UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Aplicando capacidades de Computación Autónoma a la plataforma Amazon AWS IoT

Trabajo Fin de Máster

Máster Universitario en

Ingeniería y Tecnología de Sistemas Software

Departamento de Sistemas Informáticos y Computación

Autor/a: Fenollar Onrubia, Daniel

Tutores: Fons Cors, Joan

Pelechano Ferragud, Vicente

Curso 2019-2020

Resumen

El crecimiento de la computación móvil junto con la Internet de las Cosas (IoT) está generando un aumento en la creación de dispositivos con mayores capacidades de comunicación y procesamiento. Esto lleva al desarrollo de soluciones informáticas ligadas a estos dispositivos físicos que nos permiten comunicar el mundo digital con el real, ya sea recopilando información o monitorizándolo. La naturaleza de las aplicaciones en este dominio es dinámica, es decir, se encuentra en cambio constante.

Amazon ofrece una plataforma de computación en la nube para el desarrollo de este tipo de soluciones, denominada Amazon AWS IoT. Esta pone a nuestra disposición una serie de servicios que nos ayudan a configurar, desplegar y establecer mecanismos e interfaces para la comunicación entre los dispositivos físicos a través de distintas reglas, protocolos, tecnologías de comunicación, etc. Nos centraremos en el servicio de Gestión de Dispositivos (AWS IoT Device Management) y haremos uso del concepto de Shadow Device.

Sin embargo, actualmente, la gestión y configuración de los dispositivos IoT, que interactuarán en una o varias aplicaciones, es fija. Es decir, solo se pueden realizar cambios manualmente, requiriendo que un desarrollador, operador o propietario de los mismos accedan y realicen dicha gestión.

El objetivo del trabajo es identificar que extensiones realizar sobre la infraestructura para permitir el desarrollo de soluciones más flexibles en la plataforma y conseguir una configuración y gestión dinámica. Haciendo uso de técnicas de computación autónoma basadas en bucles de control, aplicados de manera conceptual.

Finalmente, para la validación de la propuesta se describirán distintos escenarios donde aparezcan este tipo de necesidades, que no se pueden obtener con una solución AWS IoT actual. Desarrollando un pequeño prototipo funcional que simule el nuevo funcionamiento que la plataforma podría ofrecer tras la realización del proyecto.

Palabras clave: IoT, computación autónoma, Amazon Web Services.

Abstract

The growth of mobile computing along with the Internet of Things (IoT) is generating an increase in the creation of devices with greater communication and processing capabilities. This leads to the development of computer solutions linked to these physical devices that allow us to communicate the digital world with the real one, either by collecting information or monitoring it. The nature of applications in this domain is dynamic, that is, it is constantly changing.

Amazon offers a cloud computing platform for the development of these types of solutions, called Amazon AWS IoT. This puts at our disposal a series of services that help us configure, deploy and establish mechanisms and interfaces for communication between physical devices through different rules, protocols, communication technologies, etc. We will focus on the Device Management service (AWS IoT Device Management) and we will make use of the concept of Shadow Device.

However, currently, the management and configuration of IoT devices, which will interact in one or more applications, is fixed. In other words, changes can only be made manually, requiring that a developer, operator or owner access and carry out said management.

The objective of the work is to identify the extensions made on the infrastructure to allow the development of more flexible solutions on the platform and achieve dynamic configuration and management. Making use of autonomous computing techniques based on control loops, applied conceptually.

Finally, for the validation of the proposal, different scenarios will be described where these types of needs appear, which cannot be obtained with a current AWS IoT solution. Developing a small functional prototype that simulates the new operation that the platform could offer after the completion of the project.

Keywords: IoT, autonomous computing, Amazon Web Services.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

1. Introducción

El mundo IoT ha experimentado un crecimiento notable en la última década, dando paso a la creación de áreas de desarrollo en la vida cuotidiana como los hogares o ciudades inteligentes, agricultura e industria. Cualquier tipo de dispositivo u objeto puede formar parte del ecosistema IoT, ya sea una luz, un electrodoméstico o una máquina industrial. Todos estos dispositivos pueden comunicarse entre sí, pudiendo compartir información que recogen del mundo real y transmitirla a través del mundo digital, formando una red de dispositivos IoT.

Podemos definir IoT como una red que envuelve la configuración, el control y la red de dispositivos mediante Internet, utilizando una serie de protocolos específicos para la comunicación y sensores que permiten la obtención de datos e información del entorno que nos rodea. Una de las características más destacables de este mundo es la capacidad de conectar dispositivos completamente diferentes y permitir una comunicación sencilla y configurable de forma remota.

IoT nace con el objetivo de facilitar nuestras vidas haciéndolas más seguras, eficientes y cómodas redefiniendo la gestión de sistemas tanto críticos como no críticos. Pero no todo lo que nos proporciona este mundo es positivo, los sistemas atraen la atención de usuarios maliciosos que pretenden infiltrarse en la red y manipularla para su propio beneficio, estos son conocidos como ciber ataques.

Por otro lado, la computación autónoma también está en aumento debido, en gran parte, al crecimiento de los vehículos autónomos. Los sistemas son cada vez más complejos y con un gran tamaño, además, los clientes quieren un producto seguro, eficiente, fácil de utilizar y no desean estar involucrados de manera directa en el mantenimiento, funcionamiento o control del mismo.

En este trabajo se plantea aplicar los principios de la computación autónoma a las soluciones IoT utilizando el servicio dedicado que ofrece Amazon Web Services, añadiendo una capacidad al sistema para que los procesos o tareas importantes puedan ejecutarse sin la intervención humana, es decir, de manera autónoma.

Tras analizar los puntos en los que es posible la aplicación de la computación autónoma, se diseñará el bucle de control que nos permitirá esta aplicación y pasaremos al desarrollo de un pequeño prototipo en un caso de estudio concreto donde se visualizará el funcionamiento de una solución IoT lista para ser auto-adaptativa.

1.1. Motivación

Como hemos mencionado anteriormente, tanto el mundo de la IoT como la computación autónoma, son campos que actualmente se encuentran en constante desarrollo y evolución, siendo el dominio de la computación autónoma el más reciente.

Las soluciones IoT, al igual que cualquier solución software, se basan principalmente en los diseños monolíticos y tradicionales, que son los más utilizados y fáciles de desarrollar. Con la computación autónoma, conseguimos facilitar todavía más la vida de los clientes de estos productos, ya que nos aporta capacidades de autoadaptación, a cambio de un producto cuyo desarrollo y diseño final es más complejo.

La motivación que ha llevado a la creación de este proyecto es la investigación para la aplicación de comportamientos auto-adaptativos a las soluciones desarrolladas con la plataforma AWS IoT, de modo que, se desarrolle un pequeño caso de estudio en el que se prepare la base de una solución IoT utilizando dicha plataforma y dotándolo de autonomía para su propia gestión, pudiendo juntar ambos dominios en una solución única.

1.2. Objetivos

El objetivo del trabajo es mostrar como diseñar y desarrollar soluciones que permitan aplicar la computación autónoma, utilizando AWS IoT como plataforma para dichas soluciones. De esta forma, a diferencia de las soluciones IoT actuales, se consigue un desarrollo más flexible, además de una configuración y gestión dinámica.

De igual modo, marcaremos otro objetivo, cómo aplicarlo en un dominio como es el de las SmartCity. Para ello se hace uso de técnicas de computación autónoma, basadas en bucles de control, las cuales se describen de manera conceptual y se aplican a un pequeño escenario desarrollado por el autor.

1.3. Metodología y plan de trabajo

Este trabajo, aun teniendo una parte de desarrollo, es mayoritariamente teórico y, por tanto, se le ha dedicado un tiempo al estudio de las diferentes plataformas que ofrecen la creación de soluciones IoT, al análisis de dichas soluciones para la incorporación de capacidades autónomas y el diseño del bucle de control que posteriormente se desarrollará en el caso de estudio.

Hemos seguido GitFlow (1) como flujo de trabajo para el desarrollo del prototipo, este favorece el desarrollo en paralelo y la colaboración entre los distintos miembros del equipo. Además, se armoniza con la metodología de desarrollo ágil, debido a que permite crear productos de manera incremental por el importante papel que obtienen las ramas.

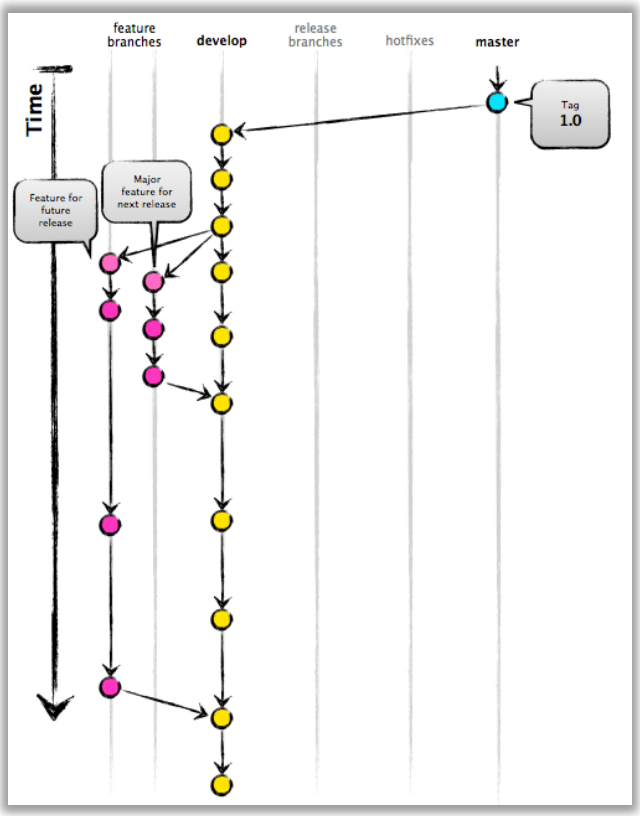

Ilustración 1. GitFlow.

El funcionamiento es el siguiente, partimos de una rama principal, denominada “master”, la cual contiene el producto que se muestra de manera incremental al cliente. A partir de esta rama, se crea otra que se utiliza para el desarrollo, llamada “develop”, donde se trabaja hasta finalizar las distintas funcionalidades, que a su vez se desarrollan en distintas ramas, las que llamamos “feature”.

Así, a medida que las funcionalidades se terminan, se vuelven a incorporar y fusionar con la rama “develop”. Finalmente, cuando se termina la iteración, se fusiona con la rama “master” donde se encuentra la versión del software establecida.

En la figura 2, vemos una captura de pantalla del programa de control de versiones SourceTree, donde observamos el flujo de trabajo que se ha realizado, distinguiendo las

diversas ramas por sus etiquetas y colores, junto con el grafo posicionado a la izquierda de los “commit”.

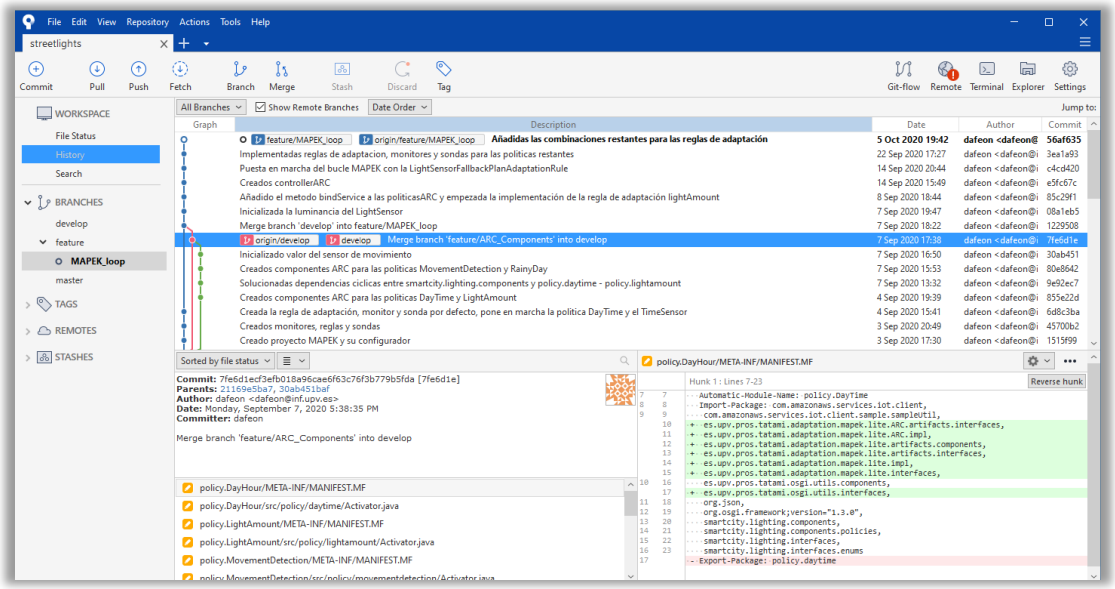

Ilustración 2. SourceTree.

Este trabajo es, principalmente, teórico, aunque se realiza un pequeño desarrollo donde se muestra la aplicación del mismo. Por tanto, se ha dividido el trabajo en bloques donde se divide el tiempo dedicado a diferentes tareas, con la finalidad de abordarlo de una manera más organizada y estructurada.

Primeramente, se ha realizado un trabajo de documentación previo de dos meses. En este periodo de tiempo, se han estudiado las distintas plataformas IoT que existen, analizado como incorporar la computación autónoma, además de aprender las bases de la misma. Para el aprendizaje se realiza el proyecto de la asignatura del máster MITSS, Diseño de Sistemas Ubicuos y Adaptativos, donde se dota a un simulador de vehículos autónomos de las capacidades auto-adaptativas utilizando las herramientas proporcionadas por el equipo de investigación TaTami en el centro PROS de la UPV.

En segundo lugar, con lo aprendido en el proyecto del simulador, se plantea el diseño del bucle para abordar la solución propuesta en este trabajo, incorporando la computación autónoma a los proyectos software desarrollados en AWS IoT.

Una vez terminado el diseño, se inicia el desarrollo del prototipo que demostrará la aplicación del bucle de control MAPE-K a una solución funcional que hace uso de la plataforma de Amazon.

Finalmente, se dedica un trabajo de dos meses a la documentación del desarrollo y el análisis de los resultados obtenidos para demostrar si es posible convertir soluciones reales en soluciones auto-adaptativas.

1.4. Estructura

Este trabajo se divide en siete capítulos, además de una bibliografía y un glosario de términos. En este apartado vamos a explicar de manera breve como se estructuran los contenidos de la memoria.

- **Capítulo 1 – Introducción:** En este capítulo se habla acerca del propósito del trabajo. Se comenta como se ha decidido a realizar este proyecto, la motivación, los objetivos a cumplir y la metodología utilizada para alcanzarlos.

- **Capítulo 2 – Estado del arte:** Nada más introducir el trabajo, se realiza un estudio de las tecnologías que existen en el mundo acerca de los dominios con los que se va a trabajar, concluyendo con una crítica al mismo.

- **Capítulo 3 – Análisis del problema:** Se realiza un análisis con los datos recopilados y se profundiza en la solución, mostrando como podemos aplicar nuestra solución al mercado actual.

- **Capítulo 4 – Diseño de la solución:** Una vez analizado, se propone el diseño de nuestra solución, donde se explica todo el proceso llevado a cabo para la creación de ambos subsistemas.

- **Capítulo 5 – Desarrollo de la solución propuesta:** Se documenta el desarrollo de la solución, donde se sigue el diseño detallado anteriormente. Este se expone explicando cómo se han desarrollado los dos subsistemas por separado.

- **Capítulo 6 – Validación del prototipo:** Antes de finalizar, se realiza una traza de ejecución para mostrar el funcionamiento del prototipo desarrollado en nuestro caso de estudio, explicando dos adaptaciones diferentes.

- **Capítulo 7 – Conclusiones:** Finalmente, se resuelve si se han cumplido los objetivos del trabajo, comentando el trabajo realizado y las dificultades que ha tenido el autor.

2. Estado del arte

El crecimiento del mercado IoT ha traído la creación de un gran número de plataformas que ofrecen servicios y herramientas para el desarrollo de soluciones en este dominio.

De entre estas, podemos destacar plataformas propietarias como Amazon Web Services (AWS) que ofrece recolección y envío de datos a la nube para su posterior análisis además de administración y mantenimiento para dispositivos. Google Cloud es otra plataforma que proporciona mantenimiento predictivo para el equipo, soluciones para ciudades inteligentes o el seguimiento en tiempo real. Google también ofrece en su plataforma capacidades de Machine Learning o Inteligencia Artificial para IoT además de soporte para un amplio rango de sistemas operativos embebidos.

Por otro lado, existen otro tipo de plataformas como la que nos ofrece Arduino. En este caso, es una plataforma open-source que distribuye tanto software como hardware. Sensores, placas, procesadores entre otros, son los componentes que podemos obtener en su tienda. Por la parte software, también ofrecen su propio IDE permitiéndonos desarrollar para todos los dispositivos Arduino que poseamos, además de una amplia documentación y una gran comunidad detrás.

Gracias a estas plataformas y a la versatilidad de la IoT, podemos aplicar esta tecnología a una gran cantidad de escenarios como, por ejemplo:

- Los hogares conectados, donde se encuentran aplicaciones y dispositivos domóticos conectados para ofrecer una seguridad y monitorización en redes domésticas.

