Automazione di test di accettazione per dispositivi IoT embedded integrati nel cloud

Alessandro Righi

31 gennaio 2023

Indice

1 Introduzione

1.1 contesto "IOTINGA"

IOTINGA s.r.l. è una giovane realtà informatica veronese. Fra i vari settori in cui opera ricopre particolare rilevanza quello dell'IoT e dei prodotti connessi al cloud.

IRSAP s.p.a. è invece una grossa azienda rodigina leader nel settore del comfort termico. Storicamente produttrice di radiatori si distingue oggi per prodotti dal design altamente ricercato (i "termoarredi"), non che dall'elevato contenuto tecnologico, quali impianti VMC, radiatori elettrici connessi, e sistemi di gestione remota di impianti di riscaldamento.

IOTINGA ha aiutato IRSAP nella realizzazione proprio di queste ultime categorie di prodotti, grazie alla realizzazione della piattaforma "IRSAP NOW", un cloud al quale è possibile connettere tutti i prodotti smart del catalogo IRSAP così da controllarli da remoto mediante un'applicazione mobile iOS/Android, sempre sviluppata da IOTINGA, nonché tramite assistenti vocali quali Alexa o Google Home.

All'interno di questa piattaforma si innesta il prodotto oggetto di questa tesi, ovvero sia la gamma di radiatori elettrici connessi IRSAP. Il catalogo si compone di decine di prodotti, uno su tutti il "Polygon", vincitore del "CES Best of Innovation 2022", nonché di numerosi altri premi a livello internazionale, grazie al suo design altamente ricercato ed al suo contenuto tecnologico, a cui noi di IOTINGA abbiamo contribuito.

Tale dispositivo è dotato internamente di elettronica in grado di connettersi mediante Wi-Fi direttamente al cloud "IRSAP NOW", ed integra oltre alla funzione scaldante anche un'illuminazione ambientale mediante led RGBW.

Come IOTINGA abbiamo seguito direttamente lo sviluppo del firmware eseguito sul dispositivo stesso, oltre che l'integrazione nel cloud e nell'applicazione mobile esistente.

1.2 Motivazioni (diffcolta' testing IoT/cloud)

Uno dei problemi principali da evitare specialmente in questa tipologia di prodotti sono i problemi che si hanno dall'utente finale, per una serie di motivi:

- la prima impressione sul cliente è quella che conta, se il cliente si ritrova un prodotto che funziona male ne parlerà male e creerà un danno d'immagine all'azienda
- quando il problema si verifica dal cliente è difficile da diagnosticare. I clienti, e spesso anche gli installatori stessi, non hanno competenze tecniche e pertanto l'assistenza è difficile
- se il problema non è risolvibile mediante un aggiornamento firmware è necessario effettuare un reso, che ha dei costi molto elevati per il produttore, in quanto durante il trasporto molto spesso il prodotto viene danneggiato e quindi deve essere rimpiazzato con un nuovo

- il prodotto svolge comunque una funzione critica per l'utente, in quanto un suo malfunzionamento potrebbe voler dire rimanere senza riscaldamento, con tutti i disagi ed i danni che questo può comportare

Viene quindi da se che è di fondamentale importanza assicurarsi di far sì che quanti più problemi possibili vengano identificati prima che il software arrivi nelle mani dell'utente finale.

Inoltre, visto che la produzione di software esente completamente da bug è matematicamente impossibile, è fondamentale garantire che i prodotti una volta installati possano sempre essere aggiornati, in modo da poter rilasciare correzioni agli inevitabili (e sperabilmente pochi) problemi segnalati dagli utenti.

Tuttavia l'invio di aggiornamenti dovrebbe essere limitata allo stretto indispensabile: per quante misure di protezione si possano prendere l'operazione di aggiornamento è sempre molto delicata, in quanto vi sono numerose cose che possono andare storte, ad esempio può venire a mancare l'alimentazione durante una fase critica dell'aggiornamento, comportando una corruzione della meoria flash che rende il dispositivo inutilizzabile.

1.3 Prassi attuale testing (problemi aperti)

Quando detto al capitolo precedente fa capire quanto sia fondamentale testare in maniera ancora più approfondita (rispetto a quanto viene attualmente fatto per l'applicazione mobile ed il cloud) il firmware di questi radiatori.

Attualmente vi sono tre fasi di test:

1. test da parte dello sviluppatore
2. test di accettazione interna (in IOTINGA)
3. test di accettazione del cliente finale (IRSAP)

1.3.1 Test di sviluppatore

Quando uno sviluppatore finisce di implementare una nuova funzionalità o risolve un bug all'interno del firmware prima di considerare l'attività conclusa ed integrare il proprio lavoro nel ramo di sviluppo principale ed effettua i propri test.

