**Caso di studio**

Con l’evolversi della tecnologia, ed in particolare con l’avvento dell’IoT,

l’aspettativa delle applicazioni basate sul cloud è cambiata. La necessità

di elaborare un numero sempre maggiore di dati ha spinto, sempre

più aziende, ad adottare strumenti cloud based forniti da terzi come ad

esempio Microsoft Azure o Amazon AWS. Ed è utilizzando questi servizi

che l’architettura classica del software monolitico ha mostrato i suoi limiti.

Con architettura monolitica si intende una applicazione self-contained

indipendente dalle altre applicazioni presenti nel sistema. Basandosi su

questa modello, ogni software necessita di essere disinstallato e reinstallato

nel momento in cui sia necessario apportare delle semplici modifiche.

Per superare i limiti del modello monolitico, è stato introdotto il

concetto di microservzio. La differenza tra una architettura monolitica

ed una basata su microservizi è la modularità che quest’ultima garantisce.

Prendiamo come esempio una applicazione formata da un database,

un interfaccia web (client-side user interface) e un’applicazione server

(server-side application). L’applicazione server-side interpreterà le

richieste fatte dall’utente andando ad eseguire operazioni interne al server,

aggiornerà il database e fornirà il risultato all’utente finale tramite

l’interfaccia web. Nel modello classico è necessario riscrivere o aggiornare

l’intero software per apportare delle modifiche o aggiungere delle

funzionalità.

Con il termine microservices si intende una architettura basata su oggetti

chiamati component ognuno dei quali fornisce ed utilizza dei servizi.

Per component, si intende una "parte" di software che può essere

aggiornata e sostituita indipendente dalla applicazione principale. Ogni

component si basa sull’utilizzo e la condivisione di servizi. I servizi sono l’equivalente delle librerie in una architettura monolitica, al contrario di

queste ultime però, i servizi sono sviluppabili indipendentemente dalla

applicazione principale. In questo capitolo si studierà l’implementazione

di applicativi per la gestione e l’interazione di sensori basti su Lo-

Ra tramite l’utilizzo del framework EveryWare Software Framework ESF

basato su OSGi e sviluppato da Eurotech.

**4.1 OSGi**

La tecnologia OSGi (Open Services Gateway initiative), è un insieme di

specifiche che definiscono dei componenti dinamici in grado di estendere

le funzionalità dell’ambiente Java. ~~Basandosi su di una~~ Prendendo in considerazione il funzionamento di un’architettura a microservizi, OSGi permette la gestione in modo remoto di componenti chiamati Bundle o Deployment Package(dp), i quali possono essere installati,

attivati, aggiornati e fermati senza la necessità di riavviare l’applicazione

principale

Ogni bundle è composto dai files ~~contenti~~ contenenti le classi ed i metodi specifici

per le funzionalità che implementa e dei metadati utili al framework per

determinare le classi private, ed utilizzabili solo all’interno dell’esecuzione

del modulo, da quelle condivisibili con gli altri componenti.

*(La parte sottolineata ha tre congiunzioni, però non so come correggerla senza cambiarne il senso, ti consiglio di spezzarla in due o di elencare la composizione del bundle.)*

La possibilità del riutilizzo di questi moduli permette di creare software di ridotta

complessità che sfruttano API di altri bundle per fornire nuove funzionalità

.

**4.1.1 Layer**

Per rendere compatibile questa architettura con il linguaggio di programmazione

Java e la Java Virtual Machine, OSGi ha strutturato il framework

su vari livelli:

• Bundles. Bundles sono normali applicazioni JAR contenenti

• Services. Il service layer connette in maniera dinamica i bundles

offrendo un modello publish-find-bind basto su l’interfaccia Java

POJIs.

• Services Registry. Questo layer è composto da una serie application

programming interface (API) per la gestione dei servizi

• Life-Cycle. Insieme di API per gestire il ciclo vitale dei bundle

• Modules. È il layer che definisce in che modo i bundle sono in grado

di interagire importando ed esportando pezzi di codice.

• Security È un layer che opera su tutti i livelli e garantisce la sicurezza

delle applicazioni OSGi andando a limitare le funzionalità del

bundle.

