UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Smart ecoMobility
Disseny d'una solució IoT basada en *Fog/Edge Computing* aplicada a l'àmbit del trànsit sostenible a les ciutats intel·ligents

TREBALL FI DE GRAU

Grau en Enginyeria Informàtica

Autor: Daniel Pedrós i Oliver

Tutor: Joan Fons i Cors
Vicente Pelechano i Ferragud

Curs 2017-2018

Resum

L'objectiu d'aquest projecte és el desenvolupament d'un sistema aplicable a l'àmbit de les ciutats intel·ligents que permeta la computació distribuïda de manera segura en una xarxa de dispositius IoT. El *fog computing* és l'arquitectura emprada per tal de distribuir la computació (i la intel·ligència del sistema) i aproparla on realment són aplicables les mesures que aquesta prenga. Gràcies a aquest repartiment de la intel·ligència, és possible l'aplicació de mesures efectives en temps real.

Una ciutat no és un ent estàtic, sinó que varia les condicions dinàmicament. L'adaptabilitat i escalabilitat del sistema és un factor crític que es busca des del primer moment del disseny. De la mateixa manera, la seguretat de les comunicacions és un factor crític també. Per això, aquest projecte desenvolupa també una capa de seguretat que blinda les comunicacions i assegura la informació d'aquestes.

El resultat del desenvolupament d'aquest projecte és un prototip format per un conjunt de components que poden ser desplegats distribuïdament en un entorn real. Aquest document tracta, des de l'anàlisi de la problemàtica i el disseny de la solució, fins el desplegament dels diferents components finals, passant per tot el procés d'implementació.

Paraules clau: IoT, ciutats intel·ligents, *fog computing*, escalabilitat, adaptabilitat, seguretat, comunicacions

---

Resumen

El objetivo de este proyecto es el desarrollo de un sistema aplicable en el ámbito de las ciudades inteligentes que permita la computación distribuida de manera segura en una red de dispositivos IoT. El *fog computing* es la arquitectura utilizada para distribuir la computación (i la inteligencia del sistema) i acercarla donde realmente son aplicables las medidas que ésta tome. Gracias a éste reparto de la inteligencia, es posible la aplicacón de medidas efectivas en tiempo real.

Una ciudad no es un ente estático, sinó que varia las condiciones dinámicamente. La adaptabilidad y escalabilidad del sistema es un factor crítico que se busca desde el primer momento del diseño. Del mismo modo, la seguridad de las comunicaciones es un factor crítico también. Por eso, éste proyecto desarrolla también una capa de seguridad que blinda las comunicaciones y asegura la información de éstas.

El resultado del desarrollo de este proyecto es un prototipo formado por un conjunto de componentes que pueden ser desplegados distribuidamente en un entorno real. Este documento trata, desde el análisis de la problemática y el diseño de la solución, hasta el despliegue de los diferentes compoenentes finales, pasando por todo el proceso de implementación.

Palabras clave: IoT, ciudades inteligentes, *fog computing*, escalabilidad, adaptabilidad, seguridad, comunicaciones

---

Abstract

The objective of this project is the development of a system applicable in the field of smart cities that allows securely distributed computing on a network of IoT devices. Fog computing is the architecture used to distribute computing (and system intelligence) and bring it closer to where the measures it takes are really applicable. Thanks to this distribution of the intelligence, it is possible to apply effective measures in real time.

A city is not a static entity, but one that varies conditions dynamically. The adaptability and scalability of the system is a critical factor that is considered from the very beginning of the design. Similarly, the security of communications is a critical factor as well. This project also develops a security layer that shields communications and secures their information.

The result of this project development is a prototype formed by a set of components that can be deployed in a real environment. This document covers, from the analysis of the problem and the design of the solution, to the deployment of the different final components, going throug the whole implementation process.

Key words: IoT, smart cities, fog computing, scalability, adaptability, security, communications

Índex

Índex de figures

CAPÍTOL 1
Introducció

En aquest capítol es presenta una introducció al projecte desenvolupat. Es tracten temes com la motivació per a la realització d'aquest, la problemàtica que es pretén resoldre amb el seu desenvolupament, quins són els objectius concrets del projecte que s'esperen assolir i el pla de treball o la metodologia emprada durant tot el procés de desenvolupament.

1.1 Motivació

Vivim en un món que tendeix a estar cada dia més connectat. Cada vegada més dispositius són connectats a Internet, formant així una xarxa amb nom propi, la xarxa de l'IoT (*Internet of Things*). S'estima que cap al 2022 hi hauran 50.000 milions de dispositius IoT connectats arreu del món [1]. L'IoT ha suposat una vertadera revolució i ha obert tot un ventall de possibilitats, tant per a xicotets projectes personals com per a projectes de gran envergadura.

Les *Smart Cities*, o ciutats intel·ligents, són un exemple d'aquests projectes de gran escala que han sigut possibles a partir del naixement de l'IoT. Algunes de les ciutats intel·ligents més importants del planeta són Singapur, Dubai i Londres [2]. Aquestes ciutats ofereixen moltíssims serveis als ciutadans, com, per exemple, una millora en el transport públic gràcies a l'adaptació en temps real de les freqüències dels autobusos segons les necessitats dels ciutadans.

Les ciutats intel·ligents, així com l'IoT en general, suposa un gran repte per a la seguretat dels sistemes. Aquestes ciutats estan formades per molts dispositius connectats entre ells i que poden ser compromesos amb relativa facilitat. No només els dispositius, sinó que les pròpies comunicacions que aquests estableixen entre ells també són susceptibles de ser vulnerades.

Aquest treball ve motivat pel propi repte de desenvolupar un prototip de sistema aplicable a un escenari real d'una ciutat intel·ligent i també d'aportar una capa de seguretat a les comunicacions del sistema.

1.2 Problemàtica

Les ciutats intel·ligents suposen una gran varietat de reptes [3]. Des de reptes relacionats amb la legalitat, normativa i privacitat, fins a limitacions técniques dels sensors. A més, existeixen altres reptes, com les vulnerabilitats de seguretat, l'escalabilitat del sistema o el determinisme de la xarxa de dispositius desplegats. Aquest projecte s'ha centrat en donar solucions a aquests últims aspectes comentats.

En primer lloc, es proposa una solució per a l'escalabilitat i el determinisme de la xarxa. El nombre de dispositius que poden interactuar entre ells dins d'una infraestructura de ciutat intel·ligent és potencialment gran. És per açò que el sistema ha de ser escalable des del propi disseny i ha de contemplar tant l'escalabilitat del nombre de dispositius com l'escalabilitat de les comunicacions i la infraestructura que les suporta.

La segona problemàtica que es proposa resoldre és la seguretat del sistema. Per tal de aconseguir-ho és necessari afegir una capa de seguretat a les comunicacions del sistema. Aquest aspecte és molt important per tal d'assegurar el correcte funcionament d'aquest. Un potencial atacant podria intentar sabotejar el sistema comprometent dispositius que formen part d'aquest o directament injectant o manipulant comunicacions amb algun proposit, com la denegació de servei o la modificació del funcionament legítim.

1.3 Objectius

L'objectiu principal del projecte és el **desenvolupament d'un sistema aplicable a l'àmbit de les ciutats intel·ligents que permeta la computació distribuïda de manera segura en una xarxa de dispositius IoT**. Arran les dues problemàtiques que aquest projecte proposa resoldre, es pot dividir l'objectiu principal en dos subobjectius: el disseny d'un sistema escalable de computació distribuïda mitjançant una xarxa de dispositius IoT i el disseny d'una capa de seguretat per a les comunicacions d'aquest sistema.

Per tal d'assolir el primer dels objectius és necessari dissenyar el sistema basant-se en una **arquitectura *fog/edge computing*** per tal de distribuir la càrrega de còmput entre cadascun dels nodes. A més d'aquest tipus d'arquitectura del sistema, l'arquitectura del programari també ha de tenir-se en compte per tal de desenvolupar un sistema escalable. És per açò que el propi programari ha de seguir un desenvolupament basat en components. En apartats posteriors s'explica en detall aquests conceptes que s'acaben d'introduir.

Respecte al segon dels objectius, cal considerar que sempre hi haurà algú interessat en sabotejar, manipular o inhabilitar el sistema amb algun propòsit, o pot ser, fins i tot sense ningun propòsit evident. És per açò que queda evidenciada la necessitat d'implantar mesures de seguretat per tal de dificultar molt aquests comportaments perjudicials per al sistema. Per tal d'assolir aquest objectiu, es proposa el disseny d'un protocol de *handshake* entre dispositius legítims i el xifrat de les comunicacions que s'estableixen entre aquests. El que s'aconsegueix amb

açò és evitar que dispositius no legítims puguen ser considerats part del sistema i injectar o utilitzar informació d'aquest. També s'evita que es puguen interceptar les dades transmeses pels dispositius.

1.4 Pla de treball i metodologia

Amb els objectius identificats i ben definits, queda traçar un pla de treball per tal d'assolir-los. Per al desenvolupament d'aquest projecte s'ha decidit seguir una metodologia basada en el **desenvolupament iteratiu i incremental**.

El desenvolupament iteratiu i incremental [4] és un procés de desenvolupament de programari que agrupa un conjut de tasques en etapes que es repeteixen al llarg del cicle de vida del projecte. Al final de cada etapa s'obté un producte funcional, que va millorant-se en cadascuna de les iteracions. En resum, la metodologia és iterativa perquè consisteix en diferents tasques que es repeteixen (procés iteratiu) i que aporten o incrementen la qualitat o les funcionalitats del producte (incremental).

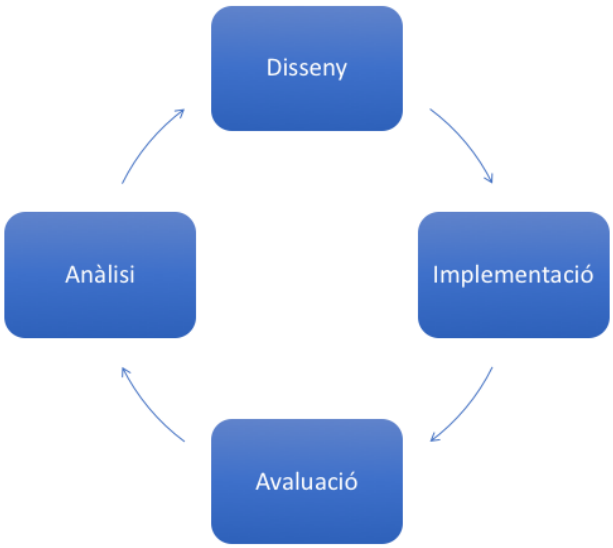

Figura 1.1: Cicle de vida del disseny iteratiu i incremental.

A la figura 1.1 es poden observar les diferents fases que componen una iteració del cicle de vida del desenvolupament iteratiu i incremental. La primera d'elles és la d'anàlisi, que, com el seu nom indica, consisteix en l'anàlisi dels requisits, tant funcionals com no funcionals, dels riscs, de l'abast del projecte, etc. A continuació es troba la fase de disseny, on es dissenya una solució que mitigue els riscs analitzats a la primera fase d'anàlisi i que permeta la satisfacció dels requisits no funcionals. Després del disseny es realitza la implementació de les diferents solucions obtingudes a l'etapa anterior. El resultat d'aquesta fase d'implementació és programari funcional. Per últim, aquestes solucions funcionals passen a una etapa d'avaluació, on es comprova que, efectivament, aquests resultats complei-

xen amb els requisits identificats a la primera fase. En aquest moment, aquestes solucions podrien desplegar-se i posar-se en producció directament. No obstant, i continuant el cicle iteratiu incremental, es torna a començar una nova iteració on s'avalua de nou l'estat del projecte, s'identifiquen nous riscs i requisits i, tenint en compte el programari resultant de la iteració anterior, s'apliquen canvis i millores a aquest per tal de satisfer els nous requisits. Aquest procés es pot repetir tantes vegades com siga necessari, aportant, en cada iteració, un extra de qualitat o funcionalitat al producte.

Aterrant aquesta metodologia sobre el projecte objecte d'aquesta memòria, s'ha decidit realitzar dues iteracions. El motiu d'aquesta decisió ha sigut la divisió del projecte en dues parts, cadascuna de les quals aporta una solució per a un dels dos objectius identificats a l'apartat d'objectius.

La primera iteració del procés se centra en el disseny i desenvolupament d'un sistema escalable que permeta la gestió dinàmica i adaptable del trànsit en l'àmbit de les ciutats intel·ligents, mentre que la segona iteració se centra en el disseny i desenvolupament d'una capa de seguretat que, aplicada al sistema desenvolupat a la primera iteració, incremente la qualitat i seguretat del sistema.

Amb la finalitat d'aterrar sobre un calendari la planificació temporal del projecte, a continuació es mostra un diagrama de Gantt amb un conjunt de tasques bastant genèriques, però representatives del projecte, i l'estimació temporal d'aquestes 1.2. La numeració que s'observa al diagrama correspon amb la numeració de les setmanes de l'any.

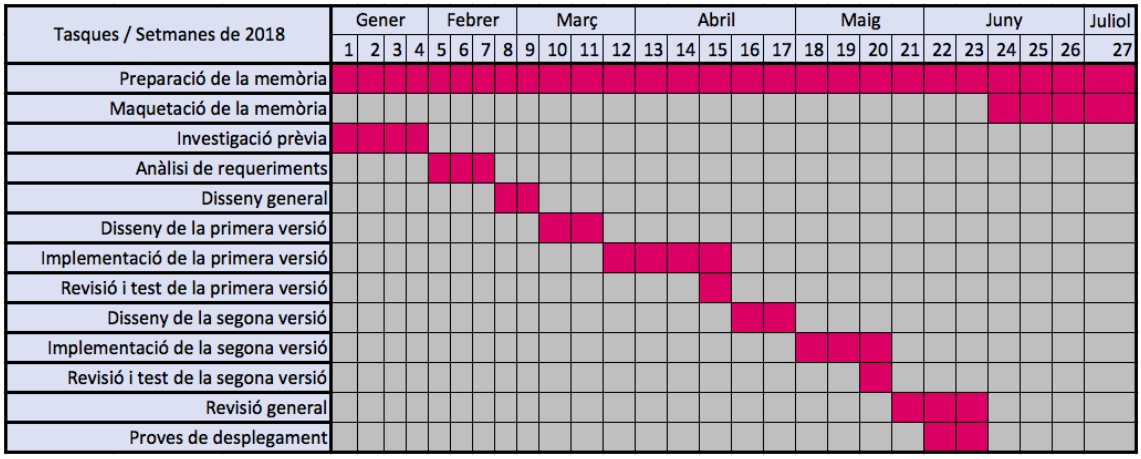

Figura 1.2: Diagrama de Gantt amb la planificació temporal del projecte.

Tal i com es pot observar al diagrama, s'aprecia la planificació de les dues iteracions proposades. Cadascuna d'aquestes iteracions incorpora les seues tasques de disseny, implementació i revisió. També es pot veure que abans de començar les dues iteracions on es duu a terme la implementació, hi ha un període de disseny general del projecte, i que al finalitzar les dues iteracións també hi ha un període de revisió general de projecte. Aquests dos períodes generals és on s'assegura que el projecte compleix amb els objectius establerts.

CAPÍTOL 2
Context tecnològic

Aquest capítol tracta d'establir un context al desenvolupament del projecte. Es comenten les diferents ferramentes, tecnologies, llenguatges de programació, etc. emprats en la realització d'aquest, així com també alternatives a aquestes i altra informació rellevant que pot ajudar a situar el projecte dins d'un context real.

2.1 Desenvolupament: entorn, llenguatge i *framework*

En aquest apartat es comenten l'entorn i el llenguatge de desenvolupament escollits per a dur a terme el projecte i s'expliquen els motius pels quals han sigut seleccionats.

2.1.1. Eclipse IDE

Eclipse és un entorn integrat de desenvolupament programat en Java. Encara que originalment va ser desenvolupat per IBM, actualment és un projecte de codi obert propietat d'Eclipse Foundation [5]. Actualment en la seua versió 4.7 (Eclipse Oxygen), està pensat per al desenvolupament de programari en Java. No obstant, permet també el desenvolupament de programari en altres llenguatges de programació, com per exemple C o C++.

Eclipse està format per diferents components, com OSGi, SWT, JFace i Workbench. L'arquitectura modular i basada en components d'aquest permet afegir-li funcionalitats en forma de *plugins*. Aquests *plugins* poden aportar a Eclipse des de la capacitat de treballar amb altres llenguatges de programació fins a la utilització d'un tema personalitzat per a l'entorn de desenvolupament. També per-

Figura 2.1: Logotip d'Eclipse IDE.

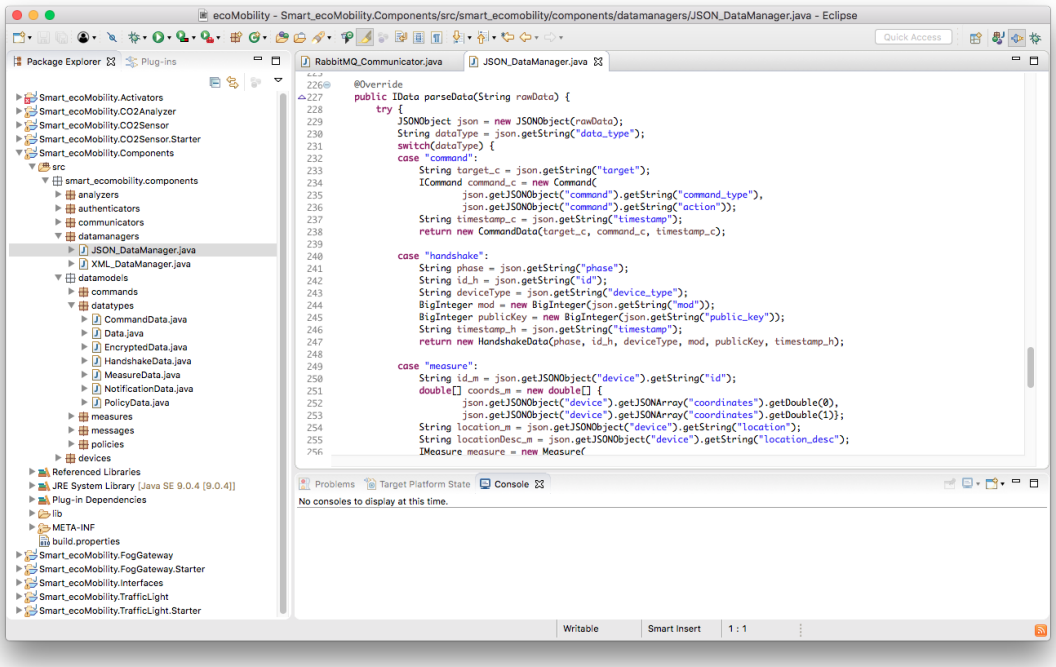

Figura 2.2: *Workbench* d'Eclipse.

meten la utilització de ferramentes externes, com els repositoris de GitHub, o el desenvolupament d'altres tipus de projectes, com per exemple *plugins*. Açò obre la possibilitat de que els usuaris desenvolupen els seus propis *plugins* per al propi Eclipse, encara que aquest ja incorpora un *Marketplace* on existeixen infinitut de *plugins* publicats per altres usuaris i que poden ser instal·lats en l'entorn de desenvolupament.

El principal motiu pel qual s'ha decidit utilitzar aquest entorn de desenvolupament és, com s'ha comentat anteriorment, perquè està especialment pensat per al desenvolupament de programari Java, que és el llenguatge triat per a la realització d'aquest projecte. A més, com a valor afegit, com que Eclipse està desenvolupat en Java, és un IDE (*Integrated Development Environment*) multiplataforma i pot ser utilitzat tant en *macOS*, com en *Windows* i qualsevol distribució de *Linux*.

2.1.2. Java

Java és un llenguatge de programació multiplataforma orientat a objectes dissenyat per *Sun Microsystems* l'any 1990. Encara que inicialment era d'ús privatiu, el 2007, *Sun Microsystems* va alliberar parts de Java com a programari lliure i de codi obert amb llicència GPL. Inspirat pel llenguatge de programació C, Java pot ser entés com una versió simplificada d'aquest.

No obstant, Java presenta diferències molt significatives amb C. La més significativa de totes és que, al contrari que C, que és un llenguatge compilat, Java és un llenguatge interpretat. Realment, Java tampoc és un llenguatge purament interpretat, com podria ser Python, sinó que és semiinterpretat. És a dir, Java és interpretat perquè requereix una Màquina Virtual de Java per a ser executat.

Figura 2.3: Logotip de Java.

Aquesta màquina virtual és qui interpreta el codi Java, però no interpreta el codi directament, sinó que interpreta un *bytecode*, que és el resultat de la compilació del codi Java.

Algunes de les caracteristiques de Java són les següents:

- **Senzillesa**: com s'ha comentat, Java és com una simplificació de C. Elimina alguns tipus de dades primàries, com els punters.
- **Orientació a objectes**: qualsevol element que forma part de Java és un objecte, exceptuant els nombres ordinals i reals, els valors booleans i els caràcters.
- **Dinàmic**: l'entorn de Java pot ser modificat en temps d'execució.
- **Concurrent**: permet l'execució paral·lela de diferents fils.
- **Portable**: el fet d'utilitzar la Màquina Virtual Java permet que els programes escrits en aquest llenguatge puguen ser executats independentment del sistema operatiu.
- **Distribuït**: permet la creació d'aplicacions distribuïdes utilitzant ferramentes pròpies de Java.

El principal motiu pel qual s'ha escollit Java com a llenguatge de desenvolupament per al projecte és per la seua portabilitat i pel fet de poder desenvolupar *plugins* que, juntament amb la tecnologia OSGi, poden ser empaquetats, distribuïts i executats en gran quantitat de dispositius.

2.1.3. OSGi

OSGi (*Open Service Gateway initiative*) és un *framework* Java pensat per al desenvolupament i desplegament de solucions *software* modulars. El principal element utilitzat en OSGi són els *bundles*, encara que existeixen altres elements. A continuació es comenten els principals.

- ***Bundle***: és un conjunt de classes Java i recursos addicionals amb declaració explícita de dependències que poden ser carregats i executats dinàmicament.

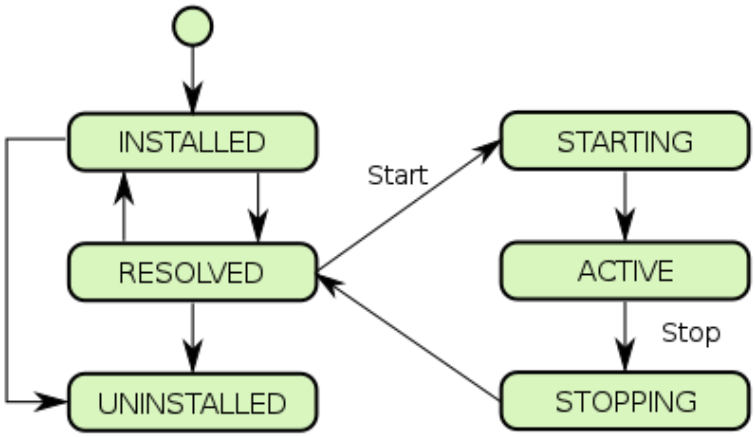

Figura 2.4: Cicle de vida d'un *bundle* OSGi.

- ***Service***: és una capa que connecta dinàmicament els *bundles* proporcionant-los un model *publish-find-bind* a través d'interfícies.

- ***Life-cycle***: s'encarrega de la gestió d'estats dels *bundles*. Aquests estats poden ser els següents i es pot observar el cicle complet a la figura 2.4.

  - ***Installed***: el *bundle* ha sigut instal·lat correctament.
  - ***Resolved***: el *bundle* té resoltes totes les dependències.
  - ***Starting***: s'ha iniciat el procés d'arranc del *bundle* però encara no ha finalitzat aquest procés.
  - ***Active***: ha finalitzat el procés d'arranc i el *bundle* està actiu i funcionant correctament.
  - ***Stopping***: s'ha iniciat el procés de parada del *bundle* però encara no ha finalitzat aquest procés.
  - ***Uninstalled***: és un estat final. El *bundle* ha sigut desinstal·lat correctament i ja no pot arribar a ningun altre estat.

Tal i com comenta la pròpia *OSGi Alliance* amb el seu lema *"The dynamic module system for Java"* [6], la tecnologia OSGi possibilita la componentització d'aplicacions i mòduls *software* i ofereix la capacitat de la gestió i interoperabilitat remòta d'aquests. A més, també facilita la capacitat de modificació d'aquests components, permetent així la fàcil evolució dels sistemes al llarg del temps.

És per aquests motius que s'ha decidit la utilització del *framework* OSGi per a la realització del projecte. Donades les grans necessitats d'escalabilitat que pot suposar una ciutat intel·ligent, la ràpida evolució dels dispositius IoT i la necessitat d'afegir funcionalitats dinàmicament a aquests dispositius, la tecnologia OSGi permet abarcar tots aquests aspectes que s'esperen del sistema.

Figura 2.5: Logotip de GitHub.

2.2 Desenvolupament: control de versions i gestió de projectes

En aquest apartat es parla sobre les ferramentes emprades per tal de dur un control de versions del programari desenvolupat durant tot el procés de programació del projecte. També sobre les ferramentes utilitzades per a mantindre's en contacte i proporcionar actualitzacions sobre l'estat del projecte a totes les persones implicades o vinculades a ell.

2.2.1. GitHub + GitHub Desktop

GitHub és una plataforma de desenvolupament colaboratiu utilitzada per a allotjar codi font de projectes. Utilitza el programari per a control de versions anomenat **Git**, que s'encarrega bàsicament de fer un seguiment dels canvis i modificacions realitzats sobre arxius compartits i d'aquesta manera coordinar de manera eficient el treball que un equip de persones realitzen sobre els fitxers.

El codi font desenvolupat s'emmagatzema en repositoris. Un repositori és una estructura de dades local (emmagatzemada en disc) que guarda les dades i les metadades d'un conjunt d'arxius o estructures de directoris. Totes les modificacions realitzades sobre aquests arxius o directoris són guardades localment i només quan es decideix fer un *PUSH*, utilitzant Git, és quan aquestes modificacions queden reflexades en el repositori remot (que seria GitHub), i per tant, visibles per a tots els membres de l'equip que tinguen accés al repositori remot.

GitHub Desktop no és més que la ferramenta que permet i facilita molt aquestes tasques de descarregar la versió més recent del repositori remot (*PULL*), revisió dels canvis o publicar modificacions al repositori remot (*PUSH*). Es pot veure a la figura 2.6, obtinguda directament des de `https://desktop.github.com`, la revisió de canvis que ofereix l'aplicació.

El motiu pel qual han sigut utilitzades les ferramentes GitHub i GitHub Desktop en aquest projecte ha sigut per tal de facilitar les revisions de codi font per part del director del projecte. Encara que aquest no realitza canvis sobre el propi codi, sempre té disponible l'última versió d'aquest, i per tant, pot realitzar un seguiment del procés de desenvolupament.

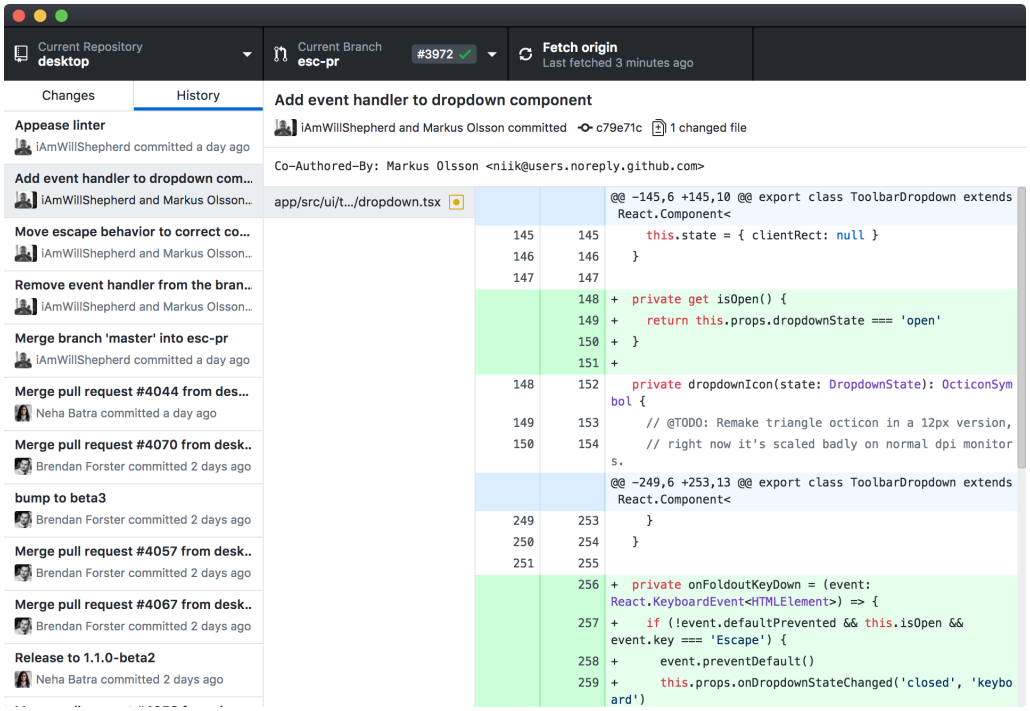

Figura 2.6: Exemple de revisió de canvis utilitzant GitHub Desktop.

2.2.2. Asana

Asana és una aplicació colaborativa amb interfície web i mòbil pensada per a facilitar la gestió de les tasques en el marc d'un projecte en general, no només de desenvolupament de codi. És considerada SaaS (*Software-as-a-Service*) i ofereix moltes funcionalitats interessants per al desenvolupament de projectes col·laboratius.

Figura 2.7: Logotip d'Asana.

Està organitzada en àrees de treball (o *workspaces*) cadascuna de les quals conté un conjunt de projectes en els quals estan involucrats el mateix equip de persones. Cada projecte està format per un grup de tasques, a les quals se'ls pot assignar un responsable, una data límit, una prioritat, etc. Aquestes tasques es poden organitzar com una llista de tasques, amb diferents criteris d'ordenació, o en columnes, seguint la metodologia Kanban, per exemple.

Les tasques ofereixen moltes funcionalitats. A part de poder tenir responsables i dates assignades, poden incloure una descripció, fitxers adjunts, subtasques o, el més important de tot, comentaris. Aquests comentaris els poden incloure els membres del projecte que tinguen visibilitat sobre la tasca i suposen una ràpida

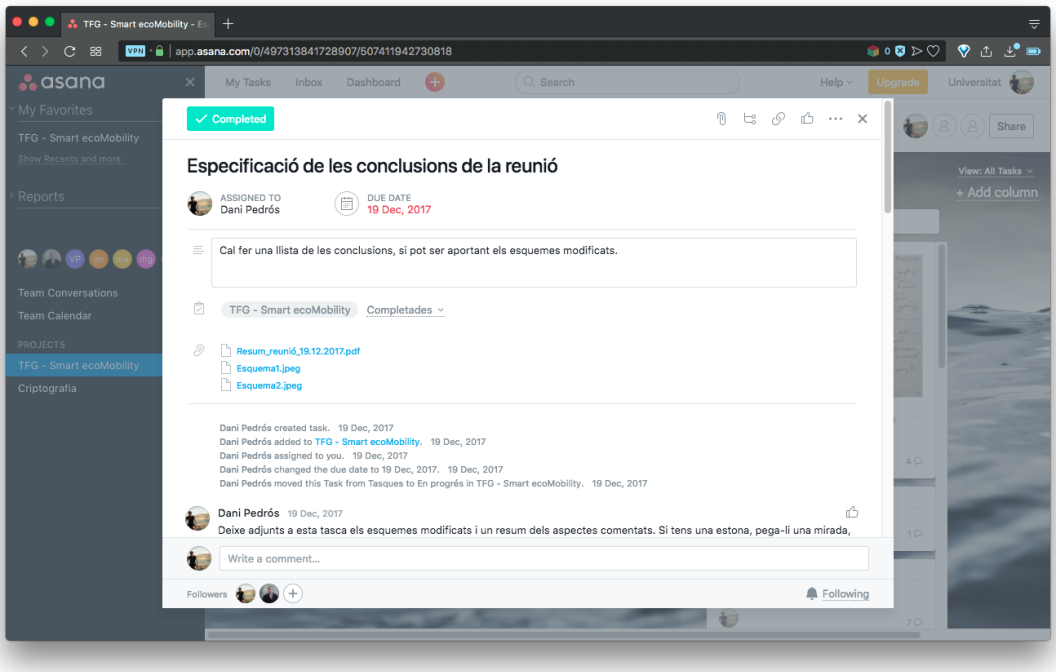

Figura 2.8: Detall d'una tasca a la plataforma Asana.

i efectiva forma de comunicació entre tots els membres. A més, aquests poden subscriure's a les diferents tasques per tal de rebre notificacions quan hi ha qualsevol modificació sobre aquestes. A la figura 2.8 es pot observar el detall d'una tasca.

Els motius pels quals s'ha decidit utilitzar Asana en aquest projecte són, bàsicament, millorar la comunicació amb el director del projecte. A l'igual que amb el cas de GitHub, definir tasques, adjuntar resultats i publicar comentaris a Asana aporta fluidesa en la comunicació. A més, el fet de tenir integrats en la mateixa plataforma els documents, els dubtes i respostes del director, una organització de tasques i un sistema de notificacions ajuda també a una millor organització de les parts implicades en el desenvolupament del projecte.

