ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

UN DISTRIBUTORE INTELLIGENTE DI BEVANDE ALLA SPINA CON SISTEMA DI PAGAMENTO BASATO SU BITCOIN

Elaborata nel corso di: Programmazione di Sistemi Embedded

Tesi di Laurea di:
LUDOVICO DE NITTIS

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2014–2015
SESSIONE II

PAROLE CHIAVE

Bitcoin
IoT
Raspberry Pi
Vending Machine
Distributore Intelligente

"Computers are like Old Testament gods; lots of rules and no mercy." - Joseph Campbell

Indice

Introduzione

I distributori automatici sono attualmente molto diffusi perché permettono di automatizzare l'erogazione di un prodotto o servizio senza la necessità di avere una persona addetta a quello scopo.

Il problema di fondo di tali sistemi rimane comunque la continua necessità di manutenzione dovuta sia alla raccolta delle monete utilizzate dai clienti sia per il rifornimento dei prodotti.

Questa tesi si pone l'obiettivo di progettare e sviluppare un'applicazione per sistemi embedded Linux che consenta agli utenti di richiedere, in modo autonomo, il pagamento tramite Bitcoin con la successiva erogazione di bevande alla spina.

Utilizzando i Bitcoin come metodo di pagamento si risolve completamente il problema dovuto alla raccolta delle monete. Invece per mitigare il problema del rifornimento dei prodotti i distributori saranno in grado di controllare il proprio quantitativo rimanente e, grazie al collegamento alla rete LAN, potranno comunicare tale informazione direttamente agli addetti alla manutenzione. I distributori saranno anche in grado di comunicare tra loro per informare gli utenti a quale distributore più vicino debbano recarsi nel caso in cui uno si esaurisca.

Nel primo capitolo viene introdotto lo stato dell'arte attuale per quanto riguarda i distributori automatici. Nel secondo capitolo vengono spiegati Internet of Things, Bitcoin e i vantaggi che si possono avere utilizzandoli. Dal terzo capitolo viene presentato il caso di studio con i relativi requisiti funzionali. Successivamente viene effettuata l'analisi, una panoramica sulle tecnologie utilizzate, la progettazione e l'implementazione. Nell'ottavo capitolo vengono esposti i controlli effettuati per verificare la corretta implementazione del progetto. Nell'ultimo capitolo ci sono i possibili futuri sviluppi del progetto e le conclusioni.

Capitolo 1

Stato dell'arte

1.1 Introduzione e cenni storici

Un distributore automatico è una macchina che distribuisce prodotti come per esempio snack, bevande, biglietti e sigarette. I prodotti vengono erogati in modo automatico dopo il pagamento da parte dei clienti che, in genere, può avvenire utilizzando i contanti o le carte di credito.

La prima macchina che possa essere definita distributore automatico è stata inventata da Erone di Alessandria, un ingegnere e matematico greco. Nel 219 a.C. realizzò una macchina che accettava monete e successivamente erogava acqua utilizzata per le cerimonie nei templi. Quando una moneta veniva inserita cadeva su una leva che, grazie ad una valvola, ditribuiva una quantità prefissata di acqua [10].

Il meccanismo con cui i distributori erogano il prodotto acquistato può essere di diverso tipo, ad esempio:

- Fanno cadere il prodotto in un compartimento aperto raggiungibile dai clienti.
- Rilasciano un contenitore usa e getta in cui viene erogata la bibita acquistata.
- Viene preparato il prodotto/servizio sul momento, come per esempio la stampa di un ticket per il parcheggio.

I primi distributori automatici in Italia risalgono al 1953. Furono importati dagli USA e distribuivano Coca Cola.

Secondo l'ultima indagine di mercato redatta dalla European Vending Association (EVA) si conta che solo in Europa ci siano approssimativamente più di 3.7 milioni di distributori automatici [3].

1.2 Distributori disponibili

La maggior parte dei distributori automatici attualmente disponibili accettano come forma di pagamento le monete. I più sofisticati e recenti hanno aggiunto anche la possibilità di pagare con le carte di credito.

Alcuni nuovi distributori dispongono di un sistema che tiene traccia del numero di prodotti che sono stati venduti e sono in grado di capire quando la quantità residua sarà prossima all'esaurimento. Questo tipo di distributori di solito necessitano di essere collegati ad internet per poter inviare delle notifiche al personale addetto alla manutenzione.

1.2.1 Rilevazione dati anagrafici

In Italia i distributori di prodotti vietati ai minori, come ad esempio le sigarette e gli alcolici, devono prevedere la rilevazione dei dati anagrafici del cliente utilizzando sistemi di lettura dei documenti oppure devono essere presenziati da personale addetto ad effettuare controlli sui dati anagrafici [1].

1.2.2 Brevetti sulle vending machines

Diverse funzionalità dei distributori automatici sono state bevettate come ad esempio:

- Il brevetto *US 6575363 B1* che prevede l'utilizzo di un proof-of-purchase (POP) contenente uno sconto su un eventuale prossimo acquisto [11]. Per lo scopo i distributori devono essere in grado di verificare la correttezza del POP e di distribuire i prodotti a prezzi variabili (con o senza sconto). Utilizzando questo metodo è possibile incentivare gli utenti a non limitarsi ad un singolo acquisto ma a diventare clienti abituali.
- Il brevetto *US 7490054 B2* che propone l'utilizzo di tag RFID per aiutare i clienti nello scegliere i prodotti che desiderano acquistare [15].

I clienti possono scegliere un determinato prodotto e il distributore, utilizzando uno scanner RFID, visualizza a schermo le informazioni relative a quel determinato prodotto.

- La richiesta di brevetto *US 20130035787 A1*, in modo simile al brevetto *US 7490054 B2*, che prevede di utilizzare uno schermo presente sul distributore per visualizzare un quick response (QR) code associato ai vari prodotti [6]. I clienti, ad esempio utilizzando il proprio cellulare, potranno leggere il QR code che li manderà su una pagina internet contenente alcune informazioni sui prodotti come i valori nutrizionali e gli ingredienti.

1.2.3 Distributori intelligenti

Dalla mia ricerca è emerso che attualmente sono state avanzate diverse proposte per rendere i distributori automatici maggiormente intelligenti e funzionali.

Una tra le proposte più rilevanti è stata la pubblicazione redatta nel 2014 dalla Intel, in collaborazione con la ADLINK Technology, su come aumentare i profitti dalle vending machine utilizzando l'Internet of Things (IoT) [9]. Connettendo in rete i distributori e utilizzando processori maggiormente performanti, in grado di eseguire un numero maggiore di operazioni, è possibile incrementare il guadagno potendo contare su minori costi operativi, una messa in esercizio semplificata, un incremento delle vendite grazie a sconti mirati e a nuove fonti di guadagno derivanti da pubblicità diretta.

In un'altra pubblicazione redatta dall'AT&T vengono esposti i vantaggi che si otterrebbero avendo i distributori automatici connessi ad Internet e di come il mondo IoT stia rivoluzionando le vending machines [14]. Tra i vantaggi espressi dall'AT&T troviamo la possibilità di effettuare pagamenti senza l'utilizzo di contanti, una migliore opportunità di marketing, l'automatizzazione nella manutenzione e il rifornimento dei distributori in base alla domanda corrente.