- La ganadería y agricultura, utilizando dispositivos IoT para automatizar tareas como el riego, la detección de plagas o la fase de cosecha, entre otras. También utilizados para el cuidado de los animales, tanto alimentación como geolocalización.

- Para ámbito comercial, esta tecnología impulsa un mejor monitoreo de la mercancía y el transporte, contribuyendo a la conservación de la carga, localización, optimización de rutas y la disposición de reportes de tráfico en tiempo real.

- En el ámbito industrial, se han ido diseñando aplicaciones industriales para poder realizar mantenimientos predictivos y garantizar una calidad. También ayudan a monitorizar las operaciones realizadas a distancia.

- La salud es un punto muy importante, aplicable tanto a mascotas como a seres humanos. La IoT se puede aplicar a aplicaciones de medición y detección de variaciones en los signos vitales del organismo. Esto facilita el trabajo de los sanitarios y de la administración de medicamentos de manera automatizada.

- Finalmente, los wearables, dispositivos pequeños, eficientes y dotados con el hardware para poder tomar diferentes mediciones, sobre todo en los seres humanos, facilitando la vida de las personas en la mayor medida.

2.1. Plataformas IoT

Como hemos visto en este apartado, existen una gran cantidad de plataformas dedicadas a proporcionar herramientas para el desarrollo de soluciones IoT, como las que hablaremos a continuación.

2.1.1. Sentilo[1]

Es una plataforma de sensores y actuadores de código abierto diseñada para adaptarse a la arquitectura de Smart City, de modo que cualquier ciudad consiga cierto nivel de interoperabilidad. Utilizado y respaldado por una comunidad activa y diversa de ciudades y empresas que creen que el uso de estándares abiertos y software libre es la primera decisión inteligente para abordar una Smart City. Diseñado como una plataforma cruzada para evitar soluciones verticales, compartiendo información entre los sistemas heterogéneos y fácilmente integrable con aplicaciones heredadas. Sentilo nos proporciona una serie de características:

- La plataforma está diseñada centrándose en el **rendimiento** para procesar miles de mensajes en un tiempo de respuesta muy rápido.

- Siendo **modular** y **extensible**, la plataforma define una arquitectura de componentes para ampliar la funcionalidad de la plataforma sin modificar el sistema central.

[1] https://www.sentilo.io/wordpress/

- Permite una fácil distribución de la carga en distintos equipos gracias a la **escalabilidad horizontal**.

- Diseñada como un **producto cruzado** no enfocado en ningún requisito comercial concreto, huyendo de soluciones verticales.

- Es de **código abierto** lo que permite compartir las contribuciones y mejoras, y poder crear un ecosistema de empresas a su alrededor.

- Proporciona una **interfaz REST simple**, fácil de usar e intuitiva para la comunicación con el servidor.

- El **módulo de estadísticas básicas** registra y muestra los indicadores básicos de rendimiento de la plataforma.

- El **catálogo y consola de administración** son destinados a administrar los dispositivos instalados en la calle y sus usuarios tanto proveedores como las aplicaciones.

2.1.2. Ditto by Eclipse[2]

Ditto es una tecnología utilizada en IoT que implementa el patrón de diseño “digital twins”, en el que se crea un dispositivo virtual en la nube el cual es una representación de su contraparte en el mundo real. Esta tecnología permite simular millones de dispositivos digitales conectados a sus respectivos físicos. Simplificando el desarrollo de soluciones IoT ya que, no es necesario saber dónde están realmente conectados los dispositivos físicos, con Ditto cualquier “digital twin” puede usarse con otro servicio.

La principal diferencia con los otros servicios IoT es que este no se trata de una plataforma IoT, debido a que no proporciona software que se ejecute en puertas de enlace ni tampoco define ni implementa protocolos para la comunicación con los dispositivos. Se centra en escenarios “back-end” al proporcionar una web API para simplificar el trabajo con los dispositivos ya conectados y los objetos o “cosas” de las

[2] https://www.eclipse.org/ditto/

aplicaciones del cliente u otro software “back-end”. Ditto se enfoca en resolver las responsabilidades que tiene un "back-end" típico en tales escenarios.

Su objetivo es liberar las soluciones IoT de la necesidad de implementar y operar un “back-end” personalizado. En cambio, al usar Ditto, pueden centrarse en los requisitos comerciales, en la conexión de dispositivos con la nube, el “back-end” y en la implementación de las aplicaciones.

2.1.3. SmartWorks by Altair[3]

Del mismo modo que Sentilo, SmartWorks es una plataforma de arquitectura abierta que ofrece una flexibilidad optima dada su amplia compatibilidad con hardware, tecnologías de comunicación, aplicaciones de terceros o su ecosistema de licencias y socios globales.

SmartWorks está integrado con HyperWorks, una plataforma de simulación, y PBS Works, una tecnología de computación en la nube desarrollada por Altair que permite a los usuarios desarrollar proyectos IoT altamente escalables y hacer uso del concepto de “digital twins”, al igual que Ditto.

Respecto a la gestión de los dispositivos, SmartWorks hace uso de dos tipos de flujos, el flujo de datos y el flujo de estado. Con ellos se puede controlar si la recepción de datos entre los dispositivos es correcta y conocer el estado de los mismos, permitiendo así, autenticarlos en la red IoT. Con la frecuencia de flujo tanto de datos como de estado se permite verificar la recepción esperada de ambos y, si no se recibe la transmisión dentro del rango esperado, se cambia el estado interno del dispositivo.

2.1.4. AWS IoT[4]

Amazon ha creado una plataforma llamada AWS IoT con el objetivo de facilitar el desarrollo de soluciones IoT. Ofrece un servicio para ayudar a los desarrolladores a asegurarse de que sus dispositivos funcionan correctamente y de forma segura. Este servicio se llama AWS IoT Device Management, que facilita el registro, organización,

[3] https://www.altairsmartworks.com/

[4] https://aws.amazon.com/iot/

monitorización y administración remota de dispositivos IoT a gran escala. Permite al desarrollador registrar los dispositivos conectados de forma individual o masiva y administrar los permisos para que los dispositivos permanezcan seguros, haciendo uso del concepto “digital twin” o “shadow device”.

También ofrece herramientas para organizar los dispositivos, monitorizarlos y solucionar problemas de funcionalidad que puedan surgir, consultar el estado de cualquier dispositivo IoT de su flota y enviar actualizaciones de firmware de forma transparente (OTA). Este es independiente del sistema operativo o el dispositivo, lo que permite administrar todo tipo de dispositivos IoT.

2.2. Computación Autónoma

Cada vez más, junto con la aparición de nuevas tecnologías, como la computación en la nube o los microservicios, se crean, a su vez, sistemas más complejos, flexibles y fiables. Pero, no todo es tan maravilloso, en 2001, IBM publicó un manifiesto donde se referenció una inminente crisis de complejidad del software, causada por este mismo incremento de la complejidad tanto en la instalación, configuración, modificación y mantenimiento de los sistemas software (2) (3).

Una de las soluciones a este problema es utilizar sistemas informáticos autónomos que pueden administrarse por sí mismos dados los objetivos de alto nivel de los administradores.

La esencia de la computación autónoma en estos sistemas es la autogestión, cuyo objetivo es liberar a los administradores de sistemas de las tareas de operación y mantenimiento del sistema y, proporcionar a los usuarios un producto que funcione con el maximo rendimiento las 24 horas del día. Para conseguir este comportamiento autónomo, los sistemas mantendrán y adaptarán su funcionamiento siguiendo una serie de principios o aspectos, agrupados bajo el concepto de paradigma para la autonomía denominado “self-*” (4) (5).

- **Auto-configuración:** Permite al sistema realizar configuraciones sobre el mismo, basándose en los datos obtenidos del entorno.

La instalación, configuración e integración en los grandes sistemas es un desafío, que requiere mucho tiempo y es propenso a errores, incluso para los expertos. Este principio

tiene como objetivo que los sistemas autónomos se configuren automáticamente siguiendo las políticas de alto nivel, que especifican que se desea, no como se consigue.

- **Auto-optimización:** Busca oportunidades para mejorar el rendimiento y eficiencia del sistema.

Los sistemas pueden tener cientos de parámetros ajustables que deben configurarse de cierta manera para que el funcionamiento sea correcto. Estos a su vez pueden ser integrados con otros subsistemas igual de complejos. Para ello, surge el aspecto de la auto-optimización, donde los sistemas autónomos buscan continuamente formas de mejorar la ejecución, identificando y aprovechando oportunidades para hacerse mas eficientes y reduciendo el coste. Los propios sistemas se monitorizan y aprenden de los cambios que realizan para conseguir mantener las funcionalidades deseadas.

- **Auto-curación:** El sistema detecta, diagnostica y repara problemas en el software o en el hardware.

Las empresas tienen departamentos dedicados a identificar y determinar la causa de fallos en los sistemas informáticos complejos. Los equipos encargados de ello pueden tardar incluso semanas en detectar los fallos y, también, corregirlos. Dicho esto, los sistemas autónomos detectan y reparan los problemas que hayan localizado, ya sean fallos software como hardware.

- **Auto-protección:** El sistema se defiende de ataques maliciosos o fallos en cascada, también anticipándose y previniendo fallos a gran escala.

A pesar de la existencia de firewalls y herramientas para la detección de intrusos, son los humanos los encargados de decidir cómo proteger los sistemas ante ataques maliciosos o fallos en cascada. Los sistemas autónomos se autoprotegen en dos sentidos, el primero, defendiéndose de fallos en cascada que puedan provocar estos ataques maliciosos y, en segundo lugar, anticipándose a los problemas basándose en informes de sensores tomando medidas para evitarlos.

La computación autónoma presenta una serie de retos, al igual que otras tecnologías:

En primer lugar, la computación autónoma tiene como objetivo crear una solución software con capacidades autónomas, es decir, trasladar la carga de la gestión de los sistemas, de las personas a las propias máquinas. Esto implica que el ser humano deja de tener control absoluto sobre el sistema y es él mismo el encargado de gestionarse.

Este problema se le conoce como *control shift* (6), donde los sistemas tienen cada vez más responsabilidad mientras que los humanos tienden a ser supervisores.

Eso supone un reto ya que necesitamos que el sistema autónomo sea capaz de mostrar al usuario que está realizando y en qué estado se encuentra en tiempo real, para que, de este modo, el usuario tenga una confianza en el sistema.

Por último, hemos de tener en cuenta que la computación autónoma tiene un grado de complejidad elevado y debemos de definir qué características deben de abordarse, los límites del sistema, etc. (7) Tanto la definición del entorno como el tipo de solución suponen un reto a la hora de crear estos sistemas ya que, dependiendo de estos factores será necesario un tipo de arquitectura u otra.

Antes de desarrollar una solución software autónoma, debemos de definir el espacio de diseño y límites del mismo. (8) Han surgido diversas metodologías para poder identificar y representar distintos espacios de diseño. En (9) se definen un conjunto de dimensiones de modelaje, y en (10) se define un espacio de diseño como un conjunto de decisiones que debe tomar el programador. En (11) se argumentan distintas propiedades de diseño a la hora de crear este tipo de sistemas autónomos.

De los trabajos mencionados se extrae la idea de identificar un conjunto de propiedades de diseño relevantes y representativas para el dominio que se pretende abordar. Con el objetivo de facilitar la identificación del carácter del sistema y la interacción entre las características de autoadaptación y los bucles de control. Se pretende seguir con el planteamiento anterior acerca de delegar sobre el sistema las distintas tareas de gestión y al usuario con tareas triviales. Sin embargo, se debe de facilitar la modificación del sistema, en tareas de un impacto elevado sobre el mismo, a un técnico especializado, de modo que se pueda realizar un cambio en este.

2.2.1. Sistemas auto-adaptativos

Como hemos dicho anteriormente, los sistemas auto-adaptativos se presentan como una de las soluciones a los problemas de complejidad de los productos software actuales. Estos sistemas se caracterizan por la capacidad de ajustar su comportamiento en tiempo de ejecución para lograr los objetivos del sistema, sin necesidad de intervención de usuarios (2).

Circunstancias impredecibles como cambios en el entorno, fallos del sistema o nuevos requisitos son algunas de las razones por las cuales el sistema debe de desencadenar las debidas acciones para la autoadaptación. Para poder lidiar con estas incertidumbres el sistema se monitoriza continuamente, recopila datos y los analiza para decidir si requiere una adaptación.

Esto lleva a un desafío importante, un sistema autoadaptable debe aplicar cambios en tiempo de ejecución, además de cumplir con los requisitos de alto nivel que se establecen. Por ello, el diseño de estos sistemas es complicado, ya que el conocimiento previo al desarrollo del mismo, no es el adecuado para anticipar todas las incertidumbres que pueden ocurrir en el momento del despliegue y ejecución.

2.2.2. Bucle de control MAPE-K

Una forma de abordar los desafíos de los sistemas auto-adaptativos es adoptar una perspectiva de sistemas de control. La forma de lograrlo es agregar sensores para monitorizar su estado en tiempo de ejecución, ejecutores para cambiarlo y un mecanismo de razonamiento que decide cuando es apropiado realizar cambios en el sistema y cuál es la mejor manera de conseguirlo. Este método es conocido como el bucle de control MAPE-K, monitorizar-analizar-planificar-ejecutar usando una base de conocimiento compartida (13).

Esta perspectiva ayudó a que investigadores y desarrolladores consideraran las arquitecturas de sistemas auto-adaptativos como áreas de estudio de primera clase. A pesar de que la arquitectura MAPE-K abarca la mayoría de los esfuerzos de los investigadores, siguen surgiendo numerosos desafíos para la obtención correcta de detalles (12).

La autonomía que se consigue a través del bucle MAPE-K se divide en dos componentes: un administrador autonómico, el propio bucle encargado de la monitorización y gestión, y un recurso autónomo gestionado, el sistema software en sí. El bucle de control se divide en una serie de módulos, cada uno encargado de realizar las tareas necesarias tanto para la recolección y análisis de datos como para la ejecución de los planes de adaptación (4) (14).

- El módulo **monitor** es el responsable del proceso de adquisición o recolección de datos y la observación de cualquier posible cambio en el entorno en el que

se encuentre. Estos datos externos al sistema son los que posteriormente, pasaran a analizarse para identificar si se realiza una adaptación o no. También se monitorizan datos del propio sistema, ya que es posible que se deban adoptar medidas debido a fallos en el mismo.

- El módulo de **análisis** ayuda a derivar información beneficiosa después de haber recopilado los datos, comparando el funcionamiento del sistema con los objetivos establecidos. De modo que, a partir de los datos reportados por el módulo monitor, determina cuando es necesaria la adaptación, en que parte debe de realizarse, priorizar sobre otros problemas cuando se detectan varios, etc. Una vez analizados los datos, este módulo lanza una solicitud de cambio al planificador con las modificaciones necesarias.

- El módulo **planificador** guía al sistema con la ayuda de políticas definidas por el usuario, reglas y regulaciones, determinando las acciones correctivas a realizar. Así, el planificador crea una serie de medidas o procedimientos para realizar la modificación deseada por el módulo de análisis. Esta puede tratarse desde una pequeña alteración hasta un cambio complejo en todo el sistema.

- El módulo de **ejecución**, como su propio nombre indica, gestiona la ejecución del flujo de trabajo en un sistema autónomo. Es decir, controla la implementación del plan desarrollado y proporciona la retroalimentación adecuado al módulo monitor.

- Finalmente, el módulo de **conocimiento** mantiene información relevante sobre propiedades directamente relacionadas con el soporte a procesos de adaptación. Esta se representa como un tipo de datos con una sintaxis y semántica, como unas políticas, peticiones de cambio o planes de cambio.

