Università degli Studi di Milano - Bicocca
**Scuola di Scienze**
**Dipartimento di Informatica, Sistemistica e Comunicazione**
**Corso di laurea in Informatica**

# Provisioning One Touch of Cpe Enterprise by branded Android App

**Relatore:** Prof. Arcelli Fontana Francesca

**Correlatore:** PhD Ing. Marco D'Orlando

**Relazione della prova finale di:**
Poveromo Marco
Matricola 830626

**Anno Accademico 2019-2020**

*Vorrei ringraziare profondamente tutte le persone che hanno contribuito alla mia crescita professionale durante il mio percorso di laurea.*

*In particolare, ringrazio i miei amici del corso di studi, con il quale ho condiviso la mia passione per l'informatica.*

*Ringrazio Cybaze, la quale, nonostante l'emergenza sanitaria, sia riuscita a fornirmi materiale e supporto per la tesi.*

*Infine, ringrazio la mia famiglia, senza la quale tutto questo non sarebbe stato possibile.*

# Abstract

Lo studio mostra la **fattibilità** di automazzare il processo di provisioning degli apparati CPE (Customer Premises Equipment) enterprise, per esempio router **Huawei**, **Cisco** e **Juniper**, tipicamente destinati all'uso aziendale.

L'automatizzazione del processo di provisiong è un **problema** rilevante e ben noto nelle Telco, quelle società di telecomunicazioni che installano e configurano apparati per fornire diverse tipologie di servizi, tra cui la connessione Internet e Intranet.

In secondo luogo, la tesi mostra la progettazione e la realizzazione di una **soluzione**, cioè il prototipo di un sistema per il provisioning automatico, basato su un'applicazione per Android.
Esso ha come obiettivi principali il collegamento tramite porta seriale tra un dispositivo Android ed un CPE, il riconscimento automatico di vendor e model, la successiva riconfigurazione del CPE per l'attivazione di tutti i servizi necessari nella sede del Cliente finale.

Il prototipo realizzato non è da considerarsi un sistema completo e funzionante, ma il punto di partenza di un progetto complesso, che richiede uno studio di molteplici aspetti, il testing, l'analisi di una eterogeneità di apparati relativi a vendor distinti e l'implementazione di una serie di funzionalità che, al fine di realizzare un prototipo, sono state omesse.

# Indice

# Capitolo 1

# Introduzione

La seguente tesi presenta il lavoro svolto durante lo stage presso **Cybaze S.p.A**.

Per **provisioning** si intendono i processi e i sistemi, all'interno di una Telco, coinvolti nell'attivazione e nella successiva erogazione di un servizio al Cliente finale. Ad esempio una azienda richiede ad una Telco il servizio Internet su rete fissa e in questo il provisioning è l'intero processo messo in atto dalla Telco per attivare il servizio al cliente finale: dalla verifica della copertura della linea, alla eventuale posa della stessa, dalla fornitura di un apparato CPE fino alla configurazione dei comandi sul CPE per consentire l'erogazione e il controllo da remoto.
Il termine **one touch** si riferisce ad un tipo di provisioning nel quale è sufficiente un solo "tocco" da parte di un essere umano.
La tesi prende in esame il provisioning one touch dei CPE: nella pratica sono router destinati a grosse aziende (da qui **enterprise**) i quali devono garantire requisiti di affidabilità e prestazioni elevate.
Questo processo di provisioning può essere efficacemente erogato tramite un'**applicazione Android**, la cui implementazione è l'obiettivo della tesi.
Il risultato descritto dalla tesi e sviluppato durante lo stage è un **prototipo** di un'app dedicata al provisioning sopra descritto.

Le **esigenze** che hanno generato lo sviluppo di questo prototipo sono legate alla possibilità di ottimizzare, in termini di tempi e costi, i processi di provisioning tradizionali utilizzati oggigiorno dai System Integrator (società esterne specializzate nel attivare i servizi per conto delle Telco stesse). Attualmente alcuni vendor, come ad esempio Cisco, hanno iniziato ad utilizzare app android simili per il provisioning, realizzando strumenti efficaci e applicabili soprattutto in ambito Data Center.

Nella fase iniziale del progetto è stata prevista un'analisi delle tecnologie, raccolta di requisiti e progettazione, con il successivo sviluppo di un prototipo. Quest'ultimo è ancora lontano da essere considerato un sistema completo e funzionante.

I vincoli di progetto hanno riguardato essenzialmente il sistema operativo, il linguaggio di programmazione e l'uso della porta seriale, in particolare è stato deciso di sviluppare per il sistema **Android** con il linguaggio **Java**.
Come ambiente di svulippo è stato usato **Android Studio**.

La tesi è suddivisa in **tre parti**:

- **Il problema**
  - Descrive il problema delle Telco relativo al provisioning
- **La soluzione**
  - Descrive come è stata progettata una soluzione al problema e di come questa è stata implementata.
- **Il prototipo**
  - Presenta l'applicazione sviluppata, destinata agli operatori della Telco e ai suoi attivatori in campo. Nello stesso capitolo sono riportati anche le conclusioni e i possibili sviluppi futuri.

Le tre parti della tesi includono i **capitoli**:

- **Capitolo 2: Attivazione servizi clienti enterprise per una Telco**
  - Descrive come è strutturato un sistema informativo di una Telco e quali sono le fasi che devono essere affrontate per il provisioning dei servizi.
- **Capitolo 3: Attivazione servizio tradizionale vs One Touch Provisioning**
  - Mette in relazione i processi tradizionali di provisioning con quelli automatizzati e ne descrive vantaggi e svantaggi.
- **Capitolo 4: Analisi e progettazione del sistema**
  - Descrive il lavoro svolto per l'analisi e la progettazione del sistema, include alcuni diagrammi e i disegni dell'interfaccia grafica.
- **Capitolo 5: Aspetti di basso livello del sistema**
  - Descrive la trasmissione seriale e l'interfaccia utilizzata per la comunicazione con il CPE.
- **Capitolo 6: Assunzioni e ipotesi iniziali**
  - Descrive le assunzioni e le ipotesi iniziali precedenti allo svolgimento del progetto.
- **Capitolo 7: Presentazione App Android**
  - Viene presentato il prototipo dell'applicazione.
- **Capitolo 8: Conclusione e sviluppi futuri**
  - Descrive la futura evoluzione del progetto, gli obiettivi raggiunti e lo stato attuale del lavoro.

Nel documento vengono utilizzati i termini "CPE", "router" e "apparato" in modo intercambiabile ed equivalente, poichè, al fine di sviluppare il prototipo, è stato analizzato come CPE un router Huawei AR 19-15.

# Parte I

# Il problema

# Capitolo 2

# Attivazione servizi clienti enterprise per una Telco

Il capitolo presenta la struttura tipica di un sistema informativo di una **Telco** e descrive, in termini generali, il funzionamento di un provisioning di servizi enterprise.
L'introduzione di questi concetti è di fondamentale importanza per capire il **problema** affrontato in questa tesi e **dove** viene collocato il prototipo realizzato.

## 2.1 Cos'è una Telco

Una **Telephone Company**[1], anche conosciuta come **Telco**, **telephone service provider** ma anche come **telecommunications operator**, è un tipo **communications service provider - CSP** e più precisamente un **telecommunications service provider - TSP** che fornisce servizi di telecomunicazione come telefonia e accesso alla comunicazione di dati.

Le **attività** che una Telco svolge al fine di erogare i servizi sono molteplici, includono la posa di cavi tra edifici e il collegamento di sedi aziendali sfruttando diversi mezzi di comunicazione come il rame, fibra ottica, microonde, ponte radio o trasmissione satellitare. Includono la vendita di **servizi** come collegamento alla rete e servici telefonoci tradizionali oppure più recenti come le tecnologie VoIP.
Una Telco, e in particolare il System integrator, effetuano la posa degli apparati CPE ed eventualmente firewall e switch che servono a collegarsi alla rete IT dell'azienda ed interconnettere, eventualmente, apparati di rete esistenti del cliente. Essa si occupa anche della programmazione del sistema, tramite processi più o meno automatici e della ricerca/correzione dei guasti sulla linea.
Inoltre una Telco può essere in grado di sviluppare sistemi software oltre alla vendita e l'installazione di servizi.

Nell'ampia varietà di Telco esistono anche i fornitori di servizi internet **Internet Service Provider - ISP** o trasmissioni televisive.

In conclusione il settore delle telecomunicazioni è in continua evoluzione tecnologica e in esso le Telco rappresentano un ruolo centrale.

[1] https://en.wikipedia.org/wiki/Telephone_company

## 2.2 Il sistema informativo

La **struttura interna** di una Telco è gestita da due sistemi informativi distinti: i sistemi **BSS** e i sistemi **OSS**.
Essi sono alla base delle Telco, in quanto permettono di veicolare la necessità di un determinato servizio verso apparati di telecomunicazione specifici per la sua implementazione.
A grandi linee si può affermare che l'integrazione tra i sistemi OSS e i sistemi BSS in maniera economica e semplice rimane l'**obbiettivo fondamentale** delle grandi aziende di telecomunicazione.

Un **esempio** utile alla comprensione di questi sistemi è quello legato al credito di una SIM telefonica, nel quale la disponibilità del servizio di fonia (–>OSS) dipende dalla disponibilità di credito sufficiente sull'account dell'utente per effettuare le chiamate, piuttosto che l'esistenza di abbonamenti flat, ecc.. (–>BSS).

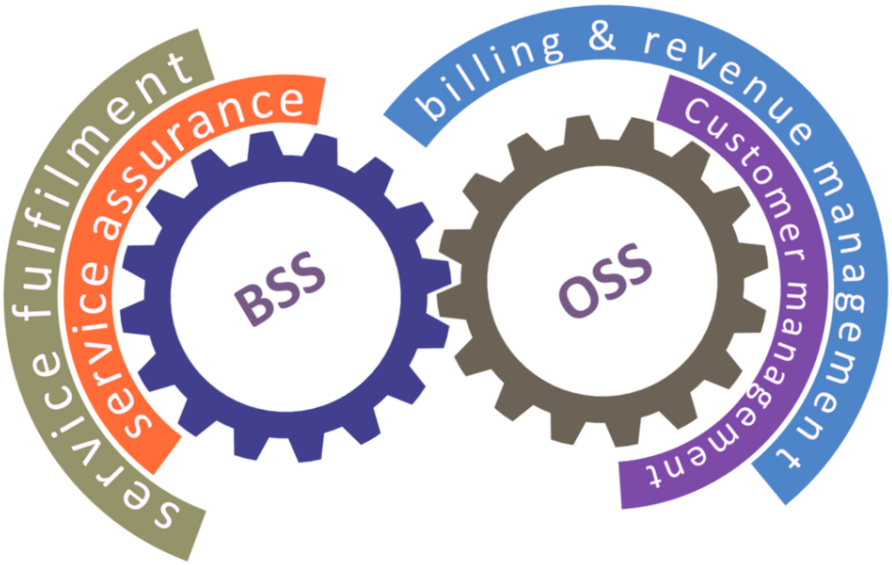


Figura 2.1: BSS vs OSS

### 2.2.1 BSS

I sistemi informativi **BSS - Business Support System** collezionano e gestiscono i dati anagrafici del cliente e lo associano ad un insieme di servizi che la Telco gli fornisce.
Un BSS è quello più vicino al consumatore, in questo senso, viene utilizzato anche per il pagamento dei servizi richiesti.

### 2.2.2 OSS

Il sistemi informativi **OSS - Operational Support System** sono quelli più vicini all'infrastruttura di rete, essi si occupano di far si che i servizi pagati dal cliente vengano effettivamente erogati.

## 2.3 Provisioning di servizi enterprise

In questa sezione viene illustrata la procedura tipica che esegue una Telco per l'**erogazione di un servizio enterprise** ad un'azienda. Per servizi enterprise si indicano tutti quei servizi di telecomunicazione forniti dalla Telco alle aziende, come ad esempio servizi di **telefonia mobile**, **VoIP**, **connessione a internet veloce**, **posta elettronica**, **VPN Layer2** fra sedi remote della stezza azienda, VPN Layer2 fra sedi, **connettività infra sedi**, **natting**, VPN per collegamenti dei dipendenti in ufficio da remoto, **servizi professionali di sicurezza e monitoraggio** della rete aziendale e protezione da malware, phishing, botnet ecc..
Per **provisioning di servizi**, nel contesto delle telecomunicazioni, si intende quel processo di preparazione ed equipaggiamento (hardware e software) di una rete per consentire a quest'ultima di fornire servizi ad un cliente. Questo termine è utilizzato tipicamente per indicare la fase iniziale di installazione e messa a punto di un servizio di rete, include quindi anche tutte quelle operazioni di predisposizione degli apparati fisici della rete stessa.