2.3 Comunicacions

En aquest apartat es comenten algunes tecnologies que són utilitzades per tal d'establir comunicacions entre components i que, per les seues característiques, podrien ser aplicades a un entorn real d'una ciutat intel·ligent.

2.3.1. RabbitMQ

RabbitMQ és un *broker* de missatgeria de codi obert desenvolupat per *Rabbit Technologies Ltd.*, companyia propietat de *VMWare* actualment. Desenvolupat amb el llenguatge de programació Erlang, inicialment implementava el protocol AMQP (*Advanced Message Queuing Protocol*). No obstant, actualment ja implementa tam-

Figura 2.9: Logotip de RabbitMQ.

bé els protocols STOMP (*Streaming Text Oriented Messaging Protocol*), MQTT (*Message Queuing Telemetry Transport*) i altres [7].

RabbitMQ forma part de la capa denominada MOM (*Message Oriented Middleware*), la funció de la qual és facilitar la integració entre les aplicacions. Està format principalment per dues parts. Està, per un costat, el servidor d'intercanvi de missatgeria i, per l'altre costat, les llibreries per als clients Java o els *frameworks* .NET disponibles, a més d'altres llibreries disponibles per a altres tipus de clients.

S'ha decidit utilitzar RabbitMQ per a la realització del projecte ja que implementa cues de missatgeria basades en publicadors-subscriptors que permet assegurar un disseny escalable del sistema. A més, és de codi obert i disposa de clients per a Java, llenguatge de programació sobre el qual s'ha parlat anteriorment, la qual cosa facilita també l'accessibilitat a documentació, etc.

2.3.2. LoRa

LoRa és una tecnologia per a la comunicació sense fils de dades entre dispositius IoT, actualment propietat de l'empresa *Semtech*. Utilitza rangs de radiofreqüència inferiors al GHz i per als quals no es requereix la possessió d'una llicència per a la seua utilització. Aquests rangs de freqüència són 169 MHz, 433 MHz i 868 MHz, per al cas d'Europa, i 915 MHz per al cas dels Estats Units.

Figura 2.10: Logotip de LoRa.

LoRa permet la transmissió de dades a grans distàncies (des de < 2 KM en zones urbanes d'alta densitat, fins a entre 15 i 30 KM en zones rurals) utilitzant una quantitat molt reduida d'energia. Açò permet la utilització del dispositiu sense esgotar la bateria de fins a 10 anys [8].

LoRa està formada per dues parts. La primera d'elles és la part física, anomenada LoRa PHY, que és el protocol utilitzat a la capa física. Aquest protocol és de propietari i no existeix documentació pública disponible. No obstant, existeixen

estudis [9] que han analitzat el protocol. La segona part que composa LoRa és LoRaWAN, que és un MAC (*Media Access Control*) i s'encarrega de la gestió de la comunicació entre els diferents nodes LoRa.

2.4 Hardware

En aquest apartat es comenten alguns dispositius que podrien haver sigut utilitzats per al desenvolupament del projecte. Encara que en un entorn real no és probable que fóren utilitzats, podrien ser-ho perfectament. No obstant, per a la realització de d'aquest projecte, que se centra en la creació d'un model realista però reduit, aquests dispositius ofereixen totes les característiques desitjades.

2.4.1. Raspberry Pi

Raspberry Pi és un computador de placa reduïda desenvolupat per la Funcació Raspberry Pi. Incorpora un processador tipus ARM i una targeta gràfica integrada. Existeixen diferents models, les característiques dels quals difereixen un poc. Les velocitats dels diferents processadors varien entre 700 MHz i 1.4 GHz, i la quantitat de memòria RAM entre 256 MB i 1 GB.

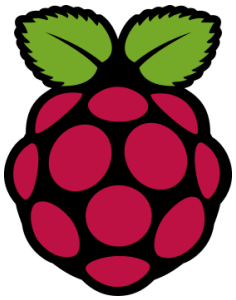

Figura 2.11: Logotip de Raspberry Pi.

Dels diferents models de Raspberry Pi que existeixen, el més adequat per al projecte és el model Raspberry Pi 3B o el Raspberry Pi 3B+. Aquests models incorporen connectivitat sense fils integrada directament a la placa, la qual cosa els fa més atractius de cara a la seua utilització al projecte per la mobilitat que això suposa.

Les característiques del model Raspberry Pi 3B més interessants són les següents:

- **CPU**: 1.2GHz 64-bit quad-core ARMv8
- **GPU**: Broadcom VideoCore IV
- **Memòria SDRAM**: 1 GB compartit amb la GPU
- **Ports USB**: 4 ports USB 2.0
- **Emmagatzematge integrat**: MicroSD

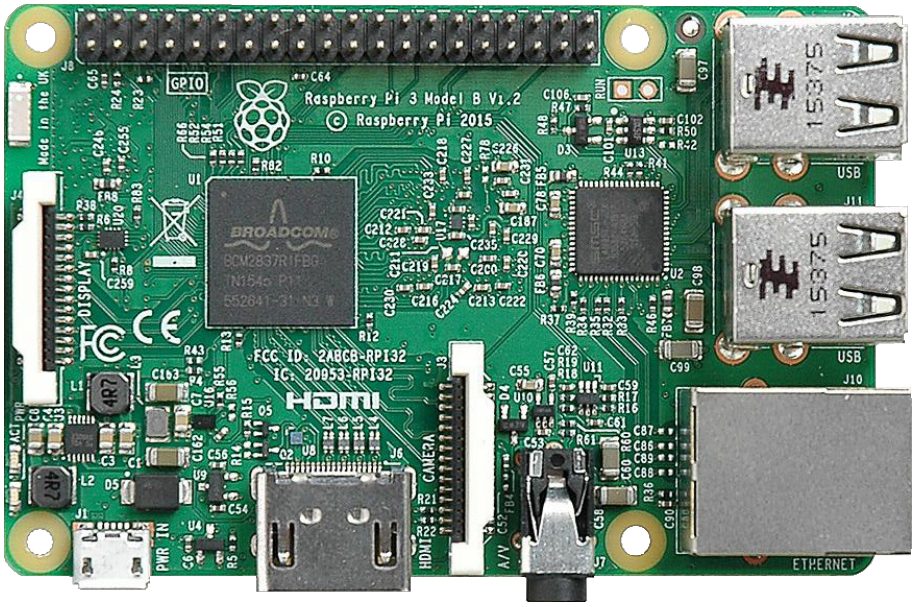

Figura 2.12: Fotografia d'una Raspberry Pi 3 Model B.

- **Connectivitat**: 10/100 Ethernet (RJ-45), WiFi 802.11n i Bluetooth 4.1
- **Perifèrics de baix nivell**: 17 pins GPIO i un bus HAT ID
- **Consum energètic**: 800 mA, 4.0 W
- **Font d'alimentació**: 5 V mitjançant port Micro USB o els pins GPIO

La Raspberry Pi té suport per a diferents sistemes operatius. Raspbian, un derivat de Debian, és el sistema operatiu oficial per a la Raspberry Pi. Aquest sistema operatiu està dissenyat per a oferir un entorn senzill i atractiu a qui comença a interessar-se pel món de la informàtica i la programació. No obstant, i encara que Raspbian permet fer totes les coses que qualsevol *Linux* permet fer, existeixen alternatives més centrades en el món de l'IoT, com per exemple Windows 10 IoT Core, o altres alternatives en general, com Ubuntu Mate, Snappy Ubuntu Core, OSMC o PiNet [10].

2.4.2. Arduino

Arduino és una placa de circuit imprès basada en *open-source*. Aquest maquinari lliure, del qual es poden trobar esquemes i especificacions públiques, utilitza un processador Atmel AVR i incorpora pins d'entrada i sortida d'informació. Per al desenvolupament de programes per a l'Arduino s'utilitza un llenguatge de programació semblant a C++ però més simplificat. Aquestes plaques Arduino poden ser utilitzades per a desenvolupar dispositius autònoms que realitzen alguna funció o com a perifèric d'algun computador, comunicant-se amb el programari d'aquest.

Existeixen diferents models de placa Arduino, com per exemple Diecimila, Duemilanove, Fio, Nano o LilyPad. No obstant, el més conegut i utilitzat és l'Arduino UNO. Les característiques d'aquesta placa són les següents:

Figura 2.13: Logotip d'Arduino.

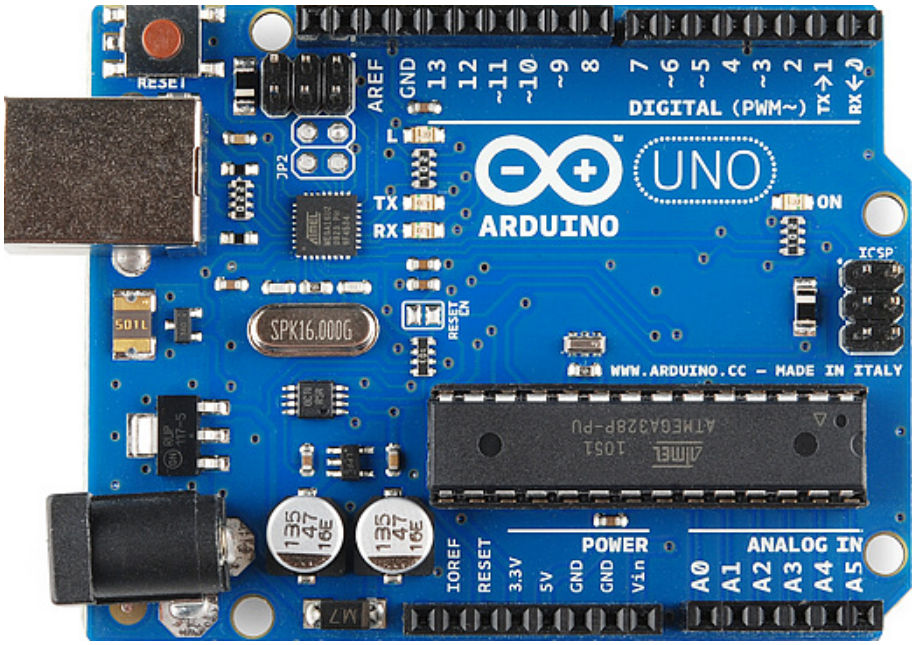

Figura 2.14: Fotografia d'una placa Arduino UNO.

- **Processador**: ATmega328P
- **Memòria FLASH**: 32 KiB
- **Memòria EEPROM**: 1 KiB
- **Memòria SRAM**: 2 KiB
- **Pins d'E/S digital**: 14
- **Pins d'ES analògica**: 6
- **Intefície USB**: ATmega8U2
- **Dimensions**: 68.6mm x 53.3mm

2.5 Desplegament

En aquest apartat es comenten algunes tecnologies utilitzades per tal de desplegar el programari desenvolupat al projecte. No tant com en l'apartat de *hardware*, en aquest cas és altament probable que aquestes tecnologies que es comenten

fóren utilitzades per al desplegament del programari a un entorn de producció real.

2.5.1. Docker

Docker és una plataforma per a desenvolupadors i administradors de sistemes pensada per al desenvolupament, desplegament i posada en marxa d'aplicacions utilitzant contenidors. El que fa Docker, bàsicament, és virtualitzar a nivell de sistema operatiu aprofitant parts del nucli del sistema operatiu amfitrió. Açò és una característica molt interessant ja que el funcionament de diferents contenidors que utilitzen el nucli de *Linux*, per exemple, implica que aquest nucli es carrega una única vegada, no tantes com contenidors hagen sigut arrancats.

Al contrari que utilitzant Docker, una virtualització convencional utilitzant màquines virtuals, necessitaria carregar no només el nucli, sinó tot el sistema operatiu tantes vegades com màquines virtuals s'arranquen. És per açò que Docker resulta tan interessant, ja que estalvia gran utilització de recursos del sistema operatiu amfitrió i del maquinari en general. Es poden apreciar les diferències a la figura 2.15.

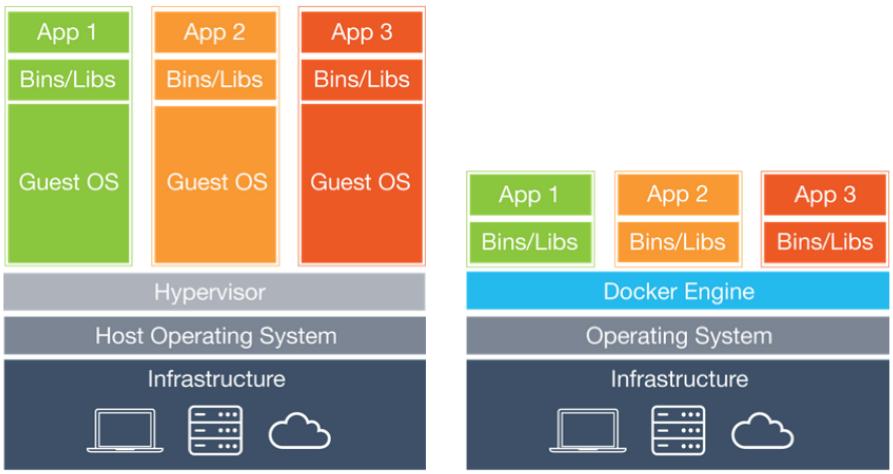

Figura 2.15: Diferències entre utilització de contenidors Docker i virtualització amb màquines virtuals.

Algunes característiques que ofereix Docker són les següents [11]:

- **Flexibilitat**: la capacitat de crear contenidors d'aplicacions no es veu afectada per la complexitat d'aquestes.
- **Lleugeresa**: els contenidors aprofiten i comparteixen el propi nucli del sistema operatiu amfitrió.
- **Dinamicitat**: els contenidors desplegats poden ser modificats i actualitzats dinàmicament.
- **Portabilitat**: es pot desenvolupar localment, desplegar online i arrancar en qualsevol sistema.

Docker està format per diferents components. El principal d'ells, que és el que ofereix tota la funcionalitat, és el propi *software* de Docker, que, al mateix temps, està format per diversos components. El primer d'ells es tracta d'un procés persistent que s'encarrega de la gestió dels contenidors i objectes. Aquest procés té també una API associada i atén a peticions del Docker API Engine. El segon component és el client Docker, que és l'encarregat d'interactuar amb el primer mitjançant el Docker API Engine. Permet a l'usuari la gestió dels contenidors i de Docker en general mitjançant una interficie CLI (*Command Line Interface*).

El segon component que forma Docker són els propis objectes de Docker. Existeixen diferents tipus d'objecte de Docker. Els principals són els contenidors. Un contenidor és un entorn encapsulat i estandaritzat preparat per a arrancar aplicacions i que és gestionat mitjançant la interficie CLI de Docker. Per tal de crear contenidors es necessita un altre objecte de Docker, anomenat imatge. Una imatge és una plantilla no modificable utilitzada únicament per a la creació de contenidors.

Docker ofereix també ferramentes per tal de facilitar el desplegament de solucions. Els Dockerfile són documents de text que inclouen instruccions que Docker interpreta i permeten crear imatges personalitzades. A més dels Dockerfile, existeix també una ferramenta anomenada Docker Compose. Aquesta ferramenta permet la definició i desplegament de solucions multi-contenidor. Utilitza un document YAML per a la configuració dels serveis i s'encarrega de la creació i la posada en funcionament dels contenidors.

2.5.2. Shell script

Un *shell script* és un programa dissenyat per a ser executat utilitzant el *shell* de *UNIX*. El *shell* de *UNIX* és un programa que s'encarrega d'interpretar una seqüència de línies de text introduides per un usuari o llegides des de qualsevol altre tipus de font de dades, com per exemple un arxiu, en el cas dels *scripts*.

```
1  #!/bin/bash
2
3  bashtrap() {
4      echo "CTRL+C detected! Executing bash trap..."
5  }
6
7  trap bashtrap INT
8
9  for a in `seq 1 10`; do
10     echo "$a/10 to exit."
11     sleep 1;
12 done
13 echo "Exit!"
```

Figura 2.16: Exemple de shell script.

Existeixen determinats tipus d'*script* que s'encarreguen de preparar un entorn d'execució, executar un programa, gestionar els *logs* i realitzar una "neteja"de l'entorn en finalitzar l'execució del programa. Aquest tipus d'*script* s'anomenen *wrappers*. Aquestes tasques són, concretament, les següents:

- **Control de l'execució del programa principal**: el *wrapper* s'encarrega d'establir les variables d'entorn necessaries, crear els àlies corresponents, etc.
- **Proporcionar informació sobre l'execució del programa principal**: el *wrapper* s'encarrega de mostrar el temps d'execució, redirigir l'*output* de l'execució a un fitxer de *log*, mostrar el codi de retorn de l'execució, etc.
- **Facilitar un únic punt d'entrada per a l'execució de multipes scripts**: per exemple, si el programa principal està format per l'execució de diferents *scripts*, el *wrapper* és qui s'encarrega d'executar totes aquestes parts que formen el programa principal.

Com que existeixen diferents interprets que poden encarregar-se de l'execució dels scripts, cal indicar quin de tots és qui deuria realitzar l'execució. Es pot observar a la figura 2.16 com la línia 1 inclou el text "#! /bin/bash". Açò no és realment un comandament, sinó que la seua funció és indicar al sistema quin interpret és l'encarregat d'executar l'*script*. En el cas de l'exemple es tracta del *shell* anomenat Bash. No obstant, podrien utilitzar-se altres *shells* natius de *UNIX*, com per exempls SH (#! /bin/sh), o fins i tot interprets d'altres tipus de llenguatges, com per exemple Python (#! /usr/bin/python). Realment, el que s'indica en aquesta línia és la ruta fins a l'executable de l'interpret que s'ha d'encarregar de l'execució de l'*script*.

2.6 Amenaces reals

En aquest apartat es comenten atacs perpetrats i amenaces que afecten al món dels dispositius IoT. És important coneixer i analitzar antecedents que han afectat a sistemes similars per tal d'intentar dissenyar el sistema pensant en mitigar o eliminar aquests riscs des del primer moment.

2.6.1. Botnet Mirai

Mirai és un malware que es dedica a infectar dispositius connectats a Internet i que utilitzen el sistema operatiu *Linux*. Una vegada infectats, aquests disposius es converteixen en xicotets "robots"que poden ser controlats remotament, creant així una *botnet* que pot ser utilitzada en ciberatacs d'escala massiva.

Descoberta a l'agost de 2016 per *MalwareMustDie*, la *botnet* Mirai ha sigut utilitzada en alguns dels majors ciberatacs DDoS (*Distributed Denial-of-Service*). Entre aquests atacs és interessant mencionar la denegació de servei que va sofrir la pàgina web de Brian Krebs, periodista del món de la informàtica. A més d'aquest, la *botnet* Mirai també va ser utilitzada en l'atac *Dyn cyberattack*, que va tindre lloc a l'octubre de 2016 i, com a conseqüència, va deixar sense serveis a grans plataformes d'Internet, com Amazon, GitHub, Netflix, PayPal, etc.

El codi font de Mirai va ser publicat a alguns fòrums. Brian Krebs, el periodista que va veure el seu lloc web afectat per un atac DDoS associat a Mirai, va publicar un article on apuntava a que el possible autor del codi era Paras Jha baix el pseudònim d'Anna-senpai. És rellevant ja que Paras Jha és el propietari d'una

empresa que ofereix serveis de mitigació front a atacs DDoS. No obstant, aquest va desmentir que fóra ell l'autor del malware.

Els dispositius infectats per Mirai estan constantment escanejant les direccions IP d'Internet buscant dispositius IoT que siguen vulnerables. Una vegada identificat un dispositiu vulnerable, aquest és infectat i així continua el cicle. És complicat identificar un dispositiu afectat per Mirai ja que aquests no presenten "sintomes"apreciables. Funcionen amb total normalitat i només es nota un xicotet augment en l'ample de banda consumit pel dispositiu.

Un dispositiu està infectat fins que aquest és reiniciat. Una vegada reiniciat aquest dispositiu ja no està afectat per Mirai, però és molt probable que torne a ser infectat en un temps inferior a un parell de minuts. El fet de que el dispositiu deixe d'estar infectat després de ser reiniciat no suposa massa problema per a Mirai, ja que, com s'ha comentat, és un *malware* dirigit a dispositius IoT, i aquests dispositius solen estar disponibles quasi el 100% del temps, és a dir, no són reiniciats de manera habitual.

El cas de Mirai és rellevant per al projecte ja que una ciutat intel·ligent està formada per molts dispositius IoT connectats a Internet i interconnectats entre ells. Si no es prenen mesures i consideracions, un *malware* com Mirai podria impactar de manera crítica sobre el sistema que dona suport a una ciutat intel·ligent, en el millor dels casos només per a dur a terme atacs de denegació de servei, o en el pitjor d'ells, oferint control total d'aquesta infraestructura a un atacant.

CAPÍTOL 3
Plantejament inicial del problema

Una vegada introduides les problemàtiques que es pretenen resoldre en aquest projecte, establits els objectius necessaris per a aconseguir-ho i posades en context les ferramentes i tecnologies, amenaces i riscs i el projecte en sí, és moment de parlar de solucions. En aquest apartat es comenta com es pretén abordar la solució de les diferents problemàtiques i donar compliment als objectius establerts.

3.1 Sobre dispositius IoT

Tal i com s'ha comentat al context tecnològic, els dispositius IoT són petits computadors. En l'àmbit del projecte i de les ciutats intel·ligents, aquests són els més apropiats per al disseny de les solucions. Compleixen amb les necessitats de còmput mínimes i el seu tamany i portabilitat els fa molt atractius per al seu desplegament en zones obertes.

A més d'aquestes característiques, els dispositius IoT es caracteritzen també per un consum mínim d'energia, permetent la seua alimentació a traves de xicotetes plaques solars, per exemple. D'aquesta manera s'aprofita així l'energia distribuïda del Sol i s'augmenta la possibilitat de dispersió d'aquests, deslligant-los de les fonts d'alimentació cablejades. Açò augmenta en gran mesura l'àrea que aquests dispositius són capaços de cobrir i facilita la seua mobilitat.

Respecte al projecte, s'ha decidit diferenciar dos tipus diferents de dispositiu. El primer d'ells és el que s'anomena *gateway* i fa referència al node central del *fog*, concepte que s'explica al següent apartat. Aquests nodes requereixen d'una capacitat de còmput un poc superior a la que requereixen el segon tipus de dispositiu, que són els sensors i actuadors. El sensor o actuador no requereix d'aquest extra de capacitats ja que la seua funció és més simple i no necessita realitzar tasques de computació elevades.

Tenint en compte les diferents tecnologies *hardware* comentades al context tecnològic, la Raspberry Pi, amb capacitat computacional de sobres, podria ser sel·leccionada com a *gateway*, mentre que l'Arduino podria fer funcions de sensor o actuador sense ningun problema. No obstant, la Raspberry Pi pot ser utilitzada tant com a node central o com a sensor o actuador. Fins i tot, podria utilitzarse com a sensor o actuador una Raspberry Pi connectada a una Arduino, fent la segona placa, funció de perifèric de la primera.

3.2 *Fog computing*: disseny i aplicació

El *fog computing*, o computació en el *fog*, fa referència a l'extensió de la computació en el núvol, o *cloud computing*, descentralitzant aquesta computació, apropant-la a la vora de la xarxa i distribuïnt-la entre diferents dispositius [12].

Els serveis basats en *fog computing* ofereixen una latència reduida i una millora en la qualitat del servei. Açò és possible perquè la font de les dades i l'encarregat del processament d'aquestes estan "prop", gràcies al fet d'acostar a la vora de la xarxa aquest segon element. És per aquest motiu que la computació en el *fog* és especialment aprofitable en sistemes que requereixen resposta en temps real o en un període de latència predictible. Exemples d'aquests sistemes són els sistemes de transport automatitzat industrial o les xarxes de sensors i actuadors, que és exactament el sistema que proposa aquest projecte.

A més de les característiques comentades, el *fog computing* n'ofereix algunes més. Aquestes característiques són:

- **Reducció del trànsit de dades per la xarxa**: ja que al distribuïr la computació entre diferents nodes que estan relativament prop de les fonts de dades, aquestes dades no han de travessar tota la xarxa fins a arribar on han de ser processades. Açò redueix la congestió de la xarxa i evita colls de botella que podrien donar-se en sistemes centralitzats.
- **Descentralització de la computació**: reduïnt la criticitat del punt de fallada que seria el sistema centralitzat.
- **Augment de l'escalabilitat del sistema i la tolerància a fallades**: gràcies a la utilització d'una arquitectura distribuïda en la qual cada component afecta només a un subconjunt del sistema. Açò significa que si falla un node del *fog* només es veuen afectats els nodes que depenen d'aquest i tots els altres poden continuar amb el seu funcionament normal.

El *fog computing* sembla apareixer per a llevar-li el lloc a la computació en el *cloud*, però no és així. Més bé, ambdós tipus de computació són complementaris i necessaris al mateix temps. A la figura 3.1 s'observa les diferències i similituts que existeixen entre la computació en el *cloud* tradicional i la basada en *fogs*.

A més de totes les característiques exposades, el *fog computing* resulta ser una arquitectura òptima per al sistema que es proposa en aquest projecte, ja que s'adapta perfectament a la segmentació geogràfica de l'arquitectura. Al següent capítol s'exposa de manera detallada aquest concepte de la segmentació geogràfica del sistema.

La computació en el *fog* és aplicable a molts sistemes, la majoria d'ells relacionats amb el món de l'IoT. Alguns exemples d'aquests sistemes són els IoT CPSs (*Cyber-Physical Systems*), o també les SDN (*Software Defined Networks*). No obstant, els que major interés aporten per a l'objectiu d'aquest projecte són els sistemes WSAN (*Wireless Sensor and Actuator Network*), CV (*Connected Vehicles*) i els SBC descentralitzats (*Decentralized Smart Building Control*).

Els sistemes WSAN són sistemes on hi ha un flux bastant definit de la informació. Un conjunt de generadors d'informació (sensors) generen dades que són

Característica	*Cloud computing*	*Fog computing*
Latència	Alta	Baixa
Retard del Jitter	Alta	Molt baixa
Localització del servei	A qualsevol lloc d'Internet	A la vora de la xarxa local
Distància entre client i servidor	Multiples salts	Un únic salt
Seguretat	No definida	Pot ser definida
Atac a l'enrutament de les dades	Alta probabilitat	Molt baixa probabilitat
Personalització del servei segons ubicació	No	Sí
Distribució geogràfica	Centralitzada	Distribuïda
Quantitat de nodes	Pocs	Molts
Suport per a la mobilitat	Limitat	Suportada
Interacció en temps real	Suportada	Suportada

Figura 3.1: Diferències entre la computació *cloud* i la computació *fog*.

transmeses unidireccionalment cap als processadors de dades. Aquests s'encarreguen de tractar eixa informació i generar comportaments que són transmesos fins als receptors finals (actuadors). Aquest tipus de sistema s'adapta perfectament al plantejament de solució que proposa aquest projecte. No obstant, és important tenir en compte també els sistemes CV i SBC ja que en un futur poden ser integrats al resultat d'aquest projecte.

3.3 Comunicacions en el *fog* amb l'IoT

Tenint en compte l'arquitectura naturalment distrubuïda del sistema que es proposa, i sabent, també, que l'IoT es defineix com a una xarxa de moltíssims dispositius connectats entre ells, és evident l'aparició de determinades necessitats que encaixen perfectament amb els objectius del projecte. Aquestes necessitats són:

- **Potenciar l'escalabilitat tenint en compte les comunicacions**
- **Gestió del consum d'energia i de l'ample de banda**
- **Identificar tots els dispositius connectats**
- **Xifrar i autenticar les comunicacions**
- **Disseny de les dades**

Potenciar l'escalabilitat tenint en compte les comunicacions és un punt important ja que s'ha de considerar des del primer moment del disseny del sistema. És important la segmentació de les comunicacions per tal de tenir sempre acotada la quantitat de dades que ha de suportar el sistema en cas de que aquest escale. Com millor siga la segmentació d'aquestes comunicacions, menor serà la sobrecàrrega

del sistema, produint una menor degradació del rendiment amb l'augment de la quantitat de dades que siguen transmeses.

A més d'una correcta segmentació de les comunicacions, el disseny d'aquestes ha de permetre la fàcil i ràpida integració dels dispositius. Açò també influeix en l'escalabilitat del sistema, ja que afegir nous dispositius és fàcil i ràpid, no perjudica greument el rendiment global del sistema. De la mateixa manera que l'adició de dispositius, l'eliminació o desactivació d'aquests tampoc penalitza greument al sistema. S'aconsegueix així el disseny d'un sistema dinàmic que varia el nombre de dispositius que el formen de manera dinàmica i sense comprometre el seu rendiment.

La gestió del consum de l'energia consumida per aquests dispositius també és un factor que s'ha de contemplar des del moment en el qual es dissenya la solució. Els diferents dispositius van a estar desplegats en zones obertes i és possible que no disposen de fonts estàndar d'alimentació. Podria donar-se el cas de que l'alimentació hagués de ser a través de plaques solars, la qual cosa suposa una limitació i s'ha d'estudiar. Per tant, seria interessant la utilització de tecnologies que permeten la transmissió d'informació amb un baix impacte energètic, com per exemple LoRa.

No obstant, i suposant que els dispositius disposaran de fonts d'alimentació fiables, aquest problema passa a un segon plà. No significa això que no haja de ser estudiat, però sí que no cal limitar tant el consum d'energia. D'aquesta manera, podrien utilitzar-se tecnologies que ténen un impacte energètic major, com per exemple la tecnologia WiFi, però que ofereixen millores respecte a altres tecnologies més economitzadores del consum.

A més de la gestió del consum d'energia, és important també tenir en compte el consum d'ample de banda que permeten les diferents tecnologies. Si aquest està limitat, com en el cas de la tecnologia LoRa, açò influeix també en el disseny de les dades que seran transmeses, com es comenta en paràgrafs vinents. És important establir un balanç òptim entre el consum d'energia i la necessitat d'ample de banda del sistema. Si aquest necessita la transmissió de molta informació, l'impacte energètic del dispositiu serà major, i s'haurà d'escollir una tecnologia que permeta optimitzar el consum d'energia respecte l'ample de banda que siga necessari.

Sabent que l'IoT és una xarxa de moltíssims dispositius connectats, i que la solució que es proposa ha de ser escalable i permetre la integració de tots aquests dispositius de manera dinàmica, seria remarcable també el fet de disposar d'un registre de tots els dispositius que formen part del sistema. Per tal de la realitzar una gestió i identificació de tots aquests dispositius, s'han de dissenyar mecanismes que ho permeten. El disseny d'un protocol de *handshake* pot donar solució a aquest problema de la gestió i identificació dels dispositius.

El protocol de *handshake* hauria d'encarregar-se d'identificar i autenticar els dispositius que formen part del sistema. És a dir, una vegada realitzat el *handshake*, és indiscutible que el dispositiu que ha fet la petició està autoritzat a formar part del sistema. A més, aquest dispositiu queda registrat i és identificable.

A més de la funció d'identificació, és possible l'extenció de les possibilitats que ofereix aquest protocol. La principal, i que dóna peu al que s'explica al següent

paràgraf, és la de la derivació d'un secret compartit entre qui sol·licita el *handshake* i qui el resol. Aquest secret compartit es pot utilitzar, entre altres coses, per al xifrat i autenticat de les comunicacions.

El xifrat de les comunicacions permet que, en cas de que aquestes siguen interceptades, no puguen ser interpretades, ja que no seria possible accedir al contingut de la informació, i per tant, impossible també el coneixer les dades que són transmeses. A més del xifrat, també és important l'autenticació de les dades. Autenticar les dades permet al receptor saber que les dades són realment d'un dispositiu que està identificat i que és legítim. Conjuntament, les dades autenticades i xifrades impossibilita a algú que intercepte les dades, que aquestes puguen ser modificades o interpretades.

Per últim, el disseny de les dades que van a ser transmeses és molt important també. El format d'aquestes i la quantitat d'informació influeixen de manera important en el rendiment del sistema. Un format que incorporte molta informació de control, com per exemple simbols { o , augmenta la quantitat de dades a ser transmeses sense aportar informació útil. De la mateixa manera, si les comunicacions envien molta informació que no és útil per a qui l'ha de processar, s'estan consumint molts recursos sense ser necessari. Enllaçant amb el que s'ha comentat sobre el consum d'energia i l'ample de banda de les comunicacions, en cas de que aquests siguen limitats, un bon disseny de les dades és crític per a l'òptim funcionament del sistema.

3.4 Conclusions prèvies

Es poden obtindre algunes conclusions prèvies al disseny concret del projecte. Tenint en compte el primer apartat d'aquest capítol es desprén la diferenciació de dos tipus de dispositius dins del sistema. Sensors i actuadors, que són els més senzills d'ells i no requereixen de gran capacitat de còmput. I *gateways*, o nodes centrals, que ténen unes necessitats de còmput un poc més elevades.

Continuant amb les conclusions que es desprenen del segon apartat, és evident que la computació en el *fog*, o *fog computing*, és l'arquitectura més apropiada per al disseny d'aquest projecte, ja que permet la descentralització de la intel·ligència i l'adaptació d'aquesta segons la seua àrea d'influència, personalitzant el funcionament del sistema respecte a la geolocalització de les diferents parts d'aquest. A més d'açò, la reducció del transit de dades per la xarxa augmenta el rendiment del sistema, permetent així l'augment de l'escalabilitat i la tolerància a fallades d'aquest.