Questi si occupano sia di verificare che quanto è stato implementato è conforme alla specifica approvata dal cliente (nel caso di nuove funzionalità) oppure che il bug sia stato risolto, sia che non siano stato modificato il funzionamento del sistema nelle parti che sono state impattate dalla modifica.

Tali test sono a discrezione dello sviluppatore, che avendo modificato il codice sa quali comportamenti sono impattati dalla modifica che ha realizzato e quindi devono essere provati.

1.3.2 Test di accettazione interna

Questi test sono effettuati prima di ogni rilascio di un nuovo artefatto verso il cliente.

Si occupano di validare che il software garantisca il funzionamento di una serie di casi d'uso critici, senza i quali il sistema stesso non sarebbe utilizzabile e quindi ulteriormente testabile. Solo se una versione del software passa tutti questi test può essere consegnata al cliente.

Essi si pongono dal punto di vista dell'utente finale, pertanto sono effettuati su un hardware completo, isolato però dal resto del sistema, ovvero dalla componente cloud e dall'applicazione mobile. Questo per evitare che vi sia il dubbio che il bug sia nel cloud o nella app anziché nel dispositivo stesso.

Attualmente vengono effettuati seguendo un documento contenente una serie di passi, ognuno idicante:

- azione: un'operazione da effettuare sul sistema mediante l'interazione fisica con il dispositivo (pressione di pulsanti) oppure mediante cloud (tramite un'apposito strumento che consente di simulare i messaggi inviati dal cloud e dalla app)

- risultato atteso: postcondizioni da verificare dopo aver effettuato l'azione, ad esempio "i led sono rossi" oppure "entro 5 secondi viene inviato al cloud un messaggio". Nel caso la postcondizione sia verificata è possibile procedere al passo successivo, altrimenti il test viene interrotto con esito negativo, e deve essere segnalato ad uno sviluppatore il problema.

Preferibilmetne questa procedura viene fatta eseguire da chi non ha preso parte allo sviluppo del sistema stesso. Questo per evitare che chi esegue la procedura, conoscendo le logiche interne del software, possa essere portato a saltare o ignorare determinati passaggi in quanto "ovvi", mentre chi non conosce il sistema è più propenso a seguire i passaggi alla lettera e segnalare ogni singolo comportamento discordante con quanto atteso.

L'eseguire la procedura citata sopra tuttavia non è esente da problematiche:

- tempo: la procedura richiede del tempo (all'incirca un ora) per essere eseguita. Nel caso il test non abbiano successo e si trovi un bug dopo averlo risolto è necessario rieseguire i test da capo.

- possibilità di errori: essendo una procedura meccanica e ripetitiva è fafile introdurre un errore umano nel proprio svolgimento.

Oer ovviare a questi problemi andremo a vedere come è stato sviluppato un sistema in grado di eseguire queste operazioni automaticamente ed in maniera integrata nel flusso di CI (Continuos Integration) attualmente già presente in azienda.

1.3.3 Test di accettazione del cliente finale

Sul cliente finale ricade la responsabilità (anche a livello legale) del prodotto che viene immesso sul mercato con il proprio nome sopra, e questo include anche il software. Questo comporta il fatto che a sua volta deve svolgere dei test per assicurarsi che il software sia conforme a quanto atteso, e nel caso segnalare i problemi riscontrati in maniera tale che vengano corretti.

Questi test sono volti a testare tutte le funzionalità del prodotto in tutte le loro possibili configurazioni, anche mediante l'ausilio di strumentazione altamente specializzata quali camere climatiche per valutarne l'efficacia di termoregolazione.

Nel caso questi test abbiano successo l'artefatto testato passa da stato di candidato al rilacio a produzione, e viene quindi installato in fabbrica su tutti i nuovi radiatori prodotti, nonché viene lanciato un aggiornamento OTA su tutti i dispositivi già installati presso i clienti finali.

Questi test, essendo svolti dal cliente finale, sono fuori dai nostri scopi di analisi.

2 Related work

2.1 Tool commerciali/open-source

3 Approccio

3.1 Panoramica

Andiamo a vedere come è realizzato il dispositivo in oggetto di questa tesi.