### 2.3.1 Definizione dell'offerta commerciale

La prima fase che viene identificata nel processo di provisioning è quella in cui si definisce l'offerta commerciale, in questo senso, un **project manager** della Telco comunica con l'azienda per chiarire **quali servizi** devono essere attivati ed a **quale prezzo**.

L'offerta commerciale varia in base a molteplici aspetti, per esempio, la posizione geografica di una sede aziendale potrebbe aumentare il costo di un servizio se essa si trova in una zona remota o dove si rende necessario ad esempio dover attestare un Ponte Radio per offrire i servizi di connettività. Questo comporta l'interazione tra i servizi BSS e OSS per ricercare l'offerta migliore.

### 2.3.2 Attivazione della linea

L'attivazione della linea prevede il corretto funzionamento degli apparati dedicati alla **rete di trasporto**[2] e tipicamente consiste nel trovare una soluzione per portare la connettività alle sedi aziendali, scegliendo di volta in volta la tecnologia di accesso più opportuna: ADSL, SHDSL, FrameRealy, VDSL, Fibra ecc.
Questa fase si concretizza nell'arrivo di un tecnico nelle varie sedi aziendali che porterà la connettività, per esempio con fibra ottica, vsdl o ponte radio. Essi fanno parte dell'**equiment**, utile per effettuare la connessione.

L'attivazione si conclude con i **test di linea**, i quali verificano il suo corretto funzionamento: vengono solitamente effettuati dei monitoraggi continui e degli speedtest per stressare e caricare la linea al fine di misurarne la stabilità nel troughput, jitter, ritardo e packet loss.

[2]https://it.wikipedia.org/wiki/Rete_di_trasporto

### 2.3.3 Attivazione dei servizi

Il project manager della Telco, che è coinvolto nel processo di attivazione, ha le conoscenze commerciali per sapere quello che l'azienda ha chiesto e si preoccupa di fare da interfaccia a tutti i tecnici per attivare i servizi. Per questo scopo, il **System Integrator**, porta nell'azienda un **CPE**, un apparato in grado di connettere l'azienda in rete, tipicamente esso è un dispositivo affidabile, in grado di supportare il traffico generato da molte postazioni e inserito all'interno di un rack[3].

Il System integrator, in generale, si reca nella sede aziendale **più volte** per configurare il CPE e, nel corso delle varie uscite deve, parlando con i responsabiliti IT dell'azienda e con la Telco, configurare i servizi interagendo **manualmente** con l'apparato.
Tipicamente si connette via cavo console e abilita la raggiunbilità e il controllo da remoto da parte del **Customer Care** della Telco.

[3]https://it.wikipedia.org/wiki/Rack_(informatica)

# Capitolo 3

# Attivazione servizio tradizionale vs One Touch Provisioning

Il provisioning dei servizi enterprise, tipicamente, richiede una lunga serie di operazioni per la corretta messa a punto del sistema. Una attivazione tradizionale di servizi, infatti, è oggigiorno molto utilizzata nelle Telco, e presenta una serie di problematiche che incidono sui **costi**, sui **tempi** e sull'**efficacia** del provisioning, giusto per citarne alcune.

In questo capitolo verranno illustrate varie tipologie di provisioning dei servizi, percorrendo alcune delle tecniche più usate nelle Telco, fino ad arrivare a soluzioni sempre più innovative e automatizzate.

## 3.1 I problemi di una attivazione tradizionale

L'attività di attivazione dei servizi, mostrata nel capitolo precedente, viene solitamente svolta dalle Telco in maniera tradizionale. Questa prevede una grande quantità di **lavoro manuale**, svolto dal **system integrator**, a tutti gli effetti un tecnico della Telco.
Esso si occupa di presentarsi di persona nelle sedi aziendali per configurare gli apparati CPE, interfacciandosi da una parte con i tecnici interni alla Telco, tramite chiamate telefoniche, e dall'altra con i responsabili IT dell'azienda stessa.

Quello che accade tipicamente è che il system integrator svolge un lavoro molto **complesso**, utilizzando spesso **strumenti ingombranti** per la configurazione, con molti cavi da collegare e scollegare nelle porte del CPE.
Non è di aiuto il fatto che questo tipo di lavoro deve essere svolto **in velocità**, poichè i System integrator hanno molti clienti da assistere durante una giornata, con il rischio di commettere **errori**, non solo di battitura, ma anche logici, i quali potrebbero compromettere l'integrità del lavoro svolto. L'effetto di queste operazioni sbagliate potrebbe, in primo luogo **compromettere l'affidabilità** dei servizi erogati ed in secondo luogo inserire **vulnerabilità** nella configurazione dell'apparato esponendo l'azienda a potenziali **attacchi informatici**.

Inizialmente il lavoro è svolto telefonicamente, in questo senso sono i tecnici della Telco che suggeriscono i comandi da eseguire nella console del CPE, con l'obiettivo di effettuare una **configurazione base**.

Essa garantisce l'accesso da remoto, con cui la Telco riconfigura successivamente l'apparato, avvalendosi di strumenti per la riconfigurazione oppure eseguendo comandi tramite **SSH - Secure SHell**.

In conclusione, l'attivazione tradizionale presenta moltissimi svantaggi, le attività sono lunghissime, complesse e non vanno quasi mai a buon fine per colpa della grande quantità di lavoro manuale che comportano, inoltre i system integrator sono spesso costretti a presentarsi molte volte in azienda per rimediare agli errori commessi.

## 3.2 I vantaggi di un provisioning automatizzato

Il processo di configurazione dell'apparato, effettuato dal system integrator, è un lavoro manuale che può essere in gran parte automatizzato, permettendo di **risolvere** molte delle problematiche di un deployment tradizionale.

Il punto centrale di un provisioning automatico è minimizzare il lavoro svolto manualmente dal system integrator. Supponiamo che tramite un software si riesca a scaricare nel CPE ed installare configurazioni corrette: questo permetterebbe di avere un apparato **privo di errori in partenza** e soprattutto con un **risparmio** enorme in termini di **tempo e costi**.
L'obiettivo di queste procedure vuole anche essere quello di massimizzare il lavoro svolto da software automatici, permettendo in primo luogo al system integrator di installare la **configurazione di base** (quella che permette di accedere al CPE da remoto) corretta, ed in secondo luogo, quando possibile, attivare automaticamente anche i servizi enterprise durante la prima installazione.
Il processo che si vuole automatizzare si colloca, quindi, nella fase di **attivazione dei servizi**, e si traduce nella corretta **configurazione del CPE**, la quale richiede:

- File di configurazione del CPE
- Firmware aggiornato per il CPE
- Patch personalizzate per il firmware

A livello pratico, le operazioni manuali che svolge il system integrator sono installare questi 3 file nel CPE, e verificare con l'aiuto della Telco il funzionamento dei servizi.
Assumiamo che esista una funzione $f$ che riesca a produrre i 3 file di configurazione corretti, dati in ingresso: il cliente, i servizi da attivare, la linea che il cliente ha deciso di attivare per una sede, e il CPE associato alla sede.

$$f(cliente, servizi[], linea, cpe) = (firmware, patch, configurazione)$$

Il lavoro manuale di un system integrator sarebbe ridotto al minimo, e quello che dovrebbe fare concretamente sarebbe solamente prelevare i file di configurazione ottenuti come output di $f$, ed installarli nel CPE con uno strumento completamente automatico.

In conclusione, i vantaggi di un provisioning automatizzato sono molteplici, dal risparmio in termini di costi e tempi all'efficacia ed efficienza dell'installazione.

## 3.3 Tipologie di provisioning

La terminologia utilizzata dalle Telco per riferirsi a strumenti automatici di configurazione di CPE è molto variegata, in questo senso si possono distinguere però le varie tipologie di provisioning suddividendole per il loro **grado di automatizzazione**, misurando quanto l'essere umano deve effettivamente intervenire manualmente nel processo.
Alcuni termini ampiamente utilizzati in questo contesto sono **Zero touch provisioning - ZTP**, **One touch provisioning** oppure ancora **Minimal touch provisioning**, i quali vengono utilizzati per indicare la quantità di "touch", cioè di tocchi/operazioni manuali effettuate per completare il provisioning.

### 3.3.1 Zero touch provisioning

Solitamente si usa il termine **Zero Touch Provisioning - ZTP** per riferirsi a apparati di rete che vengono installati e configurati senza l'intervento umano, cioè senza che qualcuno lo configuri manualmente. In questo senso, ZTP si riferisce ad un provisioning che ha come obiettivo inviare un CPE ad una azienda e lasciare che chi lo riceva, anche se non ha competenze tecniche, riesca a collegarlo, accenderlo e lasciare che l'apparato completi in totale autonomia il processo di provisioning.
Un sistema ZTP prevede il collegamento in rete, sfruttando un Management System per scaricare i file di configurazione ed auto-configurarsi, quindi, sfrutta a tutti gli effetti una configurazione di default per installare una versione aggiornata e corretta della configurazione.

In figura 3.1 viene illustrato uno schema di funzionamento ZTP nel quale, l'apparato si connette ad un sistema di management in cloud per scaricare la configurazione.

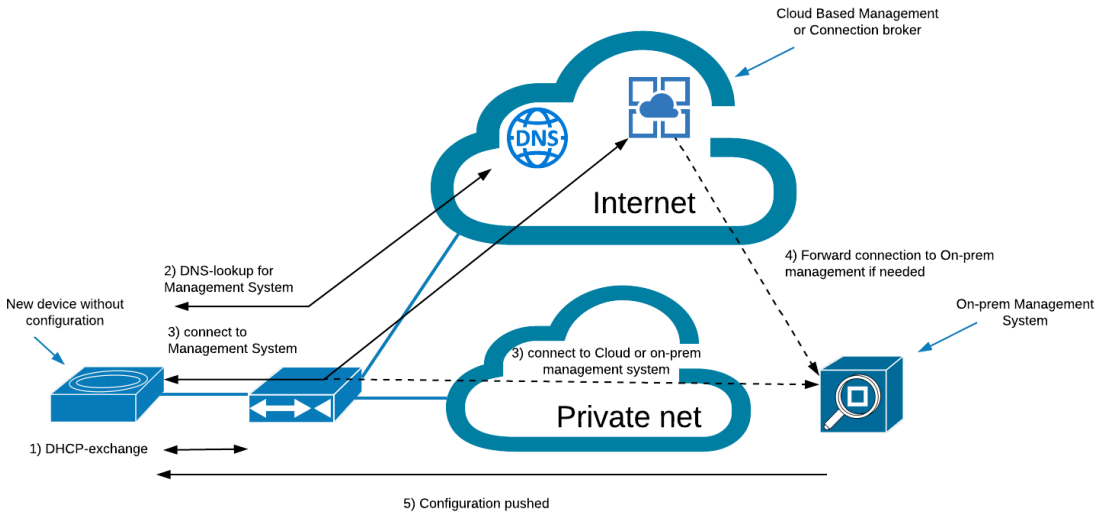


Figura 3.1: Schema Zero touch provisioning

### 3.3.2 One Touch e Minimal touch provisioning

Come già accennato in precedenza, i sistemi che operano il provisioning dei CPE, si possono distinguere in base al loro grado di automatizzazione rispetto ad un intervento manuale da parte di un system integrator.
Alcuni di questi provisioning sono soprannominati **One Touch** o **Minimal Touch** perché hanno un grado di automatizzazione leggermente più basso rispetto a ZTP.
Queste tipologie di provisioning prevedono un intervento umano, seppur minimo, per la configurazione.

Un processo di provisioning one touch o minimal touch, nel contesto di una attivazione dei servizi enterprise, consiste tipicamente in:

- Posa del CPE
- Avvio manuale di un sistema progettato per la auto-configurazione del CPE

Un sistema di provisioning **one touch**, cioè ad "un tocco", può essere realizzato tramite un'applicazione per Android installata su uno smartphone. Questa soluzione dà la possibilità di collegarsi, da una parte, tramite cavo con il CPE, dall'altra, offre una connessione ad Internet indipendente dal CPE, la quale potrebbe essere utilizzata per recuperare i file di configurazione da un **ACS - Auto Configuration Server**.
Lo smarthphone con la sua APP agisce da **attuatore** e, insieme all'ACS, rappresenta un'architettura per il setup automatico dei servizi enterprise su CPE eterogenei.

