criado, L. Iribarne. (2018): A Cross-Device Architecture for Modelling Authentication Features in IoT Applications. Journal of Universal Computer Science, 24(12):1758–1775, 2018.ISSN: 0948-695X. DOI: `https://doi.org/10.3217/jucs-024-12-1758`

(4) D. Alulema, J. Criado, L. Iribarne. (2019): IoTV: Merging DTV and MDE Technologies on the Internet of Things. International Conference on Information Technology and Systems (ICITS'2019), Universidad de Las Fuerzas Armadas, Quito, Ecuador, 6-8 February 2019. Advances in Intelligent Systems and Computing, Vol. 918, pp. 255–264, Springer. ISBN: 978-3-030-11889-1. DOI: https://doi.org/10.1007/978-3-030-11890-7-25

(5) D. Alulema, J. Criado, L. Iribarne. (2019): A model-driven approach for the integration of hardware nodes in the IoT. 7th World Conference on Information Systems and Technologies (WorldCist'2019), La Toja Island, Galicia, Spain, 16-19 April 2019. Advances in Intelligent Systems and Computing, Volume 930, pp. 801–811, Springer. ISBN: 978-3-030-16180-4. DOI: https://doi.org/10.1007/978-3-030-16181-1-75

(6) D. Alulema, J. Criado, L. Iribarne. (2019): RESTIoT, A model-based approach for building RESTful web services in IoT systems. XXIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2019), SISTEDES, ISDM Ingeniería del Software Dirigida por Modelos. 2-4 septiembre. 2019, Cáceres, España. DOI: http://hdl.handle.net/11705/JISBD/2019/030

(7) D. Alulema, J. Criado, L. Iribarne, A.J. Fernández-García, R. Ayala. (2020): A model-driven engineering approach for the service integration of IoT systems. Cluster Computing 23:1937–1954, Springer. ISSN: 1386-7857. DOI: http://doi.org/10.1007/s10586-020-03150-x

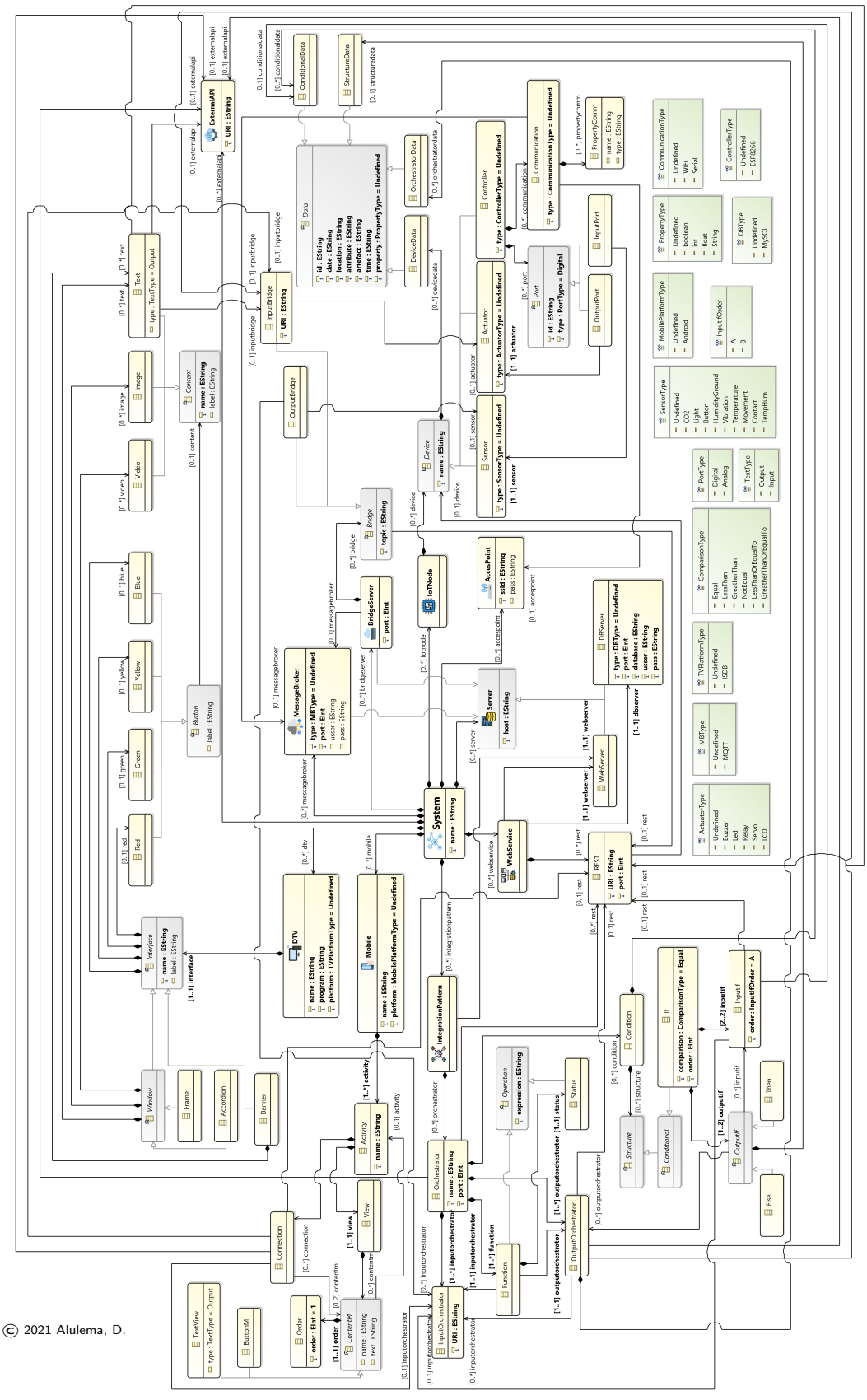
(8) D. Alulema, J. Criado, L. Iribarne. (2021): An approach to integrate IoT systems with no-web interfaces. International Conference on Information Technology and Systems (ICITS'21), February 10-12, 2021, Península de Santa Elena, Ecuador. DOI: https://doi.org/10.1007/978-3-030-68285-9_40

Anexo A

Metamodelo de la Metodología

Anexo A

Metamodelo de la metodología



Anexo B

Cuestionarios de evaluación

Anexo B

Cuestionarios de evaluación

Contenidos

En esta sección se adjunto como ejemplo uno de los cuestionario aplicados para las escalas SUS y PANAS.