Finalment, de l'últim apartat s'obté la necessitat de segmentar les comunicacions per tal de propiciar l'escalabilitat del sistema, acotant l'abast de les dades que es transmeten. A més de la segmentació, el disseny de les comunicacions ha de contemplar la gestió del consum d'energia i ample de banda de les tecnologies emprades, així com la incorporació d'un protocol que permeta la identificació dels dispositius i el xifrat i autenticat de la informació transmesa. Per últim, el disseny de les pròpies dades és on finalment es veu reflexat tot el que s'acaba de comentar.

CAPÍTOL 4
Anàlisi del problema

Amb una idea del problema mitjanament definida al capítol anterior, en aquest capítol es presenta una anàlisi en profunditat d'aquest. S'abarca, des d'un plànol més conceptual, on es tracten temes com els diferents tipus de components que formaran el sistema i protocols relacionats amb la seguretat de les comunicacions, fins a un plànol més organitzatiu, on es parla de la segmentació del sistema per tal de modelar la ciutat. Aquests temes que es tracten en l'anàlisi del problema són els que marcaran posteriorment el disseny de la solució.

4.1 Organització del sistema

En aquest apartat es tracta l'organització general del sistema. Es plantegen diferents maneres de dissenyar el sistema amb l'objectiu de modelar la ciutat de tal manera que es facilite la solució de la problemàtica que haurà de resoldre el sistema que s'està dissenyant. És a dir, per a la solució d'un determinat problema, de les diferents possibilitats de disseny s'escollirà aquella que proporcione solucions més òptimes per al problema.

Sobre el modelat del sistema objecte d'estudi d'aquest projecte, i tenint present l'arquitectura distribuïda de les solucions basades en *fog computing*, es plantegen diferents maneres de segmentar la ciutat. Aquesta segmentació té sentit amb la intenció de distribuir la intel·ligència i apropar-la a la zona d'acció on realment ténen efecte les decisions preses per aquesta. Encara que es podrien pensar més maneres de segmentar la ciutat per tal de satisfer determinades necessitats, en el que aquest projecte respecta, es plantegen dos principals perfils de segmentació.

4.1.1. Segmentació basada en la geografia

La primera forma de segmentar el sistema per tal de modelar la ciutat és la que basa la segmentació d'aquest en com està dividida la ciutat. Aquesta possibilitat és la que, pot ser, sembla més "natural"per als humans, ja que és la manera tradicional de particionar les ciutats.

Bàsicament, la segmentació basada en la geografia consisteix en partir el sistema i distribuïr-lo seguint un esquema de partició ja establert i que està aplicat a la ciutat. Un clar exemple d'aquest tipus de particionat seria el modelat de la ciutat

tenint en compte els diferents districtes d'aquesta. El resultat de seguir aquest tipus de segmentació amb l'exemple que s'acaba de plantejar seria tants *fogs*, o unitats intel·ligents amb capacitat de decisió, com districtes tinga la ciutat.

Figura 4.1: Districtes de la ciutat de València.

Utilitzant una segmentació basada en la geografia i prenent en consideració la figura 4.1 on es poden veure els diferents districtes de la ciutat de València, la solució que es dissenyaria constaria de tants fogs com districtes s'observen a la figura. En aquest cas, un total de 19 *fogs* serien els que formarien part del sistema. Òbviament, i amb la finalitat de proporcionar una solució el més òptima possible, pot donar-se el cas de que alguns dels districtes queden fora del disseny de la solució, bé siga perquè no estan alineats amb els objectius, o perquè la seua consideració no aporta res massa significatiu a la solució.

A més de la partició basada en els districtes de la ciutat, existeixen moltes més aproximacions que s'englobarien també dins de la segmentació basada en la geografia. Exemples són la partició basada en la posició respecte a l'antic llit del Túria o basada en els punts cardinals. En el primer dels exemples s'obtindrien dos *fogs*, un per a la part de la ciutat que queda al Nord de l'antic llit del Túria i l'altre per a la part que queda al Sud. I en el segon exemple, el resultat consistiria en 4 *fogs*, un per cada punt cardinal, i que contindrien les parts de la ciutat que es troben al Nord, al Sud, a l'Est i a l'Oest d'un determinat punt central, com per exemple, l'Ajuntament de València.

Cal remarcar que no existeixen particions que puguen ser considerades millors o pitjors considerant únicament aquest factor. Com ja s'ha comentat al començament de l'apartat, la millor segmentació és la que ofereix una solució més òptima per a resoldre alguna problemàtica. Per aquest mateix motiu, projectes diferents que afectaren a una mateixa ciutat podrien considerar una segmentació diferent obtenint així un repartiment dels *fogs* també diferent, però que en cadascun d'ells proporcione d'alguna manera una solució òptima.

4.1.2. Segmentació basada en la funcionalitat

En aquest cas, en lloc de calcar un particionat de la ciutat ja definit, es planteja la possibilitat de dissenyar-ne un pensant en la funcionalitat final del sistema. És a dir, tenint en compte des d'un primer moment quin va a ser l'efecte que s'espera obtenir del funcionament del sistema en determinades zones, es dissenya un particionat de la ciutat que permeta optimitzar el funcionament del sistema en eixes zones.

Un exemple que pot ajudar a clarificar un poc aquesta segmentació és el següent. Si es consideren els principals accessos a la ciutat de València o avingudes i carrers crítics per la gran quantitat de trànsit que flueix per ells, es podrien particionar de tal manera que s'obtingueren segments com la Ronda Nord, pista de Silla o Avinguda d'Aragó + Avinguda Blasco Ibáñez. Utilitzant aquesta segmentació es podria influir sobre el trànsit en zones crítiques concretes.

Així doncs, i en contraposició als exemples comentats a la segmentació basada en la geografia, el *fog* deixaria de tenir sentit com a partició del mapa de la ciutat i seria més interessant la seua existència com a conjunt de variables que ofereixen una funcionalitat concreta a la ciutat, com per exemple, la gestió dels accessos a aquesta o a les seues zones crítiques de trànsit. En aquest cas, el sistema estaria composat per tants *fogs* com zones crítiques de trànsit es desitjara controlar a la ciutat.

Per tal d'il·lustrar que la segmentació basada en la funcionalitat pot abarcar moltíssims tipus de funcionalitats que no ténen a veure amb l'objecte d'estudi d'aquest projecte, que és la gestió intel·ligent del trànsit, es presenta també el següent exemple. En aquest cas, la funcionalitat que es pretendria optimitzar és la gestió del reg de diferents zones de València.

La partició que es proposaria en aquest exemple prendria en consideració la quantitat d'aigua que necessiten les diferents varietats de plantes sembrades en diferents zones de la ciutat. D'aquesta manera, es podria diferenciar entre zones d'alta necessitat hídrica, zones de mitjana necessitat hídrica i zones de baixa necessitat hídrica. En aquest cas, en el particionat de la ciutat s'obtindrien 3 *fogs* que permetrien optimitzar el reg de les zones amb els 3 tipus de necessitats hídriques definides.

4.2 Estratègies de comunicació

En aquest apartat es tracten les diferents maneres d'establir comunicacions entre els dispositius que formen part de la unitat intel·ligent amb capacitat de decisió, o *fog*. Tenint en compte quants dispositius intervenen en la comunicació i quina és la finalitat d'aquesta, es poden identificar, independentment de com s'implementen, dos tipus genèrics de comunicació.

Aquests dos tipus de comunicació que es diferèncien són, per una banda la comunicació de molts dispositius a un (concentració) o d'un dispositiu a molts (difusió), i per altra banda, la comunicació punt a punt, que s'estableix entre un únic emisor i un únic receptor. Com ja s'ha comentat, aquest esquema de comunicació és independent de la seua implementació i, per tal de que quede més clar, existeix la possibilitat d'establir comunicacions dels dos tipus utilitzant la mateixa tecnologia.

Tenint en compte els dos tipus de comunicació que s'han identificat es plantegen dues estratègies per a l'establiment de comunicacions. En el cas de les comunicacions punt a punt s'està fent referència a l'establiment d'una comunicació directa o síncrona. I si es parla de comunicació molts a un o un a molts, es fa referència a una estratègia de comunicació indirecta o assíncrona.

4.2.1. Comunicació síncrona o directa

Un esquema de comunicació síncrona es caracteritza perquè sempre hi ha un receptor esperant una connexió (o una petició de comunicació) per tal de que la comunicació puga establir-se entre emissor i receptor. Açò obliga a tenir ben definit l'ordre d'aparició dels interventors, de forma que sempre hi haja un receptor en el moment en que apareix un emissor.

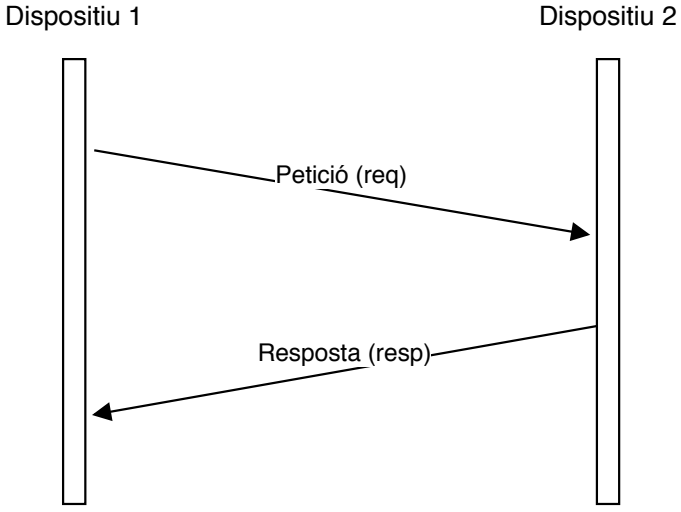

Figura 4.2: Esquema d'interacció entre dos dispositius utilitzant comunicació síncrona.

En alguns sistemes, açò no suposa ningun problema per la manera en la que està dissenyat el propi sistema. No obstant, en aquest projecte es desenvolupa un sistema escalable i adaptable, i el fet d'haver de fixar ordres d'aparició de com-

ponents limita en certa mesura aquesta adaptabilitat i escalabilitat. No obstant, aquest esquema de comunicació segueix sent atractiu per tal de satisfer alguns aspectes del sistema.

Exemples de comunicacions síncrones en són les que s'estableixen entre navegadors web i servidors web. En aquest cas s'estableix una comunicació punt a punt directa entre el client web i el servidor. Tal i com s'ha comentat al paràgraf anterior, l'ordre d'aparició dels interventors és fixe, de tal manera que, per tal de que el navegador web puga establir una comunicació, deu haver aparegut prèviament un servidor web, que fa funció de receptor, i que està esperant peticions HTTP dels emissors amb la finalitat de servir-les. Un altre exemple de comunicacions síncrones, molt relacionat amb l'exemple anterior, és el propi protocol TCP.

4.2.2. Comunicació assíncrona o indirecta

En els esquemes de comunicació assíncrona, i en contraposició als de comunicació síncrona explicada a l'apartat anterior, en aquest cas no és necessari que existeixen receptors per tal de que un emissor puga enviar informació. És a dir, no existeix la necessitat de fixar l'ordre d'aparició dels diferents interventors de la comunicació.

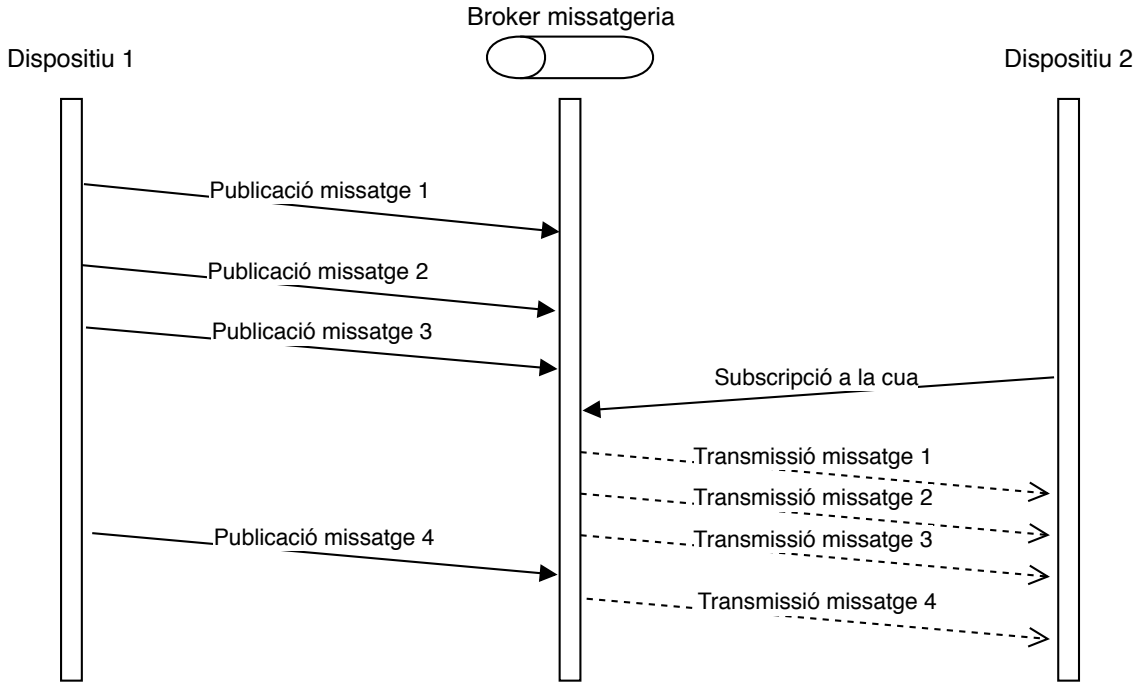

Figura 4.3: Esquema d'interacció entre dos dispositius utilitzant comunicació assíncrona.

Per exemple, podria existir un sistema en el que hi ha molts emissors funcionant i ningun receptor disponible. El funcionament d'aquests emissors no es veu compromés de ningúna manera, l'únic detall a tenir en compte és que ningú està processant ni rebent la informació que els emissors generen. No obstant, si en un moment determinat apareix un receptor en la topologia, aquest es posarà a processar les dades que començarà a rebre sense interferir tampoc en ningun cas amb els emissors.

Aquesta característica de la comunicació assíncrona permet als sistemes que la implementen dispondre d'una major capacitat d'adaptació. Com es pot veure a l'exemple explicat, la no disponibilitat de receptors o d'emissors no afecta al sistema. En el moment que apareixen, aquests s'adapten i s'integren al sistema a partir de l'estat d'aquest en el moment de la seua incorporació.

Exemples de sistemes que utilitzen comunicació assíncrona en són el correu electrònic, els grups de notícies (o *news feed* en anglés), o els fòrums. En aquests sistemes, l'emissor emet el seu missatge i quan apareix un receptor aquest ja s'encarrega de processar la informació que s'havia emés. Entre el moment de l'emissió de la informació i el processament d'aquesta pot transcórrer un lapse de temps que varia entre el processament quasi inmediat o fins a dies o mesos després de l'emissió de la informació.

4.3 Organització de les comunicacions

A l'igual que en el cas de l'organització general del sistema, resulta interessant l'organització també de les comunicacions que es realitzen dins del sistema. En aquest apartat es plantegen diferents maneres de repartir i classificar les diferents comunicacions que s'estableixen entre els dispositius del sistema. La primera classificació consisteix en agrupar les comunicacions per tipus de dispositiu, la segona, en segmentar en capes aquestes comunicacions, prenint com a criteri de classificació la funcionalitat d'aquestes comunicacions, i la tercera en classificar les comunicacions segons l'abast d'aquestes.

No només existeixen les tres aproximacions que es presenten en aquest apartat, sinó que se'n podrien plantejar moltes més. No obstant, de la mateixa manera que amb l'organització general del sistema, no existeix una segmentació correcta com a tal, sinó que, en funció de la finalitat del sistema i de com s'estructure aquest, s'escollirà l'aproximació que proporcione una solució més òptima al problema que es preten resoldre.

4.3.1. Segmentació per tipus de dispositiu

Tal i com indica el nom d'aquesta secció, en aquesta aproximació se separen i classifiquen les comunicacions en funció dels dispositius que intervenen en aquesta. D'aquesta manera s'aconsegueix modelar i acotar perfectament el tipus de relacions que s'estableixen entre els dispositius.

Per tal d'exemplificar un poc com es realitzaria la classificació de les comunicacions utilitzant aquest criteri es planteja el següent exemple. Considere's un sistema format per tres tipus de dispositius, com és el cas d'estudi objecte d'aquest projecte. En aquest escenari, els sensors es comuniquen amb nodes *gateway*, aquests es comuniquen amb actuadors i els actuadors es comuniquen entre ells també.

En l'exemple exposat, i seguint el criteri que es pretén exemplificar, es diferencien tres tipus de comunicació. El primer tipus fa referència a la comunicació (bidireccional) sensors-*gateway*, el segon a la comunicació (bidireccional també)

gateway-actuadors i l'últim a la comunicació entre actuadors. No existeix cap altre tipus de comunicació possible ja que no existeix ninguna altra relació de comunicació entre dispositius.

La segmentació de les comunicacions basant-se en el tipus de dispositiu que les estableix té sentit si es pretén tenir molt ben definides les relacions entre els diferents dispositius que formen part del sistema. En cas d'utilitzar-se, existirien tants segments de comunicació com tipus de relacions s'estableixen entre els dispositius.

4.3.2. Segmentació per funcionalitat

Una vegada exposada la possibilitat de segmentar les comunicacions basant-se en el tipus de dispositius que formen part d'aquesta, es planteja la possibilitat d'anar més enllà i no només classificar pel tipus de relació que s'estableix entre els dispositius, sinó per la funcionalitat de la pròpia comunicació. És a dir, no només es basa la classificació en una anàlisi superficial de la relació que s'estableix amb la comunicació, sinó que, a partir d'una anàlisi més profunda, es classifica la comunicació entenent la funció que té aquesta dins del sistema.

Amb la intenció de clarificar el concepte, es considera el següent escenari. En un sistema format per tres tipus de dispositiu es troben sensors, nodes centrals i *loggers*. Els sensors produeixen dades i les envien al node central del *fog* i els *loggers* només s'encarreguen de llegir aquestes dades del node central i escriure-les en un fitxer de text. A més, pot existir un conjunt de dispositius o sistemes que s'encarreguen de la gestió del sistema.

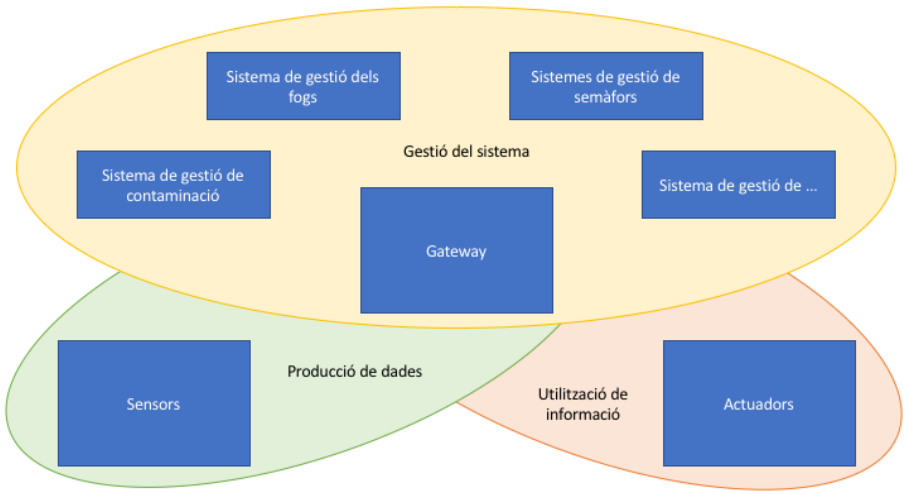

Figura 4.4: Esquema de la segmentació per funcionalitat utilitzada a l'exemple.

A l'exemple exposat, l'esquema del qual es mostra a la figura 4.5, s'identifiquen tres tipus de funció de la comunicació. Aquestes dues funcions són l'aportació de dades noves, la consumició de dades i la gestió del sistema. En aquest cas, i independentment del tipus de dispositius que intervenen en la comunicació, seguint l'aproximació de la segmentació per funcionalitat, s'establirien tres categories.

Aquest tipus de segmentació té sentit en el cas que es desitge definir molt clarament quin és el motiu de l'existència de les comunicacions. Es poden identificar moltes funcions de les comunicacions. Per aquest motiu, l'òptim particionat i classificació d'aquestes ve molt lligat a l'estructuració general i als objectius del sistema que es dissenya.

4.3.3. Segmentació per abast

Aquest últim tipus de segmentació que es planteja se centra en classificar les comunicacions en funció de l'abast (no geogràfic, sinó organitzatiu) d'aquestes. És a dir, dins de l'organització general del sistema, fins a quin punt han de ser accessibles les comunicacions? Quins dispositius poden comunicar-se amb quins? La resposta a aquestes preguntes és el que resol aquest tipus de segmentació.

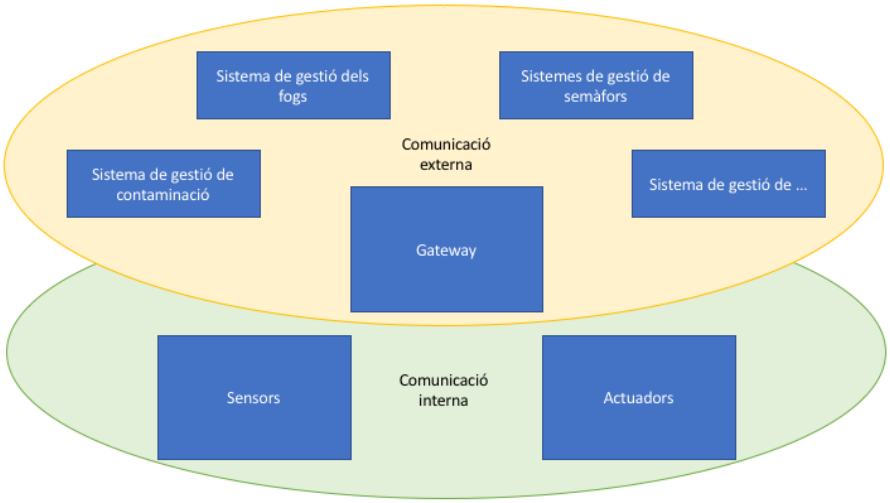

Figura 4.5: Esquema de la segmentació per abast utilitzada a l'exemple.

Com a exemple, es proposa prendre en consideració un sistema organitzat en *fogs*. Tenint en compte només aquest aspecte, existeixen dos tipus de comunicacions. Les comunicacions internes del *fog* i les que són externes al *fog*. D'aquesta manera, s'establirien dues classificacions per a les comunicacions: les comunicacions *intrafog* i les comunicacions *extrafog*.

Aquest tipus de segmentació resulta interessant quan es vol limitar i acotar perfectament l'abast de les comunicacions. Quan es pretén establir quins dispositius poden accedir a quines dades, aquesta aproximació és la que millor s'adapta. A més de les consideracions en l'aspecte organitzatiu del sistema, aquesta manera de classificar les comunicacions pot ser útil també per tal d'augmentar la seguretat del sistema.

4.4 Organització de les dades

En aquest apartat es parla sobre els diferents tipus de dades i missatges que necessita gestionar el sistema objecte d'estudi d'aquest projecte. No només s'aborda açò en termes generals sinó que, en cada tipus de dades que es diferencia, es

planteja el contingut d'aquests. A més, es comenten altres aspectes que han de ser considerats en el disseny de les dades, com per exemple l'escalabilitat o el xifrat d'aquestes.

4.4.1. Tipus de dades

Aquesta secció presenta els diferents tipus de dades que deuen existir en el context del sistema objecte d'estudi de l'actual projecte. Aquest conjunt de dades no és tancat i és possible que en un escenari futur siga necessari afegir-ne de noves. Per aquest motiu, un correcte disseny d'aquestes és imprescindible per a assegurar l'adaptabilitat del sistema als nous tipus de dades que poden ser necessaris en un futur. En aquest projecte, es proposen 5 tipus generals de dades.

- **Comandament**: dades, la funció de les quals és ordenar canvis o modificacions del funcionament als dispositius als que van destinades.
- ***Handshake***: dades destinades a l'establiment de claus i identificació/autenticació de dispositius.
- **Mesura**: dades utilitzades per a aportar informació generada per sensors a nodes centrals del *fog*.
- **Notificació**: dades que ténen l'única funció de notificar a altres dispositius aspectes, com per exemple quin tipus de comandament estan executant o quina política estan aplicant.
- **Política**: dades que contenen informació sobre polítiques aplicables. Contenen criteris de funcionament i ordres directes.

4.4.2. Camps de les dades

Per tal de reduir la càrrega dels sistemes que processen les dades i l'ample de banda que es consumeix per tal de comunicar-les, és important dissenyar de manera correcta les dades. Aquest disseny passa per assegurar que les dades contenen tota la informació necessària per a la seua correcta gestió i no més de la que és necessaria. Per aquests motius, en aquest apartat es presenten els camps necessaris de cada tipus de dades.

Comandament

- **Destinatari**: determina si el dispositiu receptor ha d'executar o no el comandament.
- **Tipus de comandament**: identifica el tipus de comandament.
- **Acció ordenada**: acció que ha d'executar el dispositiu receptor si és destinatari.
- **Marca temporal**: marca de la generació de les dades.

Handshake

- **Fase**: fase del protocol en la que es generen les dades.
- **Identificador del dispositiu**: identificació de l'emissor.
- **Tipus de dispositiu**: tipus de dispositiu emissor.
- **Dades relacionades amb la clau**: dades necessaries per a derivar i establir la clau.
- **Marca temporal**: marca de la generació de les dades.

Mesura

- **Dades del sensor**: dades del sensor que genera la mesura.
- **Tipus de mesura**: classificació de la mesura.
- **Valor de la mesura**: valor numèric de la mesura registrada.
- **Unitat i element**: unitat de mesura i quin element s'ha mesurat.
- **Marca temporal**: marca de la generació de les dades.

Notificació

- **Dades del dispositiu**: dades de l'emissor de les dades.
- **Dades a notificar**: dades que es notifiquen per part de l'emissor.
- **Marca temporal**: marca de la generació de les dades.

Política

- **Tipus de política**: classificació de la política.
- **Identificació de la política**: identificació concreta de la política.
- **Dades relacionades amb la política**: criteris de funcionament que es volen establir.
- **Duració de l'aplicabilitat de la política**: quant de temps és aplicable la política.
- **Marca temporal**: marca de la generació de les dades.

4.5 Establiment de claus i protocols de xifrat

Amb la finalitat d'afegir una capa de seguretat al sistema, ja s'havia introduït la necessitat de generar claus per al xifrat i autenticat de les dades. S'havia explicat també la necessitat de dissenyar un protocol de *handshake* que fóra l'encarregat d'identificar els dispositius i de l'assignació de claus que després serien utilitzades per al xifrat dels missatges.

Prenent en consideració únicament les claus, es plantegen dues maneres de procedir. Per una banda, es pot triar la utilització de criptografia de clau simètrica (o compartida), i per l'altra, criptografia de clau assimètrica (o pública). Però a més d'aquesta problemàtica, és interessant qüestionar-se també com s'aconsegueix la clau final que s'utilitzarà per a xifrar. Es pot pensar en una assignació estàtica de la clau per a cada dispositiu, o que d'alguna manera aquests siguen capaços de derivar una clau quan són posats en funcionament. I, per últim, es pot pensar també en quin valor és el que aporta la posessió d'una clau. Les dues principals possibilitats són l'autenticació únicament del dispositiu o l'autenticació del dispositiu dins d'un concret i determinat context.

4.5.1. Criptografia de clau simètrica vs. clau assimètrica

La criptografia de clau simètrica, o de clau compartida, consisteix en el xifrat i desxifrat de dades utilitzant únicament una clau. Aquesta clau, que com el seu nom indica, és compartida tant per l'emisor (qui xifra) com pel receptor (qui desxifra). La clau simètrica és utilitzada per a xifrar, però també per a desxifrar les dades.

La criptografia de clau assimètrica, o de clau publica, consisteix en el xifrat i desxifrat de dades utilitzant una clau formada per dues components. Aquestes components són denominades clau pública i clau privada. La clau pública és la utilitzada per al xifrat de les dades, i la privada per al desxifrat. Aquestes dues components estan profundament relacionades en la seua formació i només ténen sentit utilitzades conjuntament. Les dades que es xifren utilitzant una deteminada component pública només pot ser desxifrat utilitzant la seua component privada.

La utilització de la criptografia de clau simètrica ofereix un ràpid desxifrat de les dades [13]. No obstant, és necessari mantindre una clau compartida per cadascun dels dispositius que necessiten establir comunicacions entre ells. D'aquesta manera, i suposant un escenari on existeixen un nombre n de sensors i un node central, aquest hauria d'emmagatzemar n-1 claus compartides per tal de poder atendre a tots els sensors, i aquests només necessitarien emmagatzemar-ne una. A més d'aquesta problemàtica, el fet d'haver de compartir la clau és una debilitat del sistema, ja que açò s'ha de fer abans de començar amb el xifrat de la comunicació.

Per contra, amb la utilització de la criptografia de clau assimètrica, no és ningun problema compartir de manera oberta la component pública de la clau, i a més, només s'han d'emmagatzemar les claus públiques dels dispositius amb els que s'ha d'establir comunicació. Aprofitant l'escenari suposat al paràgraf anteri-

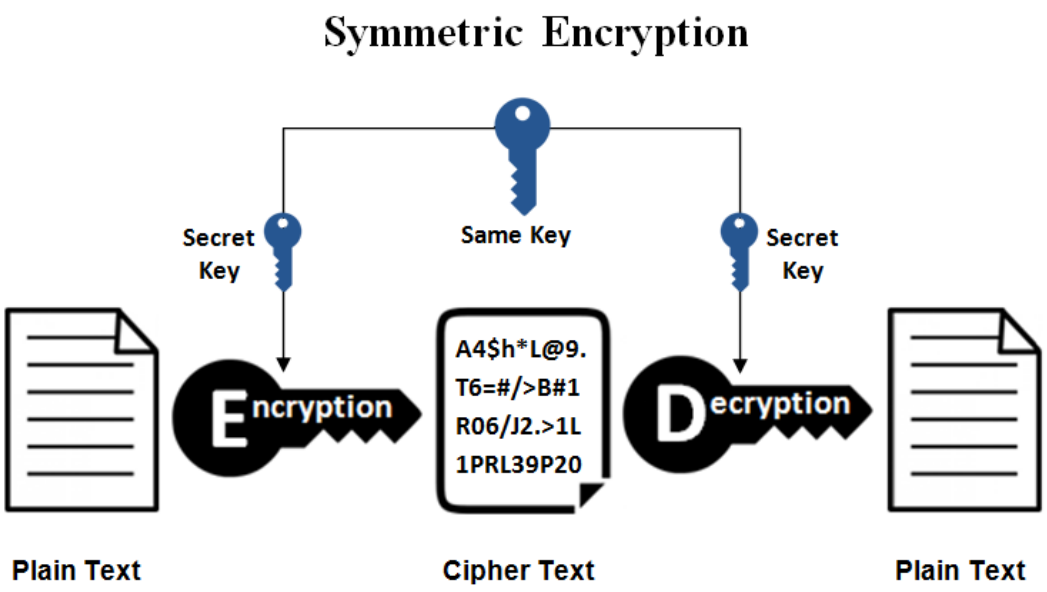

Figura 4.6: Esquema de xifrat i desxifrat amb criptografia de clau simètrica.

or, en aquest cas el node central només hauria de recordar la seua clau privada i no hauria d'emmagatzemar ninguna clau pública, ja que ell només estaria desxifrant. Els sensors, en canvi, només haurien d'emmagatzemar una única clau pública, que seria la del node central. La part negativa de la criptografia de clau assimètrica és que el procés de desxifrat és molt més costós que amb la utilització de criptografia de clau simètrica.

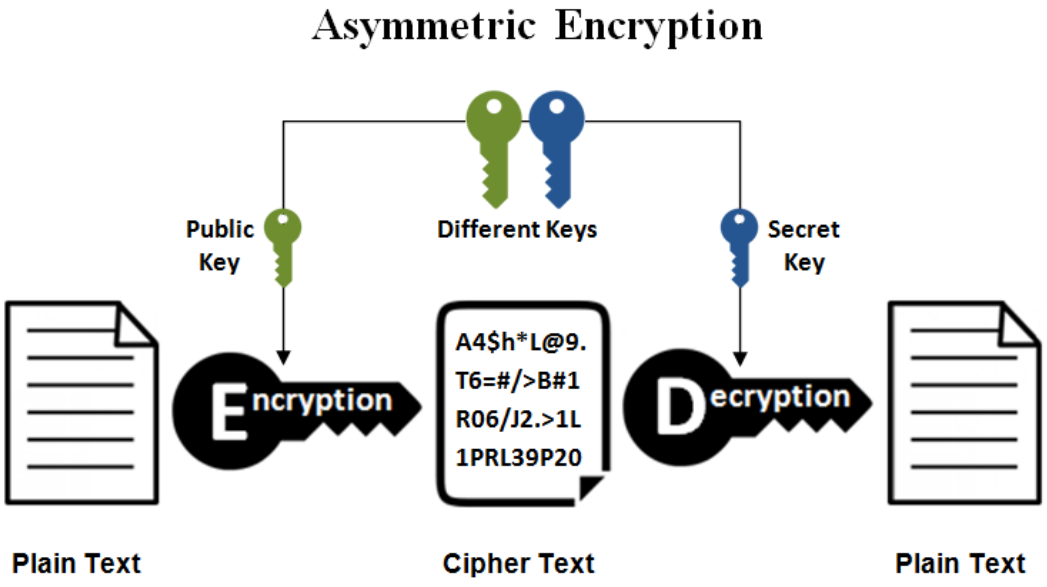

Figura 4.7: Esquema de xifrat i desxifrat amb criptografia de clau assimètrica o pública.