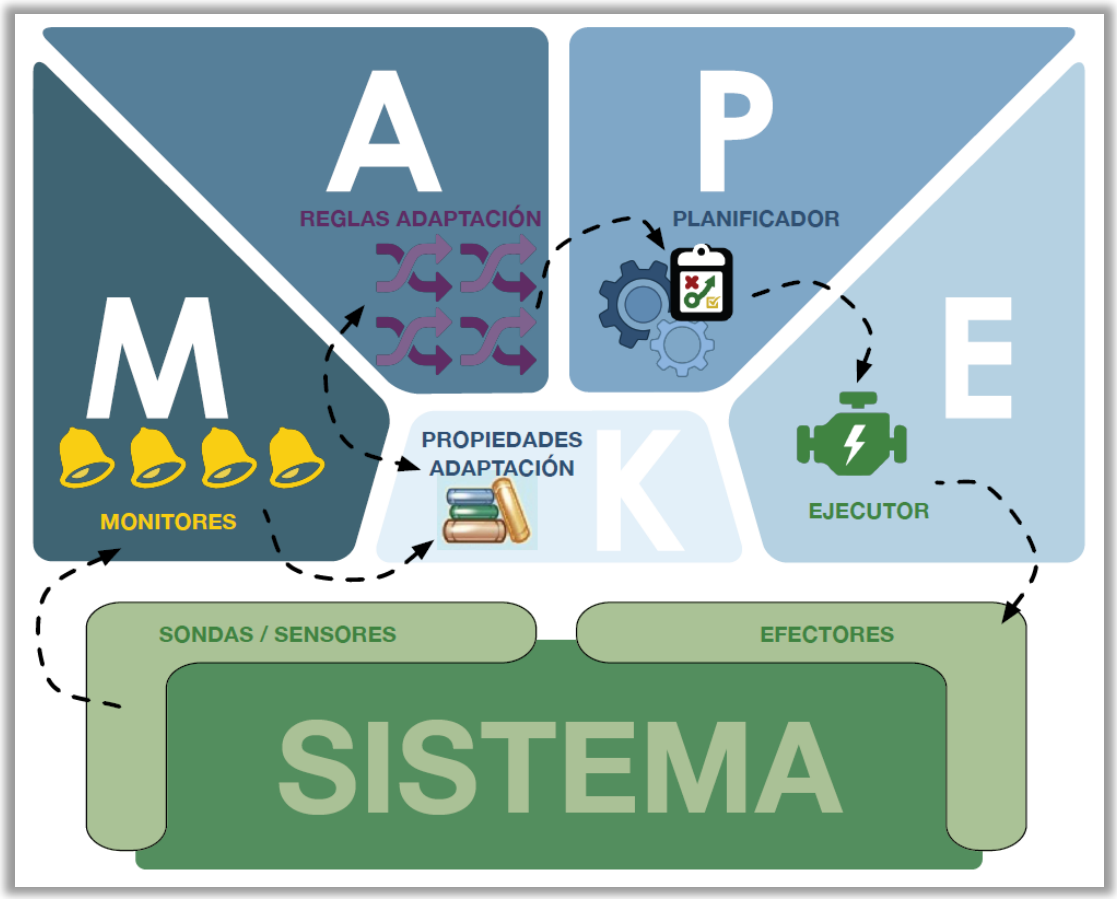

Ilustración 3. Bucle MAPE-K

Además de todos los módulos descritos, como vemos en la imagen 3, existen los sensores y efectores, los cuales forman parte del recurso autónomo gestionado. En primer lugar, los sensores o sondas son los encargados de recopilar cierta información, la que se describa para cada uno, que se reporta al monitor cuando se detecte el cambio solicitado. Por otro lado, se encuentran los efectores, que son los encargados de realizar los cambios en el sistema, trabajando junto con el módulo de ejecución.

2.3. Crítica al estado del arte

En la actualidad, la gestión y configuración de los dispositivos IoT, es fija. Esto significa que, en todas las plataformas descritas, cuando es necesario aplicar cambios o realizar un control sobre los dispositivos, solo puede hacerse manualmente, requiriendo que un desarrollador u operador tenga que acceder a los servicios y gestionarlos. Como hemos dicho anteriormente, para solucionar este problema

podemos aplicar los principios de la computación autónoma, entre otras, y otorgar al sistema de las capacidades auto-adaptativas que esta nos ofrece.

También se han detectado otros problemas a parte de la gestión y configuración fija o estática de los dispositivos, que se pueden solucionar utilizando la aproximación de computación autónoma.

- De manera similar al problema anterior, el mantenimiento y resolución de inconvenientes con los dispositivos que son desplegados en localizaciones remotas es otro punto que necesita solución.

- Cada vez se crean sistemas y soluciones más sofisticadas y complejas que pueden exceder la capacidad humana de administrar los dispositivos o el sistema en sí. La computación autónoma ayudaría a simplificar el mantenimiento y gestión de los mismos.

- La mayoría de implementaciones IoT se basan en una arquitectura cliente-servidor centralizada, la cual no es adecuada debido a la naturaleza dinámica y distribuida característica de este dominio. Este modelo es propenso a cuellos de botella, tiempo de inactividad y ataques coordinados que afectarían al funcionamiento de la red.

Un caso donde es posible aplicar la computación autónoma como solución son las conocidas como Smart City, los cuales son proyectos que pretenden convertir la ciudad un poco más inteligente, recopilando datos de las zonas urbanas para así gestionar mejor los recursos y servicios. En nuestro caso, utilizaremos la plataforma AWS IoT para la gestión y comunicación entre dispositivos, centrándonos en el alumbrado de las calles de la ciudad, donde son necesarios unos sensores que capten la luminosidad del ambiente o detecten la proximidad de las personas para encender, apagar o ajustar a cierta potencia las luces.

2.4. Tecnología y herramientas utilizadas

2.4.1. Entorno de desarrollo

Ilustración 4. Icono Eclipse.

Eclipse[5] es un entorno de desarrollo open-source creado originalmente por IBM y actualmente apoyado por desarrolladores y vendedores de software. Este forma parte de *Eclipse Foundation*, una organización sin ánimo de lucro que proporciona a su comunidad un entorno maduro, escalable y amigable para las empresas que facilita la colaboración e innovación en el desarrollo de software de código abierto. La *Eclipse Foundation* incluye el *IDE Eclipse* y *Jakarta EE* entre otros, que ofrecen herramientas y frameworks para una amplia gama de tecnologías como IoT, automoción, geoespacial, etc.

2.4.2. Lenguaje de programación

Ilustración 5. Icono Java.

Java[6] es un lenguaje de programación orientado a objetos basado en clases de propósito general. Esta diseñado de forma que los desarrolladores escriben el código y lo ejecutan en cualquier parte, es decir, este código puede ejecutarse en cualquier plataforma que soporte *Java* sin necesidad de recompilarlo. Las aplicaciones escritas en este lenguaje permiten ser ejecutadas sin tener en cuenta la arquitectura del ordenador, ya que dichas aplicaciones son compiladas a *bytecode* y ejecutadas en una *Java Virtual Machine* (*JVM*).

[5] https://www.eclipse.org/org/

[6] https://www.java.com/en/

2.4.3. Herramientas

Ilustración 6. Icono OSGi Alliance.

OSGi[7] es un consorcio mundial de innovadores de tecnología sin ánimo de lucro destinado a la creación de especificaciones abiertas que permiten el ensamblaje modular de software construido sobre *Java*. Sigue el principio de modularidad, el cual reduce la complejidad del software, es por ello que *OSGi* está considerado como el mejor modelo para modularizar soluciones en *Java*.

Esta tecnología facilita la creación de componentes de los módulos software y asegura una gestión remota e interoperabilidad entre las aplicaciones y servicios divididos entre los distintos dispositivos, incrementando la productividad en el desarrollo y consiguiendo una modificación y evolución más sencilla.

Ilustración 7. Icono MQTT.

MQTT[8] es un protocolo de mensajería estándar de OASIS[9] para IoT. Su diseño es ligero y eficiente permitiendo tanto a microcontroladores como a dispositivos muy pequeños, hacer uso de dicho protocolo, requiriendo la mínima cantidad de recursos. Permite el envío de mensajes entre dispositivos a través de la nube, haciendo posible el broadcast de estos mensajes en grupos de dispositivos, de manera bidireccional. Es usado en una amplia variedad de industrias, como la automoción, telecomunicaciones, fabricación, etc.

[7] https://www.osgi.org/about-us/

[8] https://mqtt.org

[9] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

3. Análisis del problema

De entre los problemas existentes en las soluciones software IoT actuales que hemos identificado anteriormente, este trabajo se centra exclusivamente en resolver el problema acerca de la configuración y gestión de los sistemas, la cual es fija y se necesita de un operario o desarrollador para realizar dichas tareas.

Este problema se puede solucionar mediante la aplicación de la computación autónoma, de modo que se aporta dinamismo al sistema y se dota de capacidades para la autogestión. En consecuencia, elimina la necesidad de un usuario encargado de modificar el sistema y adaptarlo manualmente.

El sistema que desarrollamos en este trabajo parte de una solución software tradicional, en la cual disponemos de unos controladores que se encargan de controlan a las farolas. A dichos dispositivos se incrustan una serie de componentes IoT encargados de recopilar información del entorno y del estado de las farolas, además de enviar los diferentes valores de luminosidad a los controladores, los cuales deben de adoptar las farolas. Finalmente, se diseña el bucle encargado de controlar toda la parte auto-adaptativa del sistema.

3.1. Sistema de control de alumbrado de una Smartcity

Tal y como se indica en los objetivos, en este trabajo se propone aplicar la computación autónoma a una solución AWS IoT para dotarla de capacidades auto-adaptativas, en concreto, autocuración y autoconfiguración. Esto lo conseguimos diseñando un bucle de control, el bucle MAPE-K, el cual se combina con un pequeño prototipo de aplicación IoT desarrollada en dicha plataforma.

Así, dividimos el trabajo en dos componentes principales, el administrador autonómico, en nuestro caso el bucle de control y el recurso autónomo gestionado, el prototipo desarrollado en la plataforma de AWS IoT.

El prototipo a desarrollar es una porción de un proyecto de una SmartCity, concretamente utilizaremos el alumbrado de la ciudad inteligente. El alumbrado de una ciudad está compuesto por las farolas y los controladores, encargados de ajustar el

comportamiento de las farolas, ya sea apagando o estableciendo un valor lumínico. Como vemos en la ilustración 8, el controlador y la farola están ubicados por tramos de calle y para nuestro caso de estudio, controlaremos el alumbrado de dos calles utilizando cuatro farolas, dos por cada calle, las cuales están conectadas a un controlador. A estos componentes se les suman los sensores, ubicados en los controladores y en las calles, y son los encargados de reportar a nuestro sistema si se detecta algún cambio o fallo en estos. Por último, disponemos de un componente que llamamos política, la cual se encarga de gestionar como debe ser el comportamiento del controlador y, en consecuencia, de las farolas.

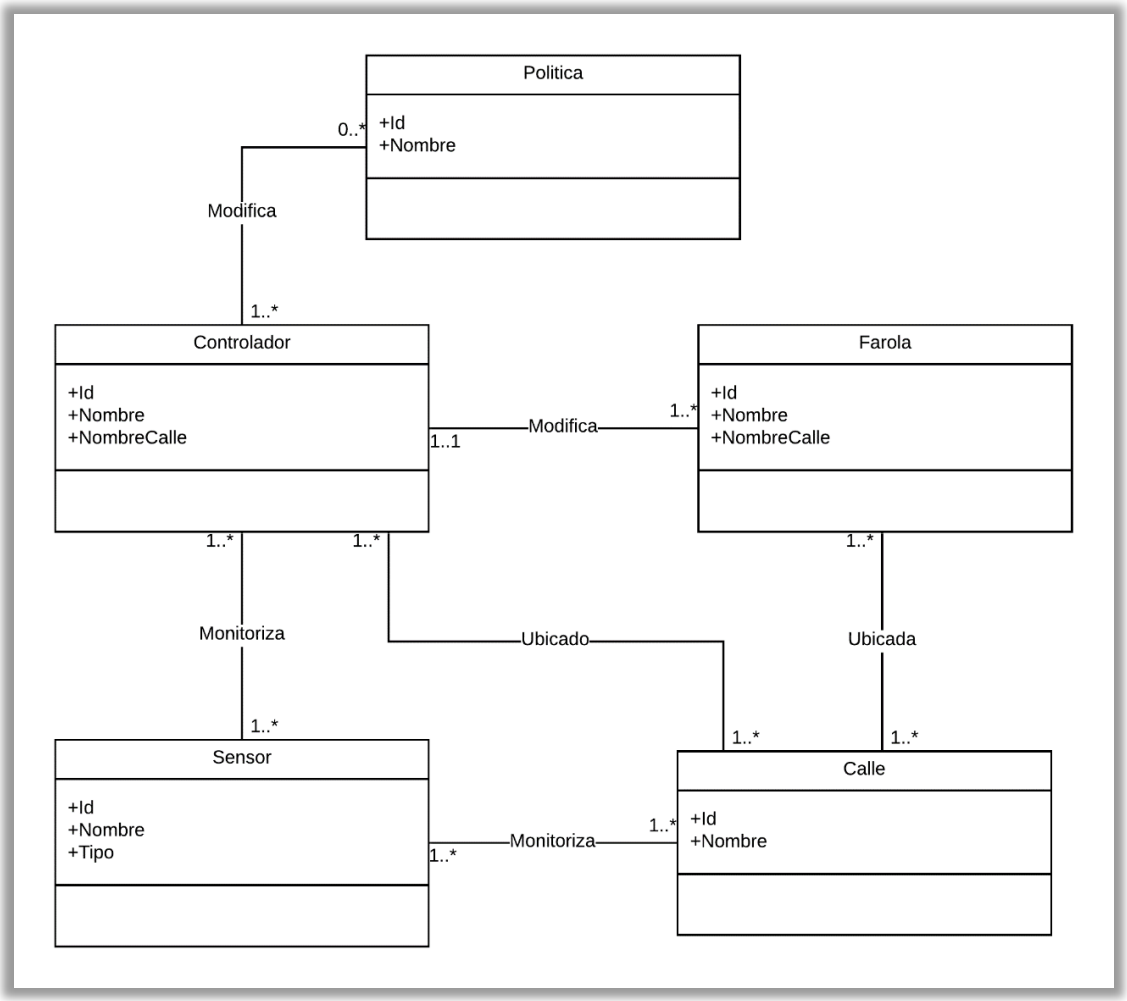

Ilustración 8. Diagrama de clases de análisis para SmartCity Streetlights.

En la actualidad, los ajustes de intensidad sobre las farolas se realizan manualmente o de forma remota, por un operario del ayuntamiento. Lo que se busca es convertir este proceso y hacerlo más “smart”, esto lo logramos utilizando dispositivos IoT que incrustamos en las farolas y controladores, permitiendo la comunicación entre ellos y su control remoto, además de la incorporación de la computación autónoma.

Por lo que respecta al bucle MAPE-K, describimos una serie de componentes para su funcionamiento. Primeramente, disponemos de una serie de sensores o sondas que se encargaran de reportar cambios en el entorno, como la luminosidad de la calle, y en el propio sistema, por ejemplo, si fallara alguna lectura. Estos envían los valores obtenidos a los monitores, los cuales se encargan de procesar los datos y deciden si estos son relevantes para iniciar un cambio en las propiedades de adaptación. A continuación, y una vez analizados los datos, se crea el plan de adaptación utilizando una serie de reglas o políticas que establecemos para, finalmente ejecutar el plan mediante el módulo de ejecución.

3.2. Requisitos de adaptación

En este trabajo nos enfocamos en la unión de dos dominios diferentes para así, poder aplicar lo mejor de ambos en una única solución. En nuestro caso, dotamos al sistema de control de alumbrado de una SmartCity de dos capacidades auto-adaptativas: la auto-configuración y auto-curación. Para poder aplicar dichas capacidades es necesario identificar que comportamientos se deben tener en cuenta y con qué datos se van a trabajar para aplicarlos.

Los requisitos de adaptación, los cuales vamos a llamar “políticas” a lo lardo del trabajo, son una descripción de los comportamientos que van a afectar al estado de los controladores y, a su vez, de las farolas. De modo que, para describirlos, definimos unos datos de entrada, unas condiciones para que entren en ejecución y las acciones que van a aplicar sobre el sistema.

3.2.1. Política: DayTime

Esta política se encarga de ajustar el nivel de luminosidad de las farolas en función de la hora del día. De modo que, si se acerca la noche se encienden las luces con una mayor intensidad. La política se activa por defecto con la ejecución de escenario y, en caso de fallo, se activa la política de cantidad de luz, que actúa como un plan de contingencia. También, al tratarse de la política por defecto, cuando alguna de las otras políticas está activa y surge un fallo, entra en ejecución “DayTime”, así se aplica la capacidad de autocuración y, a su vez, la autoconfiguración.

Tabla 1. Política DayTime.

Requisito de adaptación	Política para la hora del día.
Datos de entrada	Franja horaria establecida entre un rango de horas.
Condiciones	Se aplica por defecto al iniciar la solución. Si se encuentra activa otra política, esta se ejecuta cuando se detecta un fallo en la política actual.
Acciones a aplicar	Cuando se realiza un cambio entre un rango de horas a otro, se envía al controlador la intensidad de luz a la que deben de ajustarse las farolas.

3.2.2. Política: RainyDay

Esta política, a diferencia de “DayTime”, solo se activa con la detección de un mínimo de intensidad de lluvia. Su función es realizar cambios en la intensidad de luz de las farolas según la cantidad de lluvia detectada. Así, cuando deja de detectarse lluvia o falla, se activa la política “DayTime” y logramos aplicar la autoconfiguración y autocuración.

Tabla 2. Política RainyDay

Requisito de adaptación	Política para un día lluvioso.
Datos de entrada	Intensidad de lluvia.
Condiciones	Si otra política esta activa, cuando se detecta lluvia, se analiza en que grado llueve y se cambia a esta política.
Acciones a aplicar	Según la cantidad de lluvia detectada, se ajusta la intensidad de luz de las farolas. A mayor intensidad de lluvia, mayor luminosidad en las farolas.

3.2.3. Política: LightAmount

Esta política se crea con la finalidad de aplicar la autocuración, actuando como plan de contingencia para la política “DayTime”. De modo que, cuando dicha política falla, se activa “LightAmount”, la cual, según la cantidad de luz detectada en la calle, se adapta la potencia de las farolas para conseguir una visibilidad correcta. Si esta detecta algún fallo, por ejemplo, en la lectura del sensor, se vuelve a cambiar a la política “DayTime”.

Tabla 3. Política LightAmount.

Requisito de adaptación	Política para la cantidad de luz.
Datos de entrada	Intensidad de luz.
Condiciones	Si la intensidad de luz pasa de unos rangos establecidos, se ajusta la intensidad de las farolas de acuerdo a estos.
Acciones a aplicar	Según la intensidad de luz detectada, se aumenta la potencia de las farolas si esta baja, y viceversa.