B.1. PANAS

Instrucciones	Las respuestas son tratadas anónimamente y no tienen ninguna repercusión sobre las notas obtenidas en la asignatura. Como la experiencia es totalmente anónima, debes introducir un código (que solo tú sabrás), para poder compararlos. El código se formará introduciendo: Los tres últimos dígitos de tu CI + letra de tu Apellido + los tres últimos dígitos de tu número de teléfono. **Por ejemplo:** Si mi CI acaba en 334 y mi móvil acaba en 881, mi código sería 334A881.

Código	517G515

Sexo:	Masculino	✓	Femenino	
Edad:	20			

Lea cada ítem y luego marque la respuesta adecuada en el espacio próximo a cada palabra, en base a lo que usted siente en estos momentos utilizando la siguiente escala.

1. Nada
2. Muy poco
3. Algo
4. Bastante
5. Mucho

5	Interesado	2	Irritado
4	Dispuesto	2	Tenso
4	Animado	1	Avergonzado
2	Disgustado/enfadado	4	Inspirado
3	Enérgico	4	Nervioso
2	Culpable	4	Decidido
1	Temeroso	4	Atento
1	Enojado	1	Intranquilo
4	Entusiasmado	4	Activo
3	Orgulloso	2	Asustado

B.2. SUS

SUS IOT

Instrucciones	Las respuestas son tratadas anónimamente y no tienen ninguna repercusión sobre las notas obtenidas en la asignatura. Como la experiencia es totalmente anónima, debes introducir un código (que solo tú sabrás), para poder compararlos. El código se formará introduciendo: Los tres últimos dígitos de tu CI + letra de tu Apellido + los tres últimos dígitos de tu número de teléfono. **Por ejemplo:** Si mi CI acaba en 334 y mi móvil acaba en 881, mi código sería 334A881.

Código	217G515

Sexo:	Masculino	√	Femenino	
Edad:	20			

Por favor lea cuidadosamente las aseveraciones y marque con una X qué tan de acuerdo o qué tan en desacuerdo está con cada una de ellas. Registre su respuesta inmediata a cada una.

Parámetros:

- 5 Muy de acuerdo
- 1 No de acuerdo
- 3 No está seguro de que contestar

	1	2	3	4	5
1) Creo que me gustará usar el sistema frecuentemente.				X	
2) Encuentro el sistema innecesariamente complejo					X
3) Pensé que el sistema era fácil de usar.					X
4) Creo que necesitaré la ayuda de un técnico para poder usar el sistema.			X		
5) Encontré que las distintas funciones del sistema estaban bien integradas.				X	
6) Pensé que había mucha inconsistencia en el sistema.			X		
7) Me imagino que la gente aprenderá a usar el sistema rápidamente.			X		
8) Encuentro el sistema engorroso de usar.				X	
9) Me sentí muy seguro usando el sistema.			X		
10) Necesito aprender muchas cosas antes de poder utilizar el sistema					X

Acrónimos

Acrónimos

ALM	Application Lifecycle Management
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ASP	Active Server Pages
ATL	ATL Transformation Language
ATSC	Advanced Television Systems Committee
BPMN	Business Process Model and Notation
CEP	Complex Event Processing
CIM	Computation Independent Model
CoAP	Constrained Application Protocol
CRUD	Create, Read, Update and Delete
CSS	Cascading Stylesheets
DMB-T	Digital Multimedia Broadcast-Terrestrial
DSL	Domain-Specic Language
DTV	Digital TV
DVB-T	Digital Video Broadcasting Terrestrial
EDA	Event-Driven Architecture
EMF	Eclipse Modeling Framework
FPA	Function Point Analisys
GEM	Global Executable MHP
Ginga-CC	Ginga Common Core
HDTV	High Definition Television

HTML	HyperText Markup Language
IFPUG	International Function Point Users Grup
IoT	Internet of Things
IoTWF	IoT World Forum
IPC	Inter-Process Communication
ISDB-T	Integrated Services Digital Broadcasting
ITU	International Telecommunications Union
ITU-T	Telecommunication Standardization Sector of the ITU
JSON	JavaScript Object Notation
JSP	Java Server Pages
LGPL	Lesser General Public License
M2M	Model-to-Model
M2T	Model-to-Text
MCU	Micro-Controller Unit
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MDSD	Model-Driven Software Development
MOF	Meta-Object Facility
MQTT	Message Queue Server Telemetry Transport
NCL	Nested Context Language
OCL	Object Constraint Language
OMG	Object Management Group
PANAS	Positive and Negative Afect Schedule
PIM	Platform Independent Model
PLC	Programmable Logic Controller
PSM	Platform Specific Model
REST	REpresentational State Transfer
SBTVD-T	Sistema Brasileno de Television Digital Terrestre
SDTV	Standard Definition Television

SLR	Systematic Literature Review
SMS	Systematic Mapping Stud
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SUS	System Usability Scale
TV	Television
UFT	Unified Functional Testing
UML	Unified Modeling Language
URI	Uniform Resource Identier
W3C	World Wide Web Consortium
WoS	Web of Science
WoT	Web of Things
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

Bibliografía

Bibliografía

[Acceleo, 2020] Acceleo (2020). Acceleo. https://www.eclipse.org/acceleo/.

[Adamko, 2014] Adamko, A. (2014). Internet Tools and Services - Lecture Notes. https://gyires.inf.unideb.hu/GyBITT/08/index.html.

[Al-Osta et al., 2019] Al-Osta, M., Bali, A., and Gherbi, A. (2019). Event driven and semantic based approach for data processing on IoT gateway devices. *Journal of Ambient Intelligence and Humanized Computing*, 10(12):4663–4678. DOI:http://dx.doi.org/10.1007/s12652-018-0843-y.

[Almendros and Iribarne, 2008] Almendros, J. and Iribarne, L. (2008). An extension of UML for the modeling of WIMP user interfaces. *Journal of Visual Languages and Computing*, 19(6):695–720. DOI:https://doi.org/10.1016/j.jvlc.2007.12.004.

[Alonso et al., 2019] Alonso, A., Pozo, A., Choque, J., Bueno, G., Salvachua, J., Diez, L., Marin, J., and Alonso, P. L. C. (2019). An Identity Framework for Providing Access to FIWARE OAuth 2.0-Based Services According to the eIDAS European Regulation. *IEEE Access*, 7(July):88435–88449. DOI:http://dx.doi.org/10.1109/ACCESS.2019.2926556.

