

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DOMENIUL: CALCULATOARE ȘI TEHNOLOGIA  
INFORMAȚIEI  
SPECIALIZAREA: CALCULATOARE ÎNCORPORATE

# **Implementarea actualizării programelor prin comunicație fără fir**

LUCRARE DE DISERTAȚIE

Coordonator științific

Ș.l.dr.ing Nicolae-Alexandru Botezatu

Ion Vrabie

Iași, 2020



**DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII  
LUCRĂRII DE LICENȚĂ**

Subsemnatul(a) VRABIE ION,  
legitimat(ă) cu CI seria VS nr. 816852, CNP 1941016410086  
autorul lucrării IMPLEMENTAREA ACTUALIZĂRII PROGRAMELOR PRIN

---

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea FEBRUARIE a anului universitar 2019-2020, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data

---

Semnătura

# Cuprins

Capitolul 1. Introducere.....	1
1.1. Exemplu de aplicații.....	2
1.2. Așteptările de la aplicație.....	2
Capitolul 2. Fundamentarea teoretică și documentarea bibliografică pentru tema propusă.....	3
2.1 <i>Descriere generală a conceptului clasic de actualizare</i> .....	4
2.2 Descrierea generală a conceptului de actualizare fără fir.....	5
Conceptul de actualizare fără fir este alcătuit din două părți importante modelul de comunicare și proiectarea arhitecturală a programelor de pe dispozitivul care trebuie actualizat.....	5
2.2.1 Descrierea generală a aplicației de actualizare fără fir.....	5
2.2.3 Securitatea actualizării.....	7
2.1.4 Modele de comunicare specifice actualizării fără fir.....	10
2.2 Protocoale de comunicare specifice procesului de <i>Update Over The Air</i> ( <i>Actualizare fără fir abr. OTA</i> ).....	13
2.2.1 Bluetooth Low Energy(abr. BLE).....	13
2.2.2 Descriere generală a stivei TCP/IP.....	18
2.3 <i>Cercetări privind procesul de Update Over The Air</i> ( <i>Actualizare fără fir abr. OTA</i> ).....	21
Capitolul 3. Proiectarea aplicației.....	23
3.1. Descrierea platformei și componentelor hardware.....	23
3.2. Descrierea generală a arhitecturii de sistem.....	26
3.2.1. Tehnologii folosite.....	27
3.2.2. Descrierea comunicării interfață utilizator - micro-serviciu – nod gateway.....	27
3.2.3. Descriere serviciului BLE de actualizare fără fir.....	27
3.2.4. Descrierea comunicării nod gateway – nod rețea.....	29
3.2.5. Descriere organizare memorie flash pentru nodurile din rețea.....	30
3.3. Proiectarea aplicațiilor software pentru nodul gateway și nodurile din rețea.....	32
Capitolul 4. Implementare.....	35
4.1. Implementarea interfeței utilizator.....	35
4.2. Implementarea micro-serviciului de management a fișierelor de program.....	37
4.3. Implementarea aplicațiilor pentru nodul gateway și pentru nodurile din rețea.....	38
4.3.1. Șabloane de programare folosite.....	38
4.3.2. Implementarea automatelor cu stări finite pentru procesul de actualizare.....	39
4.3.3. Implementarea aplicațiilor.....	42
Capitolul 5. Testarea aplicației.....	46
Concluzie.....	49
BIBLIOGRAFIE.....	51
Anexa 1 Funcția de inițializare a aplicației.....	52
Anexa 2 Funcția ciclică a aplicației de actualizare fără fir pe nodul gateway.....	53

# ***Implementarea actualizării programelor prin comunicație fără fir***

Ion Vrabie

## **Rezumat**

Domeniul Internet of Things (Internetul tuturor lucrurilor, abr. IoT) și domeniul senzorilor sunt 2 domenii care se extind în momentul actual foarte rapid datorită componentelor hardware care au devenit mai ieftine și a crescut puterea lor de procesare, o parte importantă pentru aceste domenii sunt protocoalele de comunicare care s-au dezvoltat odată cu aceste domenii. Un protocol de comunicare interesant este Bluetooth Low Energy (abr BLE) care este dezvoltat pentru a micșora consumul de curent și creșterea timpului de viață a dispozitivelor (majoritatea dispozitivelor din aceste domenii sunt alimentate de baterii). O altă problemă în domeniile respective este mentenanța rețelelor care sunt în utilizare, pentru a micșora costurile de mentenanță și a crește flexibilitatea acestor tipuri de rețele s-a propus conceptul de actualizare a programelor prin protocoale de comunicație fără fir. Un alt avantaj al acestui concept este că dispozitivele pot fi aduse mai rapid pe piață deoarece o parte din funcționalități pot fi livrate mai târziu când dispozitivul este deja în uz. Conceptul de actualizare fără fir presupune actualizarea programelor pe care le rulează rețele de dispozitive fără fir la distanță fără implicarea factorului uman, lipsa factorului uman în procesul de actualizare crește complexitatea procesului de actualizare deoarece sistemul devine mai puțin tolerabil la erori și presupune implementarea de mecanisme de detectare a erorilor și recuperare a dispozitivului în cazul apariției de erori.

Soluția propusă în această lucrare implementează actualizarea fără fir pentru o rețea de dispozitive ce folosesc stiva de comunicație BLE. Platformă hardware folosită este ESP32 care are integrate module de WiFi și BLE care o fac foarte bogată în conectivitate, această platformă vine și cu partea software pentru stiva BLE și stiva TCP/IP adaptate pentru ea. Implementarea soluției de actualizare a fost despărțită în 4 componente. Interfața cu utilizatorul care este o aplicație web implementată folosind platforma Angular, utilizatorul sau administratorul rețelei de dispozitive va încărca noua imagine de program, imaginea va fi validată după folosind o cerere de tip *HTTP POST* care este trimisă către micro-serviciul de management al fișierelor cu program. Micro-serviciul este un server web care oferă interfețe *HTTP* care sunt consumate de către nodul gateway și interfața cu utilizatorul, acest server web este implementat folosind micro platforma pentru servere web Flask care este implementată în Python. Nodul gateway este un dispozitiv care se poate conecta la o rețea WiFi care are acces la Internet, el va verifica dacă conține ultima versiune de program prin trimitere unei cereri *HTTP GET* către micro-serviciu, dacă micro-serviciul conține o nouă versiune de program nodul gateway va trimite o nouă cerere *HTTP GET* în care va cere fișierul cu noua versiune de program pe care o va stoca pe cardul de memorie. Nodul gateway va scana rețeaua BLE după dispozitive, în cazul detectării de dispozitive se va începe procesul de actualizare. Nodurile din rețea sunt punctul final al sistemului și a căror program se dorește actualizat, aceste dispozitive vor difuza pachete de publicitate pentru a putea fi detectate de nodul gateway. Procesul de actualizare presupune ca nodurile finale trebuie să stocheze în memoria flash 2 imagini de program în paralel, de pe o partiție se va rula programul, cealaltă va fi folosită pentru stocarea noii imagini de program. Nodul gateway și nodurile din rețea vor trebui să folosească un client și un server specific protocolului de atribut generice care este folosit de stiva BLE.

## Capitolul 1. Introducere

Piața dispozitivelor inteligente fără fir a crescut foarte mult în ultimele două decenii datorită dezvoltării a mai multor tehnologii și domenii precum sisteme încorporate, inteligența artificială, tehnologia senzorilor. Internet of Things (Internetul tuturor lucrurilor, abr. IoT) este un domeniu care a apărut datorită acestor tehnologii și care permite transferul de date prin rețea de la o gamă largă de dispozitive fără a fi nevoie de factorul uman. Domeniul Internet of Things (Internetul tuturor lucrurilor, abr. IoT) se împarte în câteva domenii esențiale: mașini conectate, dispozitive portabile, case și orașe conectate( ex.monitorizarea consumului de energie), aplicații industriale( ex.monitorizare agricultura și utilaje), aplicații militare.

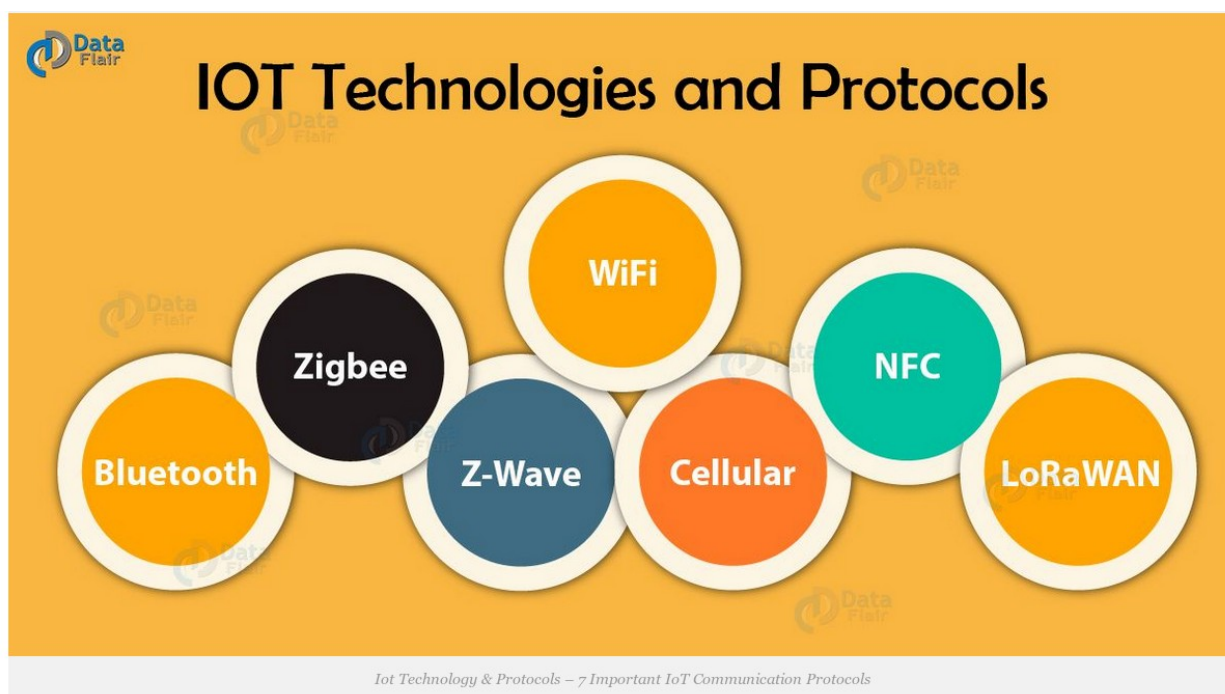


Figura 1.1: Exemplu de protocoale de comunicare specifice IoT

(<https://data-flair.training/blogs/iot-technology/>)

În toate domeniile descrise mai sus este nevoie de o modalitate ușoară de actualizare a programelor pe care le rulează dispozitivele respective fără implicarea factorului uman sau cu o implicare minimală a sa. O soluție propusă în acest sens este Update Over The Air (Actualizare fără fir abr. OTA), această soluție își propune distribuirea de actualizări la nodurile dintr-o rețea prin intermediul a diferite tehnologii de comunicare fără fir. În general dispozitivele folosite în Internet of Things (Internetul tuturor lucrurilor, abr. IoT) sunt senzori mici care au resurse mici, putere de procesare și memorie puțină iar domeniul lor de aplicare este foarte divers, din aceste cauze adaptarea unei stive comune pentru toate aplicațiile nu a fost posibilă. Stiva TCP/IP nu este aplicabilă pentru aplicații în acest domeniu, astfel au fost create și proiectate mai multe stive de comunicare: Bluetooth Low Energy (abr. BLE), Long Range (abr. LoRa), Sigfox, ZigBee.

Conceptul de Update Over The Air (Actualizare fără fir abr. OTA) nu este nou, el este folosit pe larg pentru telefoanele mobile unde factorul uman ar putea interveni în cazul apariției unei erori. Soluția aceasta are ca obiectiv rezolvarea de erori, îmbunătățirea funcționării sistemelor și actualizarea programelor care lucrează pe sistemele respective, conceptul de Update Over The Air (Actualizare fără fir abr. OTA) aduce următoarele beneficii:

- Scăderea prețurilor – managementul dispozitivelor fără fir aflate în utilizare devine mult mai ușor
- Actualizări continue – erorile din programe pot fi rezolvate pentru produsele aflate deja în utilizare
- Scalabilitate – dispozitivele pot primi funcționalități noi fără a fi nevoie de o conectare fizică
- lansări rapide pe piață – un produs poate fi lansat cu mai puține funcționalități astfel pe parcurs vieții sale pot fi adăugate noi aplicații

### *1.1. Exemplu de aplicații.*

Un domeniu în care Update Over The Air (Actualizare fără fir abr. OTA) încearcă să fie folosit astăzi este domeniul automotiv, datorită maturizării tehnologiilor fără fir marii producători de vehicule consideră această soluție pentru rezolvarea problemei de actualizare a programelor care rulează pe calculatoarele încorporate din interiorul mașinii, astfel se va micșora prețul de mentenanță.

Compania Tesla a fost prima care a implementat această soluție în 2012, astfel au reușit să rezolve într-un timp foarte rapid multe probleme în vehiculele care sunt deja în utilizare de asemenea datorită acestei soluții compania reușește să aducă noi aplicații pe vehiculele existente. Reușita companiei Tesla cu soluția Update Over The Air (Actualizare fără fir abr. OTA) a creat o noua tendință pe piață automotiv și nu numai (ex. IoT), astfel și ceilalți producători de mașini au început să cerceteze și să proiecteze propriile implementări a acestei soluții.

### *1.2. Așteptările de la aplicație*

Proiectul își propune implementarea conceptului de Update Over The Air (Actualizare fără fir abr. OTA) pentru o rețea de noduri care comunică prin Bluetooth Low Energy (abr. BLE), și nu au o conexiune directă la Internet, implementarea v-a fi creată din 4 componente:

- o aplicație web cu care v-a reprezenta interfața prin care v-a interacționa administratorul rețelei de noduri și prin intermediul căreia v-a putea să aducă ultima versiune de program în rețea
- un serviciu REST (engl. Representational State Transfer, abr. REST) care v-a primi un fișier în format binar cu programul de la aplicația web și la cererea v-a trimite fișierul către nodul gateway
- un nod gateway care v-a avea rolul de a cere de la serviciul REST ultima versiune de program și o v-a salva pe un card de memorie, după care v-a distribui noul program prin intermediul stivei BLE nodurilor din rețea, astfel un nod gateway va avea două stive de comunicare: stiva TCP/IP (de asemenea v-a folosi HTTP – HyperText Transfer Protocol) și stiva BLE
- nodurile rețelei care vor primi noul program de la nodul gateway și îl v-a salva pe partiția nefolosită de nod, nodurile din rețea vor avea doar stiva BLE.

Platforma folosită pentru nodul gateway și nodurile rețelei este ESP32 oferită de compania ESPRESSIF, această platformă suportă o interfață BLE și o interfață WiFi (vezi Capitolul 3. pentru mai multe informații despre platforma de dezvoltare) de asemenea compania producătoare oferă un mediu de dezvoltare foarte bine documentat, iar prețul mic face această platformă perfectă pentru dezvoltarea de aplicații de demonstrare a conceptelor. Implementarea unui asemenea sistem va face posibilă observarea limitărilor și dificultăților întâlnite în timpul implementării, de asemenea vor fi observate dependențele și punctele unde componentele sistemului vor trebuie să fie sincronizate.

## Capitolul 2. Fundamentarea teoretică și documentarea bibliografică pentru tema propusă

Conceptul de Update Over The Air (Actualizare fără fir abr. OTA) este o nouă soluție de a livra o nouă versiune de program pe un dispozitiv prin intermediul unui protocol de comunicare fără fir (ex. engl. Bluetooth Low Energy abr. BLE, WiFi, engl. Long Range abr. LoRa), astfel actualizare se petrece fără a fi nevoie ca dispozitivul să fie conectat fizic. O realizare completă a conceptului de Update Over The Air (Actualizare fără fir abr. OTA) are nevoie de mai multe componente la nivel de sistem:

1. O infrastructură engl. Cloud care va permite livrare și managementului proceselor de actualizare.
2. Nodurile din rețea vor avea nevoie de conectivitate la infrastructura Cloud sau în cazurile când nodurile din rețea nu au conectivitate atunci în rețea v-a apărea un nod gateway care v-a avea două stive de comunicare: o stivă TCP/IP care v-a permite comunicare cu infrastructura Cloud și o stivă specifică rețelei din care face parte.
3. Nodurile din rețea vor avea nevoie de o aplicație care v-a permite actualizare programului pe care îl rulează

Implementarea unei aplicații de actualizare pentru sistemele încorporate este o sarcină dificilă deoarece sistemele încorporate au constrângeri de resurse și toleranță mică la erori, câteva provocări specifice sunt:

- managementul memoriei – sistemul trebuie să asigure integritatea actualizării, astfel încât dacă noua versiune de program are erori trebuie să fie posibil reîntoarcerea la versiunea anterioară de program
- protocolul de comunicare – sistemul trebuie să tolereze erori de comunicare, astfel aceste erori trebuie să fie detectate sau chiar să existe un mecanism de recuperare a comunicației
- securitate – comunicarea este de tipul server client, trebuie să fie asigurată integritatea serverului, pachetele transmise trebuie să fie ofuscate pentru posibila atacatori, deoarece pachetele pot conține informații sensibil (ex. cheile de criptare)

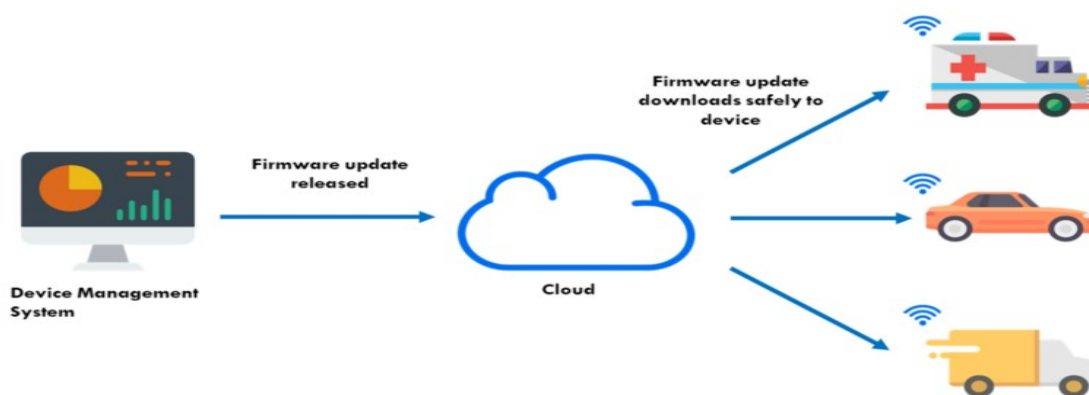


Figura 2.1: Viziune de sistem a conceptului OTA

(<https://dzone.com/articles/over-the-air-firmware-the-critical-driver-of-iot-s>)



## 2.1 Descriere generală a conceptului clasic de actualizare

Dispozitivele care folosesc conceptul clasic de actualizare au cel puțin două programe care există simultan pe același microcontroler: un program care v-a rula aplicațiile și programul de actualizare (engl. Bootloader), de obicei este prezent și un mic program de start care decide ce v-a rula dispozitivul: programul de actualizare sau programul cu aplicații. Programul de actualizare are rolul de a actualiza dispozitivul fără a fi nevoie de un hardware specializat pentru programare(ex programator de tip JTAG). Programul de actualizare poate comunica printr-o varietate de protocoale (ex. USART, CAN, I2C, Ethernet, USB,SPI) de obicei se alege un singur protocol. Efectuare actualizării are loc prin conectarea unui calculator care suportă protocolul de comunicare a dispozitivului pe care dorește să-l actualizeze, calculatorul trebuie să conțină un program special care poate transmite fișierul cu noul program către dispozitiv.

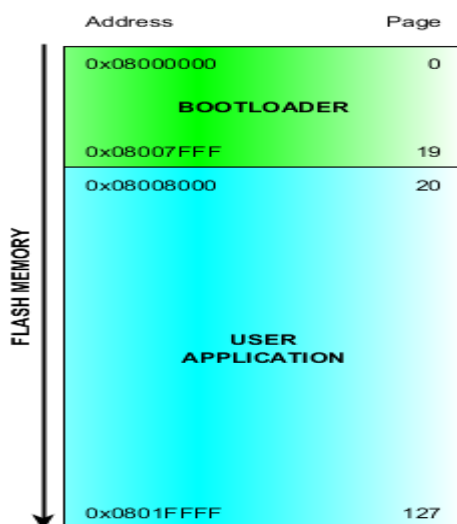


Figura 2.1: Exemplu de maparea memoriei flash

(<https://github.com/ferenc-nemeth/stm32-bootloader>)

Programul de actualizare nu este foarte diferit de o aplicație standard, defapt el este o aplicație care are permisiunea să rescrie zona de memorie flash specifică aplicației cu un nou program, programul de actualizare ar trebuie să fie foarte simplu, folosind minimum de resurse ale microcontrolerului pentru a își micșora amprenta având o mărime foarte mică. Programul de actualizare poate fi activat de obicei prin două metode:

- 1) Calculatorul care conține programul care poate face actualizarea se va conecta la dispozitiv și va cere printr-o comandă de la programul ce rulează aplicații să facă saltul în programul de actualizare. Saltul în programul de inițializare are 2 pași:
  1. Programul de aplicații v-a primi comanda de salt de la calculator și apoi v-a seta un pin al microcontrolerului pe o anumită valoare sau v-a scrie o valoare specifică la o adresă de memorie predefinită(possibil să fie folosită și alte metode de semnalare a saltului ), după care v-a face un reset al microcontrolerului de tip soft
  2. Programul de start v-a detecta noua valoare a pinului sau v-a citi valoarea de

- la adresa predefinită și va lua decizia să pornească programul de actualizare
- 2) Metoda această implică ca programul de start să poată detecta dacă zona de memorie flash a aplicației este coruptă și să pornească în acest caz programul de actualizare.