**4.1.2 Ciclo vitale dei bundle**

In un modello così dinamico, è necessario che il framework sia basato

su di un software fail-safe in grado di gestire le eccezioni che si possono

verificare durante l’installazione e disinstallazione di nuovi moduli. Non

è raro infatti che nuovi bundle ,installati nel sistema, richiedano servizi

non ancora disponibili o attualmente utilizzati da altri bundle. Per far

fronte ai problemi riscontrabili, OSGi, assegna ad ogni i bundle uno stato

tra:

• Installed Il bundle è stato installato nel sistema, ma non sono presenti

alcune delle sue dipendenze.

• Resolved Il bundle è installato e le sue dipendenze presenti nel

sistema.

• Starting Uno stadio temporaneo attraverso il quale il bundle passa

prima di essere attivato.

• Active Il bundle è stato correttamente attivato e sta eseguendo le

sue funzioni all’interno del sistema.

• Stopping Uno stadio temporaneo in cui il bundle passa prima di

essere disattivato.

• Uninstalled Il bundle è stato rimosso dal container OSGi

**4.1.3 Module layer**

Il module layer è dove il framework OSGi gestisce la "modularità" di un

bundle. È in questo layer che vengono processati i metadati contenuti

all’interno del file MANIFEST.MF. Tramite questo file il framework OSGi

è in grado di determinare le dipendenze del bundle e quali servizi è in

grado di esportare.

**4.1.4 Registrazione del servizio**

Un servizio in OSGi è definito tramite una classe Java standard. Per implementare

un nuovo servizio, è necessario definire quale classe o quale

interfaccia si vuole fornire il servizio. La soluzione a questa problematica ~~a questo,~~ è l’utilizzo

di registri di servizio. Ogni bundle può esporre dei metodi ~~ed~~ e registrarli

attraverso il service registry. In questo modo, altri bundle possono accedere

al service registry e utilizzare i metodi listati nel registro. Un bundle

quindi può registrare un servizio, può utilizzare un servizio oppure può

mettersi in attesa aspettando che un servizio venga registrato o eliminato.

Inoltre i servizi sono dinamici, ogni bundle può mettersi in lista per

richiedere un servizio mentre altri stanno ancora utilizzando il servizio.

**4.2 Everyware Software**

ESF o Everyware Software è un framework che si interpone tra il sistema

operativo e le applicazioni utente. Basato su Java/OSGi, ESF si pone

l’obiettivo di offrire la possibilità di sviluppare applicativi, per il mercato M2M ,in maniera semplice e veloce . Ideato per operare all’interno di gateway

industriali, fornisce al consumatore servizi e librerie per l’accesso

delle più comuni porte di comunicazione RS232/485, GPIO, CAN. ESF

tramite la sua interfaccia web, permette di sviluppare applicativi basati

su OSGi Declarative Service, consentendo di configurare ogni bundle

tramite una interfaccia web. 4.3.

**4.3 Architettura del software**

Per integrare ESF con il ricevitore lora SX1301, è stato necessario l’utilizzo

di due software aggiuntivi. Il primo è LoRa packet forwarder fornito

da Semtech. Questo applicativo permette di comunicare con le periferiche

di basso livello presenti nel ReliaGATE 10-11 in modo da astrarre ad

un più alto livello i dati ricevuti dal ricevitore SX1301. L’altra applicazione

utilizzata, è LoRa Gateway Bridge. Tramite questa applicazione è

stato possibile reindirizzare i dei dati forniti dal packet forwarder ad un

server MQTT.

Lo scopo di questa tesi era l’implementazione di due moduli osgi installabili

all’interno del framework ESF, in grado di interagire con le due

applicazioni LoRa packet forwarder e LoRa Gateway Bridge per control-

Aggiunger link lare da remoto il comportamento del ricevitore SX1301. .

**4.3.1 Semtech packet forwarder**

Il packet forwarder è un software che permette la ricezione e l’invio di

pacchetti radio Lora , tramite una connessione SPI con il device SX1301.