[Alulema et al., 2017] Alulema, D., Criado, J., and Iribarne, L. (2017). A DSL for the development of heterogeneous applications. *5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 251–257. DOI:https://doi.org/10.1109/FiCloudW.2017.108.

[Alulema et al., 2018a] Alulema, D., Criado, J., and Iribarne, L. (2018a). A Cross-Device Architecture for Modelling Authentication Features in IoT Applications. *Journal of Universal Computer Science*, 24(12):1758–1775. DOI:http://dx.doi.org/10.3217/jucs-024-12-1758.

[Alulema et al., 2018b] Alulema, D., Criado, J., and Iribarne, L. (2018b). Una Propuesta de Editor Grafico para el desarrollo de Aplicaciones multiplaforma IoT. *Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2018)*. DOI:http://hdl.handle.net/11705/JISBD/2018/042.

[Alulema et al., 2019a] Alulema, D., Criado, J., and Iribarne, L. (2019a). A model-driven approach for the integration of hardware nodes in the IoT. *Advances in Intelligent Systems and Computing*, 930(April):801–811.
DOI:https://doi.org/10.1007/978-3-030-16181-1_75.

[Alulema et al., 2019b] Alulema, D., Criado, J., and Iribarne, L. (2019b). IoTV: Merging DTV and MDE Technologies on the Internet of Things. *Advances in Intelligent Systems and Computing)*, 198(January):255–264.
DOI:https://doi.org/10.1007/978-3-030-11890-7_25.

[Alulema et al., 2019c] Alulema, D., Criado, J., and Iribarne, L. (2019c). RESTIoT: A model-based approach for building RESTful web services in IoT systems. *Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2019)*.
DOI:http://hdl.handle.net/11705/JISBD/2019/030.

[Alulema et al., 2021] Alulema, D., Criado, J., and Iribarne, L. (2021). An approach to integrate IoT systems with no-web interfaces. *In Advances in Intelligent Systems and Computing book series*, 1330(April).
DOI:https://doi.org/10.1007/978-3-030-68285-9_40.

[Alulema et al., 2020] Alulema, D., Criado, J., Iribarne, L., Fernández-García, A., and Ayala, R. (2020). A model-driven engineering approach for the service integration of IoT systems. *Cluster Computing*, 23(August):1937–1954.
DOI:http://doi.org/10.1007/s10586-020-03150-x.

[Angulo et al., 2011] Angulo, J., Calzada, J., and Estruch, A. (2011). Selection of standards for digital television: The battle for Latin America. *Telecommunications Policy*, 35(8):773–787.
DOI:http://dx.doi.org/10.1016/j.telpol.2011.07.007.

[Araujo et al., 2019] Araujo, V., Mitra, K., Saguna, S., and Åhlund, C. (2019). Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities. *Journal of Parallel and Distributed Computing*, 132(October):250–261.
DOI:https://doi.org/10.1016/j.jpdc.2018.12.010.

[AVSystem, 2020] AVSystem (2020). Shaping the world of connected devices. *Shaping the world of connected devices* – https://www.avsystem.com.

[Babau et al., 2009] Babau, J.-P., Blay-Formarino, M., Champeau, J., Robert, S., and Sabetta, A. (2009). *Model Driven Engineering for Distributed Real-Time Embedded Systems*. Wiley.

[Bettini, 2016] Bettini, L. (2016). *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing.

[Biswas and Giaffreda, 2014] Biswas, A. and Giaffreda, R. (2014). IoT and Cloud Convergence: Opportunities and Challenges. *IEEE World Forum on Internet of Things (WF-IoT)*, (April):375–376.
DOI:https://doi.org/10.1109/WF-IoT.2014.6803194.

[Blum, 2019] Blum, J. (2019). *Exploring Arduino*. Willey.

[Bormann, 2014] Bormann, C. (2014). *Internet of Things: Concepts and System Design*. Springer.

[Bosmans et al., 2019] Bosmans, S., Mercelis, S., Denil, J., and Hellinckx, P. (2019). Testing IoT systems using a hybrid simulation based testing approach. *Computing*, 101(7):857–872.
DOI:https://doi.org/10.1007/s00607-018-0650-5.

[Boubacar et al., 2016] Boubacar, B., Kamsu-Foguem, B., and Tangara, F. (2016). Integrating MDA and SOA for improving telemedicine services. *Telematics and Informatics*, 33(3):733–741.
DOI:http://dx.doi.org/10.1016/j.tele.2015.11.009.

[Boubeta-Puig et al., 2015] Boubeta-Puig, J., Ortiz, G., and Medina-Bulo, I. (2015). MEdit4CEP: A Model-Driven Solution for Real-Time Decision Making in SOA 2.0. *Knowledge-Based Systems*, 89(November):97–112.
DOI:http://dx.doi.org/10.1016/j.knosys.2015.06.021.

[Bouquet et al., 2014] Bouquet, F., Gauthier, J.-M., Hammad, A., and Peureux, F. (2014). Transformation of SysML Structure Diagrams to VHDL-AMS. *Second Workshop on Design, Control and Software Implementation for Distributed MEMS*, pages 74–81.
DOI:http://doi.org/10.1109/dMEMS.2012.12.

[Brambilla et al., 2018] Brambilla, M., Umuhoza, E., and Acerbis, R. (2018). Model-Driven Development of User Interfaces for IoT Systems via Domain-Specific Components and Patterns. *Journal of Internet Services and Applications*, 8(1):1–21.
DOI:http://doi.org/10.1186/s13174-017-0064-1.

[Broil et al., 2009] Broil, G., Paolucci, M., Wagner, M., Rukzio, E., Schmidt, A., and Hußmann, H. (2009). Perci: Pervasive service interaction with the internet of things. *IEEE Internet Computing*, 13(6):74–81.
DOI:https://doi.org/10.1109/MIC.2009.120.

[Broring et al., 2017] Broring, A., Schmid, S., Schindhelm, C., Khelil, A., Kabisch, S., Kramer, D., Le-Phuoc, D., Mitic, J., Anicic, D., and Teniente, E. (2017). Enabling IoT Ecosystems through Platform Interoperability. *IEEE Software*, 34(1):54–61.
DOI:http://doi.org/10.1109/MS.2017.2.

[Bruneliere et al., 2017] Bruneliere, H., Burger, E., Cabot, j., and Wimmer, M. (2017). A Feature-Based Survey of Model View Approaches. *Software and Systems Modeling*, (September):1–11.
DOI:https://doi.org/10.1007/s10270-017-0622-9.

[Burd and Mueller, 2020] Burd, B. and Mueller, J. (2020). *Android Application Development All-in-One For Dummies*. Willey.