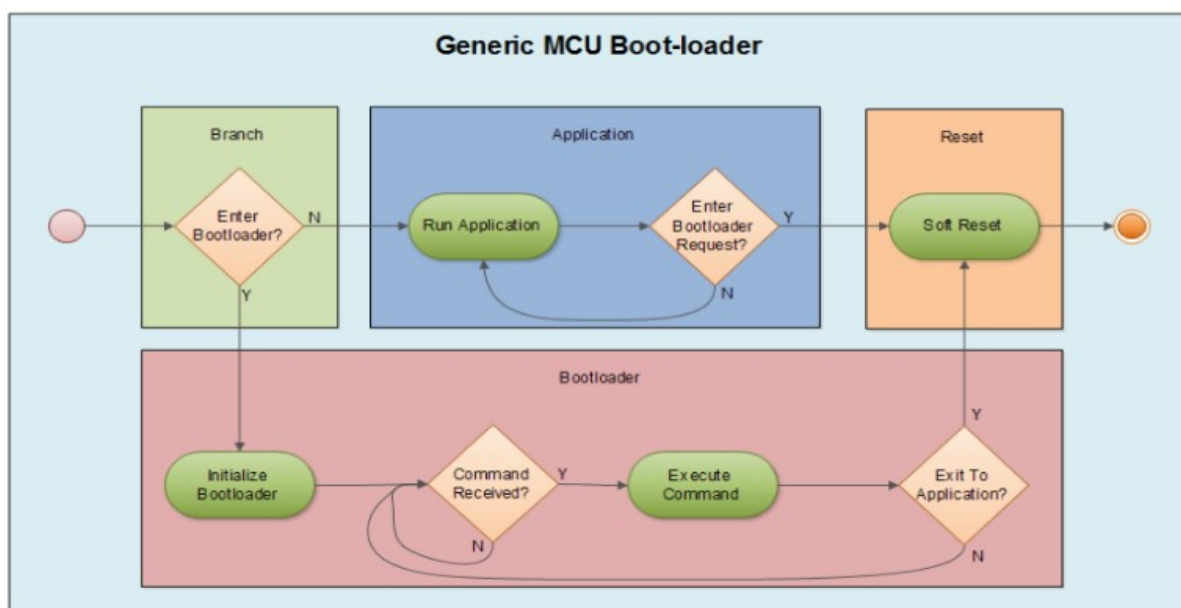


Figure 2.2: Diagram pentru metoda 1 de activare a programului de actualizare

([https://www.benigo.com/wp-content/uploads/images/Papers/bootloader\\_design\\_for\\_microcontrollers\\_in\\_embedded\\_systems%20.pdf](https://www.benigo.com/wp-content/uploads/images/Papers/bootloader_design_for_microcontrollers_in_embedded_systems%20.pdf))

## 2.2 Descrierea generală a conceptului de actualizare fără fir

Conceptul de actualizare fără fir este alcătuit din două părți importante: modelul de comunicare și proiectarea arhitecturală a programelor de pe dispozitivul care trebuie actualizat.

### 2.2.1 Descrierea generală a aplicației de actualizare fără fir

În ordine ca conceptul de actualizare fără fir să lucreze este nevoie ca dispozitivele aflate în utilizare să conțină o aplicație care v-a permite încărcarea unui nou program fără a fi nevoie de hardware extern, există mai multe abordări de realizarea a procesului de actualizare.

O abordare este folosirea unui program inițial și a unui program care conține aplicațiile astfel la startarea microcontrolerului programul inițial primar v-a fi primul care se va executa, el este responsabil pentru startarea programului cu aplicații, programul respectiv conține o aplicație care v-a permite startarea procesului de actualizare, la finalizare procesului de actualizare se v-a starta noua imagine. Această abordare se numește inițializare într-o singură etapă. Avantajul acestei abordări sunt :

1. Toate aplicațiile dispozitivului vor rula pe perioada procesului de actualizare.
2. Nu este nevoie de o resetare a sistemului pentru startarea procesului de actualizare.

Dezavantajele acestei metode sunt:

1. Programul inițial primar este specific producătorului, unii producători nu permit modificarea sa.
2. În multe sisteme pentru startarea noii imagini este nevoie de o resetare a dispozitivului actualizat.
3. Aplicația de actualizare este prezentă în ambele imagini.
4. Dimensiunea memoriei flash trebuie să permită aflare simultan a două imagini de program.
5. Aplicația de actualizare rulează în paralel cu alte aplicații, astfel ar putea apărea interferențe în procesul de actualizare.

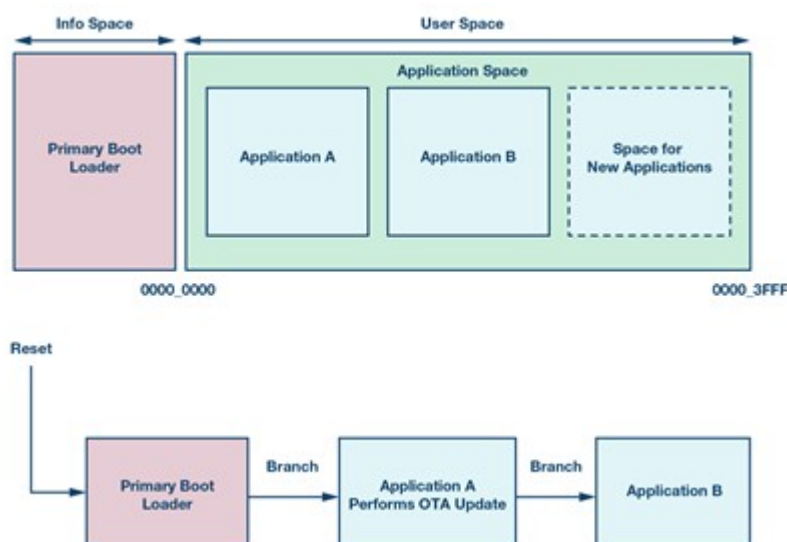


Figura 2.3: Exemplu de mapare de memorie pentru abordare de inițializare într-o singură etapă

(<https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR2DW9gZnOSdTUvatXbyQ17uAtnVJod5ggv5Y4JkUXDvN2Dgp3oBxJ8ttj8>)

O altă abordare este folosirea unui program de inițializare secundar, astfel în acest caz programul de inițializare primar îl va starta pe programul de inițializare secundar acesta conține aplicația de actualizare care va verifica dacă se dorește startarea procesul de actualizare, după care în funcție dacă avut loc actualizare se va starta imaginea de program deja existentă sau noua imagine de program. Această abordare poartă numele de inițializare în etape multiple și oferă următoarele avantaje:

1. Programul de inițializare secundar poate fi modificat de către dezvoltatori.
2. Aplicația de actualizare rulează independent, astfel va avea acces la toate resursele microcontrolerului.
3. Aplicația de actualizare nu mai este prezentă în programele cu aplicații.

Dezavantajele metodei sunt:

1. Startarea procesului de actualizare v-a necesita o resetare a dispozitivului dacă dispozitivul se află în starea normală de funcționare.
2. Aplicațiile dispozitivului vor fi inactive pe perioada actualizării

3. Dimensiunea memoriei flash trebuie să permită aflare simultan a două imagini de program și a unui program de inițializare secundar.

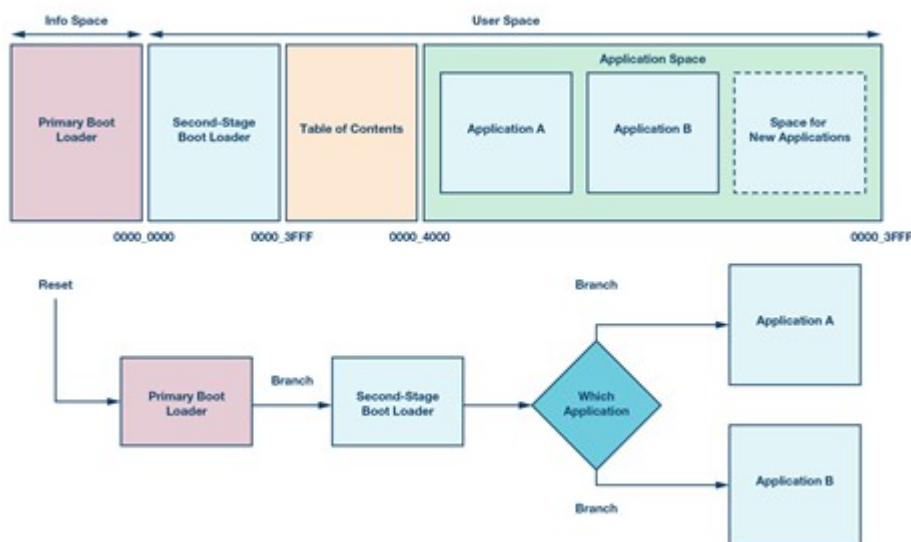


Figura 2.4: Exemplu de mapare a memoriei flash pentru inițializare în mai multe etape

(<https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR2DW9gZnOSdTUvatXbyQ17uAtnVJod5ggv5Y4JkUXDvN2Dgp3oBxJ8ttj8>)

Alegerea unei abordări depinde de constrângerile sistemului în care rulează dispozitivul care acceptă actualizare fără fir. În cazul unor sisteme critice este posibil combinarea acestor două abordări, programul cu aplicații v-a conține aplicația care v-a efectua actualizarea în caz că aceasta aplicație eșuează să realizeze actualizare atunci v-a avea loc resetare sistemului și pornirea unui nou proces de actualizare de către programul de inițializare secundar.

Dezvoltarea unei aplicații de actualizare vine cu câteva provocări tehnice:

- alegerea unui protocol de comunicare fără fir
- alegerea formatului în care va fi transmisă noua imagine de program
- alegere unei platforme potrivite deoarece se poate observa că actualizare fără fir necesită multă memorie de tip flash
- toleranța la erori precum: pierderea comunicării, erori de comunicare sau noua imagine a programului nu funcționează conform așteptărilor, această toleranță crește foarte mult complexitatea algoritmului de actualizare, în cazul actualizării clasice factorul uman poate interveni în cazul unei erori
- integritatea și securitatea actualizării (vezi Capitolul 2.2.3)

### 2.2.3 Securitatea actualizării

Securitatea și integritatea are o mare importanță pentru actualizarea fără fir, factorul uman nu mai este prezent în procesul de actualizare. Astfel trebuie asigurat că procesul de actualizare este făcută de un server de încredere și nu este o încercare de corupere a dispozitivului, de asemenea trebuie să fie asigurată posibilitatea de a detecta o corupere a imaginii programului care este actualizat (ex. erori de comunicare, încercarea unei părți rău

intenționate de a schimba conținutul programului). O soluție pentru problemele descrise sunt operațiile de criptare: criptarea propriu zisă și funcțiile de dispersie(funcțiile hash<sup>1</sup>).

Criptarea este folosită pentru a ofusca conținutul programului, astfel chiar dacă imaginea cu programul va fi interceptată în procesul de actualizare fără cheia de criptare/decriptare nu va fi posibil aplicarea ingineriei inverse asupra programului. Cheia de criptare/decriptare trebuie să fie cunoscută doar de serverul care urmează să realizeze actualizarea și dispozitivul ce va fi actualizat. Algoritmi de criptare cei mai cunoscuți sunt AES-128/256<sup>2</sup>, operațiile de criptare/decriptare sunt foarte costisitoare din punct de vedere a timpului, multe platforme oferă accelerare hardware.

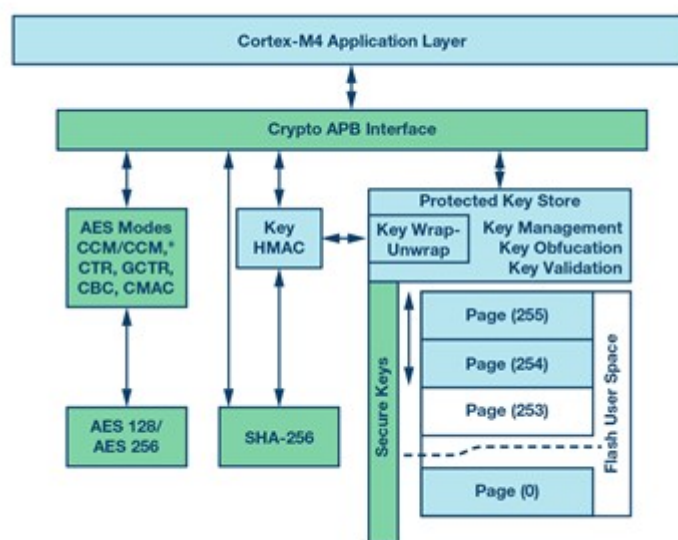


Figura 2.5: Digrama hardware pentru accelerarea criptografică pe platforma ADuCM4050

([https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR0A5XB5Q-Btgzlx-wkDPU-MOXIMw3ekcKSJ2\\_98KOFdFd0saVrtxuG5scs](https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR0A5XB5Q-Btgzlx-wkDPU-MOXIMw3ekcKSJ2_98KOFdFd0saVrtxuG5scs))

Funcțiile de dispersie (funcțiile hash pentru mai multe informații[10]) sunt folosite pentru semnarea imaginilor care conțin programul, astfel fișierul binar cu programul este folosit ca date de intrare pentru funcția de dispersie(ex. Algoritmul de hash securizat Engl. Secure hash algorithm SHA-256/512<sup>3</sup>), aceasta va genera o sumă de control de o lungime fixă, dispozitivul după terminarea procesului de actualizare va calcula suma de control a noii imagine și o va verifica cu suma de control obținută de la server astfel se va putea asigura că serverul care a realizat actualizare este de încredere. O soluție pentru problema autentificării este folosirea unei

- 1 Funcție de dispersie - sunt funcții definite pe o mulțime cu multe elemente (posibil infinită) cu valori într-o mulțime cu un număr fix și mai redus de elemente. Funcțiile hash nu sunt inversabile.
- 2 AES-128/256 în [limba engleză](#), **Standard Avansat de Criptare**), cunoscut și sub numele de **Rijndael**, este un algoritm standardizat pentru criptarea simetrică, pe blocuri, folosit astăzi pe scară largă în aplicații și adoptat ca standard de organizația [guvernamentală americană NIST](#)
- 3 SHA2 - este un set de [funcții hash criptografice](#) conceput de [Agenția de Securitate Națională \(NSA\)](#) a Statelor Unite ale Americii. Aceasta are la bază structurile Merkle–Damgård, care la rândul ei este o funcție de compresie unidirecțională făcută cu structuri Davies–Meyer cu un [cifru pe blocuri](#) specializat.

funcții hash și unui algoritm de criptare asimetric (ex. engl. Rivest–Shamir–Adleman RSA<sup>4</sup>) care folosește două chei: o cheie publică și o cheie privată (mai multe informații [11]).

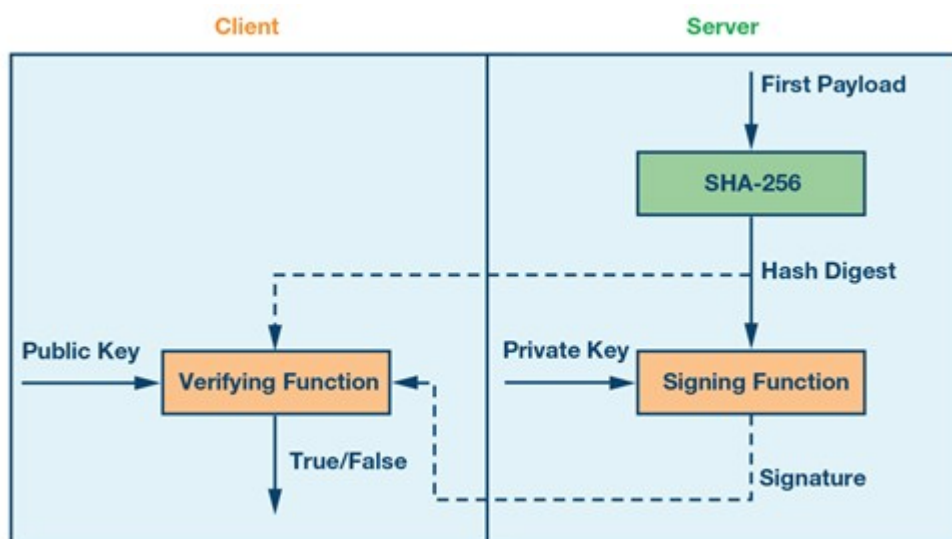


Figura 2.6: Diagrama de autentificare

([https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR0A5XB5Q-Btgzlx-wkDPU-MOXlMw3ekcKSJ2\\_98KOFdFd0saVrtxuG5scs](https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR0A5XB5Q-Btgzlx-wkDPU-MOXlMw3ekcKSJ2_98KOFdFd0saVrtxuG5scs))

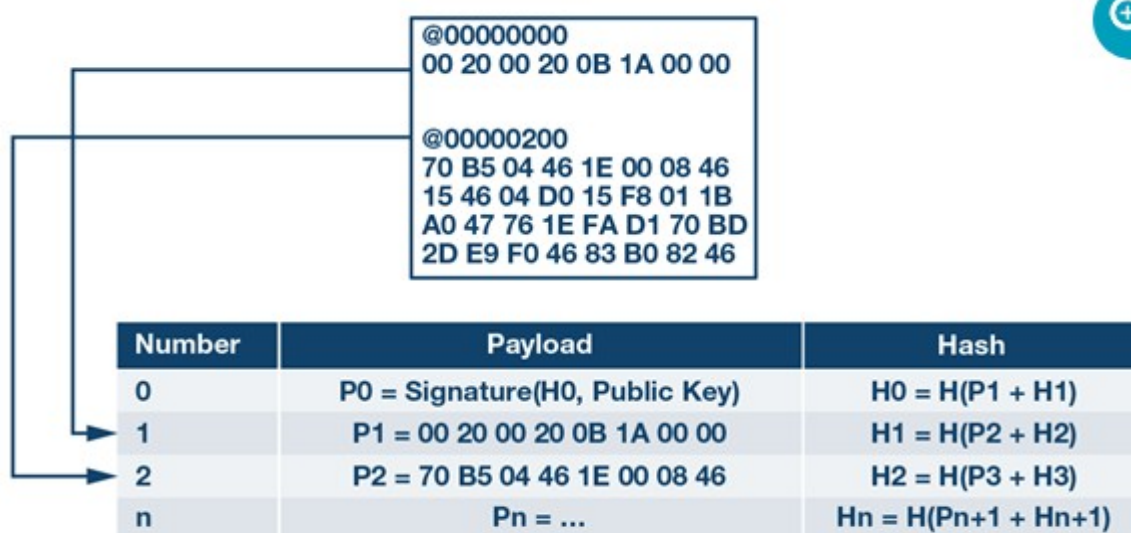


Figura 2.7: Exemplu de aplicare a sumelor de control înlănțuite

([https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR0A5XB5Q-Btgzlx-wkDPU-MOXlMw3ekcKSJ2\\_98KOFdFd0saVrtxuG5scs](https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html?fbclid=IwAR0A5XB5Q-Btgzlx-wkDPU-MOXlMw3ekcKSJ2_98KOFdFd0saVrtxuG5scs))

- 4 RSA face parte din clasa algoritmilor de criptare cu cheie publică ce au drept principală caracteristică existența a două chei: o cheie publică ce poate fi cunoscută de orice expeditor și o cheie privată cunoscută doar de destinatar.



Serverul va calcula cu ajutorul funcției hash suma de control după care suma de control v-a fi criptată cu cheia privată și va fi trimisă la dispozitivul actualizat. La rândul său dispozitivul va folosi cheia publică pentru a decripta suma de control și o va verifica dacă coincide cu suma de control calculată pe baza noi imagini de program primite astfel dispozitivul poate fi sigur că serverul care a realizat actualizare este de încredere. Abordarea această presupune primirea integrală a noului program, după care are loc verificarea integrității acestuia, noul program nu este trimis integral într-un singur pachet de către server astfel verificarea integrității va avea loc abia la primirea tuturor pachetelor, se dorește detectarea intruziunii cât mai rapid posibil.

O soluție pentru detectarea unui pachet corupt pe parcursul procesului de actualizare este algoritmul sume de control înlanțuite care poate fi observat în figura 2.7. Pachetul cu numărul 0 va conține suma de control a următorului pachet și va fi criptat de către server, dispozitivul actualizat va decripta pachetul și va salva suma de control. Pachetul cu numărul 1 va fi format din două părți o parte din noul program și suma de control pentru pachetul cu numărul 2, dispozitivul va calcula suma de control și o va compara cu suma de control din pachetul cu numărul 0 dacă sumele de control coincid va salva suma de control din pachetul cu numărul 1 și secvența se va repeta pentru următoarele pachete. Un avantaj al acestui algoritm ca o mare parte din procesare este asignată serverului care a startat procesul de actualizare și procesarea este distribuită de-a lungul actualizării.

#### 2.1.4 Modele de comunicare specifice actualizării fără fir

Modelul de comunicare decide cum va fi implementată soluția de actualizare fără fir, în domeniul Internet of Things (Internetul tuturor lucrurilor, abr. IoT) și a senzorilor fără fir se pot observa trei modele populare: comunicare dispozitiv la dispozitiv, comunicare dispozitiv la Cloud și comunicare dispozitiv la dispozitiv de tip gateway.

Specificul comunicației dispozitiv la dispozitiv este că dispozitivele din rețea se conectează direct între ele astfel nu mai este nevoie de un nod special de tip gateway sau de un server, asemenea rețele se numesc rețele de tip mesh<sup>5</sup>. Protocoalele folosite pentru acest tip de rețele sunt: Bluetooth Low Energy(vezi Capitolul 2.2.1), Z-wave<sup>6</sup>, Zigbee<sup>7</sup> sau LoRaWAN<sup>8</sup> (un exemplu se poate observa în Figura 2.8), aceste stive de comunicare nu folosesc foarte multe resurse ale dispozitivului aspect foarte important pentru domeniu Internet of Things (Internetul tuturor lucrurilor, abr. IoT) unde nodurile sunt limitate în putere de procesare și memorie. Tipul de comunicare dispozitiv la dispozitiv este specific aplicațiile care au nevoie să transmită pachete mici de date, exemple de aplicare sunt: case inteligente, orașe inteligente, în aceste aplicații se poate observa că un nod va fi pe post de senzor și un nod care va acționa actuatorul pe baza datelor de la nodul senzor.

Comunicația dispozitiv la Cloud implică conectarea dispozitivului la rețeaua Internet și la un serviciu de tip Cloud, conectarea se realizează prin intermediul unui fir de Ethernet sau prin conexiune WiFi. Tipul acesta de comunicație implică ca dispozitivele să suporte stiva TCP/IP care poate fi foarte scumpă pentru dispozitiv. Aplicațiile specifice acestui tip de comunicație sunt aplicațiile de monitorizare(ex. Monitorizare temperaturii, monitorizarea calității aerului) sau aplicație în care este nevoie controlul la distanță(ex. Figura 2.9).