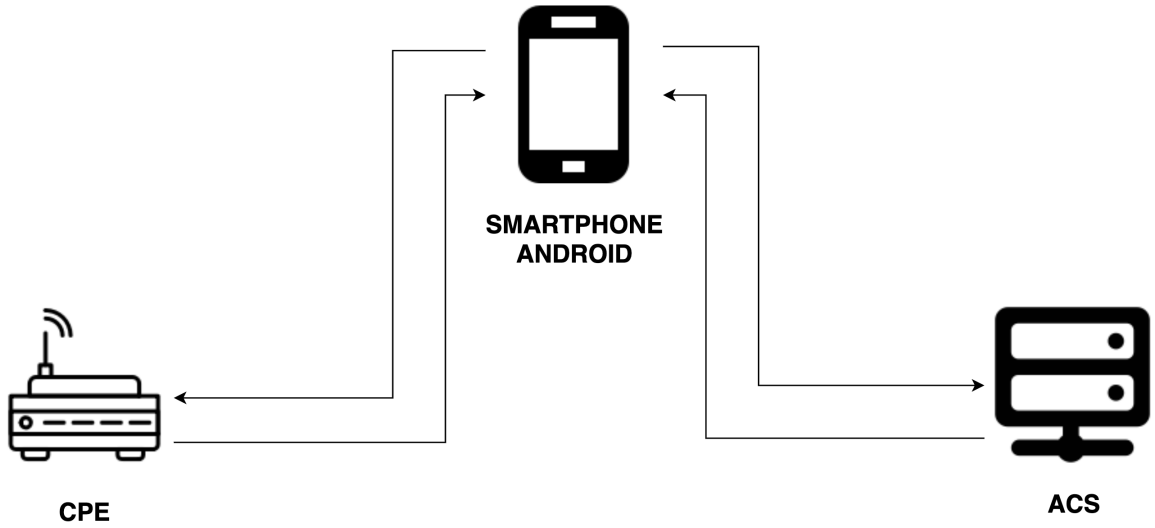


Figura 3.2: One touch - Minimal touch

Un'applicazione android per il provisioning one touch, potrebbe essere considerata, inoltre, un servizio completo per apparati enterprise dedicato al **delivery automatico**, con la possibilità di poter gestire un'eterogeneità di vendor distinti.

# Parte II

# La soluzione

# Capitolo 4

# Analisi e progettazione del sistema

Il sistema che si vuole realizzare è una applicazione Android che permette di automatizzare il lavoro di un **System integrator** durante la configurazione di un **CPE**, e che risolva quindi i problemi del traditional deployment descritti nei capitoli precedenti.
Il sistema rientra nella tipologia di provisioning denominata **One Touch** o **Minimal Touch**, e viene denominato **SoulDuct**.
La progettazione viene corredata di diagrammi e casi d'uso per facilitarne la comprensione e lo sviluppo.

## 4.1 Casi d'uso

In questa sezione verranno illustrati i casi d'uso tramite **diagramma** e **formato informale**.

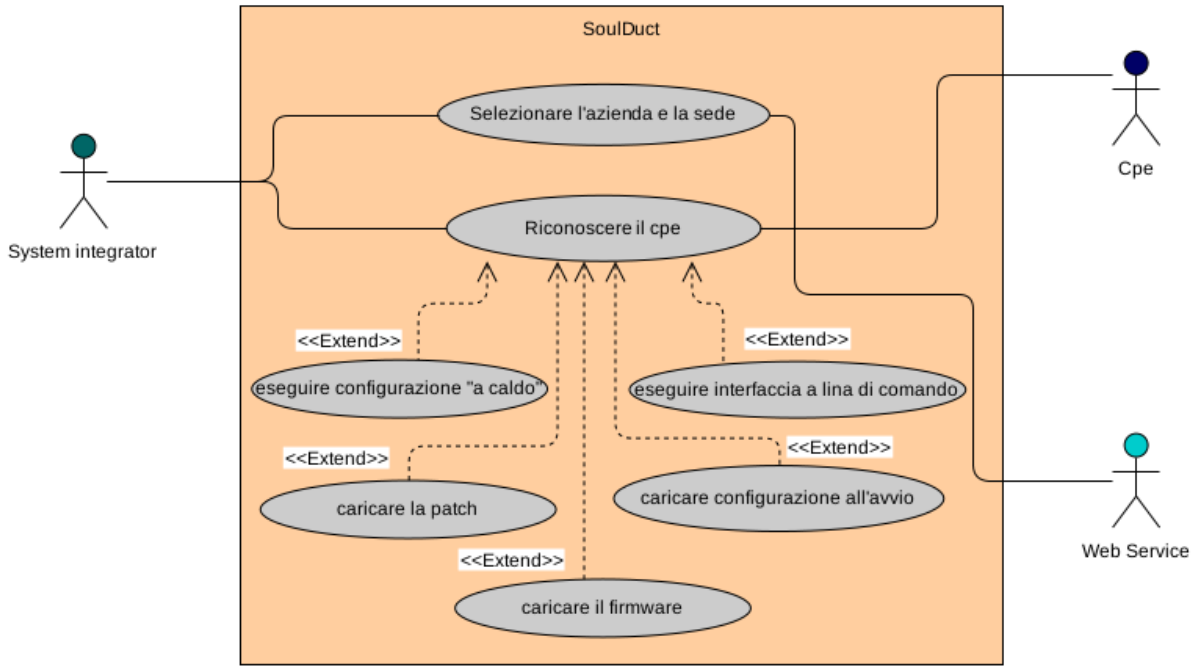


Figura 4.1: Diagramma dei casi d'uso di SoulDuct

### 4.1.1 Selezionare l'azienda e la sede

**Scenario principale di successo**: il System integrator inserisce il numero identificativo dell'azienda, esso viene trovato nel database. Successivamente seleziona la sede in cui vuole effettuare la configurazione del CPE.

**Scenario alternativo**: il System integrator inserisce il numero identificativo dell'azienda ma esso non viene trovato nel database. Il System integrator viene informato dell'errore.

### 4.1.2 Riconoscere il CPE

**Scenario principale di successo**: il System integrator collega il terminale Android con il CPE tramite porta seriale, il sistema verifica correttamente che il CPE collegato corrisponda effettivamente a quello associato alla sede selezionata.

**Scenario alternativo**: il System integrator collega il terminale Android con il CPE tramite porta seriale, il sistema si accorge che il CPE collegato non corrisponde a quello associato alla sede selezionata. Il System integrator viene informato dell'errore.

### 4.1.3 Eseguire configurazione "a caldo"

**Scenario principale di successo**: il System integrator esegue la configurazione "a caldo" e attende che tutti i comandi vengano lanciati sul CPE.

**Scenario alternativo**: il database non rende disponibile un file di configurazione "a caldo". Il System integrator viene informato dell'errore.

**Scenario alternativo**: il sistema perde la connessione con il CPE e torna allo stato iniziale.

### 4.1.4 Eseguire interfaccia a linea di comando

**Scenario principale di successo**: il System integrator esegue l'interfaccia a linea di comando nel quale esegue i comandi in modo del tutto manuale, interfacciandosi con la shell del CPE.

**Scenario alternativo**: il sistema perde la connessione con il CPE e torna allo stato iniziale.

### 4.1.5 Caricare la patch

**Scenario principale di successo**: il System integrator esegue il caricamento della patch, essa viene trasferita nella memoria del CPE. Il System integrator conferma l'installazione della patch.

**Scenario alternativo**: il System integrator esegue il caricamento della patch ma il trasferimento non va a buon fine. Il System integrator viene informato dell'errore.

**Scenario alternativo**: il sistema perde la connessione con il CPE e torna allo stato iniziale.

### 4.1.6 Caricare configurazione all'avvio

**Scenario principale di successo**: il System integrator esegue il caricamento della configurazione all'avvio, essa viene trasferita nella memoria del CPE. Il System integrator conferma l'installazione della configurazione all'avvio del CPE. Il CPE viene riavviato con la nuova configurazione.

**Scenario alternativo**: il System integrator esegue il caricamento della configurazione ma il trasferimento non va a buon fine. Il System integrator viene informato dell'errore.

**Scenario alternativo**: il sistema perde la connessione con il CPE e torna allo stato iniziale.

### 4.1.7 Caricare il firmware

**Scenario principale di successo**: il System integrator esegue il caricamento del firmware, essa viene trasferita nella memoria del CPE. Il System integrator conferma l'installazione del firware all'avvio del CPE. Il CPE viene riavviato con il nuovo firmware.

**Scenario alternativo**: il System integrator esegue il caricamento delfirmware ma il trasferimento non va a buon fine. Il System integrator viene informato dell'errore.

**Scenario alternativo**: il sistema perde la connessione con il CPE e torna allo stato iniziale.

## 4.2 Nome del sistema

Il nome scelto per l'applicazione è **SoulDuct**. Poichè l'obiettivo primario di questo sistema è configurare un CPE allo stato di fabbrica tramite un cavo, il nome scelto vuole evocare sinteticaemente questa attività.
Infatti **Soul** è stato selezionato perchè la configurazione fà pensare di "dare un anima" all'apparato, mentre **Duct**, in inglese "condotto" si riferisce al cavo che viene utilizzato. Il nome è anche ispirato dal gesto del "soffio della vita" mentre il logo dell'applicazione è un elmetto militare perchè la pronuncia di SoulDuct, in italiano, è in assonanza con **soldato**.

Figura 4.2: Logo di SoulDuct

## 4.3 Interfaccia grafica

L'interfaccia grafica è stata progettata inizialmente mediante **sketch e wireframe**, mentre il prototipo (non "usa e getta") è stato costruito con il tool **Android Studio**.
In seguito vengono riportate le immagini di come si presentava il wireframe in origine, mentre nei capitoli successivi viene mostrato il prototipo.


Figura 4.3: Wireframe


Figura 4.4: Wireframe

## 4.4 Diagramma di attività

Il seguente diagramma delle attività descrive il processo di provisioning one touch.
Il diagramma, suddivide il processo tramite le swimlane, evidenziando in particolare il flusso delle attività del **System integrator** e di **SoulDuct**.