Capitolo 2

Sistemi Internet of Things e Bitcoin

2.1 Internet of Things

L'Internet of Things (IoT, in italiano "Internet delle cose") è la rete di oggetti fisici con elettronica, software, sensori e connessione di rete che permette loro di acquisire e trasferire dati. L'IoT consente alle "cose" di essere monitorate e controllate da remoto ottenendo una visione maggiormente dettagliata dell'ambiente circostante. Secondo una recente ricerca si stima che i dispositivi connessi saranno almeno 50 miliardi nel 2020 [8].

Il termine "Internet of Things" è stato introdotto da Kevin Ashton nel 1999 [4]. I dispositivi che rientrano nella definizione di IoT vanno dai dispositivi wearable intelligenti fino alle automobili con diversi sensori al loro interno.

Gli oggetti diventano rintracciabili e intelligenti potendo contare sullo scambio di dati anche con altri dispositivi. Per questo l'IoT non è semplicemente aggiungere un tag RFID ai dispositivi passivi, ma si tratta di aggiungere una qualche forma di intelligenza, in modo che acquisiscano un ruolo attivo e possano svolgere un numero maggiore di funzioni [13].

Le architetture alla base di Internet of Things non sono standardizzate, ma in generale si può suddividere la struttura in livelli. Al primo livello ci sono i sensori e gli attuatori che permettono la raccolta dei dati e una interazione con il mondo esterno. Come rappresentato in figura 2.1, i dati così raccolti verranno inviati e gestiti dagli strati superiori.

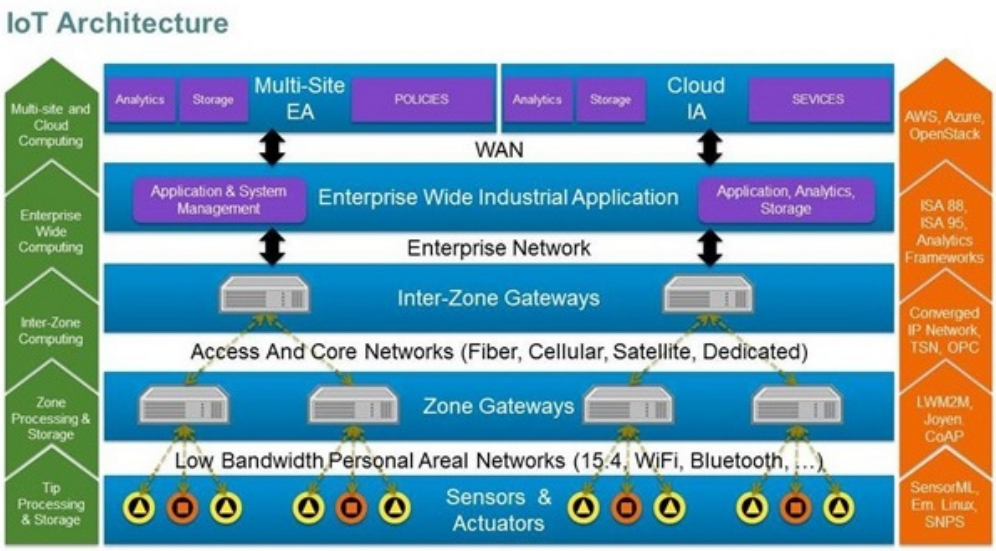

Figura 2.1: Esempio di tipica architettura IoT

Al fine di realizzare uno standard in ambito IoT è attualmente attivo il progetto denominato IEEE P2413 [2].

2.2 Bitcoin

2.2.1 Cosa sono i Bitcoin e la loro storia

Bitcoin è la prima implementazione del concetto di "cryptocurrency", descritto inizialmente da Wei Dai nel 1998 sotto il nome di "b-money" [7]. Wei Dai suggerisce l'idea di una nuova moneta che utilizzi la crittografia per controllare la sua creazione e le transazioni invece di un'autorità centrale, come ad esempio le banche.

L'invenzione di Bitcoin è stata pubblicata nel 2008 con un whitepaper da Satoshi Nakamoto [12] e successivamente nel 2009 è stata rilasciata la prima implementazione del client open-source Bitcoin-core. Satoshi Nakamoto ha successivamente lasciato il progetto nel 2010 senza mai rivelare nessuna informazione personale, infatti anche il suo nome potrebbe essere uno pseudonimo. Da allora è la comunità open-source che sta portando avanti lo sviluppo.

Figura 2.2: Logo Bitcoin

Dal punto di vista degli utenti finali Bitcoin non è nient'altro che un programma per PC, un'app per mobile o un account su un sito che permette di avere un wallet digitale Bitcoin e consente di inviare e ricevere Bitcoin attraverso Internet.

Per tenere traccia di tutte le transazioni, e quindi del numero di Bitcoin presenti nei vari indirizzi, viene utilizzato un registro pubblico e condiviso chiamato "blockchain". Il blockchain è una lista contenente tutte le transazioni Bitcoin confermate che sono avvenute da quando è stata rilasciata la prima implementazione nel 2009. Le transazioni presenti sul blockchain sono composte da indirizzi mittenti, destinatari e la quantità di Bitcoin che è stata trasferita. I programmi Bitcoin per poter controllare il saldo attualmente disponibile sui wallet personali scaricano localmente l'intero blockchain o si affidano a terzi evitando di occupare un gran quantitativo di memoria.

Dal 28 Novembre 2012 ogni circa 10 minuti, con la creazione di nuovi blocchi, vengono generati 25 nuovi Bitcoin. Il numero di Bitcoin che verranno generati viene dimezzato circa ogni 4 anni fino al raggiungimento del limite imposto arbitrariarmente di 21 milioni di Bitcoin che dovrebbe avvenire intorno al 2140.

2.2.2 Funzionamento nella pratica

I Bitcoin utilizzano il meccanismo di chiave pubblica e chiave privata. Quando si richiede la generazione di un nuovo indirizzo Bitcoin viene generata una chiave privata dalla quale verrà generato l'indirizzo Bitcoin vero e pro-

prio (la chiave pubblica). Se si desidera inviare una certa quantità di Bitcoin dal proprio indirizzo, si dovrà inviare un pacchetto contente l'indirizzo del mittente e quello del destinatario (possono anche essere molteplici) e il messaggio dovrà essere firmato utilizzando la propria chiave privata. Il messaggio verrà inviato in broadcast a tutti i nodi attualmente connessi sulla rete Bitcoin che avranno il compito di verificare che la richiesta sia avvenuta effettivamente dal mittente stesso. Per farlo utilizzano la chiave pubblica del mittente e controllano la validità della firma. Nel caso il messaggio non sia valido non viene inoltrato agli altri nodi della rete e viene semplicemente scartato. Quando un utente esegue un programma Bitcoin avrà a disposizione un wallet che comprende un insieme di più indirizzi con le relative chiavi private. Per questo motivo quando sono presenti dei Bitcoin in un wallet l'utente possiede il controllo su quei Bitcoin.