<sup>5</sup> O rețea fără fir (radio) de tip mesh (traducerea expresiei [engleze Wireless Mesh Network](#)) reprezintă o rețea destinată transportării datelor, instrucțiunilor și vocii (informație sonoră) prin nodurile de rețea. Acest tip de rețea oferă conexiuni continue și dispune de algoritmi de autoreconfigurare în caz de noduri blocate sau neoperaționale

<sup>6</sup> <http://www.z-wave.com/>

<sup>7</sup> <http://www.zigbee.org/>

<sup>8</sup> <https://www.lora-alliance.org/>

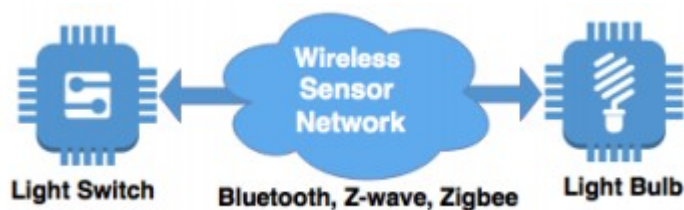


Figure 2.8: Exemplu de comunicare dispozitiv la dispozitiv[3]

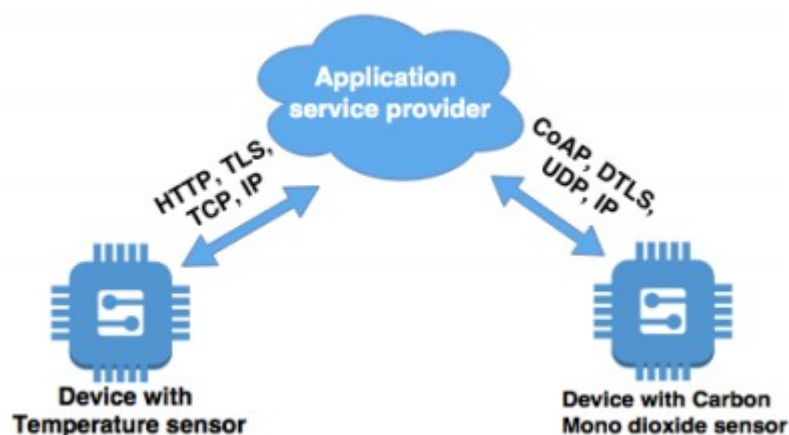


Figura 2.9: Exemplu de comunicare dispozitiv la serviciu Cloud[3]

Comunicarea dispozitiv la gateway caracteristica specială e apariția unui nod în rețea care are rolul de intermediar între un nod și serviciile Cloud. Abordarea aceasta este specifică când dispozitivele dintr-o rețea au nevoie de o legătură cu o altă rețea ce are o stivă diferită, astfel nodul gateway poate suporta mai multe stive de comunicare și are rolul de a face legătura între ele(ex. Figura 2.10).

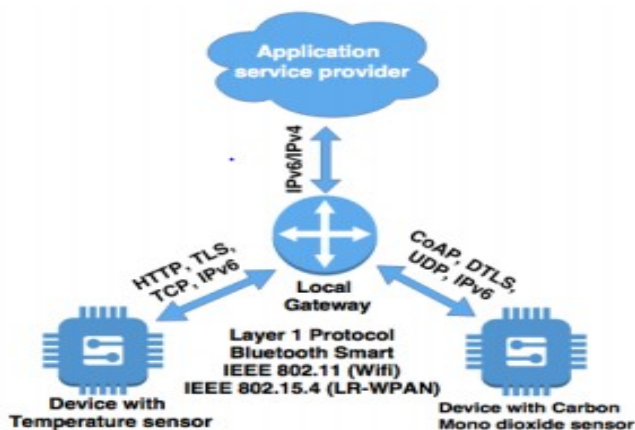


Figura 2.10: Exemplu de comunicare dispozitiv la nod gateway



Actualizarea fără fir are o arhitectură generică realizată dintr-un client și un server (poate fi observată în figura 2.11), în dependență de cerințele aplicației va fi folosită o stivă de comunicare. În figura 2.11 se poate observa un schimb de mesaje între server și client, serverul va notifica clientul de o nouă imagine de program, după care clientul și serverul vor negocia proprietățile și parametrii procesului de actualizare când părțile stabilesc acești parametri se începe transmisia noului program. La finalul procesului de actualizare clientul va notifica serverul dacă procesul de actualizare a avut loc cu succes, în caz contrar se va activa mecanismul de tratare a erorilor.

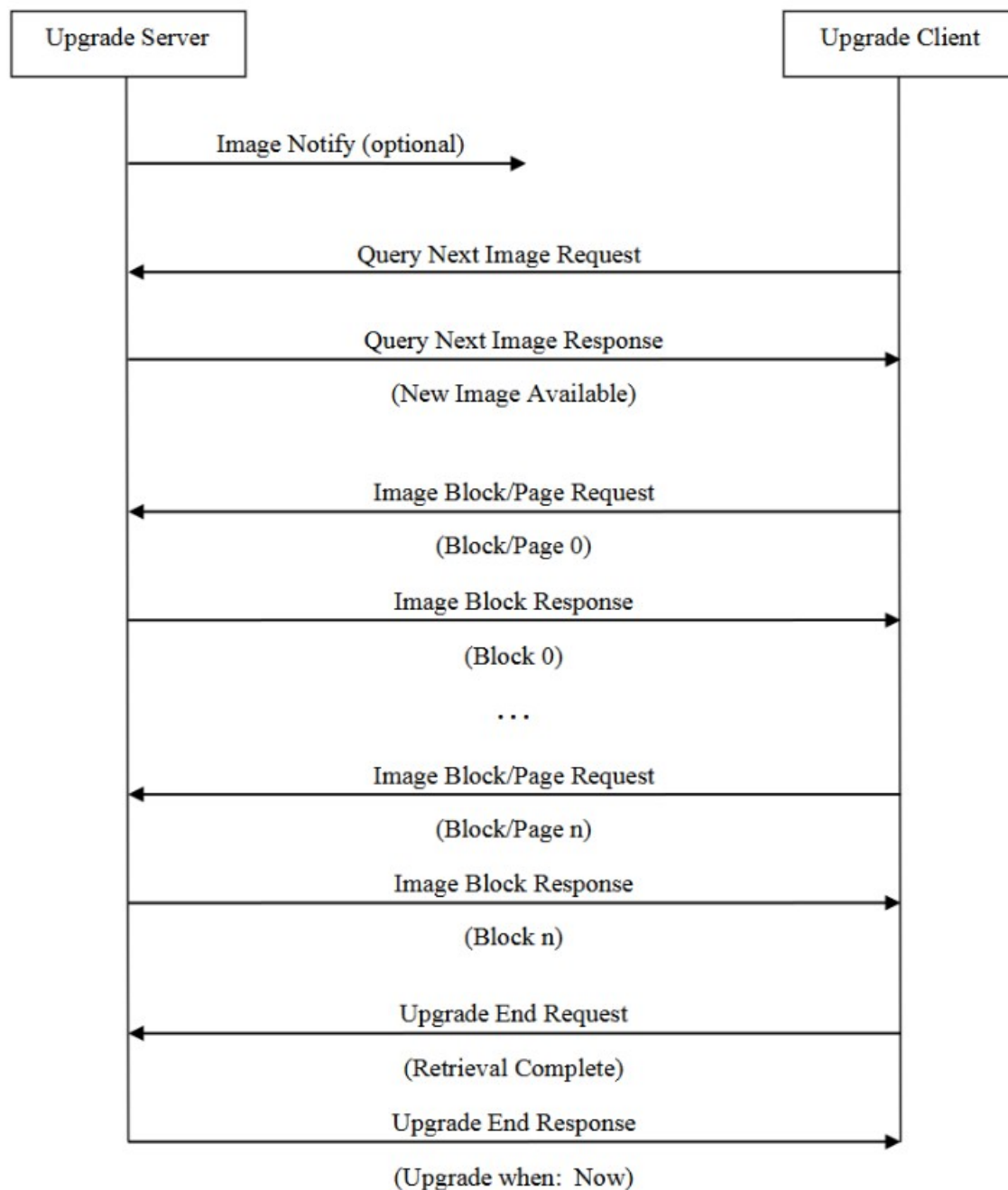


Figura 2.11: Exemplu de diagramă generică de mesaje [2]

## 2.2 Protocoale de comunicare specifice procesului de Update Over The Air (Actualizare fără fir abr. OTA)

### 2.2.1 Bluetooth Low Energy (abr. BLE)

Stiva de comunicare BLE este organizată în 3 niveluri: gazda (engl. host), controller și interfața între nivelul gazdă și controller (engl. Host-Controller Interface, abr. HCI). La rândul lor, fiecare din aceste nivele are la bază câteva subnivele. Nivelul gazdă are următoarele nivele:

- Protocolul de control și adaptare (engl. Logical link and adaptation protocol abr. L2CAP)
- Managerul de securitate (engl. Security Manager, abr. SM)
- Protocolul de atribut (engl. Attribute Protocol, abr. ATT)
- Profilul generic de atribut (engl. Generic Attribute Profile abr. GATT)
- Profilul de acces generic (engl. Generic Access Profile abr. GAP)

Nivelul controller are următoarele nivele:

- Nivelul de legătură
- Nivelul fizic

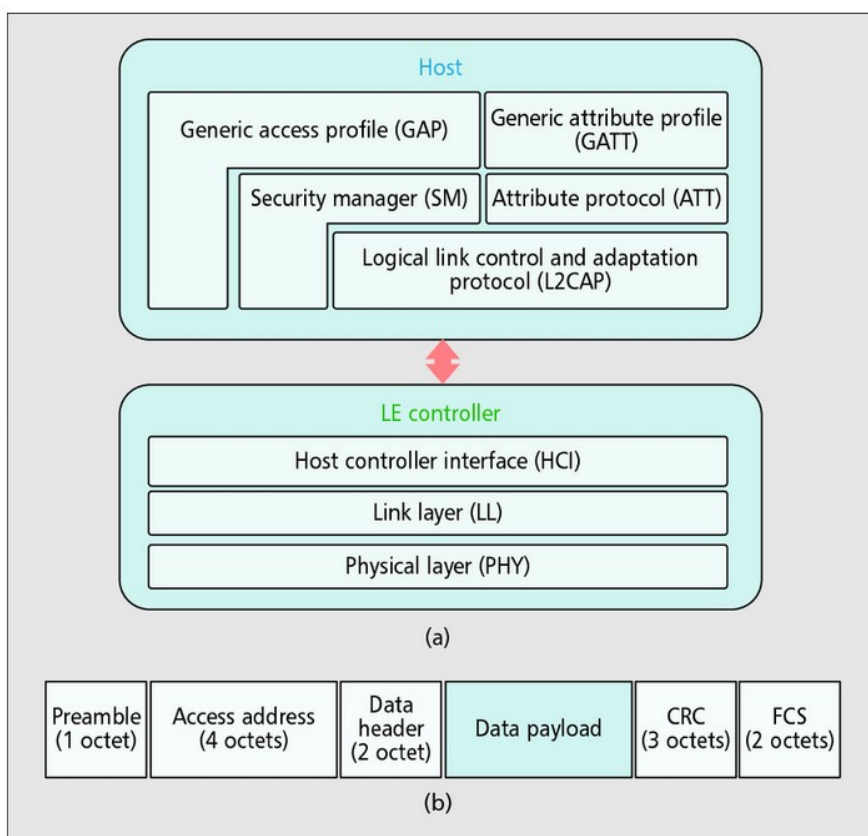


Figura 2.12: Stiva de comunicare BLE

([https://www.researchgate.net/figure/The-protocol-stack-and-frame-format-of-Bluetooth-Low-Energy-a-protocol-stack-b-frame\\_fig1\\_271327094](https://www.researchgate.net/figure/The-protocol-stack-and-frame-format-of-Bluetooth-Low-Energy-a-protocol-stack-b-frame_fig1_271327094))

Nivelul fizic conține circuitul analogic folosit pentru modulare și demodularea semnalelor și transformarea lor în simboluri digitale. Comunicarea BLE folosește în jur de 40 de canale pe gama de frecvențe de la 2.4 GHz la 2.4835GHz. 37 de canale sunt folosite pentru conexiuni și ultimele 3 canale (37, 38 și 39) sunt folosite pentru mesaje de tip publicare (engl. Advertising), pentru crearea și stabilirea de conexiuni și pentru difuzia de date. BLE folosește conceptul de salt între frecvențe, fiecare conexiune având loc pe o frecvență diferită astfel se minimizează efectul de interferență.

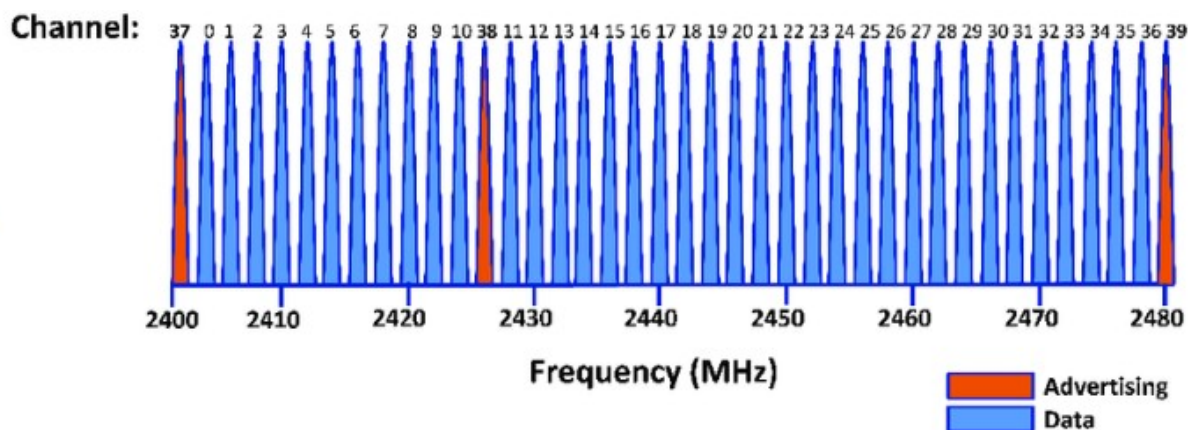


Figura 2.12: Canalele de frecvență specifice BLE

(<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt>)

Nivelul de legătură de date se interfațează direct cu nivelul fizic, acest nivel fiind o combinație de componente hardware și software, el este responsabil pentru respectarea cerințelor de timp conform specificațiilor, de asemenea face managementul circuitului radio, pe baza parametrilor radio configurați realizează codare și decodarea pachetelor radio. Acest nivel definește rolul unui dispozitiv:

- Advertiser – dispozitiv care face difuzie la mesaje de tip publicitate
- Scanner – dispozitiv care scanează după mesaje de tip publicitate
- Master – un dispozitiv care creează și face managementul unei conexiuni
- Slave - un dispozitiv care acceptă o conexiune și respectă parametrii impuși de master

Nivelul de legătură de date are responsabilitatea de a stabili conexiuni între dispozitive, filtrează pachetele de date pe baza adresei de Bluetooth<sup>9</sup> sau a datelor din pachet, face managementul intervalelor de conexiune<sup>10</sup> și poate configura și realiza criptarea pachetelor recomandat în cazurile în care multe dispozitive se află în aceeași zonă.

Protocolul de adaptare și control are 2 sarcini principale:

- Preia datele de la mai multe protocoale superioare și le încapsulează în formatul specific protocolului BLE și invers.
- Fragmentare și recombinație: pachetele de dimensiune mare de la nivele superioare le fragmentează în pachete de 27 de octeți dimensiunea maximă a unui pachet specific BLE, de asemenea primește pachete de la nivele inferioare și le recombina în pachete de dimensiune mare pentru a le trimite nivelurilor superioare

Protocolul de adaptare și control este folosit pentru rutarea pachetelor către nivelele superioare în special către nivelul de atribut și nivelul de securitate.

Protocolul de atribut este un protocol de timp server/client care este bazat pe atribut,

<sup>9</sup> Adresa Bluetooth – este un număr unic pe 48 de biți care identifică un dispozitiv într-o rețea Bluetooth

<sup>10</sup> Interval de conexiune – timpul între startarea a 2 conexiuni consecutive

clientul cere datele de la un server și serverul transmite datele către client, o nouă cerere nu poate fi transmisă cât timp nu s-a răspuns la cererea precedentă. Serverul va menține datele organizate sub formă de atribute, fiecare atribut are asignat un identificator universal unic (abr. UUID) care poate fi pe 16 biți, 32 biți sau 128 biți de asemenea unui atribut i se pot asigna permisiuni (ex. Atributul poate fi doar citit sau atributul poate fi doar scris). Un client poate face cerere de scriere sau citire a unui atribut pe baza identificatorului universal, serverul poate rejecta cu un pachet specific dacă identificatorul universal nu este cunoscut de server sau permisiunea atributului este nu permite răspunsul la cerere, în cazul unei cerere de citire serverul va răspunde cu valoarea atributului, în cazul unei cerere de scriere serverul va actualiza valoarea atributului cu valoarea obținută de la client și va răspunde la client un pachet de confirmare doar dacă clientul dorește confirmare la cererea de scriere. Serverul la rândul său poate anunța clientul că valoarea unui atribut s-a schimbat, printr-un pachet de tip notificare sau printr-un pachet de tip indicare în urma căruia serverul va aștepta un pachet de confirmare de la client (tipurile de transfer de date se pot observa în figura 2.13).

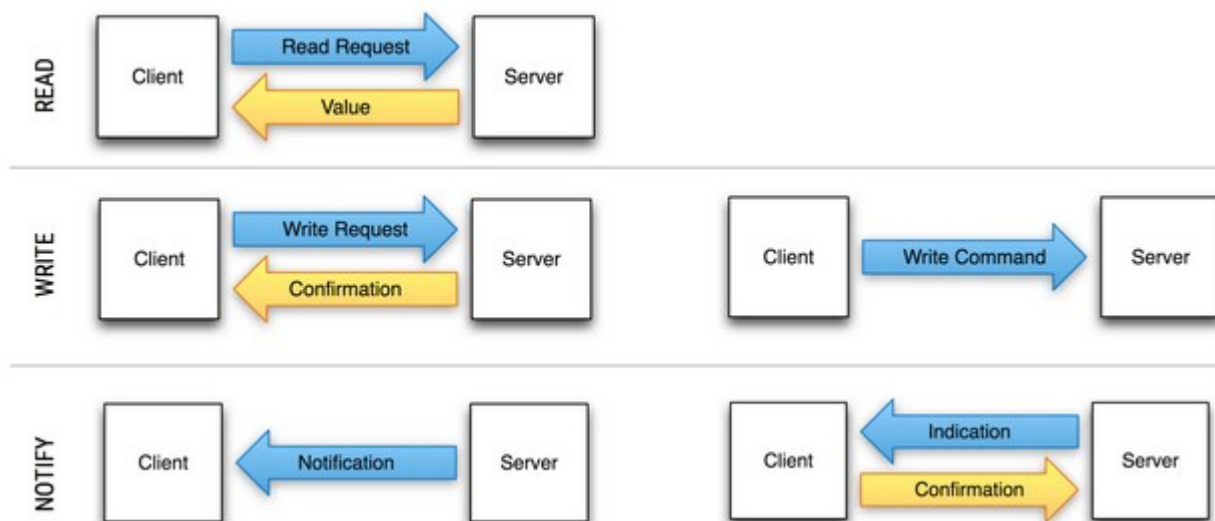


Figura 2.13: Tipurile de transfer de date specifice protocolului de atribute

(<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt>)

Managerul de securitate definește metodele și protocoalele pentru împerecherea dispozitivelor (engl. Pairing) și distribuirea de chei. Împerecherea are rolul de a stabili cheile de criptare și cheile pentru semnături. În general procesul de împerechere a dispozitivelor este compus din 3 faze:

- schimbul de parametri și configurări între dispozitive
- generarea de chei pe termen scurt/lung (valabil din specificațiile Bluetooth 4.2)
- distribuirea cheilor specifice (vezi mai multe în figura 2.14)

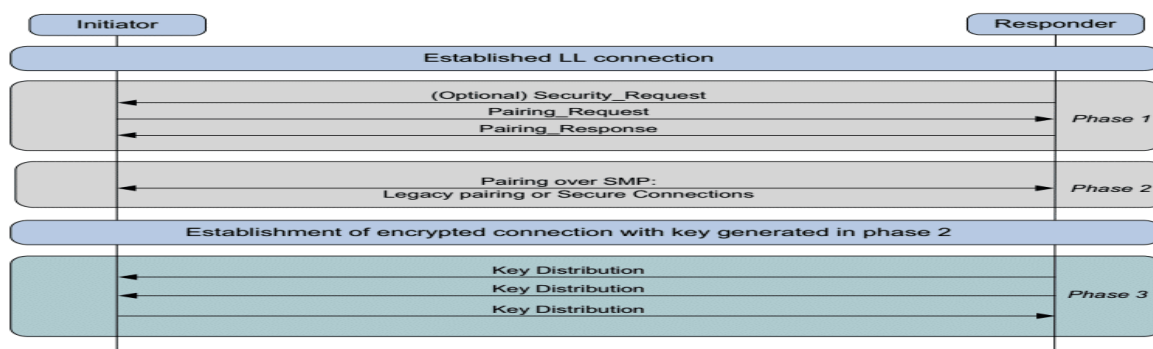


Figura 2.14: Schimb de mesaje pentru împerecherea dispozitivelor

(<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt>)

Profilul de acces generic este nivelul care controlează conexiunile și transmiterea de pachete de tip publicitate, alte responsabilități a acestui nivel este de a face dispozitivul vizibil în rețea și de a determina cum interacționează 2 dispozitive între ele. Profilul de acces generic definește 2 tipuri de dispozitive:

- dispozitiv periferic – sunt dispozitive mici cu resurse limitate care se pot conecta la dispozitive mai performante, dispozitive periferice tipice sunt senzori de temperatură, de umiditate.
- dispozitiv central - este dispozitivul care de obicei scanează după dispozitive și dorește să obțină date de la dispozitivele periferice(ex.o tabletă sau un telefon mobil).