Existeixen molts algoritmes que utilitzen la criptografia de clau simètrica i clau assimètrica. Alguns exemples en són, en el cas de la clau simètrica, l'algoritme DES (*Data Encryption Standard*) o l'algoritme AES (*Advanced Encryption Standard*) [14]. I en el cas de la criptografia de clau assimètrica, el més conegut és RSA (*Rivest–Shamir–Adleman*) [15].

4.5.2. Assignació estàtica de clau vs. derivació a partir de *token*

En el cas de la utilització de la criptografia de clau simètrica és important la manera de compartir la clau entre els dos dispositius que van a establir la comunicació. Tal i com s'ha comentat a la secció anterior, en aquest punt resideix una gran debilitat del sistema. És per això que es plantegen diferents maneres d'abordar aquest problema intentant minimitzar la debilitat que suposa.

La primera manera de fer-ho pot ser establint directament, i de forma manual, la clau en els dos dipositius que van a intervindre en la comunicació. Encara que, pot ser, aquesta sigua la manera més segura de fer-ho, ja que només algú autoritzat a la programació o desplegament dels dispositius és qui coneix i pot indicar la clau als dispositius, suposa un gran impediment de cara a l'escalabilitat del sistema, ja que s'han de modificar tots els dispositius amb els quals el nou dispositiu va a establir comunicacions. Suposant un dispositiu x que haurà d'establir comunicacions amb un conjunt d'n dispositius, s'hauria d'indicar al dispositiu x un conjunt d'n claus, una per a cadascun dels dispositius amb els que ha d'establir comunicació, i, a més, a cadascun d'eixos dispositius se li hauria d'afegir la clau que comparteix amb el dispositiu x.

Amb l'exemple que s'acaba de presentar, és evident que l'assignació estàtica de claus suposa una greu penalització per a l'escalabilitat del sistema. És per eixe motiu que es pot pensar en permetre que els dispositius siguen capaços de derivar les claus per ells mateixos a partir d'un *token*, del qual es parla més endavant. D'aquesta manera, utilitzant el mateix *token* per a tots els dispositius, aquests podrien ser capaços de derivar claus entre ells. Encara que tots parteixen del mateix *token*, en cada negociació s'obté com a resultat una clau diferent. Gràcies a açò, s'estableix una clau diferent per a cada parella de dispositius. Al derivar totes les claus a partir del mateix *token*, si no es disposa d'aquest, resulta impossible la derivació d'una clau funcional. Aquest procés és possible gràcies a la utilització d'un protocol de *handshake*, que resol la derivació i compartició de la clau final de manera segura.

4.5.3. Valor aportat pel secret compartit o pel *token*

Una vegada un dispositiu disposa d'un *token* o d'una clau compartida, és interessant considerar quines implicacions això suposa. Tal i com s'ha introduït abans, les dues principals implicacions serien la d'únicament autenticació dels dispositius, o la d'autenticació dels dispositius i vinculació d'aquests a un context o escenari concret.

En el cas de només autenticar els dispositius, un únic *token* seria necessari per a la utilització en el sistema sencer. Aquest permetria al dispositiu comunicar-se amb qualsevol altre dispositiu, fora quina fora la seua funció i estiguera situat on estiguera situat dins de l'esquema del sistema. Encara que suposa una reducció significativa de la quantitat de *tokens* que s'han de mantindre i gestionar, és possible que es desitge acotar més l'abast del dispositiu que disposa del *token*.

Per aquest motiu, es planteja la possibilitat de que la utilització d'un determinat *token*, a més d'identificar i autenticar al dispositiu dins del sistema, el vincule a una determinada part d'aquest. Per tal d'exemplificar-ho, pot ser interessant

que la possessió d'un determinat *token* només permeta l'accés del dispositiu a un determinat *fog*. Més restrictiu encara, pot ser seria desitjable també no només acotar el *fog*, sinó també la capa de comunicació a la que el dispositiu té accés. El resultat d'açò seria que, amb la utilització d'un determinat *token*, el dispositiu seria identificat, autenticat i vinculat a un *fog* i una capa de comunicació concreta.

4.6 Model conceptual

En aquest apartat es presenta ja un senzill model conceptual del sistema. Es presenten els principals components i es planteja la utilització de altres components complementaris per tal d'afegir o completar funcionalitats dels components principals. A més, es planteja la possibilitat de reestructurar l'arquitectura a mesura que escala el sistema i per últim es presenta un mapa conceptual que modela el sistema.

4.6.1. Components principals

El sistema està format per 3 components principals. Aquests components són els sensors, actuadors i nodes centrals del *fog*, també anomenats *gateways*. Cadascún d'aquests components té una funció diferenciada dins del sistema. A continuació s'explica la funció dels diferents components principals.

Sensor

Els sensors són els encarregats de generar dades rellevants per al sistema. Existeixen sensors per al mesurament de moltes i molt variades variables. Exemples en serien el sensors encarregats de mesurar els nivells de CO_2, o sensors encarregats de la monitorització de la humitat relativa a la ciutat.

Aquests sensors, a més de publicar periòdicament les dades que generen, podrien ser també consultats, mitjançant interficie web o API REST, per exemple, per tal de poder obtenir informació sota demanda. Com ja s'ha comentat, l'única funció que ténen aquests dispositius és la generació d'informació. La manera de ser consultada aquesta depén només del propòsit i estructuració del sistema.

Node central o *gateway*

Els dispositius que fan funció de node central i actuen com a representant del *fog* són els encarregats de rebre dades dels sensors, processar-les i decidir actuacions. Aquests dispositius són els més importants dins de l'àmbit del *fog* ja que són els que disposen del que podriem considerar intel·ligència i decideixen les actuacions que cal dur a terme en temps real.

Aquests dispositius, a més de prendre les seues pròpies decisions a partir de l'anàlisi de la informació que reben dels sensors, també són capaços d'acceptar ordres de dispositius que estan a un nivell superior dins de la jerarquia del sistema. Aquests dispositius que se situen a un nivell superior poden solicitar a la

gateway l'aplicació de determinades polítiques, i aquesta, des del coneixement del seu entorn gràcies a la informació proporcionada pels sensors i dels dispositius que formen part del *fog*, és qui decideix les mesures a aplicar per tal de donar compliment a les polítiques.

Altra part important de les funcions dels nodes centrals és l'anàlisi de la informació que rep dels sensors. Aquest procés d'anàlisi pot ser adaptat, ajustat o modificat per les polítiques que s'apliquen per part dels dispositius de nivell superior a la jerarquia o pel coneixement que adquireix la propia *gateway* a partir del seu funcionament. Una vegada realitzat l'anàlisi i preses les decisions pertinents, la *gateway* notifica als dispositius actuadors adequats les accions que han de prendre.

Actuador

Tal i com el seu nom indica, un dispositiu actuador és l'encarregat d'actuar per tal de materialitzar les decisions que pren el node central del *fog*. Existeixen molts i, com en el cas dels sensors, molt diferents tipus d'actuador. Exemples en són des d'un semàfor intel·ligent fins a un sistema de gestió del reg de la gespa d'un parc.

En funció del tipus d'actuador en qüestió, les accions concretes que pot dur a terme són molt divergents. Des de les més senzilles, com podrien ser les accions d'apagar o deixar en standby l'actuador, fins a altres més complexes, com per exemple l'augment de 20 segons en la duració del verd en tots els semàfors de determinat carrer. La idea bàsica que es vol transmetre és que els actuadors són els que transformen al món real les decisions que pren el node central del *fog*.

4.6.2. Components complementaris

Els components complementaris del sistema són tots aquells que no són sensors, actuadors o nodes centrals, però que afegeixen o complementen d'alguna manera les funcionalitats que ofereixen o duen a terme els components principals del sistema. Existeixen moltes possibilitats i es poden crear components complementaris per a solucionar problemes o incrementar les funcionalitats del sistema en general. A continuació s'expliquen els més rellevants d'aquests components.

Analitzadors de mesures

Tal i com es pot suposar amb el nom del component, l'analitzador de mesures és l'encarregat de realitzar una anàlisi de les mesures produïdes pels sensors. Com és d'esperar, aquest component està associat directament a les funcions del component principal *gateway*. La funció d'un analitzador de mesures aïllat no té sentit al sistema. És per això que és considerat un component complementari perquè afegeix la funció d'analitzar mesures als components *gateway*.

Existeixen tants analitzadors com tipus de mesures es vulguen analitzar. Per exemple, si es volen gestionar mesures de CO_2, mesures d'humitat i mesures de micropartícules, va a ser necessari disposar de 3 analitzadors diferens. Cadascun dels analitzadors està especialitzat únicament en l'anàlisi d'un tipus de mesura i

trasllada les seues conclusions a la *gateway*. La *gateway* és qui s'encarrega de, prenent en consideració les conclusions de tots els analitzadors dels quals disposa, decidir l'actuació a dur a terme.

Gerents de dades

Amb un nom també bastant explícit, els components gerents de dades són els encarregats de gestionar la creació i interpretació de dades en un determinat format. Al contrari que en el cas dels analitzadors de mesures, els gerents de dades no estan associats únicament a un tipus de component principal. En aquest cas, els components gerents de dades poden ser associats a qualsevol tipus de component principal.

Existeixen tants gerents de dades com formats de dades es vulguen gestionar al sistema. Prenint en consideració els formats que s'han comentat anteriorment, JSON i XML, per tal de que el sistema siga capaç de gestionar ambdós tipus de dades és necessari la creació de dos gerents de dades diferents, un per a gestionar dades en format JSON i l'altre per a dades en format XML. Aquests components són els encarregats d'interpretar dades i fer-les entendibles per al dispositiu al que estan associats.

Gerents de comunicacions

Continuant amb l'explicitat dels noms, els components gerents de comunicacions són els encarregats de proporcionar capacitats de comunicació. És a dir, aquests components s'encarreguen únicament de rebre o enviar dades. Per la pròpia funció del component, és evident que la seua existència no té sentit si no és associada d'algun component principal. Aquest tipus de components poden associar-se amb qualsevol dels components principals, proporcionant-los a aquests les capacitats necessaries per a poder establir comunicacions.

A l'igual que passa amb els gerents de dades i els components analitzadors, en el cas dels gerents de comunicacions també n'existeixen tants com tipus de comunicació es desitge implementar al sistema. Tractant els exemples que s'han comentat anteriorment, si el sistema requereix la comunicació mitjançant cues de missatgeria i també APIs REST, aleshores necessitarà l'existència de dos tipus diferents de component gerent de comunicació, un per al cas de les cues de missatgeria i l'altre per al cas de les APIs REST.

Gerents d'autenticació

Una vegada més, el nom del component deixa poc a la imaginació. Els gerents d'autenticació són els components encarregats de la gestió dels aspectes relacionats amb la seguretat de les comunicacions. Aquests, a l'igual que amb els altres components complementaris, també ténen sentit únicament vinculats a un component principal. Poden associar-se amb qualsevol dels tres tipus de component principal.

Les principals funcions d'aquests components són la derivació de claus. És a dir, els gerents de comunicacions duen a terme el protocol de *handshake*. Però

a més d'aquesta funció, també són els responsables del xifrat i desxifrat de les dades i mantenen una base de dades de dispositius autenticats amb el dispositiu al qual estan vinculats.

Existeixen tants gerents d'autenticació com protocols de *handshake* i xifrat/-desxifrat suporte el sistema. D'aquesta manera i prenent els protocols de xifrat/-desxifrat comentats anteriorment, existiria un gestor d'autenticació que implementaria un protocol de *handshake* i el xifrat/desxifrat RSA i altre gestor d'autenticació que implementaria un protocol de *handshake* diferent i el xifrat/desxifrat AES.

4.6.3. Reestructuració de l'esquema

Com ja s'ha tractat anteriorment, el sistema s'estructura en *fogs* que agrupen conjunts de dispositius amb la finalitat de proporcionar un determinat servei acotat geogràficament o per funcionalitat. Però aquest esquema no ha de ser necessàriament fixe i pot ser interessant modificar-lo dinàmicament segons les necessitats del sistema. Ja s'ha parlat de l'addició o eliminació de dispositius dinàmicament, però no de la reestructuració de *fogs*.

Aquesta reestructuració dels *fogs* també és possible. El concepte més adient per a aquesta reestructuració seria la consideració de "*fogs* de *fogs*". És a dir, un *fog* que està format per un conjunt de *fogs*.

Aquest nou concepte pot concebre's com a una estructuració per capes de *fogs* del sistema. Cada capa afegeix un extra d'abstracció i, a més, distribueix més encara la intel·ligència del sistema. D'aquesta manera, es segmenta de manera més concisa el sistema, la qual cosa aporta millores en la seguretat i l'escalabilitat. Aquesta nova forma d'estructuració permet la creació d'esquemes més complexes del sistema final, permetent actuacions a més alt nivell i de manera més abstracta.

4.6.4. Diagrama d'anàlisi

En aquest apartat es presenta un diagrama d'anàlisi on es representen els diferents components explicats a l'apartat anterior. Aquest diagrama pretén modelar de forma general una representació del sistema final. Es poden observar les associacions de components complementaris als components principals i la relació que existeix entre els diferents components principals.

A la figura 4.8 s'observen en color taronja els components principals del sistema i en color blau els components complementaris. Es pot veure també la relació que existeix entre els components principals i els complementaris, entenent, per exemple, que el node central del *fog*, la *gateway* incorpora un gestió de comunicacions per cada capa de comunicacions a la que estableix comunicacions. A més, aquesta també incorpora un nombre n d'analitzadors de mesures, un gerent de dades i un altre d'autenticació.

El cas dels sensors i actuadors és més senzill ja que aquests necessiten menys funcionalitats. Com es pot veure, aquests només necessiten un gerent de comunicacions perquè només estableixen comunicacions a una capa. I, a l'igual que la

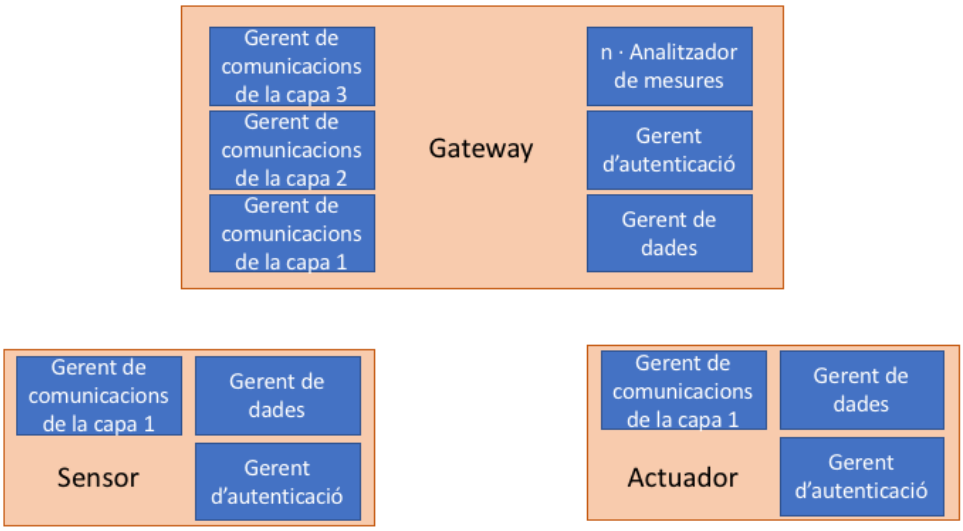

Figura 4.8: Diagrama d'anàlisi amb el modelat d'un *fog*.

gateway, també requereixen la funcionalitat que ofereixen el gerend de dades i el gerent d'autenticació.

Una característica digna de comentar en aquest punt és que tampoc existeix la necessitat de que tots els dispositius que estableixen comunicació entre ells utilitzen el mateix tipus de gerent de dades. Encara que no serien capaços d'interpretar directamet les dades que l'altre interventor de la comunicació envia, açò és fàcilment solventable afegint una capa d'integració entre els dos dispositius de tal manera que les dades s'adapten a un format que el receptor sí pot interpretar. Aquest detall augmenta l'escalabilitat del sistema, deslligant el tipus de gerent de dades que implementa cada dispositiu.

CAPÍTOL 5
Disseny de la solució

Comptant ja amb una anàlisi en profunditat del problema i alguns detalls de disseny, en aquest apartat es proposa un disseny concret d'una solució. L'objectiu d'aquest és la determinació de tot el necessari de cara a implementar el prototip. Per açò, es comenten aspectes relacionats amb el disseny dels diferents dispositius, també la manera en la que s'implementen les comunicacions, així com el disseny de les dades que es transmetran entre els diferents dispositius. A més, es planteja una solució basada en components i les interfícies d'aquests, definint així la implementació que es comenta al següent capítol.

5.1 Disseny dels dispositius

Tenint en compte tots els aspectes comentats al capítol anterior en el que respecta als dispositius que podrien utilitzar-se per a formar els sensors, actuadors i *gateways*, en aquest apartat es comenta quins dispositius físics se seleccionen i els motius d'aquesta selecció. Seguint amb el que ja s'havia comentat en apartats anteriors, on es diferencia entre dos tipus de dispositius segons les seues necessitats de còmput, aquest apartat també s'esctructura seguint aquest criteri, parlant en primer lloc sobre sensors i actuadors, i després sobre *gateways*.

5.1.1. Sensors i actuadors

El primer tipus de dispositiu a comentar són els sensors i actuadors. Tal i com s'ha anticipat i comentat en apartats anteriors, aquest tipus de dispositius no requereixen tanta capacitat de còmput com un node central. Encara que una placa Arduino hagués sigut suficientment potent per a suportar la funcionalitat del component, s'ha decidit utilitzar una Raspberry Pi per a materialitzar el component.

Aquesta decisió s'ha prés per diferents motius. El primer, i més important, és que una Raspberry Pi, a l'oferir una major potència de càlcul, dóna per satisfetes les necessitats de còmput que podria resoldre una Arduino també, i per tant, compleix amb els requeriments dels component. Un altre factor decisiu ha sigut el fet de la disponibilitats d'aquest tipus de dispositius. El departament disposava de gran nombre d'aquests dispositius, i per tal d'aprofitar material, s'ha decidit

utilitzar-los. I un motiu més que ha determinat aquesta decisió ha sigut el fet de que la Raspberry Pi funciona amb un sistema *Linux* i que aquest sistema ve dins d'una targeta SD.

Aquest últim detall comentat és important per diferents motius. El primer d'ells és que els sistemes *Linux* ofereixen serveis que poden arrancar juntament amb l'arranc del sistema operatiu. D'aquesta manera, es pot configurar l'arranc del component com a un servei i habilitar aquest servei per a que s'execute amb l'arranc del sistema i, d'aquesta manera, automatitzar l'arranc dels diferents components. Complementari a açò seria el segon motiu. Com ja s'ha comentat, el sistema ve integrat en una targeta SD. Açò significa que el sistema és totalment independent del dispositiu físic, permetent així el ràpid canvi de funció del dispositiu físic únicament intercanviant les targetes SD d'aquests. Aquest motiu facilita l'adaptabilitat del sistema.

5.1.2. Node central o *gateway*

Aquest segon tipus de dispositiu té unes necessitats de còmput un poc superiors. En aquest cas, a part de la major potència de càlcul necessitada pel component per tal de la realització d'anàlisis, etc., s'afegeix un requeriment extra que és que ha de suportar un servidor de missatgeria, que més avant es comenta. S'ha decidit la utilització, també, de Raspberry Pi com a dispositiu físic que dona suport al component *gateway*.

En aquest cas hi ha un motiu evident que obliga prendre aquesta decisió, i és que una placa Arduino no compleix amb els requeriments mínims que necessita el component. Una Raspberry Pi sí que ofereix suficient potència de càlcul com per a suportar la necessitat d'un component *gateway*, que ha de realitzar processos d'anàlisi de mesures, etc. Un altre dels motius que propicia aquesta decisió és un que ja s'ha comentat en el cas dels sensors i actuadors, i es tracta de la possibilitat d'automatitzar l'arranc del component utilitzant els serveis del sistema *Linux*. A més, i com s'ha comentat en el cas dels sensors i actuadors, la gran disponibilitat d'aquests dispositius també ha sigut un factor important.

A més, un factor determinant completament ha sigut la necessitat de que el node central suporte un servidor de missatgeria en el mateix dispositiu físic que el component. Una Raspberry Pi ofereix suficient potència com per a suportar tant el component *gateway* com la càrrega adicional que suposa el mantenir un servidor de missatgeria. A més, al ser un sistema *Linux*, permet una fàcil instal·lació i configuració d'aquest servidor. Per aquests motius s'ha considerat que una Raspberry Pi és la millor opció per tal de donar suport a un component *gateway*.

5.2 Disseny de les comunicacions

Prenent en consideració l'aproximació exposada al capítol 4 sobre els diferents tipus de comunicació que existeixen, així com les estratègies que es proposen per tal d'abordar aquestes comunicacions, en aquest apartat es presenten dues tecnologies que poden ser utilitzades per tal de donar suport a aquestes estratègies

comentades. Es parla, en primer lloc, sobre la tecnologia que dóna suport a la comunicació síncrona i posteriorment sobre la que suporta la comunicació assíncrona.

5.2.1. APIs REST

Una interfície REST (*Representational State Transfer*) es defineix com una interfície situada entre diferents sistemes i que utilitza el protocol HTTP per a obtenir o modificar dades. Aquestes interfícies no necessiten abstraccions addicionals com altres protocols basats en patrons d'intercanvi de missatges. Les dades que es transmeten o es reben poden estar en qualsevol format (normalment JSON, encara que també pot ser XML o qualsevol altre) [16]. Sobre les dades es parla en major profunditat en una secció posterior.

Les principals característiques de implementacions REST són:

- **Utilització d'un protocol client/servidor sense estat**: l'ús del protocol HTTP, que és un protocol client/servidor sense estat, permet que ninguna de les dues parts que intervenen en la comunicació hagen de guardar estats d'aquesta. És a dir, cada petició HTTP conté tota la informació necessaria per a resoldre-la.
- **Operacions definides**: les operacions que implementen les interfícies REST són les mateixes que implementa el protocol HTTP, és a dir: POST, GET, PUT, DELETE. Aquestes operacions s'utilitzen per a actualitzar, consultar, afegir o eliminar dades, respectivament.
- **Utilització d'una sintaxi universal**: ja que cada recurs que es pot consultar o modificar s'identifica amb una única URI (*Uniform Resource Identifier*).

Coneguent les principals característiques de les APIs REST, i principalment el fet de que estiguen dissenyades per a la utilització amb protocols client/servidor, és fàcilment deduïble que aquest tipus de comunicació s'inclou clarament en el grup de la comunicació síncrona o directa. En qualsevol dels dos sentits de la comunicació, un dels dos dispositius que intervenen fa la funció de client i l'altre de servidor.

En relació al sistema objecte d'aquest projecte, aquest tipus de comunicació és interessant per tal de consultar estats de dispositius concrets o per tal de reprogramar també un dispositiu en concret. Açò aporta una major capacitat d'adaptació al sistema, ja que permet l'adaptació de parts molt concretes d'aquest.

5.2.2. Cues de missatgeria

Una cua de missatgeria és una forma de comunicació asíncrona entre serveis (entre diferents sistemes) que s'utilitza en arquitectures que no són client/servidor i que es basa en la transmissió de missatges. Aquests missatges són emmagatzemats en cues fins que són processats. El processat d'aquests missatges consisteix en fer-los arribar als consumidors que siguen necessaris. Les cues de missatgeria, poden ser utilitzades amb la finalitat de desacoplar diferents processos dins d'un

mateix sistema, per a classificar la informació o per a balancejar la càrrega del sistema.

Existeixen molts protocols basats en cues de missatgeria, com per exemple MQTT (*Message Queuing Telemetry Transport*) o AMQP (*Advanced Message Queuing Protocol*). Tal i com s'ha comentat a l'inici de l'apartat, la comunicació basada en cues de missatgeria permet la utilització d'aquesta implementació tant en l'esquema de comunicació punt a punt com en l'esquema d'un a molts (o molts a un). En aquest cas es parlarà només de l'esquema un a molts.

Aquesta comunicació d'un a molts (o de molts a un) es pot aconseguir utilitzant un esquema de comunicació basat en la diferenciació de dos tipus d'interventors en la comunicació [17]. El primer tipus d'interventor són els publicadors, que com el seu nom indica, són qui publica informació a la cua de missatgeria. La qual cosa ens duu al segon tipus d'interventor, que són els subscriptors. Amb aquesta classificació tan descriptiva, el segon tipus d'interventor en la comunicació és qui indica a quina cua de missatgeria vol subscriure's i de la qual rebra els missatges que siguen publicats.

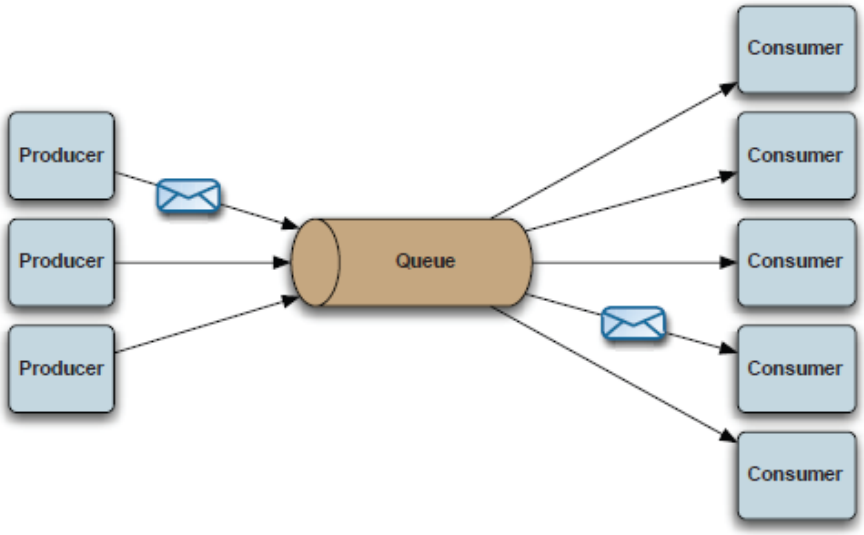

Figura 5.1: Esquema d'una cua de missatgeria basada en publicadors i subscriptors.

L'exemple que s'observa a la figura 5.1 mostra clarament el concepte del tercer tipus de comunicació que es pot considerar i que deriva de la comunicació molts a un. En aquest exemple es veu clarament que si molts productors envien les seues dades a la mateixa cua i, al mateix temps, molts subscriptors se subscriuen a aquesta, s'està establint una comunicació que definiriem de molts a molts.

Existeixen moltes maneres, per part dels subscriptors, de filtrar la informació amb la finalitat de rebre només la que resulta d'interés. La més utilitzada és la utilització de tòpics. Aquests tòpics els estableix el publicador i marca els missatges que genera amb ell. D'aquesta manera, un subscriptor, quan se subscriu a determinada cua de missatgeria, demana que se li proporcionen només els missatges marcats amb el tòpic que ell indica. El resultat és l'enrutament cap al subscriptor de només la informació que li resulta d'interés, reduint així el con-

sum d'ample de banda i de capacitat de processament (i per tant, d'energia) del dispositiu subscriptor.

En el que respecta al projecte, aquest esquema de comunicació pot resultar molt interessant de cara a l'escalabilitat del sistema, permetent la ràpida incorporació de nous sensors. Aquests nous sensors només haurien de publicar a la cua on està subscrit el dispositiu que s'encarrega de processar mesures (establint així una comunicació de molts a un). De la mateixa manera, també permet l'escalabilitat del nombre d'actuadors finals del sistema, ja que només s'harien de configurar aquests per a la subscripció a la cua corresponent.

En general, les cues de comunicació, en relació al projecte, permeten el desacoblament dels processos de producció d'informació (generació de mesures per part dels sensors), de processament de la informació (anàlisi per part de nodes centrals o *gateways*) i d'actuació final sobre l'entorn físic (accions finals dutes a terme pels actuadors). Aquest desacoblament dels diferents processos del sistema facilita la distribució de la càrrega i l'escalabilitat general del sistema, fent possible l'addició de nous processos de manera relativament senzilla.

5.3 Capes de comunicació

Una vegada establides les diferents arquitectures de comunicació, resulta interessant també la segmentació d'aquestes comunicacions. Aquesta segmentació, a part d'oferir un esquema molt més clar de quins són els dispositius que intervenen en les diferents comunicacions, ténen sentit per tal d'acotar l'àmbit d'aquestes. Aporta seguretat al sistema ja que confina la informació en la capa en la qual aquesta té sentit i no pot ser accedida des de dispositius que es troben en altres capes.

La segmentació de les comunicacions pot abordar-se des de moltes perspectives. Els factors que influeixen en aquest particionat de les comunicacions poden ser molts. Exemples en són el tipus de dispositiu, el tipus d'informació que es necessita processar, la distribució geogràfica dels dispositius, etc. Però la segmentació no necessàriament ha de dependre només d'un únic factor. Podria definir-se un esquema de particionat prenent en consideració dos o més factors conjuntament.

En el cas d'aquest projecte, pot resultar interessant prendre en consideració tant el tipus de dispositius com la informació que necessiten gestionar aquests. D'aquesta manera, es proposa un esquema dividit en 3 capes de comunicació. Cadascuna d'aquestes capes representa unes necessitats de comunicació diferents i implica a diferents dispositius. A la figura 5.2 s'observa la segmentació proposada amb la diferenciació de les 3 capes i els dispositius que intervenen en cadascuna i a continuació s'expliquen les diferents capes.

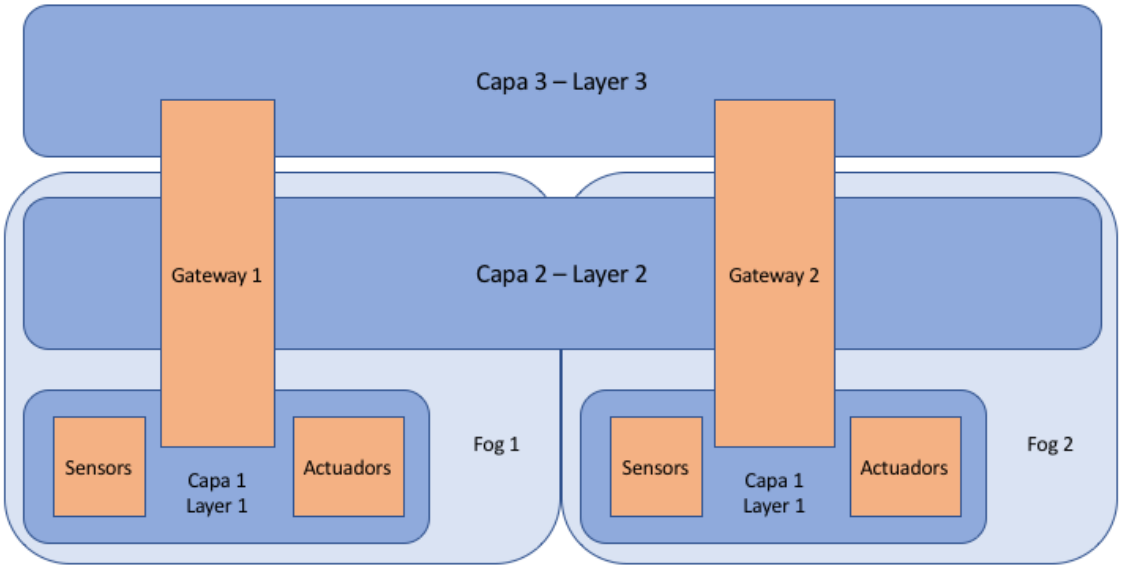

Figura 5.2: Esquema de les capes de comunicació i els dispositius que intervenen.

5.3.1. Capa 1 - Comunicació *intrafog*

Tal i com s'observa a l'esquema de la figura 5.2, la capa 1, destinada a les comunicacions internes dins del *fog*, està continguda dins de l'abast del *fog* al qual està assignada. Es pot veure també quins dispositius formen part d'aquesta capa.

Els sensors i actuadors només produeixen o consumeixen dades que circulen per aquesta capa. Aquests dispositius no ténen visibilitat ni accés a la informació que estiga continguda en altres capes. A més dels sensors i actuadors, les *gateways*, o nodes centrals del *fog*, també ténen accés a la informació d'aquesta capa.

La capa 1 és la capa més propera a l'extrem de la xarxa i del sistema. És a dir, és la més propera als dispositius productors i consumidors finals d'informació. En aquesta capa és on es troben totes les mesures dels sensors del fog i les ordres dirigides als actuadors del mateix.

5.3.2. Capa 2 - Comunicació *interfog*

La capa 2, que, com el seu nom indica, està destinada per a la comunicació *interfog*, és una capa que podria ser considerada *peer-to-peer*, ja que la comunicació s'establex entre iguals. Aquest concepte, el que intenta definir és que la capa 2 es tracta d'una capa destinada a la comunicació d'informació entre els diferents nodes centrals, o gateways, que estan actuant com a representants dels *fogs* del sistema.