[Bures, 2017] Bures, M. (2017). Framework for Integration Testing of IoT Solutions. *International Conference on Computational Science and Computational Intelligence*, pages 1838–1839.
DOI:https://doi.org/10.1109/CSCI.2017.335.

[Buyya and Srirama, 2019] Buyya, R. and Srirama, S. (2019). *Fog and Edge Computing*. Wiley.

[Cai et al., 2018] Cai, H., Gu, Y., Vasilakos, A., Xu, B., and Zhou, J. (2018). Model-Driven Development Patterns for Mobile Services in Cloud of Things. *IEEE Transactions on Cloud Computing*, 6(3):771–784.
DOI:https://doi.org/10.1109/TCC.2016.2526007.

[Castillo-Vergara et al., 2018] Castillo-Vergara, M., Alvarez-Marin, A., and Placencio-Hidalgo, D. (2018). A bibliometric analysis of creativity in the field of business economics. *Journal of Business Research*, 85(December):1–9.
DOI:https://doi.org/10.1016/j.jbusres.2017.12.011.

[Cedeno et al., 2013] Cedeno, E., Robles, T., Alcarria, R., and Morales, A. (2013). On the Characterization of Collaborative Mobile Services for the Internet of Things. *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 416–420.
DOI:http://dx.doi.org/10.1109/IMIS.2013.154.

[Chakraborty et al., 2018] Chakraborty, S., Mukherjee, S., Nag, T., Biswas, B., Garang, B., and Nayak, A. (2018). A low cost autonomous multipurpose vehicle for advanced robotics. *9th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2018*, pages 1067–1078.
DOI:http://dx.doi.org/10.1109/UEMCON.2018.8796543.

[Chakray, 2020] Chakray (2020). What is Ballerina? The new programming language for integrations. http://bit.ly/3puVmJX.

[Chen and Englund, 2017] Chen, L. and Englund, C. (2017). Choreographing Services for Smart Cities: Smart Traffic Demonstration. *IEEE Vehicular Technology Conference*, pages 1–5.
DOI:https://doi.org/10.1109/VTCSpring.2017.8108625.

[Cheng et al., 2017] Cheng, B., Wang, M., Zhao, S., Zhai, Z., Zhu, D., and Chen, J. (2017). Situation-Aware Dynamic Service Coordination in an IoT Environment. *IEEE/ACM Transactions on Networking*, 25(4):2082–2095.
DOI:https://doi.org/10.1109/TNET.2017.2705239.

[Cheng et al., 2018] Cheng, B., Zhao, S., Qian, J., Zhai, Z., and Chen, J. (2018). Lightweight Service Mashup Middleware With REST Style Architecture for IoT Applications. *IEEE Transactions on Network and Service Management*, 15(3):1063–1075.
DOI:https://doi.org/10.1109/TNSM.2018.2827933.

[Chin, 2020] Chin, R. (2020). *A DIY Smart Home Guide: Tools for Automating Your Home Monitoring and Security Using Arduino, ESP8266, and Android*. McGraw-Hill.

[Choi et al., 2019] Choi, S. C., Ahn, I. Y., Park, J. H., and Kim, J. (2019). Towards real-time data delivery in oneM2M platform for UAV management system. *International Conference on Electronics, Information, and Communication*, pages 21—23. DOI:https://doi.org/10.23919/ELINFOCOM.2019.8706417.

[Cirani et al., 2018] Cirani, S., Ferrari, G., Picone, M., and Veltri, L. (2018). *Internet of Thing*. Wiley.

[Confluent, 2020] Confluent (2020). What is CEP? Complex Event Processing Guide. https://www.confluent.io/learn/complex-event-processing/.

[Cooper and Kolovos, 2019] Cooper, J. and Kolovos, D. (2019). Engineering hybrid graphical-textual languages with sirius and xtext: Requirements and challenges. *International Conference on Model Driven Engineering Languages and Systems Companion*, pages 322–325. DOI:https://doi.org/10.1109/MODELS-C.2019.00050.

[Costa et al., 2016] Costa, L., Hira, C., Biase, M., and Soares, F. (2016). Cost-effective Hybrid Ginga-NCL Interactive Set-top Box Laisa. *International Symposium on Consumer Electronics Cost-effective*, pages 1–2. DOI:https://doi.org/10.1109/ISCE.2016.7797331.

[Crettaz and Dean, 2019] Crettaz, V. and Dean, A. (2019). *Event Streams in Action*. Manning Publications.

[Criado et al., 2021] Criado, J., Iribarne, L., and Padilla, N. (2021). Heuristics-based mediation for building smart architectures at run-time. *Computer Standards & Interfaces*, 75(April):103501. DOI:https://doi.org/10.1016/j.csi.2020.103501.

[Crowd-Machine, 2020] Crowd-Machine (2020). Enterprise Application Integration. https://www.crowdmachine.com/enterprise-application-integration/.

[Dai et al., 2009] Dai, N., Mandel, L., and Ryman, A. (2009). *Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional.

[Dar et al., 2015] Dar, K., Amir, T., Baraki, H., Eliassen, F., and Geihs, K. (2015). A Resource Oriented Integration Architecture for the Internet of Things: A Business Process Perspective. *Pervasive and Mobile Computing*, 20(July):145–159. DOI:https://doi.org/10.1016/j.pmcj.2014.11.005.

[Darabseh and Freris, 2019] Darabseh, A. and Freris, N. (2019). A Software-Defined Architecture for Control of IoT Cyberphysical Systems. *Cluster Computing*, 22(4):1107–1122. DOI:https://doi.org/10.1007/s10586-018-02889-8.

[Dhatri et al., 2019] Dhatri, D., Pachiyannan, M., Rani, J., and Pravallika, G. (2019). A Low-Cost Arduino based Automatic Irrigation System using Soil Moisture Sensor: Design and Analysis. *2nd International Conference on Signal Processing and Communication*, pages 104–108.
DOI:http://dx.doi.org/10.1109/ICSPC46172.2019.8976483.

[Di Rocco et al., 2020] Di Rocco, J., Di Ruscio, D., Härtel, J., Iovino, L., Lämmel, R., and Pierantonio, A. (2020). Understanding MDE projects: megamodels to the rescue for architecture recovery. *Software and Systems Modeling*, 19(2):401–423.
DOI:https://doi.org/10.1007/s10270-019-00748-7.

[DiMarzio, 2016] DiMarzio, J. (2016). *Beginning Android Programming with Android Studio*. Wrox.