Pachetele specifice acestui nivel sunt mesaje de tip publicitate și mesaje de tip răspuns la scanare, ambele pachete au aceeași dimensiune 31 de octeți, mesajul de tip publicitate este difuzat de către un dispozitiv periferic pentru a anunța dispozitivele centrale de prezența lor. Pachetul de tipul răspuns la scanare este opțional poate fi cerut de către dispozitivul central și care va conține informații suplimentare despre dispozitivul periferic(un exemplu de schimb de mesaje se pot observa în figura 2.15).

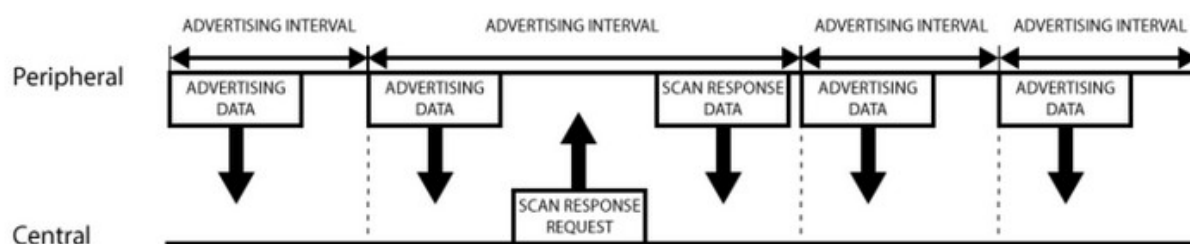


Figura 2.15: Exemplu de mesaje specifice profilului de acces generic

(<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap>)

Profilul generic de atribute are rolul de a controla transferul de date între 2 dispozitive care deja au creat o conexiune exclusivă, acest nivel folosește protocolul de atribute, în acest nivel dispozitivul periferic devine un server și se se poate conecta la un singur dispozitiv central care la rândul său devine client, un client se poate conecta la mai multe dispozitive periferice rețeaua formând o rețea de tip stea care poate fi observată în figura 2.16. Profilul generic de atribute aduce câteva componente specifice: profile, servicii, caracteristică și descriere.

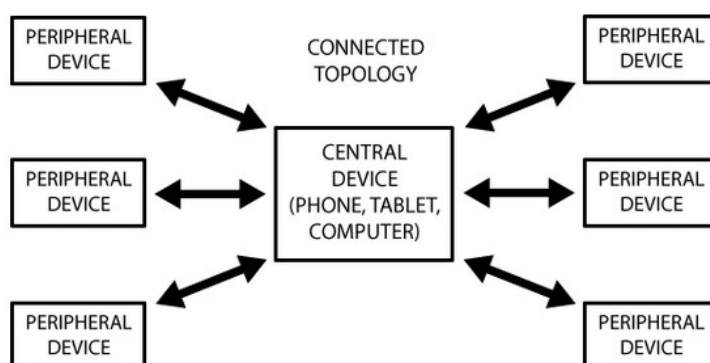


Figura 2.16: Topologie BLE

(<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt>)

Profilul nu există în sine o componentă, profilul reprezintă o colecție de servicii predefinite care sunt descrise de specificațiile oficiale pentru stiva BLE (vezi [5]) sau profilul poate fi specificat de către dezvoltatorul dispozitivului periferic. Un exemplu este profilul de monitorizare a inimii care este standardizat și este compus din serviciul de monitorizare a inimii și serviciul ce oferă informații despre dispozitiv.

Serviciul are rolul de a transforma și a împărtăși datele în entități logice, aceste entități logice poartă numele de caracteristică. Un serviciu are una sau mai multe caracteristici, serviciile se deosebesc între ele cu ajutorul identificatorului unic (abr. UUID) care poate fi reprezentat pe 16 biți sau 128 biți. La fel ca și în cazul profilelor sunt servicii oficiale descrise de specificațiile BLE, serviciile oficiale folosesc identificatori unici reprezentat pe 16 biți iar serviciile neoficiale sau specifice folosesc reprezentarea pe 128 de biți.

Caracteristica încapsulează o dată specifică (ex coordonatele x/y/z a unui accelerometru), la fel ca și serviciile caracteristicile sunt identificate printr-un identificator unic universal (abr. UUID). O caracteristică poate avea permisiuni asupra sa: doar citire, doar scriere sau ambele.

Descriptorul este nivelul cel mai jos a unui serviciu, rolul său este de a aduce informații despre caracteristică. Un exemplu în cazul unui serviciu care calculează distanța, este nevoie de un descriptor care să ofere informații în ce format este întoarsă distanța.

Heart Rate Service				
	Handle	UUID	Permissions	Value
Service	0x0021	SERVICE	READ	HRS
Characteristic	0x0024	CHAR	READ	NOT 0x0027 HRM
	0x0027	HRM	NONE	bpm
Descriptor	0x0028	CCCD	READ/WRITE	0x0001
Characteristic	0x002A	CHAR	READ	RD 0x002C BSL
	0x002C	BSL	READ	finger

Figura 2.17: Exemplu de serviciu specific BLE

(<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt>)

Un nivel special este interfața între nivelele de protocol și nivelul de controler are rolul de a separa aceste 2 nivele. Acest nivel are rolul de abstractiza și standardiza comunicația cu perifericul de BLE astfel încât nivelele superioare să fie independente de partea hardware și invers. Acest nivel este opțional și apare în configurațiile când perifericul de BLE nu face parte din microcontroler dar este o componentă independentă cu propriul procesor. În aceste cazuri acest nivel va trimite comenzi și va recepționa datele de la perifericul de BLE prin protocoale precum UART, SPI sau USB.



### 2.2.2 Descriere generală a stivei TCP/IP

Stiva TCP/IP reprezintă un set de protocoale folosite în internet și pentru rețele de calculatoare și este cea mai folosită stivă în lume deoarece suportă o mulțime de platforme hardware și software, este un standard non-proprietar și oferă posibilitatea de comunicare între diferite tipuri de rețele. Stiva are 4 nivele: nivelul legături de date, nivelul de rețea, nivelul de transport și nivelul de aplicație.

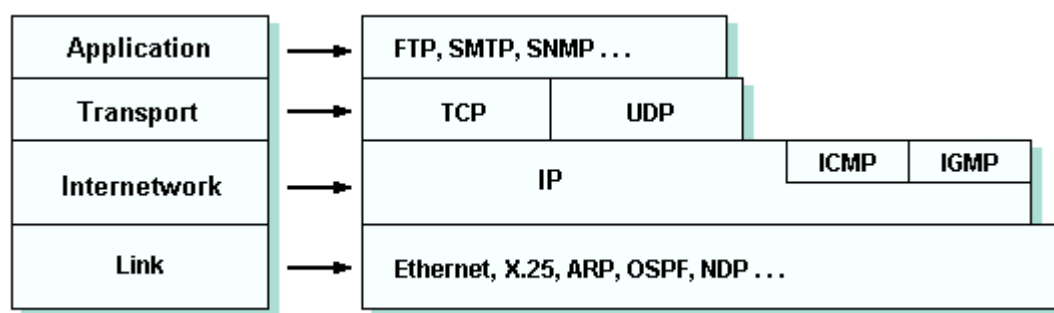


Figura 2.17: Nivele stivei TCP/IP

(<http://www.technologyuk.net/computing/computer-networks/internet/tcp-ip-stack.shtml>)

Nivelul legături de date este cel mai puțin definit nivel, deoarece el consistă din tehnologii specifice diferitor implementări de hardware dar de asemenea concepte virtuale folosite pentru rețele virtuale private<sup>11</sup> și pentru tuneluri între rețele<sup>12</sup>. În general stiva TCP/IP a fost proiectată astfel încât să fie independentă de hardware acceptând diferite implementări a nivelului legături de date. Nivelul legături de date este responsabil de traficul de date între dispozitive aflate în aceeași rețea locală astfel este responsabil pentru transmiterea și recepționarea de date, pregătirea cadrelor de date pentru transmisie prin adăugarea adreselor de control de acces media (engl. Media Acces Control, abr. MAC) și filtrarea cadrelor pe baza adreselor de control de acces media, de asemenea este responsabil pentru accesarea nivelului fizic.

Nivelul de rețea este responsabil pentru transmiterea de pachete între diferite rețele, astfel datele sunt trimise de la o rețea sursă către o rețea destinație, acest proces este numit rutare. Nivelul de rețea are 2 funcții de bază:

- Adresarea și identificarea gazdelor: acest proces este realizat prin sistemul de adrese ierarhice.
- Rutarea pachetelor: are rolul de a transmite pachetul de la sursă către destinație folosind cel mai apropiat ruter de destinație.

Nivelul de rețea este agnostic în ceea ce privește datele transferate, astfel el transferă date către

11 Rețea privată virtuală extinde o rețea privată peste o rețea publică, precum internetul. Permite unui calculator sau unui dispozitiv ce poate fi conectat la rețea să trimită și să primească date peste rețele publice sau comune ca și cum ar fi conectat la rețeaua privată, beneficiind în același timp de funcționalitatea, securitatea și politicile rețelei publice.

12 Un tunel între rețele este un protocol de comunicare care permite transmiterea datelor între 2 rețele, astfel o rețea privată poate să transmită date folosind o rețea publică, acest proces include o fază de reîmpachetare a datelor cu aplicare unui standard de criptare astfel datele fiind ofuscate.

diferite protocoale superioare. Două protocoale superioare care extind funcționalitatea nivelului de rețea sunt:

- Internet Control Message Protocol (abr. ICMP) care este folosit pentru a diagnostica și semnaliza problemele din rețea.
- Internet Group Management Protocol (abr. IGMP) este un protocol de comunicare folosit de gazde și routere adiacente pe rețelele pentru a stabili apartenențe la grupuri multicast.

Protocolul Internet este principala componentă a nivelului de rețea folosește adrese pe 32 biți pentru a identifica și localiza membrii unei rețele, aceste adrese permit identificarea a 4 miliarde de dispozitive, acest tip de adresare este specific pentru versiunea 4, în versiunea 6 a protocolului dimensiunea adresei a crescut la 128 de biți.

Nivelul de transport stabilește conexiunile între diferite aplicații, acest nivel este responsabil de livrarea pachetelor de la dispozitiv la dispozitiv fiind independent de rețeaua în care se află și de asemenea este agnostic de datele pe care le trimite aplicațiile. Acest nivel oferă funcționalități precum: detectarea erorilor, segmentarea pachetelor, managementul ratelor de transmisie, managementul congestiei și adresarea de aplicații (numere de port). Transferul de date și conexiunile de la punct la punct pot fi categorizate în 2 categorii: orientate pe conexiune implementat în protocolul de control al transmisiei (engl. Transmission Control Protocol, abr. TCP) sau fără conexiune implementat în protocolul datagramelor utilizator (engl. User Datagram Protocol, abr. UDP). Protocolul de control al transmisiei are rolul de a crea conexiuni între aplicații pe baza numerelor de port. Numărul de port este un număr pe 16 biți reprezintă o construcție logică care identifică un canal de comunicație care poate fi: un proces specific, un serviciu de rețea sau o aplicație. Protocolul de control al transmisiei oferă încredere în comunicare deoarece este responsabil ca pachetele să ajungă în ordinea corectă, detectarea și corectarea pachetelor corupte, eliminarea pachetelor duplicate, pachetele pierdute sunt retransmise și controlul congestiei. Protocolul datagramelor utilizator nu este la fel de sigur ca protocolul de control al transmisiei, singura verificare pe care o face acest protocol este o sumă de control, însă acest protocol vizează aplicațiile care au nevoie de transmisie de date în timp real, pierderea sau coruperea unui pachet nu este o problemă în aplicații precum: transmisie în timp real de sunet sau video.

TCP Segment Header Format								
Bit #	0	7	8	15	16	23	24	31
0	Source Port				Destination Port			
32	Sequence Number							
64	Acknowledgment Number							
96	Data Offset	Res	Flags		Window Size			
128	Header and Data Checksum				Urgent Pointer			
160...	Options							

UDP Datagram Header Format							
Bit #	0	7	8	15	16	23	24 31
0	Source Port				Destination Port		
32	Length				Header and Data Checksum		

Figura 2.18: Datele adăugate la pachete de către protocolul de control al transmisiei și de protocolul datagramelor utilizator



Nivelul de aplicație oferă protocoale și servicii care permit aplicațiilor transferul de date, acest nivel tratează nivelul de transport ca o cutie neagră care oferă o conexiune stabilă cu punctul final dar cunoaște informații precum numărul de port și adresa folosită de nivelul de rețea. Nivelul de aplicație este caracterizat de protocoale pe care le include, cele mai populare din ele sunt:

- Hypertext Transfer Protocol (abr.HTTP) – este metoda cea mai des utilizată pentru accesarea informațiilor în Internet care sunt păstrate pe servere World Wide Web(WWW). Protocolul HTTP este un protocol de tip text, fiind protocolul "implicit" al WWW. Adică, dacă un URL nu conține partea de protocol, aceasta se consideră ca fiind http. HTTP presupune că pe calculatorul destinație rulează un program care înțelege protocolul. Fișierul trimis la destinație poate fi un document HTML (abreviație de la HyperText Markup Language), un fișier grafic, de sunet, animație sau video, de asemenea un program executabil pe server-ul respectiv sau și un editor de text. După clasificarea după modelul de referință OSI, protocolul HTTP este un protocol de nivel aplicație. Realizarea și evoluția sa este coordonată de către World Wide Web Consortium (W3C)[6]
- Protocolul pentru transfer de fișiere(sau FTP, din engl. *File Transfer Protocol*) este un protocol (set de reguli) utilizat pentru accesul la fișiere aflate pe servere din rețele de calculatoare particulare sau din Internet. FTP este utilizat începând de prin anul 1985 și actualmente este foarte răspândit. Numeroase servere de FTP din toată lumea permit să se facă o conectare la ele de oriunde din Internet, și ca fișierele plasate pe ele să fie apoi transferate (încărcate sau descărcate). Webul nu aduce aici mari schimbări, ajută doar ca obținerea fișierelor să se realizeze mai ușor, având o interfață mai prietenoasă decât aplicațiile (prgramele) de FTP. Este posibil să se acceseze un fișier local prin adresa sa URL, ca și la o pagină de Web, fie utilizând protocolul "*file*" (fișier), fie pur și simplu utilizând calea și numele fișierului. Această abordare este similară utilizării protocolului FTP, dar nu necesită existența unui server. Desigur funcționează numai pentru fișiere locale.[7]
- Protocolul simplu de transfer al corespondenței - Simple Mail Transfer Protocol (prescurtat, SMTP; în traducere aproximativă *Protocolul simplu de transfer al corespondenței*) este un protocol simplu din suita de protocoale Internet, care este folosit la transmiterea mesajelor în format electronic în rețea de calculatoare. SMTP folosește numărul de port 25 TCP („smtp”) și determină adresa unui server SMTP pe baza înregistrării MX (*Mail eXchange*, „schimb de corespondență”) din configurația serverului DNS. Protocolul SMTP specifică modul în care mesajele de poștă electronică sunt transferate între procese SMTP aflate pe sisteme diferite. Procesul SMTP care are de transmis un mesaj este numit client SMTP iar procesul SMTP care primește mesajul este serverul SMTP. Protocolul nu se referă la modul în care mesajul ce trebuie transmis este trecut de la utilizator către clientul SMTP, sau cum mesajul recepționat de serverul SMTP este livrat utilizatorului destinatar și nici cum este memorat mesajul sau de câte ori clientul SMTP încearcă să transmită mesajul.[8](<https://ro.wikipedia.org/wiki/SMTP>)
- Secure Shell (SSH) este un protocol de rețea criptografic ce permite ca datele să fie transferate folosind un canal de securizat între dispozitive de rețea. Cele două mari versiuni ale protocolului sunt SSH1 sau SSH-1 și SSH2 sau SSH-2. Folosit cu precădere în sistemele de operare multiutilizator linux și unix, SSH a fost dezvoltat ca un înlocuitor al Telnet-ului și al altor protocoale nesigure de acces de la distanță, care trimit informația, în special parola, în clar, făcând posibilă descoperirea ei prin analiza traficului. Criptarea folosită de SSH intenționează să asigure confidențialitatea și integritatea datelor

transmise printr-o rețea nesigură cum este Internetul.[8]

### 2.3 Cercetări privind procesul de Update Over The Air (Actualizare fără fir abr. OTA)

Tema de actualizare fără fir a fost abordată în lucrarea *IoT Firmware Management: Over the Air Firmware Management for Constrained Devices using IPv6 over BLE* [1] care își propune implementarea unui sistem care nu folosește stiva de BLE nativă dar o înlocuiește parțial cu stiva de IPv6 adaptată pentru protocolul de BLE. (vezi figura 2.19)

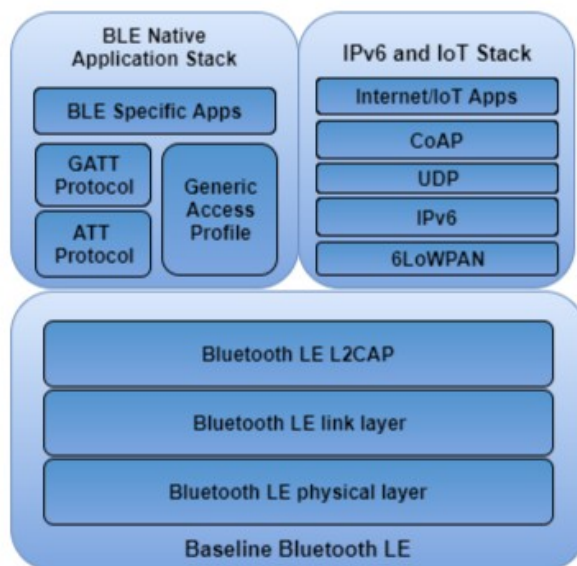


Figura 2.19: Comparare stiva BLE nativă cu stiva IPv6 adaptată pentru BLE

În figura 2.19 se poate observa ca nivelul fizic, nivelul de legături de date și protocolul de adaptare și control rămân aceleași ca și în stiva nativă de BLE, însă nivelele superioare sunt înlocuite cu următoarele protocoale:

- 6LoWPAN este un acronim pentru protocolul internet versiunea 6 pentru rețele personale cu putere și resurse limitate (engl. *IPv6 over Low-Power Wireless Personal Area Network*) definește mecanisme pentru încapsularea și compresia antetului care permit transferul de pachete IPV6 prin rețele bazate pe standardul IEE802.15.4<sup>13</sup>.
- Protocolul de Internet versiunea 6 (abr. IPv6)
- Protocolul datagramelor utilizator (abr. UDP) vezi capitolul 2.2.2
- Protocolul pentru aplicații cu constrângeri (engl. *Constrained Application Protocol* abr. CoAP)<sup>14</sup> este un protocol specializat pentru dispozitive cu putere de procesare mică, astfel le permite să comunice cu alte aplicații din Internet care suportă acest protocol.

În lucrarea această se propune implementarea unui sistem de actualizare fără compus din 4 componente:

- Server LwM2M<sup>15</sup> - este responsabil pentru managementul și controlul de dispozitive din rețea, de asemenea managementul programelor care rulează pe nodurile din rețea

<sup>13</sup> IEE802.15.4 – este standard tehnic care definește operațiile pentru rețelele de tip personal cu rate mici de transfer (engl. *low-rate wireless personal area network* abr. LR-PWAN)/

<sup>14</sup> <https://tools.ietf.org/html/rfc7252>

- Repozitoriu de programe – acest server are rolul de a servi fișierele cu programele noi către clienți care suportă protocolul LwM2M.
- Gateway – este un dispozitiv intermediar care va transfera comunicare de la nod către servere și invers.
- Dispozitiv nod – este un dispozitiv cu putere de procesare și resurse reduse, dispozitivul va suporta un client LwM2M care va comunica cu serverul, de asemenea trebuie să fie capabil să suporte scrierea în memorie un nou program.

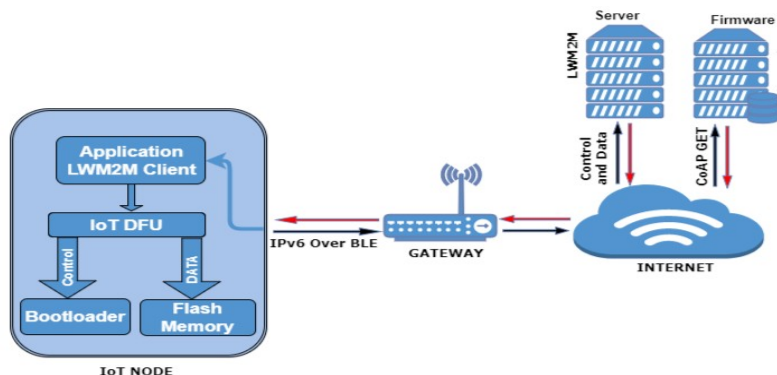


Figura 2.20: Arhitectura sistemului din lucrarea IoT Firmware Management: Over the Air Firmware Management for Constrained Devices using IPv6 over BLE [1]

O altă soluție interesantă care încearcă să rezolve problema de securitate în actualizarea fără fir este prezentată în lucrarea *UpdaThing: A secure and open firmware update system for Internet of Things devices* [2]. În această lucrare se propune un sistem compus din 4 componente:

- Server de actualizare – este responsabil de monitorizarea clienților pentru a detecta dispozitivele care au nevoie de actualizare, notificarea clienților de prezența unui noi imagini de program, servirea noi imagini clienților. Server-ul folosește protocolul HTTPS care implementează protocoale de nivel de transport securizat (engl. Transport Layer Security abr.TLS)<sup>15</sup>
- Server de semnare – este responsabil de includerea în imaginea de program a cheii publice a serverului de actualizare și certificatul, semnează imaginea de program care este oferită de dezvoltator după care va încărca imaginea semnată în serverul de actualizare. Serverul de semnare și actualizare sunt separate deoarece se dorește o implementare bazată pe servicii.
- Clientul actualizat – este dispozitivul care urmează să fie actualizat, este responsabil pentru cererea noi imagini de program de la server și scrierea acestei imagini în memorie. Clientul la fel ca serverul de actualizare implementează protocolul HTTPS, la prima startarea dispozitivului cu noua imagine de program va genera un certificat propriu pe baza cheii și certificatului inclus în imagine de către server-ul de semnare, cu acest noi certificat dispozitivul se va putea autentifica la server-ul de actualizare.
- Programele de dezvoltare – este un set de scripturi care automatizează crearea sistemului de fișiere Linux cu noile funcționalități și configurații pentru noua imagine de program, după care noua imagine de program este trimisă către server-ul de semnare.