Tal i com s'observa a l'esquema 5.2, al contrari que la capa 1 que està continguda dins del *fog*, aquesta, en canvi, se superposa als diferents *fogs*. Formada només per dispositius *gateway*, que representen al *fog*, per aquesta capa circula només informació d'interés dels que serien els nodes centrals dels diferents *fogs* del sistema.

Per tal de clarificar la decisió de segmentar les comunicacions d'aquesta manera, es pot plantejar la pregunta: per què els sensors i actuadors no formen part d'aquesta capa? La resposta és evident. Aquests dispositius no ténen la necessitat d'accedir a la informació que circula per aquesta capa perquè, com s'ha comentat a l'anterior paràgraf, eixa informació és només de l'interés dels dispositius *gateway*.

Aplicacions que se li poden atribuir a aquesta capa són, per exemple, la capacitat de realitzar difusions entre els diferents *fogs*. Un altre exemple d'aplicació de la capa seria la consulta d'estat entre els *fogs*. Aquest segon exemple podria utilitzar-se per a l'adaptació més personalitzada i ràpida del sistema a canvis dins d'aquest.

5.3.3. Capa 3 - Comunicació de gestió

Aquesta última capa, també referenciada com a capa de gestió, és l'encarregada de connectar els diferents *fogs* del sistema a un sistema més complex per a la gestió integral de tots. Tal i com s'observa a l'esquema 5.2, d'aquesta capa només en formen part els nodes centrals de cadascun dels *fogs* i, no apareix a l'esquema, qualsevol sistema de control que interactue amb els *fogs*.

Per aquesta capa de la comunicació només circula informació que és d'interés per als nodes *gateway* i pot ser de molts tipus. Els més rellevants per al projecte són les dades que modifiquen el comportament dels *fogs* del sistema i les dades que proporcionen informació als sistemes que formen part d'aquesta capa.

Una de les principals aplicacions que poden atribuir-se a aquesta capa és, per exemple, l'aplicació de polítiques que modifiquen el funcionament del sistema per part dels sistemes de control. Aquests sistemes de control, que poden estar controlats directament per persones o no, poden aplicar certes polítiques. Aquestes són transmeses als nodes centrals dels *fogs*, i són aquests qui decideixen quines mesures prendre per tal de donar compliment a les polítiques que s'han aplicat.

Un altre exemple d'aplicació és la transmissió de dades estadístiques. Per exemple, els diferents nodes poden proporcionar dades de contaminació, nombre de sensors vinculats, etc. i sistemes que formen part d'aquesta capa 3 podrien aprofitar-les per a oferir altres serveis, etc. És evident que les aplicacions d'aquesta capa queden molt obertes al tipus de sistemes o serveis que formen part d'aquesta.

5.4 Format de les dades

Tal i com s'havia avançat en apartats anteriors, en aquest es parla de les dades. Es prenen en consideració alguns formats de dades que existeixen i que s'utilitzen actualment i es plantegen els avantatges i inconvenients de la seua utilització en relació al sistema objecte d'estudi.

En aquest apartat es parla de dues formats reconeguts, que són JSON i XML. No obstant, el disseny del sistema no limita, en ningun cas, el format de les dades. Factors que sí limiten i que determinen quin és el format més òptim per a la seua

utilització en el sistema són, com ja s'ha comentat en altres apartats i seccions, qüestions com l'ample de banda disponible, la capacitat de càlcul dels dispositius, etc.

5.4.1. JSON

JSON (*JavaScript Object Notation*) és un format de text lleuger basat en *open-standard* pensat i dissenyat per a l'intercanvi de dades. Aqust format utilitza objectes formats per una parella de clau-valor de text llegible. És un dels formats més utilitzats per a la comunicació asíncrona dels serveis web.

Una de les característiques de JSON que el fan realment atractiu per a la seua utilització és que permet la realització d'un fàcil *parsing* de les dades. De fet, llenguatges com JavaScript permeten la construcció d'objectes directament des de les dades en format JSON utilitzant una funció que ve inclosa en el propi llenguatge. No obstant, altres llenguatges de programació, com per exemple Java, disposen de llibreries públiques que també permeten la fàcil creació d'objectes a partir de dades JSON.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Figura 5.3: Exemple de dades en format JSON.

En relació al sistema objecte d'aquest projecte, JSON proporciona un format de dades lleuger, la qual cosa redueix l'ample de banda consumit i permet l'augment de l'escalabilitat del sistema i la reducció del consum dels dispositius. A més, la facilitat d'aquest per a ser parsejat permet la simplificació de la complexitat del programari dels diferents dispositius.

5.4.2. XML

XML (*Extensible Markup Language*) és un llenguatge de marcat que defineix un conjunt de regles amb la finalitat de codificar documents en un format textual

que és llegible per als humans però que també és interpretable per màquines. Existeixen moltes especificacions d'XML, totes elles *open-standard*, però la més representativa és la proposada per l'organització W3C.

Encara que el disseny d'XML se centra en la creació i codificació de documents, aquest és utilitzat de manera generalitzada per a la representació d'estructures de dades personalitzades, com seria el cas en aquest projecte. Aplicacions que utilitzen el format XML per a representar les dades són, per exemple, els serveis web.