Ogni circa 10 minuti i "miner" creano un nuovo blocco contenente alcune delle transazioni che gli utenti hanno richiesto e firmato digitalmente. Il nuovo blocco andrà a far parte del blockchain e le transazioni al suo interno verranno considerate confermate. Il compito dei miner è cercare di creare un nuovo blocco contenente le transazioni valide non ancora confermate. Per rendere il blocco valido esso deve indicare l'hash del suo predecessore (l'ultimo blocco valido fino a quel momento) e il proprio hash deve iniziare con un numero prestabilito di bit a zero. Il numero di bit a zero cambia continuamente perché il protocollo Bitcoin cerca di mantenere una distanza di 10 minuti tra la creazione dei vari blocchi. Quindi se un numero maggiore di persone tenta di minare un nuovo blocco la probabilità che venga trovato prima dei 10 minuti cresce e di conseguenza il numero di bit a zero necessari potrebbe aumentare.

Quando un nuovo blocco viene minato, con esso vengono creati nuovi Bitcoin che vengono distribuiti come premio ai miner per il proprio contributo.

2.2.3 Confermare le transazioni e il double spending

La conferma di una transazione serve ad impedire il double spending, cioè che gli stessi soldi vengano spesi due volte. Dopo la propagazione di una transazione ai nodi Bitcoin, quando essa viene inclusa in un blocco, viene detto che la transazione è stata minata ad una profondità di 1 blocco. Con la successiva creazione di ogni altro blocco il numero della profondi-

tà aumenterà di uno ogni volta. Per essere sicuri che i Bitcoin ricevuti non vengano spesi una seconda volta dal mittente è necessario considerare la transazione confermata solo dopo una certa profondità di blocchi. Di solito i client Bitcoin mostrano le transazioni come confermate dopo aver raggiunto una profondità di 6 blocchi. Considerando che la media con cui i blocchi vengono creati è di 10 minuti, bisognerebbe aspettare circa 60 minuti per considerare confermate le transazioni. Fortunatamente riuscire ad effettuare un double spending è dispendioso e per questo motivo, per transazioni di piccola entità, si può tranquillamente evitare di aspettare la conferma di 6 blocchi. Infatti di solito gli esercenti che accettano i Bitcoin come pagamento, una volta che la transazione si è propagata e una percentuale consistente dei nodi ha ricevuto la transazione, la considerano già confermata senza aspettare neanche che venga inserita nel primo blocco.

2.3 Bitcoin nel mondo IoT

I Bitcoin sono particolarmente adatti al mondo IoT perché rispetto ai contanti e le monete permettono di automatizzare il processo di pagamento senza avere la necessità di dover andare periodicamente a raccogliere i soldi. Questo consente di avere tanti dispositivi a cui i clienti possono effettuare pagamenti senza avere l'obbligo di una manutenzione continua.

Rispetto al pagamento tramite carte di credito ci sono vantaggi sia per gli esercenti che per i clienti:

- I pagamenti effettuati utilizzando le carte di credito hanno dei costi di commissione per gli esercenti che tipicamente vanno dal 2% al 5%. I bitcoin invece hanno il vantaggio di avere costi sulle transazioni nulle, o fino a qualche centesimo.

- Per pagare con carta di credito è necessario strisciare la propria carta o, nel caso si tratti di carte contactless, avvicinarla e inserire il proprio eventuale PIN. Quando si tratta di pagamenti su dispositivi incustoditi, come possono essere i distributori di bevande, alcune persone evitano di usare la propria carta di credito per paura che la carta possa venire clonata a causa di una macchinetta compromessa. Utilizzando i Bitcoin questo problema non si ha in quanto, per il pagamento, verrà generato un QR code contenente l'indirizzo a cui

l'utente dovrà effettuare il pagamento. La transazione vera e propria avverrà sul dispositivo in possesso dell'utente e quindi, anche se la macchinetta dovesse essere stata compromessa, nel peggiore dei casi il cliente perderà i soldi che ha inviato per il pagamento ma il suo wallet non potrà mai venire intaccato.

Capitolo 3

Caso di Studio: Bit2Draft

3.1 Introduzione

L'idea alla base del progetto è quella di realizzare un distributore per bevande alla spina intelligente e maggiormente autonomo rispetto a quelli comunemente in commercio.

Con l'ausilio di sensori e dell'accesso ad Internet si è riusciti a fornire ai clienti informazioni utili in tempo reale sullo stato del distributore e su quello dei loro vicini. Infatti le macchinette sono in grado di tenere traccia del proprio quantitativo di bevande ancora a disposizione e di comunicarlo ai clienti attraverso un display. Tale informazione viene anche inviata ad un server centrale per poter rimanere sempre a conoscenza dello stato di tutti i distributori.

A differenza dei modelli attualmente in commercio i distributori, passando dal server centrale, si scambiano messaggi in modo da poter comunicare ai clienti non solo le informazioni relative alla propria quantità di prodotti rimanenti ma anche quella dei distributori che si trovano nelle vicinanze.

Per evitare che i dipendenti debbano continuamente controllare personalmente il livello di bevande rimanente, i distributori inviano una notifica in modo autonomo via email se il livello di bevande rimaste è prossimo all'esaurimento.

Come scritto in precedenza nella sezione 2.3 i distributori automatici classici hanno bisogno di una manutenzione continua per prelevare le monete utilizzate dai clienti negli acquisti. Invece utilizzando i Bitcoin si

risolve completamente questo problema rendendo i distributori ancora più autonomi.

3.2 Misure di sicurezza

I distributori dovranno poter operare anche in locali incustoditi e comunicheranno utilizzando una rete LAN con un server centralizzato. Per questo motivo si è scelto di utilizzare TLS per tutte le comunicazioni in modo da poter garantire la riservatezza e l'autenticità dei messaggi anche nel caso in cui qualcuno riuscisse a collegarsi alla rete LAN.

Bisogna poter garantire agli utenti l'erogazione del prodotto che acquistano. Nel caso in cui avvenga un problema software o hardware l'eventuale somma già pagata dai clienti deve poter essergli restituita.

Il server centrale utilizzerà un DBMS per il salvataggio dei dati. Grazie a backup periodici e alle funzionalità offerte dai dbms come il controllo sugli accessi, la verifica sull'integrità dei dati, la criptazione e l'atomicità delle transazioni, si potrà garantire la sicurezza delle informazioni.

3.3 Tipologia di prodotto

In Italia è consentito distributire bevande analcoliche senza l'utilizzo di particolari accorgimenti oltre quelli igienici. Invece se si vuole distribuire alcolici è necessario identificare i clienti ad esempio attraverso la lettura dei documenti.

Per questo motivo il distributore alla spina intelligente che si progetterà si ipotizza essere orientato all'erogazione di bevande analcoliche come acqua e succhi di frutta.

Capitolo 4

Analisi dei requisiti

In questo capitolo si descrivono i requisiti che l'applicazione deve rispettare e le funzionalità che deve implementare.

4.1 Descrizione del progetto

Il progetto consiste nella realizzazione di un'applicazione per sistemi embedded che consenta agli utenti di richiedere, in modo autonomo, il pagamento e la successiva erogazione di bevande alla spina tramite Bitcoin.

Il software da progettare e sviluppare sarà composto da una applicazione per i dispositivi terminali (i distributori automatici, client) e una per il server. Il progetto dovrà essere reso il più modulare possibile in modo da consentire modifiche alle funzionalità del software o all'hardware utilizzato senza che questo richieda difficili modifiche al programma.

4.1.1 Funzioni necessarie

Le specifiche verranno descritte separatamente per quanto riguarda la parte client e server.