<sup>15</sup> LwM2M – este un protocol pentru comunicare de tip mașină la mașină și managementul dispozitivelor dintr-o rețea de dispozitive fără fir, este un protocol la nivelul de aplicație și este de tipul server client(vezi [9])

<sup>16</sup> Nivelul de transport securizat sunt protocoale criptografice care permit comunicații sigure pe Internet.

## Capitolul 3. Proiectarea aplicației

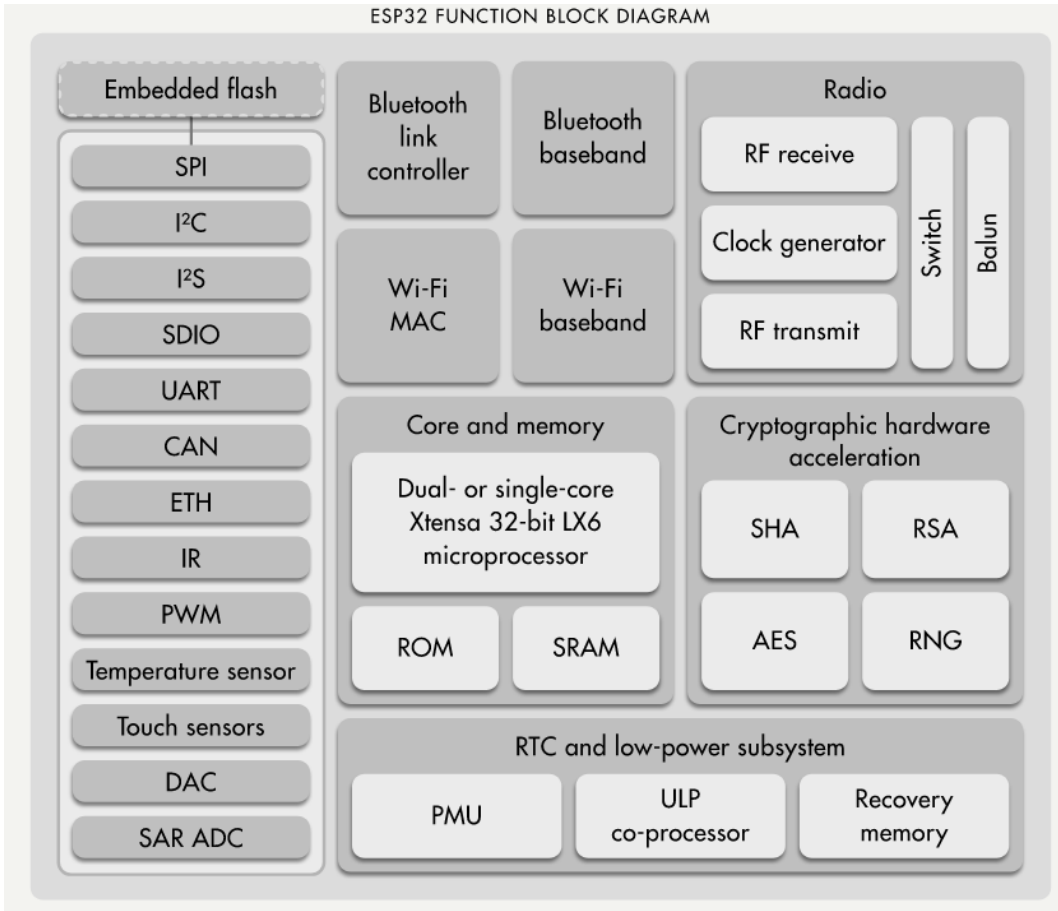
### 3.1. Descrierea platformei și componentelor hardware

Pentru implementarea conceptului de actualizare fără fir a fost folosită platforma hardware ESP32 care este un sistem pe chip, are integrat modulul de Wi-Fi și modul de Bluetooth care poate funcționa în modul clasic dar și în modul de Bluetooth Low Energy (abr. BLE, Bluetooth 4.2), astfel această platformă este foarte bine dotată în ceea ce privește conectivitatea. ESP32 are următoarele componente:

- Procesorul folosit este Tensilica Xtensa cu 2 nuclee sau 1 nucleu pe 32 de biți versiunea LX6 și poate opera la frecvență de 160 Mhz sau 240 Mhz, de asemenea este dotat cu un co-procesor care funcționează în mod de consum redus, poate accesa convertorul analogic digital când toată platforma se află în modul de funcționare somn adânc.
- Suportă conectivitate fără fir:
  - ESP32 implementează stiva TCP/IP și protocolul 802.11 b/g/n/e/i și specificațiile pentru Wi-Fi direct
  - ESP32 suportă comunicare Bluetooth clasic cât și comunicarea Bluetooth Low Energy astfel poate comunica cu toată gama de dispozitive care suportă acesta stivă
- ESP32 conține următoarea memorie internă: 448 KBytes de memorie ROM care este folosită pentru startare și funcțiile de bază, 520 KBytes de SRAM, 8 KBytes de RTC rapid SRAM folosit pentru stocare de date și de nucleele de bază când se face tranziția din modul somn adânc în modul normal, 8 Kbytes de RTC încet SRAM care este folosit de co-procesor în timpul funcționării în modul somn adânc, 1 KBit de memorie de tipul eFuse<sup>17</sup> dintr care 256 de biți sunt folosiți de către sistem pentru adresele de acces media (abr. MAC) și configurări ale chip-ului restul 768 biți pot fi folosiți de către dezvoltator, în funcție de versiunea de platformă memoria flash poate avea dimensiunile: 0 MBytes (ESP32-D0WDQ6, ESP32-D0WD, și ESP32-S0WD), 2 MBytes (ESP32-D2WD) și 4 MBytes (ESP32-PICO-D4 SiP). ESP32 suportă memorie flash și SRAM externă: până la 16 MBytes de memorie flash mapată în zona memoriei de cod a CPU-ului, poate fi accesată pe 8 biți, 16 biți și 32 biți, se poate executa cod din această memorie și până la 8 MBytes de memorie flash/SRAM poate fi mapată în zona memoriei de date a CPU-ului, suportă accesare pe 8 biți 16 biți sau 32 de biți, citirea este suportată de memoria flash și de memoria SRAM, citirea este suportată doar de memoria SRAM.
- ESP32 este foarte bogat în periferice precum: un convertor analog digital prin aproximări succesive care suportă până la 18 canale, 2 convertoare digital analogice, 10 senzori capacitivi de atingere, 4 interfețe SPI, 2 interfețe I2S, 2 interfețe I2C, 3 interfețe UART, 1 controler master pentru SD/SDIO/CE-ATA/MMC/eMMC, 1 controler sclav pentru SDIO/SPI, 1 interfață de Ethernet cu controler de DMA dedicat și suport pentru protocolul de precizie de timp IEEE1588, 1 interfață de CAN versiunea 2.0, 1 controler de infraroșu până la 8 canale, PWM pentru motor, PWM pentru led-uri, 1 senzor Hall.
- ESP32 suportă funcționalități de securitate precum: implementarea completă a standardului de securitate IEEE802.11 incluzând WPA, WPA/WPA2 și WAPI, startare securizată (engl. Secure boot), criptarea memorie flash și accelerare hardware pentru pentru operațiile criptografice: AES, SHA-2, RSA, curbe eliptice criptografice (abr. ECC), generatoare aleatoare de numere.

<sup>17</sup> eFuse – este o tehnologie inventată de către IBM care permite în timp real și în mod dinamic reprogramarea chip-urilor de calculator.





### Figura 3.1: Diagrama bloc a de funcții ESP32

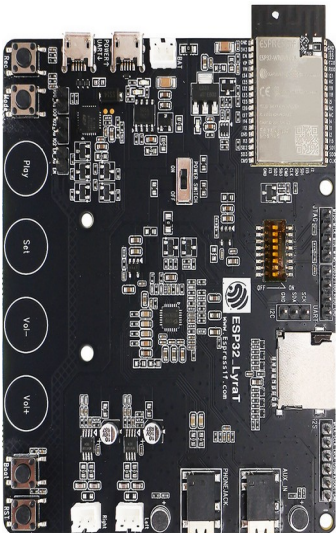


Figura 3.2: ESP32-LyraT

([https://  
www.espressif.com/en/  
products/hardware/  
development-boards](https://www.espressif.com/en/products/hardware/development-boards))

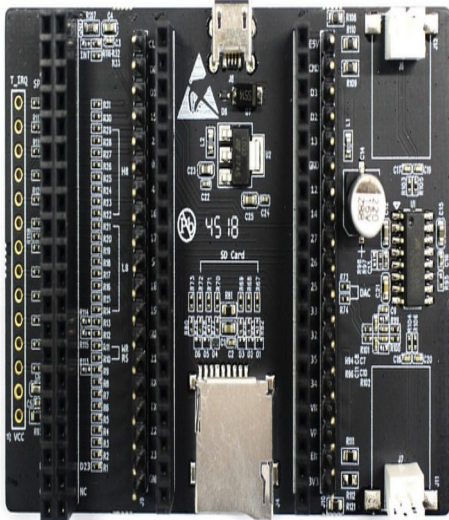


Figura 3.2: ESP32-LCDKit

(<https://www.espressif.com/en/products/hardware/development-boards>)

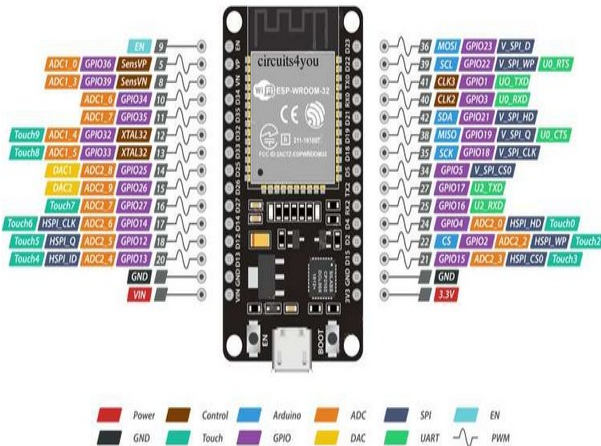


Figura 3.3: ESP32-Devkit maparea pinilor

(<https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/>)

Pentru stocarea imaginilor de program sa folosit un cititor de carduri de tip SD care are o interfață SPI iar cu ajutorul unui driver pentru sisteme de fișiere se poate scrie și citi date de pe un SD card. Acest modul are un regulator de 3.3V și un comutator automat de nivele de tensiune astfel se poate interfața cu microcontrolere care folosesc 3.3V și 5 V.

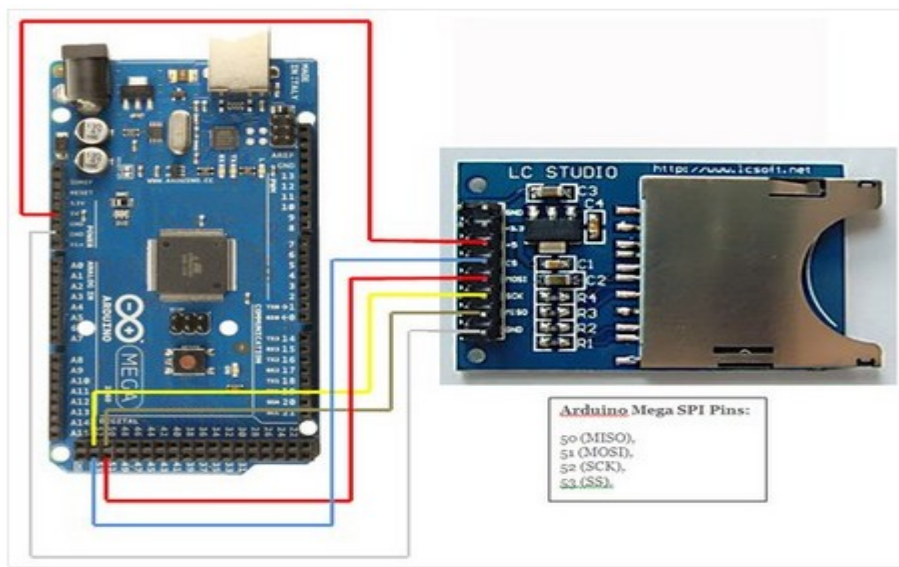


Figura 3.5: Conectarea modului cititor card SD la un Arduino Mega

(<https://www.indiamart.com/proddetail/sd-card-module-for-arduino-21138316730.html>)

Ecranul Oled SSD1306 cu rezoluția de pixeli 128x64 a fost folosit pentru afișarea de informații despre dispozitivul la care este conectat. Oled SSD1306 are chip CMOS OLED/PLED pe post de driver care controlează un ecran cu o matrice de diode în formă de puncte care emit o lumină organic/polimerică. Acest ecran folosește interfața de comunicare I2C și suportă o gamă tensiuni între 3.3V și 6V astfel se poate folosi cu o gamă largă de microcontrolere, consumă mai puțin de 10mA de curent.



Figura 3.4: Exemplu de afișaj a ecranului Oled SSD1306

(<https://www.electronics-lab.com/project/using-i2c-oled-display-with-arduino/>)

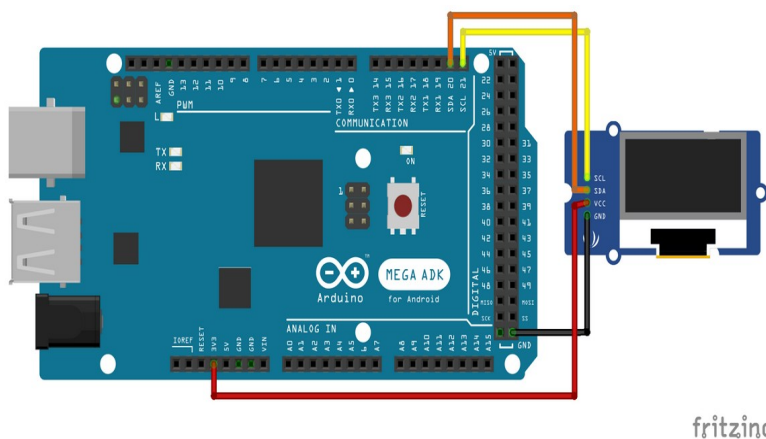


Figure 3.5: Exemplu de conectare a ecranului Oled SSD1306 la Arduino Mega

(<https://www.electronics-lab.com/project/using-i2c-oled-display-with-arduino/>)

### 3.2. Descrierea generală a arhitecturii de sistem

Sistemul de actualizarea fără fir propus în această lucrare va fi construit din 4 componente independente:

- o aplicație Angular(vezi mai multe capitolul 3.2.1) care va fi interfața cu utilizatorul, unde utilizatorul va putea încărca imaginea programului care urmează să fie distribuite către nodurile dintr-o rețea BLE, aplicația va transmite noua imagine către un micro-serviciu
- un micro-serviciu care va fi responsabil de recepționarea de noi imagini de la interfața cu utilizatorul, de asemenea este responsabil pentru transmiterea imaginii de program dacă nodul gateway va face o cerere pentru o nouă versiune, micro-serviciu realizează managementul versiunii imaginilor pe care le menține. Acest micro-serviciu folosește un server Flask(vezi mai multe capitolul 3.2.1) care este implementat în limbajul de programare Python.
- un nod gateway care va fi responsabil să ceară de la micro-serviciu noul program după care să-l stocheze local pe SD card, după care este responsabil să distribuie nodurilor rețelei noul program.
- nodurile rețelei se vor difuza un mesaj că sunt prezente, nodul gateway le va detecta, se va conecta și va starta procesul de actualizare.

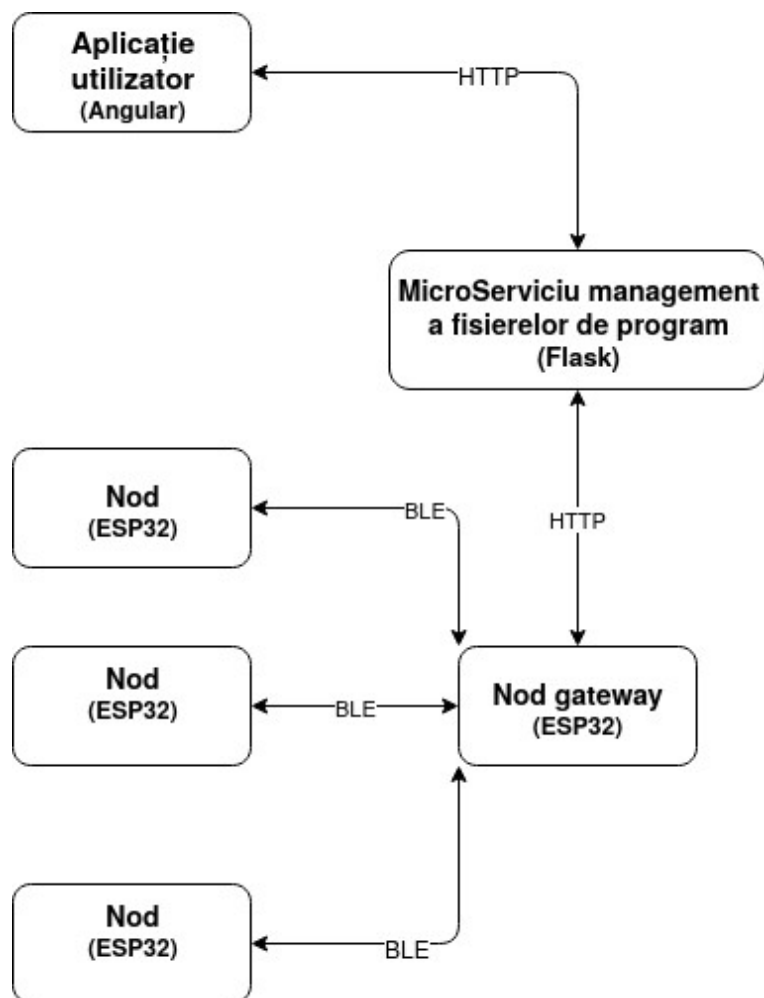


Figura 3.6: Digrama bloc a sistemului

### 3.2.1. Tehnologii folosite

Flask este o micro platformă web care este scrisă în Python. Este clasificat ca o micro-platformă deoarece nu folosește instrumente și biblioteci specifice unei platforme web, nu are un nivel abstractizare a bazei de date, validare de formulare, sau oricare altă componente preexistentă în alte platforme web. Flask suportă extensii care pot adăuga funcționalități ca și cum ele ar fi implementate în Flask în mod nativ. Există extensii precum: mapare relaționale de obiecte, validare de formulare, biblioteci de autentificare și alte instrumente existente în alte platforme.

Angular este o platformă web care este dezvoltată de compania Google care folosește limbajul de programare Typescript, este folosit pentru dezvoltarea de interfețe web fiind dotat cu multe componente grafice predefinite care permit dezvoltarea rapidă.

### 3.2.2. Descrierea comunicării interfață utilizator - micro-serviciu – nod gateway

Comunicarea interfață utilizator - micro-serviciu – nod gateway este compusă din două părți independente comunicarea interfața utilizator – micro-serviciu și micro-serviciu – nod gateway. În ambele cazuri micro-serviciul se va comporta ca server care va răspunde cererilor realizate de către celelalte componente, astfel utilizatori va introduce fișierul cu noul program în interfața cu utilizatorul după care interfața va crea o cerere de tip *HTTP POST {/upload, new\_program}* care va trimite fișierul către micro-serviciu, micro-serviciu va verifica integritatea fișierului. Fișierul trebuie să conțină un antet creat dintr-un număr specific pe 32 de biți cunoscut doar de producători și un număr pe 32 biți care va reprezenta versiunea de program. Micro-serviciul va verifica dacă antetul e cunoscut și versiunea primită trebuie să fie mai mare în comparație cu versiune conținută de server, astfel dacă aceste 2 condiții sunt îndeplinite, micro-serviciul va răspunde cu un răspuns *HTTP POST {/upload, {file\_status="Version accepted"}}* care va conține un JSON<sup>18</sup>, interfața cu utilizatorul va recepționa acest răspuns și va verifica dacă micro-serviciul a acceptat fișierul caz în care va afișa un mesaj de succes, dacă fișierul nu îndeplinește una din condițiile de integritate micro-serviciul va răspunde cu un răspuns *HTTP POST {/upload, {file\_status="Failed layout/Version old"}}* în acest caz interfața cu utilizator va afișa un mesaj de eroare. O altă parte din secvența de comunicare este comunicarea micro-serviciu – nod gateway, astfel nodul gateway trimite o cerere *HTTP GET {/version}* către micro-serviciu la fiecare 30 de secunde, aceasta va răspunde *HTTP GET {/version, latestVersion}* cu cea mai nouă versiune de program pe care o conține, nodul gateway va verifica dacă versiunea de program pe care a recepționat-o de la micro-serviciu este mai mare ca cea pe care o are o are stocată pe SD card, dacă condiția este îndeplinită atunci nodul gateway va trimite o nouă cerere *HTTP GET {/download}*, la recepționarea acestei cereri micro-serviciu va răspunde cu *HTTP GET {/download, program\_file}* care va conține noua versiune de program, nodul gateway va stoca noua versiune de program pe SD card. În cazul în care condiția de versiune nu va fi îndeplinită nodul gateway va aștepta următorul ciclu de 30 secunde. (vezi Figura 3.8)

### 3.2.3. Descriere serviciului BLE de actualizare fără fir

Serviciul BLE folosit în această lucrare nu este un serviciu standardizat, este un serviciu specific acestui concept de actualizare fără fir, care are identificatori unici universali pe 16 biți. Identificatorul unic universal folosit pentru serviciul de actualizare fără fir este 0xF000 și are

<sup>18</sup> JSON este un acronim în limba engleză pentru *JavaScript Object Notation*, și este un format de reprezentare și interschimb de date între aplicații informatice. Este un format text, inteligibil pentru oameni, utilizat pentru reprezentarea obiectelor și a altor structuri de date și este folosit în special pentru a transmite date structurate prin rețea, procesul purtând numele de serializare.



următoarele attribute:

- versiunea curentă de program, identificator universal unic:0xF100, permisiuni: citire - această caracteristică conține versiune de program pe care o rulează dispozitivul la momentul actual
- blocul de date, identificator universal unic:0xF200, permisiuni: scriere – această caracteristică este folosită pentru a stoca blocurile de date recepționate de la nodul gateway în timpul actualizării fără fir, urmând să fie scrise în memoria flash a dispozitivului. Această caracteristică are un descriptor:
  - starea blocului de date, identificator universal unic:0xF201, permisiuni: citire și scriere – în acest descriptor se salvează starea blocului de date recepționat și poate avea următoarele valori: 0x00 - *BLOCK\_STATUS\_NOT\_INIT* – blocul de date nu este folosit, 0x01 - *BLOCK\_STATUS\_INPROGRESS* – procesul de actualizare este în progres, 0x02 - *BLOCK\_STATUS\_ALLFLASHED* – o imagine de program nouă a fost complet recepționată, 0x03 - *BLOCK\_STATUS\_FLASHED\_APROVED* – această valoare este scrisă de nodul gateway dacă imaginea a fost transferată cu succes, 0x04 - *BLOCK\_STATUS\_FLASHED\_NOTAPROVED* – imaginea transferată de nodul gateway nu este completă
- dispozitiv pregătit, identificator universal unic:0xF300, permisiuni: citire – această caracteristică menține informația dacă dispozitivul este pregătit pentru startarea procesului de actualizare
- dimensiunea de fișiere așteptată, identificator universal unic:0xF400, permisiuni: scriere – caracteristică care conține dimensiunea noului program pe care dispozitivul urmează să îl recepționeze.