Nel caso di ricezione di un pacchetto, l’applicativo incapsula i dati ricevuti

in un formato UDP, e li ritrasmette nella rete internet/intranet. Per

la sua configurazione viene utilizzato un file Json nel quale troviamo tutte

le varie opzioni di configurazione per i moduli radio presenti al interno

del chip.

**4.3.2 LoRa Gateway Bridge**

LoRa Gateway Bridge, è un applicativo in grado di identificare i vari pachetti

UDP inviati dal packet forwarder e inotrarli ad un Broker MQTT.

Il software è scritto nel linguaggio GO e permette una configurazione

tramite linea di comando. Per renderlo interfaciabile con ESF, sono state

apportate delle modifiche al codice, in particolare è stata aggiunta

la possibilità di specificare il publish e il subscribed topic, i quali erano

hard-coded. Il codice sottostante, rappresenta un esempio di pacchetto

LoRa inoltrato dal LoRa Gateway Bridge ad un broker MQTT

**4.4.1 SX1301**

Per la ricezione dei pacchetti LoRa, è stato utilizzato il chip SX1301

prodotto da Semtech e collegato al gateway ReliaGate 10-11.

**Chip interno**

Per quanto riguarda la struttura interna del modulo radio, non si hanno

molte informazioni dato che la tecnologia è proprietaria di Semtech. Nella

documentazione ufficiale è presente una rappresentazione grafica dei

vari blocchi presenti al interno del modulo radio.

Come scritto nella documentazione e intuibile dalla figura 4.5 il chip inserire link

è in grado di scansionare contemporaneamente 8 canali diversi (IF0 a

IF7) permettendoli di rimanere in ascolto per garantire la ricezione dei

preamboli. Inoltre data la quasi ortogonalità degli Spreading Factor , il

chip è in grado di ricevere un pacchetto con uno Spreading Factor i anche

nel caso in cui si sovrapponga ad un altro pacchetto con Spreading

Factor pari a j, fintanto che i 6= j. Questa pseudo-ortogonalità utilizzata

in LoRa, permette di utilizzare differenti SF per ottenere un maggiore

throughput rispetto a schemi di modulazione tradizionale. Il ricevitoreSX1301 può demodulare fino ad un massimo di 8 pacchetti contemporaneamente.

Nella tabella 4.1 sono riportate le caratteristiche elettriche

massime del chip SX1301. Il Chip, supporta tensioni di alimentazione fino

a 4V e come e possibile osservare il range di temperatura in cui il chip

può operare è molto ampio, rendendolo ideale per applicazioni esterne

ed interne.

**4.5 Realizzazione**

Per gestire i due software preinstallati si è optato per la creazione di due

applicativi osgi distinti.

**4.5.1 Lora Config**

Il primo applicativo chiamato Lora Config si pone il compito di leggere

ed interpretare il file di configurazione utilizzato dal programma Packet

Forwarder, per poi andare ad esporre i parametri principali all’utente.

La libreria utilizzata per manipolare i file di tipo Json è

C’è qualcosa che non va passando da pag 45 a 46

Tramite la quale vengono riempiti i campi della classe LoraSettings.

Il file Json è composto da due parti. Nella prima parte troviamo tutte le

impostazioni per la configurazione dei canali (IF0 a IF7) , nella seconda

parte sono presenti le impostazioni per il forward dei pacchetti. Per

semplificare la gestione del bundle, sono state create due classi distinte

SX1301Configuration e GatewayConfiguration, accessibili tramite la

classe LoraSettings.

Per applicare le modifiche apportate alla configurazione, è necessario

che il packet forwarder venga riavviato. Per eseguire questa operazione,

si è scelto di utilizzare la libreria

la quale fornisce delle API per chiamare processi esterni, in particolare

si è scelto di utilizzare l’utility di sistema pkill per terminare il

processo del pkt forwarder.