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
  </address>
  <phoneNumber>
    <type>home</type>
    <number>212 555-1234</number>
  </phoneNumber>
  <phoneNumber>
    <type>fax</type>
    <number>646 555-4567</number>
  </phoneNumber>
</person>
```

Figura 5.4: Exemple de dades en format XML.

Encara que d'alguna manera similar al format de dades JSON, XML presenta una estructura bastant més complexa i pesada que la de JSON. En contraposició a l'estructura en forma de parella clau-valor que proposa JSON, XML utilitza més estructures de control, tal i com es pot observar a l'exemple representat a la figura 5.4.

Un dels factors que fan el format XML més pesat que el format JSON és, per exemple, que XML necessita una etiqueta d'inici i una de final per tal de marcar l'abast de l'element. Açò permet que les dades siguen més fàcilment llegibles per part dels humans, però aporta càrrega d'informació no aprofitable que acaba consumint ample de banda. A més, altres estructures de control de les dades, com els atributs, ofereixen més informació sobre les dades, però pagant també una penalització en la lleugeresa de les dades.

A pesar de ser un format més pesat, no suposa una penalització crítica de cara a les necessitats del projecte i, tant el format JSON com el format XML podrien ser vàlids per a la representació i transmissió de dades. A més, a més, els diferents components del sistema no han de lligar-se necessàriament a ningun format de dades ja que es pot afegir fàcilment una capa d'integració de dades entre els dispositius que s'encarrega de l'adaptació dels formats.

5.5 Disseny de la capa de seguretat

En aquest apartat es presenta el disseny de diferents mesures de seguretat per tal d'afegir una capa que proporcione seguretat al sistema. L'objectiu és identificar i autenticar els diferents dispositius que formen part del *fog* i, només després de que aquests siguen validats, el xifrat de les comunicacions que s'estableixen. D'aquesta manera s'implementa un doble control. En primer lloc, ningú que no siga autoritzat pot formar part del *fog*. I en segon lloc, suposant que algu aconsegueix saltar el primer control, seguiria sense tenir accés a les comunicacions d'altres dispositius.

5.5.1. Disseny del protocol de *handshake*

En primer lloc es presenta un protocol de *handshake*. Tal i com es pot interpretar pel nom, la funció d'aquest protocol és presentar, identificar i autenticar el dispositiu amb el node central del *fog*. Com a resultat d'aquesta presentació, tant el node central com el dispositiu que solicita el *handshake* deriven una clau simètrica que és utilitzada posteriorment per al xifrat i desxifrat de les comunicacions.

S'ha decidit dividir el protocol de *handshake* en dues fases. La primera, la qual s'anomena petició o *request*, és la que executen els dispositius que soliciten la incorporació a un determinat *fog* i l'autenticació per part del node central com a dispositius legítims. La segona fase, anomenada resposta o *response*, és on es realitza la ressolució del protocol. La funció d'aquesta fase és la d'acabar d'establir la clau simètrica i registrar el dispositiu per part del node central.

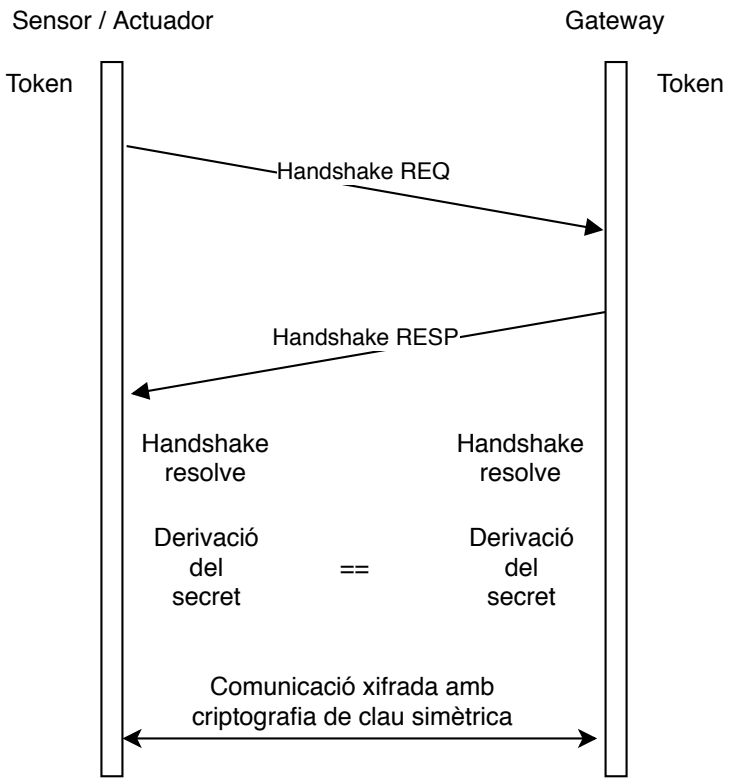

Figura 5.5: Esquema del funcionament del protocol de *handshake*.

Tots els dispositius que formen part d'un *fog* han de disposar del mateix *token*. Aquest *token* és utilitzat per a l'autenticació del dispositiu que requereix el *handshake* com a dispositiu legítim i que té permesa la incorporació al *fog*. A més, com ja s'ha comentat a la segona fase del protocol de *handshake*, s'utilitza per a derivar la clau simètrica que s'estableix entre els dos dispositius.

Si un dispositiu i un node central no disposen del mateix *token*, és a dir, no pertanyen al mateix *fog*, en la segona fase del protocol de *handshake* derivaran claus simètriques diferents. Com que aquesta clau és utilitzada per al posterior xifrat de les dades, la implicació que té el no disposar del *token* adequat és que les comunicacions entre els dos dispositius seran indesxifrables i, per tant, ignorades.

5.5.2. Xifrat i desxifrat de dades

Com ja s'ha comentat a la secció anterior, una de les finalitats del protocol de *handshake* és la derivació d'una clau simètrica entre els dos dispositius que intervenen en la comunicació. Aquesta clau derivada és utilitzada per al xifrat i desxifrat de dades una vegada el dispositiu ha sigut autenticat dins del *fog*.

Aquesta clau pot ser utilitzada amb algoritmes de xifrat de criptografia basada en clau simètrica. Exemples d'aquests algoritmes n'hi ha molts. Qualsevol d'aquests pot ser utilitzat per al xifrat i desxifrat de les dades. Com que les claus simètriques només les ténen emissor i receptor, ningun altre dispositiu és capaç de desxifrar les comunicacions entre aquests, encara que estiga correctament autenticat dins del *fog*.

5.6 Componentització / modularització

En aquest apartat es presenten els diferents components que formen part del sistema. Encara que en altres apartats ja s'ha parlat de components, en aquest es fa referència a components en el sentit de classes. Es presenten totes les classes que formen part del projecte, així com una xicoteta descripció d'aquestes.

5.6.1. Components

Analyzers

Components encarregats de l'anàlisi de mesures. Associats a dispositius *gateway*. N'existeixen tants de diferents com tipus de mesures haja d'analitzar la *gateway*.

- **CO_2Analyzer**: component encarregat de l'anàlisi de mesures de CO_2.

Authenticators

Components que s'encarreguen de la gestió del protocol de *handshake*. Associats a qualsevol tipus de dispositius, en el cas de sensors i actuadors s'encarreguen d'autenticar-se al node central del *fog* i en el cas de dispositius *gateway* gestionen

les autenticacions d'altres dispositius. A més, són els encarregats dels processos de xifrat i desxifrat de dades.

- **DefaultSensorAuthenticator**: tipus genèric d'autenticador dissenyat per sensors.
- **DefaultGatewayAuthenticator**: tipus genèric d'autenticador dissenyat per *gateways*.
- **DefaultActuatorAuthenticator**: tipus genèric d'autenticador dissenyat per actuadors.

Commands

Components que representen les dades dels comandaments. N'existeixen tants tipus com tipus de components hagen d'interpretar comandaments.

- **GatewayCommand**: comandaments que executen dispositius *gateway*.
- **TrafficLightCommand**: comandaments que executen els dispositius actuadors de tipus TrafficLight.

Communicators

Components encarregats de l'establiment i gestió de les comunicacions. Existeixen tants components d'aquest tipus com tipus de comunicació haja de suportar el sistema. Aprofitant la discussió sobre comunicació síncrona i assíncrona, si el sistema requereix de comunicacions dels dos tipus, deuen existir, com a mínim, un component per a comunicacions síncrones i un altre per a assíncrones.

- **RabbitMQ_Communicator**: component encarregat de la comunicació assíncrona mitjançant cues de missatgeria, concretament la implementació de RabbitMQ, que es comenta més endavant.

DataManagers

Aquests components s'encarreguen de la generació o interpretació de dades en un determinat format. Se'n requereixen tants com tipus de dades ha d'utilitzar el sistema. Aprofitant la discussió prèvia, si el sistema utilitza JSON i XML, han d'existir almenys un component per a cada format de dades.

- **JSON_DataManager**: component especialitzat en el tractament de dades en format JSON.
- **XML_DataManager**: component especialitzat en el tractament de dades en format XML.

DataTypes

Components que modelen les dades amb les que treballa el sistema. Existeixen, com a mínim, tants components com tipus de dades s'han comentat a l'apartat sobre tipus de dades del capítol 4.

- **CommandData**: model de les dades que incorpora un missatge de tipus comandament.
- **HandshakeData**: model de les dades que incorpora un missatge de tipus *handshake*.
- **MeasureData**: model de les dades que incorpora un missatge de tipus mesura.
- **NotificationData**: model de les dades que incorpora un missatge de tipus notificació.
- **PolicyData**: model de les dades que incorpora un missatge de tipus política

Devices

Component que modela els diferents dispositius que formen part del sistema. Es consideren els components principals comentats en apartats anteriors.

- **Gateway**: model d'un dispositiu del tipus *gateway*.
- **Sensor**: model general dels dispositius de tipus sensor.

  CO_2Sensor: dispositiu sensor especialitzat en la producció de mesures de l'element CO_2.
- **Actuator**: model general dels dispositius de tipus actuador.

  TrafficLight: dispositiu actuador, concretament representa un controlador semafòric.

Measures

Component que representa les mesures dels sensors. Incorpora tots els camps que s'han vist al capítol 4.

- **Measure**: model de les mesures de qualsevol tipus.

Messages

Component que representa els missatges que són formatats i comunicats entre els sensors. Incorporen la informació necessaria per a la seua transmissió i les dades que transporten.

- **Message**: model general de missatge.
- **EncryptedMessage**: model de missatge que transporta dades xifrades. Aquest tipus de missatge incorpora alguns camps addicionals al model Message.

Policies

Representació de les dades que formen part d'una política. Incorpora els camps que s'han vist al capítol 4.

- **Policy**: model de política de qualsevol tipus.

5.6.2. Diagrama de classes

Després d'haver presentat i comentat els diferents components que formen part del sistema, en aquest apartat es presenta un diagrama de classes que representa l'estructura interna del sistema. A més de les relacions que s'estableixen entre els diferents components del sistema, es mostra també els atributs i funcions dels que disposa cadascun d'ells. Encara que es tracta d'un diagrama de classes simplificat i no està complet al 100%, s'ha considerat que és el més representatiu per a l'enteniment de la composició i funcionament del programari que es desenvolupa.

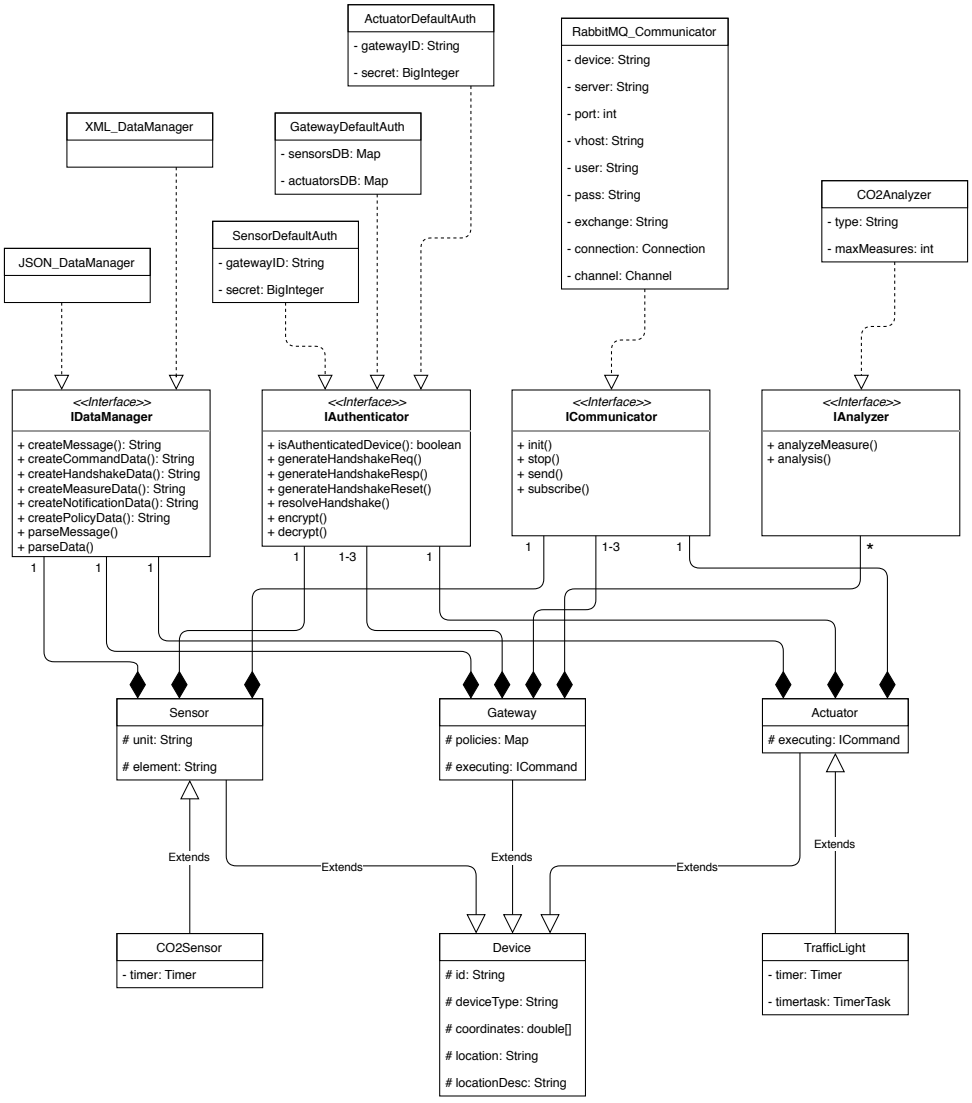

Figura 5.6: Diagrama de classes simplificat del projecte.

Al diagrama de la figura 5.6 s'observa clarament l'estructura comentada durant tot el document. Es poden diferenciar fàcilment els tres tipus de components principals dels quals s'ha parlat a l'apartat del model conceptual. Aquests tres components principals estan representats per les classes *Sensor*, *Gateway* i *Actuator*. També es pot veure que existeixen especialitzacions de les classes *Sensor* i *Actuator*. Aquestes especialitzacions representen els diferents tipus de sensors i actuadors suportats pel sistema.

Seguint amb l'esquema comentat al model conceptual sobre components principals i components complementaris, al diagrama de classes representat a la figura 5.6 s'observen algunes relacions de composició que relacionen els components principals amb unes interfícies. Aquestes interfícies representen els components complementaris comentats al model conceptual i la seua funció és especificar les funcions que han d'oferir aquests components.

La pròpia definició de la relació de composició indica que no té sentit l'existència independent o autònoma dels components complementaris, representats per les interfícies, sinó que el sentit de l'existència d'aquests és complementar i acabar de completar els components principals. Aquests components principals necessiten utilitzar les funcionalitats que ofereixen les interfícies i que estan representades al diagrama de classes.

Per últim, és important comentar les relacions d'implementació. Tal i com es pot veure al diagrama de classes de la figura 5.6, existeixen diferents classes que implementen una determinada interfície. La classe *JSON_DataManager* i la classe *XML_DataManager* en són exemples. Ambdues implementen la interfície *IDataManager*. Açò significa que les dues classes comentades ofereixen les mateixes funcionalitats, però cadascuna d'una manera diferent. Per acabar d'exemplificar-ho, tant *JSON_DataManager* com *XML_DataManager* permeten crear missatges o interpretar dades, que són les funcions que ofereixen, però *JSON_DataManager* ho fa amb dades de tipus JSON mentre que *XML_DataManager* ho fa amb el format XML.

CAPÍTOL 6
Implementació

Una vegada realitzada una anàlisi en profunditat del problema, estudiades les diferents possibilitats de disseny i seleccionada la més adequada d'aquestes per al projecte, és moment de plasmar aquest disseny i fer-lo real. Aquest capítol tracta aquest procés d'implementació del disseny escollit, centrant-se en el paradigma de programació i en com s'implementen les comunicacions, la gestió de les dades i la seguretat de les comunicacions.

6.1 Programació orientada a components

La programació orientada a components (o basada en components), CBSE, de l'anglés *Component-Based Software Engineering*, és una metodologia de desenvolupament de programari que dóna especial importància a la separació de les funcionalitats d'un sistema. En aquest apartat es presenta aquest paradigma de programació i, a més, la tecnologia emprada per a aplicar aquest paradigma al projecte, en aquest cas OSGi.

6.1.1. Definició del paradigma

Tal i com ja s'ha presentat, la programació orientada a components és un paradigma de programació que emfatitza en la diferenciació i separació de les funcionalitats d'un sistema. Aquestes funcionalitats diferenciades són assignades, o suportades, per diferents components.

Un component és un mòdul que encapsula diferents funcionalitats del sistema, molt relacionades entre elles. De fet, és possible que un component realitze una única funció. Açò es fa per tal de que totes les dades o funcions que incorpora un determinat component estiguen relacionades semànticament. És a dir, aquestes dades i funcions ténen un significat més complet quan estan juntes.

Els components es comuniquen entre ells utilitzant interfícies. Una interfície no és més que una declaració de les funcions que pot dur a terme un determinat component. D'aquesta manera, altres mòduls que facen ús d'aquest component saben què és el que pot fer aquest, però no necessiten saber de ninguna manera quin és el funcionament intern d'ell. Per tal de crear components que propor-

cionen unes determinades funcionalitats, aquests només han d'implementar la interfície que les defineix.

Per tal de deixar més clar aquest concepte es presenta el següent exemple. En el cas del projecte, hi ha un tipus de components, els Communicators, que s'encarreguen d'establir les comunicacions. Existeix una implementació concreta que utilitza RabbitMQ per tal d'establir aquestes comunicacions, però podria existir-ne una altra que utilitzara serveis RESTlet. La solució passa per que les dues implementacions implementen la interfície Communicator. D'aquesta manera, els altres components només saben que eixes implementacions ofereixen les funcions que declara la interfície Communicator, però no saben internament si les comunicacions s'estableixen amb RabbitMQ o RESTlet.

El principal motiu de la utilització de la programació basada en components és que permet una gran reutilització de programari, ja que si un altre sistema o projecte requereix fer una funció ja implementada per algun component utilitzat en algun projecte anterior, aquest pot integrar-se directament al projecte nou. Però a més, aquest paradigma de programació permet augmentar l'escalabilitat i adaptabilitat del programari desenvolupat, ja que és fàcil canviar components que implementen la mateixa funcionalitat perquè aquest paradigma es basa, des del propi disseny, en el desacoblament de components i funcionalitats gràcies a la utilització d'interfícies.

6.1.2. Desenvolupament basat en OSGi

OSGi és un *framework* que es basa en la programació de components, els quals s'anomenen també *bundles* o *plug-ins*. Tal i com s'ha comentat en la definició del paradigma de programació basada en components, cada component està format per un conjunt de funcionalitats del sistema. No obstant, OSGi no és un paradigma de programació, com sí ho és la programació orientada a components, sinó una aplicació directa d'aquesta. És a dir, OSGi és l'aplicació d'aquest paradigma i alguns conceptes més a les implementacions que es desenvolupen en el llenguatge de programació Java.

A la figura 6.1 s'observen les diferents relacions que es poden establir entre *bundles* d'OSGi en un sistema. El *bundle* B és una interficie que ofereix unes determinades funcionalitats. Aquesta publica les funcionalitats que ofereix al *Service Registry*. El *Service Registry* és un característica que ofereix el *framework* OSGi i que permet als *bundles* publicar les funcionalitats que ofereixen. Açò és útil per a altres *bundles* que busquen determinades funcionalitats de manera dinàmica. Més tard es comenta aquest aspecte.

Les funcionalitats que ofereix la interfície del *bundle* B té dues implementacions diferents. Aquestes dues implementacions vénen en els *bundles* C i D. Aquests dos *bundles* realitzen les funcions que indica el *bundle* B, però cadascun d'una manera diferent. Aquest aspecte és el que s'ha comentat abans, que implica que la resta de components que necessiten aprofitar les funcionalitats del *bundle* B no necessiten coneixer si qui realment fa la funció és el *bundle* C o el D.

Una vegada publicada determinada funcionalitat al *Service Registry* i implementada per algun component, altres *bundles* poden buscar al *Service Registry*

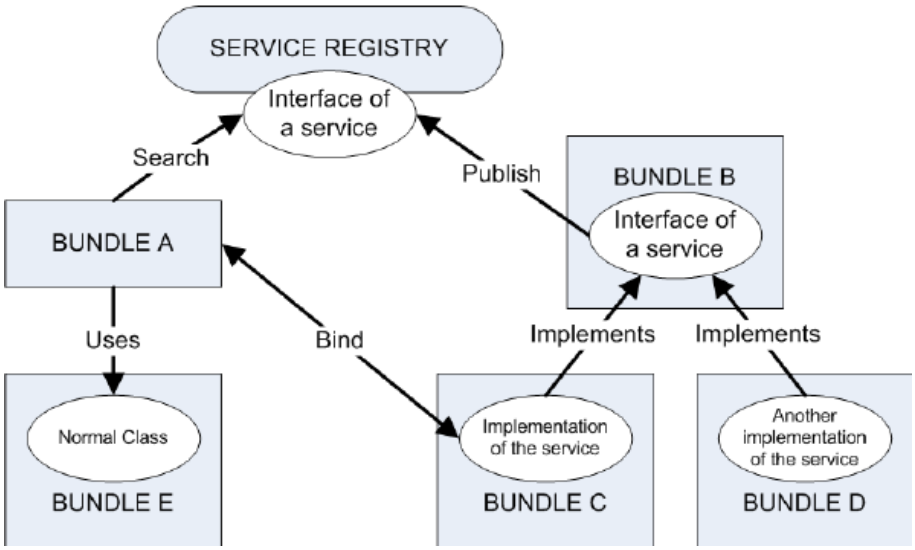

Figura 6.1: Relacions entre diferents *bundles* d'OSGi.

aquesta funcionalitat perquè necessiten incorporar-la. És el cas del *bundle* A, que busca la funcionalitat que ofereix el *bundle* B i se li proporciona un component capaç de satisfer-la, en aquest cas el *bundle* C. Com es pot observar a la figura, aleshores s'estableix un *binding* entre el *bundle* A i C.

Encara que a la figura 6.1 no s'observa, hi existeixen un tipus de *bundles* que incorporen una classe anomenada *Activator*. Aquest tipus de components es poden considerar com als components que executen la funció principal, o *main*. És a dir, aquests *bundles* són els que realment poden arrancar, parar, etc. En definitiva, aquests *bundles* són els que passen pel diagrama d'estats de la figura 2.4, comentada al context tecnològic.

6.2 Implementació de les comunicacions

En aquest apartat es tracta la implementació del sistema de comunicació utilitzat en el prototip que es desenvolupa en aquest projecte. Una vegada analitzades totes les possibilitats en l'anàlisi del problema i considerats els motius pertinents a la fase de disseny de la solució, finalment s'ha decidit implementar la solució, i concretament les comunicacions, utilitzant les cues de missatgeria RabbitMQ.

6.2.1. Servidor RabbitMQ

En aquesta secció es parla de la preparació i configuració del servidor de cues RabbitMQ. S'expliquen els motius de l'elecció d'aquesta implementació concreta i es presenta una forma de desplegar-lo en local per tal de realitzar proves del prototipus final, utilitzant la ferramenta Docker, comentada també a l'apartat del context tecnològic.

RabbitMQ és un *broker* de missatgeria basat en un projecte *open-source* que ofereix un servei de missatgeria assincrònica. Pot funcionar sobre un gran nombre de sistemes operatius diferents i no sobrecarrega aquests. A més, disposa de

moltes ferramentes i llibreries que permeten utilitzar-lo amb molts llenguatges de programació, com és el cas de Java. Per aquests motius, s'ha considerat RabbitMQ la millor implementació per a utilitzar en el prototip que es desenvolupa al projecte.

No obstant, RabbitMQ ofereix algunes característiques més, que el fan encara més atractiu de cara a la seua utilització en aquest projecte. A continuació es comenten algunes d'aquestes característiques, com la possibilitat de desplegar els serveis de RabbitMQ de manera distribuïda, en forma de *clusters* que ofereixen un alt rendiment i una altra disponibilitat. A més, una altra característica que ofereix RabbitMQ és la fàcil monitorització i gestió d'aquest a través d'una interfície web.

En aquest projecte es proposa la instal·lació d'un servidor RabbitMQ en cada node central del *fog*. D'aquesta manera, la càrrega de missatges és repartida entre cada *fog* i permet una major escalabilitat del sistema. Els servidors RabbitMQ poden desplegar-se instal·lant-se directament sobre el sistema operatiu del node central de cada *fog*, o poden desplegar-se com a contenidors de Docker. Aquesta segona opció és l'utilitzada en aquest projecte.

El desplegament dels servidors RabbitMQ mitjançant Docker permet la instal·lació i autoconfiguració d'aquests de manera ràpida. Aquest procés es realitza mitjançant la utilització d'un xicotet *bash script* i un Dockerfile. Docker és l'encarregat de construir els contenidors seguint les instruccions que conté el Dockerfile. D'aquesta manera, amb la combinació de l'script i el Dockerfile es desplega o arranca de manera fàcil els serveis de RabbitMQ en qualsevol node central. A l'annex C es poden consultar aquests fitxers i el procés de desplegament dels diferents servidors.

6.2.2. *Exchanges* i cues

En aquest apartat es presenten els diferents *exchanges* i cues que es declaren al servidor RabbitMQ per tal de segmentar les comunicacions de la manera que s'ha comentat als capítols d'anàlisi del problema i disseny de la solució. A la figura 6.2 es pot veure una representació esquemàtica de les relacions de comunicació que s'estableixen entre components tenint en compte la implementació dels *exchanges* i cues de RabbitMQ.

A la figura 6.2 es distingueixen 4 tipus d'elements. Aquests són: components principals del sistema més el sistema de gestió, representats amb un rectangle de color blau; els diferents *exchanges* declarats al servidor RabbitMQ, representats per rectangles de color verd amb les puntes arrodonides; les cues de missatgeria contingudes dins dels *exchanges*, representades per una canonada de color taronja; i per últim, les relacions de publicació (PUB) o subscripció (SUB) que s'estableixen entre els components del sistema i les cues de missatgeria.

Aquesta representació esquemàtica pot observar-se directament a la configuració del servidor RabbitMQ a través de la seua interfície web. A la figura 6.3 s'observen els *exchanges layer1* i *layer2* que es presentaven a l'esquema de la figura 6.2. A més, a la figura 6.4 s'observen també les cues contingudes a l'*exchange layer1* de l'esquema.

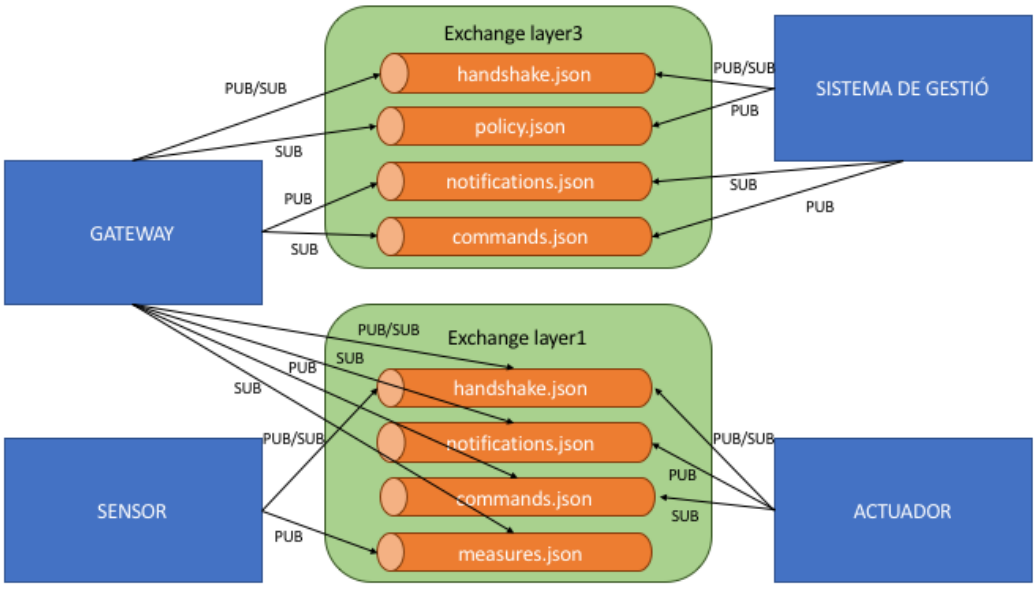

Figura 6.2: Representació de les relacions de comunicacions establides entre components.

Aprofitant que les captures de pantalla de les figures 6.3 i 6.4 es van realitzar mentre el prototip estava funcionant, es poden observar alguns detalls més de la interfície web del servidor RabbitMQ. A la figura 6.3 s'observa la taxa d'entrada i sortida de missatges dels diferents *exchanges*. Al seleccionar qualsevol d'ells es mostra informació més detallada sobre aquests. Com es pot veure a la figura 6.4, a més de la taxa d'entrada i sortida de missatges i la informació de les cues vinculades a l'*exchange*, es proporciona també una gràfica del rendiment d'aquest.

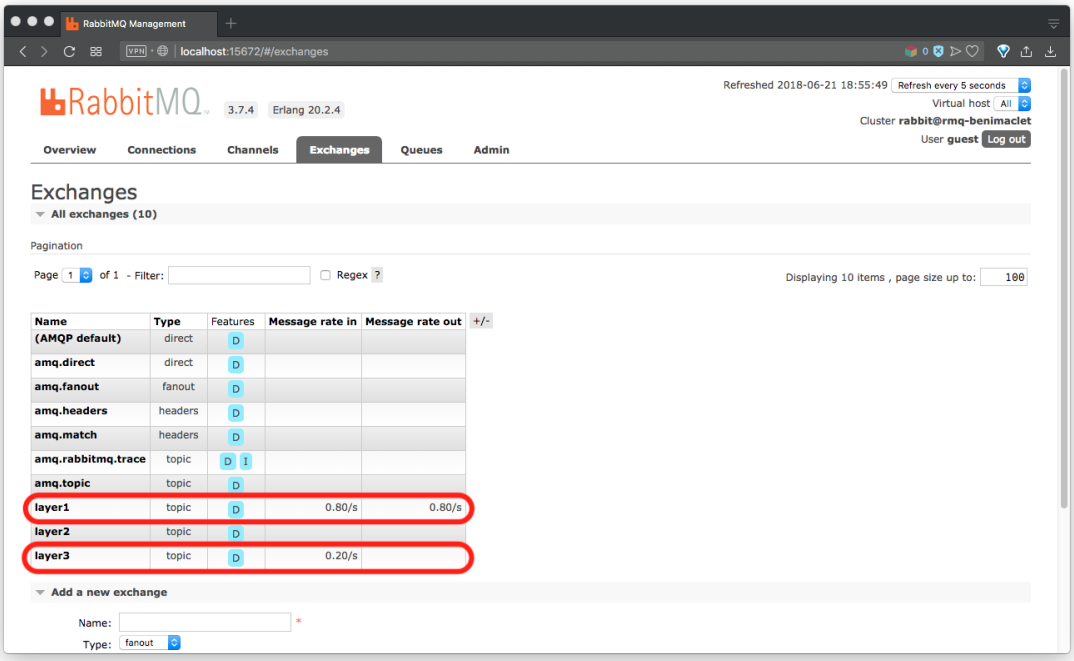

Figura 6.3: *Exchanges layer1* i *layer3* al servidor RabbitMQ.

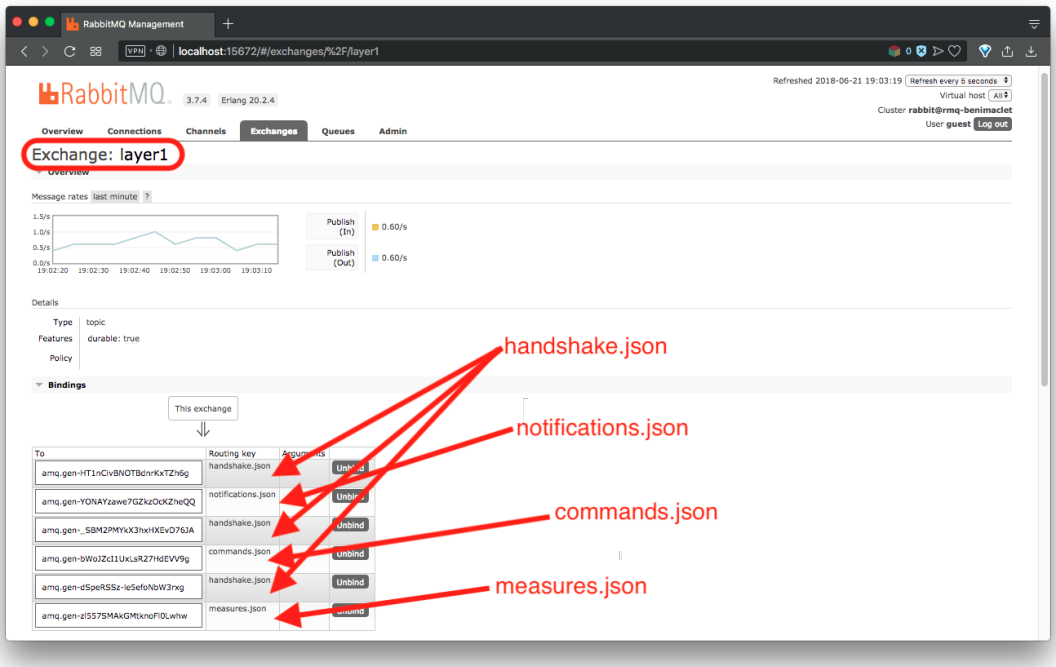

Figura 6.4: Cues contingudes dins de l'*exchange layer1*.

6.3 Implementació de la gestió de les dades

En aquest apartat es tracta la implementació de la gestió de les dades que utilitza el sistema en els diferents formats comentats en altres apartats, que són JSON i XML. Així doncs, es parla sobre les dues llibreries de Java que ho fan possible i com s'han utilitzat aquestes. A l'annex A es mostra la representació formatada de cada tipus de dades en els dos formats dels quals s'ha parlat en capítols anteriors.

6.3.1. Llibreria org.json

La llibreria *org.json* és una llibreria per al llenguatge de programació Java que permet modelar de manera ràpida i senzilla en forma d'objecte Java les representacions de dades formatades en format JSON. Aquesta llibreria permet la creació de nous objectes Java i l'exportació d'aquests en forma de text en format JSON, així com també permet la interpretació de text en format JSON i la creació d'un objecte Java que modela la informació de la representació en format JSON.

Aquesta llibreria permet la fàcil implementació d'analitzadors (o *parsers*) de text en format JSON. De fet, per tal d'obtindre una representació en format objecte de Java de qualsevol text en format JSON només cal una instrucció. Aquesta instrucció és *JSONObject javaObject = new JSONObject(<String de text en format JSON>);*. El resultat d'açò és un objecte Java de nom *javaObject* que és una representació de la informació de l'String de text en format JSON.

A més de facilitar el *parsing* de text en format JSON, la llibreria *org.json* també permet crear directament objectes Java que representen text en format JSON. Existeixen dos tipus principals d'objectes, que són *JSONObject* i *JSONArray*. Ambdós

poden contindre objectes del mateix tipus o de l'altre tipus, a més de tots els que són tipics de Java. La creació d'aquests objectes és tan senzilla com *JSONObject object = new JSONObject();* o *JSONArray array = new JSONArray();*.

Com ja s'ha vist, el format JSON està compost per parelles de clau-valor. Per tal d'afegir dades dins de qualsevol dels dos objectes que s'han comentat, només cal executar *jsonObject.put(<String clau>, <objecte valor>);*, on *jsonObject* és l'objecte on es vol introduir les dades i String clau i objecte valor fan referència a la parella clau-valor comentada. De la mateixa manera s'afegeixen dades als *JSONArray*. Afegint objectes i dades dins dels objectes *JSONObject* i *JSONArray* és com va creant-se l'arbre.

Però no només es poden introduir dades als objectes comentats. També poden ser llegides des d'aquests. Per exemple, una vegada realitzat el *parsing* d'un objecte, és probable que s'hagen de consultar determinades dades d'aquest. Per tal de realitzar açò no cal més que demanar a l'objecte pertinent el contingut de determinada clau. Executar *jsonObject.getString(<clau>);* permetria obtindre el valor (de tipus String) etiquetat amb la clau indicada. És important remarcar que cal utilitzar el mètode adequat, ja que si l'objecte que s'espera obtenir és un Integer, per exemple, no es pot utilitzar el mètode *getString()* o qualsevol altre, s'hauria d'utilitzar *getInt()*.

6.3.2. Llibreria org.dom4j

La llibreria *org.dom4j*, a l'igual que la llibreria *org.json*, és una llibreria per al llenguatge de programació Java que permet modelar objectes Java que representen dades en format XML. De la mateixa manera que l'anterior llibreria, aquesta també permet la creació de text en format XML a partir d'objectes Java i la interpretació de text XML per tal de modelar objectes Java.

Aquesta llibreria permet implementar de manera fàcil *parsers* per a les dades en format XML. De la mateixa manera que la llibreria *org.json*, aquesta també ofereix la funcionalitat de representar en un objecte de Java les dades en format XML que se li indiquen. En aquest cas, la representació de l'XML es realitza en un objecte Document de Java. Per tal de realitzar el *parsing* de les dades XML només cal executar *Document doc = DocumentHelper.parseText(<text en format XML>);*, que modelarà el contingut del text en format XML i retornarà un objecte de tipus *Document*.

Una vegada representat l'XML en un objecte *Document*, la llibreria *org.dom4j* també permet l'accés a les dades d'aquest objecte. Com que els diferents components dins d'un arbre XML també vénen denotats per una etiqueta, *org.dom4j* permet la selecció d'aquests components utilitzant la notació DOM. Una vegada es té el component adequat seleccionat, es poden consultar les dades d'aquest, ja siga el propi contingut, els atributs, etc. La instrucció *String msgType = doc.selectSingleNode("//<etiqueta>").getStringValue();* mostra ambdós comportaments. En la primera part, *doc.selectSingleNode("//<etiqueta>")*, s'està realitzant la selecció de l'element amb l'etiqueta que s'indique; i en la segona part, *getStringValue();*, s'extrau el contingut de l'element seleccionat, que en aquest cas, el contingut és una cadena de text.

La llibreria *org.dom4j* no només permet el *parsing* i consulta de dades en format XML. A més, aquesta llibreria permet també la creació de noves dades en format XML. De la mateixa manera que es pot fer amb *org.json*, aquesta llibreria permet la creació de documents en format XML a partir d'objectes de Java. Açò s'aconsegueix, en primer lloc, creant un objecte de tipus *Document* amb la instrucció *Document doc = DocumentHelper.createDocument();*. A aquest objecte se li poden afegir elements amb etiquetes.

Aquests elements poden tenir contingut, atributs, etc. Tot açò es pot afegir amb instruccions de l'estil de *doc.addElement(«etiqueta>").addText(<contingut en format textual de les dades>);*. Aquesta instrucció es pot dividir en dues parts. En la primera, *doc.addElement(«etiqueta>")*, s'indica qui és el pare de l'element que s'està afegint, en aquest cas el propi document, i quin és el nom o etiqueta de l'element afegit. I en la segona part, *addText(<contingut en format textual de les dades>);* s'indica el contingut del nou element.

6.4 Implementació de la capa de seguretat

A partir del disseny de la capa de seguretat del capítol 5, en aquest apartat es presenta com s'implementa realment aquesta capa de seguretat. Es passa de l'abstracció del disseny a la utilització de protocols concrets per tal de derivar claus simètriques, autenticar dispositius i xifrar o desxifrar dades.

6.4.1. Adaptació del protocol *Diffie-Hellman Key Exchange*

El disseny del protocol de *handshake* és una adaptació d'un protocol molt conegut d'intercanvi de claus. Es tracta de *Diffie-Hellman Key Exchange*. Aquest protocol s'utilitza per a l'establiment de claus simètriques de manera segura entre dos dispositius a través d'un canal naturalment insegur. Encara que es podria aprofundir molt en els principis que ho permeten, per tal de simplificar s'explica que la seguretat d'aquest algoritme d'intercanvi de claus resideix en la utilització d'una funció unidireccional, que és una funció que es caracteritza perquè, a partir del resultat, no es poden obtenir els factors inicials.

Tal i com es pot veure a l'esquema de la figura 6.5, l'element etiquetat amb el nom *Alice* genera uns valors a, que actua de clau privada, g, que actua de generador i p, que és un valor modular. L'element *Alice* calcula el valor A com es mostra a la figura, utilitzant la funció unidireccional, que és l'exponenciació modular. Una vegada calculat el valor A, aquest element transmet a l'element *Bob* els valors g, p i A, el càlcul que ha realitzat.

Com ja s'ha comentat, el canal és insegur i algú podria interceptar aquests valors. És ací on la seguretat de la funció unidireccional assegura que ningú pot coneixer el valor a utilitzat per al càlcul d'A, ja que encara que es dispose dels valors g, p i A, no es pot obtenir a invertint l'exponenciació modular.

Una vegada l'element *Bob* rep els valors generats per l'element *Alice*, aquest genera un valor b que actua de clau privada i calcula el valor B de la mateixa manera que s'ha generat A i l'envia a l'element *Alice*. En aquest moment, ambdós

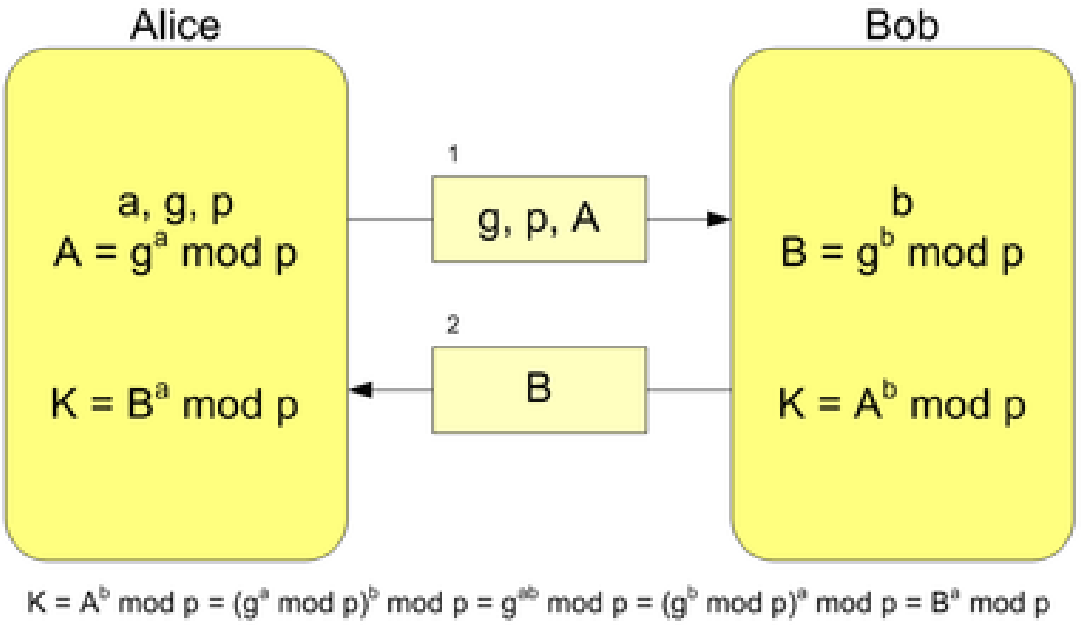

Figura 6.5: Esquema de l'intercanvi de claus Diffie-Hellman.

elements, *Alice* i *Bob* poden generar la mateixa clau simètrica aplicant l'exponenciació modular al valor que han rebut de l'altre element i elevant-lo a la seua clau privada, *a* i *b* respectivament. Com a resultat, ambdos element generen el valor *K*.

La modificació que es proposa sobre l'algoritme de *Diffie-Hellman* és la següent: considere's que el generador, a l'esquema representat per *g*, en lloc de ser decidit per l'element *Alice* i comunicat a l'element *Bob* és incorporat de manera manual tant als dispositius solicitants del *handshake* com al node central del *fog*. La derivació final de la clau simètrica seria correcta sí i només sí tant el node central com el dispositiu solicitant duen incorporat el mateix generador. Però a més, la seguretat del protocol no es veu compromesa de ninguna manera, ja que segueix residint en que l'exponenciació modular és una funció unidireccional.

Si a aquest generador se l'anomena *token*, ja es disposa d'una autenticació dins del *fog* basada en la possessió d'aquest *token*. D'aquesta manera, qualsevol dispositiu que no dispose del *token* corresponent és incapaç de derivar una clau simètrica correcta en la negociació amb el node central o *gateway*.

Açò soluciona la part d'autenticació dins de la capa de seguretat, ja que només dispositius legítims disposen del *token*, i el *token* determina de manera única a quin *fog* pot ser associat el dispositiu. I, a més, el resultat és una clau simètrica derivada amb el node central del *fog* que serveix per a xifrar i desxifrar les dades que es comuniquen amb aquest.

6.4.2. Xifrat i desxifrat amb *Advanced Encryption Standard*

La segona part de la capa de seguretat consisteix en el xifrat de les dades que es comuniquen dins del *fog*, de manera que cap dispositiu que no siga l'emissor

o receptor d'aquestes siga capaç d'interceptar-les i desxifrar-les. Aquest procés de xifrat i desxifrat es realitza, com ja s'ha adelantat al capítol 5, amb algoritmes basats en criptografia de clau simètrica.

L'algoritme seleccionat per a realitzar aquestes funcions és AES. Aquest protocol de clau simètrica utilitza la clau simètrica derivada del protocol de *handshake* i un vector d'inicialització per al xifrat de les dades. La clau simètrica sí és estàtica, però en canvi, el vectori d'inicialització és un valor aleatori que canvia amb cada conjunt de dades que es xifra. És per aquest motiu que el valor del vector d'inicialització ha de transmetre's juntament amb les dades xifrades.