L'elettronica viene prodotta in due modelli differenti:

- luxury: versione basilare della scheda elettronica, installata tipicamente sui prodotti da bagno (scaldasalviette)
- design: elettronica riservata ai prodotti di fascia più alta. Offre in aggiunta a tutto quanto offerto dalla precedente elettronica un sensore VOC (qualità dell'aria) aggiuntivo e la possibilità di collegare una striscia a led RGBW per l'illuminazione ambientale. Inoltre è progettata in maniera da integrarsi direttamente all'interno del radiatore, di modo che non sia direttamente visibile.

Inoltre, un radiatore a parità di elettronica può o meno avere a disposizione le seguenti funzionalità:

- Fil Pilote: uno standard francese per l'interconnessione dei radiatori ad una centralina di controllo mediante un secondo ingresso di segnale a 230V
- led RGBW: disponibili solo su radiatore con scheda design, consente di illuminare l'ambiente circostante al radiatore per creare atmosfera

Tutte le versioni di scheda elettronica eseguono la stessa versione del firmware, che ha una fase iniziale in cui identifica la tipologia di scheda e le funzionalità opzionali abilitate, e di conseguenza configura le periferiche del microcontrollore di conseguenza.

Tuttavia per essere esaustivi è necessario svolgere i test di accettazione su entrambe le versioni di scheda elettronica, in quanto possono esservi comportamenti differenti.

3.2 Caratteristiche hardware

La piattaforma hardware ruota attorno ad un unico microcontrollore Wi-Fi della Telit, il GS2200M. Le caratteristiche di questo chip sono notevoli:

- processore dual core ARM Cortex M3 fino a 120Mhz, di cui un core dedicato alla gestione del Wi-Fi ed uno all'esecuzione dell'applicativo
- 1Mb di RAM in totale, di cui all'incirca 500kB utilizzabile dall'applicazione, il resto dedicata all'uso della parte Wi-Fi
- 4Mb di memoria flash interna, in parte dedicata al codice del firmware ed in parte come filesystem interno in cui memorizzare i dati dell'applicazione
- interfaccia Wi-Fi b/g/n a 2.4Ghz in grado di operare sia in modalità station (client) sia che access-point (AP) a cui connettersi direttamente
- 19 input/output digitali
- 3 output PWM
- 2 ingressi analogici mediante ADC, uno a 10 ed uno a 12 bit
- un interfaccia I2C hardware
- un interfaccia SPI hardware
- due interfacce seriali UART
- modulo RTC interno

Tale microcontrollore si interfaccia con le seguenti periferiche hardware presenti sull'elettronica:

- una resistenza (cartuccia) a 230V che provvede a riscaldare il fluido termoconduttivo presente all'interno del radiatore
- una sonda di temperatura NTC usata per rilevare la temperatura ambiente
- una pulsantiera dotata di due pulsanti capacitivi e dei led RGB di illuminazione come feedback verso l'utente
- un buzzer utilizzato per dare un feedback uditivo all'utente

- solo per le schede “design” un sensore di qualità dell’aria in grado di misurare i livelli VOC e CO2
- solo per i modelli che lo prevedono un ingresso per il segnale “Fil Pilote”
- solo per i modelli che li prevedono una striscia a led RGBW di illuminazione ambientale
- sempre solo per alcuni modelli una seconda sonda di temperatura in grado di misurare la temperatura del corpo riscaldante, in maniera tale da migliorare l’accuratezza degli algoritmi di termoregolazione

3.3 Caratteristiche software

Il firmware che gira sul dispositivo è scritto in linguaggio C99. La piattaforma Telit mette a disposizione un ambiente di sviluppo basato sul compilatore IAR C compiler.

L’SDK fornito dal produttore include il RTOS ThreadX e la relativa implementazione dello stack di rete NetX, oltre che una serie di librerie per:

- connessione e configurazione del Wi-Fi
- filesystem interno simil-FAT
- connessioni TLS
- server e client HTTP
- aggiornamento OTA con doppia partizione

Il firmware è essenzialmente suddiviso in 3 moduli:

- gestione della termoregolazione, che si occupa di tutto quel che è necessario per regolare la temperatura ambiente secondo le modalità di funzionamento del dispositivo
- gestione della comunicazione cloud, si occupa di mantenere sincronizzato lo stato interno del dispositivo con il server “IRSAP NOW” mediante protocollo MQTT
- gestione dell’interfaccia utente, si occupa di rispondere agli input dell’utente sulla pulsantiera e di darne relativo feedback mediante l’uso dei LED RGB e del buzzer

3.4 Stati interni del dispositivo

Il dispositivo ad alto livello può trovarsi in 3 stati distinti:

- non abbinato: in attesa di un primo abbinamento da app. Ogni funzione del dispositivo è esclusa finché l’utente non lo collega mediante l’applicazione

- non connesso: il dispositivo è stato in passato abbinato ma al momento non è connesso al cloud, perché ad esempio la connessione Wi-Fi non è momentaneamente disponibile
- connesso: il dispositivo è connesso e sincronizzato con il cloud

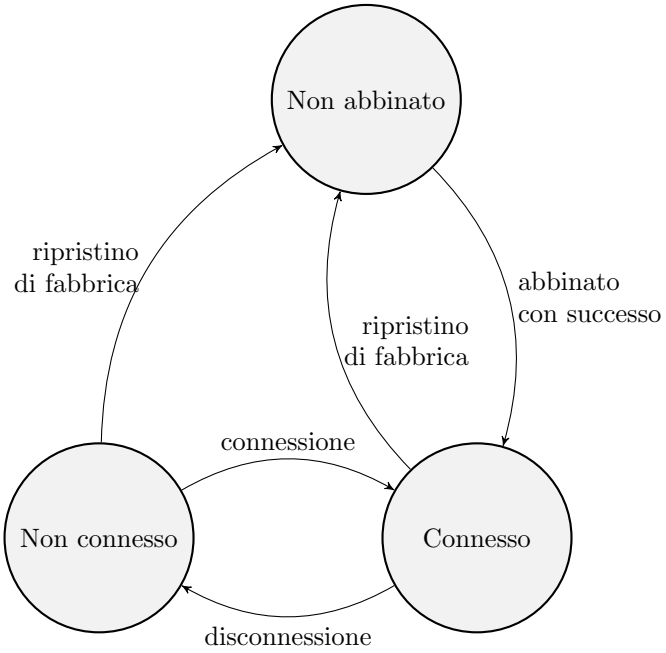

Figura 1: Stati del radiatore

3.5 Termoregolazione

La componente di termoregolazione si occupa di regolare la temperatura ambiente portandola il più vicino possibile a quanto desiderato dall'utente (set-point) mediante il controllo dell'accensione (on/off) dell'elemento riscaldante. Il feedback sulla temperatura ambiente è ottenuto mediante la sonda di temperatura NTC.

Il dispositivo ha diversi modi di funzionamento:

- standby: dispositivo completamente spento, sia per quanto riguarda il riscaldamento che per l'illuminazione LED
- antigelo: il dispositivo mantiene una temperatura di sicurezza (impostata dall'utente) per prevenire danni dati da una tempratura ambiente troppo bassa (ad es. congelamento delle tubature)

- vacanza: all'interno di un intervallo temporale impotato dall'utente funziona in modalità antigelo
- away: imposta un set-point ridotto (ECO) in quanto l'utente non è in casa
- programmato: segue una programmazione settimanale che consente per ogni giorno della settimana di creare fino ad 8 fasce orarie
- manuale temporaneo: segue un set-point manuale per un determinato tempo
- manuale: segue il set-point utente che è fisso e non varia mai configurato dall'utente, quindi torna a funzionare nella modalità precedente
- fil pilote: il dispositivo è controllato (ove disponibile) da un segnale esterno in ingresso sul cavo fil pilote

È possibile mediante interfaccia utente muoversi fra le varie modalità come dettagliato in figura 2.

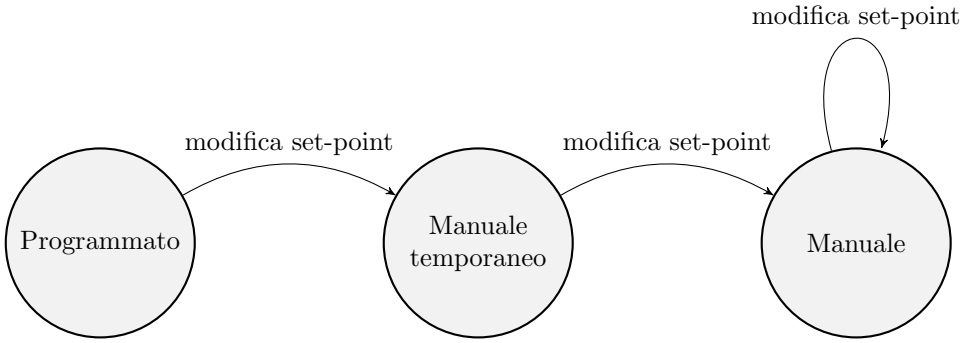

Figura 2: Modi del radiatore

3.6 Comunicazione cloud

La componente cloud è implementata su AWS. La connessione avviene grazie al protocollo MQTT usando il servizio broker di AWS IoT Core.