[Dunbar, 2020] Dunbar, N. (2020). *Arduino Software Internals: A Complete Guide to How Your Arduino Language and Hardware Work Together*. Apress.

[Durán et al., 2013] Durán, F., Troya, J., and Vallecillo, A. (2013). *Desarrollo de software dirigido por modelos*. FUOC. Fundación para la Universitat Oberta de Catalunya.

[Díaz et al., 2016] Díaz, M., Martín, C., and Rubio, B. (2016). State-of-the-Art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing. *Journal of Network and Computer Applications*, 67:99–117.
DOI:https://doi.org/10.1016/j.jnca.2016.01.010.

[Emmerich, 2009] Emmerich, P. (2009). *Beginning Lua with World of Warcraft Addons*. Apress.

[Erl et al., 2012] Erl, T., Carlyle, B., Pautassi, C., and Balasubramaian, R. (2012). *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*. Prentice Hall.

[Etzion and Niblett, 2010] Etzion, O. and Niblett, P. (2010). *Event Processing in Action*. Manning Publications.

[Fernández, 2009] Fernández, P. (2009). Un Análisis Crítico de la Aproximación Model-Driven Architecture. Master's thesis, Universidad Complutense Madrid.

[Fernández-Isabel and Fuentes-Fernández, 2015] Fernández-Isabel, A. and Fuentes-Fernández, R. (2015). Analysis of intelligent transportation systems using model-driven simulations. *Sensors*, 15(6):14117—-14141.
DOI:https://doi.org/10.3390/s150614116.

[Ferrucci et al., 2014] Ferrucci, F., Gravino, C., and Sarro, F. (2014). Conversion from IFPUG FPA to COSMIC: Within-vs without-company equations. *Euromicro Conference Series on Software Engineering and Advanced Applications*, pages 293–300.
DOI:https://doi.org/10.1109/SEAA.2014.76.

[Fielding, 2020] Fielding, R. (2020). Representational State Transfer (REST). *Representational State Transfer (REST)* – https://bit.ly/2ZtY8od.

[García and Suárez, 2019] García, A. and Suárez, F. (2019). WOTPY: A Framework for Web of Things Applications. *Computer Communications*, 147(3):235–251. DOI:https://doi.org/10.1016/j.comcom.2019.09.004.

[García-Molina et al., 2012] García-Molina, J., García Rubio, F., Pelechano, V., Vallecillo, A., Vara, J. M., and Vicente-Chicote, C. (2012). *Desarrollo de Software Dirigido por Modelos*. Ra-Ma.

[Gartner, 2017] Gartner (2017). Leading the IoT. https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf.

[Ginga, 2020] Ginga (2020). Ginga. http://www.ginga.org.br/es.

[Gomes and Junqueira, 2012] Gomes, L. and Junqueira, S. (2012). *Programando em NCL 3.0*. Pontifícia Universidade do Rio de Janeiro (PUC-Rio).

[González-García et al., 2017] González-García, C., Meana-Llorián, D., G-Bustelo, C., Cueva, J., and Garcia-Fernandez, N. (2017). Midgar: Detection of People through Computer Vision in the Internet of Things Scenarios to Improve the Security in Smart Cities, Smart Towns, and Smart Homes. *Future Generation Computer Systems*, 76(November):301–313. DOI:http://dx.doi.org/10.1016/j.future.2016.12.033.

[Goubet, 2020] Goubet, L. (2020). Acceleo vs Acceleo vs Xpand. https://bit.ly/3jWvgOZ.

[Grace et al., 2016] Grace, P., Pickering, B., and Surridge, M. (2016). Model-Driven Interoperability: Engineering Heterogeneous IoT Systems. *Annales Des Telecommunications/Annals of Telecommunications*, 71(3-4):141–150. DOI:http://dx.doi.org/10.1007/s12243-015-0487-2.

[Granada et al., 2015] Granada, D., Moreno, , Juan, V., Verónica, B., and Marcos, E. (2015). CEViNEdit: mejorando el proceso de creación de editores gráficos cognitivamente eficaces con GMF Introducción. *Jornadas de Ingeniería del Software y Bases de Datos*. DOI:http://hdl.handle.net/11705/JISBD/2015/024.

[Guinard et al., 2010] Guinard, D., Trifa, V., and Wilde, E. (2010). A Resource Oriented Architecture for the Web of Things. *IEEE Trans. on Cloud Computing*, pages 1–8. DOI:http://dx.doi.org/10.1109/IOT.2010.5678452.

[Gupta, 2019] Gupta, A. (2019). *The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things*. Apress.

[Hagar, 2018] Hagar, J. D. (2018). Software test architectures and advanced support environments for IoT. *International Conference on Software Testing, Verification and Validation Workshops*, pages 252–256. DOI:https://doi.org/10.1109/ICSTW.2018.00057.

[Hames et al., 2017] Hames, D., Salgueiro, G., and Barton, R. (2017). *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things*. Cisco Press.

[Hohpe, 2003] Hohpe, G. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional.

[IBM, 2017a] IBM (2017a). Microservices: What's the Difference? `https://www.ibm.com/cloud/blog/soa-vs-microservices`.

[IBM, 2017b] IBM (2017b). Simplifique el desarrollo de sus soluciones de IoT con arquitecturas de IoT. `https://ibm.co/3jW23nx`.

[Iglesias-Urkia et al., 2020] Iglesias-Urkia, M., Gómez, A., Casado-Mansilla, D., and Urbieta, A. (2020). Automatic Generation of Web of Things Servients Using Thing Descriptions. *Personal and Ubiquitous Computing*, (July):1–17.
`DOI:https://doi.org/10.1007/s00779-020-01413-3`.

[IoTIFY, 2020] IoTIFY (2020). IoTIFY. `https://iotify.io/`.

[IoTWF, 2020] IoTWF (2020). Internet of Things World Forum - IoTWF Home. `https://bit.ly/3baZb1T`.

[Iribarne et al., 2004] Iribarne, L., Troya, J. M., and Vallecillo, A. (2004). A trading service for cots components. *The Computer Journal*, 47(3):342–357.
`DOI:https://doi.org/10.1093/comjnl/47.3.342`.

[Jazayeri et al., 2015] Jazayeri, M. A., Liang, S., and Huang, C. Y. (2015). Implementation and Evaluation of Four Interoperable Open Standards for the Internet of Things. *Sensors*, 15(9):24343–24373.
`DOI:https://doi.org/10.3390/s150924343`.