6.5 Components finals del sistema

En aquest apartat es presenten els components que finalment s'empaqueten i es despleguen en el prototip. Tornant als tres tipus de components principals als quals s'ha fet referència durant tot el document, els components que s'empaqueten per al seu desplegament són tres també, un de cada tipus. Aquests tres components empaquetats són un node central genèric, que fa la funció de *gateway*, un CO_2Sensor, que fa les funcions de sensor, i un TrafficLight, que actua com a actuador, valga la redundància.

6.5.1. *Gateways*

Tal i com s'ha comentat, la implementació del sistema es basa en la componentització i en la utilització del *framework* d'OSGi. És per aquest motiu, que a l'hora de desplegar els components del sistema es necessiten diferents *bundles* d'OSGi. En concret, i per a empaquetar el component *gateway* que fa les funcions de node central del *fog* en el prototip són necessaris, a part dels *bundles* OSGi generals per a l'execució, els següents *bundles* desenvolupats al projecte.

- Smart_ecoMobility.CO2Analyzer_1.0.0.jar
- Smart_ecoMobility.Components_1.0.0.jar
- Smart_ecoMobility.FogGateway.Starter_1.0.0.jar
- Smart_ecoMobility.FogGateway_1.0.0.jar
- Smart_ecoMobility.Interfaces_1.0.0.jar

6.5.2. *CO_2Sensors*

A l'igual que en el cas del component anterior, per al desplegament dels sensors especialitzats en la mesura de CO_2 són necessaris diferents *bundles*. També obviant els que són bàsics per a l'execució, els *bundles* relacionats amb el projecte i que són necessaris per a donar forma a aquest component són els següents.

- Smart_ecoMobility.CO2Sensor_1.0.0.jar

- Smart_ecoMobility.Components_1.0.0.jar
- Smart_ecoMobility.CO2Sensor.Starter_1.0.0.jar
- Smart_ecoMobility.Interfaces_1.0.0.jar

6.5.3. *TrafficLights*

Per últim, els actuadors que representen el funcionament d'un controlador semafòric també han de ser empaquetats per al seu desplegament en el prototip. Els diferents *bundles* que formen part d'aquest component empaquetat són els que es comenten a continuació, obviant els necessaris per a l'execució.

- Smart_ecoMobility.TrafficLight_1.0.0.jar
- Smart_ecoMobility.Components_1.0.0.jar
- Smart_ecoMobility.TrafficLight.Starter_1.0.0.jar
- Smart_ecoMobility.Interfaces_1.0.0.jar

CAPÍTOL 7
Escenari concret d'ús

Una vegada dissenyada i implementada la solució, és moment d'aplicar-la sobre un escenari concret d'ús. En aquest capítol es presenta l'aplicació del projecte sobre un entorn real, com és la ciutat de València. En concret, es parla de l'organització i segmentació del sistema per tal d'adaptar-se a l'entorn, de la creació dels *fogs* i de la distribució dels diferents dispositius en aquests *fogs*.

7.1 Organització del sistema

L'objectiu d'aquest apartat és presentar de quina manera el sistema modela l'entorn sobre el que s'aplica. Concretament, es parla sobre el particionat de la ciutat, explicant els motius que han propiciat l'elecció d'eixe particionat i no de qualsevol altre, i a més, se seleccionen els diferents *fogs* que han servit per a la implementació del model.

7.1.1. Particionat de la ciutat

Per al modelat del sistema s'ha decidit basar-se en una segmentació basada en la geografia de la ciutat de València. En aquest cas, s'ha considerat el més adient adoptar el particionat natural de la ciutat i definir tants potencials *fogs* com districtes té la ciutat. Com es port veure a la figura 4.1, mostrada a l'apartat que parla sobre la segmentació basada en la geografia, amb aquest esquema de particionat de la ciutat s'obtindrien uns 19 potencials *fogs*.

S'ha decidit l'adopció d'aquest tipus de segmentació per diferents motius. El primer d'ells és perquè l'esquema de particionat resulta molt natural i fàcil d'entendre ja que encaixa en el mapa mental de la ciutat que tots ténen en ment. D'aquesta manera és més fàcil l'aplicació de polítiques sobre el sistema per part de qui no coneix l'estructura interna d'aquest. Per tal de posar un exemple fàcil, per a l'encarregat de medi ambient de l'Ajuntament de València serà més fàcil l'aplicació de poltítiques sobre el sistema si aquest encaixa en el mapa mental de la ciutat que l'encarregat coneix.

El segon dels motius pels quals s'ha decidit adoptar aquest esquema de particionat és perquè les decisions basades en la contaminació de determinada zona té sentit prendre-les tenint en compte zones definides. En aquest cas, les decisions

que afecten a un determinat districte vénen determinades per les mesures registrades en eixe mateix districte. D'aquesta manera s'aconsegueix acotar l'abast de les decisions i de les mesures de manera clara.

7.1.2. Elecció de *fogs*

Com s'ha comentat a l'apartat anterior, existeixen 19 possibles *fogs*. Però el prototip entregable d'aquest projecte no pot abarcar un nombre tan gran de *fogs* perquè no es disposa de suficient material disponible. Cal remarcar que el fet de no poder abarcar la creació dels 19 *fogs* no ve determinada per limitacions de disseny de la solució ni tampoc per limitacions en la implementació, sinó per, com ja s'ha comentat, la no disponibilitat del gran nombre de dispositius que serien necessaris.

En aquest cas s'ha decidit l'elecció de dos *fogs*. S'ha escollit un *fog* de tamany un poc més petit i un altre amb una extensió major. Aquesta decisió s'ha pres tenint en compte el tamany dels *fogs* per tal de proposar una distribució representativa dels diferents dispositius i acotar també el nombre d'aquests. Al següent apartat es detalla un poc més el motiu d'aquesta elecció.

Al les figures 7.1 i 7.2 s'observa l'abast dels dos *fogs* que s'han pres en consideració. Les decisions que prendrà el node central del *fog* estan acotades a aquest rang ressaltat, així com també la informació que rep aquest i que condiciona les decisions que pren.

Figura 7.1: Mapa del districte de Benimaclet.

Figura 7.2: Mapa del districte d'Extramurs.

7.2 Organització dels *fogs*

En aquest apartat es comenta l'organització interna dels *fogs*, així com el nombre i la localització dels diferents dispositius que es desplegaran en l'àrea. Aquesta organització interna ve determinada, en aquest cas, pel nombre de barris que forment part del districte.

Encara que es podrien estudiar altres tipus d'organització interna dels *fogs*, s'ha decidit aplicar aquest criteri per tres principals motius. El primer d'ells és per tal de continuar amb el particionat tradicional de la ciutat. Una ciutat es divideix en districtes, i cada districte es divideix en barris. Açò s'ha decidit que havia d'estar plasmat a la pròpia organització del sistema per tal de continuar en la línia de facilitar la comprensió del sistema.

El segon dels motius que han propiciat l'adopció d'aquest criteri és la coherència i la distribució dels sensors. S'ha considerat que la utilització d'un sensor per cada barri proporciona una informació representativa de l'estat general del districte (representat com a *fog*). A més, les distribucions dels barris pels districtes sol ser bastant equilibrada, així que la distribució de sensors per l'extensió del *fog* també ho serà.

I per últim, el tercer motiu que ha determinat l'adopció d'aquesta organització interna dels *fogs* i l'elecció dels districtes de Benimaclet i Extramurs ha sigut el nombre de barris que aquests dos districtes contenen. Benimaclet conté els barris de Benimaclet i Camí de Vera, la qual cosa suma un total de 2 sensors més un node central, determinant un total de 3 dispositius necessaris per al modelat del districte i la creació del *fog*.

El mateix amb el districte d'Extramurs. Amb els barris d'El Botànic, La Roqueta, La Petxina i Arrancapins, es necessiten un total de 5 dispositius, 4 sensors i un node central, per tal de modelar el districte. Sumant els 5 dispositius d'aquest districte més els 3 del districte de Benimaclet, es necessiten un total de 8 dispositius per tal de desplegar el projecte, un nombre de dispositius que sí és accessible.

Amb un total de 8 dispositius es pot estar monitoritzant informació sobre els dos *fogs* i observant els canvis de funcionament del node central. No obstant, i per tal de afegir més realisme al prototip, es proposa l'addició de dos dispositius extra. Aquests dispositius són dispositius que fan funció d'actuador, i s'afegeixen un a cada fog. D'aquesta manera, quedaria un total de 4 dispositius per al *fog* de Benimaclet i un total de 6 dispositius per al *fog* d'Extramurs. En total, tot el prototip està format per uns 10 dispositius, que segueix siguent un nombre accessible de dispositius però afegeix un increment significatiu en la representativitat del prototip.

7.2.1. Benimaclet *fog*

Tal i com s'ha comentat, per tal de modelar el districte de Benimaclet s'utilitzen 4 dispositius, dos sensors, un node central i un actuador. La distribució d'aquests dispositius s'ha determinat seguint els aspectes comentats anteriorment i a més, intentant obtenir una ubicació el més cèntrica possible per al node central, de

tal manera que si es decidira afegir més dispositius al *fog*, la distància d'aquests respecte al node central fóra el més equidistant possible.

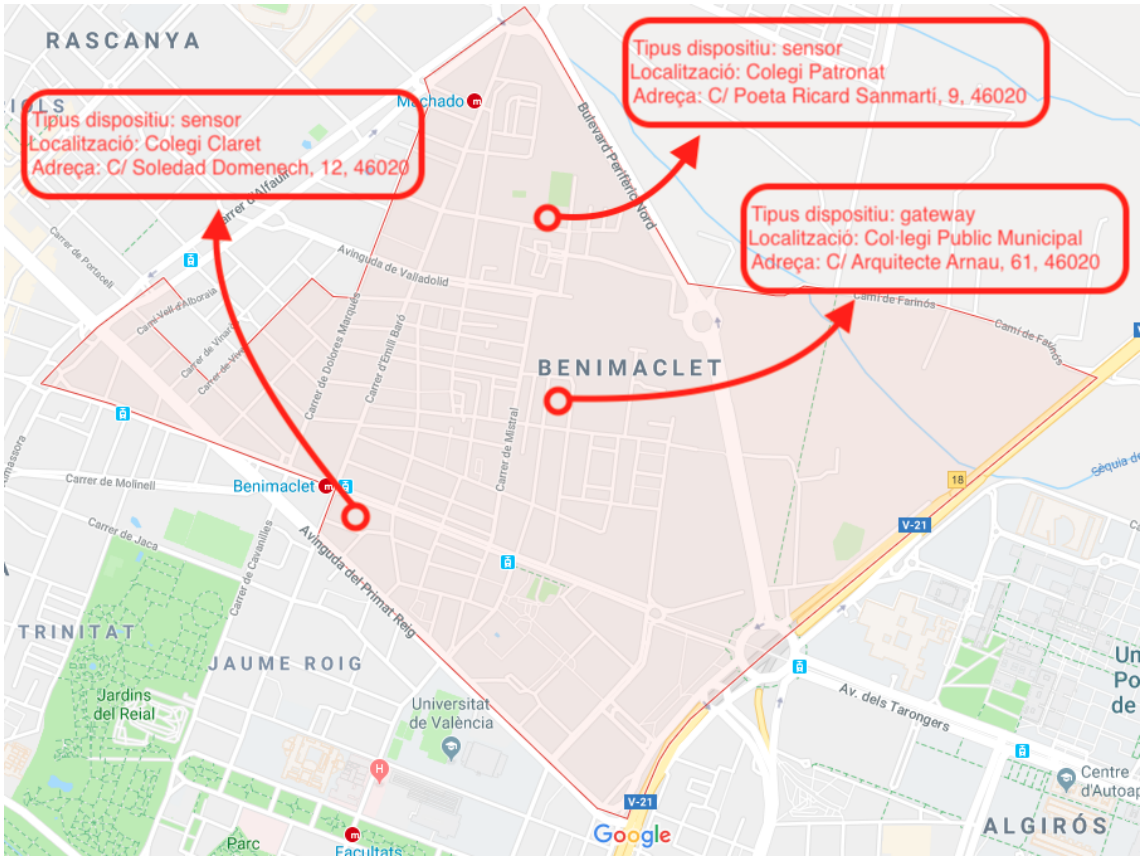

Figura 7.3: Distribució dels diferents dispositius que formen el Benimaclet *fog*.

A la figura 7.3 s'observa la distribució dels diferents sensors i el node central sobre el mapa del districte de Benimaclet. Com es pot veure, el node central del *fog* se situa en una posició bastant cèntrica, pels motius que ja s'han comentat, i cadascun dels sensors entren dins de l'abast del barri al que han sigut assignats. Encara que no apareix en aquesta figura, el controlador semafòric se situa en una posició d'interés per al trànsit de la zona.

La informació completa dels diferents dispositius que formen el *fog*, representats a la figura 7.3, és:

- Dispositiu *gw01*

  Tipus: Node central o *gateway*

  Coordenades: 39.4864589, -0.3567447

  Localització: Col·legi Públic Municipal. Arquitecte Arnau, 46020

- Dispositiu *co2s01*

  Tipus: Sensor (tipus CO_2)

  Coordenades: 39.2382409, -0.7201455

  Localització: Col·legi Claret. Soledad Domenech, 46020

- Dispositiu *co2s02*

  Tipus: Sensor (tipus CO_2)

  Coordenades: 39.4900201, -0.3591817

  Localització: Colegi Patronat. Poeta Ricard Sanmartí, 46020

- Dispositiu *tl01*

  Tipus: Actuador (tipus controlador semafòric)

  Coordenades: 39.490579, -0.361583

  Localització: Ronda Nord amb Carrer d'Alfauir

7.2.2. Extramurs *fog*

El districte d'Extramurs, al ser més extens que el districte de Benimaclet, està format per un major nombre de barris. En aquest cas, un total de 5 dispositius són necessaris per tal de modelar-lo. La distribució dels dispositius que formen part d'aquest *fog* segueix els mateixos aspectes que s'han comentat en el cas de la distribució del barri de Benimaclet. La informació dels 5 dispositius que formen part d'aquest *fog* és la següent.

La figura 7.4 representa la distribució dels sensors i el node central sobre el districte d'Extramurs. A l'igual que en el cas del *fog* de Benimaclet, en aquest cas s'ha situat la *gateway* també en una posició bastant centrada del districte per tal d'equidistar-la dels extrems del districte. Els sensors estan distribuits seguint la directiva de distribuir un sensor per cada barri del districte i, encara que no apareix a la figura, el controlador semafòric que fa funció d'actuador se situa també, a l'igual que en el cas de Benimaclet, en una posició de gran interés.

La informació detallada dels diferents dispositius que formen el *fog* i que s'han distribuït com es mostra a la figura 7.4 és:

- Dispositiu *gw02*

  Tipus: Node central

  Coordenades: 39.4676921, -0.3894919

  Localització: IES Abastos. Carrer d'Alberic. 46008

- Dispositiu *co2s03*

  Tipus: Sensor (tipus CO_2)

  Coordenades: 39.4732884, -0.3920861

  Localització: Centre d'Especialitats Joan Llorenç. Carrer de Joan Llorenç, 46008

- Dispositiu *co2s04*

  Tipus: Sensor (tipus CO_2)

  Coordenades: 39.4756939, -0.3892853

  Localització: Jardí Botànic. Beat Gaspar Bono, 46008

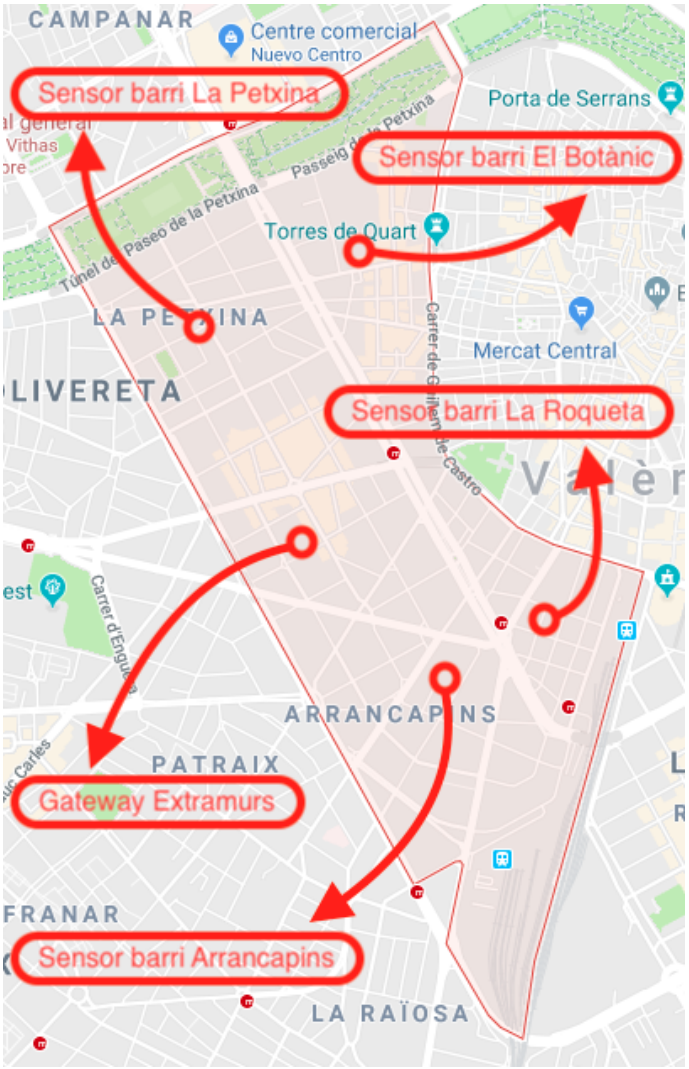

Figura 7.4: Distribució dels diferents dispositius que formen l'Extramurs *fog*.

- Dispositiu *co2s05*

  Tipus: Sensor (tipus CO_2)

  Coordenades: 39.4664949, -0.3812494

  Localització: Institut Valencià d'Oncologia. Carrer de l'estrela, 46007

- Dispositiu *co2s06*

  Tipus: Sensor (tipus CO_2)

  Coordenades: 39.465088, -0.3836757

  Localització: Oficina de recaudació de multes. Carrer d'Albacete, 46007

- Dispositiu *tl02*

  Tipus: Actuador (tipus controlador semafòric)

  Coordenades: 39.4905786, -0.3615826

  Localització: Pont de les glòries Valencianes

7.3 Exemple funcional

En aquest apartat es presenta un exemple del funcionament real del projecte desenvolupat. S'ha dividit en subapartats que tracten, des de la derivació de claus entre els dispositius fins a les modificacions de l'estat del *fog*, passant per la producció, xifrat i desxifrat de dades. S'acompanya les explicacions escrites de captures de pantalla i figures per tal d'aclarir aquestes i demostrar el propi funcionament dels components.

7.3.1. Derivació de claus simètriques

El primer aspecte que s'aborda és la derivació de les claus compartides (claus simètriques) que deriven els dispositius entre ells. La derivació d'aquesta clau és el resultat de l'execució d'un protocol de *handshake*, que ja s'ha explicat anteriorment, i és utilitzada per al xifrat i desxifrat de dades posteriorment.

A les figures 7.5 i 7.6 s'observa com dos dispositius, una *gateway*, en el cas de la figura 7.5, i un sensor, en el de la figura 7.6, deriven la mateixa clau simètrica després del procés de *handshake*. Es pot observar que ambdós dispositius relacionen a l'altre amb el secret que apareix a les figures.

Açò és possible, encara que no s'ha comentat en aquest apartat, a que els dos disposen del mateix *token* i estan comunicant-se amb el mateix servidor RabbitMQ. És a dir, és possible perquè els dos dispositius han estat configurats per a pertanyer al mateix *fog*. A partir del moment inmediatament posterior a les fi-

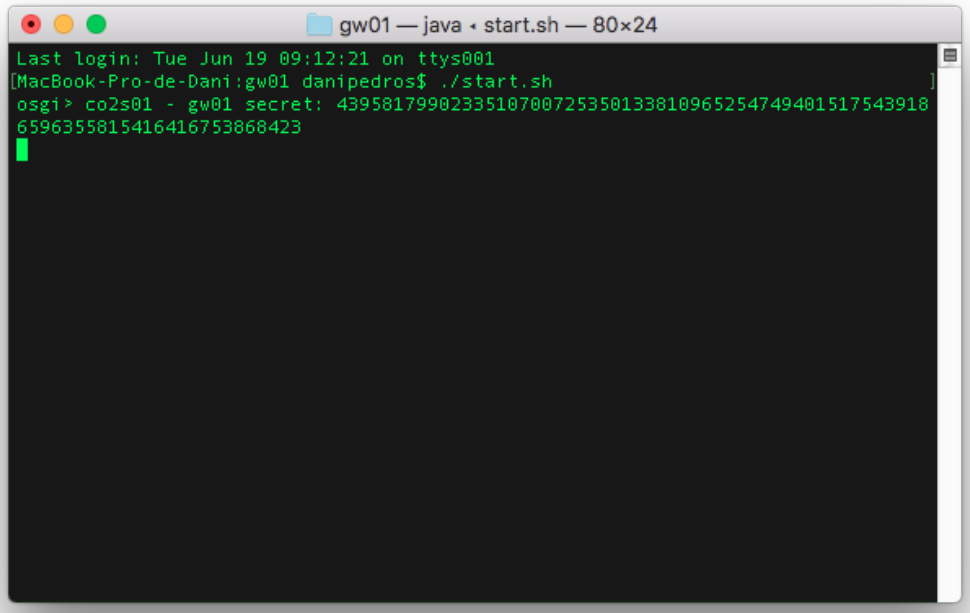

Figura 7.5: Derivació de la clau compartida entre un sensor i un node central per part del node central.

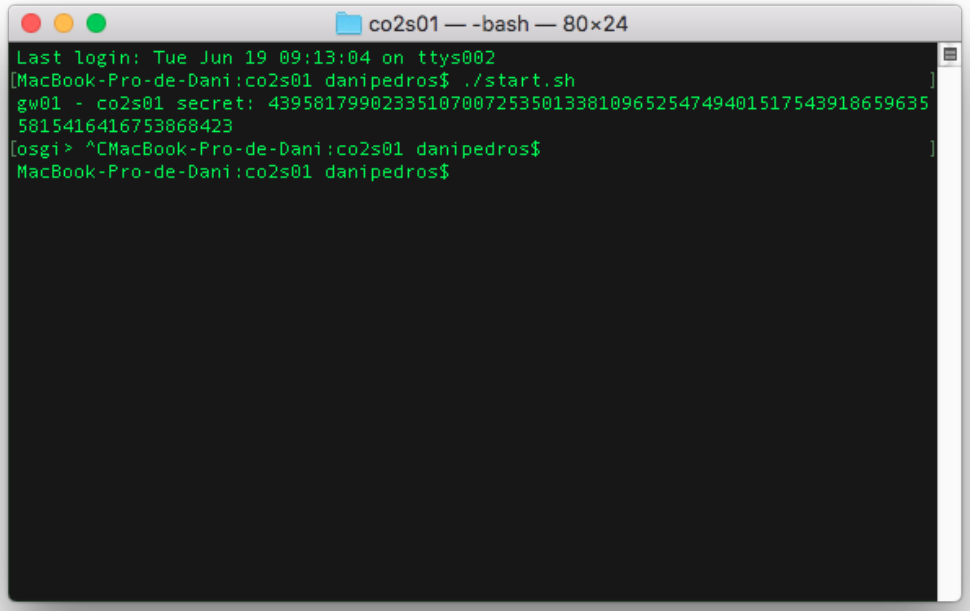

Figura 7.6: Derivació de la clau compartida entre un sensor i un node central per part del sensor.

gures, aquests dispositius començaran a xifrar i desxifrar les dades que s'envien entre ells amb la clau que han derivat, que apareix a les figures.

Així doncs, amb les dades que envien xifrades, qualsevol altre dispositiu que forme part del *fog* no va a poder llegir ni modificar les dades. Únicament els dos dispositius implicats en la comunicació són capaços de desxifrar les dades. Encara que tots disposen del mateix *token*, en el procés de *handshake* es deriven claus diferents, com s'ha explicat en anteriors apartats.

7.3.2. Producció i consumició de missatges

En aquesta secció es mostra un exemple de com es produeixen i consumeixen dades dins del *fog* per part dels dispositius. L'exemple consisteix en un sensor, una *gateway* i un actuador. El sensor produeix mesures fictícies amb la finalitat de forçar a la *gateway* a canviar el mode de funcionament. Una vegada aquesta canvia de mode, notifica als actuadors què és el que han de fer.

A les figures 7.7 i 7.8 es mostra com es generen i es consumeixen dades per part dels dispositius dels sistema. S'ha decidit mostrar la generació de dades per part d'un sensor (figura 7.7) i el consum de dades per part d'un actuador (figura 7.8) perquè són els exemples més senzills. De la mateixa manera s'hauria pogut mostrar la generació o consum de dades per part d'un node central, però, com que aquest suporta un gran trànsit de dades, hauria sigut més complicat identificar un únic missatge.

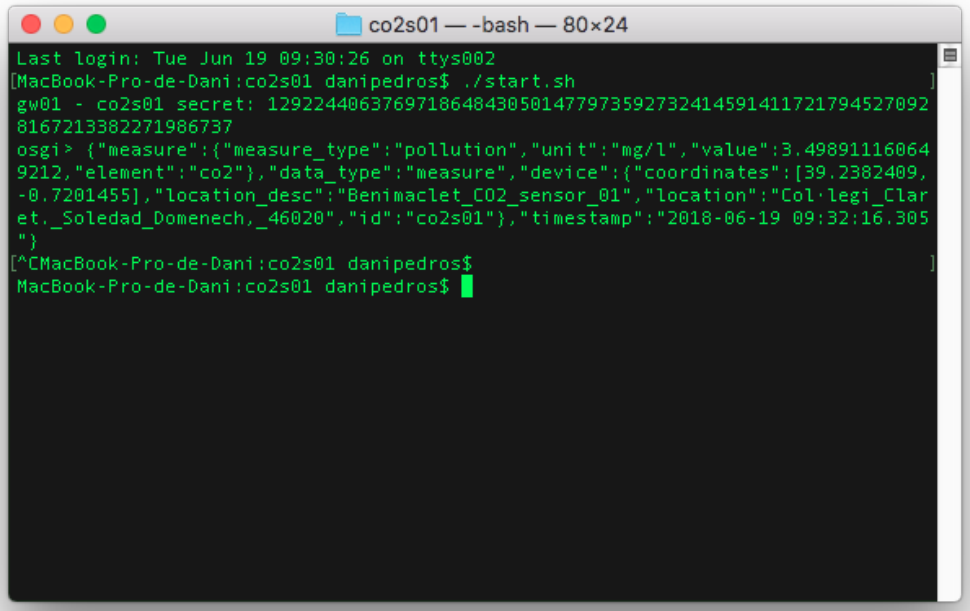

Figura 7.7: Generació d'una mesura per part d'un sensor.

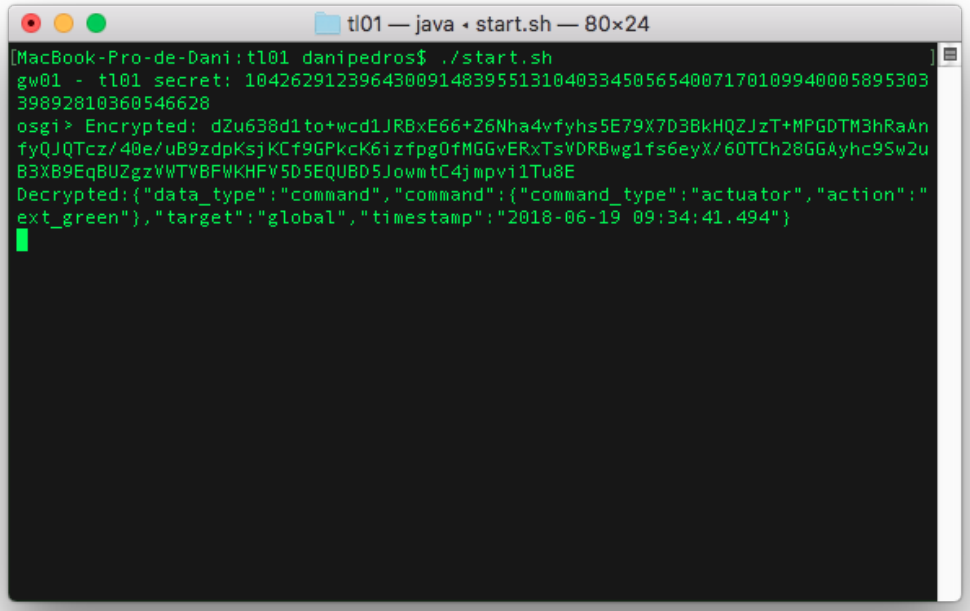

Figura 7.8: Consumició d'un comandament per part d'un actuador.

En el cas de la figura 7.7 s'observa el camp de les dades d'un missatge que envia un sensor. Aquestes dades estan en format JSON i es poden entendre fàcilment. Entre altres coses, es pot veure l'identificador del dispositiu que genera les dades (co2s01), quin tipus de dades és (measure), el valor de la mesura (3.4989111...) o la data i hora de generació (2018-06-19 09:32:16).

La figura 7.8 mostra la recepció d'un comandament en un actuador. A l'igual que en el cas de la mesura, aquest camp de dades també està en format JSON i es pot entendre. La *gateway* està ordenant a l'actuador l'execució del comandament extgreen en el moment 2018-06-19 09:34:41.

7.3.3. Xifrat i desxifrat de dades

En aquest apartat es presenta el resultat de xifrar i desxifrar el camp de dades. Es mostra com un sensor genera una mesura i de quina manera la rep el node central. A més, es mostra també quin és el resultat de desxifrar el que rep el node central. Aquest procés és posterior a l'establiment de la clau simètrica entre els dos dispositius amb el protocol de *handshake*.

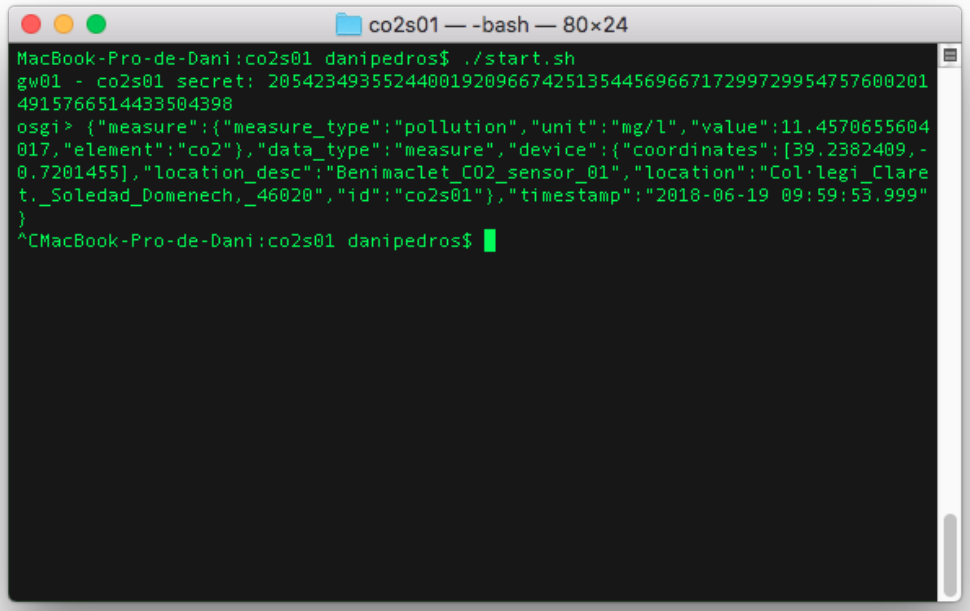

Figura 7.9: Mesura original generada per un sensor.

A la figura 7.9 s'observa el camp de dades d'una mesura generada per un sensor. Aquest, abans d'enviar les dades a la *gateway* les xifra utilitzant la clau simètrica que ha derivat amb aquesta. Una vegada xifrades les dades, les envia. La *gateway* rep el que s'observa a la figura 7.10 després de l'etiqueta *Encrypted*. Aquestes dades, a més de xifrades estan codificades en Base64. No obstant, es pot observar, a continuació de l'etiqueta *Decrypted* el resultat de la decodificació i desxifrat.

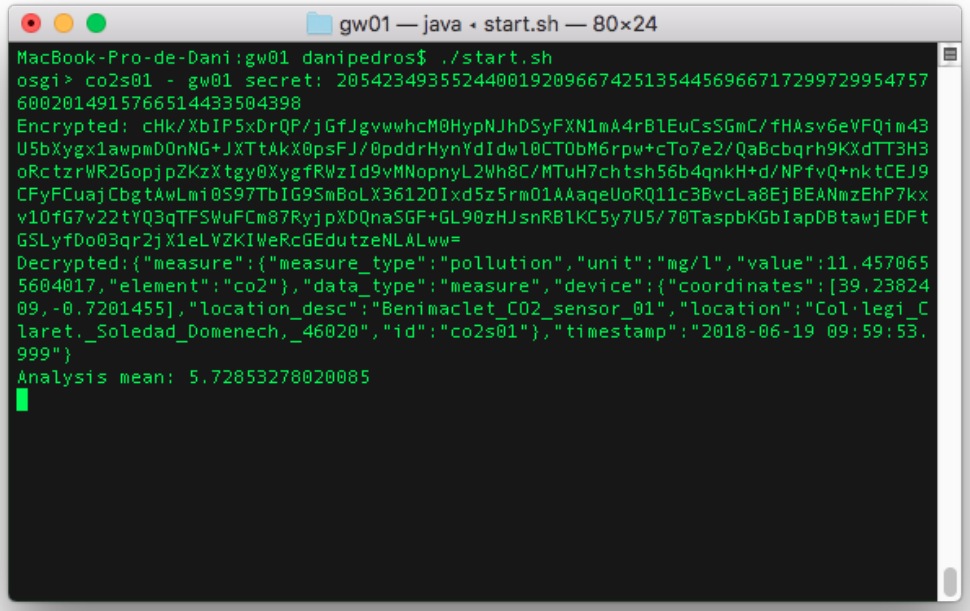

Figura 7.10: Dades xifrades rebudes i desxifrades per un node central.

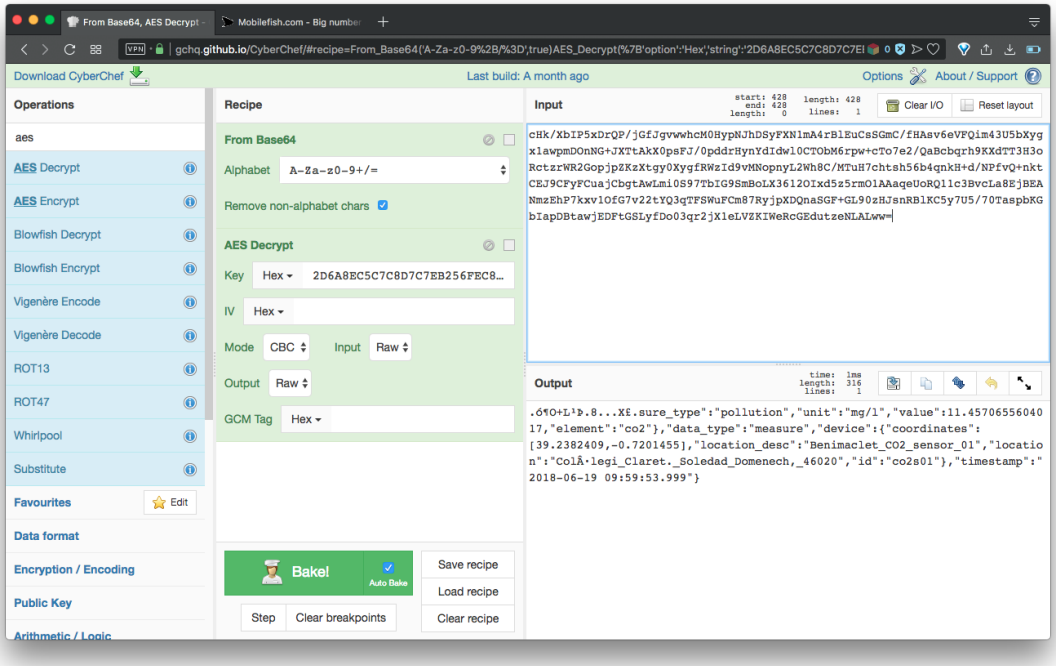

Figura 7.11: Desxifrat manual de les dades.

Directament a les figures 7.9 i 7.10 ja es pot veure que les dades originals i les que resulten del desxifrat són les mateixes. No obstant, i per a que no quede ningun dubte de que les dades es xifren i es dexifren utilitzant la clau derivada

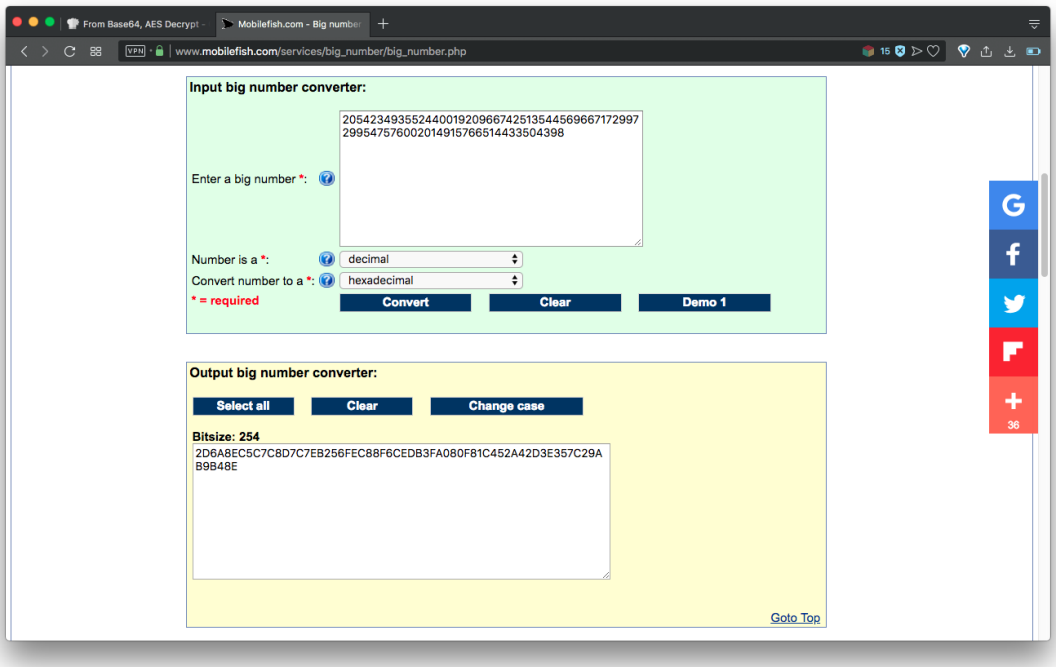

Figura 7.12: Conversió de decimal a hexadecimal de la clau simètrica.

del protocol de *handshake*, a la figura 7.11 es mostra el procés manual de desxifrat de les dades.

A la figura 7.11 s'observa que l'*input* és el contingut que rep el node central a la figura 7.10 sense decodificar ni desxifrat. A l'esquerre, a la columna *Recipe*, en primer lloc es decodifica el contingut d'*input* de Base64 a format *raw* i posteriorment es desxifra mitjançant l'algoritme AES. La clau que s'ha utilitzat no és més que la conversió de decimal a hexadecimal de la clau simètrica que han derivat ambdos dispositius.

A la figura 7.12 s'observa la conversió de la clau i es pot comprovar que és la mateixa que deriven els dispositius a les figures 7.9 i 7.10. Utilitzant el resultat d'aquesta conversió s'aconsegueix desxifrar les dades originals, com es pot veure al camp *output* de la figura 7.11. Només cal comentar que els primers 16 caracters (128 bits, que és la longitut de bloc d'AES) no s'han desxifrat correctament de manera manual perquè no es disposa del vector d'inicialització (IV), però els dispositius sí disposen d'ell.

7.3.4. Modificacions de l'estat del *fog*

Per últim, es vol exemplificar també com, a partir de l'anàlisi de mesures generades per sensors, és possible canviar l'estat i el funcionament d'un *fog*. Encara que a l'apartat de generació i consumició de dades ja s'ha mostrat com un actuador consumeix comandaments provocats per un canvi de funcionament del *fog*, aquesta secció només se centra en aquests canvis a nivell de node central.

En aquest exemple hi ha en funcionament dos sensors generant mesures i un node central processant i analitzant aquestes mesures. Si es prenen en considera-

ció les dues últimes línies que s'observen a la figura 7.13 es veu que el *fog* té un valor mitjà de 19.63 i per tant està en estat *OPEN*.

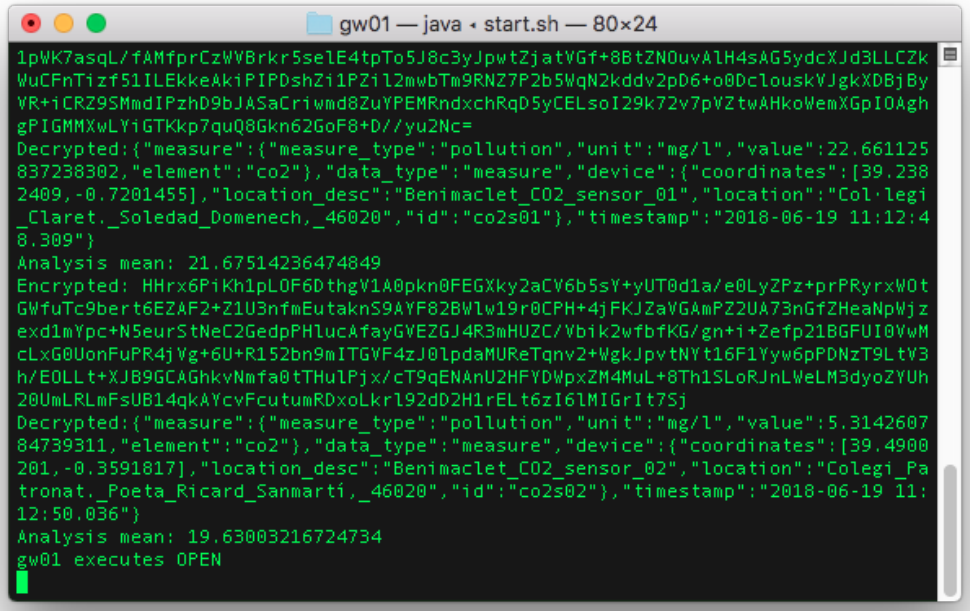

Figura 7.13: *Gateway* funcionant en mode OPEN.

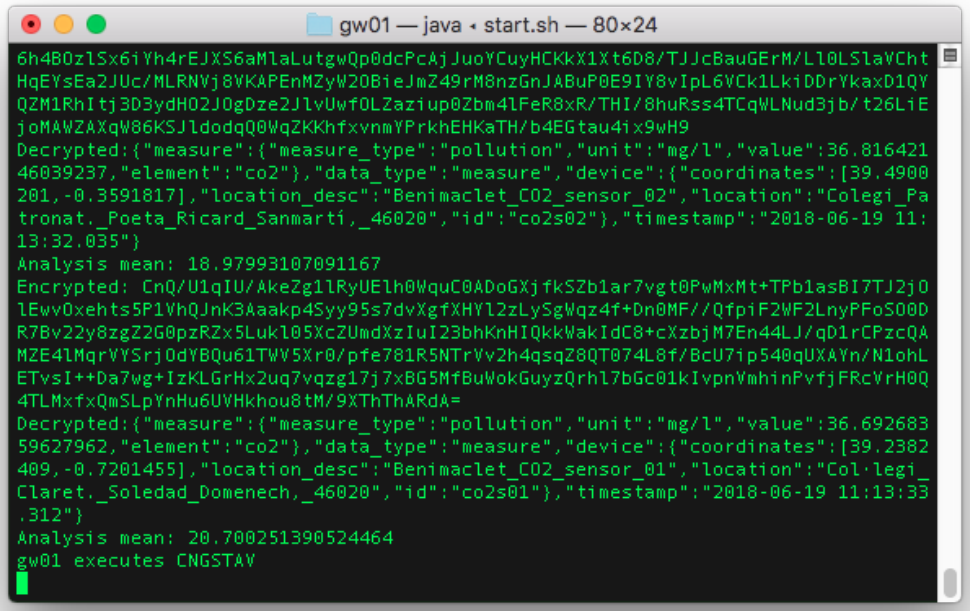

Figura 7.14: Canvi de funcionament d'una *gateway* d'OPEN a CNGSTAV.

A mesura que continuen arribant noves medicions, la *gateway* va realitzant anàlisis d'aquestes i arriba un moment en el que la mitjana puja per sobre de 20, concretament, aquesta arriba a un valor de 20.70. Segons està programat el *fog*, en cas de que la mitjana de les mesures sobrepasse un valor de 20, el node central ha d'aplicar determinades mesures de restricció. És a dir, en el moment que es supere un valor de 20, el *fog* canvia d'estat i aplica determinades restriccions.

A la figura 7.14, considerant les dues últimes línies també, es pot observar el valor de la mitjana, que és 20.70, com ja s'ha comentat, i la notificació de canvi d'estat d'execució. En aquest cas es mostra que passa a executar-se en estat *CNGSTAV*, que fa referència a un determinat estat que evita les congestions per tal de reduir la quantitat de CO_2 present a la zona.

Una vegada el *fog* canvia d'estat, la *gateway* s'encarrega de notificar les accions concretes que han de dur a terme els diferents actuadors del *fog*. L'exemple d'açò és el que ja s'ha vist a la figura 7.8, on, després del canvi d'estat del *fog*, el node central notifica a l'actuador vinculat que execute el comandament *extgreen*.

CAPÍTOL 8
Conclusions

Aquest capítol presenta les conclusions obtingudes del desenvolupament del projecte. Es tracten els objectius que s'han aconseguit assolir i quin ha sigut el procés per tal de fer-ho possible, així com també els que no s'han pogut assolir i els motius que no ho han permés. A més de tractar els objectius del projecte, també es proposa una visió de futur presentant possibles projectes que poden derivar de l'actual. I per tal de concloure aquest document, es presenta també la valoració personal sobre el projecte.

8.1 Objectius assolits

Prenint en consideració l'apartat d'objectius presentat a la introducció d'aquest document, es pot considerar que s'ha aconseguit assolir l'objectiu principal de desenvolupar un sistema aplicable a l'àmbit de les ciutats intel·ligents que permeta la computació distribuïda de manera segura en una xarxa de dispositius IoT. Encara que el sistema desenvolupat es tracta d'un prototip, és evident que seria possible desplegar-lo en una ciutat, que desenvolupa compotació distribuïda amb dispositius del món de l'IoT, aportant intel·ligència i adaptabilitat a la ciutat (ciutats intel·ligents) i que incorpora mesures de seguretat per tal de fer més segura aquesta intel·ligència.

Al paràgraf anterior es parla de l'objectiu general, però si es tracten els dos subobjectius plantejats a la introducció per separat, s'entén millor que han sigut assolits. En el cas de la computació distribuïda mitjançant una arquitectura *fog/edge computing*, l'exemple més clar de la distribució d'aquesta intel·ligència, o capacitat de còmput, és el fet d'organitzar el sistema en *fogs* independents per cada districte de la ciutat. Encara que tot forma part del mateix sistema, la intel·ligència d'aquest es reparteix entre diferents unitats intel·ligents, que són els *fogs*.

El segon subobjectiu, que tracta la seguretat de les comunicacions i del sistema, també es pot considerar assolit. La més clara demostració està en que, gràcies al protocol de *handshake* desenvolupat i al xifrat dels missatges, ningun dispositiu no autoritzat pot ni interceptar i interpretar comunicacions, ni establir noves comunicacions no legítimes. D'aquesta manera, s'evita la manipulació il·legítima del sistema i la exfiltració d'informació d'aquest.

8.2 Objectius no assolits

Encara que no es pot considerar que no han sigut assolits, els objectius d'afegir seguretat i escalabilitat al sistema són considerats processos, i per tant, no finalitzen mai. Per açò mateix s'inclouen en aquest apartat, ja que no es pot considerar que s'ha acabat d'afegir seguretat al sistema mai, ni tampoc que el sistema és 100% escalable, sinó que el nivell de seguretat i d'escalabilitat actual compleix amb els objectius establits.

Encara que ja existeix una capa de seguretat, i per tant l'objectiu ha sigut assolit, cal revisar i incrementar aquesta capa de seguretat progressivament. Un sistema que estableix mesures de seguretat i no les torna a revisar, passat un temps torna ser quasi igual de vulnerable que si no incorporara ninguna seguretat. Com a dissenyador del sistema, és important estar contínuament plantejant-se si el sistema segueix sent segur, i afegint més capes de seguretat de manera incremental i progressiva.

De la mateixa manera que passa amb la seguretat del sistema ocórre amb l'escalabilitat d'aquest. Augmentar l'escalabilitat és un procés continu i ha de revisar-se periòdicament. Encara que és veritat que s'ha complit l'objectiu i s'ha desenvolupat un sistema que és escalable, aquest sempre pot ser més escalable (o, al menys, més fàcilment escalable). Per exemple, encara que aquest projecte ja està desenvolupat basant-se en components que agrupen funcionalitats, seria possible especialitzar encara més aquestes funcionalitats, de tal manera que existiren més components més especialitzats. Açò disminuiria l'acoblament entre els diferents components, augmentant així l'escalabilitat.

8.3 Futurs projectes

Sobre futurs projectes relacionats amb el que s'ha presentat en aquest document, existeixen moltes possibilitats. Es poden proposar diferents tipus de projectes. Des de l'addició de funcionalitats a l'actual prototip fins a la realització de nous projectes sencers que adapten l'actual a diferents marcs o estandars.

Sobre el primer exemple comentat, pot resultar interessant l'addició de funcionalitats al sistema per tal de convertir-lo en un sistema més adaptable a l'entorn de la ciutat. Per exemple, es pot afegir el suport a altres tipus de mesures. A més de les mesures de CO_2, el sistema podria incorporar suport també per a mesures de CO, o de sulfurs, etc. D'aquesta manera, i gràcies a disposar de més informació, es podrien aplicar polítiques més complexes que s'adaptaren de millor manera a l'entorn de la ciutat.

I tractant el segon exemple, l'adaptació del sistema a marcs comuns o estàndars podria resultar en un nou projecte complet. Aprofitant guies sobre les ciutats intel·ligents avalades per grans companyies o organitzacions governamentals, es pot redissenyar el sistema per tal d'adaptar-lo a aquestes. Açò podria traduir-se en certificacions que augmentarien el ressò del projecte o la confiança que genera. Però no només existeix la possiblitat d'adaptar-se a guies existents, sinó que tam-

bé es poden crear guies noves a partir d'una anàlisi i investigació sobre el sistema d'aquest projecte i altres semblants.

8.4 Valoració personal

Per tal de concluir el present document, es presenta la valoració personal del projecte. Es comenten els coneixements aportats per les assignatures cursades durant el grau i que han sigut determinants per a la realització del projecte, així com les tecnologies descobertes durant el desenvolupament d'aquest, els errors comesos i les dificultats que s'han trobat i les coses apreses de la realitzacó del projecte.

Algunes assignatures cursades han aportat coneixements d'interés per a la realització del projecte. Entre aquestes assignatures destaquen l'assignatura de Criptografia (CRI), que ha sigut determinant per al coneixement dels fonaments matemàtics que s'amaguen darrere de l'algoritme *Diffie-Hellman*, i que han permés el desenvolupament del protocol de *handshake*. A més, Integració d'Aplicacions (IAP) ha marcat de manera significativa el desenvolupament de les comunicacions, ja que en eixa assignatura es van tractar els diferents tipus de comunicació exposats al present document, així com les implementacions d'aquestes comunicacions (cues de missatgeria, REST, etc.) i els formats de dades utilitzats. L'assignatura d'Enginyeria del Programari (ISW) també va aportar coneixements per al desenvolupament de programari, tractant aspectes com els diagrames d'anàlisi i de classes o els diferents paradigmes de programació. I per últim, cal comentar també l'assignatura de Gestió de Projectes (GPR), que va aportar els coneixements necessaris per a planificar temporalment el projecte (diagrames de Gantt, etc.).

Aquest projecte també m'ha aportat coneixements sobre tecnologies que no s'han utilitzat durant el transcurs del grau. Exemples en són tecnologies com OSGi, sobre la qual he aprés que existeix la possibilitat de desenvolupar components autònoms i que poden integrar-se entre ells, fins i tot dinàmicament utilitzant les ferramentes que OSGi proporciona. Altres tecnologies que no s'han tractat durant la formació Universitària han sigut els dispositius Raspberry Pi o Arduino, sobre els quals he treballat i he realitzat proves per motivació personal, ja que em resultaven molt interessants ambdues tecnologies. I també és digna de comentar la tecnologia Docker, sobre la qual s'havia vist un poc a la Universitat, però molt superficialment. Durant el projecte he aprofundit en el coneixement de la ferramenta i la utilització d'aquesta.

Com en qualsevol projecte, sempre apareixen dificultats no esperades i també és inevitable cometre errors. Per exemple, una dificultat que vaig tenir en el desenvolupament del projecte per culpa d'un error comés va ser la utilització d'OSGi. Encara que no deuria haver sigut una dificultat més gran del que suposa treballar amb una tecnologia nova, l'error que vaig cometre va ser començar a treballar amb la tecnologia sense haver pres el temps necessari en coneixer-la i entendre-la internament. I altra dificultat que vaig trobar en el desenvolupament del projecte va ser la depuració de tots els errors derivats de la importació de llibreries externes. Aquests tipus d'errors em van pareixer incomprensibles i em

van suposar, en alguns moments, fins i tot una frustració. No obstant, de tots els errors i dificultats es desprén coneixement i millora.

A més de l'aprenentatge derivat dels errors comesos, el projecte també m'ha aportat nous coneixements que podré aplicar en un futur. Entre altres coses, el més important que he aprés ha sigut a treballar amb la programació orientada a components, a utilitzar conceptes matemàtics purs de criptografia per a desenvolupar protocols que s'adapten a les necessitats requerides pels projectes, la possibilitat de realitzar desplegaments distribuïts de solucions amb ferramentes com Docker i la capacitat d'organitzar i gestionar un projecte.

La realització d'aquest projecte m'ha aportat moltíssim com a professional de les tecnologies de la informació. Però no només m'ha servit per a creixer com a professional, sinó que també ha sigut útil per a creixer com a persona. Aprendre a gestionar el temps i la frustració és necessari per a la vida personal i professional. A més, la realització d'aquest projecte m'ha despertat l'interés per seguir descobrint el món de la informàtica, de l'electrònica i de la tecnologia en general, així com la criptografia i les matemàtiques. Estic molt content i orgullós del projecte, dels resultats d'aquests i de tot el que m'ha aportat.

Bibliografia

[1] Article sobre el creixement del nombre de dispositius IoT en el futur proper, publicat a NetworkWorld el 13 de juny de 2018. Consultat a https://goo.gl/2SqUVo.

[2] Article sobre les millors ciutats intel·ligents del món, publicat a SmartCitiesWorld el 2 de maig de 2018. Consultat a https://www.smartcitiesworld.net/connectivity/connectivity/singapore-tops-the-smart-city-rankings.

[3] La revolució de l'IoT: Reptes i oportunitats, publicat a GenevaBussinesNews l'11 de juliol de 2016. Consultat a https://www.gbnews.ch/the-iot-revolution/.

[4] Larry L. Constantine, Lucy A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison Wesley, primera edició, abril 1999.

[5] Documentació sobre la història d'Eclipse i d'Eclipse Foundation. Consultat a http://www.eclipse.org/org/#history.

[6] Documentació sobre OSGi. Consultat a https://www.osgi.org/about-us/.

[7] Documentació sobre RabbitMQ. Consultat a http://www.rabbitmq.com/documentation.html.

[8] Informació sobre l'abast de les transmissions de la tecnologia LoRa. Consultat a https://www.postscapes.com/long-range-wireless-iot-protocol-lora/.

[9] Anàlisi sobre el protocol LoRa PHY. Consultat a https://pubs.gnuradio.org/index.php/grcon/article/download/8/7.

[10] Diferents sistemes operatius disponibles per a la Raspberry Pi. Consultat a https://www.raspberrypi.org/downloads/.

[11] Conceptes i primers passos en Docker. Consultat a https://docs.docker.com/get-started/.

[12] Definició i informació rellevant sobre la computació en el *fog*. Consultat a https://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing.

[13] Douglas R. Stinson *Cryptography: Theory and Practice*. Chapman and Hall/CRC, tercera edició, 2006.

[14] Daemen, Joan, Rijmen, Vincent. *The Design of Rijndael: AES -The Advanced Encryption Standard*. Springer, 2002.

[15] A. J. Menezes, P. C. van Oorshot and S. A. Vanstone. *Handbook of applied crytography*. CRC Press, 1996.

[16] Roy Thomas Fielding *Architectural Styles and the Design of Network-based Software Architectures (Ph.D.)*, capítol 6. Universitat de California, Irvine.

[17] Arquitectura basada en publicadors/subscriptors i MQTT. Consultat a https://www.iso.org/standard/69466.html.

[18] Estudi relacionat amb el projecte actual. Consultat a https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1182&context=pacis2015.

APÈNDIX A
Format de les dades

Aquest apartat mostra els diferents tipus de dades que s'utilitzen al sistema en els dos formats suportats per aquest. Per cadascun dels tipus de dades comentats al capítol d'anàlisi del problema, el capítol 4, es mostra la seua representació en els dos formats comentats al capítol 5. En primer lloc es mostren les dades en format JSON i després en format XML.

A.1 Dades en format JSON

Commandament