Specifiche client

- Leggere l'eventuale contenuto del tag NFC per identificare l'utente.
- Permettere ai clienti di iniziare la procedura per la richiesta di pagamento premendo un pulsante.

- Ottenere il cambio usd-btc (o eur-btc) attuale e richiedere un indirizzo Bitcoin a cui l'utente dovrà effettuare il pagamento.
- Creare un QR code, contenente l'indirizzo e il totale da pagare, e mostrarlo a schermo.
- Informare l'utente appena il pagamento sarà stato ricevuto e sbloccare l'erogazione della bevanda.
- Se nei pressi del sistema embedded non venisse rilevata nessuna presenza per un tempo prolungato, lo schermo collegato al sistema si dovrà spegnere e dovrà essere annullata un'eventuale richiesta di pagamento in corso.
- Nel caso in cui la quantità residua di bevande sia poca il distributore deve informare il server e inviare una email per segnalare l'imminente esaurimento del distributore.
- Se il distributore finisse completamente le bevande deve chiedere al server se ce ne siano di non esauriti nelle vicinanze e, in caso di risposta affermativa, deve consigliare agli utenti di recarsi alle macchinette nelle vicinanze.

Specifiche server

- Controllare i punti attuali di un utente e se ha diritto ad un'eventuale bevuta gratis.
- Controllare il saldo di un indirizzo Bitcoin e informare il client appena il pagamento sarà stato ricevuto.
- Annullare un'eventuale richiesta di pagamento in corso se richiesto dal client o comunque dopo 10 minuti di attesa.
- Tenere traccia dei distributori senza più bevande disponibili.

4.1.2 Ambiente di utilizzo

Il software lato client dovrà funzionare su un Raspberry Pi con una qualunque distribuzione Linux.

Invece il software lato server dovrà poter essere eseguito su un qualunque Sistema Operativo Linux, OSX o Windows.

4.1.3 Interfaccia utente

L'interfaccia utente dovrà essere molto minimale e consistere solo di messaggi testuali per informare gli utilizzatori. L'interazione deve avvenire con un singolo bottone avente la funzione di notificare il sistema dell'inizio del processo di acquisto.

4.2 Casi d'uso

Nella figura 4.1 sono rappresentati i casi d'uso principali dell'intero sistema.

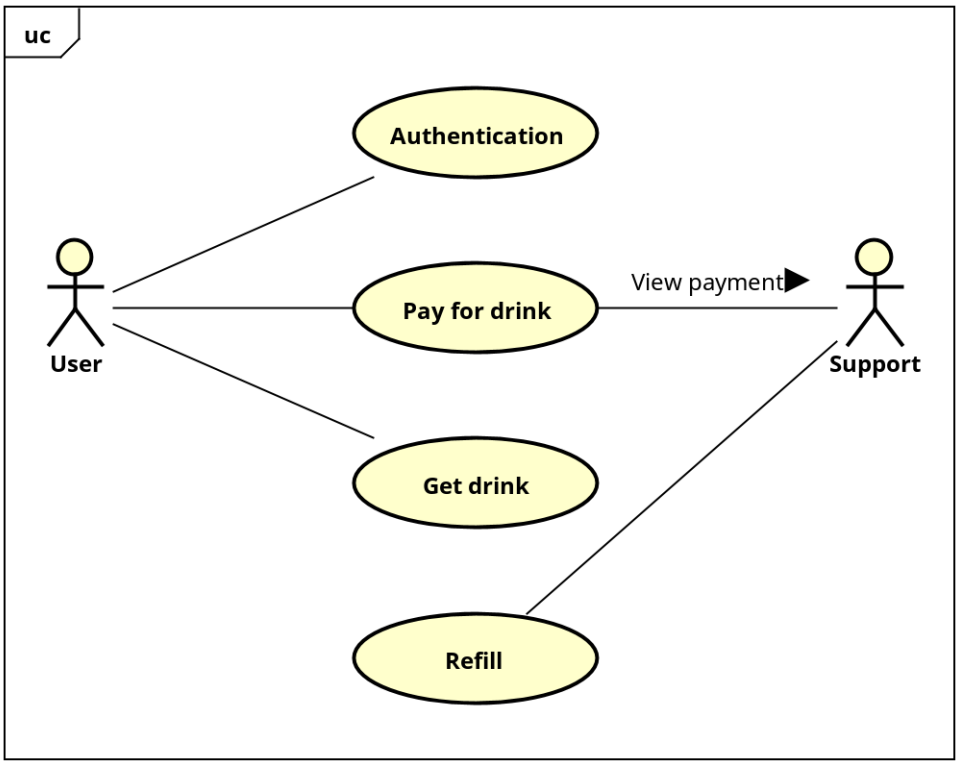

Figura 4.1: Use cases di Bit2Draft

4.3 Funzionalità del sistema nel dettaglio

In questa parte viene descritta in maniera dettagliata le funzionalità del sistema e i requisiti funzionali.

4.3.1 Rete di Comunicazione Client Server

I client e il server si ipotizzano essere collegati attraverso una rete dedicata. Qualora la si volesse condividere con altri dispositivi, si consiglia l'utilizzo di VLAN per mantenere domini di collisione separati.

Lo scambio di messaggi dovrà avvenire in modo da garantire la consegna e l'integrità dei messaggi stessi. Inoltre per evitare che la comunicazione possa venire compromessa i messaggi dovranno essere crittografati.

4.3.2 Gestione e salvataggio dei dati

Quando un utente si registra gli verrà consegnato un tag NFC contenente un id univoco. Quando un utente passerà il proprio tag su un distributore, ed effettuerà un pagamento, il server dovrà tenere traccia dei punti accumulati da quell'id perché, nel caso ne raggiungesse un determinato numero, avrà diritto ad una bevuta gratis. Per questo motivo il server deve mantenere una lista degli id degli utenti con il relativo saldo punti. Per lo scopo si richiede l'utilizzo un DBMS in modo da preservare le informazioni anche dopo un eventuale spegnimento del server.

4.3.3 Ricezione e invio dati

Lato Client

All'accensione delle macchinette esse comunicheranno al server la loro zona di competenza e la loro quantità di bevande disponibile. Il software una volta letto l'id univoco salvato sui tag NFC deve mantenerlo in memoria fino alla conclusione del processo di pagamento perché in seguito all'erogazione della bevanda, per fare in modo che i suoi punti vengano aumentati, dovrà comunicare al server l'id della persona che ha completato con successo il processo.

Lato Server

Il server dovrà rimanere sempre in ascolto per poter ricevere gli eventuali messaggi che verranno inviati dai client. All'arrivo di un pacchetto verrà controllato il contenuto per capire il tipo della richiesta, che potrebbe essere:

- Una nuova richiesta di pagamento, quindi sarà necessario controllare un determinato indirizzo Bitcoin fino all'avvento del pagamento.
- Viene informato il server che non è più necessario controllare un indirizzo perché l'utente ha annullato la richiesta.
- Bisogna controllare sul database i punti attuali di un determinato utente e risponde al client se essi raggiungono il numero minimo per ottenere una bevuta gratis.
- Viene richiesto l'aumento dei punti di un determinato utente.
- Un client notifica al server la propria zona di competenza e se ha ancora bevande a disposizione
- Viene richiesto al server di verificare se in una determinata zona ci sono ancora distributori non esauriti.