[Jie et al., 2013] Jie, Y., Pei, J. Y., Jun, L., Yun, G., and Wei, X. (2013). Smart home system based on IoT technologies. *International Conference on Computational and Information Sciences*, pages 1789–1791.
`DOI:https://doi.org/10.1109/ICCIS.2013.468`.

[Jung, 2007] Jung, K. (2007). *Beginning Lua Programming*. Wrox.

[Kahani and Cordy, 2015] Kahani, N. and Cordy, J. R. (2015). Comparison and Evaluation of Model Transformation Tools. *Software and Systems Modeling*, 24(3):1–42.
`DOI:https://bit.ly/3bcXCAD`.

[Kathiravelu et al., 2018] Kathiravelu, P., Roy, P., and Veiga, L. (2018). SD-CPS: Software-Defined Cyber-Physical Systems. Taming the Challenges of CPS with Workflows at the Edge. *Cluster Computing*, 8(November):1–17.
`DOI:https://doi.org/10.1007/s10586-018-2874-8`.

[Kharade et al., 2020] Kharade, M., Katangle, S., Kale, G. M., Deosarkar, S. B., and Nalbalwar, S. L. (2020). A NodeMCU based fire safety and air quality monitoring device. *International Conference for Emerging Technology*, pages 8–11. DOI:http://dx.doi.org/10.1109/INCET49848.2020.9153983.

[Khononov, 2019] Khononov, V. (2019). *EMF: eclipse modeling framework*. O'Reilly Media, Inc.

[Kitchenham, 2007] Kitchenham, C. (2007). *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. University of Durham.

[Krishnamurthi, 2018] Krishnamurthi, R. (2018). Teaching Methodology for IoT Workshop Course Using Node-RED. *International Conference on Contemporary Computing*, pages 1–3. DOI:https://doi.org/10.1109/IC3.2018.8530664.

[Kumar et al., 2019] Kumar, D., Maurya, R. K., and Dwivedi, K. (2019). IoT based home automation using computer vision. *International Journal of Innovative Technology and Exploring Engineering*, pages 5044–5047. DOI:https://doi.org/10.35940/ijitee.L3771.1081219.

[Kuncoro and Saputra, 2017] Kuncoro, L. and Saputra, P. (2017). Implementation of Air Conditioning Control System rotocol Based on NodeMCU ESP8266. *International Conference on Smart Cities, Automation & Intelligent Computing Systems*, pages 126–130. DOI:http://dx.doi.org/10.1109/ICON-SONICS.2017.8267834.

[Lapshina et al., 2019] Lapshina, P., Kurilova, S., and Belitsky, A. (2019). Development of an Arduino-based CO2 Monitoring Device. *IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering*, pages 595–597. DOI:http://dx.doi.org/10.1109/EIConRus.2019.8656915.

[Lea, 2020] Lea, P. (2020). *IoT and Edge Computing for Architects*. Wiley.

[Leitao et al., 2019] Leitao, B., Guedes, A., and Colcher, S. (2019). Toward Web Templates Support in Nested Context Language. *Communications in Computer and Information Science*, 1202(August):16–30. DOI:https://doi.org/10.1007/978-3-030-56574-9_2.

[Lekic and Gardasevic, 2018] Lekic, M. and Gardasevic, G. (2018). IoT sensor integration to Node-RED platform. *International Symposium on INFOTEH-JAHORINA*, pages 1–5. DOI:https://doi.org/10.1109/INFOTEH.2018.8345544.

[LoadRunner, 2020] LoadRunner (2020). LoadRunner. *LoadRunner Professional* – https://bit.ly/37oqkNY.

[Mahbub, 2020] Mahbub, M. (2020). A smart farming concept based on smart embedded electronics, Internet of Things and wireless sensor network. *Internet of Things*, 9(1):1–30.
DOI:http://dx.doi.org/10.1016/j.iot.2020.100161.

[Maila-Maila et al., 2019] Maila-Maila, F., Intriago-Pazmiño, M., and Ibarra-Fiallo, J. (2019). Evaluation of Open Source Software for Testing Performance of Web Applications. *Advances in Intelligent Systems and Computing*, 2(1):75–82.
DOI:https://doi.org/10.1007/978-3-030-16184-2.

[Mainetti et al., 2015] Mainetti, L., Mighali, V., and Patrono, L. (2015). A Software Architecture Enabling the Web of Things. *IEEE Internet of Things Journal*, 2(6):445–454.
DOI:https://doi.org/10.1109/JIOT.2015.2477467.

[Malyuga et al., 2020] Malyuga, K., Perl, O., Slapoguzov, A., and Perl, I. (2020). Fault Tolerant Central Saga Orchestrator in RESTful Architecture. pages 278–283.
DOI:http://doi.org/10.23919/FRUCT48808.2020.9087389.

[Margolis et al., 2020] Margolis, M., Jepson, B., and Weldin, N. (2020). *Arduino Cookbook*. O'Reilly Media, Inc.

[Medhat et al., 2019] Medhat, N., Moussa, S., Badr, N., and Tolba, M. F. (2019). Testing Techniques in IoT-based Systems. *International Conference on Intelligent Computing and Information Systems*, pages 394–401.
DOI:https://doi.org/10.1109/ICICIS46948.2019.9014711.

[Meier and Lake, 2018] Meier, R. and Lake, I. (2018). *Professional Android*. Wrox.

[Microsoft, 2020] Microsoft (2020). Saga distributed transactions pattern. https://bit.ly/2N3viJ3.

[Miller, 2015] Miller, M. (2015). *The Internet of Things: How Smart TVs, Smart Cars, Smart Homes, and Smart Cities Are Changing the World*. Que.

[Miori and Russo, 2014] Miori, V. and Russo, D. (2014). Domotic Evolution towards the IoT. *IEEE 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 809–814.
DOI:https://doi.org/10.1109/WAINA.2014.128.

[Monguel, 2018] Monguel, J. (2018). Una Arquitectura Orientada a Servicios y Dirigida Por Eventos Para El Control Inteligente de UAVs Multipropósito. Master's thesis, Universidad de Extremadura.

[Muniz, 2019] Muniz, B. (2019). *Hands-On RESTful Web Services with TypeScript 3*. Packt Publishing.

[Murad et al., 2018] Murad, G., Badarneh, A., Quscf, A., and Almasalha, F. (2018). Software Testing Techniques in IoT. *International Conference on Computer Science and Information Technology*, pages 17–21.
DOI:https://doi.org/10.1109/CSIT.2018.8486149.