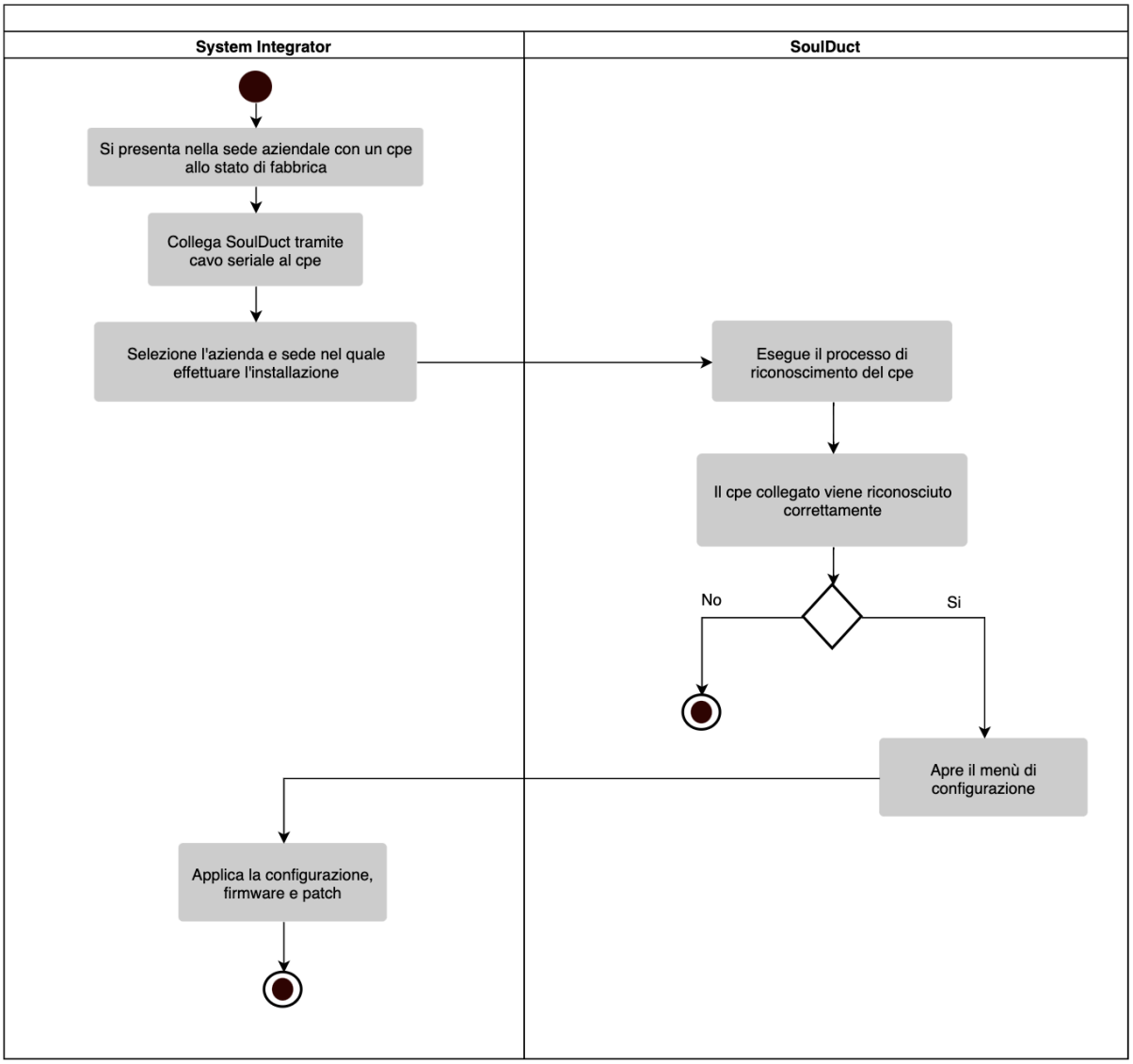


Figura 4.5: Diagramma delle attività

La configurazione del CPE viene applicata dal System integrator solamente dopo che il sistema SoulDuct ha riconosciuto correttamente l'apparato.
In questa fase, viene prelevato il nome del **Vendor**, il **Model** e altre informazioni utili al riconscimento dell'apparato stesso.

# Capitolo 5

# Aspetti di basso livello del sistema

Un aspetto fondamentale è quello che considera i **protocolli** e i **componenti fisici** per il trasferimento dei dati, infatti un dispositvo CPE è solitamente in grado di offrire molteplici soluzioni che consentono ad un terminale di connettersi ad esso.

Il sistema progettato in questa tesi utilizza principalmente l'interfaccia **RS-232** per la **trasmissione seriale**, essa viene corredata con alcuni adattatori per consentire l'utilizzo delle porte con funzionalità **USB-OTG** di Android e della porta **RJ-45** dei CPE.

## 5.1 La trasmissione seriale

La trasmissione seriale (Figura 5.1) è un tipo di comunicazione nel quale i bits delle words da trasmettere sono inviate in maniera sequenziale, **un bit alla volta**.
La trasmissione viene effettuata tramite un singolo bus nel quale scorrono i bits.

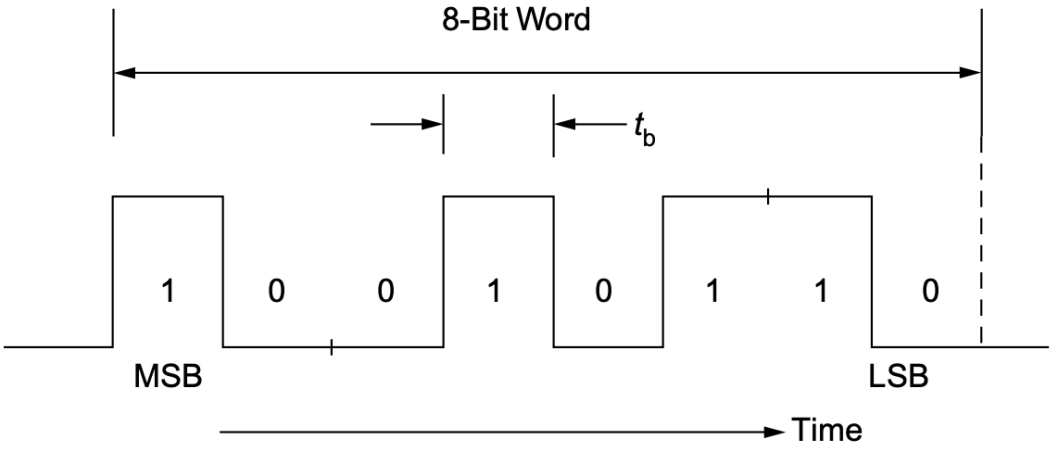


Figura 5.1: La trasmissione seriale

### 5.1.1 Il modello OSI per la trasmissione seriale

Qualsiasi tipo di comunicazione seriale utilizza il modello **OSI - Open Systems Interconnection** (Figura 5.2), in particolare, il **phisical layer** è responsabile della tra-

smissione e della recezione di dati non strutturati tra il dispositivo e il mezzo fisico di trasmissione. Questo layer, inoltre, è responsabile della conversione dei bits digitali in segnali elettrici, radio o ottici. [1] E' in questo layer che vengono definite le **specifiche delle interfacce seriali**, sebbene la maggior parte dei dispositivi si arrampichi leggermente sugli strati sovrastranti aggiungendo **bit di stop e parità**, che a scopi pratici, possono essere considerati parte della specifica.

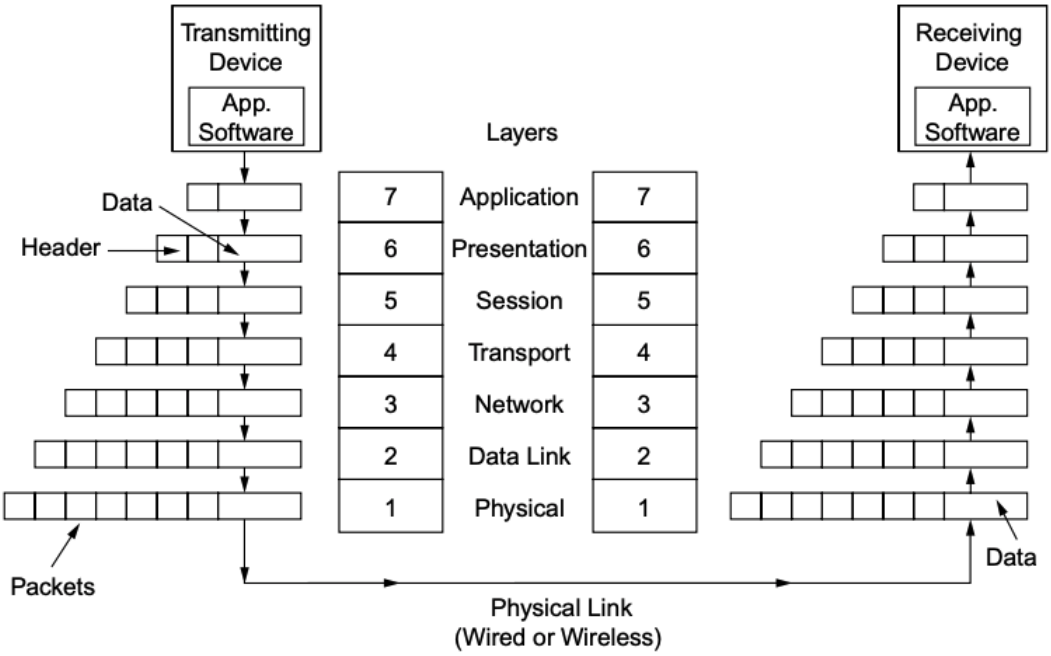


Figura 5.2: Il modello OSI

### 5.1.2 La topologia

La **topologia** definisce come i nodi della rete sono fisicamente interconnessi, la connessione più semplice è quella peer-to-peer, nel quale ci sono solamente due nodi che comunicano, come ad esempio un CPE ed un terminale android.

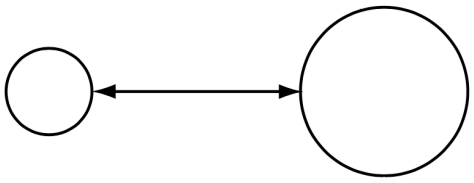

Figura 5.3: Topologia peer-to-peer

[1] <https://matrixti.com/matrix-on-manufacturing/rs-232-is-simple-robust-serial-communications-method>

### 5.1.3 Trasmissione sincrona e asincrona

La trasmissione seriale può trasmettere i dati in modo **sincrono o asincrono**, a seconda dell'intervallo di tempo che intercorre tra l'invio di una word e della successiva.
Solitamente nel caso **asincrono** 5.4 viene trasmessa una word alla volta e ciascuna di esse è accompagnata da bit di inizio e di fine che definiscono l'inizio e la fine di una word.
La trasmissione asincrona è molto affidabile ma è inefficiente rispetto a quella sincrona per via dei bit di inizio e di fine, infatti questi bit rallentano la trasmissione.
Tuttavia, per alcune applicazioni, in cui l'affidabilità è essenziale, la mancanza di velocità è trascurabile.

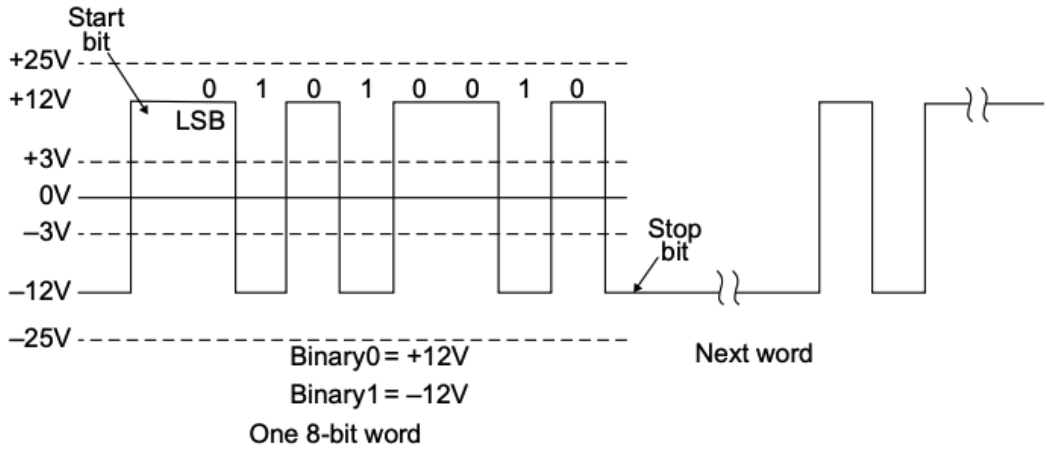


Figura 5.4: Trasmissione asincrona di dati

### 5.1.4 Il circuito integrato UART

Le interfacce e protocolli seriali che utilizzano il metodo asincrono sono ampiamente diffuse, così tanto che molti dispositivi, compresi quelli Android, incorporano un circuito integrato standard che implementa automaticamente questo processo di trasmissione dati, esso è noto come **UART - universal asynchronous receiver transmitter** (Figura 5.5).

**UART** è un'interfaccia generica per lo scambio di dati "grezzi" con un dispositivo periferico. [2] È **universale** perché sia la velocità di trasferimento dei dati che il formato dei byte di dati sono configurabili. È **asincrono** in quanto non sono presenti segnali di clock per sincronizzare il trasferimento di dati tra i due dispositivi.
L'hardware del dispositivo raccoglie tutti i dati in arrivo in un **buffer FIFO** (first-in first-out) fino a quando non viene letto dall'app.
Il trasferimento di dati UART è **full duplex**, cioè, i dati possono essere inviati e ricevuti contemporaneamente, ma la **mancanza di un clock condiviso** significa che entrambi i dispositivi devono concordare una velocità di trasferimento dati comune, per esempio 9600 bit/s.