3.2.4. Política: MovementDetection

En caso de detectar algún movimiento por la calle, ya sea un vehículo o una persona, se configura el sistema para activar la política “MovementDetection”. Esta tiene la finalidad de iluminar el tramo de la calle para conseguir la visibilidad correcta, estableciendo las farolas a un valor fijo de potencia. Si surge un fallo durante su funcionamiento, esta se desactiva y pasa a configurarse la política DayTime, de modo que el sistema es capaz de autocurarse y autoconfigurarse.

Tabla 4. Política MovementDetection.

Requisito de adaptación	Política para la detección de movimiento.
Datos de entrada	Detección de movimiento.
Condiciones	Si se detecta movimiento, se activa la política, en caso contrario se desactiva.
Acciones a aplicar	Cuando se activa, se establece un nivel de luminosidad establecido a las farolas.

4. Diseño de la solución

Tras identificar los componentes y requisitos de adaptación que debe de soportar el sistema, vamos a exponer el diseño de la solución a nivel arquitectónico. El diseño se va a dividir en dos secciones, una para el sistema gestionado, donde hablaremos de la solución que engloba la parte IoT de AWS, y el bucle de control MAPE-K, el encargado de dotar al sistema de las capacidades auto-adaptativas mencionadas en capítulos anteriores.

4.1. Arquitectura del sistema gestionado

Para el diseño del sistema gestionado, utilizaremos, por una parte, los componentes físicos como los controladores y las farolas que hemos descrito y, por otro lado, los dispositivos IoT combinados con los anteriores. Así, dotamos de comunicación entre dispositivos físicos haciendo uso de la red, mediante el envío y recepción de mensajes.

Para ilustrar los componentes que forman parte de la solución a desarrollar, hemos creado un modelo de componentes que los recoge. En él, como podemos ver en la figura 5, vemos que las farolas, llamadas “streetlight”, las cuales iluminan el tramo de la calle en la que se sitúan y los controladores, llamados “controller”, encargados de modificar el estado de las farolas. A parte, vemos los cuatro tipos de sensores que vamos a utilizar en combinación con las políticas que describimos en el apartado anterior.

A continuación, se detalla cada componente de la solución IoT, describiendo las dependencias entre ellos y detallando todas las características que, posteriormente, se desarrollaran en el prototipo de SmartCity.

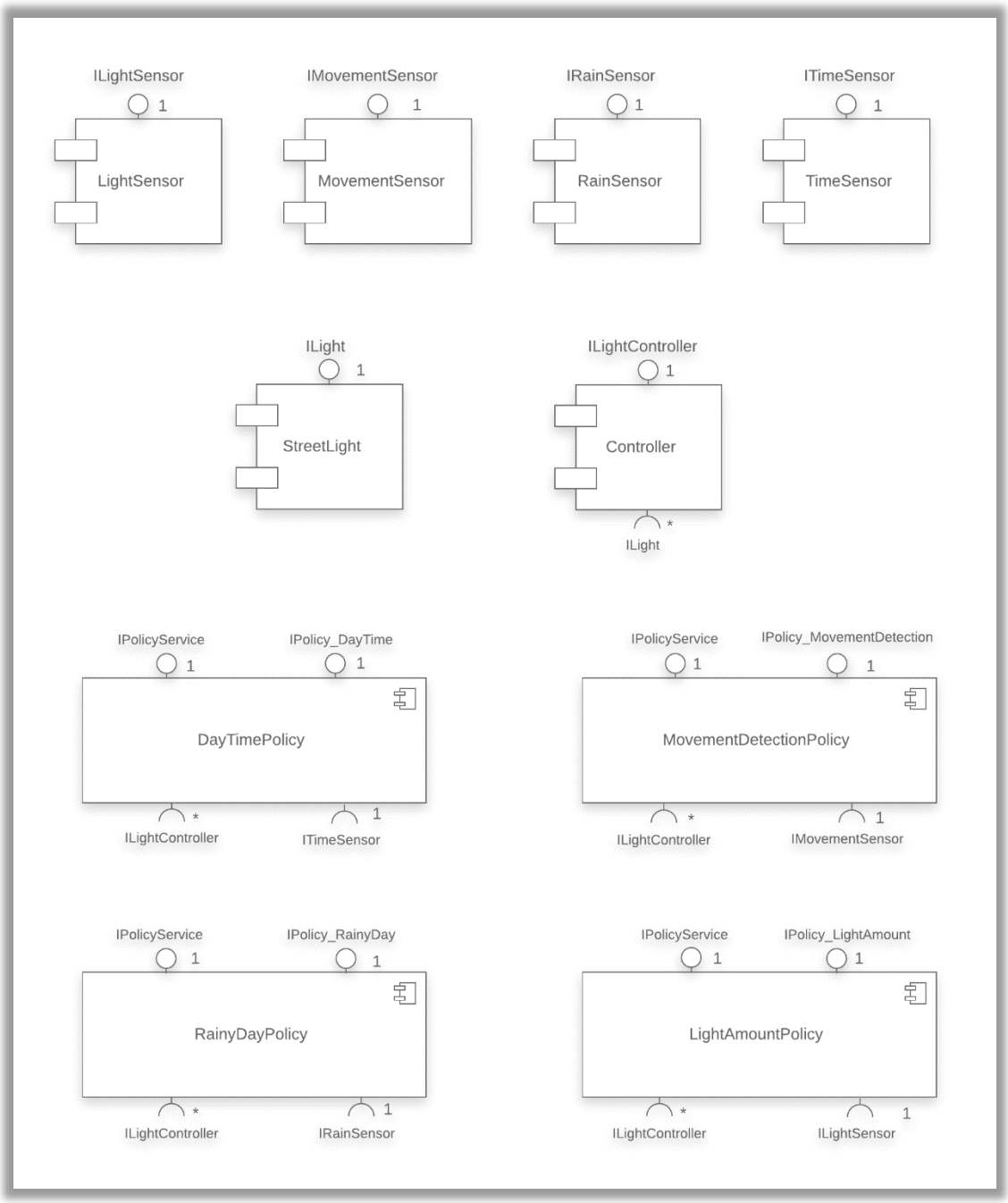

Ilustración 9. Diagrama de componentes de SmartCity Streetlights.

4.1.1. Sensores

Los sensores o sondas nos permiten monitorizar y recopilar diferentes datos acerca del entorno. Estos son reportados a los monitores, que describimos en el siguiente apartado, para analizarlos y continuar con el proceso de adaptación dentro del bucle de control.

4.1.1.1. LightSensor

Permiten leer la intensidad de luz que reciben, utilizando lux (iluminancia) como unidad para medir la incidencia de la luz sobre una superficie. Estos se encuentran situados en las farolas y nos ofrecen la posibilidad de enviar a los monitores la cantidad de luz que se está recibiendo actualmente en la calle y actuar conforme a ello.

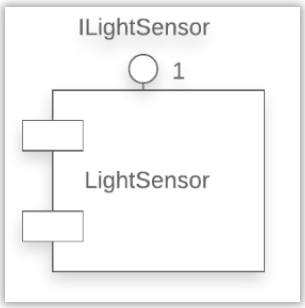

Ilustración 10. Componente LightSensor.

El componente ofrece una única interfaz `ILightSensor`, que permite obtener la cantidad de luz recibida por dicho sensor, en lux.

4.1.1.2. MovementSensor

Detectan si existe movimiento o no en el tramo o debajo de la misma farola donde se sitúa. De este modo, nos permite enviar una señal a los monitores indicando que se ha detectado movimiento cercano a la farola para poder encender, o apagar en caso contrario.

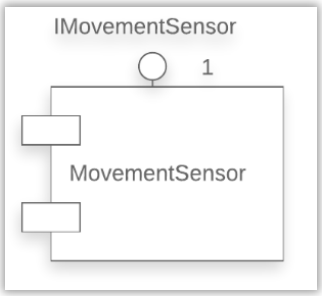

Ilustración 11. Componente MovementSensor.

El componente ofrece una única interfaz `IMovementSensor`, que comprueba si se ha detectado algún movimiento cercano.

4.1.1.3. RainSensor

En caso de lluvia, estos sensores envían una señal a los monitores para poder tomar medidas. Así, se aumenta la intensidad de luz en el tramo si llueve o se decrementa en caso contrario, aportando el nivel de visibilidad adecuado a la calle.

El componente ofrece una única interfaz `IRainSensor`, que comprueba si está lloviendo y con qué intensidad.

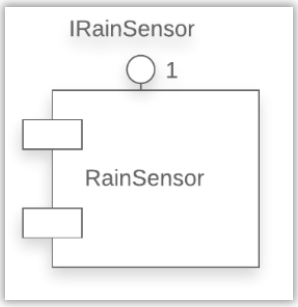

Ilustración 12. Componente RainSensor.

4.1.1.4. TimeSensor

Indica la franja horaria en la que se encuentra en el día, permitiendo que las farolas se enciendan o se apaguen dependiendo de dicha hora.

El componente ofrece una única interfaz `ITimeSensor`, obtiene la hora actual y envía la franja en la que se encuentra a los monitores.

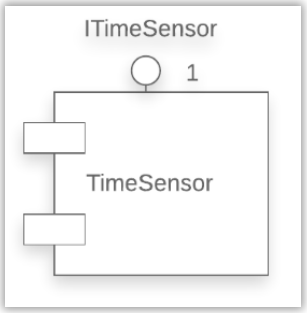

Ilustración 13. Componente TimeSensor.

4.1.2. Dispositivos

Respecto a los dispositivos, hacemos uso de la plataforma AWS IoT, concretamente utilizamos “Device Management”, dentro del servicio de AWS IoT Core, para la gestión de nuestros dispositivos. Esta plataforma crea los dispositivos de manera digital, esto se le conoce comúnmente como “Digital Twin”, Amazon lo denomina “Shadow Device” (15), aunque este posee la misma funcionalidad. Estos dispositivos digitales son un reflejo virtual del objeto físico, el cual nos permite acceder a su estado desde cualquier aplicación o servicio que esté conectado a la red AWS IoT.

4.1.2.1. Controller

En él se encuentra un sistema de regulación de intensidad de flujo de corriente, que permite regular la intensidad de luz de las farolas.

El componente ofrece la interfaz `ILightController` y necesita una interfaz (0..*) `ILight` para realizar cambios sobre las farolas.

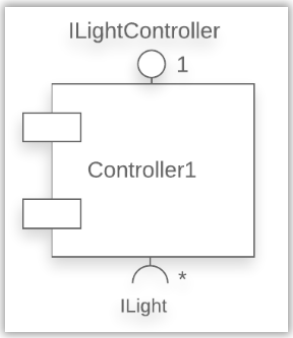

Ilustración 14. Componente Controller.

4.1.2.2. StreetLight

Dispositivo LED que admite cinco niveles de intensidad distintos. Se conecta al controlador que permite realizar configuraciones entre las distintas intensidades.

El componente ofrece una única interfaz `ILight`, que permite encender o apagar las farolas y regular la intensidad de luz.

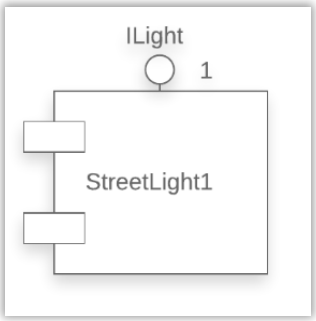

Ilustración 15. Componente StreetLight.

4.1.3. Políticas

4.1.3.1. DayTime

Esta política toma como valor de entrada una franja horaria y según si se encuentra dentro de una u otra de las establecidas, ajusta la potencia de las farolas a los valores de cada franja mediante el controlador de farolas.

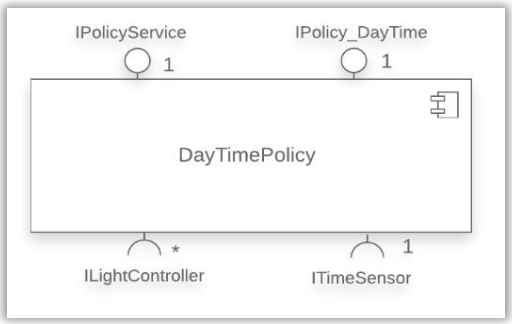

Ilustración 16. Componente DayTimePolicy.

El componente ofrece dos interfaces `IPolicyService`, `IPolicy_DayTime` y necesita la interfaz `ILightController` y `ITimeSensor` de modo que permite que la política controle las farolas y obtenga la hora actual que reporta el sensor.

En la tabla se muestra las distintas franjas en las que se dividen las 24 horas del día y la potencia a la que deben de encenderse las farolas:

Tabla 5. Relación Hora – Potencia de las farolas.

Franja horaria	**Desde**	**Hasta**	**Potencia de las farolas**
Morning	5:00 AM	11:59 AM	50%
Afternoon	12:00 PM	4:59 PM	0%
Evening	5:00 PM	4:59 AM	80%

4.1.3.2. LightAmount

Esta política tiene como finalidad la recepción de un valor en lux por parte del sensor de luz. Una vez obtenido el valor, se comprueba en que rango se encuentra para finalmente indicar la potencia a la que deben de ajustarse las farolas.

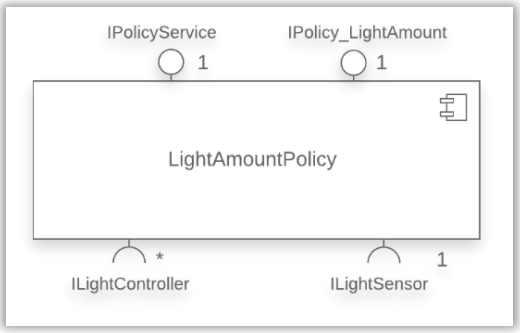

Ilustración 17. Componente LightAmountPolicy.

El componente ofrece dos interfaces `IPolicyService`, `IPolicy_LightAmount` y necesita la interfaz `ILightController` y `ILightSensor` de modo que permite que la política controle las farolas y obtenga la cantidad de lux, proporcionada por el sensor de luz.

A continuación, se muestra una tabla donde se incluye la potencia a la que se deben de encender las farolas según la cantidad de lux recibida por el sensor de luz:

Tabla 6. Relación Rango de lux – Potencia de las farolas.

Rango de lux	**Potencia de las farolas**
[120 000 – 20 000]	0%
[19 999 – 2 000]	20%
[1 999 – 1 000]	50%
[999 – 400]	80%
[399 – 0]	100%

4.1.3.3. MovementDetection

Utilizando el sensor de movimiento, el cual reporta si se ha detectado movimiento o no debajo de una farola, la política ajusta el valor de la farola al 80% en caso de ser positivo y a 0% en caso de ser negativo.

El componente ofrece dos interfaces `IPolicyService`, `IPolicy_MovementDetection` y necesita la interfaz `ILightController` y `IMovementSensor` de modo que permite que la política controle las farolas y obtenga si se ha detectado movimiento alrededor de la farola o no.

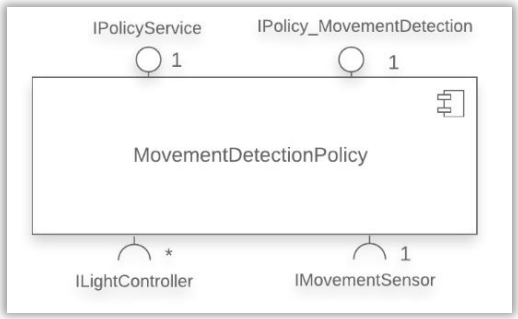

Ilustración 18. Componente MovementDetectionPolicy.

Tabla 7. Relación Detección de movimiento – Potencia.

Detección de movimiento	**Potencia de las farolas**
true	80%
false	0%

4.1.3.4. RainyDay

En esta política, el sensor de lluvia reporta si llueve de manera ligera, normal o fuerte y la política ajusta la potencia en función de estos tres valores. Cuanta más cantidad de agua se detecta, más potencia se necesita en las farolas del tramo si deseamos un nivel de visibilidad seguro.

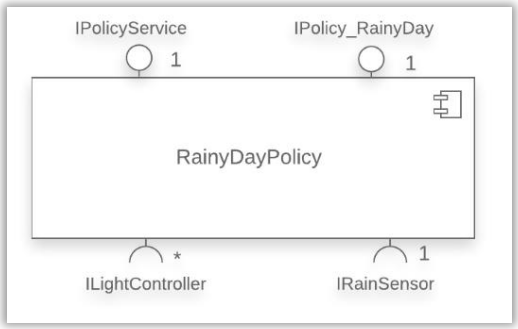

Ilustración 19. Componente RainyDayPolicy.

El componente ofrece dos interfaces `IPolicyService`, `IPolicy_RainyDay` y necesita la interfaz `ILightController` y `IRainSensor` de modo que permite que la política controle las farolas y obtenga la intensidad de lluvia actual que reporta el sensor.

Tabla 8. Relación intensidad de lluvia – Potencia de las farolas.

Intensidad de lluvia	**Potencia de las farolas**
Low	20%
Medium	50%
High	80%

4.2. Bucle de control MAPE-K

Para la parte auto-adaptativa del sistema, la cual vamos a combinar con la solución IoT tradicional, utilizamos la aproximación propuesta por IBM para el bucle de control, el bucle MAPE-K. Este consta de los cinco módulos que identificamos en el estado del arte, para los cuales debemos de describir una serie de monitores, propiedades y reglas de adaptación. Así, conseguimos integrar y, como se pretende en este trabajo, combinar el subsistema gestionado con el bucle de control autónomo de manera correcta.