4.3.4 Zone di competenza

Quando un distributore finisce completamente la bibita alla spina a propria disposizione deve smettere di accettare nuove richieste dai clienti e deve chiedere al server se ci sono distributori non esauriti nella sua stessa zona di competenza.

Ad ogni client deve venire assegnata una zona di competenza in modo da poter scartare tutti i distributori che non si trovano nelle vicinanze.

4.4 Piano di collaudo

Tenendo conto delle funzionalità necessarie al sistema esposte nella sezione 4.1.1, il piano di collaudo prevederà la verifica dei seguenti scenari:

- Quando un cliente appoggia il proprio tag NFC sul lettore, il sistema deve essere in grado di identificarlo correttamente.

- Dopo che il cliente ha espresso la propria volontà di effettuare un acquisto, il distributore deve generare correttamente un QR code con le informazioni per effettuare il pagamento.
- I distributori devono poter comunicare correttamente con il server ad esempio informandolo sul proprio stato attuale.
- L'erogazione della bevanda non deve essere possibile fino a quando il pagamento non verrà confermato dal server. A pagamento effettuato l'erogazione deve avvenire fino a quando la quantità prefissata non verrà raggiunta.

Capitolo 5

Panoramica sulle tecnologie utilizzate

5.1 Raspberry Pi

Il Raspberry Pi, realizzato in UK dalla Raspberry Pi Foundation, è un single-board computer delle dimensioni di una carta di credito dal prezzo contenuto (circa 35$). Nel corso degli anni sono state rilasciate diverse revisioni fino ad arrivare all'attuale versione 2.

La versione 1 del Raspberry Pi è basata sul system on a chip (SoC) Broadcom BCM2835 che include un processore ARM1176JZF-S 700 MHz, una GPU VideoCore IV e 256 MB di RAM (successivamente aumentati a 512 MB nei modelli B e B+).

A Febbraio 2015 è stata rilasciata la seconda generazine chiamata Rasperry Pi 2. A differenza della prima generazione il SoC utilizzato è un Broadcom BCM2836 con una CPU quad-core ARM Cortex-A7, una GPU VideoCore IV dual-core e 1 GB di RAM.

Raspberry Pi usa principalmente sistemi operativi (OS) basati su kernel Linux e l'OS ufficialmente supportato è Raspbian (basato su Debian). Inoltre il Raspberry Pi 2 ha aggiunto il supporto a Windows 10 IoT Core.

Figura 5.1: Raspberry Pi modello B+

5.2 NFC

Near field communication (NFC) è un set di protocolli che abilita i dispositivi elettronici a stabilire una comunicazione radio (RF) bidirezionale a corto raggio toccando i dispositivi tra di loro o avvicinandoli tipicamente a 10cm o meno.

Figura 5.2: Logo NFC

I dispositivi NFC possono operare in modalità:

- Card Emulation: per permettere ai dispositivi come gli smartphone di emulare una smart card e quindi poter effettuare transazioni come i pagamenti.

- Reader/Writer: per poter leggere o scrivere le informazioni presenti sui tag NFC passivi.

- Peer-to-Peer: per fare in modo che due dispositivi NFC possano comunicare tra di loro scambiandosi informazioni.

5.3 QR code

Quick Response Code (abbreviato QR code) è un codice a barre bidimensionale sviluppato inizialmente da un'azienda automobilistica Giapponese per tenere traccia dei veicoli durante la fabbricazione. I codici a barre sono un insieme di elementi grafici, leggibili da lettori ottici, che contengono informazioni riguardanti l'oggetto a cui sono legati. I QR code salvano le informazioni utilizzando quattro encoding standard: numerico, alfanumerico, binario e kanji.

Il QR code è formato da punti neri posizionati su uno schema di forma quadrata su sfondo bianco. Diversi dispositivi che dispongono di una fotocamera possono leggere i QR code e interpretare il contenuto.

È possibile memorizzare fino a 4.296 caratteri alfanumerici o 7.089 caratteri numerici o 2.953 byte.

Nei casi in cui una parte del QR code sia illeggibile è comunque possibile tentare di leggere il contenuto perché viene utilizzato il codice di Reed-Solomon che permette di rilevare e correggere gli errori e i dati persi. Per questo motivo a volte i QR code sono volutamente in parte coperti da scritte commerciali senza che queste ne compromettano il corretto funzionamento.

5.4 Sensori e attuatori

5.4.1 Sensore ad ultrasuoni

I sensori a ultrasuoni rilevano e convertono le onde ultrasoniche in segnali elettrici o viceversa. Alcuni sensori possono sia emettere che rilevare gli ultrasuoni.

I sensori attivi, come avviene per i sonar, generano onde sonore ad alta frequenza e rilevano l'eco di ritorno generato dalla presenza di un oggetto nella traiettoria. Misurando l'intervallo di tempo trascorso tra l'invio del

segnale e la reicezione dell'eco è possibile determinare la distanza a cui si trova l'oggetto.

5.4.2 Turbine flow meter

Il flow measurement serve a quantificare il flusso di liquido in transito.

Un tipo di flow measurement è il flow meter a turbina che traduce l'azione meccanica della rotazione della turbina, provocata dal passaggio di un flusso di liquido, dando una quantificazione numerica della quantità di liquido in transito. I vantaggi utilizzando le turbine sono che, durante il passaggio del liquido, si ha una ridotta ostruzione e una bassa perdita di pressione.

5.5 Scambio di messaggi tra i dispositivi

5.5.1 Ethernet e Wi-Fi

Ethernet è una famiglia di tecnologie di rete per le LAN (local area networks) e le MAN (metropolitan area networks). È stato introdotto commercialmente nel 1980 e successivamente revisionato diverse volte per aumentarne il bit rate supportato e permettere una maggiore distanza massima dei cavi.

Lo stream delle informazioni prima di essere inviato su Ethernet viene diviso in parti chiamate frame. Ogni frame contiene un checksum per

Figura 5.3: Logo Wi-Fi

controllare la correttezza dei dati inviati e le informazioni sull'indirizzo del mittente e del destinatario.

L'alternativa più utilizzata per le reti LAN è il wireless IEEE 802.11 (Wi-Fi). Il Wi-Fi è una tecnologia che permette ai dispositivi elettronici di avere connettività creando una rete locale wireless (WLAN), principalmente utilizzando le frequenze a 2.4 gigahertz e 5 gigahertz. Il Wi-Fi permette di realizzare una rete LAN in modo economico e senza la necessità di dover utilizzare i cavi per i collegamenti. Utilizzando un access point è possibile fornire l'accesso ad una rete, come Internet, a tutti i dispositivi collegati in Wi-Fi. La copertura di un access point dipende sia dal numero di ostacoli presenti nell'ambiente che dal numero e qualità delle antenne utilizzate.

La sicurezza del Wi-Fi può essere inferiore rispetto all'Ethernet perché un attaccante non necessita di connettersi fisicamente all'access point. Le connessioni che utilizzano TLS sono sicure grazie al protocollo stesso, ma tutte le informazioni non criptate potrebbero essere lette da chiunque nel raggio della rete Wi-Fi. Per consentire lo scambio di dati in modo sicuro, durante gli anni sono state adottate diverse misure di sicurezza fino alla più recente WPA2.