[2] <https://developer.android.com/things/sdk/pio/uart>

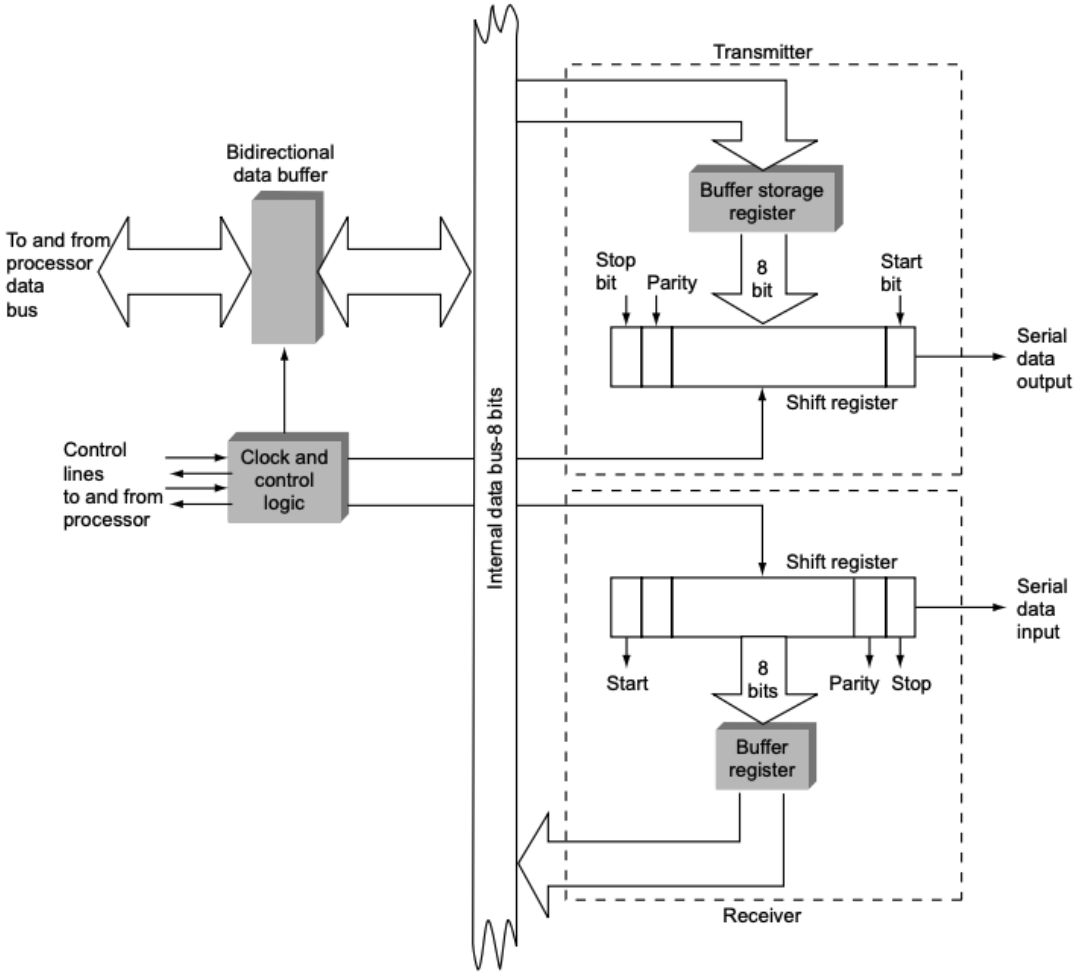


Figura 5.5: UART - universal asynchronous receiver transmitter

Questo circuito integrato è utile per **interfacciare** il flusso seriale di dati, in ingresso e in uscita, con la cpu del dispositivo che lo ospita, il quale, invia e riceve i dati al UART tramite un bus parallelo, e memorizza temporaneamente la trasmissione in alcuni buffer dedicati.

La modalità di **invio dei dati in output** prevede che i byte vengano trasferiti al buffer storage register tramite i bus interni del UART, e successivamente un byte alla volta verso lo shift register. Quest'ultimo è responsabile dell'aggiunta dei bit di **inizio, fine e parità**. La parità è generalmente facoltativa e può essere impostata a "dispari" o "pari". Successivamente un segnale di **clock** trasmette la word in serie.
Un procedimento simile, ma invertito, descrive come i dati in input siano **inoltrati verso la cpu**, la principale differenza è la presenza del buffer register per la memorizzazione delle word in arrivo.
Sebbene questi circuiti integrati siano disponibili singolarmente, solitamente, sono inclusi dentro altri controller oppure in altri circuiti integrati.

### 5.1.5 Sincronizzazione del segnale di clock

Un aspetto importante della trasmissione seriale è il **segnale di clock**, infatti i circuiti che inviano e ricevono i dati richiedono che esso sia stabilito per interpretare correttamente il flusso di dati.
Esistono vari metodi per fornire il segnale di clock: un primo metodo prevede che i due circuiti che trasmettono e ricevono abbiano clock uguali ma non sincronizzati, questa piccola differenza può essere tollerata ed è il caso delle trasmissioni asincrone.
Un secondo approccio prevede di inviare il segnale di clock tramite un bus separato, questo permette di avere il segnale di clock perfettamente sincronizzato.
Un terzo metodo consiste nel dedurre il segnale di clock dalla trasmissione in arrivo, questo metodo è conosciuto come **CDR - clock and data recovery** e per funzionare correttamente richiede che ci sia una quantità sufficiente di bit in arrivo.

### 5.1.6 Protocolli

Un **protocollo** è un insieme di regole e procedure che definiscono come i nodi di una rete possono comunicare.
Esso definisce il modo con cui i dati devono essere impacchettati prima di essere spediti, in particolare, il formato del pacchetto, e la sequenza con cui i pacchetti vengono comunicati.
Un pacchetto tipico di una trasmissione seriale è indicato nella figura 5.6, esso inizia con **Sync** cioè uno o più byte che definiscono l'inizio del frame e permettono al circuito del ricevente di capire che è in arrivo un pacchetto.

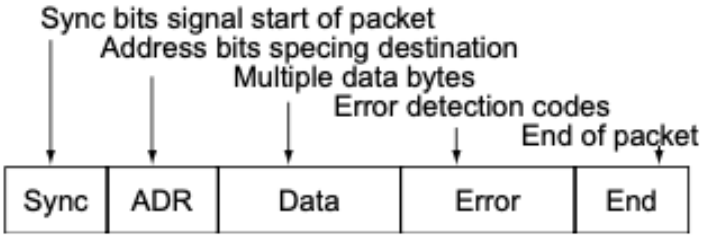


Figura 5.6: Tipico pacchetto di una comunicazione seriale

Il campo successivo è **ADR**, cioè un insieme di bit che identificano il nodo ricevente. I dati effettivi da trasmettere vengono appesi tramite il campo **Data**, la quantità di word presenti in questo campo è variabile e può essere anche un numero molto grande di byte. **Error** è un insieme di bit che definiscono il correction code, e vengono utilizzati dal ricevente per capire se è avvenuto un errore nella trasmissione e, nel caso, correggerlo.
L'insieme di bit di **End** definisce la fine del pacchetto e della trasmissione.
Solitamente, nella trasmissione sincrona, il campo Data è costituito da un elevato numero di byte, mentre in quella asincrona viene inserito un solo byte.

### 5.1.7 Error Detection and correction

Alcune interfacce seriali permettono di rilevare gli errori presenti nel pacchetto in arrivo dovuti a problemi nella trasmissione. Queste tecniche migliorano l'affidabilità e riducono il **BER - bit error rate**.
La forma più semplice di rilevamento degli errori è il **Parity bit** o bit di parità, questa tecnica consiste nel contare il numero di 1 nella word da trasmettere, e aggiungere, nel campo Data del pacchetto, l'informazione sulla parità/diparità di questo numero. Questa informazione può essere rappresentata da 1 solo bit e solitamente viene usato lo 0 per indicare il pari e l'1 per indicare il dispari.
Il ricevitore conterà il numero di 1 nel pacchetto per capire se è pari o dispari, e confronterà questa informazione con quella ricevuta nel campo Data del pacchetto, per scoprire se è avvenuto un errore nella trasmissione.
Questa tecnica, tuttavia, non consente di identificare il bit di errore e richiede la ritrasmissione del pacchetto.

Oltre a questo metodo ne esistono molti altri, per esempio il **Block Check Code**, **CRC** e **FEC**, essi sono molto più elaborati e sfuttano algoritmi complessi. Alcuni di loro sono in grado di identificare i bit erronei, e richiedono di aggiungere molti bit di informazione nel pacchetto da trasmettere.

### 5.1.8 Duplexing

Le trasmissioni seriali possono essere suddivise in due categorie, full-duplex e half-duplex.
**half-duplex** alterna le operazioni di trasmissione a quelle di trasmissione, in modo che le parti non possono trasmettere nello stesso momento.
Le trasmissioni **full-duplex** permettono di inviare e ricevere simultaneamente, tramite due canali o linee.

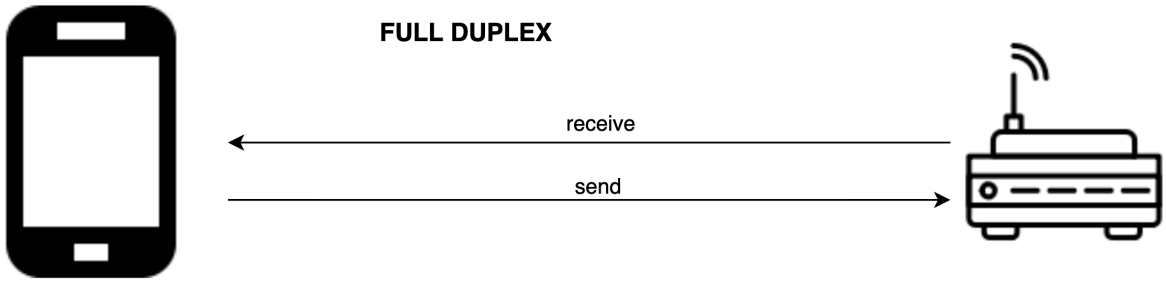


Figura 5.7: full duplex

## 5.2 L'interfaccia RS232

[3] **RS-232** (Figura 5.8) naque negli anni '60 grazie alla **Electronic Industries Association (EIA)** e veniva utilizzata per la comunicazione tra computer e modem.
Successivamente fu utilizzata in larga scala anche per connettere i personal computer a periferiche esterne, ad esempio stampanti.
RS-232 è stata ampiamente diffusa, tanto da essere **conosciuta generalmente come "porta seriale"** o serial communication interface (SCI) ed è ancora oggi presente in molti macchinari industriali, macchine utensili, PLC, microcontrollori integrati e altri tipi di apparecchiature.

Le **velocità di trasmissione** comunemente utilizzate con questa porta sono 75, 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, e 115200 **bit/s**, e devono essere uguali per il trasmettitore e per il ricevitore. Velocità basse e cavi di pochi metri sono generalmente associati ad una trasmissione più affidabile, mentre l'aumento di queste due variabili potrebbe costituire un problema per la trasmissione.

Questa porta viene utilizzata per scambiare dati tra due dispositivi, il **DTE (Data Terminal Equipment)**, per esempio un computer, e il **DCE (Data Communication Equipment)**, per esempio un modem.

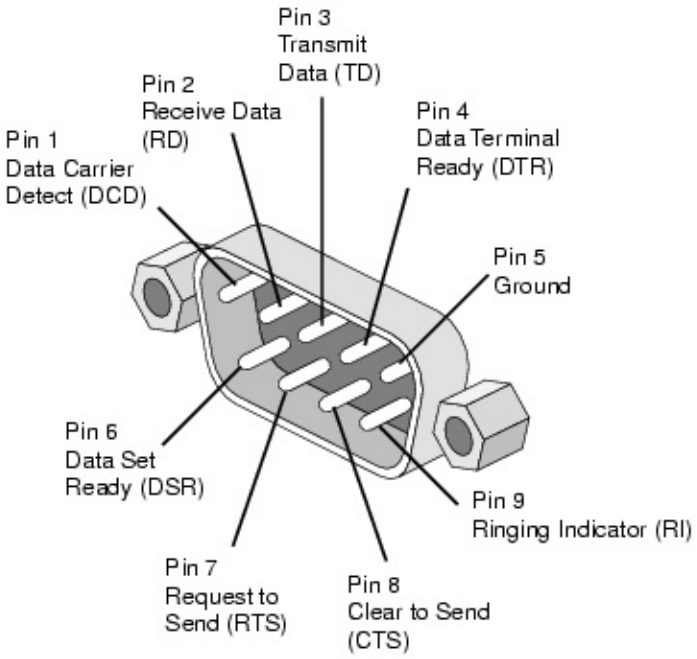


Figura 5.8: Interfaccia RS-232 a 9 pin, maschio

[3] https://www.db9-pinout.com/

I pin relativi alla porta RS-232 hanno le seguenti funzionalità:

- **1 - DCD**, DCE accetta un corriere da un DTE situato in una posizione remota.
- **2 - RD**, invia dati da DCE a DTE.
- **3 - TD**, per trasmettere dati da DTE a DCE.
- **4 - DTR**, DTE è pronto ad accettare la richiesta.
- **5 - Signal Ground**, livello di tensione zero di riferimento per tutti i segnali di controllo.
- **6 - DSR**, DCE è pronta a inviare e ricevere le informazioni.
- **7 - RTS**, chiamata DTE per DCE per inviare i dati.
- **8 - CTS**, DCE è pronto per accettare i dati provenienti da DTE.
- **9 - RI**, rileva la suoneria in arrivo sulla linea telefonica.