En esta solución utilizamos el *framework* FADA MAPE-K lite, desarrollada por el grupo TaTami, en el centro PROS-UPV. Nos proporciona un diseño para los componentes del bucle que vamos a integrar, además de su implementación para los módulos, como el planificador y el ejecutor.

A continuación, nos centramos en identificar y describir los elementos que van a formar parte del bucle para, posteriormente, realizar su implementación y poder alcanzar los objetivos establecidos.

4.2.1. Monitores y propiedades de adaptación

Anteriormente, hemos comentado los distintos módulos que forman el bucle MAPE-K, en este apartado nos vamos a centrar en los monitores, del módulo monitor, y las propiedades de adaptación, del módulo de conocimiento.

Los monitores son los encargados de recibir los datos que son enviados por los sensores, los cuales son filtrados y transformados en información útil para identificar si el sistema se está gestionando correctamente o si se detecta algún fallo. Toda esta información recibida por parte de los sensores es la que, una vez es “entendida” por los monitores, pasa a convertirse a propiedades de adaptación dentro del bucle de control y forma parte del módulo de conocimiento (*Knowledge*) de MAPE-K.

Hay que definir una serie de monitores y propiedades de adaptación que recibirán los datos de los sensores:

- **Monitor de intensidad de luz captada:** Recibe un valor lumínico, utilizando la unidad de medida lux. Su finalidad es identificar a que potencia deben establecerse las farolas, además de detectar si existe un fallo en la lectura que

puede estar ocasionado por el mismo sensor. El monitor trabaja con la **propiedad de adaptación "estado del sensor de luz"**. De este modo, si la cantidad de luz que hay en la calle aumenta, por ejemplo, al mediodía, se actualiza la propiedad con el nuevo valor reportado y activaría la regla de adaptación correspondiente. En caso de fallo, lanzaría la regla de adaptación encargada de configurar el sistema y solventarlo la pérdida del sensor.

- **Monitor de movimiento detectado:** Recibe si se ha detectado movimiento o no en el tramo de calle donde se encuentran las farolas ubicadas. La finalidad del monitor es, en caso de detección de movimiento, encender las farolas, si estaban apagadas, y ajustarlas a una potencia suficiente para tener una correcta visibilidad. Este trabaja con la **propiedad de adaptación "estado del sensor de movimiento"**. Así, además de permitirnos ajustar la potencia y encender o apagar las farolas, podemos comprobar si el sensor sigue operativo para poder lanzar la regla de adaptación correspondiente.

- **Monitor de intensidad de lluvia captada:** Del mismo modo que el monitor de intensidad de luz, este recibe, por parte del sensor, la cantidad de agua que llueve en la calle. Su objetivo es identificar si es necesario el aumento o disminución de potencia de las farolas según la intensidad de lluvia. Este monitor trabaja con la **propiedad de adaptación "estado del sensor de lluvia"**. De modo que, en caso de fallo, se activa una regla de adaptación que inicia un plan de emergencia y cambio de política.

- **Monitor de franja horaria del día:** Recibe una franja horaria leída desde el sensor de tiempo del sistema, de forma similar a un reloj. La finalidad del monitor es comprobar si la franja horaria del bucle es la correcta, además de identificar si hay fallo en las lecturas del sensor. El monitor trabaja con la **propiedad de adaptación "estado del sensor de tiempo"**. Cuando el monitor detecta un cambio en la hora, por ejemplo, de la mañana a la tarde, lanza la regla de adaptación que ajusta la potencia de las farolas al valor establecido.

- **Monitor de configuración inicial:** Este monitor es utilizado para poder dotar de una configuración inicial al prototipo. Para este se trabaja con la **propiedad de adaptación "autoconfiguración"**, de modo que desde la sonda de configuración inicial se reporta una orden al monitor para lanzar la regla que realiza dicha configuración.

4.2.2. Reglas de adaptación

Las encargadas de realizar las adaptaciones en el sistema, tras el análisis de los datos de entrada realizado por los monitores, son las reglas de adaptación. En ellas, se define el estado en el que se encontrará el sistema cuando es ejecutada. Las reglas trabajan sobre una propiedad de adaptación, como los monitores, realizando las adaptaciones según los cambios que se realicen en dicha propiedad.

A continuación, definimos las reglas de adaptación necesarias para completar el comportamiento auto-adaptativo del sistema:

- **Regla de adaptación para el cambio de franja horaria a lo largo del día:** Esta trabaja con la propiedad de adaptación “estado del sensor de tiempo”, en caso de cambio en la franja horaria, por ejemplo, de mañana a la tarde, la regla envía el nuevo estado al que deben de estar las farolas, en este caso, aumentando su potencia. Si se detectara un fallo en el sensor, se inicia el plan de contingencia y se aplica un cambio para adoptar la política “LightAmount”.

- **Regla de adaptación para cambios en el estado del sensor de luz:** Escucha los cambios en la propiedad de adaptación “estado del sensor de luz”. En este caso, se pueden dar dos situaciones, una en la que el estado del sensor es correcto reportando valores dentro del rango de lux establecido y, en el cual, la regla calcula la nueva potencia a la que deben de establecerse las farolas. Por otro lado, si el sensor reporta valores anómalos, esta inicia el plan de emergencia y cambiaría a la política “DayTime” que utiliza el sensor de tiempo.

- **Regla de adaptación para la detección de movimiento:** Esta regla trabaja con la propiedad de adaptación “estado del sensor de movimiento”, en caso de reporte de detección de movimiento por parte del monitor, la regla calcula la nueva potencia a la que deben de adaptarse las farolas. Si el monitor reporta un fallo en el sensor, esta regla inicia una adaptación y cambio a la política por defecto “DayTime”.

- **Regla de adaptación en caso de detección de lluvia:** Escucha cambios en la propiedad de adaptación “estado del sensor de lluvia”. Cuando empieza a detectar lluvia, según el reporte del monitor, calcula la potencia a la cual deben de establecerse las farolas. En caso de fallo del sensor, se inicia el plan de emergencia, el cual introduce un cambio en el sistema y se cambia a la política “DayTime”.

- **Regla de adaptación para la configuración inicial:** Esta regla se ejecuta nada más el monitor recibe la orden de autoconfiguración, al entrar el prototipo en ejecución. Es encargada de lanzar la política “DayTime” y sus componentes necesarios para funcionar, ya que es la política por defecto del sistema.

4.3. Arquitectura completa

Una vez hemos identificado los componentes que forman tanto el sistema gestionado como el bucle de control que lo gestiona, obtenemos el diagrama de la ilustración 20. En este vemos diferenciados ambos subsistemas, en la parte superior mostramos los módulos que forman el bucle de control MAPE-K donde, además, se sitúan los monitores y reglas de adaptación que hemos diseñado anteriormente. Respecto a los módulos planificador, ejecutor y sondas (mostrados de color gris) se detallan en el siguiente apartado de desarrollo de la solución propuesta.

Para el subsistema gestionado, el sistema IoT que vemos en la parte inferior de la ilustración 20, hemos detallado como interactúan los componentes entre si para el prototipo que desarrollamos. De modo que, las políticas necesitan su sensor correspondiente (LightAmountPolicy requiere LightSensor) y el controlador, en nuestro caso el “Controller1” que, a su vez, controla y requiere al menos una farola. Para el prototipo utilizamos un controlador que envía a las farolas (Streelight1 y Streetlight2) la intensidad de luz a la que se deben ajustar.

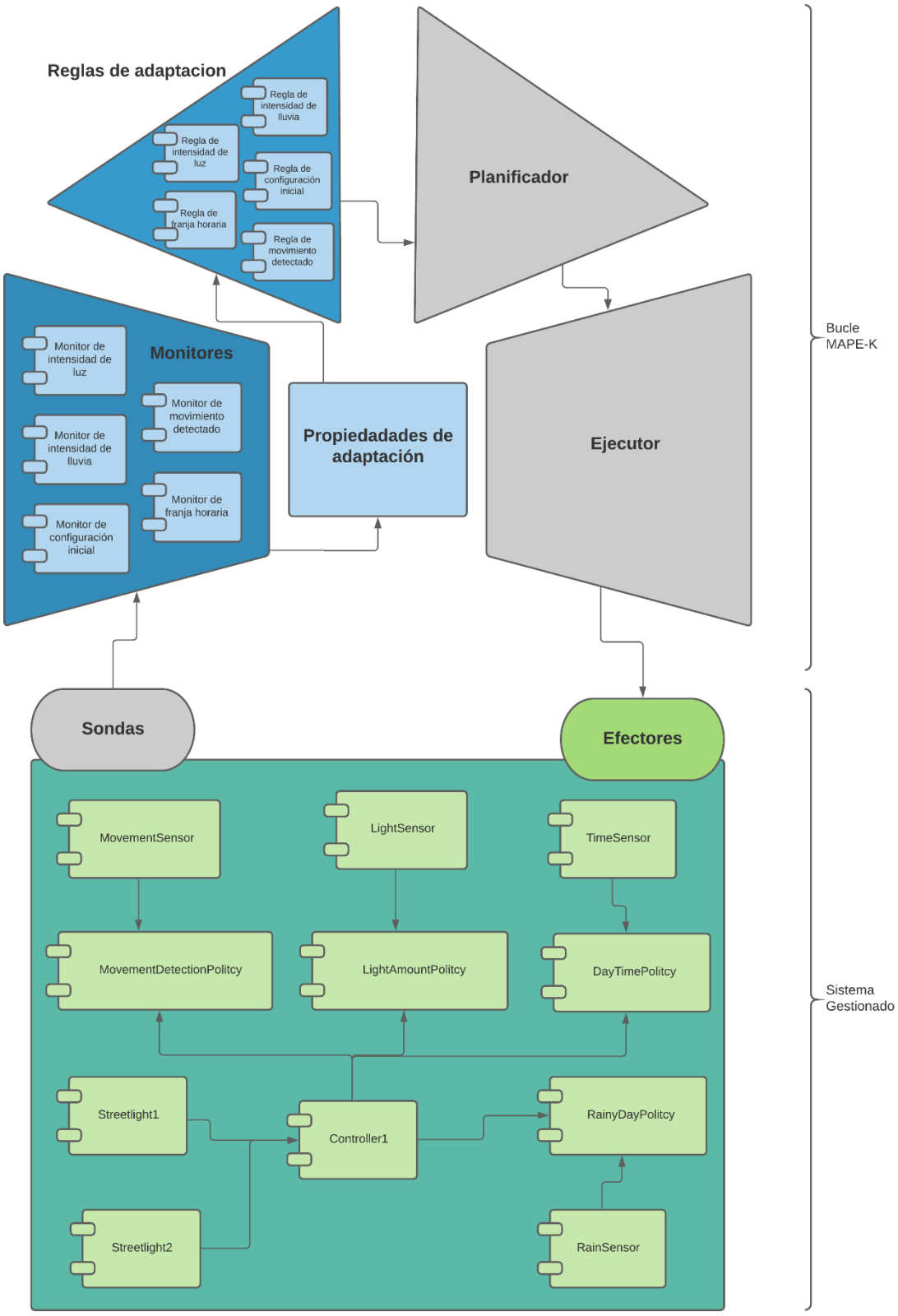

Ilustración 20. Diagrama de componentes del sistema SmartCity Streetlights.

5. Desarrollo de la solución propuesta

Por la naturaleza de la solución que proponemos, se necesita de un *framework* que nos permita gestionar, de manera flexible, los componentes del sistema una vez se encuentra en ejecución. OSGi nos ofrece este tipo de desarrollo de soluciones dinámicas, permitiendo crear aplicaciones modulares sobre el lenguaje Java. En este apartado vamos a detallar tanto el desarrollo del subsistema gestionado como del bucle de control MAPE-K, para el cual hemos usado la implementación FADA MAPE-K lite, desarrollada por el grupo TaTami, en el centro PROS-UPV.

Primero hablamos de la estructura del proyecto, donde describiremos los distintos paquetes en los que se ha distribuido la solución. A continuación, comentamos, por una parte, el desarrollo del subsistema gestionado y, por otro lado, la implementación del bucle de control MAPE-K.

Como hemos expuesto en el diseño, la solución está dividida en dos subsistemas, uno consta de sensores, farolas y controladores, la parte IoT del trabajo y, la del bucle de control, en la que se incluyen las políticas, sondas, propiedades y reglas de adaptación descritas.

5.1. Estructura del proyecto Eclipse

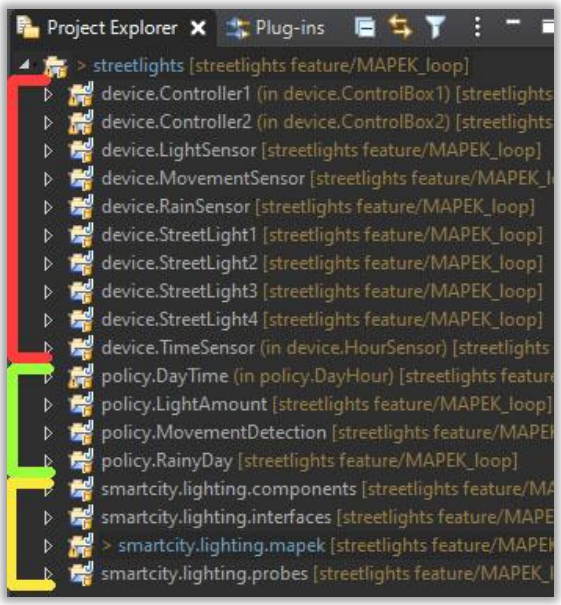

Ilustración 21. Estructura del proyecto Eclipse.

Como vemos en la ilustración 21, estructuramos el proyecto completo y los componentes del prototipo en tres grupos, diferenciados por colores.

El primero de ellos está formado por los componentes device.* que son los equivalentes a los sensores y dispositivos físicos mencionados anteriormente. En este caso, se trata de dispositivos Shadow Device, de modo que utilizamos una representación digital de los dispositivos reales y que se gestiona a través de la plataforma AWS

IoT Device Management. Así, nos permite comunicarnos con el dispositivo real mediante su copia virtual, ofreciendo la ventaja de que, en caso de pérdida de conexión con el dispositivo físico, aún es posible la recuperación de los últimos datos recibidos.

El segundo grupo está formado por las políticas, en ellas se describe el comportamiento que deben realizar las farolas, transmitido a través de los controladores. Estas no tienen representación física, ya que se encuentran dentro de la misma aplicación donde se encuentra el bucle de control.

Por último, el tercer grupo contiene todos los componentes necesarios para el funcionamiento del bucle, incluidas las comunicaciones con los dispositivos a través de la API de AWS.

5.2. Subsistema gestionado

Los componentes que englobamos en este apartado son todos los dispositivos IoT que utilizamos en el sistema gestionado. Estos son los que hemos detallado en el apartado de diseño de la solución, como vemos en la ilustración 22.

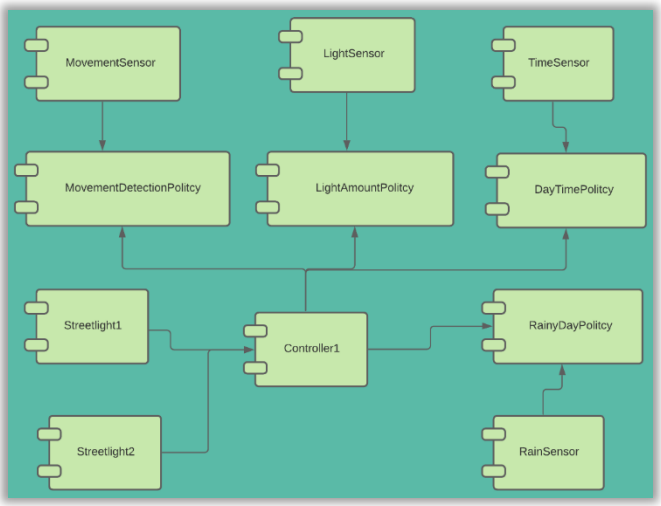

Ilustración 22. Diagrama de componentes del sistema gestionado.

Todos estos componentes se dividen en distintos paquetes dentro del proyecto de eclipse. A continuación, se detalla en que paquete está situado cada componente, ya sea las farolas, el controlador, los sensores y las políticas. Además, se muestra cómo se han realizado las comunicaciones entre los dispositivos y la plataforma de AWS IoT Core.

5.2.1. Componentes del dominio

En el paquete smartcity.lighting.components, se encuentran todos los componentes que utilizamos en el prototipo (controladores, farolas y sensores), denominados “thing”. Esta “thing” es el concepto que se utiliza en el dominio de la IoT, donde cada dispositivo se identifica como una “cosa”. También se denominan “thing” a las políticas, pero se han estructurado en otro paquete como comentamos en el siguiente apartado.

Cada dispositivo en nuestro proyecto hereda de la clase Thing, la cual permite gestionar acciones comunes de los dispositivos, como gestionar el ciclo de vida, iniciando o deteniendo el componente, y proporcionar la infraestructura OSGi necesaria para el funcionamiento correcto de los mismos. Estas “things” proporcionan otro nivel de abstracción que nos permite la reutilización de los componentes para distintas soluciones.