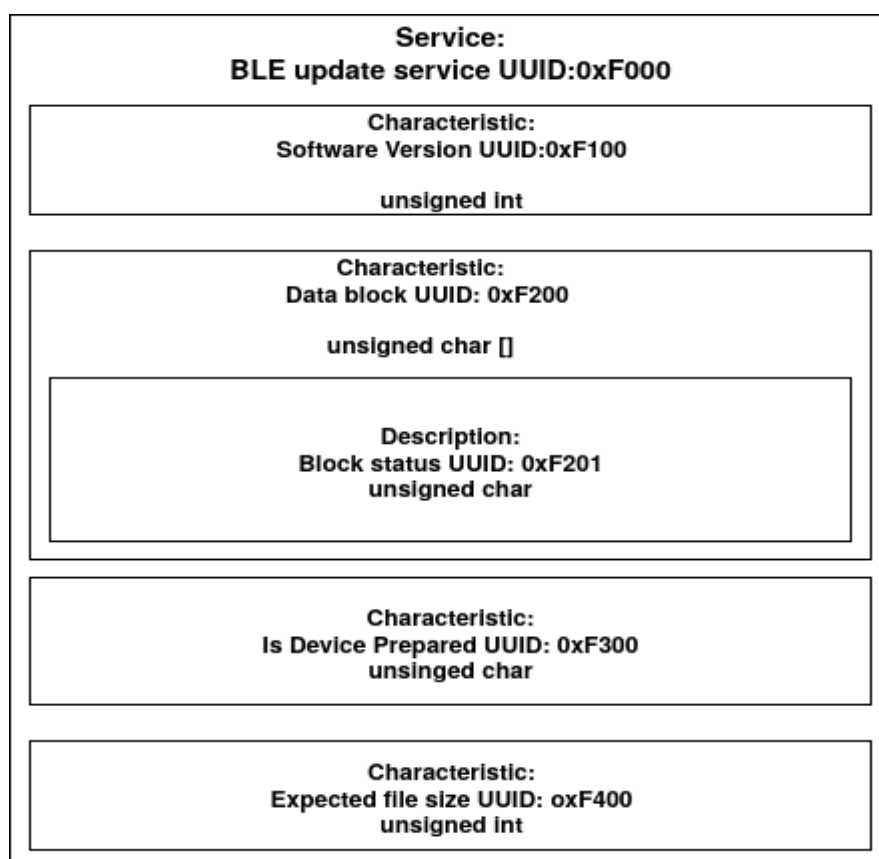


Figura 3.7: Serviciul de actualizare fără fir

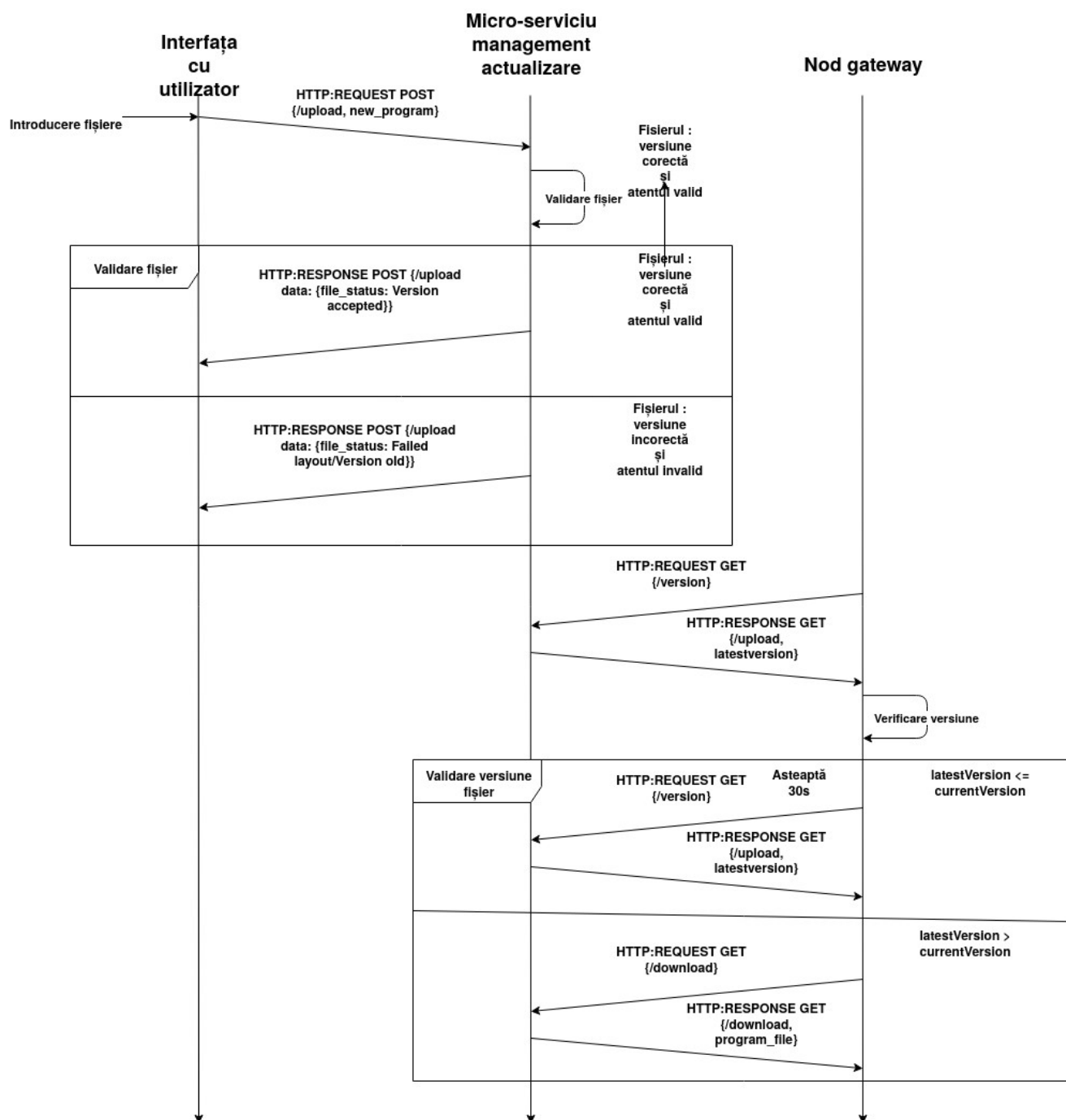


Figura 3.8: Lanțul de comunicare Interfață utilizator - micro-serviciu - nod gateway

### 3.2.4. Descrierea comunicării nod gateway – nod rețea

Comunicarea nod gateway – noduri rețea se realizează prin stiva de BLE, și folosește serviciul de actualizare descris în capitolul 3.2.4, pentru această comunicare aplicația de actualizare se va interfața cu 2 componente din stiva BLE: protocolul de atribut generic și protocolul de acces generic. Astfel conform protocolului de atribut generic (abr GATT), nodul gateway va devine nod client și nodurile din rețea vor deveni servere, iar conform protocolului de acces generic (abr.GAP) nodul gateway va deveni nod central și nodurile din rețea vor deveni

noduri periferice (vezi mai multe 2.2.1).Comunicația între nodul gateway – nodurile periferice este compusă din 2 secvențe: secvența de descoperire și secvența de actualizare. Secvența de descoperire folosește protocolul de acces generic și este compusă din următorii pași(vezi Figura 3.9):

1. La startare, nodurile periferice vor începe difuzia la pachete de publicitate, iar nodul central va starta scanarea pentru a recepționa pachetele de publicitate pentru 5 secunde.
2. Nodul central va verifica pachetele de publicitate dacă au formatul așteptat caz în care va înregistra dispozitivul în lista dispozitivelor de încredere.
3. La finalul scanării dacă lista de dispozitive de încredere nu este goală atunci nodul central va starta procesul de actualizare. În cazul în care lista de dispozitive este goală nodul central va restarta procesul de scanare.

Secvența de actualizare folosește protocolul de attribute generice și conține următorii pași(vezi Figura 3.9):

1. Nodul gateway care este client, creează o conexiune cu primul nod din lista dispozitivelor de încredere creată în secvența de descoperire care se va comporta ca un server.
2. Nodul gateway va trimite o cerere de citire pentru caracteristica cu identificatorul unic universal 0xF100, serverul va răspunde cu versiunea de program care rulează pe nod, nodul central va verifica dacă versiunea este mai veche ca versiunea de program pe care o are stocat pe SD card. În cazul în care versiunea este mai veche se va continua cu pasul 3, în caz contrar nodul central va șterge dispozitivul din lista dispozitivelor de încredere și va continua cu pasul 1.
3. Nodul gateway va trimite o cerere de citire pentru caracteristica cu identificatorul unic universal 0xF300, serverul va răspunde dacă este pregătit pentru acceptarea unei noi imagini de program, dacă serverul răspunde ca nu este pregătit se va aștepta 5 secunde după care se va repeta cererea, dacă după 3 încercări serverul nu este pregătit, clientul va trece forțat la pasul 4.
4. Nodul gateway va trimite o cerere de scriere pentru caracteristica cu identificatorul unic universal 0xF400 cu dimensiunea imaginii care urmează să fie transferată către server.
5. Nodul gateway va trimite o cerere de scriere pentru caracteristica cu identificatorul unic universal 0xF200 care va conține următorul bloc 500 de octeți din noua imagine.
6. Nodul gateway va trimite o cerere de citire pentru descriptor cu identificatorul unic universal 0xF201, serverul poate răspunde cu următoare valori:
  1. 0x01 – *BLOCK\_STATUS\_INPROGRESS* se va merge la pasul 5
  2. 0x02 – *BLOCK\_STATUS\_ALLFLASHED* se va merge la pasul 7
7. Nodul gateway va verifica dacă a transferat imaginea de program în întregime, dacă imaginea a fost transferată complet va face o cerere de scriere pentru descriptorul 0xF201 cu valoarea 0x03 – *BLOCK\_STATUS\_FLASHED\_APROVED* caz în care serverul va schimba partiția pe care o va starta programul de inițializare, se va deconecta de la client și va restarta dispozitivul, în caz că imaginea nu a fost transferată în întregime clientul va face o cerere de scriere pentru descriptorul 0xF201 cu valoarea 0x04 – *BLOCK\_STATUS\_FLASHED\_NOTAPROVED* serverul se va deconecta de la client. În ambele cazuri nodul gateway va șterge dispozitivul din lista dispozitivelor de încredere și va continua cu cu pasul 1.

### 3.2.5. Descriere organizare memorie flash pentru nodurile din rețea

O condiție importantă pentru actualizarea fără fir este ca dispozitivul trebuie să fie capabil să mențină în memoria flash 2 imagini de program, astfel la startarea programul inițial va decide care din aceste 2 programe va fi rulat de către dispozitiv. Acest lucru este valabil și

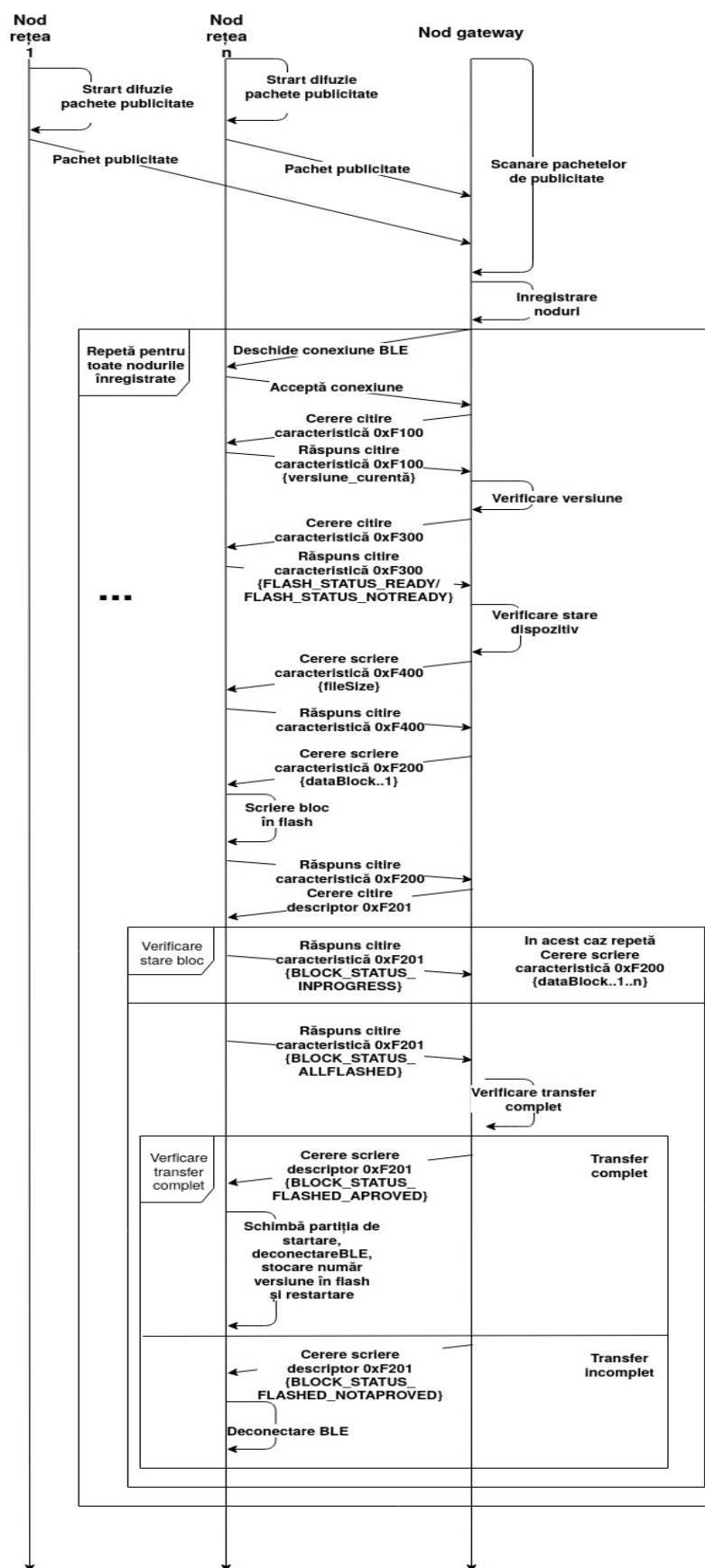


Figura 3.9: Comunicarea proces de actualizare nod gateway - nod rețea

pentru soluția propusă în această lucrare, în această lucrare se folosește startarea într-un singur nivel (vezi mai multe detalii capitolul 2.2.1). Memoria flash pentru nodurile din rețea are următoarea organizare:

- memoria flash non-volatilă - vor fi salvate datele pentru care trebuie asigurată persistență de către aplicațiile care rulează pe dispozitiv, blocul de memorie are adresa de start 0x9000 și dimensiunea 0x6000
- memoria flash de init – sunt stocate datele care le folosește programul de inițializare pentru a configura platforma hardware, blocul de memorie are adresa de start 0xf000 și dimensiunea 0x1000
- memoria flash folosită pentru a stoca informațiile despre partițiile de program și care din partiții trebuie încărcată de către programul de inițializare, blocul de memorie are adresa de start 0x210000 și dimensiunea 0x2000
- memoria flash pentru partiția 1 – menține codul programului, blocul de memorie are adresa de start 0x10000 și dimensiunea 0x100000
- memoria flash pentru partiția 2 – menține codul programului, blocul de memorie are adresa de start 0x110000 și dimensiunea 0x100000

Formatul fișierul care configurează memoria flash:

### 3.3. Proiectarea aplicațiilor software pentru nodul gateway și nodurile din rețea

Soluția de actualizare fără fir a fost implementată în limbajul de programare C++, astfel a fost nevoie de crearea unor clase de adaptare pentru interfețele propuse de ESP SDK care folosește limbajul de programare C. Nodul gateway și nodurile din rețea împart o parte comună a arhitecturii software deoarece folosesc câteva componente comune (vezi mai multe Figura 3.10):

- ESP SDK este partea ce mai importantă din arhitectură, este un pachet de utilități și API-uri la nivel de dispozitiv pentru seriile noastre de chipset-uri fără fir ESP32. Bibliotecile precompilate optimizate și bibliotecile de drivere gata de compilat reduc timpul de introducere pe piață, asigurând în același timp libertatea de personalizare. SDK-urile se compun cu instrumentul Xtensa GCC gratuit de utilizat. Acest SDK folosește sistemul de operare FreeRTOS<sup>19</sup> care este configurat pentru platforma ESP32.
- Clasa de adaptare pentru BLE (*BleAdapter*) abstractizează controlul perifericului modului radio astfel oferă interfețe pentru inițializarea și configurarea modului de BLE.
- Clasa de adaptare pentru protocolul de adaptare de acces generic (*GAPAdapter*) oferă interfețe pentru startarea difuziei a pachetelor de publicitate în cazul nodurilor periferice și startarea scanării în cazul nodurilor centrale.
- Modulul de adaptare pentru protocolul de atribut generic (*GATTAdapter*) este construit dintr-un set de clase care implementează un serviciu specific BLE, un client BLE și un

<sup>19</sup> FreeRTOS este un sistem de operare pentru dispozitive încorporate care a fost portat pentru 35 de microcontrolere, este distribuit sub licența de MIT

server BLE și oferă posibilitatea de recepție și transmitere a de cereri și răspunsuri conform specificațiilor standardului BLE. Această modul de adaptare folosește un set de clase cele mai de interes clase sunt:

- Clasa de serviciu generic BLE (*GattService*) care este generică și permite configurarea logică de caracteristici și descriptori.
- Clasa server pentru protocolul de attribute generice(*GattServer*) care permite configurare stivei BLE în modul nod server și oferă interfețe pentru realizarea operațiilor specifice unui nod server(vezi capitolul 2.2.1)
- Clasa client pentru protocolul de attribute generice(*GattClient*) care permite configurare stivei BLE în modul nod client și oferă interfețe pentru realizarea operațiilor specifice unui nod client(vezi capitolul 2.2.1)
- Clasa de serviciu client (*GattServiceClient*) care extinde clasa de serviciu generic și folosește o instanță de clasă client, astfel îmbină conceptul de client și serviciu generic BLE.
- Clasa de serviciu server (*GattServiceServer*) care extinde clasa de serviciu generic și folosește o instanță de clasă server, astfel îmbină conceptul de server și serviciu generic BLE.
- Clasa de adaptare pentru acces la memoria non-volatilă(*NvsAdapter*) – oferă interfețe pentru accesul la memoria non-volatilă.
- Clasa de control al ecranului OLED care permite actualizarea datelor afișate pe ecran, această clasă folosește o clasă de adaptare pentru protocolul de comunicare I2C.

Module specifice nodului gateway:

- Clasa de SD card (*SdcardAdapter*) oferă posibilitatea de creare, citire, ștergere, modificare a fișierele de pe modulul SdCard.
- Clasa de acces la WiFi(*WifiAdapter*) permite conectarea dispozitivului la o rețea fără fir de tip Wifi.
- Clasa de protocolului HTTP(*HttpAdater*) reprezintă un client HTTP care permite creare și transmiterea de cereri de tip *HTTP\_POST*, *HTTP\_GET*, *HTTP\_DELETE*, *HTTP\_UPDATE* și recepția răspunsurilor pentru aceste cereri.
- Clasa de aplicație(*SWL\_Gateway*) reprezintă partea de logică a aplicației și folosește clasele descrise mai sus pentru configurarea platformei hardware și realizarea procesului de actualizare, această clasă folosește 2 clase ajutătoare:
  - Clasa de actualizare fișier(*UpdateFileApp*) care este responsabilă pentru verificarea dacă micro-serviciul conține o nouă versiune de imagine de program, recepționarea și salvarea noii imagini de program pe SD card dacă este cazul.
  - Clasa de manager de dispozitive (*FlashDeviceManager*) este responsabilă pentru înregistrarea de dispozitive în lista dispozitivelor de încredere, crearea de conexiune și realizarea procesului de actualizare pentru dispozitivele respective. La rândul său acesta folosește o clasă dispozitiv(*FlashDevice*) care conține informații despre dispozitivele care urmează să fie actualizare și automatul cu stări pentru procesul de actualizare.