Il protocollo relativo a questa porta utilizza la **comunicazione seriale asincrona** tramite **UART**, con i bit di start di stop e di parità, mentre i dati sono solitamente di 1 byte.

La comunicazione tra due dispositivi può essere effettuata utilizzando solamente **3 pin**: RD, TD e il Ground, come in figura 5.9. [4]

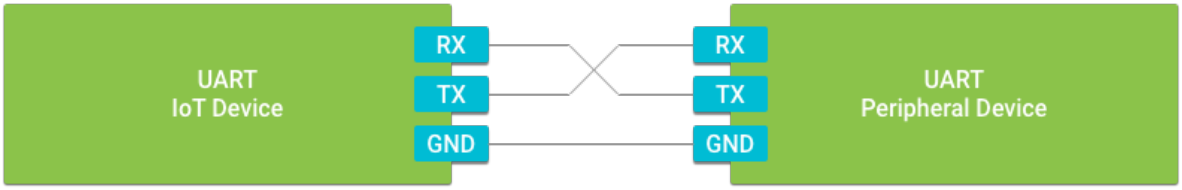


Figura 5.9: Comunicazione con 3 pin

Inoltre, abilitando il **flow control**, si può aumentare l'affidabilità della trasmissione, e solitamente questo si associa a velocità di trasferimento maggiori. Questo si ottiene utilizzando in aggiunta **altri 2 pin**: RTS e CTS come illustrato in figura 5.10.

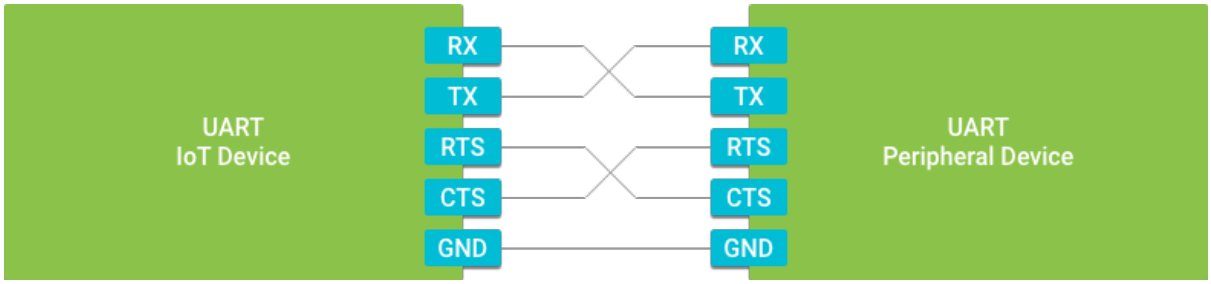


Figura 5.10: Comunicazione con 5 pin

[4] <https://developer.android.com/things/sdk/pio/uart>

# Capitolo 6

# Interfacciamento Android-Cpe

Un aspetto fondamentale del sistema OneTouch riguarda il **collegamento** tra dispositivo Android e il CPE. Si tratta di un collegamento **via cavo**, che utilizza la tecnologia **seriale** per la comunicazione.
In termini pratici essa viene effettuata per mezzo di cavi e adattatori e con il supporto di alcune librerie.

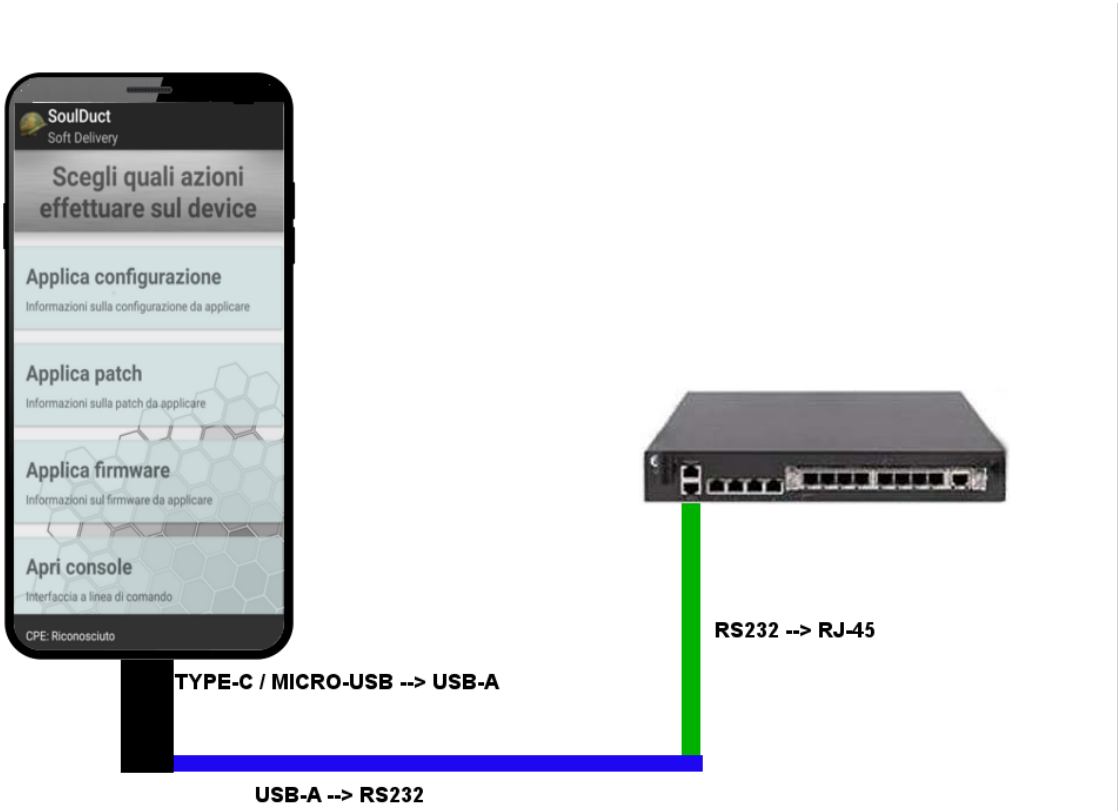


Figura 6.1: Interfacciare uno smartphone Android con un Cpe

## 6.1 Cavi e adattatori

Negli anni le tecnologie seriali si sono evolute, e sono ancora oggi in continua evoluzione, basti pensare alla recente specifica usb-c che ha preso il posto della micro-usb nei comuni smartphone.
Questa evoluzione consente di trasferire i dati ad una maggiore velocità e in maniera ottimizzata rispetto alle vecchie specifiche, ma di conseguenza si devono prevedere delle strategie per garantire la retrocompatibilità.
E' molto difficile trovare in commercio pc che possiedano porte RS232, mentre è molto comune trovarne con porte USB-A e USB-C. Per questi motivi la connessione tra il dispositivo Android e il Cpe deve essere correttamente predisposta tramite opportuni **adattatori**, chiamati anche **driver** o **convertitori**.

La **big picture** del collegamento è illustrata in Figura 6.1, l'obiettivo è connettere la porta USB di uno smartphone Android con la **console port** di un Cpe, tramite una **catena di adattatori**.

### 6.1.1 Adattatore 1: USB-C –> USB-A

I dispositivi Android, e in particolar modo gli smartphone, dispongono di porte usb, per la ricarica del dispositivo e per il trasferimento seriale dei dati.
Queste porte si distinguono in base al tipo di connettore: i più comuni sono **micro-usb** e **usb-c**, i quali hanno dimensioni ridotte rispetto alla comune USB-A. Il collegamento, **in primo luogo**, deve prevedere un adattatore **USB-C –> USB-A** oppure, nel caso si tratti di uno smartphone più datato, di uno **MICRO-USB –> USB-A**.
Si supponga di disporre del primo, anche se il ragionamento è facilmente estendibile al secondo caso.

Figura 6.2: Primo adattatore della catena USB-C –> USB-A

Grazie ad esso, lo smartphone Android è in grado di accogliere una qualsiasi periferica con interfaccia **USB-A (maschio)**.

### 6.1.2 Adattatore 2: USB-A –> RS232 DB-9

A questo punto è possibile collegare l'adattatore **USB-A –> RS232 DB-9** (Figura 6.3), il quale permette di avere a disposizione una interfaccia **RS-232 (femmina)**.

Figura 6.3: Secondo adattatore USB-A –> RS232 DB-9

### 6.1.3 Adattatore 3: RS232 –> RJ45

L'ultimo cavo è di tipo ethernet, ed è utilizzato per collegare la porta rs232 alla **Console port** del CPE con interfaccia rj45.

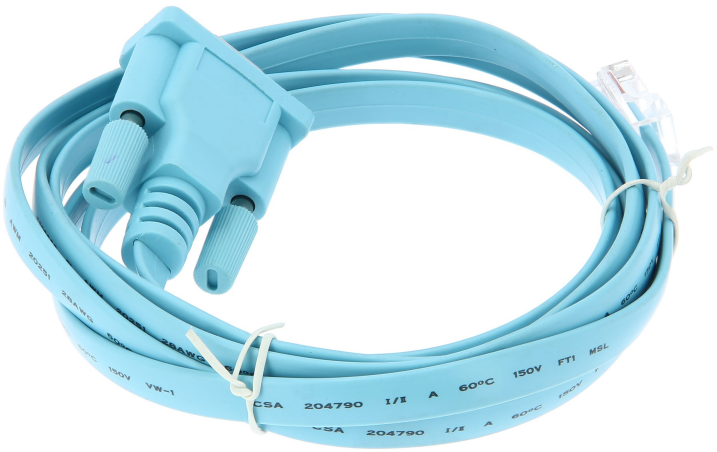


Figura 6.4: Terzo adattatore rs232 –> rj45

## 6.2 Osservazione

Lo schema di adattatori presentato in questo capitolo dipende dal dispositivo android nel quale viene installata l'app e dall'apparato CPE.
Il sistema necessita quindi dell'utilizzo dei giusti adattatori in base **contesto nel quale viene utilizzato**.

Per **esempio**, la console port di alcuni apparati ha una interfaccia seriale rs-232, mentre alcuni dispositivi Android possiedono una porta micro-usb.

## 6.3 Metodi per I/O

In questa sezione viene mostrato come l'app android si **connette** e **comunica** con il CPE, vengono inoltre riportati alcuni frammenti di codice relativi alle parti più importanti di questi aspetti.

Il progetto è realizzato sfruttando l'interfaccia **UART** di Android, in particolare, tramite l'uso della libreria **UsbSerial**, che implementa i **driver** per gli adattatori RS-232 più diffusi e fornisce un livello di astrazione ripetto alle chiamate **write** e **read** nella comunicazione seriale. [1].