A los sensores, farolas y controlador se les asigna un ID que nos permite identificar dichos componentes en tiempo de ejecución. También se agregan las variables para poder identificar el estado actual del dispositivo, por ejemplo, “controller_status” nos permite saber si el controlador se encuentra “ON” u “OFF”.

5.2.2. Políticas del sistema

Las políticas son las encargadas de encapsular el comportamiento que deben tomar las farolas, a través de los controladores, dependiendo del estado del entorno, reportado por los sensores. Estas se encuentran en el paquete smartcity.lighting.components.policies. Del mismo modo que el resto de dispositivos, estas heredan de la clase “Thing”. De esta manera, podemos gestionar su ciclo de vida pudiendo acoplar y desacoplar las políticas en ejecución como un componente OSGi.

Cada política tiene asociada un ID, como vemos en la línea 15 de la ilustración 24, el cual nos permite buscar el componente durante la ejecución para realizar las distintas acciones: iniciar, detener, acoplar o desacoplar. En el caso de las políticas, tras la

adaptación y cambio de una política en ejecución a otra, la anterior pasa a un estado "desactivado", por tanto, al desacoplar, la política debe detenerse.

```
11 public abstract class Policy_DayTimeService extends PolicyService implements IPolicy_DayTime {
12
13     protected String timeSensor = null;
14
15     public Policy_DayTimeService(BundleContext context, String id) {
16         super(context, id);
17         this.addImplementedInterface(IPolicy_DayTime.class.getName());
18     }
```

Ilustración 23. Constructor de la clase DayTimeService.

5.2.3. Comunicaciones entre dispositivos

Todas las comunicaciones que se realizan entre los dispositivos en la nube de AWS y los componentes del proyecto están recogidas dentro del paquete smartcity.lighting.components.communications. Para poder realizar estas conexiones hemos utilizado la API aws-iot-device-sdk-java, esta nos permite realizar comunicaciones con los dispositivos mediante el protocolo MQTT.

5.2.3.1. MQTT y los topic

Esta comunicación consiste en el envío y recepción de mensajes, a traves de unos canales denominados "topic", utilizando los mecanismos de "publish" y "subscribe". De este modo, un dispositivo que quiere recibir mensajes debe suscribirse a un canal o "topic" por el cual podrá leer dichos mensajes, mientras que el dispositivo que necesite enviarlos deberá publicarlos en el correspondiente "topic". Es el cliente emisor, llamado "publisher", el que decide en que formato envía el mensaje. En nuestro caso hemos decidido utilizar JSON como formato de mensajes ya que nos permite estructurar los mensajes mediante pares clave-valor.

Un "topic", en MQTT, es una cadena de caracteres utilizada por los distintos clientes para filtrar los mensajes. Estos consisten en uno o más niveles, separados por una barra "/", por ejemplo, de la siguiente manera: myhome/groundfloor/livingroom (16). Hemos seguido las guías de AWS IoT para los Shadow Device, en las que nos muestran una lista de topics reservados para su uso (17). En la tabla 9 vemos como se han estructurado los topics:

Tabla 9. Topics reservados.

Client subscribe topics	**Client publish topics**
controllers/controllerID/get	controllers/controllerID/get/accepted
	controllers/controllerID/get/rejected
controllers/controllerID/update	controllers/controllerID/update/accepted
	controllers/controllerID/update/rejected
streetlights/streetlightID/get	streetlights/streetlightID/get/accepted
	streetlights/streetlightID/get/rejected
streetlights/streetlightID/update	streetlights/streetlightID/update/accepted
	streetlights/streetlightID/update/rejected

Somos los encargados de gestionar el envío y recepción de mensajes de todos los dispositivos del prototipo, es por ello, que en el paquete de comunicaciones tenemos tanto los topic de los clientes suscritos como los que publican.

En la tabla vemos que disponemos de dos tipos de topics por dispositivo, los *get* y los *update*. Los topic get se utilizan para obtener el estado del dispositivo, de modo que "preguntas" por su estado y el cliente responde por el topic *accepted*, si el mismo acepta la petición, o por el topic *rejected*, en caso de rechazar la recepción del mensaje. Por otro lado, los topic update se utilizan para enviar el estado al que deben de actualizarse estos dispositivos y, del mismo modo, pueden aceptar o rechazar la nueva petición de cambio de estado, por ejemplo, por no estar autorizado el emisor en la red.

Hemos decidido utilizar JSON como formato para los mensajes, como ya hemos comentado, de modo que aprovechamos esta estructura de clave-valor para permitir que los clientes filtren los mensajes publicados en los topic. Por ejemplo, si en el topic:

streetlights/streetlightID/update, se publica un mensaje dirigido a los controladores, las farolas seguirán escuchando en dicho topic, al mensaje pertinente.

En los controladores y farolas hemos utilizado la misma estructura en el get. Así, cuando hacemos una petición get al dispositivo, este nos responde por el topic accepted, su “id” y “status” en el que se encuentra. A la hora de realizar un cambio en el estado de los dispositivos diferenciamos dos mensajes. En el caso del controlador, este recibe un mensaje en el que se indica en qué estado deben de encontrarse tanto las farolas como el mismo controlador, como podemos observar en la tabla 10. Ese mismo mensaje posee un segundo mensaje el cual es publicado por el controlador en los topic update de las farolas, que contiene el estado al que deben de cambiar, detallado en la fila cuatro.

Los sensores utilizan una estructura distinta ya que, al tratarse de una simulación, emulamos el comportamiento de los mismos mediante el envío de los mensajes. Hemos decidido crear unos topics para este caso, donde cada sensor esta suscrito y reciben los mensajes con el estado del entorno simulado.

Tabla 10. Mensajes por topic.

Device	**Publish in topic**	**Message**
Controller	controllers/controllerID/get/accepted	{“id”: “controllerID”, “status”: “ON/OFF”}
	controllers/controllerID/update	{“controllers”: {“status”:“ON/OFF”}, “streetlights”: {“status”:“POWER_X”}}
Streetlight	streetlights/streetlightID/get/accepted	{“id”: “streetlightID”, “status”: “POWER_X”}
	streetlights/streetlightID/update	{“status”: “POWER_X”}
Sensor	sensors/time	{“timeStatus”: “AFTERNOON/MORNING”}
	sensors/movement	{“movementStatus”: “true/false”}

	sensors/rain	{“rainStatus”: “LOW/MED/HIGH”}
	sensors/light	{“lightStatus”: [0..20000]}

5.3. Bucle de control MAPE-K

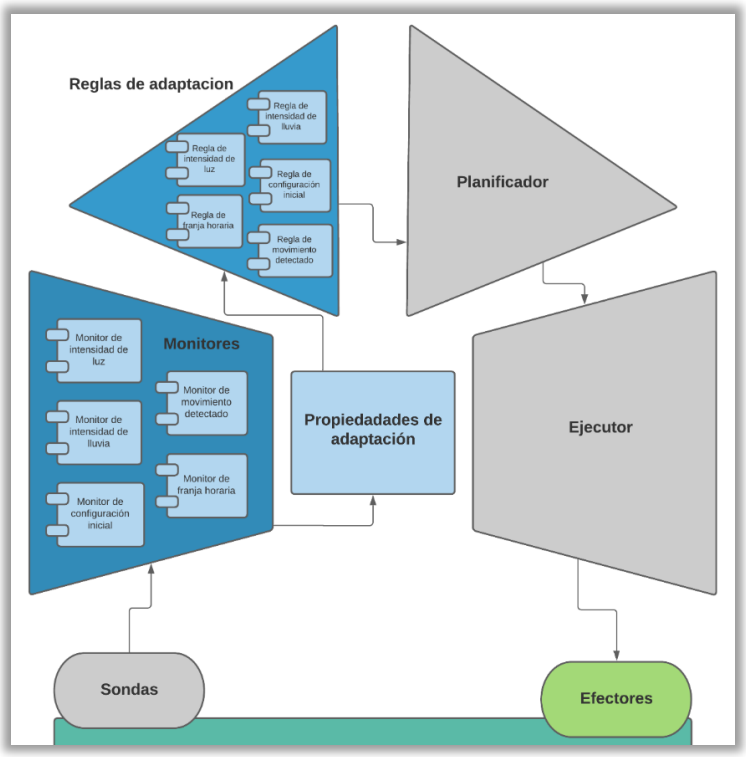

Ilustración 24. Bucle de control MAPE-K.

En este punto del desarrollo, ya disponemos del subsistema IoT sin capacidades de auto-adaptación. Para empezar a dotar al sistema de estas capacidades es necesario el desarrollo del bucle de control encargado de gestionar el sistema.

Como vemos en la ilustración 24, ya hemos detallado en el diseño de la solución, que reglas de adaptación, monitores y propiedades de adaptación vamos a utilizar en este proyecto. A continuación, vamos a describir como van a implementarse, además de añadir la implementación de las sondas ya que, el modulo planificador y ejecutor se encuentra implementado dentro de la aproximación FADA del bucle MAPE-K que utilizamos.

Para ello, primero debemos crear una serie de componentes, llamados componentes ARC (Adaptative-Ready Component). Seguidamente, creamos el resto de componentes que forman el bucle de control, como los monitores, las reglas de adaptacion y las sondas.

5.3.1. ARC Components

Ilustración 25. Paquete components.arc.

A continuación, debemos hablar de los componentes ARC, que se encuentran dentro del paquete smartcity.lighting.components.arc. Estos son componentes software preparados para ser instalados y/o desinstalados, además de poder ser configurados dinámicamente, es decir, en tiempo de ejecución. En nuestro caso, utilizamos los dispositivos "thing" que hemos mencionado anteriormente y creamos las clases ARC siguiendo una nomenclatura como, por ejemplo, ControllerARC. Así, con estas clases "convertimos" todos los componentes software convencionales a unos componentes preparados para realizar adaptaciones y, en consecuencia, poder utilizarlos en el bucle MAPE-K.

Como ejemplo, vamos a utilizar el sensor de lluvia, en el que creamos el sensor ARC a partir del sensor de lluvia creado anteriormente. A este se le asigna un ID y un estado inicial, como vemos en las líneas 19 y 20 de la ilustración 26. En este caso, las clases extienden de *AdaptativeReadyComponent*, la cual nos permite realizar las funciones de "deploy" y "undeploy" que vemos en las líneas 29 y 36 respectivamente.

```
11 public class RainSensorARC extends AdaptiveReadyComponent implements IAdaptiveReadyComponent {
12
13     public final static String SUPPLIED_RAIN_SENSOR = "Sensor";
14
15     protected RainSensor rainSensor = null;
16
17     public RainSensorARC(BundleContext context, String id) {
18         super(context, id);
19         this.rainSensor = new RainSensor(this.getBundleContext(), id);
20         this.rainSensor.setRainStatus(ERainStatus.UNKNOWN);
21         try {
22             this.rainSensor.start();
23             } catch (Exception e) {
24             e.printStackTrace();
25             }
26         }
27
28     @Override
29     public IAdaptiveReadyComponent deploy() {
30         super.deploy();
31         this.rainSensor.registerThing();
32         return this;
33     }
34
35     @Override
36     public IAdaptiveReadyComponent undeploy() {
37         super.undeploy();
38         this.rainSensor.unregisterThing();
39         return this;
40     }
```

Ilustración 26. Clase RainSensorARC.

Cabe destacar que, en este punto del desarrollo, la solución está preparada para ser auto-adaptativa. Cualquier futuro cambio en la especificación del sistema, puede añadirse sin problemas. Esto se obtiene gracias a la modularidad y flexibilidad ofrecida

por OSGi, de modo que, podemos crear cualquier nuevo componente como, por ejemplo, cuando necesitemos añadir más farolas a más tramos o queramos ampliar la solución abarcando semáforos o señales.

5.3.2. Monitores

Como detallamos en el diseño, los monitores trabajan con las propiedades de adaptación, modificándolas para posteriormente ser utilizadas por las reglas de adaptación. Para cada monitor, se le asigna un ID, como vemos en la línea 13 de la ilustración 27.

```
10 public class LightSensorMonitor extends Monitor {
11
12     public LightSensorMonitor(BundleContext context) {
13         super(context, "light-sensor-status-monitor");
14     }
15
16     @Override
17     public IMonitor report(Object measure) {
18
19         this.Logger.debug(String.format("Received measure: %s", measure.toString()));
20
21         try {
22             IKnowledge knowledge = this.getTheMonitoringModule().getTheKnowledgeModule().getTheKnowledge();
23             IKnowledgeProperty kp = knowledge.getKnowledgeProperty("light-sensor-status");
24             kp.setValue(measure);
25         } catch (Exception e) {
26             return this;
27         }
28
29         return this;
30     }
31 }
```

Ilustración 27. Clase LightSensorMonitor.

Además, ofrecen el método “report”, utilizado por las sondas, para notificar y modificar el valor de la propiedad de adaptación. Primero, se obtiene el módulo de conocimiento antes de obtener la propiedad de adaptación, en este caso “light-sensor-status”, como observamos en las líneas 22 y 23. Finalmente se establece el valor de la propiedad a partir del valor obtenido por la sonda (línea 24).

5.3.3. Reglas de adaptación

Como ya hemos comentado, las reglas de adaptación son las encargadas de realizar las configuraciones sobre el sistema. Estos cambios se realizan mediante la variable “theNextSystemConfiguration”, la cual contiene la configuración actual del sistema y cualquier actualización debe hacerse sobre la misma.

Existen varias diferencias respecto a la regla de adaptación de la configuración inicial y el resto de reglas, que nos permiten dotar de autocuración y autoconfiguración. Primero, detallamos la creación de las reglas, la cual es idéntica en cualquiera de ellas y, a continuación, se detalla el contenido que las diferencia.

Para crear las reglas de adaptación seguimos el mismo procedimiento que con los monitores, asignando una ID como vemos en la línea 26 de la ilustración 28. Además,

```
25    public SelfConfigureAdaptationRule(BundleContext context) {
26        super(context, "self-configure-rule");
27        this.setListenToKnowledgePropertyChanges("self-configure");
28    }
```

Ilustración 28. Constructor de la clase SelfConfigureAdaptationRule.

debemos de utilizar la función “setListenToKnowledgeChanges” para lograr que la regla se lance a partir de los cambios que se realicen en el módulo de conocimiento, sobre la propiedad de adaptación que indiquemos en la línea 27.

Las reglas sobrescriben el método “onExecute” heredado de la clase “AdaptationRule” de la librería de MAPE-K Lite. En este método se implementa todo el comportamiento que debe de adoptar el sistema, devolviendo la variable “theNextSystemConfiguration” con la nueva configuración deseada.

En el caso de la regla de adaptación para la configuración inicial, lo primero que debemos hacer es crear los componentes que entrarán en ejecución. Los sensores son los componentes que deben estar siempre activos a lo largo del ciclo de vida del sistema, ya que son los encargados de reportar a los monitores cualquier cambio detectado en el mismo. Además, se crea la política “DayTime”, es la política que queremos que se ejecute por defecto, junto con el controlador que envía a las farolas la potencia a la cual deben ajustarse.

```
49        //Creamos los componentes con su ID
50        IComponentSpecification thePolicy_dt = ComponentSpecification.build("day-time-policy", "1.0.0");
51        IComponentSpecification theTimeSensor = ComponentSpecification.build("timeSensor", "1.0.0");
52        IComponentSpecification theLightSensor = ComponentSpecification.build("lightSensor", "1.0.0");
53        IComponentSpecification theMovementSensor = ComponentSpecification.build("movementSensor", "1.0.0");
54        IComponentSpecification theRainSensor = ComponentSpecification.build("rainSensor", "1.0.0");
55        IComponentSpecification theController1 = ComponentSpecification.build("controller1", "1.0.0");
56
57        //Anadimos la política, sensores y controlador a la configuracion
58        theNextSystemConfiguration.componentToAdd(thePolicy_dt);
59        theNextSystemConfiguration.componentToAdd(theTimeSensor);
60        theNextSystemConfiguration.componentToAdd(theLightSensor);
61        theNextSystemConfiguration.componentToAdd(theMovementSensor);
62        theNextSystemConfiguration.componentToAdd(theRainSensor);
63        theNextSystemConfiguration.componentToAdd(theController1);
```

Ilustración 29. Creación y adición de componentes a la configuración.

Una vez creados los componentes, se añaden a la configuración mediante el método “componentToAdd” como vemos en las líneas 57 a 62 de la ilustración 29.

Antes de devolver la nueva configuración, se deben de añadir también los enlaces entre los componentes correspondientes. Al entrar en ejecución la política “DayTime” esta necesita un **sensor de tiempo** y un **controlador** para su funcionamiento, como definimos en el diseño del sistema. Esto se realiza, como vemos en las líneas 64 a 69 de la ilustración 30, mediante la función “build” de “BindingSpecification”. Seguidamente,

se utiliza “bindingToAdd” para añadir a la configuración los enlaces creados para, finalmente, devolver el nuevo estado al que debe configurarse el sistema.

```
63          //Creamos los enlaces entre el sensor y el controlador con la poltica
64          BindingSpecification timeSensorBinding = BindingSpecification.build(
65                  BindingEndpointSpecification.build(thePolicy_dt, "timeSensor"),
66                  BindingEndpointSpecification.build(theTimeSensor, "timeSensor"));
67          BindingSpecification controller1Binding = BindingSpecification.build(
68                  BindingEndpointSpecification.build(thePolicy_dt, "controller1"),
69                  BindingEndpointSpecification.build(theController1, "controller1"));
70
71          //Anyadimos los enlaces a la configuracion
72          theNextSystemConfiguration.bindingToAdd(timeSensorBinding);
73          theNextSystemConfiguration.bindingToAdd(controller1Binding);
74
75          return theNextSystemConfiguration;
```

Ilustración 30. Creación de enlaces y adición a la configuración.