**4.5.2 Mqtt Bridge Config**

Il secondo bundle prende il nome di MQTT Bridge Config e ha lo scopo

di esporre , tramite l’interfaccia web di ESF, le varie opzioni a linea di

comando dell’applicativo. Lora Gateway Bridge. Anche in questo caso

è necessario il riavvio del applicativo per fare in modo che le modifiche

abbiano effetto. Come nel bundle precedente è stata usata la libreria

Tramite l’interfaccia è possibile modificare i seguenti parametri:

• il topic sul quale pubblicare i messaggi ricevuti in formato UDP dal

pkt forwarder.

• il topic al quale rimanere in ascolto .

• su quale indirizzo e porta mandare/ricevere i pacchetti UDP.

• a quale broker Mqtt iscriversi.

• l’username e password per connettersi al broker.

**4.6 Misurazioni**

Finito lo sviluppo dei bundle, si è scelto di testare la distanza massima

di comunicazione raggiungibile dall’hardware in possesso. Il device utilizzato

per l’invio dei pacchetti LoRa prodotto dalla Semtech e prende il

nome di LoRa Mote. In via sperimentale è stato installato il gateway Re-

liaGate 10-11 ad una altezza di circa 11m. La prova di ricezione è stata

condotta per tentativi cercando , per quanto possibile, di testare diversi

punti distribuiti alla stessa distanza radiale. Per constatare l’avvenuta

ricezione del messaggio è stata utilizzata l’applicazione Android gratuita

My MQTT, tramite la quale è possibile iscriversi ad un topic predefinito

ed rimanere in ascolto dei messaggi pubblicati in esso. Come broker

MQTT si è scelto di utilizzare il broker open source "mosquitto.org".

**4.6.1 Osservazioni**

Dovendo testare la massima distanza, l’algoritmo ADR è stato disattivato,

permettendo così al device di non cambiare configurazione durante i vari

test. Il device LoRa Mote è stato configurato per l’invio di messaggi con

uno Spread Factor pari a 12 e una larghezza di banda pari a 125[Khz].

L’ambiente circostante al luogo dove l’antenna è situata è un ambiente

suburbano pianeggiante. La distanza massima raggiunta ,nei punti testati,

varia molto in base alla conformazione del territorio. In assenza

di edifici in linea d’aria tra gateway e il dispositivo LoRa Mote, è stato

possibile ricevere correttamente un pacchetto alla distanza di 8,2[Km].

Nella mappa ogni colore corrisponde ad un livello RSSI ( Received Signal

**Capitolo 5**

**Conclusioni**

Lo scopo dell’elaborato era di introdurre la visione di Internet of Things

e testare un possibile sviluppo applicativo tramite l’utilizzo della tecnologia

LoRa ed il framework ESF sviluppato da Eurotech.

Dopo un’introduzione relativa agli ambiti applicativi e alle problematiche

da affrontare in questa visione, sono state trattate le relative tecnologie

abilitanti per la realizzazione di questo paradigma mettendone in evidenza

i vari pregi e difetti. Successivamente, si è parlato della modulazione

LoRa e lo standard LoRaWAN andandone a trattare gli aspetti principali

che caratterizzano questa tecnologia. Infine si è approfondito il concetto

di microservizzi e di come questo concetto è stato implementato nella

piattaforma Everyware Software Framework di Eurotech. Questa piattaforma

si è dimostrata particolarmente adatta a questa visione, poiché

permette di risolvere alcuni dei principali problemi legati allo sviluppo di

applicazioni M2M, in particolare all’eterogeneità dell’hardware relativa

ai dispositivi in gioco. L’approccio con il framework OSGi risulta molto

efficace poiché permette la scrittura di software di ridotta complessità e

dinamicamente aggiornabile, riuscendo così a ridurre il tempo di sviluppo

e di conseguenza il time to market dei nuovi prodotti. Tramite questa

piattaforma e l’utilizzo di due software aggiuntivi, sono stati sviluppati

due bundle , Mqtt Bridge Config e Lora Config, per la gestione di un ricevitore

LoRa connesso al gateway prodotto da Eurotech ReliaGATE 10-11 .

A sviluppo completato, è stata condotta una prova per testare la distanza

massima raggiungibile dal modulo radio in dotazione, la quale è risultata

coerente con i dati forniti dalla casa produttrice del ricevitore.