[Nazari and Semsar, 2017] Nazari, A. and Semsar, A. (2017). Human interaction with IoT-based smart environments. *Multimedia Tools and Applications*, 76(11):13343––13365.
DOI:https://doi.org/10.1007/s11042-016-3697-3.

[NCL, 2020] NCL (2020). Nested Context Language. http://www.ncl.org.br/en/inicio.

[Nikaj et al., 2019] Nikaj, A., Weske, M., and Mendling, J. (2019). Semi-automatic derivation of RESTful choreographies from business process choreographies. *Softw. Syst. Model*, 18(2):1195–1208.
DOI:https://doi.org/10.1007/s10270-017-0653-2.

[OneM2M, 2020] OneM2M (2020). OneM2M. https://www.onem2m.org/.

[Oram, 2019] Oram, A. (2019). *Ballerina: A Language for Network-Distributed Application*. O'Reilly Media, Inc.

[Pasika and Gandla, 2020] Pasika, S. and Gandla, T. (2020). Smart water quality monitoring system with cost-effective using IoT. *Heliyon*, 6(7):1–9.
DOI:https://doi.org/10.1016/j.heliyon.2020.e04096.

[Patel et al., 2019] Patel, N., Mehtre, B., and Wankar, R. (2019). Simulators, Emulators, and Test-beds for Internet of Things: A Comparison. *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, pages 139–145.
DOI:https://doi.org/10.1109/I-SMAC47947.2019.9032519.

[PUC-Rio, 2020] PUC-Rio (2020). Lua. https://www.lua.org/.

[Pérez, 2012] Pérez, A. (2012). Generación de código Ada para aplicaciones embebidas y de tiempo real desde modelos dinámicos UML. Master's thesis, Universidad de Cantabria.

[Raibulet et al., 2017] Raibulet, C., Arcelli Fontana, F., and Zanoni, M. (2017). Model-driven reverse engineering approaches: A systematic literature review. *IEEE Access*, 5:14516–14542.
DOI:http://doi.org/10.1109/ACCESS.2017.2733518.

[Raj et al., 2017] Raj, P., Anupama, R., and Subramanian, H. (2017). *Architectural Patterns*. Packt Publishing.

[Ramgir, 2019] Ramgir, M. (2019). *Internet of Things*. Pearson Education India.

[RFC-Editor, 2014] RFC-Editor (2014). RFC 7228. https://bit.ly/2ZpVnUZ.

[Richardson, 2018] Richardson, C. (2018). *Microservices Pattern*. Manning Publications.

[Salihbegovic et al., 2015] Salihbegovic, A., Eterovic, T., Kaljic, E., and Ribic, S. (2015). Design of a domain specific language and IDE for Internet of things applications. *38th International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 996–1001.
DOI:https://doi.org/10.1109/MIPRO.2015.7160420.

[Sangsanit and Techapanupreeda, 2019] Sangsanit, K. and Techapanupreeda, C. (2019). NodeMCU Choreography Automation by CoAP. *International Conference on Information Networking*, pages 350–353.
DOI:https://doi.org/10.1109/ICOIN.2019.8718123.

[Savaridass et al., 2020] Savaridass, P., Ikram, N., Deepika, R., and Aarnika, R. (2020). Development of smart health monitoring system using Internet of Things. *Materials Today: Proceedings*, 1(March):1–4.
DOI:https://doi.org/10.1016/j.matpr.2020.03.046.

[Schachinger and Kastner, 2015] Schachinger, D. and Kastner, W. (2015). Model-driven integration of building automation systems into Web service gateways. *IEEE International Workshop on Factory Communication Systems*, pages 1–8.
DOI:https://doi.org/10.1109/WFCS.2015.7160561.

[Schlegel et al., 2013] Schlegel, C., Lotz, A., Lutz, M., Stampfer, D., Ingles-Romero, J., and Vicente-Chicote, C. (2013). Model-driven software systems engineering in robotics: Covering the complete life-cycle of a robot. *IT - Information Technology*, 57(2):2780–2794.
DOI:https://doi.org/10.1515/itit-2014-1069.

[SETID, 2020] SETID (2020). Televisión Digital. http://bit.ly/2O3Kjut.

[Sirius, 2020] Sirius (2020). Sirius. https://www.eclipse.org/sirius/.

[Sittón-Candanedo et al., 2019] Sittón-Candanedo, I., Alonso, R., García, , Muñoz, L., and Rodríguez-González, S. (2019). Edge computing, IoT and social computing in smart energy scenarios. *Sensors*, 19(15):1–20.
DOI:https://doi.org/10.3390/s19153353.

[Sneps-Sneppe and Namiot, 2015] Sneps-Sneppe, M. and Namiot, D. (2015). On web-based domain-specific language for Internet of Things. *Int. Cong. on Ultra. Modern Telecom. and Control Systems and Workshops*, pages 287–292.
DOI:https://doi.org/10.1109/ICUMT.2015.7382444.

[Soares et al., 2010] Soares, L. F., Rodrigues, R. F., Cerqueira, R., and Barbosa, S. D. J. (2010). Variable and state handling in NCL. *Multimedia Tools and Applications*, 50(3):465–489.
DOI:https://doi.org/10.1007/s11042-010-0478-2.

[Soares Neto et al., 2010] Soares Neto, C. d. S., Soares, L. F. G., and de Souza, C. S. (2010). The Nested Context Language reuse features. *Journal of the Brazilian Computer Society*, 16(4):229–245.
DOI:https://doi.org/10.1007/s13173-010-0017-z.

[Steinberg et al., 2008] Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson Education.

[Sthal et al., 2006] Sthal, T., Volter, M., Bettin, J., Hasse, A., Helsen, S., Czarnecki, K., and Stockfleth, B. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. Wiley.

[Stopford, 2018] Stopford, B. (2018). *Designing Event-Driven Systems*. O'Reilly Media, Inc.

[Sundharam et al., 2018] Sundharam, S., Navet, N., Altmeyer, S., and Havet, L. (2018). A model-driven co-design framework for fusing control and scheduling viewpoints. *Sensors*, 18(2):1–26.
DOI:http://doi.org/10.3390/s18020628.

[Szauer, 2018] Szauer, G. (2018). *Lua Quick Start Guide*. Packt Publishing.

[Sánchez et al., 2017] Sánchez, H., González-Contreras, C., Agudo, E., and Macías, M. (2017). IoT and iTV for interconnection, monitoring, and automation of common areas of residents. *Applied Sciences*, 7(7):1–17.
DOI:https://doi.org/10.3390/app7070696.