Hemos comentado que disponemos de dos tipos de reglas, la regla de configuración inicia y el resto. Para explicar las restantes, tomamos como ejemplo la regla para cambios en el estado del sensor de luz. Tras asignar un ID en su creación, se implementa el método “onExecute” del mismo modo que la anterior. Lo primero que debemos de hacer es comprobar si la política misma política está activa, esto lo podemos observar en la ilustración 31.

```
39          IPolicy_LightAmount policy_la = OSGiUtils.getService(context, IPolicy_LightAmount.class);
40
41          if ( policy_la != null &&  policy_la.isActive() ) {
```

Ilustración 31. Obtención y comprobación de la política LightAmount.

Después de esta comprobación, creamos los componentes y enlaces que necesitamos para la adaptación, del mismo modo que en la ilustración 29 y 30.

Finalmente, y antes de devolver la nueva configuración, realizamos las comprobaciones que necesitemos para hacer una adaptación u otra. En este caso, debemos comprobar si el sensor de luz está detectando un valor anómalo y actuar en consecuencia. Como vemos en la ilustración 32, lo primero es quitar de la configuración la política que está

```
87          if ( lightStatus < 0 ) { //Si el valor reportado es anomalo
88              this.Logger.trace("Light Status < 0 -> Executing rule ...");
89              //Desactivamos la politica LightAmount y quitamos los componentes
90              theNextSystemConfiguration.componentToRemove(thePolicy_la);
91              theNextSystemConfiguration.bindingToRemove(lightSensorBindingToPolicyLA);
92              theNextSystemConfiguration.bindingToRemove(controller1BindingToPolicyLA);
93
94              //Activamos la politica DayTime y anyadimos los componentes
95              theNextSystemConfiguration.componentToAdd(thePolicy_dt);
96              theNextSystemConfiguration.bindingToAdd(timeSensorBindingToPolicyDT);
97              theNextSystemConfiguration.bindingToAdd(controller1BindingToPolicyDT);
98              }
99              return theNextSystemConfiguration;
```

Ilustración 32. Condición de adaptación: lux negativa.

en ejecución y sus enlaces (líneas 90 a 92) para, posteriormente, añadir la nueva junto con sus enlaces (líneas 95 a 97).

5.3.4. Sondas

Necesitamos dispositivos encargados de observar los cambios sobre el sistema, para ello utilizamos las sondas dentro del bucle de control. Estas notifican a los monitores acerca de un cambio en el estado de los sensores documentados anteriormente.

La estrategia a seguir para su creación es la misma que con los monitores. Se le proporciona un ID como, por ejemplo, “movement-sensor-probe” en su método constructor. Esta vez utilizamos tres funciones nuevas, dos de ellas son el método “start” y “stop”, que nos permite, como su nombre indica, iniciar y detener las sondas a lo largo de la ejecución.

En la ilustración 33 vemos que la sonda utiliza un “ServiceListener” para notificar sobre los cambios en el comportamiento del componente al que escucha. De modo que, utilizamos un filtro (línea 23) para indicar a la sonda a que componente debe de escuchar, en este caso el sensor de movimiento. A continuación, se hace uso de la función “addServiceListener” junto con el filtro creado (línea 25) para dejar la sonda en funcionamiento. Para la detención de la sonda, se elimina mediante la función “removeServiceListener”.

```
19      @Override
20      public ILifeCycleManager start() {
21          super.start();
22
23          String filter = String.format("(&(%s=%s)(id=movementSensor))", Constants.OBJECTCLASS, IMovementSensor.class.getName());
24          try {
25              this.getBundleContext().addServiceListener(this, filter);
26          } catch (InvalidSyntaxException e) {
27          }
28
29          return this.getLifeCycleManager();
30      }
31
32
33      @Override
34      public ILifeCycleManager stop() {
35          super.stop();
36
37          this.getBundleContext().removeServiceListener(this);
38
39          return this.getLifeCycleManager();
40      }
41
42      @Override
43      public void serviceChanged(ServiceEvent event) {
44
45          IMovementSensor movementSensor = (IMovementSensor)context.getService(event.getServiceReference());
46          switch (event.getType()) {
47          case ServiceEvent.MODIFIED:
48              this.reportMeasure(movementSensor.getMovementDetected());
49              break;
50          default:
51              break;
52          }
53      }
```

Ilustración 33. Métodos de la sonda MovementSensor.

Por último, se sobrescribe el método “serviceChanged” que recibe como parámetro un evento. En este, dependiendo de la sonda, se reporta al monitor el valor del sensor al que está escuchando. En el ejemplo de la ilustración 33 observamos en la línea 48 cómo, a través del método “reportMeasure”, se reporta el valor del sensor de movimiento, previamente identificado en la línea 45.

6. Validación del prototipo

El caso de estudio realizado consta de una serie de farolas (*streetlight*) gestionadas por los controladores (*controller*). Estos controladores y farolas disponen de un dispositivo IoT conectado a la red y gestionados desde AWS IoT, el cual sirve para realizar las comunicaciones entre ellos. Además, disponemos de cuatro tipos de sensores (luz, tiempo, lluvia, movimiento) para poder ajustar la potencia de las farolas según los valores que estos reportan. También necesitamos seguir unas políticas que contienen las configuraciones que se deben de realizar. Todos estos componentes se combinan con el bucle de control MAPE-K para aportar las capacidades de autocuración y autoconfiguración.

Para el prototipo hemos implementado dos farolas, gestionadas por un controlador, junto con los sensores de luz, tiempo, movimiento y lluvia. Asimismo, disponemos de cuatro políticas (*DayTime, LightAmount, RainyDay, MovementDetection*) que son las que se intercambian entre ellas para que el sistema realice un comportamiento u otro. A continuación, vamos a mostrar varias trazas de ejecución a modo de validación del correcto comportamiento del sistema.

6.1. Autoconfiguración inicial

Vamos a probar si la configuración inicial se realiza correctamente. En esta se inicializan todos los sensores, las políticas, las farolas, el controlador y los módulos del bucle de control. La finalidad de esta configuración es, nada más iniciar la ejecución, conseguir que todos los componentes estén listos para realizar distintas configuraciones, además de activar la política “DayTime”, que como decíamos es la política por defecto del sistema, y crear los enlaces con el sensor de tiempo y el controlador.

Inicialmente, el sistema no tiene ninguna configuración, es decir, se encuentran todos los sensores activos, listos para reportar datos, además de las farolas y el controlador. Nos queda indicar al sistema que se quiere una configuración por defecto. Como comentamos anteriormente, queremos que se ejecute la política DayTime, la cual necesita el controlador y el sensor de tiempo, los cuales ya están desplegados.

Se espera que el sistema se configure de un estado inicial vacío, es decir, sin configuraciones, a un estado en el que se encuentre desplegada la política DayTime, de

modo que el sistema es capaz de gestionarse gracias a que se encuentra la política desplegada y en ejecución.

En la traza de la ilustración 34, observamos como, tras iniciar el sistema, se inicializan y despliegan los diferentes módulos (Monitor, Análisis, Planificador, Conocimiento y Ejecutor) y componentes del bucle de control (monitores y reglas de adaptación). Una vez desplegados, tenemos el sistema preparado para realizar cualquier adaptación, pero para nuestro prototipo hemos definido una regla de adaptación que nos configura un nuevo estado del sistema donde entra en ejecución la política “DayTime”.

```
202011171916350551 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [Knowledge] BINDING SERVICE ...
202011171916350551 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [Knowledge] BINDING SERVICE ...
osgi> 202011171916350562 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [Knowledge] SETTING PARAMETER ...
202011171916350562 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [Knowledge] DEPLOYING ...
202011171916350564 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [KnowledgegModule] BINDING SERVICE ...
202011171916350564 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [KnowledgegModule] GET SERVICE SUPPLY ...
202011171916350565 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] BINDING SERVICE ...
202011171916350566 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [KnowledgegModule] GET SERVICE SUPPLY ...
202011171916350566 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] BINDING SERVICE ...
202011171916350568 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] BINDING SERVICE ...
202011171916350568 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] DEPLOYING ...
202011171916350569 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] GET SERVICE SUPPLY ...
202011171916350569 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] BINDING SERVICE ...
202011171916350569 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] DEPLOYING ...
202011171916350570 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [KnowledgegModule] GET SERVICE SUPPLY ...
202011171916350570 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [PlanningModule] BINDING SERVICE ...
202011171916350571 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [PlanningModule] DEPLOYING ...
202011171916350571 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [PlanningModule] GET SERVICE SUPPLY ...
202011171916350571 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] BINDING SERVICE ...
202011171916350572 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [KnowledgegModule] GET SERVICE SUPPLY ...
202011171916350572 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [ExecutingModule] BINDING SERVICE ...
202011171916350572 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [ExecutingModule] DEPLOYING ...
202011171916350572 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [ExecutingModule] GET SERVICE SUPPLY ...
202011171916350573 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [PlanningModule] BINDING SERVICE ...
202011171916350574 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [PlanningModule] BINDING SERVICE ...
202011171916350575 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [ExecutingModule] BINDING SERVICE ...
202011171916350576 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [ARC.SystemEffectors] DEPLOYING ...
202011171916350576 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [Executor] BINDING SERVICE ...
202011171916350577 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] GET SERVICE SUPPLY ...
202011171916350577 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [self-configure-monitor] BINDING SERVICE ...
202011171916350577 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [self-configure-monitor] DEPLOYING ...
202011171916350577 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] GET SERVICE SUPPLY ...
202011171916350578 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [time-sensor-status-monitor] BINDING SERVICE ...
202011171916350578 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [time-sensor-status-monitor] DEPLOYING ...
202011171916350578 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] GET SERVICE SUPPLY ...
202011171916350579 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [light-sensor-status-monitor] BINDING SERVICE ...
202011171916350579 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [light-sensor-status-monitor] DEPLOYING ...
202011171916350579 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] GET SERVICE SUPPLY ...
202011171916350579 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [rainy-day-status-monitor] BINDING SERVICE ...
202011171916350580 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [rainy-day-status-monitor] DEPLOYING ...
202011171916350580 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [MonitoringModule] GET SERVICE SUPPLY ...
202011171916350580 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [movement-sensor-status-monitor] BINDING SERVICE ...
202011171916350580 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [movement-sensor-status-monitor] DEPLOYING ...
202011171916350583 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] GET SERVICE SUPPLY ...
202011171916350583 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [self-configure-rule] BINDING SERVICE ...
202011171916350583 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [self-configure-rule] DEPLOYING ...
202011171916350584 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] GET SERVICE SUPPLY ...
202011171916350584 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [light-amount-sensor-rule] BINDING SERVICE ...
202011171916350584 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [light-amount-sensor-rule] DEPLOYING ...
202011171916350585 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] GET SERVICE SUPPLY ...
202011171916350585 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [rainy-day-sensor-rule] BINDING SERVICE ...
202011171916350585 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [rainy-day-sensor-rule] DEPLOYING ...
202011171916350586 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] GET SERVICE SUPPLY ...
202011171916350586 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [movement-sensor-rule] BINDING SERVICE ...
202011171916350586 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [movement-sensor-rule] DEPLOYING ...
202011171916350587 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [AnalyzingModule] GET SERVICE SUPPLY ...
202011171916350587 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [day-time-sensor-rule] BINDING SERVICE ...
202011171916350587 [INFO ] ....ARC.impl.AdaptiveReadyComponent: [day-time-sensor-rule] DEPLOYING ...
```

Ilustración 34. Log de despliegue de la autoconfiguración inicial.

En la ilustración 35, vemos como la sonda de la configuración inicial, nos reporta el valor “factory-reset”, el cual es un valor que hemos prestablecido para reportar nada más iniciar la ejecución. El monitor recibe ese valor y se ejecuta la regla de adaptación de autoconfiguración. Como comentamos en el desarrollo de la solución, esta es la encargada de iniciar el resto de sondas, las cuales reportan cualquier cambio en el entorno simulado quedando activas a lo largo de la ejecución. Lo siguiente que se despliega es la política “DayTime”, la cual recibe un valor por defecto del sensor para la primera adaptación, seguida del controlador. Finalmente, se realizan los enlaces o “bindings”, al final del log vemos que se realizan dos enlaces, primero la política con el sensor de tiempo, y segundo, la política con el controlador.

```
20201117191635059O [INFO ] ....lite.artifacts.components.Probe: Reporting measure: factory-reset
20201117191635059O [DEBUG] ...ite.artifacts.components.Monitor: Received measure: factory-reset
20201117191635059U [TRACE] ...ules.SelfConfigureAdaptationRule: Executing rule ...
20201117191635059V [INFO ] ...ite.modules.impl.AnalyzingModule:    >>> ## STARTING NEW ADAPTATION PROCESS (83d7d02e649b408c86d9530325bf955a) <<<
20201117191635059W [INFO ] ....ARC.impl.AdaptiveReadyComponent: [time-sensor-probe] BINDING SERVICE ...
20201117191635059W [INFO ] ....ARC.impl.AdaptiveReadyComponent: [time-sensor-probe] DEPLOYING ...
20201117191635059X [INFO ] ....ARC.impl.AdaptiveReadyComponent: [light-sensor-probe] BINDING SERVICE ...
20201117191635059X [INFO ] ....ARC.impl.AdaptiveReadyComponent: [light-sensor-probe] DEPLOYING ...
20201117191635059Y [INFO ] ....ARC.impl.AdaptiveReadyComponent: [rainy-day-probe] BINDING SERVICE ...
20201117191635059Y [INFO ] ....ARC.impl.AdaptiveReadyComponent: [rainy-day-probe] DEPLOYING ...
20201117191635059Z [INFO ] ....ARC.impl.AdaptiveReadyComponent: [movement-detection-probe] BINDING SERVICE ...
20201117191635059Z [INFO ] ....ARC.impl.AdaptiveReadyComponent: [movement-detection-probe] DEPLOYING ...
20201117191635060C [TRACE] ...facts.components.SystemEffectors: DEPLOY day-time-policy_1.0.0
20201117191635060D [DEBUG] ...facts.components.SystemEffectors: Performing the DEPLOY of day-time-policy
20201117191635060D [INFO ] ....ARC.impl.AdaptiveReadyComponent: [day-time-policy] DEPLOYING ...
Cert file:C:\Users\dfo96\eclipse-workspace\streetlights\smartcity.lighting.components\certs\2e963314d1-certificate.pem.crt Private key: C:\Users\dfo96\e
20201117191635063H [TRACE]       policy.daytime.Policy_DayTime: Morning detected, setting power to 0...
Nov 17, 2020 7:16:36 PM com.amazonaws.services.iot.client.core.AwsIotConnection onConnectionSuccess
INFO: Connection successfully established
Nov 17, 2020 7:16:36 PM com.amazonaws.services.iot.client.core.AbstractAwsIotClient onConnectionSuccess
INFO: Client connection active: policiesf4624db7-cd42-4d10-9f4e-2ac879e02def
Nov 17, 2020 7:16:36 PM com.amazonaws.services.iot.client.core.AwsIotConnection onConnectionClosed
INFO: Connection permanently closed
Nov 17, 2020 7:16:36 PM com.amazonaws.services.iot.client.core.AbstractAwsIotClient onConnectionClosed
INFO: Client connection closed: policiesf4624db7-cd42-4d10-9f4e-2ac879e02def
[ day-time-policy ] Applying the policy ...
20201117191636034K [TRACE] ...facts.components.SystemEffectors: DEPLOY controller1_1.0.0
20201117191636034K [DEBUG] ...facts.components.SystemEffectors: Performing the DEPLOY of controller1
20201117191636034K [INFO ] ....ARC.impl.AdaptiveReadyComponent: [controller1] DEPLOYING ...
20201117191636034L [TRACE] ...facts.components.SystemEffectors: BIND day-time-policy_1.0.0::timeSensor -C   o- timeSensor_1.0.0::timeSensor
20201117191636034L [DEBUG] ...facts.components.SystemEffectors: Performing the BIND of day-time-policy::timeSensor -C --- o- timeSensor::timeSensor
20201117191636034L [INFO ] ....ARC.impl.AdaptiveReadyComponent: [day-time-policy] BINDING SERVICE ...
20201117191636034M [TRACE] ...facts.components.SystemEffectors: BIND day-time-policy_1.0.0::controller1 -C   o- controller1_1.0.0::controller1
20201117191636034M [DEBUG] ...facts.components.SystemEffectors: Performing the BIND of day-time-policy::controller1 -C --- o- controller1::controller1
20201117191636034M [INFO ] ....ARC.impl.AdaptiveReadyComponent: [day-time-policy] BINDING SERVICE ...
20201117191636034N [INFO ] ...ite.modules.impl.ExecutingModule: Adaptation success. Updating the current configuration ...
20201117191636034Q [INFO ] ...ite.modules.impl.AnalyzingModule:    >>> ## ADAPTATION PROCESS ENDED (83d7d02e649b408c86d9530325bf955a) <<<
```

Ilustración 35. Log de adaptación de la auto configuración inicial.