La connessione è cifrata mediante TLS ed autenticata mediante certificato TLS client specifico per il singolo dispositivo, generato e programmato nel dispositivo in fase di produzione della scheda elettronica da una CA intermedia a sua volta firmata dalla CA root di AWS come in figura 3.

Per quanto concerne il protocollo di comunicazione in origine abbiamo valutato il protocollo Device Shadowing[1] supportato nativamente da AWS IoT Core. Tuttavia, tale protocollo aveva delle limitazioni che non ne consentivano l'utilizzo nella nostra applicazione, in particolare:

[1] `https://docs.aws.amazon.com/it_it/iot/latest/developerguide/iot-device-shadows.html`

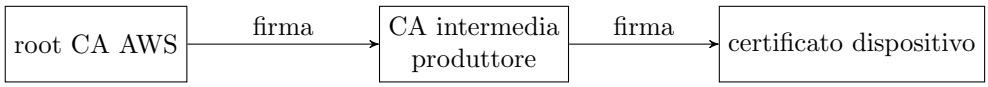

Figura 3: Catena TLS

- il pacchetto viene codificato in JSON, il che presenta un overhead di memoria e CPU notevole per il microcontrollore scelto. Inoltre la codifica JSON può introdurre dei bug di encoding
- vi è un hard-limit di 8Kb di dimensione massima di un documento di stato (shadow). Questo, seppur poteva essere sufficiente nelle prime versioni del prodotto, andava a limitare possibilità di espansione futura del prodotto
- il protocollo di comunicazione trasferisce dei delta, che sebbene riducano la dimensione di un pacchetto di dati rendono più complessa la sincronizzazione degli stati del sistema

Per tutte queste ragioni abbiamo scelto di adottare un protocollo binario proprietario, tramite il quale andiamo a trasferire stati completi del dispositivo.

Abbiamo deciso di mantenere comunque i concetti di alto livello dati dal protocollo AWS Device Shadowing, per tanto identifichiamo come:

- *state desired* come lo stato in cui si vuole portare il dispositivo, ovvero le impostazioni che l'utente può modificare agendo dalla app, quali ad esempio la modalità di funzionamento, la programmazione oraria, la configurazione dei LED RGB, etc.
- *state reported* lo stato attuale del dispositivo. È un superset dello stato desired, in quanto oltre a tutti i campi di quest'ultimo include anche tutti quei valori in sola lettura (ovvero che solo il dispositivo può modificare), ovvero i parametri statistici e di monitoraggio quali la temperatura ambiente, il livello di qualità dell'aria (VOC), gli allarmi del dispositivo, la qualità della connessione Wi-Fi, etc.

Il protocollo è quindi stateless, e consente di effettuare 3 azioni:

- **GET**: disponibile fra dispositivo e cloud, consente la richiesta dello stato *desired* corrente
- **UPDATE**: disponibile sia fra dispositivo e cloud che fra cloud e dispositivo consente di scambiari lo steso, rispettivamente stato *reported* e stato *desired*
- **DELETE**: eliminazione dello stato corrente presente su cloud

Il tipo di messaggio dipende dal topic MQTT sul quale i pacchetti sono pubblicati. Ad un messaggio pubblicato dal dispositivo verso il server il server risponde in base allo stato della richiesta sullo stesso topic con aggiunto un suffisso:

- /accepted: la richiesta è stata accettate
- /rejected: la richiesta è stata respinta dal server

Il client può identificare a quale richiesta fa riferimento ad una risposta attraverso un token (clientToken) che il client setta su ogni richiesta inviata e che il server aggiunge ad ogni risposta che invia al client.

Di base ogni messaggio prevede un header fisso che include i seguenti campi:

campo	byte	descrizione
timestamp	4	timestamp di invio del messaggio
clientToken	4	stringa random scelta dal client
version	4	versione del messaggio
length	2	lunghezza totale del messaggio (header incluso)
type	1	tipo di messaggio, identifica il payload presente

A seguito di questo vi è presente un payload che include tutti i parametri di stato del dispositivo. A seconda del campo type è possibile capire come decodificare il payload stesso del messaggio.

4 Implementazione

4.1 Interfacce

4.2 Implementazione hardware

4.3 Implemnetazione software

4.4 Integrazione (continuous integration)

5 Validazione sperimentale

5.1 scenari da testare

5.2 tempi di esecuzione (confronto automatico/manuale)

6 Comclusioni

6.1 Considarzioni

6.2 Passi futuri