[Teixeira et al., 2017] Teixeira, S., Agrizzi, B., Gonzalves, J., Filho, P., Rossetto, S., and Baldam, R. (2017). Modeling and Automatic Code Generation for Wireless Sensor Network Applications Using Model-Driven or Business Process Approaches: A Systematic Mapping Study. *Journal of Systems and Software*, 132(October):50–71.
DOI:http://dx.doi.org/10.1016/j.jss.2017.06.024.

[Thramboulidis et al., 2019] Thramboulidis, K., Vachtsevanou, D., and Kontou, I. (2019). CPuS-IoT: A Cyber-Physical Microservice and IoT-Based Framework for Manufacturing Assembly Systems. *Annual Reviews in Control*, 47:1–12.
DOI:https://doi.org/10.1016/j.arcontrol.2019.03.005.

[Tobar, 2016] Tobar, E. (2016). Sistema de control de software e interfaz de usuario para control de microclimas de cultivo con gateway IoT. Master's thesis, Universidad Andres Bello. https://bit.ly/3rjjCjL.

[Torres et al., 2020] Torres, V., Serral, E., Valderas, P., Pelechano, V., and Grefen, P. (2020). Modeling of iot devices in business processes: A systematic mapping study. *Proceedings - 2020 IEEE 22nd Conference on Business Informatics*, pages 221–230.
DOI:https://doi.org/10.1109/CBI49978.2020.00031.

[University-Leiden, 2020] University-Leiden (2020). VOSviewer Vizualizing scientific landscapes. *VOSviewer* – https://www.vosviewer.com/.

[Urbieta et al., 2017] Urbieta, A., González-Beltrán, A., Mokhtar, B., Hossain, A., and L., C. (2017). Adaptive and Context-Aware Service Composition for IoT-Based Smart Cities. *Future Generation Computer Systems*, 76(November):771–784. DOI:https://doi.org/10.1016/j.future.2016.12.038.

[Valderas et al., 2019] Valderas, P., Torres, V., and Pelechano, V. (2019). A microservice composition approach based on the choreography of bpmn fragments. *Information and Software Technology*, 127(July):1–17. DOI:https://doi.org/10.1016/j.infsof.2020.106370.

[Varna, 2012] Varna, J. (2012). *Learn Lua for iOS Game Development*. Apress.

[Vujovic et al., 2014] Vujovic, V., Maksimovic, M., and Perisic, B. (2014). Comparative analysis of DSM Graphical Editor frameworks: Graphiti vs Sirius. *Erk14*, pages 7–10. DOI:https://bit.ly/2NgG1jb.

[W3C, 2020] W3C (2020). Web of Things at W3C. *W3C* – https://www.w3.org/WoT/.

[Wang et al., 2015] Wang, L., Torngren, M., and Onori, M. (2015). Current Status and Advancement of Cyber-Physical Systems in Manufacturing. *Journal of Manufacturing Systems*, 37(2):517–527. DOI:http://dx.doi.org/10.1016/j.jmsy.2015.04.008.

[Wiki, 2020] Wiki, F. (2020). FIWARE Wiki. http://forge.fiware.org.

[WINSYSTEMS, 2017] WINSYSTEMS (2017). Cloud, fog and edge computing – what's the difference? https://bit.ly/3dsnuLo.

[Zhang et al., 2014] Zhang, Y., Duan, L., and Chen, J. (2014). Event-Driven SOA for IoT Services. *IEEE International Conference on Services Computing*, pages 629–636. DOI:http://dx.doi.org/10.1109/SCC.2014.88.

[Zheng and Carter, 2015] Zheng, D. and Carter, W. (2015). *Leveraging the Internet of Things for a More Efficient and Effective Military*. Center for Strategic and Int. Studies.

[Zhou et al., 2018] Zhou, C., Feng, Y., and Yin, Z. (2018). An Algebraic Complex Event Processing Method for Cyber-Physical System. *Cluster Computing*, 3(March):1–9. DOI:http://dx.doi.org/10.1007/s10586-018-2522-3.

[Zolotas et al., 2017] Zolotas, C., Diamantopoulos, T., Chatzidimitriou, K., and Symeonidis, A. (2017). From Requirements to Source Code: A Model-Driven Engineering Approach for RESTful Web Services. *Automated Software Eng.*, 24(4):791–838. DOI:http://dx.doi.org/10.1007/s10515-016-0206-x.

Este documento ha sido generado con LaTeX.

Una metodología basada en modelos y servicios para la integración de sistemas IoT

A methodology based on models and services for the integration of IoT systems

Darwin Omar Alulema Flores
Departamento de Informática
Grupo de Investigación de Informática Aplicada (TIC-211)
Universidad de Almería
Almería, Marzo de 2021

http://acg.ual.es

Darwin Omar Alulema Flores

Una Metodología basada en Modelos y Servicios para la Integración de sistemas IoT

La tecnología se hace presente cada vez más en la vida cotidiana con nuevos productos con mayores prestaciones y con menores costes. Una de estas tecnologías es el Internet de las Cosas (IoT, Internet of Things), el cual ha capitalizado el desarrollo en las tecnologías de telecomunicaciones, electrónica e informática. Permite disponer de aplicaciones en áreas tan diversas como la industria, agricultura, transporte, hogar, entretenimiento, gobierno, entre otras. Sin embargo, este amplio espectro de aplicaciones tiene el problema de la diversidad de productos, la falta de estandarización y la dificultad de integración. Estos problemas limitan la creación de aplicaciones complejas debido a la mayor exigencia de conocimiento sobre varias tecnologías a un desarrollador.

En esta tesis se ha propuesto una metodología basada en técnicas de Ingeniería Dirigida por Modelos (MDE, Model-Driven Engineering) que permite solventar las dificultades presentes en el diseño de sistemas IoT. Uno de los objetivos de esta metodología es facilitar el proceso de desarrollo al establecer modelos que abstraen la complejidad tecnológica implícita a cada plataforma de despliegue. Como consecuencia, los desarrolladores pueden centrarse en la lógica de negocio de la aplicación. Como parte de la metodología, se han desarrollado las siguientes actividades: (a) definición de una arquitectura de integración para sistemas IoT con componentes heterogéneos, (b) definición de una metodología de implementación basado en MDE para la construcción de aplicaciones IoT, y (c) definición de un Lenguaje Específico de Dominio (DSL, Domain-Specific Language), para el modelado de Sistemas IoT.