L'uso del componente **Service** [2] di Android permette di svolgere in background tutte le operazioni relative alla comunicazione seriale, e rimane attivo finchè l'**Activity** principale **UsbService** non viene distrutta.
Quando l'utente inserisce il cavo nella porta usb del dispositivo Android, viene chiamato UsbService. Esso richiede i permessi di utilizzo della periferica Usb, mediante chiamate di sistema.

```
private void requestUserPermission() {
        PendingIntent mPendingIntent = PendingIntent
                .getBroadcast(this,
                        0,
                        new Intent(ACTION_USB_PERMISSION),
                        0);
        usbManager.requestPermission(device, mPendingIntent);
    }
```

La richiesta dei permessi Usb, l'inserimento del cavo usb e la sua rimozione, sono controllati dal componente **BroadcastReceiver** [3] che riceve e gestisce gli Intent inviati in broacast:

```
private final BroadcastReceiver usbReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            switch(intent.getAction()){
                case ACTION_USB_PERMISSION:
                    boolean granted = intent
                       .getExtras()
                       .getBoolean(UsbManager
                                .EXTRA_PERMISSION_GRANTED);
                    manageActionUsbPermission(granted, context);
                    break;
                case ACTION_USB_ATTACHED:
                    if (!serialPortConnected)
                        findSerialPortDevice();
                    break;
                case ACTION_USB_DETACHED:
                    Intent intentDisconnected = new Intent(
                                                ACTION_USB_DISCONNECTED);
                    context.sendBroadcast(intentDisconnected);
                    if (serialPortConnected) {
                        serialPort.close();
```

[1] <https://github.com/felHR85/UsbSerial>
[2] <https://developer.android.com/guide/components/services>
[3] <https://developer.android.com/reference/android/content/BroadcastReceiver>

```
22                      }
23                      serialPortConnected = false;
24              }
25          }
26      };
```

Nel caso in cui l'utente conceda i permessi Usb e l'adattatore sia supportato, viene creato e avviato il **thread** di connessione al CPE.
In questa fase vengono anche **impostati i parametri** utili alla comunicazione seriale, quali **baud rate**, **data bits**, **stop bits**, **parity bits** e **flow control**.

```
1  private class ConnectionThread extends Thread {
2          @Override
3          public void run() {
4              serialPort = UsbSerialDevice
5              .createUsbSerialDevice(device, connection);
6              if (serialPort != null) {
7                  if (serialPort.open()) {
8                      serialPortConnected = true;
9                      serialPort.setBaudRate(BAUD_RATE);
10                     serialPort.setDataBits(UsbSerialInterface
11                                                 .DATA_BITS_8);
12                     serialPort.setStopBits(UsbSerialInterface
13                                                 .STOP_BITS_1);
14                     serialPort.setParity(UsbSerialInterface
15                                                 .PARITY_NONE);
16                     serialPort.setFlowControl(UsbSerialInterface
17                                                 .FLOW_CONTROL_OFF);
18                     serialPort.read(mCallback);
19                     .
20                     .
21                     .
```

La **Callback** impostata alla riga **18** permette al sistema di accogliere i messaggi in ingresso, i messaggi in arrivo dal CPE vengono letti in modo asincrono tramite un **Handler**[4].

```
1  @Override
2  public void handleMessage(Message msg) {
3      switch (msg.what) {
4          case UsbService.MESSAGE_FROM_SERIAL_PORT:
5              String data = (String) msg.obj;
6              mFragment.get().lastOutput += data;
7              break;
```

Infine, per inviare i dati sulla porta seriale viene utilzzato il **metodo write**, su cui viene effettuato un **overloading** per permettere di inviare con semplicità oggetti di tipo String.

```
1  public void write(String data){
2          write(data.getBytes());
3  }
4
5  public void write(byte[] data) {
6      if (serialPort != null)
7          serialPort.write(data);
8  }
```

[4] <https://developer.android.com/reference/android/os/Handler>

## 6.4 Riconoscimento automatico del CPE

Una volta predisposti i metodi per la lettura e scrittura sulla porta seriale diventa possibile effettuare una comunicazione ad alto livello con il CPE.
Il processo di riconscimento del CPE deve, in primo luogo, estrarre informazioni riguardo al **Vendor** e al **Model** dell'apparato.
Per effettuare questo processo vengono inviati dei comandi e analizzate le risposte tramite **espressioni regolari**.

Se il CPE collegato **corrisponde** con quello associato alla sede aziendale per l'attivazione, si apre automaticamente il menù per la configurazione.

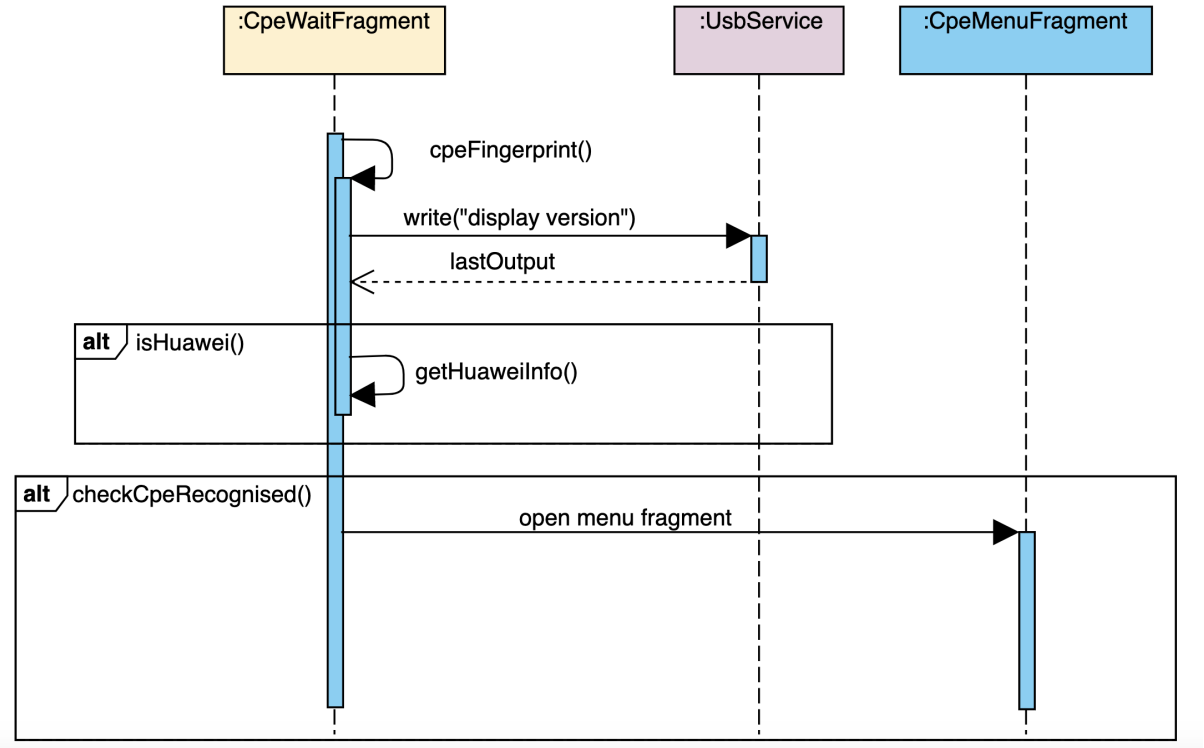


Figura 6.5: Diagramma di sequenza

In figura 6.5 viene mostrato il diagramma di sequenza dello scenario di riconscimento automatico. Il Fragment CpeWaitFragment ha il compito di attendere l'inserimento del cavo seriale e chiedere i permessi all'utente.
Successivamente viene eseguito il metodo **cpeFingerprint()**, il cui compito è riconoscere il Vendor e il Model dell'apparato collegato.
A priori non è possibile sapere se è stato collegato un router **Huawei**, **Cisco** o **Juniper**, si procede quindi per tentativi.

Ogni apparato possiede il proprio set di comandi per la visualizzazione del vendor e del model. Nel diagramma viene mostrato l'esempio di Huawei, esso prevede l'uso del comando **display version**, il quale restituisce una stringa contenente le informazioni che si vogliono cercare.

### 6.4.1 Device figerprint

La **device fingerprint**[5] si riferisce ad un processo di raccolta sulle informazioni di un dispositivo remoto per l'identificazione, il nome deriva dalla somiglianza con l'impronta digitale usata per l'identificazione. Il metodo principale che viene avviato in questa fase è il metodo **cpeFingerprint()**, esso esegue il processo appena descritto e controlla in primo luogo se il dispositivo connesso è un huawei, cisco o juniper.

In questo caso, ai fini di realizzare un prototipo che dimostri la fattibilità del progetto, è stato analizzato solo il vendor Huawei, ma il ragionamento è estendibile anche ad altri.

```
private void cpeFingerprint() {
        if(isHuawei()) {
            connectedCpe = getHuaweiInfo();
        }else if(isCisco()){
            // Retrive Cisco information
        }else if(isJuniper()){
            // Retrive Juniper information
        }
    }
```

Per verificare se il vendor sia Huawei o meno si invia un comando di **display version**, funzionante sui router huawei, e si controlla che l'output contenga "Huawei".

```
private boolean isHuawei() {
        usbService.write("display version\n" + NEWLINE);
        sleep();
        if(lastOutput.contains("Huawei"))
            return true;
        return false;
    }
```

Una volta verificato che l'apparato collegato è un Huawei, si possono inviare comandi più specifici, per verificare il modello, la versione del software, il numero seriale e altre informazioni, utilizzando le **espressioni regolari**.

```
private CpeInfo getHuaweiInfo() {
        CpeInfo cpeInfo = new CpeInfo();
        cpeInfo.setVendorName("Huawei");
        cpeInfo.setModel(searchModel("Quidway (.*?)\\s", lastOutput));
        cpeInfo.setSoftwareVersion(
                    searchSoftwareVersion("(.* Software, Version .*)",
                                          lastOutput));
        .
        .
```

[5] <https://it.wikipedia.org/wiki/Device_fingerprint>

Infine si controlla se le informazioni recuperate dalla device fingerprint **corrispondono** con quelle sul CPE associato alla sede aziendale da attivare, tramite l'utilizzo del metodo equals di CpeInfo.

```
private void checkCpeRecognised() {
        if (connectedCpe.equals(expectedCpe)) {
            Intent intent = new Intent(UsbService.FINGERPRINT_ACCEPT);
            getActivity().sendBroadcast(intent);
        } else {
            Intent intent = new Intent(UsbService.FINGERPRINT_REJECT);
            getActivity().sendBroadcast(intent);
        }
    }
```

In questo processo viene costruito un oggetto **DTO - Data Transfer Object** di tipo CpeInfo, che contiene le informazioni riguardo Model e Vendor, ma che è possibile estendere facilemente.

```
public class CpeInfo {
    private String vendorName;
    private String model;
    private String softwareVersion;
    private String osFamily;
    private String upTime;
    private String lastReboot;
    private String processor;
    private String configurationMemory; // [bytes]
    private String flashSize; // [bytes]
    private String hostName;
    private String deviceSerialNumber;
    private String deviceName;

    .
    .

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof CpeInfo)) return false;
        CpeInfo cpeInfo = (CpeInfo) o;
        return Objects.equals(getVendorName(),
                              cpeInfo.getVendorName()) &&
                Objects.equals(getModel(), cpeInfo.getModel());
    }

    @Override
    public int hashCode() {
        return Objects.hash(getVendorName(), getModel());
    }
}
```

Se i due oggetti di tipo CpeInfo corrispondono in base alla relazione di equivalenza descritta nel metodo equals, allora si procede nella apertura del menù di configurazione.

## 6.5 Iniezione della configurazione

Una volta che il CPE è stato correttamente riconosciuto è possibile automatizzare il processo di configurazione.

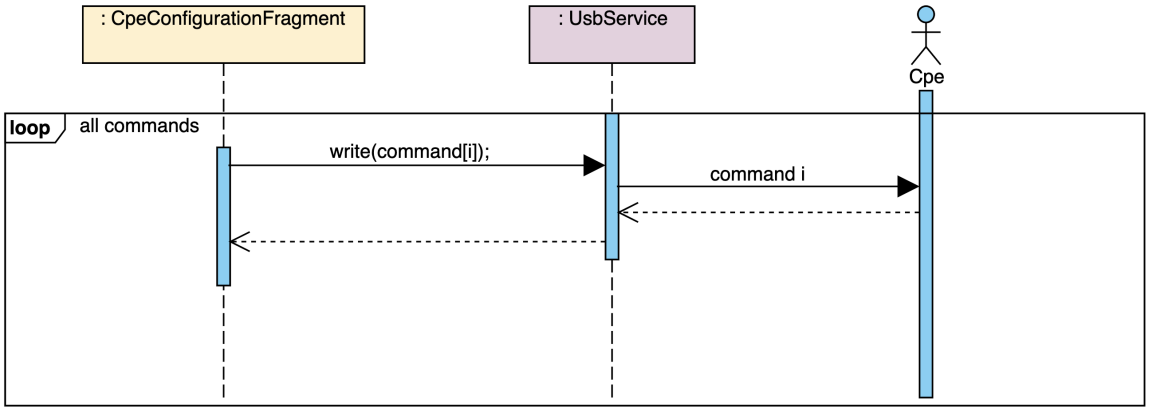


Figura 6.6: Diagramma di sequenza

In termini pratici, la riconfigurazione presentata in questa sezione riguarda il caso d'uso "configurazione a caldo", poichè i comandi vengono eseguiti tramite la **Cli - Command line interface**.

Le operazioni eseguite in questa fase **sostituiscono** quelle del system integrator, in questo senso il processo di configurazione è completamente automatico dato che il sistema conosce in anticipo i comandi da eseguire, provenienti dal web service.