5.5.2 TCP

Il Transmission Control Protocol (TCP) è uno dei principali protocolli di rete di Internet ed è al livello 4 (livello di trasporto) del modello OSI.

TCP offre una connessione affidabile tra gli host, più nel dettaglio garantisce:

- Affidabilità: grazie al sistema di acknowledgment viene garantito che tutti i segmenti vengano consegnati.
- Controllo di congestione: osservando le perdite e i ritardi nei sistemi terminali TCP riesce a dedurre se la rete è congestionata e di conseguenza limitare la velocità di trasferimento
- Controllo di flusso: viene evitato che il buffer del destinatario si riempi limitando la velocità se necessario
- Controllo sugli errori: utilizzando un campo checksum nei segmenti si controlla se i pacchetti ricevuti siano integri o corrotti

- Ordine: i segmenti, quando arrivano, sono già ordinati.

Per stabilire una connessione TCP utilizza il three-way handshake. Inizialmente il server deve eseguire un'apertura passiva: aprire e rimanere in ascolto su una determinata porta. Successivamente può essere eseguito il three-way handshake:

- SYN: Un'apertura attiva viene eseguita dal client inviando SYN al server. Il client sceglie in modo random il valore del numero di sequenza nel segmento.
- SYN-ACK: Il server risponde con un SYN-ACK. Il numero di acknowledgment è impostato al numero di sequenza ricevuto sommato uno, e il server sceglie a sua volta un numero di sequenza random.
- ACK: Il client invia un ACK al server. Il numero di sequenza è impostato al numero di acknowledgement ricevuto, e il campo acknowledgement viene impostato al numero di sequenza ricevuto sommato uno.

Alla fine del three-way handshake il client e il server instaurano con successo una connessione full-duplex.

TCP è utilizzato da molte applicazione come il World Wide Web (WWW), il Secure Shell (SSH) e da diverse applicazioni per lo scambio peer-to-peer. Questo è dovuto al fatto che TCP è progettato per avere un trasferimento accurato, anche se questo dovesse comportare un rallentamento nei tempi di trasmissione. Per questo motivo le applicazioni che sono influenzate fortemente dal tempo di risposta, generalmente, utilizzano User Datagram Protocol (UDP) come per esempio le applicazioni di Voice over IP (VOIP) o i videogiochi online.

5.5.3 Transport Layer Security

Il Transport Layer Security (TLS) è un protocollo di criptazione, inizializzato al livello 5 (livello di sessione) e operante al livello 6 (livello di presentazione), creato allo scopo di fornire sicurezza alle comunicazioni che avvengono in una rete di computer (come Internet) [16]. TLS è il successore del Secure Sockets Layer (SSL), anche se comunemente si continua a chiamare TLS

con il nome di SSL. L'obiettivo di TLS è quello di garantire la privacy e l'integrità dei dati in una comunicazione.

Per garantire l'integrità dei dati vengono utilizzate funzioni di hash come per esempio HMAC-MD5 e HMAC-SHA1.

Quando si utilizza TLS per instaurare una connessione si susseguono i seguenti passaggi:

- Il client e il server negoziano l'algoritmo da utilizzare. Esso può essere sia a chiave pubblica che a chiave precondivisa.
- Avviene lo scambio delle chiavi e dei certificati da entrambe le parti (in base all'algoritmo di cifratura scelto alcuni passaggi potrebbero non essere richiesti).
- Da questo punto in poi tutti i futuri messaggi saranno autenticati e criptati.

Capitolo 6

Progettazione

In questo capitolo viene descritta in maniera maggiormente dettagliata l'architettura del progetto e le scelte che hanno portato alla sua realizzazione.

6.1 Progettazione generale del sistema

Il progetto è stato diviso in due macro parti e in diverse sottoparti:

- Una parte che funge da client in esecuzione su un sistema embedded, suddiviso in:
  - Lettura tag NFC
  - Rilevamento presenza con il sensore di prossimità
  - Generatore di indirizzi Bitcoin e di QR code
  - Il programma principale che gestisce le varie parti, il flow blocker e il flow meter
- Una parte che funge da server in esecuzione su un computer che controlla i punti degli utenti e verifica i pagamenti

Per la realizzazione del progetto si è optato di separare la funzione di verifica dei pagamenti (facendola gestire al server) dal resto dell'applicazione. Dato che i distributori sono lasciati, di solito, incustoditi è relativamente di maggiore facilità la loro possibile compromissione. Per questo avere la verifica dei pagamenti sul server permette di aumentare la sicurezza, in quanto

è solo il server, non a contatto con i clienti, che controlla se un dato pagamento sia veramente avvenuto. La scelta di avere un server centralizzato e tanti client è servita anche per poter mantenere la lista degli utenti registrati con il relativo saldo punti. Inoltre grazie alla centralizzazione della verifica dei pagamenti, utilizzando un DBMS, sarà di maggiore facilità il collezionamento e la gestione dei dati a scopo di fatturazione e statistica.

6.2 Architettura complessiva

In figura 6.1 è presente il diagramma delle classi, relativo alle macroparti in cui il progetto verrà diviso, e come esse interagiscono.

Nel sistema diverse interazioni richiedono di essere realtime come per esempio i controlli per i pagamenti che vengono richiesti ad intervalli regolari di tempo. Comunque il progetto non è hard realtime, infatti anche se le deadline non venissero rispettate la stabilità del sistema non deve venire compromessa.