Tras la ejecución, podemos comprobar si se ha realizado la configuración indicada ya que el framwork FADA MAPE-K lite mantiene los modelos de las configuraciones. Para ello, escribimos el comando “s” en la misma consola el cual nos permite ver qué servicios están activos dentro de los componentes OSGi. Así, comprobamos que la política “DayTime” se encuentra activa, como vemos en la ilustración 36 (línea 3), también podemos ver como el controlador se encuentra activo (línea 4) además de otros componentes como la sonda de detección de movimiento (línea 2).

```
{es.upv.pros.tatami.adaptation.mapek.lite.artifacts.interfaces.IAdaptiveReadyResource, es.upv.pros.tatami.adaptation.mapek.lite.ARC.artifacts.interfaces.IAdaptiveReadyC
{es.upv.pros.tatami.adaptation.mapek.lite.artifacts.interfaces.IProbe}={enabled=true, service.id=116, service.bundleid=15, service.scope=singleton, id=movement-detectio
{smartcity.lighting.interfaces.IPolicyService, smartcity.lighting.interfaces.IPolicy_DayTime}={active=true, service.id=117, service.bundleid=3, service.scope=singleton,
{smartcity.lighting.interfaces.ILightController}={service.id=118, service.bundleid=2, service.scope=singleton, controller-status=ON, started=false, id=controller1}
```

Ilustración 36. Consola OSGi tras la configuración del estado inicial.

6.2. Autocuración

Vamos a probar como las políticas se reconfiguran automáticamente en caso de un fallo de sensor, aplicando la capacidad de autoconfiguración. Un ejemplo de este caso puede ser cuando el sensor de tiempo reporta un valor desconocido, simulando como si se tratase de un fallo en el mismo.

Partimos del estado previamente configurado en el apartado anterior, donde disponemos de los sensores, el controlador, las farolas y la política “DayTime” activa. Se pretende, desde esta configuración inicial, alcanzar un estado en el que la política “LightAmount” esté activa y, en consecuencia, la política “DayTime” se desactive. Así, se indica que ha habido un fallo en el sensor de tiempo y requiere de un plan de contingencia que permita al sistema recuperarse de dicho fallo.

Para realizar este proceso de autocuración comunicaremos al sensor de tiempo que no se está detectando un valor conocido, por tanto, publicaremos un mensaje en el topic “sensors/time” indicando que el estado del sensor de tiempo es desconocido: {timeStatus:UNKNOWN}.

En el log mostrado en la ilustración 37 se observa cómo se recibe el valor “UNKNOWN” por el topic del sensor. Este es reportado al monitor por la sonda y se lanza la regla “LightSensorFallbackPlanAdaptationRule”. Esta comprueba que la política “DayTime” está activa y comienza con el proceso de adaptación.

Lo primero es deshacer los enlaces creados anteriormente y quitar la política activa, para poder introducir los nuevos cambios. Seguidamente, se realiza tanto el despliegue de la política “LightAmount” como los enlaces del controlador y el sensor de luz con la misma, completando así, el proceso de adaptación.

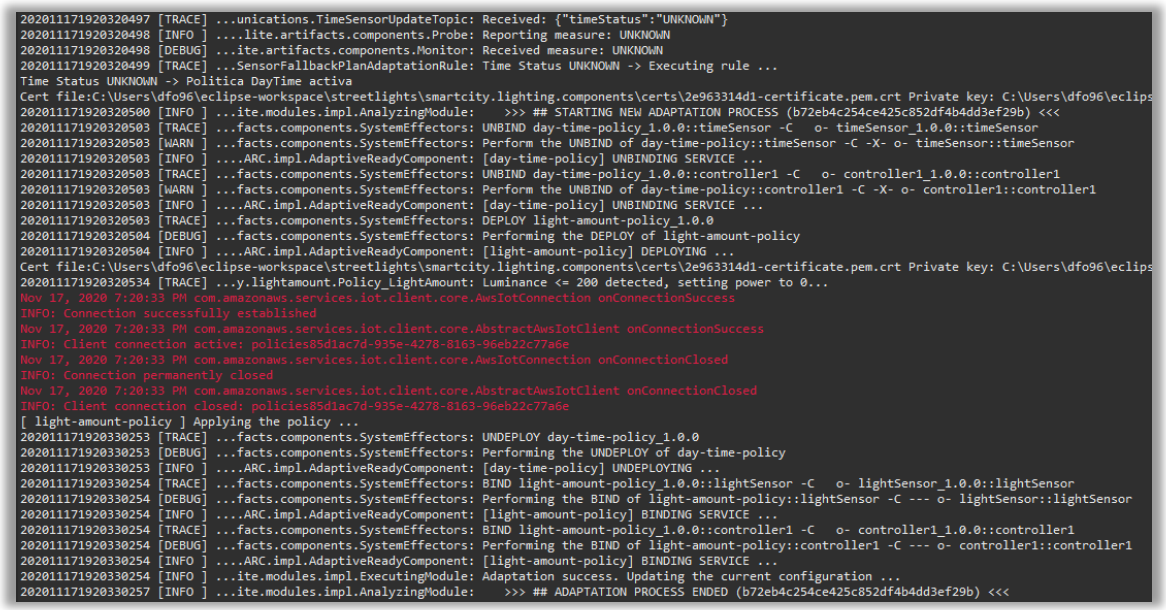

Ilustración 37. Log de la adaptación para autocuración.

En este proceso, se ha detectado un fallo en el sensor de tiempo, por tanto, se ha realizado un cambio de política que obliga a hacer uso del sensor de luz. De este modo, hemos dotado al sistema de autocuración, permitiéndolo seguir en funcionamiento a pesar de que existe un error en uno de los sensores que impide que su ejecución sea correcta.

Del mismo modo que en el caso de la configuración inicial, utilizamos el comando “s” en la consola para comprobar si la autocuración se ha realizado con éxito, desde el estado inicial al actual, con la política “LightAmount” activa. En la ilustración 38 vemos como la política “DayTime” deja de estar activa y se ha desplegado la política “LightAmount” correctamente (línea 4). Además, vemos como el controlador también se encuentra activo (línea 3).

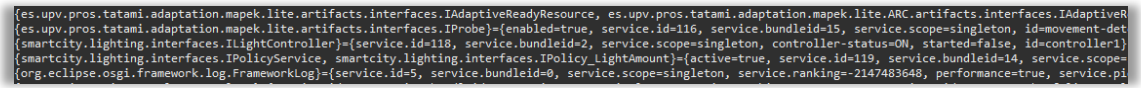

Ilustración 38. Consola OSGi en el estado tras la autocuración.

6.3. Autoconfiguración

Finalmente, vamos a comprobar si el sistema es capaz de autoconfigurarse, adaptándose a los cambios que se realicen en el entorno. Se pretende verificar si al detectar un cambio por alguno de los sensores, el sistema puede cambiar de una política activa a la correspondiente.

Para esta comprobación, partimos del estado de la adaptación anterior. Actualmente, se encuentra activa la política “LightAmount” junto con el sensor de luz y el controlador enlazados. Queremos comprobar que el sistema cambia del estado actual, que hemos realizado en el proceso de autocuración, a un estado en el que se activa la política “RainyDay” ya que en el tramo de la calle se encuentra lloviendo de manera ligera.

En este caso, necesitamos hacer que el sensor de lluvia detecte que está lloviendo, pero con un valor bajo. Esto se lo notificamos publicando un mensaje en el topic “sensors/rain” indicando que el estado que detecta el sensor de lluvia es bajo: {rainStatus:LOW}.

En el log mostrado en la ilustración 39 observamos un comportamiento similar al comentado en el proceso de autocuración. Vemos cómo se recibe el valor “LOW” por el topic del sensor. Este valor es reportado al monitor por la sonda “rainy-day-probe” y se lanza la regla “RainyDayAdaptationRule”. Lo primero que se hace es la comprobación de la política en ejecución, en nuestro caso, “LightAmount”.

A continuación, empieza el proceso deshaciendo los enlaces que existen entre la política en ejecución y su sensor, además del controlador. Seguidamente, se realiza el despliegue de la política “RainyDay” junto con la creación de los enlaces de la misma con el sensor de lluvia, terminando el proceso de adaptación.

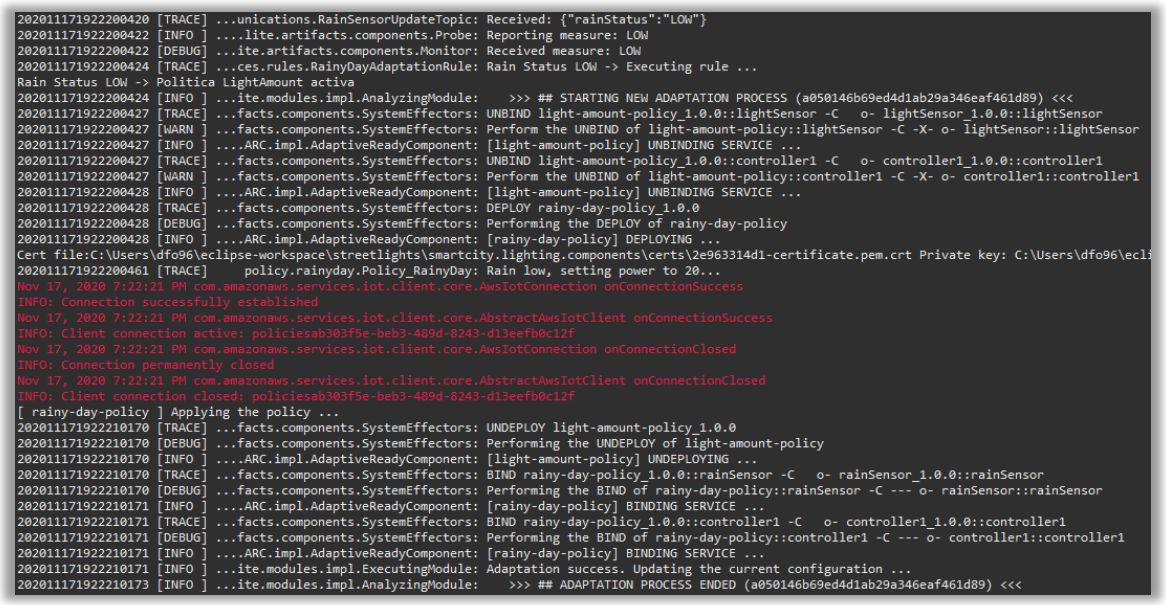

Ilustración 39. Log de la adaptación para la autoconfiguración.

La autoconfiguración se consigue permitiendo al sistema cambiar su estado, de manera dinámica, de una política a otra, tras detectar un cambio en el entorno. En este caso, antes de realizar la adaptación, el sistema se encontraba monitorizando el sensor de luz con la política “LightAmount” activa. Pasado un tiempo, el sensor de lluvia detecta que está lloviendo, y se lanza la regla que permite al sistema cambiar la política actual por la política apta para los días lluviosos, “RainyDay”. Como vemos en la ilustración 40, el sistema ha realizado con éxito el cambio a la política “RainyDay” (línea 4). De modo que, el sistema es capaz de configurarse sin intervención de un usuario.

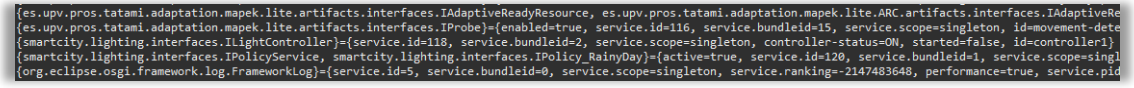

Ilustración 40. Consola OSGi en el estado tras la autoconfiguración.

7. Conclusiones

Como se ha comentado en la introducción y objetivos, este trabajo nace para solventar un problema actual acerca del desarrollo “estático” de soluciones IoT, es decir, las soluciones, una vez desarrolladas, se deben de gestionar de manera manual. Así, se pretende combinar dos dominios distintos, como son el dominio de la IoT y la computación autónoma, ofreciendo las ventajas de cada uno de ellos en un único proyecto.

Los objetivos propuestos son la demostración de cómo aplicar la computación autónoma a las soluciones desarrolladas en la plataforma de AWS IoT. Asimismo, otro de los objetivos es el desarrollo de un escenario donde comprobar si es posible dicha combinación de tecnologías, como hemos visto en la validación del prototipo.

Se ha dedicado un tiempo al estudio y documentación acerca del dominio de la IoT y la computación autónoma. También, se ha realizado un estudio de mercado sobre las diferentes plataformas que ofrecen herramientas y servicios para desarrollar soluciones IoT. Además, se han identificado los problemas actuales en dicho dominio, abordando, en este trabajo, el problema acerca de la gestión y configuración de los dispositivos, realizada de manera estática en la actualidad.

Cabe destacar que, por la complejidad del dominio del trabajo, se ha dedicado un mayor tiempo al estudio y diseño de la solución, para poder realizar la combinación de ambas tecnologías, resultando en un prototipo funcional que permita demostrar el funcionamiento de una solución de dichas características. Además, se consigue mostrar que es aplicable en el dominio presentado, pudiendo crear soluciones IoT preparadas para realizar cualquier modificación sobre el sistema, de manera dinámica y adaptándose al entorno.

Para lograr alcanzar los objetivos, hemos dividido la solución propuesta en dos subsistemas, el proyecto IoT y el bucle de control. El primer objetivo, se logra con el diseño de ambas partes de la solución, de manera conceptual, identificando las carencias del dominio IoT y dotando de las capacidades de autoadaptación que nos ofrece la computación autónoma. Para la consecución de ese objetivo, se aborda un desarrollo en el que se apliquen ambas tecnologías, resultando en el prototipo StreetLights, aplicado de forma experimental.

Al finalizar el desarrollo, se han realizado una serie de configuraciones y ejecuciones para comprobar si el funcionamiento es correcto y que se logran aplicar las capacidades auto-adaptativas de autocuración y autoconfiguración. Gracias a este apartado, hemos podido mostrar como funciona el prototipo desarrollado además de confirmar que es posible la combinación de estas tecnologías, ofreciendo las ventajas que se han identificado a lo largo del trabajo.

7.1. Relación del trabajo desarrollado con los estudios cursados

El trabajo está directamente relacionado con la asignatura “Diseño de Sistemas Ubicuos y Autoadaptativos”. Ésta, despertó el interés del autor en el desarrollo de aplicaciones auto-adaptativas llevándolo un paso más allá y combinándolo con la creación de soluciones IoT.

Además, el estudio de la computación autónoma se realizó con la ayuda de un proyecto presentado en la misma asignatura, el cual facilitó asentar las bases de dicha tecnología junto con el diseño dinámico de este tipo de aplicaciones.

8. Referencias

1. Introducing GitFlow. [En línea] Vincent Driessen. https://datasift.github.io/gitflow/IntroducingGitFlow.html#:~:text=GitFlow%20is%20a%20branching%20model,and%20scaling%20the%20development%20team.

2. *Engineering Self-Adaptive Systems through Feedback Loops Roadmap.* Y. Brun et. al. s.l. : Self-Adaptive Systems, Ed. Springer-Verlag Berlin Heidelberg, 2009. LNCS 5525.

3. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap.* R. de Lemos et. al. s.l. : Self-Adaptive Systems, Ed. Springer-Verlag Berlin Heidelberg,, 2013. LNCS 7475.

4. *An architectural blueprint for Autonomic Computing (4th Edition).* IBM. 2006.

5. *The dawning of the autonomic computing era.* A. G. Ganek, T. A. Corbi. s.l. : IBM Systems Journal, 2003. 10.1147/sj.421.0005.

6. *On how often the Supervisor should sample.* Sheridan., T.B. s.l. : IEEE Transactionson Systems Science and Cybernetics, 1970.

7. *Self-Adaptive Software: Landscape and Research Challenges.* Salehie, M. y Tahvildari, L. s.l. : ACM Transactions on Autonomous and Adaptive Systems, 2009.

8. *A Design Space for Self-Adaptive Systems.* Y. Brun et. al. s.l. : Software Engineering for Self-Adaptive Systems II, 2013.

9. *Modeling Dimensions of Self- Adaptive Software Systems.* Andersson, J., y otros. s.l. : Software Engineering for Self-Adaptive Systems, 2009.

10. *Self-Managed Systems: an Architectural Challenge.* Kramer, J. y Magee., J. s.l. : IEEE Computer Society, 2007.

11. *A Survey of Autonomic Communications.* S. Dobson, et al. s.l. : ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2006.

12. Garlan, David. Foreword by David Garlan. *Managing Trade-Offs in Adaptable Software Architectures.* 2017.

13. *A Survey of Uncertainties in MAPE-K Control Loop.* Selma, O., Boulehouache, S. y Mazouzi., S. s.l. : International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18), 2018.

14. *Towards Enabling Autonomic Computing in IoT Ecosystem.* Mohammad Tahir, Qazi Mamoon Ashraf, Mohammad Dabbagh. s.l. : 2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), 2019. 978-1-7281-3024-8.

15. Amazon IoT Device Shadow Service. [En línea] https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html.

16. MQTT Topics & Best Practices. [En línea] The HiveMQ Team. https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/.

17. Reserved topics - AWS IoT Core. [En línea] https://docs.aws.amazon.com/iot/latest/developerguide/reserved-topics.html.