Il seguente metodo è dedicato all'invio dei comandi di configurazione contenuti nella lista **commands**.

```
1 private void applyConfiguration(){
2     for(String command: commands)
3         usbService.write(command + NEWLINE);
4 }
```

I comandi possono essere estratti in precedenza da un **file di testo**, in seguito è presente un **esempio** di configurazione.

```
1     #
2     sysname tesiTriennale
3     clock timezone 1 add 3:00:00
4     header login information Tesi di laurea triennale i
5     #
6     interface ethernet0/0
7         port link-mode route
8         description TESI_DI_LAUREA_TRIENNALE
9         ip address 10.100.130.2 255.255.0.0
10        dns server 140.3.255.254
11    .
12    .
```

# Parte III

# Il prototipo

# Capitolo 7

# Presentazione App Android

In questa sezione viene presentato il **prototipo** dell'applicazione **SoulDuct**.
Per presentare l'app viene utilizzato uno smartphone con Android 7.0 e un router **Huawei Quidway AR-1915**.

La prima schermata, dopo aver aperto l'applicazione si presenta come in figura 7.1. Viene chiesto all'utente di inserire il codice cliente per l'attivazione: la **SnackBar** in fondo allo schermo avvisa se il cliente è stato trovato o meno.
Una card card con il nome del cliente viene mostrata per continuare l'attivazione, in questo esempio viene mostrata "Cybaze".

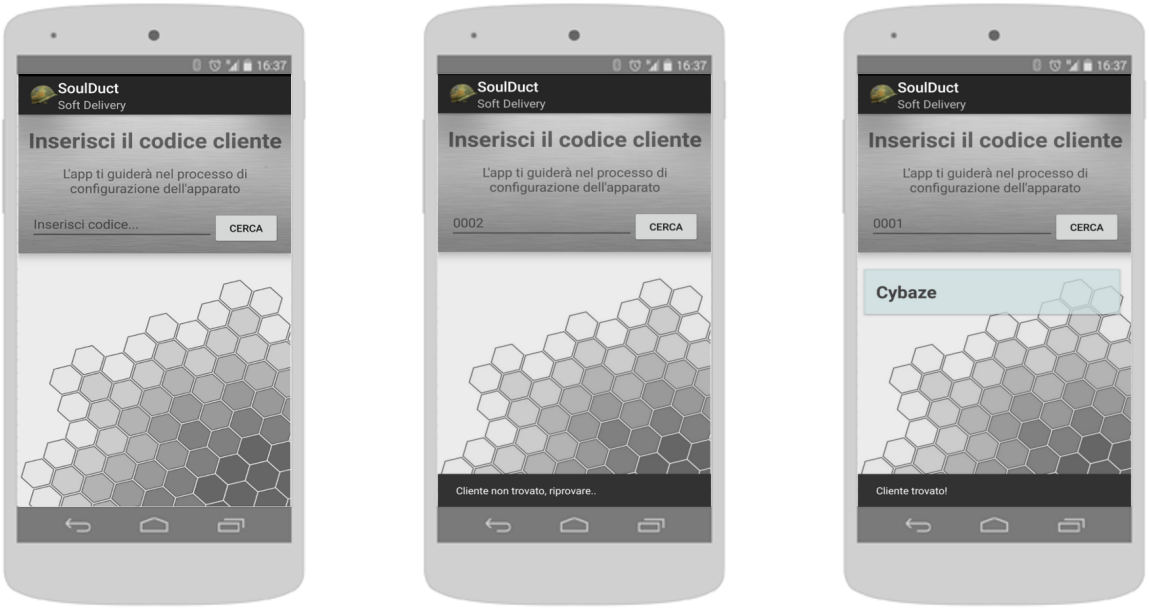


Figura 7.1: Inserire il codice cliente

Una volta selezionata l'azienda, l'app mostra tutte le **sedi in preattivazione** (Figura 7.2) associate a quest'ultima. Ogni sede corrisponde ad un CPE da configurare.


Figura 7.2: Sedi in preattivazione

Selezionando una sede, per esempio quella di **Milano**, l'applicazione aspetta che venga collegato il router **Huawei AR 19-15**, associato alla sede, tramite cavo seriale. Se l'utente concede i permessi usb, la snackbar avvisa che la connessione al dispositivo è riuscita.

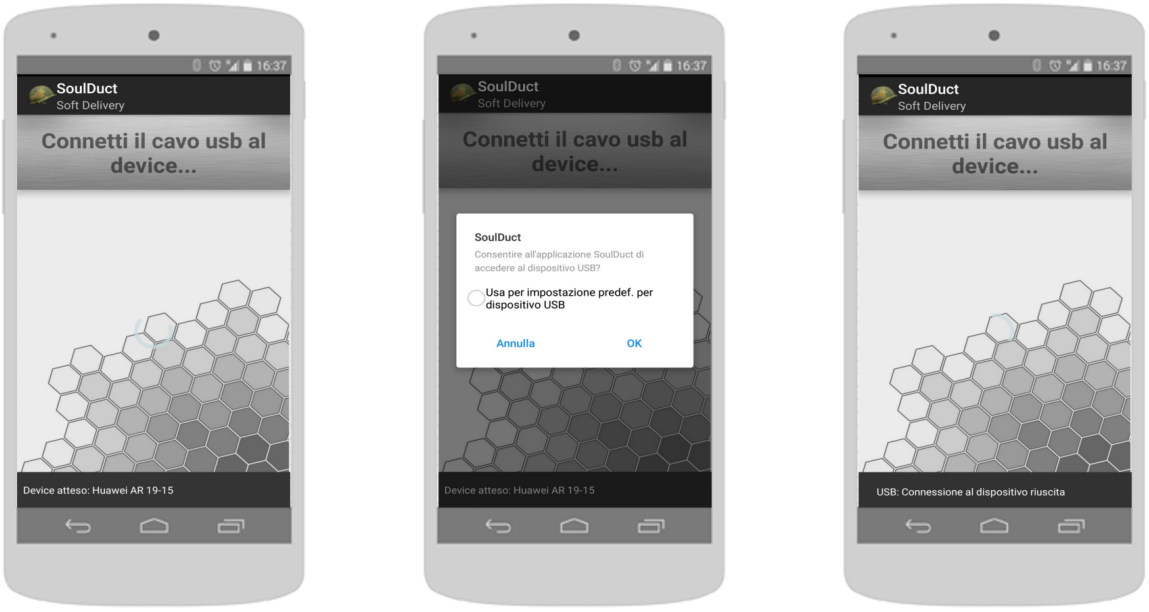


Figura 7.3: Attesa del collegamento

Il sistema, quando la connessione al dispositivo è andata a buon fine, avvia il processo di **device fingerprinting**: se il dispositivo collegato corrisponde a quello atteso per la sede aziendale allora procede con l'apertura del menu di configurazione.


Figura 7.4: Menu

Tramite il menù è possibile avviare aprire la **console** tramite la quale interagire con un interfaccia a linea di comando con il router, oppure avviare la ri-configurazione.

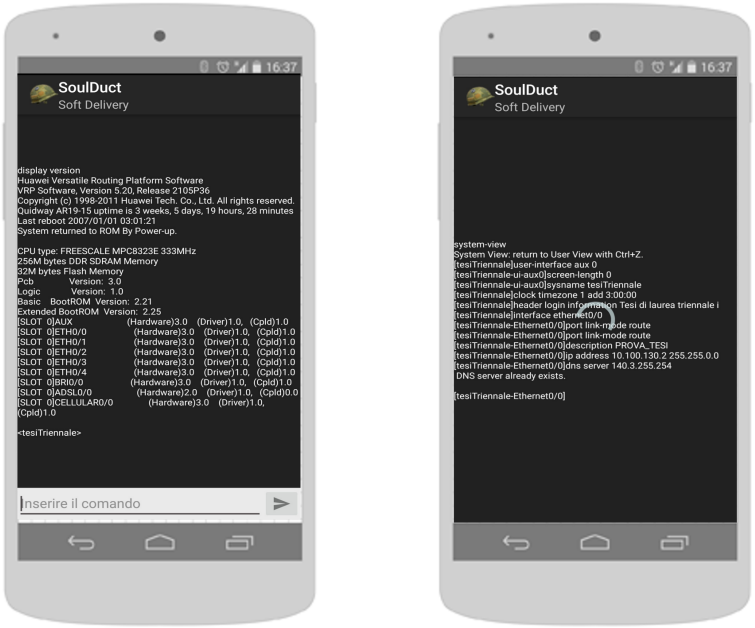


Figura 7.5: Console e configurazione

# Capitolo 8

# Conclusione e sviluppi futuri

Al fine di costruire il prototipo, è stato analizzato il vendor Huawei, in particolare, l'analisi è stata condotta nel contesto di un router Huawei AR-1915, messo gentilmente a disposizione da Cybaze. Di conseguenza, il riconoscimento automatico del vendor è relativo esclusivamente a Huawei. Questo può comunque essere considerato un obiettivo raggiunto, poichè si è dimostrata la fattibilità di riconsoscere automaticamente un CPE tramite porta seriale; oltretutto il ragionamento per estendere il riconoscimento ad altri vendor è molto simile, si tratta solamente di adattare i comandi utilizzati per identificare il vendor. Ad esempio per identificare un CPE Cisco è sufficiente inviare il comando "show ver" e ricercare "Cisco" nell'output del comando. E' possibile utilizzare logiche analoghe anche per gli altri vendor.

Il prototipo realizzato, inoltre, ha raggiunto come obiettivo quello di aprire l'interfaccia a linea di comando del CPE tramite porta seriale e automatizzare il processo di comunicazione con esso, eseguire comandi in modo automatico e analizzare l'output del CPE tramite espressioni regolari.

L'invio di file, invece, può essere considerato un obiettivo parzialmente raggiunto, in quanto inizialmente è stato previsto tramite porta seriale, ma successivamente, per un problema dovuto sia al particolare dispositivo esaminato, sia perchè un futuro sviluppo avrebbe previsto l'uso dell'interfaccia Ethernet per caricare il firmware sfruttando di trasferimento fino a 10Gpbs, è stato di proposito lasciato in sospeso.

Il lavoro svolto ha richiesto oltre 1300 linee di codice distribuite in 12 classi java, e lo studio della programmazione Java in ambiente Android.

In conclusione il lavoro svolto ha permesso di realizzare un dimostrativo in grado di automatizzare operazioni che oggi i system integrator delle Telco svolgono manualmente.

Il prototipo, nello stato attuale, è solamente un punto di partenza di un progetto più grande. Esso richiede un analisi approfondita di una varietà di apparati di vendor distinti, e le peculiarità che rendono ogni apparato diverso dagli altri. Inoltre il lavoro richiede di svolgere una fase di test, prima di essere messo in produzione e l'implementazione di funzionalità per l'interazione con il web service per recuperare la corretta configurazione/firmware, sfruttando la connessione dati del dispositivo Android.

# Bibliografia


[1] Internet Engineering Task Force (IETF). *Secure Zero Touch Provisioning (SZTP)*. URL: https://tools.ietf.org/html/rfc8572#page-16. (accessed: 12.04.2020).

[2] Kristin Marsicano Bill Phillips Chris Stewart. *Android Programming: The Big Nerd Ranch Guide*. Addison-Wesley Professional, 2017. ISBN: 978-0134706054.

[3] felHR85. *Usb Serial*. URL: https://github.com/felHR85/UsbSerial. (accessed: 13.03.2020).

[4] Google. *UART*. URL: https://developer.android.com/things/sdk/pio/uart. (accessed: 1.05.2020).

[5] Chris Lichko Helmut Zurner. *ZTA Congress 2018 - Zero Touch Provisioning for Infrastructure and Virtual Apps*. URL: https://www.youtube.com/watch?v=ZIH_UOQj70s. (accessed: 10.05.2020).

[6] Louis E. Frenzel Jr. *Handbook of Serial Communications Interfaces*. Kidlington (UK), Elseviers, 2015. ISBN: 9780128006290.

[7] Jan-Willem Keinke. *What is Zero Touch Provisioning and is it useful for me?* URL: https://www.infradata.com/news-blog/what-is-zero-touch-provisioning/. (accessed: 4.05.2020).

[8] Craig Larman. *Applicare UML e i pattern. Analisi e progettazione orientata agli oggetti*. Pearson, 2016. ISBN: 978-8891901033.

[9] Giovanni Zamengo Leonardo Chiarion. *Router Huawei. Configurazione e utilizzo*. Apogeo, 2006. ISBN: 978-8850324705.