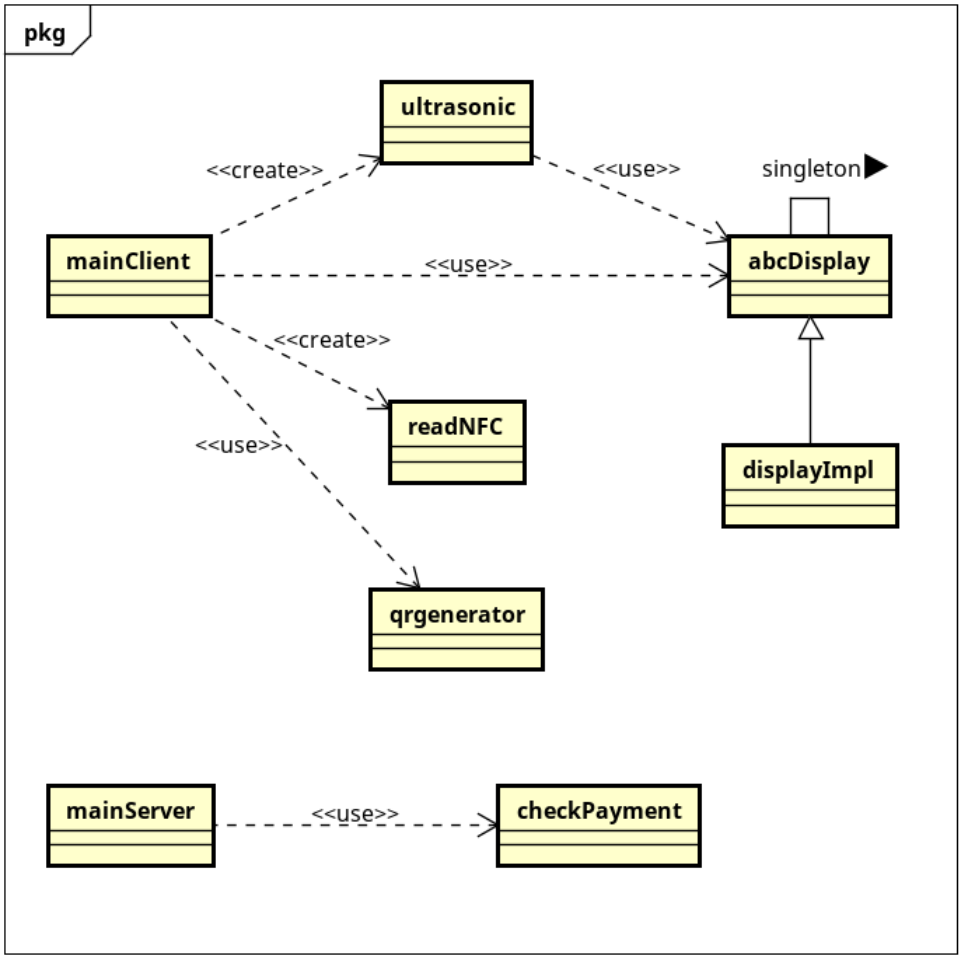

Figura 6.1: Class diagram generale del progetto

6.2.1 Funzioni concorrenti

Per come è stato pensato il progetto, il client avrà diversi compiti da svolgere e, nella maggior parte dei casi, essi dovranno essere eseguiti concorrentemente. Simultaneamente alla funzioni di generazione del QR code, alla gestione del flow meter e flow blocker il sistema deve essere in grado di gestire un sensore di prossimità per verificare la presenza dei clienti e inoltre deve gestire il lettore NFC per leggere i tag degli utenti.

Il server dovrà poter gestire contemporaneamente diverse richieste, come il controllo simultaneo di diversi indirizzi per verificare l'arrivo dei pagamenti. Inoltre dovrà essere in grado di accettare in qualunque momento i messaggi che i client potrebbero inviare.

6.2.2 Generazione dati per il pagamento

Per poter permettere ai clienti di effettuare un pagamento deve essere generato un QR code contenente l'indirizzo Bitcoin, a cui bisogna effettuare il pagamento, e la somma da pagare. Per comodità l'importo è salvato in Dollari (volendo si possono scegliere gli Euro o qualunque altra valuta) e, solo prima della generazione del QR code, l'importo viene convertito in Bitcoin.

6.2.3 Interfacciamento con l'utente

L'utente ha a disposizione uno schermo collegato alla breadboard per visualizzare lo stato del distributore e le informazioni utili ad effettuare il pagamento.

6.2.4 Salvataggio dei dati

Il server ha il compito di tenere traccia dei punti attuali degli utenti registrati e per lo scopo in fase di analisi si era ipotizzato l'utilizzo di un DBMS. Considerando però che nella progettazione del prototipo sarà presente un solo distributore con un numero esiguo di clienti registrati si è scelto di utilizzare un file csv per il salvataggio dei dati dei clienti.

Quando viene richiesta una modifica al file csv, verrà generata una copia sulla quale verrà apportata la modifica. Successivamente la copia andrà a sovrascrivere il file csv originario. In questo modo si evitano possibili corruzioni del file nel caso in cui avvengano errori durante la scrittura.

Ogni volta che un nuovo utente utilizzerà per la prima volta il proprio tag NFC per effettuare un'acquisto, nel file csv verrà effettuato un append con il nuovo id e i suoi punti. Avendo assunto che il numero di clienti non sia elevato, la lista verrà lasciata non ordinata.

Se invece gli utenti iscritti dovessero aumentare di qualche ordine di grandezza, si potrebbe pensare di ordinare la lista in base all'id per poter effettuare una ricerca dicotomica e risparmiare tempo.

6.2.5 Controllo indirizzi Bitcoin

Per verificare se un determinato pagamento sia avvenuto si controlla se il saldo attualmente disponibile sull'indirizzo Bitcoin indicato è uguale o maggiore all'importo del pagamento. È sufficiente questo controllo perché viene generato un nuovo indirizzo a fronte di ogni richiesta di pagamento e quindi il saldo inizialmente sarà sempre nullo.

Per verificare il saldo attuale di un indirizzo verranno utilizzate le api del sito *Blockchain.info* [1].

Un'alternativa all'utilizzo di api di terze parti era rendere il server direttamente un nodo Bitcoin eseguendo il client Bitcoin core. Questa scelta però avrebbe comportato la necessità di avere in locale l'intero Blockchain (diversi Gigabyte) e l'utilizzo continuativo della rete dovuto all'esecuzione del nodo Bitcoin. Per questo motivo, almeno finché il numero di distributori rimane limitato, si è optato per l'utilizzo di api di terze parti.

6.2.6 Garanzia di affidabilità

Come esposto nella sezione 3.2 dopo che gli utenti effettuano un pagamento bisogna garantirgli l'erogazione della bevanda acquistata. Se l'erogazione non fosse possibile a causa di problemi software o hardware la somma versata dal cliente deve essergli restituita.

Una volta che il cliente effettua il pagamento, il distributore viene notificato e l'erogazione può avvenire. Quando il distributore termina l'erogazione invia un messaggio per informare il server. Nel caso in cui il server non riceva più notizie dal client o il distributore rilevi l'impossibilità di erogare correttamente la bevanda, il server provvederà a restituire i Bitcoin pagati dal cliente.

6.3 Lettura tag NFC

Questa parte si occupa dell'utilizzo di un NFC reader consentendo di leggere il contenuto dei tag NFC degli utenti.

[1] *https://blockchain.info/api*

6.3.1 Comportamento

La struttura che meglio rappresenta questa parte è la Finished State Machine (FSM), infatti durante l'esecuzione si susseguono sempre una serie finita di operazioni.