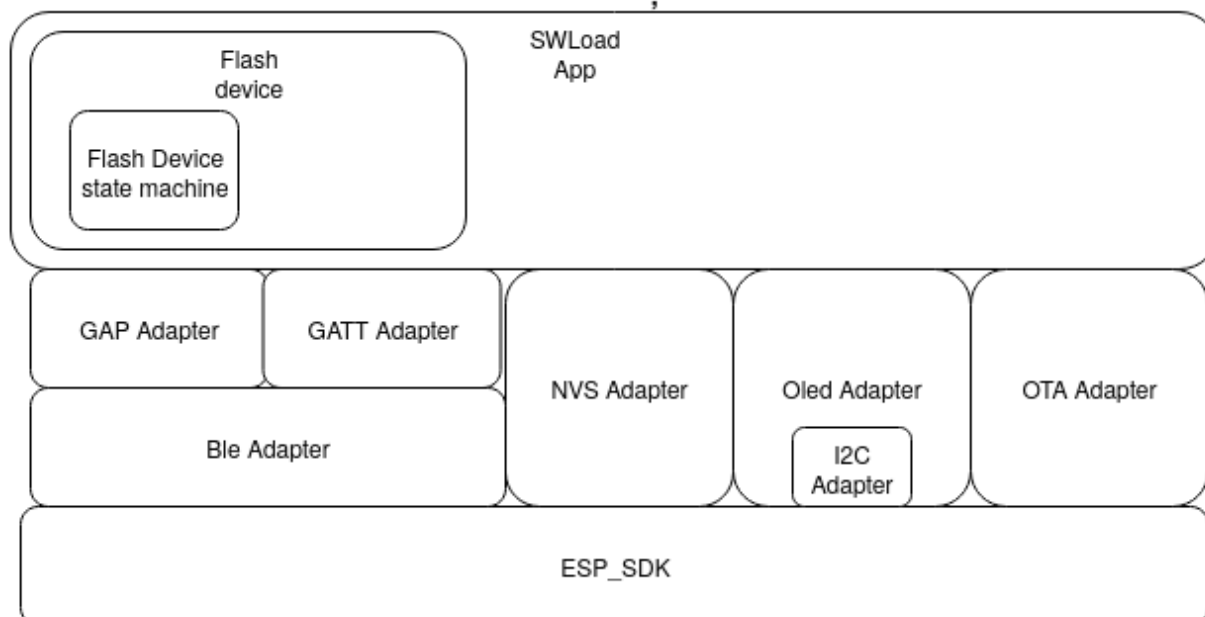
Module specifice noduri rețea:

Clasa de actualizare(*OtaAdapter*) conține informații despre partițiile care conțin codul de program, care partiție este partiția de pe care se rulează codul și pe care partiție urmează să fie scris noua imagine de program, de asemenea oferă interfețele pentru a scrie noua imagine în partiția inactivă.

Clasa de aplicație (*SWLoad*) folosește clasele de descrise mai sus pentru a configura serverul pentru protocolul de attribute generice, și conține o clasă care conține starea

dispozitivului și logica cum va avea loc actualizarea fără fir.

### Arhitectura software pentru nodurie din rețea



### Arhitectura software pentru nodul gateway

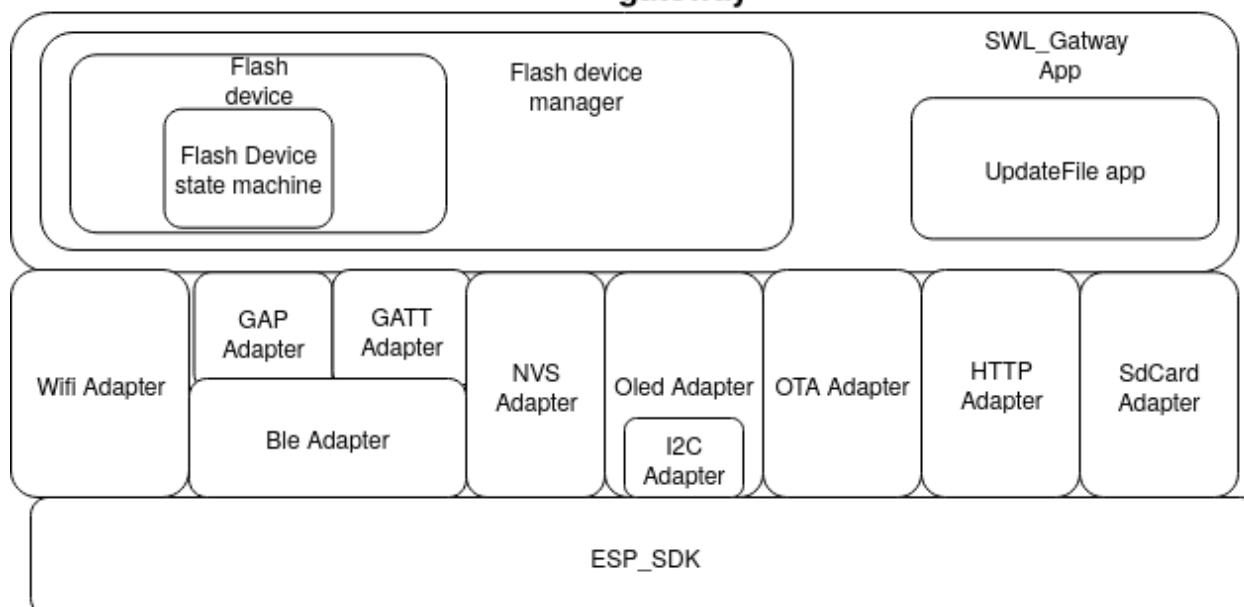


Figura 3.10: Digramele bloc cu componentele software

## Capitolul 4. Implementare

### 4.1. Implementarea interfeței utilizator

Implementarea aplicație de interfață cu utilizatorul este realizată în platforma Angular, astfel sa creat o componentă Angular care conține un câmp în care se poate introduce fișierul de program, 2 câmpuri care vor afișa dimensiunea fișierului și versiunea de program, un buton care va trimite imaginea la micro-serviciu și un câmp care va indica dacă micro-serviciul a acceptat fișierul.

Figura 4.1: Interfața cu utilizatorul

Interfața a fost creată în HTML și funcționalitatea a fost realizată în Typescript.

```
<form>
  <div><div><div class="form-group">
    <label for="file">Choose File</label>
    <input type="file"
      id="file"
      (change)="handleFileInput($event.target.files)">
    </div><br>
    <label>Major version: </label><label>{{majorVersion}}</label><br><br>
    <label>Minor version: </label><label>{{minorVersion}}</label><br><br>
    <label>Patch version: </label><label>{{patchVersion}}</label><br><br>
    <label>File name: </label><label>{{fileName}}</label><br><br>
    <label>File size: </label><label>{{fileSize}}</label><br><br>
    <label>Status: </label><label>{{status}}</label></div>
    <button type="button" (click)="UploadFile()"
      [disabled]="fileToUpload==null">Upload the file</button><br><br>
    <label>Upload status: </label> <label>{{uploadStatus}}</label>
  </div>
</form>
```



Codul pentru introducerea noului fișier

```
handleFileInput(fileBin:File)
{
  let fr = new FileReader()
  console.log(fileBin)
  if(fileBin.size != 1)
  {
    this.fileName = fileBin[0].name
    this.fileSize = fileBin[0].size + ' bytes'
    this.status = 'File accepted'
    fr.onload = () =>
    {
      var metadata = new Uint8Array(<ArrayBuffer>fr.result, 0, 8);
      console.log(metadata)
      if(metadata[0] == this.layoutFormat &&
        metadata[1] == this.layoutFormat &&
        metadata[2] == this.layoutFormat &&
        metadata[3] == this.layoutFormat)
      {
        this.majorVersion = String(metadata[4])
        this.minorVersion = String(metadata[5]<<8 | metadata[6])
        this.patchVersion = String(metadata[7])
        this.fileToUpload = fileBin[0]
      }
      else
      {
        this.status = 'Incorrect layout of the image'
      }
    }
    fr.readAsArrayBuffer(fileBin[0])
  }
  else
  {
    this.status = 'Only one file should be selected'
  }
}
```

Codul pentru transferarea fișierului cu imaginea de program către micro-serviciu:

```
UploadFile()
{ const endPoint = 'http://192.168.100.2:8000/upload/'
  const formData: FormData = new FormData()
  formData.append('NewFile', this.fileToUpload, this.fileToUpload.name);
  return this.http.post(endPoint,this.fileToUpload).subscribe(data =>{
    this.uploadStatus = data["file_status"]
    this.fileToUpload = null;},
  error=>{
    this.uploadStatus = "Request failed"
    this.fileToUpload = null;})}
```

## 4.2. Implementarea micro-serviciului de management a fișierelor de program

Micro-serviciul oferă 3 interfețe de comunicare:

- `/upload` – este folosit de interfața cu utilizatorul pentru a încărca un nouă imagine de program

```
@app.route("/upload/", methods=['POST'])
def uploadFile ():
    data = request.stream.read()
    file_status = ""
    if (data[0] == layoutFormat) and (data[1] == layoutFormat) and (data[2] == layoutFormat)
    and (data[3] == layoutFormat):
        print ("Layout accepted")
        version = (data[4] << 24) | (data[5] << 16) | (data[6] << 8) | data[7]
        with open("severState.json", 'r') as file:
            serverState = json.load(file)
            if(serverState["latestVersion"] < version):
                print("Version accepted")
                with(open(str(version) + ".bin", 'wb')) as latest:
                    latest.write(data[8:])
                file_status = "Version accepted"
                serverState["latestVersion"] = version
                with open("severState.json", 'w') as file:
                    json.dump(serverState, file)
            else:
                file_status = "Version old"
                print("Version is too old")
        else:
            print ("Layout failed")
            file_status = "Failed layout"
    return jsonify(file_status = file_status)
```

- `/version` – returnează versiunea de program la care sistemul trebuie să fie actualizat

```
@app.route("/version/")
def getLatestVersion ():
    with open("severState.json", 'r') as file:
        serverState = json.load(file)
        resp = Response(str(serverState["latestVersion"]))
        resp.headers['type'] = 'version'
    return resp
```

- `/download` – returnează fișierul cu imagine de program

```
@app.route("/download/")
def downloadFile ():
    with open("severState.json", 'r') as file:
        serverState = json.load(file)
        path = str(serverState["latestVersion"]) + ".bin"
    return send_file(path, as_attachment=True)
```

### 4.3. Implementarea aplicațiilor pentru nodul gateway și pentru nodurile din rețea.

#### 4.3.1. Șabloane de programare folosite

Implementarea aplicațiilor folosește 3 șabloane de programare:

- Șablonul creațional *Singleton* – acest șablon de programare restricționează crearea de instanțe de clase și asigură ca o singură instanță de clasă a fost creată, acest șablon a fost folosit pentru clasele care abstractizează resursele unice ale platformei hardware, astfel se poate face managementul de acces la resursă(ex. Clase: NvsAdapter, BleAdapter, WifiAdapter, HttpClient).
- Șablonul structural *Adaptor* – acest șablon de programare realizează conversia unei interfețe într-o altă interfață, acest lucru are loc când interfețele pe care le folosește o aplicației sunt diferite de interfețele oferite de sistem.
- Șablonul comportamental *Observer* – acest șablon este folosit când o abstracție are 2 aspecte, încapsulând aceste aspecte în obiecte diferite permite reutilizarea lor în mod independent, când un obiect necesită schimbarea altor obiecte și nu știe cât de multe trebuie schimbate, când un obiect ar trebui să notifice pe altele, fără să știe cine sunt acestea.

Un exemplu unde au fost aplicate toate 3 șabloane de programare este clasa „GattServer” care implementează serverul protocolului de atribut generice:

- Șablonul *Singleton* este folosit deoarece un server pentru protocolul de atribut generice este unic pe dispozitiv, această instanță de server poate fi folosită de mai multe aplicații care au nevoie de un server, în acest obiect are loc sincronizarea accesului concurent la server. Implementarea folosește o instanță statică a clasei server, este folosită această abordare deoarece nu se dorește alocarea dinamică a obiectului, și o metodă statică care va returna o referință la obiectul static.

```
static GattServer gattServer;
GattServer& GattServer::getInstance(void)
{
    if(gattsEventHandler == NULL)
    {
        gattsEventHandler = &gatts_callback;
    }
    return GattServer::gattServer;
}
```

- Șablonul *Adaptor* este folosit pentru încapsularea interfețelor oferite de mediul de dezvoltare a platformei ESP în clase specifice, astfel se pot proiecta aplicațiile în paradigma orientată pe obiect. Un exemplu este funcția care startează un serviciu în serverul protocolului de atribut generice:

```
esp_err_t GattServer::StartService(uint16_t service_handle){
    esp_err_t ret = esp_ble_gatts_start_service(service_handle);
    if(ret != ESP_OK){
        ESP_LOGI(GATT_SERVICE_NAME, "Service can't be started. Failed with
error: %d", ret);
    }
    return ret;
}
```

- Șablonul *Observer* este folosit deoarece serverul protocolului de atribute generice este independent de logica de aplicație, astfel dacă o aplicație dorește să folosească datele pe care serverul le obține din rețea trebuie să se înregistreze ca un observator la server și serverul va notifica aplicația la recepționarea datelor.

Codul clasei „GattServer”:

```
extern etl::ifunction<GattsEventInfo>* gattsEventHandler;
namespace BLE
{
    typedef etl::observer<GattsEventInfo> Gatts_Observer;
    class GattServer:public etl::observable<Gatts_Observer, GATTS_MAX_OBSERVERS>
    {
    public:
        BleAdapter& bleAdapter = BleAdapter::getInstance();
        static GattServer gattServer;
        GattServer();
        esp_err_t Init(void);
        ~GattServer();
        void RegisterApp(uint16_t appId);
        esp_err_t OpenConnection(esp_gatt_if_t gatts_if, esp_bd_addr_t remote_bda, bool
is_direct);
        esp_err_t RegisterService(esp_gatt_if_t gatts_if, GattService *service);
        esp_err_t SendResponse(esp_gatt_if_t gatts_if, uint16_t conn_id, uint32_t trans_id,
            esp_gatt_status_t status, esp_gatt_rsp_t *rsp);
        esp_err_t StartService(uint16_t service_handle);
        esp_err_t SendIndication(esp_gatt_if_t gatts_if, uint16_t conn_id, uint16_t
attr_handle,
            uint16_t value_len, uint8_t *value, bool need_confirm);
        esp_err_t AddCharDescr(uint16_t service_handle,
            esp_bt_uuid_t *descr_uuid,
            esp_gatt_perm_t perm, esp_attr_value_t *char_descr_val,
            esp_attr_control_t *control);
        esp_err_t AddChar(uint16_t service_handle, esp_bt_uuid_t *char_uuid,
            esp_gatt_perm_t perm, esp_gatt_char_prop_t property, esp_attr_value_t
*char_val,
            esp_attr_control_t *control);
        void gattsCallback(GattsEventInfo info);
        static GattServer& getInstance(void);
    };
}
```

#### 4.3.2. Implementarea automatelor cu stări finite pentru procesul de actualizare

Implementarea procesului de actualizare fără fir implică câte un automat cu stări finite pentru nodul gateway cât și pentru nodurile din rețea. Automatul cu stări a nodului gateway este alcătuit din următoarele stări:

- Deconectat – această stare semnifică că dispozitivul care urmează să fie actualizat a fost înregistrat în lista de încredere, însă clientul protocolului de atribute de pe nodul gateway

nu s-a conectat.

- Conectat – clientul protocolului de attribute de pe nodul gateway s-a conectat la nodul din rețea.
- Serviciu descoperit – nodul gateway a reușit să valideze serviciul de pe nodul din rețea pe care urmează să-l actualizeze.
- Versiune recepționată – nodul gateway a recepționat o versiune validă de la nodul din rețea.
- Transfer blocuri date – nodul gateway a început transferul blocurilor de date către nodul din rețea cu noua versiune de program.
- Actualizare completă – nodul gateway a transmis fișierul complet către nodul din rețea.
- Șterge conexiunea – nodul gateway setează flagul ca nodul să fie șters din lista dispozitivelor de încredere de către managerul de dispozitive.

Acest automat de stări acceptă următoarele evenimente:

- Conexiune - se generează când clientul protocolului de attribute de pe nodul gateway a reușit conexiune la nodul din rețea.
- Serviciu descoperit – se generează când nodul gateway validează că serverul are un serviciu de BLE care are formatul corect pentru a putea starta procesul de actualizare fără fir.
- Recepționare versiune – se generează când clientul a recepționat răspunsul la cererea de versiune.
- Dispozitiv nepregătit – se generează când nodul din rețea răspunde că nu se află într-o stare sigură pentru startarea procesului de actualizare.
- Dispozitiv pregătit – se generează când nodul din rețea răspunde se află într-o stare sigură pentru startarea procesului de actualizare.
- Versiune prezentă – se generează când nodul care se dorește actualizat are deja cea mai nouă versiune de program.
- Următorul bloc – se generează când nodul client anunță că este pregătit pentru recepționarea următorului bloc de date din fișierul cu noua imagine de program.
- Fișier complet – se generează când nodul gateway a transferat întregul fișier.
- Dispozitiv acceptat fișier – se generează când nodul din rețea răspunde dacă consideră că a recepționat un fișier integru și valid.
- Timp expirat – se generează când răspunsul la o cerere al clientului durează prea mult.

Un automat cu stări finite a fost implementat de asemenea și pentru nodul din rețea care are următoarele stări:

- Deconectat – serverul nu are nici un client conectat.
- Conectat – un client pentru protocolul de attribute generice s-a conectat la server.
- Recepționare fișier – nodul din rețea recepționează blocurile pe care nodul gateway le transmite.

Automatul de stări finite a nodurilor din rețea acceptă următoarele evenimente:

- Conexiune – se generează când un client al protocolului de attribute generice se conectează la server.
- Startare actualizare – se generează când nodul gateway transferă primul bloc de date din noul program.
- Recepționare bloc – se generează când nodul din rețea recepționează un nou bloc consecutiv din noul program.
- Acceptare fișier – se generează când noul program a fost recepționat integral de la nodul gateway.
- Decontare – clientul s-a deconectat de la server.

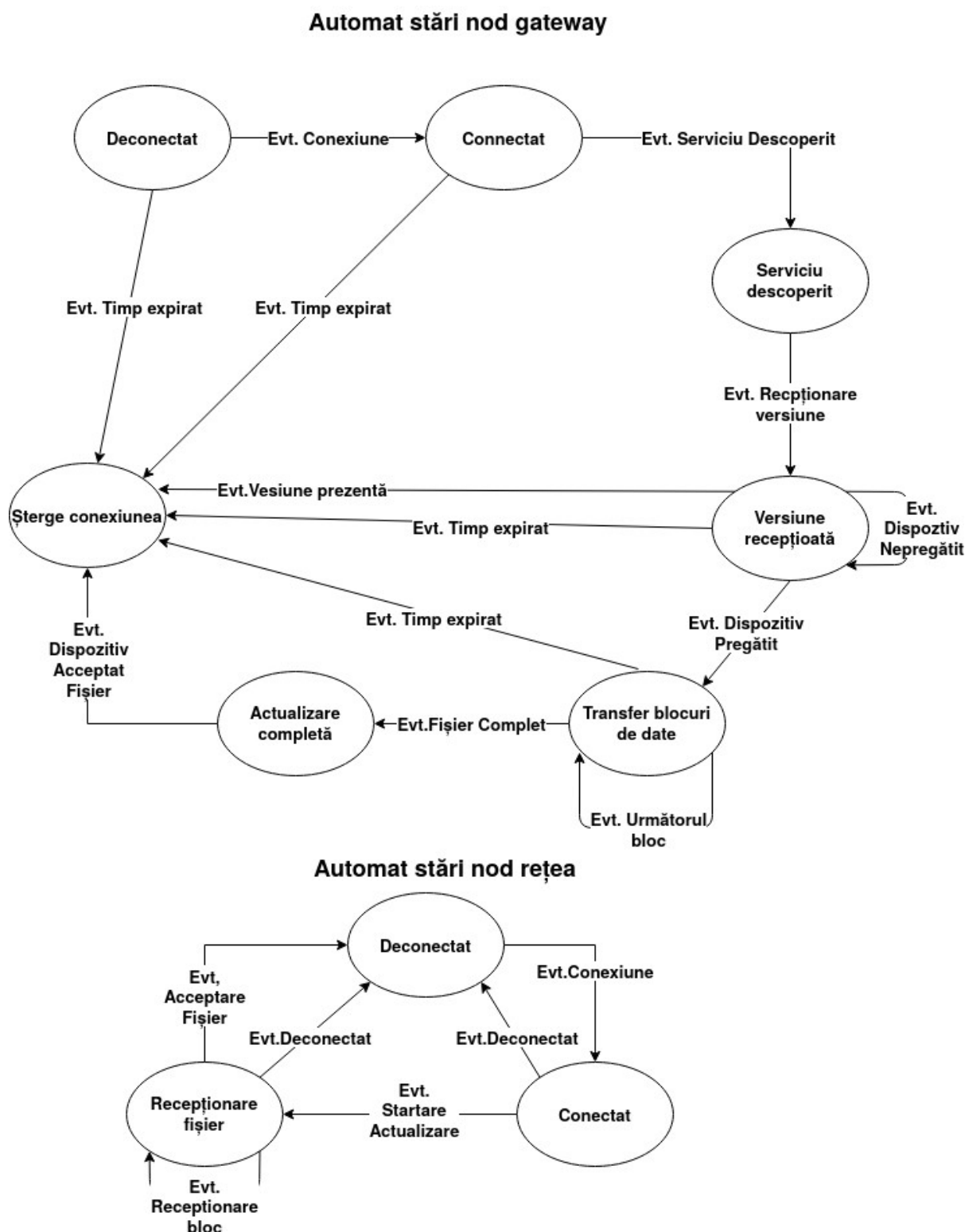


Figura 4.1: Automate stări pentru procesul de actualizare

Fiecare stare este implementată într-o clasă care este caracterizată printr-un identificator numeric și prin evenimentele pe care le acceptă, astfel pentru fiecare eveniment acceptat de către stare se supra încarcă o metodă care primește ca parametru evenimentul respectiv. Evenimentele de asemenea sunt implementate într-o clasă, ca și în cazul stărilor evenimentele

sunt caracterizate printr-un identificator numeric, clasele specifice evenimentelor pot conține și câmpuri cu informații suplimentare despre eveniment.