```
{
  "data\_type": "command",
  "target": "",
  "command": {
    "command\_type": "",
    "action": ""
  },
  "timestamp": ""
}
```

Handshake

```
{
  "data\_type": "handshake",
  "phase": "",
  "id": "",
  "device\_type": "",
  "mod": "",
  "public\_key": "",
  "timestamp": ""
}
```

Mesura

```
{
  "data\_type": "measure",
  "device": {
    "id": "",
    "coordinates": [],
    "location": "",
    "location\_desc": "",
  },
  "measure": {
    "measure\_type": "",
    "value": 0.0,
    "unit": "",
    "element": ""
  },
  "timestamp": ""
}
```

Notificació

```
{
  "data\_type": "notification",
  "device": {
    "id": "",
    "coordinates": [],
    "location": "",
    "location\_desc": "",
  },
  "command": {
    "command\_type": "",
    "action": ""
  },
  "timestamp": ""
}
```

Política

```
{
  "data\_type": "policy",
  "target": "",
  "policy": {
    "policy\_type": "",
    "policy\_id": "",
    "command": {
      "command\_type": "",
      "action": ""
    },
    "duration": 0
  },
  "timestamp": ""
}
```

A.2 Dades en format XML

Comandament

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <data\_type>command</data\_type>
  <target></target>
  <command>
    <command\_type></command\_type>
    <action></action>
  </command>
  <timestamp></timestamp>
</data>
```

Handshake

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <data\_type>handshake</data\_type>
  <phase></phase>
  <id></id>
  <device\_type></device\_type>
  <mod></mod>
  <public\_key></public\_key>
  <timestamp></timestamp>
</data>
```

Mesura

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <data\_type>measure</data\_type>
  <device>
    <id></id>
    <coordinates>
      <latitud></latitud>
      <longitud></longitud>
    </coordinates>
    <location></location>
    <location\_desc></location\_desc>
  </device>
  <measure>
    <measure\_type></measure\_type>
    <value></value>
    <unit></unit>
    <element></element>
  </measure>
  <timestamp></timestamp>
</data>
```

Notificació

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <data\_type>notification</data\_type>
  <device>
    <id></id>
    <coordinates>
      <latitud></latitud>
      <longitud></longitud>
    </coordinates>
    <location></location>
    <location\_desc></location\_desc>
  </device>
  <command>
    <command\_type></command\_type>
    <action></action>
  </command>
  <timestamp></timestamp>
</data>
```

Política

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <data_type>policy</data_type>
  <target></target>
  <policy>
    <policy_type></policy_type>
    <policy_id></policy_id>
    <command>
      <command_type></command_type>
      <action></action>
    </command>
    <duration></duration>
  </policy>
  <timestamp></timestamp>
</data>
```

APÈNDIX B
Empaquetament dels components

En aquest apartat es presenta el procés d'empaquetament dels components per tal de poder desplegar-los sobre els sistemes finals. Es tracta, des de com s'estructuren els directoris que formen aquests components fins a les configuracions necessaries dels diferents *plugins* i components.

B.1 Estructura dels directoris i arxius

A la figura B.1 es presenta l'estructura de directoris que formen un component empaquetat. En color blau es mostren els directoris, en color taronja els arxius binaris, en aquest cas arxius JAR, i en color verd els arxius que es poden llegir en text pla. Les fletxes, com és evident, indiquen quins arxius i directoris estan continguts dins de cada directori pare.

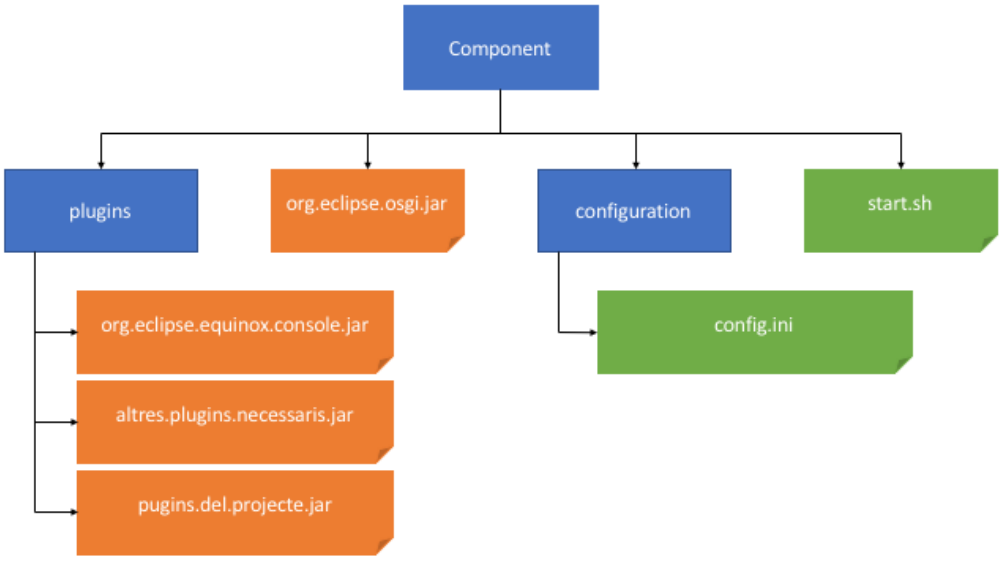

Figura B.1: Arbre de directoris i arxius que formen part d'un component empaquetat.

El directori etiquetat amb el nom *Component* és el directori pare. Aquest directori conté tot el necessari per al desplegament del component. És a dir, aquest

directori es pot considerar el paquet. El primer element contés és un directori anomenat *plugins* i que conté tots els arxius JAR necessaris per a satisfer les dependències del component. Aquestes dependències han de ser explicitades a l'arxiu *config.ini*, que està contés dins del directori *configuration*. Sobre aquest arxiu es parla a l'apartat sobre la configuració de les dependències.

El següent element que es troba dins de la carpeta que empaqueta el component és l'arxiu JAR *org.eclipse.osgi.jar*, que és l'arxiu executable que s'encarrega de posar en funcionament tot el component. Aquest executable consulta les dependències explicitades a l'arxiu *config.ini* i s'encarrega de posar en marxa els *plugins* necessaris. Per tal d'executar aquest binari és necessari l'últim element que conté el directori *Component*. Es tracta de l'arxiu *start.sh*, que és un *script* i s'encarrega de configurar la màquina virtual de Java i executar l'arxiu *org.eclipse.osgi.jar*. A l'apartat sobre la configuració dels component es parla d'aquest *script*.

B.2 Configuració de dependències

Com ja s'ha comentat, és necessari explicitar les dependències dels components per tal de que OSGi puga posar en marxa tot el necessari per a l'execució del component empaquetat. En primer lloc, cal identificar totes les dependències dels diferents components. Açò es pot fer mitjançant Eclipse. Tal i com es mostra a la figura B.2, quan es consulten les *Run configurations* d'Eclipse utilitzades per a l'execució dels components, apareixen especificats els diferents *plugins* requerits. Els arxius executables JAR que contenen aquests *plugins* han d'extraure's de la instal·lació d'Eclipse i guardar-se en la carpeta *plugins* comentada a l'estructura de directoris de l'apartat anterior, com es pot veure a la figura B.3.

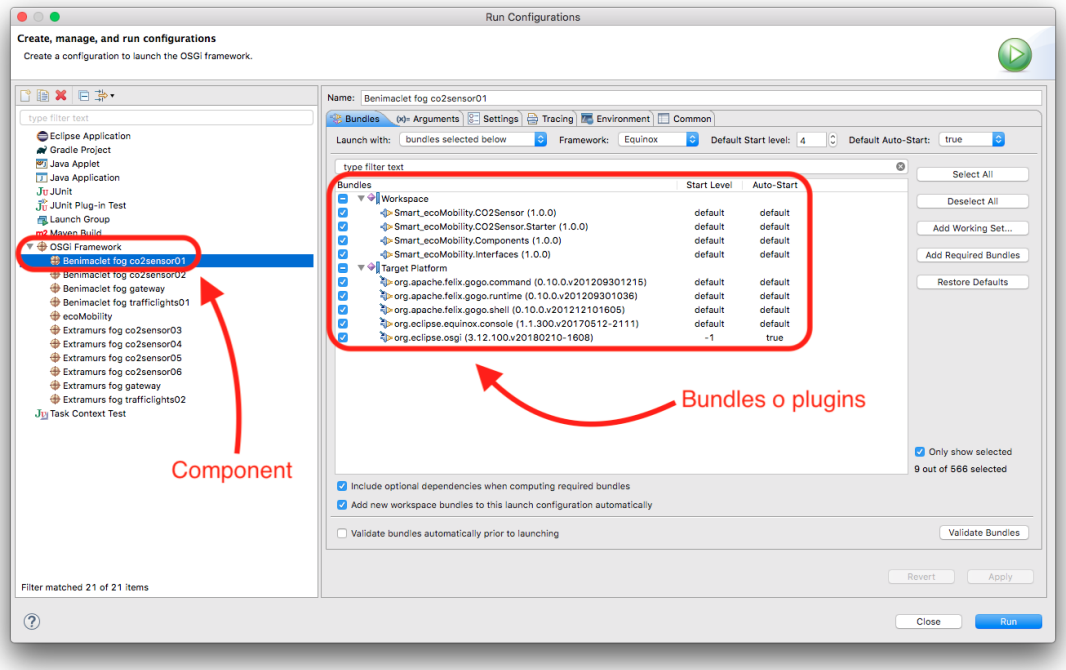

Figura B.2: Consulta de les dependències a la *Run configuration* d'Eclipse.

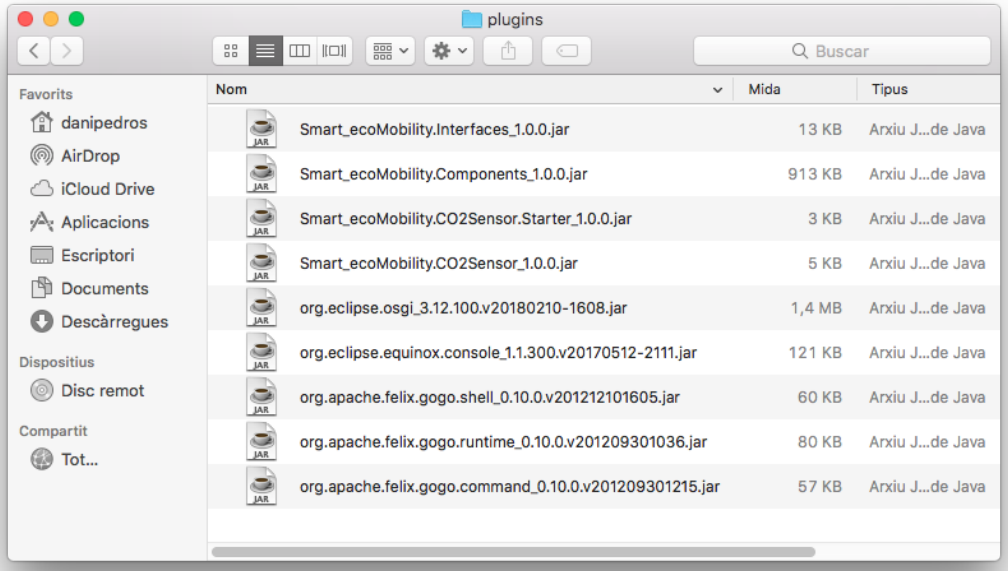

Figura B.3: Arxius JAR amb els *plugins* necessaris exportats.

Una vegada identificats i exportats tots els *plugins* necessaris s'ha d'indicar a OSGi la ruta fins als arxius JAR que contenen aquests *plugins*. Açò s'indica a l'arxiu *config.ini*, situat al directori *configuration*. En aquest arxiu s'indica la ruta relativa dels diferents arxius JAR des de la carpeta contenidora, és a dir, des de la carpeta etiquetada amb el nom *Component*. A la figura B.4 es mostra com queda configurat l'arxiu *config.ini* amb els *plugins* de les anteriors figures.

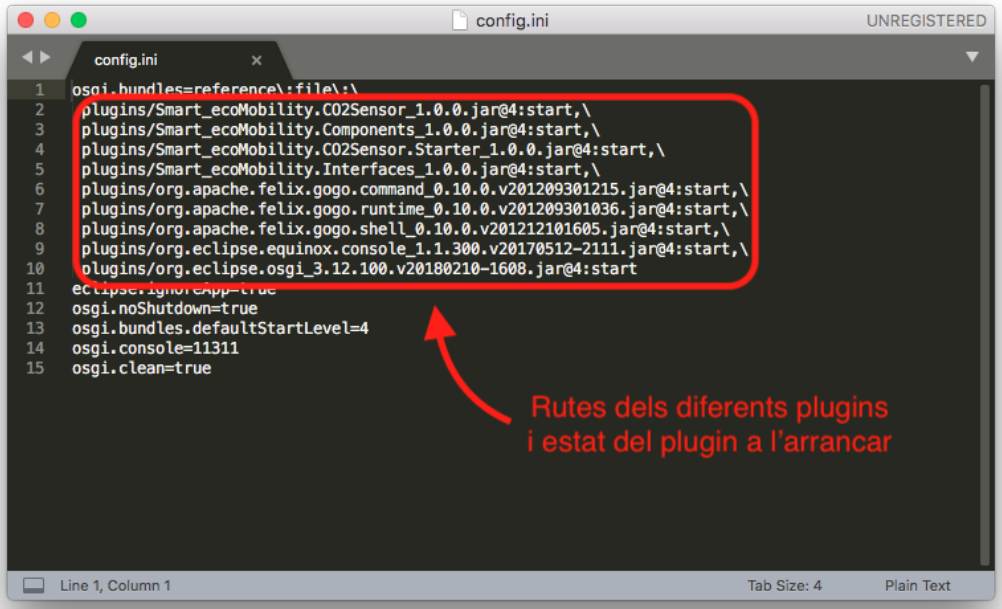

Figura B.4: Arxiu de configuració de les dependències dels *plugins* d'OSGi.

B.3 Configuració pròpia dels components

Una vegada realitzada la configuració de les dependències dels diferents *plugins* el component ja està preparat per a ser executat. Però és possible que el component també necessite configuració per a la seua pròpia lògica, com és el cas d'aquest projecte. En el cas d'aquest projecte, els components necessiten dades de configuració, com per exemple el identificador del component, un *token* per a poder establir comunicacions i autenticar-se, etc.

Aquesta configuració no és pròpia d'OSGi i s'ha de fer arribar a màquina virtual de Java, per tal de que aquestes dades de configuració siguen accessibles per les diferents parts del programari desenvolupat. Per tal de proporcionar aquestes dades de configuració dels components a l'executar-los s'utilitza un *script*, concretament es parla de l'*script* amb nom *start.sh* del qual s'ha parlat a l'apartat sobre l'estructura de directoris. A la figura B.5 es mostra el contingut d'aquest arxiu per tal d'exemplificar com s'estableixen aquests paràmetres de configuració i com se li fan arribar a la màquina virtual de Java en executar OSGi.

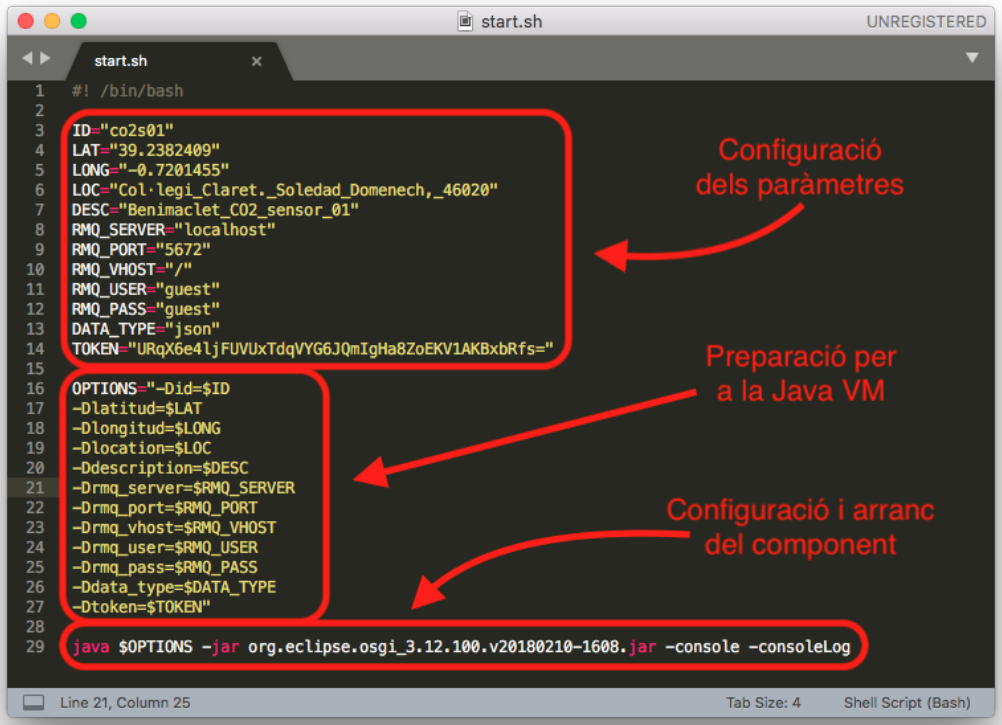

Figura B.5: *Script* de configuració i arranc del component.

APÈNDIX C
Desplegament de servidors RabbitMQ amb Docker

En aquest apartat s'explica el procés de desplegament dels servidors RabbitMQ. Com ja s'ha comentat, aquest desplegament es realitza mitjançant la ferramenta Docker. En primer lloc cal disposar d'una imatge modificada a partir de la imatge oficial disponible de RabbitMQ. Açò s'aconsegueix mitjançant un Dockerfile, el qual es pot veure a la figura C.1.

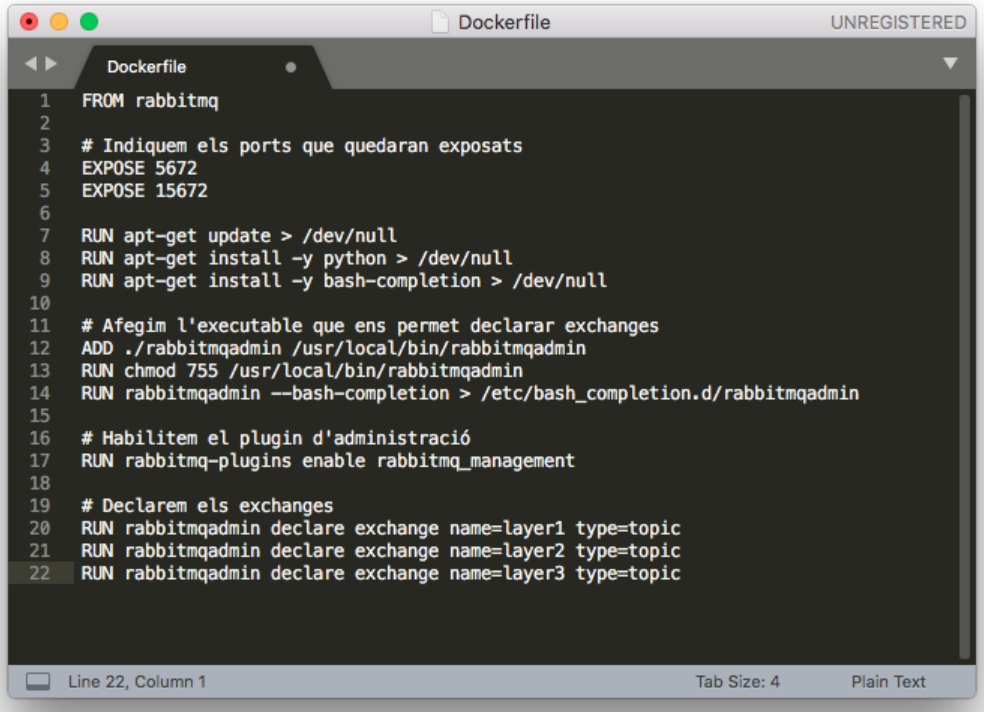

Figura C.1: Dockerfile que permet la construcció d'una imatge modificada de RabbitMQ.

Aquest Dockerfile permet construir una imatge de RabbitMQ que incorpore modificacions. Tal i com es pot veure a la figura C.1, en aquest cas, la nova imatge modificada incorpora una instal·lació de Python (línia 8), afegeix un *script*

programat en Python de nom *rabbitmqadmin* i l'executa (línies 12-14), habilita el *plugin* de gestió de RabbitMQ i declara tres *exchanges* de tipus *topic*. Gràcies a aquest Dockerfile, qualsevol contenidor que es cree a partir de la imatge resultat d'aquest Dockerfile tindrà incorporades de base aquestes modificacions que s'acaben de comentar.

simple-rmq-setup.sh

```
#! /bin/bash

# Definim els valors per defecte
SOURCE="${BASH_SOURCE[0]}"
DIR="$(dirname "$SOURCE")"
WARN='\033[0;33m'
NORMAL='\033[0m'

IMAGE="smart_ecomobility_rmq"

CONTAINER_BENIMACLET="rmq-benimaclet"
BENIMACLET_RMQSERVER_HOSTNAME="rmq-benimaclet"
BENIMACLET_RMQSERVER_PORT="5672"
BENIMACLET_RMQCONFIG_PORT="15672"

CONTAINER_EXTRAMURS="rmq-extramurs"
EXTRAMURS_RMQSERVER_HOSTNAME="rmq-extramurs"
EXTRAMURS_RMQSERVER_PORT="5673"
EXTRAMURS_RMQCONFIG_PORT="15673"

# Assegurem que la imatge de RabbitMQ modificada per a Smart ecoMobility està creada
IMG=$(docker image ls | grep "$IMAGE" | awk '{print $1}')
if [[ "$IMG" != "$IMAGE" ]]; then
    docker build -t "$IMAGE" "$DIR"
fi

# Comprovem si ja existeix el contenidor del servidor RabbitMQ de Benimaclet
CONT_BENI=$(docker ps -a | grep "$CONTAINER_BENIMACLET")
if [[ -z "$CONT_BENI" ]]; then
    docker run -d --hostname $BENIMACLET_RMQSERVER_HOSTNAME --name $CONTAINER_BENIMACLET -p $BENIMACLET_RMQSERVER_PORT:5672 -p $BENIMACLET_RMQCONFIG_PORT:15672 $IMAGE
else
    docker start $CONTAINER_BENIMACLET
fi

# Comprovem si ja existeix el contenidor del servidor RabbitMQ d'Extramurs
CONT_EXTRAM=$(docker ps -a | grep "$CONTAINER_EXTRAMURS")
if [[ -z "$CONT_EXTRAM" ]]; then
    docker run -d --hostname $EXTRAMURS_RMQSERVER_HOSTNAME --name $CONTAINER_EXTRAMURS -p $EXTRAMURS_RMQSERVER_PORT:5672 -p $EXTRAMURS_RMQCONFIG_PORT:15672 $IMAGE
else
    docker start $CONTAINER_EXTRAMURS
fi

exit 0
```

Figura C.2: *Script* de configuració i desplegament dels servidors RabbitMQ.

Una vegada es disposa d'una imatge preconfigurada de servidor RabbitMQ adaptada a les necessitats del projecte, és possible automatitzar el procés de posada en marxa dels diferents servidors. Per tal de realitzar aquesta tasca, es crea l'*script* de la figura C.2. Aquest *script* s'encarrega d'acabar de configurar els paràmetres dels servidors RabbitMQ que van a ser desplegats i de posar-los en funcionament.

Bàsicament, la funció de l'*script* és, comprovar si existeix una imatge de RabbitMQ preconfigurada i si no existeix, crear-la amb el Dockerfile comentat. Després, comprova si ja existeixen els contenidors completament configurats que s'han de desplegar. En cas de que afirmatiu, simplement arranca aquests contenidors. Però en cas de que no es detecten aquests contenidors, l'*script* s'encarrega d'acabar de configurar-los a partir de la imatge preconfigurada i dels paràmetres indicats, i de posar-los en funcionament.