Exemplu de cod pentru o clasă stare:

```
class SendBlockState : public etl::fsm_state<FlashDevice,
SendBlockState, StateId::SEND_BLOCK,
NextBlockEvt, TimeoutEvt, DeleteEvt,
CheckBlockStatusEvt,
FlashedCompleteEvt>{
public:
//*****
fsm_state_id_t on_event(etl::imessage_router& sender, const NextBlockEvt& event);
fsm_state_id_t on_event(etl::imessage_router& sender, const TimeoutEvt& event);
fsm_state_id_t on_event(etl::imessage_router& sender, const CheckBlockStatusEvt& event);
fsm_state_id_t on_event(etl::imessage_router& sender, const DeleteEvt& event);
fsm_state_id_t on_event(etl::imessage_router& sender, const FlashedCompleteEvt& event);
//*****
fsm_state_id_t on_event_unknown(etl::imessage_router& sender, const etl::imessage& event);
};
```

Exemplu de cod pentru o clasă eveniment:

```
class VersionRespEvt : public message<EventId::VERSION_RESP>
{
public:
uint32_t version;
VersionRespEvt(uint32_t version):
version(version){}
};
```

Exemplu cod pentru implementarea unei metode de recepționare eveniment:

```
fsm_state_id_t SendBlockState::on_event(etl::imessage_router& sender, const
NextBlockEvt& event)
{
FlashDevice &dev = this->get_fsm_context();
dev.SendBlock();
return StateId::SEND_BLOCK;
}
```

### 4.3.3. Implementarea aplicațiilor

Implementarea aplicațiilor de actualizare pe platforma ESP32 folosește o bibliotecă de șabloane pentru sisteme încorporate<sup>20</sup> care este o reimplementare a bibliotecii de șabloane standard prezentă în C++. Biblioteca de șabloane standard oferită de C++ folosește alocarea dinamică a memoriei care poate deveni o problemă pentru sistemele încorporate din cauza fragmentării de memorie, astfel această bibliotecă de șabloane pentru sisteme încorporate oferă colecțiile specifice bibliotecii de șabloane standard însă aceste colecții folosesc alocare statică de memorie un exemplu de declarare vector este lista dispozitivelor de încredere:

```
#define MAX_BLE_DEVICES 5u
etl::vector<FlashDevice, MAX_BLE_DEVICES> listOfDevice;
```

<sup>20</sup> <https://www.etlcpp.com/>



La startarea platformei ESP32 au loc următorii pași:

1. Se creează clasele care abstractizează resursele hardware care urmează să fie folosite de aplicații.

```
#define SWL_APP 0x0001u
#define PIN_NUM_MISO UINT8_C(2)
#define PIN_NUM_MOSI UINT8_C(15)
#define PIN_NUM_CLK  UINT8_C(14)
#define PIN_NUM_CS   UINT8_C(13)
#define PIN_NUM_I2C_SDA  UINT8_C(18)
#define PIN_NUM_I2C_SCL  UINT8_C(19)
// Init global classes
OLED oled((gpio_num_t)PIN_NUM_I2C_SCL,
(gpio_num_t)PIN_NUM_I2C_SDA,SSD1306_128x64);
SWL_OLED::SwlOled swlOled(oled);
SDCARD::SdCard sdcard((gpio_num_t)PIN_NUM_CS,(gpio_num_t)PIN_NUM_CLK,
(gpio_num_t)PIN_NUM_MOSI,(gpio_num_t)PIN_NUM_MISO);
VersionManager versMng;
UpdateFileApp updateFile(sdcard,versMng,swlOled);
SWL_GATEWAY::SwlGateway swlApp(SWL_APP,sdcard,updateFile,versMng,swlOled); //
@suppress("Abstract class cannot be instantiated")
```

2. Se creează obiectele aplicație, fiecare aplicație este abstractizată printr-un obiect care oferă o interfață care este folosită pentru a inițializa aplicația și o interfață care este apelată într-o funcție ciclică.
3. Se configurează sistemul de operare să apeleze ciclic o funcție sau mai multe la diferite intervale de timp.

```
swlApp.Init();
if( xTaskCreate(
    vTaskCode_10ms,    // Function that implements the task.
    "Task_10ms",      // Text name for the task.
    STACK_SIZE,       // Stack size in bytes, not words.
    ( void * ) NULL,   // Parameter passed into the task.
    1, // Priority at which the task is created.
    &xHandle_10ms ) != pdPASS // Variable to hold the task's data structure.
){
    ESP_LOGE("main","Task was not created correctly");
    esp_restart();}
if( xTaskCreate(vTaskCode_100ms,    // Function that implements the task.
    "Task_100ms",      // Text name for the task.
    STACK_SIZE,       // Stack size in bytes, not words.
    ( void * ) NULL,   // Parameter passed into the task.
    1, // Priority at which the task is created.
    &xHandle_100ms ) != pdPASS // Variable to hold the task's data structure.
){
    ESP_LOGE("main","Task was not created correctly");
    esp_restart();}
```

4. În funcția ciclică se vor apela interfețele oferite de obiectele care abstractizează resursele

hardware și interfețele oferite de obiectele aplicație.

```
// Function that implements the task being created.
extern "C" void vTaskCode_100ms( void * pvParameters )
{
    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount ();
    uint32_t free_heap = 0;
    while(1)
    {
        vTaskDelayUntil(&xLastWakeTime, TASK_CYCLE_100ms/portTICK_PERIOD_MS );
        free_heap = esp_get_free_heap_size();
        swlApp.TaskMain_100ms();
        if(free_heap < 1024)
        {
            ESP_LOGE("main", "low memory %d", free_heap);
        }
    }
}
```

Clasa pentru aplicația de actualizare fără fir pentru nodul gateway:

```
namespace SWL_GATEWAY{
    class SwlGateway:public Gap_Observer,public Http_Observer, public Gattc_Observer{
    private:
        bool isScanningRunning = false;
        bool isFlashingInProgress = false;
        void processScanData(esp_ble_gap_cb_param_t *param);
        void processRawData(uint8_t *data, uint8_t len, FlashDevice *flashDev);
        void buildManufactureDataBuffer(uint8_t *data, uint8_t &len);
        void triggerConnetion(void);
    public:
        uint16_t appId;
        SdCard &sd;
        UpdateFileApp& updateFileApp;
        VersionManager &vers;
        SWL_OLED::SwlOled &swlOled;
        GapAdapter &gapAdapter = GapAdapter::getInstance();
        GattClient &gattc = GattClient::getInstance();
        WifiAdapter &wifiAdapter = WifiAdapter::getInstance();
        FlashDeviceManager flashDeviceManager;
        int16_t gattc_if = 0;
        SwlGateway(uint16_t appId,SdCard &sd,UpdateFileApp&
updateFileApp,VersionManager &vers,SWL_OLED::SwlOled &swlOled);
        ~SwlGateway();
        void Init(void);
        void notification(GapEventInfo info);
        void notification(GattcEventInfo info);
        void notification(esp_http_client_event_t * evt);
        void TaskMain_100ms(void);
    };
}
```

Clasa pentru aplicația de actualizare fără fir pentru nodurile din rețea:

```
namespace SWLOAD
{
    class SwLoad:public Gap_Observer, public Gatts_Observer
    {
#define FLASH_STATE_MACHINE 0u
        typedef enum
        {
            SwLoadAppState_IDLE,
            SwLoadAppState_SCANNING,
            SwLoadAppState_WORKINING,
        } SwLoadAppStateType;
    private:
        void processRawData(uint8_t *data, uint8_t len, FlashDevice *device);
    public:
        uint16_t appId;
        SWL_OLED::SwlOled &swlOled;
        FlashDevice flashDevice;
        uint32_t currentVersion;
        SwLoadAppStateType state;
        uint16_t gatts_if;
        GapAdapter &gapAdapter = GapAdapter::getInstance();
        NVSAdapter &nvsAdapter = NVSAdapter::getInstance();
        GattServer &gattServer = GattServer::getInstance();
        SwLoad(uint16_t appId, SWL_OLED::SwlOled &swlOled);
        ~SwLoad();
        void Init(void);
        void notification(GapEventInfo info);
        void notification(GattsEventInfo info);
    };
}
```

## Capitolul 5. Testarea aplicației

Testarea soluției a fost realizată folosind 3 platforme hardware ESP32, 1 platformă este folosită pe post de nod gateway având conectat un SD card și un ecran Oled SSD1306 pe care se va afișa starea nodului, 2 platforme hardware vor fi folosite pe post de noduri din rețea care vor avea la rândul lor atașat un ecran Oled SSD1306. Micro-serviciul și interfața utilizator sunt rulate. Fiecare componentă a sistemului poate fi monitorizată pentru a verifica funcționalitatea corectă a sistemului, astfel micro-serviciul va afișa toate cererile HTTP care sunt transferate și va fi afișat dacă micro-serviciul a acceptat cererea (vezi Figura 5.3). Aplicația cu utilizatorul folosește un câmp care afișează mesaje cu starea aplicație și posibilele erori. Aplicațiile de pe nodul gateway și de pe nodurile din rețea folosesc interfața de comunicare UART pentru a afișa informații despre nod precum:

- Serverul sau clientul protocolului de attribute generice a fost start cu succes.
- Starea în care se află automatele de stări și evenimentele care sunt generate de către sistem.
- Cererile specifice stivei BLE și răspunsurile la aceste cereri. (etc)

```

File Edit View Search Terminal Help
1 (1011826) FlashDevice: File size read 500
1 (1011826) FlashDevice: File size current read 290000
1 (1011905) GATT: GATT Event: 4
1 (1011905) GATT: GATT interface: 3
1 (1011964) GATT: GATT Event: 8
1 (1011965) GATT: GATT interface: 3
1 (1011965) SLOW: Read desc status 0
Read desc conn_id: 0
Read desc hdl: 45
Read desc value_len 1
1 (1011972) SLOW: 01
1 (1011975) SD_CARD: Opening file /sdcard/16778034.bin
1 (1011992) SD_CARD: Read bytes 500
1 (1011992) FlashDevice: File size read 500
1 (1012143) SLOW: Read desc status 0
Read desc conn_id: 0
Read desc hdl: 45
Read desc value_len 1
1 (1012151) SLOW: 01
1 (1012154) SD_CARD: Opening file /sdcard/16778034.bin
1 (1012172) SD_CARD: Read bytes 500
1 (1012172) FlashDevice: File size read 500
1 (1012172) FlashDevice: File size current read 291000
1 (1012248) GATT: GATT Event: 4
1 (1012248) GATT: GATT interface: 3
1 (1012270) GATT: GATT Event: 8
1 (1012280) GATT: GATT interface: 3
1 (1012280) SLOW: Read desc status 0
Read desc conn_id: 0
Read desc hdl: 45
Read desc value_len 1
1 (1012287) SLOW: 01
1 (1012290) SD_CARD: Opening file /sdcard/16778034.bin
1 (1012310) SD_CARD: Read bytes 500
1 (1012310) FlashDevice: File size read 500
1 (1012310) FlashDevice: File size current read 291500
1 (1012399) GATT: GATT Event: 4
1 (1012400) GATT: GATT interface: 3
1 (1012474) GATT: GATT Event: 8
1 (1012475) GATT: GATT interface: 3
1 (1012475) SLOW: Read desc status 0
Read desc conn_id: 0
Read desc hdl: 45
Read desc value_len 1
1 (1012482) SLOW: 01
1 (1012485) SD_CARD: Opening file /sdcard/16778034.bin
1 (1012503) SD_CARD: Read bytes 500
1 (1012503) FlashDevice: File size read 500
1 (1012503) FlashDevice: File size current read 292000

File Edit View Search Terminal Help
1 (318880) SWLGW: ESP_GATTS_RESPONSE_EVT, status 0, handle 42
1 (318950) GATTS: GATTS Event: 15
1 (318951) GATTS: GATTS interface: 3
0 (318951) OTA: Failed to end the update incorrect state
1 (319059) BLE_GAP: GAP Event: 6
1 (319059) BLE_GAP: Status 0
1 (324534) GATTS: GATTS Event: 14
1 (324535) GATTS: GATTS interface: 3
1 (324535) SWLGW: ESP_GATTS_CONNECT_EVT, connection id 0
1 (324621) GATTS: GATTS Event: 4
1 (324623) GATTS: GATTS interface: 3
1 (325774) GATTS: GATTS Event: 1
1 (325776) GATTS: GATTS interface: 3
1 (325849) FlashDevice: 32 03 00 01
1 (325849) GATT_SERVICE: 32 03 00 01
1 (325851) GATTS: GATTS Event: 21
1 (325851) GATTS: GATTS interface: 3
1 (325855) SWLGW: ESP_GATTS_RESPONSE_EVT, status 0, handle 42
1 (325955) GATTS: GATTS Event: 15
1 (325956) GATTS: GATTS interface: 3
0 (325956) OTA: Failed to end the update incorrect state
1 (326064) BLE_GAP: GAP Event: 6
1 (326064) BLE_GAP: Status 0

cu -l ttyUSB2 -s 115200
File Edit View Search Terminal Help
1 (116845) GATTS: GATTS interface: 3
1 (116849) SWLGW: ESP_GATTS_RESPONSE_EVT, status 0, handle 44
1 (116889) GATTS: GATTS Event: 1
1 (116891) GATTS: GATTS interface: 3
1 (116891) GATT_SERVICE: 01
1 (116893) GATTS: GATTS Event: 21
1 (116895) GATTS: GATTS interface: 3
1 (116900) SWLGW: ESP_GATTS_RESPONSE_EVT, status 0, handle 45
1 (116903) GATTS: GATTS Event: 2
1 (116906) GATTS: GATTS interface: 3
1 (116997) FlashDevice: Received len 500
1 (116997) FlashDevice: Received total len 291500
1 (116997) FlashDevice: File size 657456
1 (117000) FlashDevice: Try to send resp on conn id 0
1 (117000) GATTS: GATTS Event: 21
1 (117010) GATTS: GATTS interface: 3
1 (117014) SWLGW: ESP_GATTS_RESPONSE_EVT, status 0, handle 44
1 (117069) GATTS: GATTS Event: 1
1 (117071) GATTS: GATTS interface: 3
1 (117071) GATT_SERVICE: 01
1 (117073) GATTS: GATTS Event: 21
1 (117075) GATTS: GATTS interface: 3
1 (117079) SWLGW: ESP_GATTS_RESPONSE_EVT, status 0, handle 45

```

Figura 5.1: Monitorizarea stărilor pentru nodul gateway și nodurile din rețea

Pe interfața de comunicare UART se afișează foarte multă informație care este foarte folositoare în procesul de depănare când se încearcă detectarea cauzei unei erori. Pentru a urmări starea sistemului într-un mod mai prietenos s-au folosit ecranele Oled care afișează informații despre starea internă a fiecărui dispozitiv. Ecranul Oled specific nodurilor de rețea afișează următoarele informații:

- Acțiunea pe care o realizează nodul în momentul actual precum: difuzie de pachete de publicitate, conectare la client sau actualizare activă.
- Progresul procesului de actualizare dacă acesta a început.
- Ultimul identificator universal unic a unei caracteristici sau a unui descriptor folosit de către serviciul stivei de comunicare BLE.
- Informații adiționale despre starea curentă a dispozitivului.
- Versiunea de program care este rulată de către dispozitiv
- Partiția de memorie flash care este activă

Ecranul Oled folosit de nodul gateway afișează următoarele informații:

- Starea conexiunii cu micro-serviciul și acțiunile pe care nodul gateway le realizează către micro-serviciu.
- Progresul transferului fișierului cu noua imagine de program către nodul gateway de la micro-serviciu.
- Ultima versiune a fișierului pe care nodul gateway o are stocată în SD card.
- Starea conexiunii nodului gateway cu nodurile din rețea.
- Numărul de dispozitive din rețeaua BLE care au fost descoperite de nodul gateway și înregistrate în lista dispozitivelor de încredere, de asemenea câte din dispozitivele înregistrate au fost actualizate.
- Progresul procesului de actualizare pentru un dispozitiv din rețea.

Oled SSD1306 Nod gateway	Oled SSD1306 Nod rețea
<b>UpApp:Status:</b> <b>UpApp:Progress:</b> <b>UpApp:Version:</b>	<b>Status:</b> <b>Progress:</b> <b>Uuid used:</b>
<b>GwApp:Status:</b> <b>GwApp:DevInfo:</b> <b>GwApp:Progress:</b>	<b>Log:</b> <b>Version:</b> <b>Partition:</b>

Figura 5.2: Informație afișată de ecranule Oled pentru nodul gateway și pentru nodurile din rețea

```

192.168.100.14 - - [13/Jan/2020 23:00:51] "GET /version/ HTTP/1.1" 200 -
192.168.100.14 - - [13/Jan/2020 23:01:04] "GET /version/ HTTP/1.1" 200 -
192.168.100.14 - - [13/Jan/2020 23:01:16] "GET /version/ HTTP/1.1" 200 -
192.168.100.2 - - [13/Jan/2020 23:01:17] "OPTIONS /upload/ HTTP/1.1" 200 -
Layout accepted
Version accepted
192.168.100.2 - - [13/Jan/2020 23:01:17] "POST /upload/ HTTP/1.1" 200 -
192.168.100.14 - - [13/Jan/2020 23:01:30] "GET /version/ HTTP/1.1" 200 -
192.168.100.14 - - [13/Jan/2020 23:01:31] "GET /download/ HTTP/1.1" 200 -
    
```

Figura 5.3: Monitorizare cereri HTTP la micro-serviciu

Componentele sistemului au fost implementate și testate independent, astfel o parte dificilă a soluției a fost conectarea și interfațarea componentelor, astfel nodul gateway conține partea cea mai complexă a proiectului, aplicația de recepționare a fișierului cu noua imagine de la micro-serviciu și aplicația de transferare a noii imagini de la nodul gateway folosesc ambele accesul la SD card , astfel aplicația de recepționare a noii imagini de la micro-serviciu este inactivă dacă nodul gateway pornește procesul de actualizare fără fir cu un nod din rețea.

## Concluzie

Soluția prototip prezentată în această lucrare pentru actualizarea fără fir propune o soluție completă unde factorul uman are un impact foarte mic asupra procesul de actualizare. Factorul uman trebuie să propună o nouă imagine de program validă, după care sistemul de actualizare va starta procesul de actualizare independent. Lipsa factorului uman aduce însă o complexitate crescută sistemului deoarece este foarte puțin tolerabil la erori iar în lipsa factorului uman nu va exista cine să restarteze procesul de actualizare sau să rezolve problemele apărute pe parcursul actualizării. Un sistem de actualizare fără fir va trebui să implementeze mecanisme complexe de detectare a erorilor și chiar mecanisme de recuperare în urma unei erori, această complexitate crește și mai mult având în vedere că aceste mecanisme vor fi implementate pe platforme ce au putere de procesare și resurse limitate.

Implementarea propusă în această lucrare propune o soluție de actualizare fără fir pentru o rețea de dispozitive ce folosesc stiva de comunicare BLE. Sistemul este compus din 4 componente: interfața cu utilizatorul care va fi punctul de intrare pentru noile imagini de program, micro-serviciu care face managementul fișierelor cu imaginile de program, un nod gateway care va fi responsabil pentru recepționarea imaginilor de program de la micro-serviciu și transferarea programelor noi către nodurile din rețea, nodurile rețelei. Nodul gateway folosește mai multe stive de comunicare în paralel precum: stiva BLE, stiva de WiFi și stiva TCP/IP. Platforma hardware folosită pentru această soluție este ESP32, o platforma foarte versatilă care are integrate modulele radio pentru protocolul WiFi și pentru protocolul BLE, astfel fiind o platformă perfectă pentru concepte ce au nevoie de o conectivitate foarte bogată.

O posibilă aplicare a conceptului de actualizare fără fir este domeniul senzorilor fără fir și domeniul IoT, folosind acest concept noi produse vor putea fi livrate mai rapid pe piața cu o funcționalitate de bază, ulterior adăugând noi aplicații și îmbunătățind aplicațiile existente în funcție de nevoile domeniului sau a clientului. Conceptul de actualizare fără fir crește flexibilitate sistemului, scade costul de mentinere și oferă posibilitate de extindere.





**BIBLIOGRAFIE**

- [1] IoT Firmware Management:Over the Air Firmware Management for Constrained Devices using IPv6 over BLE Manas Marawaha, B.Tech Electronics and Communication 2017
- [2] UpdaThing: A secure and open firmware updatesystem for Internet of Things devices Tomás Alexandre Diniz de Pinho 2016
- [3] Over-the-Air Firmware Update for Bluetooth Low Energy Devices Tuan Nguyen Anh 2019
- [4] [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)
- [5] <https://www.bluetooth.com/specifications/>
- [6] [https://ro.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://ro.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- [7] [https://ro.wikipedia.org/wiki/File\\_Transfer\\_Protocol](https://ro.wikipedia.org/wiki/File_Transfer_Protocol)
- [8] <https://ro.wikipedia.org/wiki/Internet>
- [9][https://www.openmobilealliance.org/release/LightweightM2M\\_Lightweight\\_Machine\\_to\\_Machine-v1\\_1-OMASpecworks.pdf](https://www.openmobilealliance.org/release/LightweightM2M_Lightweight_Machine_to_Machine-v1_1-OMASpecworks.pdf)
- [10] [https://en.wikipedia.org/wiki/Hash\\_function](https://en.wikipedia.org/wiki/Hash_function)
- [11] [https://en.wikipedia.org/wiki/RSA\\_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

## Anexa 1 Funcția de inițializare a aplicației

```
void SwlGateway::Init(void)
{
    uint8_t manufacture_data[MANUFACTURE_DATA_SIZE];
    uint8_t len;
    //this->swlOled.Init();
    this->sd.Init();
    this->flashDeviceManager.swlOled = &this->swlOled;
    this->vers.Init();
    this->gapAdapter.Init();
    this->gattc.Init();
    this->gattc.RegisterApp(this->appId); // @suppress("Invalid arguments")
    this->buildManufactureDataBuffer(manufacture_data, len);
    this->gapAdapter.SetDeviceName((uint8_t*)SWLGW_NAME, sizeof(SWLGW_NAME));
    this->gapAdapter.StartAdvertising(manufacture_data, len);
    this->wifiAdapter.Init();
    this->wifiAdapter.Connect();
    this->gapAdapter.StartScan(5u);
    this->swlOled.GwUpWriteStatus("SCANNING");
    this->updateFileApp.Init();
    this->isScanningRunning = true;
}
```

## Anexa 2 Funcția ciclică a aplicație de actualizare fără fir pe nodul gateway

```
void SwlGateway::TaskMain_100ms(void)
{
    if(this->isScanningRunning == false &&
        this->flashDeviceManager.listOfDevice.size() == 0)
    {
        this->flashDeviceManager.ResetInternalInfo();
        this->gapAdapter.StartScan(5u);
        this->isScanningRunning = true;
        this->isFlashingInProgress = false;
        this->updateFileApp.EnableUpdateFileApp(true);
        this->swlOled.GwUpWriteStatus("SCANNING");
    }

    if(this->isScanningRunning == false &&
        this->flashDeviceManager.listOfDevice.size() != 0
        && this->isFlashingInProgress == false
        && this->updateFileApp.EnableUpdateFileApp(false) == true)
    {
        this->triggerConnetion();
        this->isFlashingInProgress = true;
        this->swlOled.GwUpWriteStatus("UPDATING");
    }
    this->updateFileApp.UpdateFileAppMainFunction(100);
    this->swlOled.GwUpWriteProgress(this->flashDeviceManager.GetCurrentDevProgress());
}
```