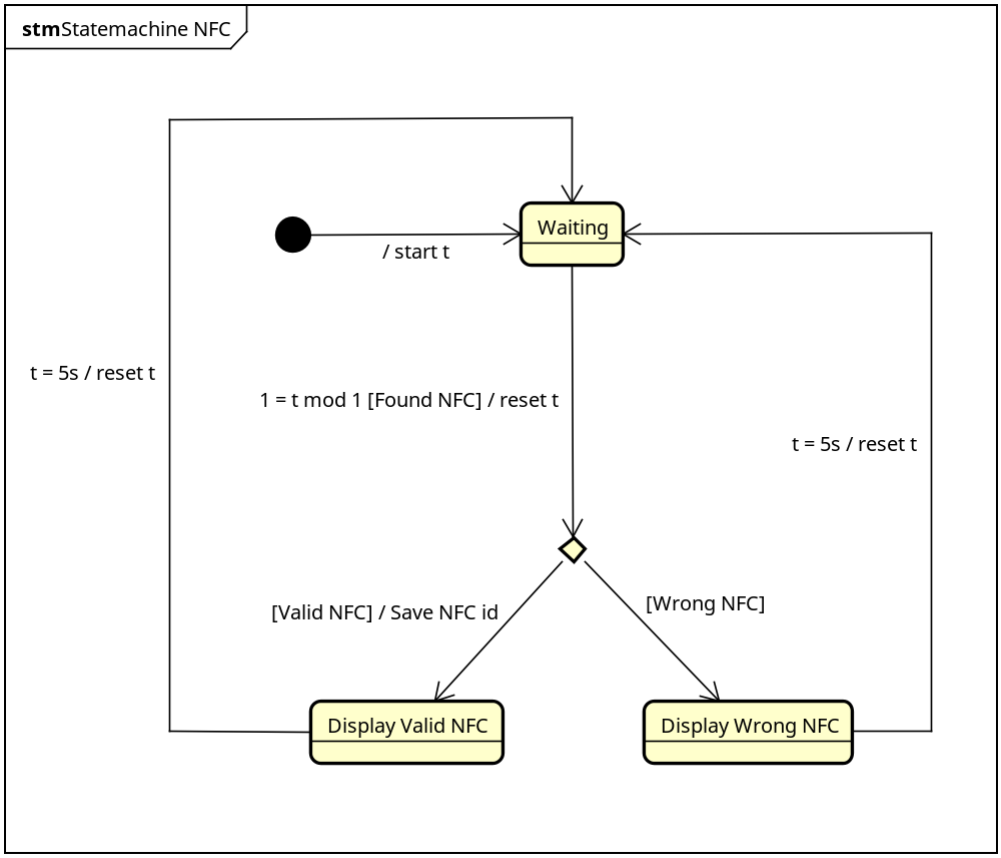

Figura 6.2: Finished State Machine riguardante la parte di lettura tag NFC

Com'è possibile notare in figura 6.2 inizialmente viene inizializzato un contatore "t" per mantenere il tempo e rimane nello stato "Waiting" fino a quando il timer non raggiunge un secondo (o successivi multipli) ed è presente un tag NFC da leggere.

Successivamente viene controllato il contenuto del tag e, nel caso risultasse valido, il suo valore verrà salvato in memoria. In seguito verrà visua-

lizzato a schermo per 5 secondi il risultato della lettura del tag e tornerà nello stato di attesa.

6.3.2 Diagramma di sequenza

Per la rappresentazione del tipico utilizzo della parte relativa all'NFC viene utilizzato un sequence diagram (digramma di sequenza).

Come è rappresentato in figura 6.3 è l'utente ad iniziare il processo appoggiando il proprio tag NFC sul lettore. Successivamente il sistema legge il contenuto del tag e verifica se l'id rispetta determinati requisiti (deve iniziare con dei caratteri prestabiliti, avere una certa lunghezza ecc...). Nel caso i requisiti siano soddisfatti l'id viene salvato in memoria per fare in modo che i suoi punti possano venire aumentati alla fine del processo di pagamento. Nel diagramma di sequenza si è omesso lo schermo per focalizzare l'attenzione sull'NFC, ma bisogna comunque tenere presente che i messaggi di risposta vengono letti dall'utente tramite il display collegato al sistema.

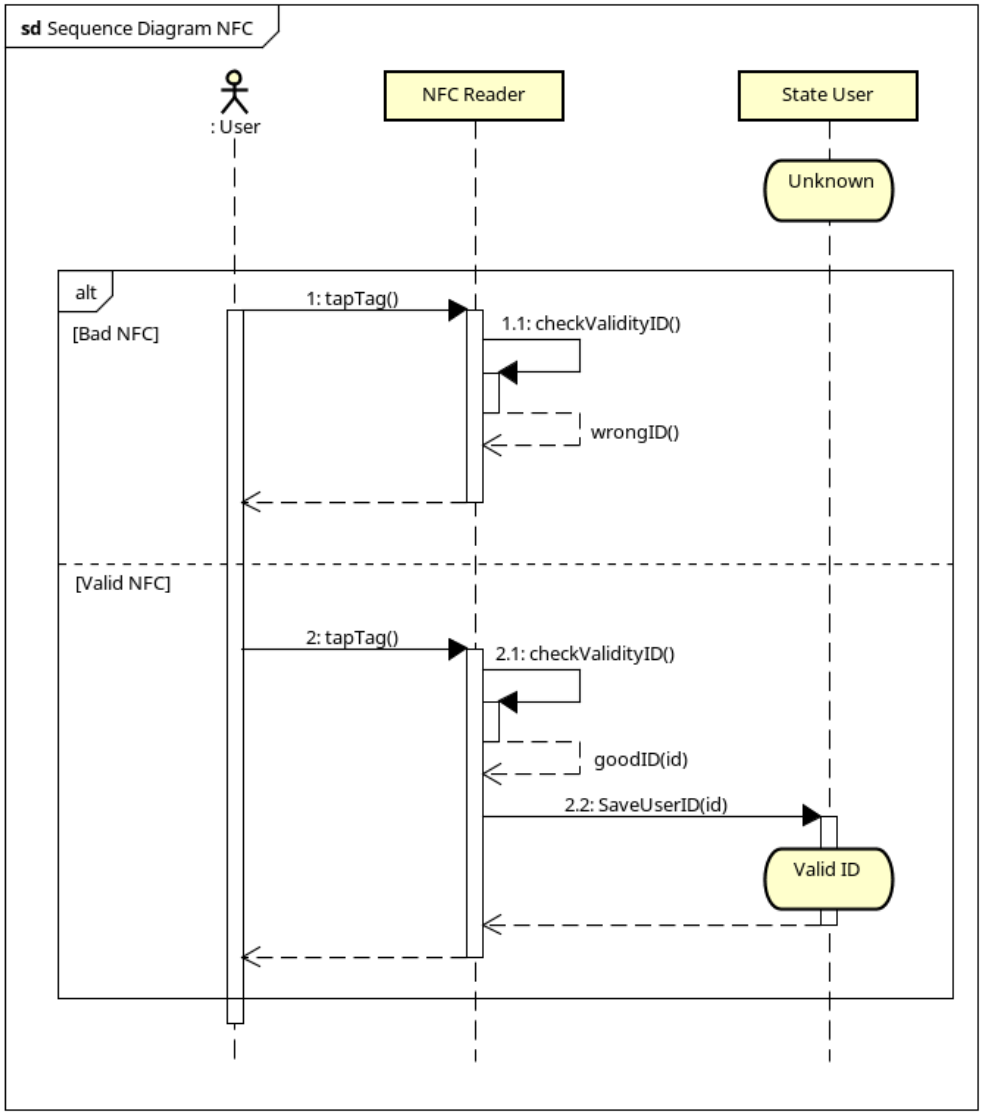

Figura 6.3: Sequence Diagram relativo ad un utilizzo tipico dell'NFC

6.4 Controllo di presenza

Questa parte si occupa di controllare la presenza di eventuali clienti. Utilizzando un sensore di prossimità periodicamente viene calcolata la distanza tra il sensore e l'ostacolo più vicino per valutare se una persona possa essere nelle vicinanze del distributore.

6.4.1 Comportamento

Per la rappresentazione dell'architettura è stato scelto di utilizzare una Finished State Machine.

In figura 6.4 lo stato "Alone" rappresenta la situazione in cui non ci siano persone nelle vicinanze del sensore e invece "Together" è quando un utente è nei pressi del distributore.

Per questo quando si è nello stato "Alone" per più di 30 secondi il display del distributore viene spento e notificato al server di cancellare l'eventuale richiesta di pagamento in corso.

La variabile "t" rappresenta il tempo mentre la variabile "d" è la distanza misurata dal sensore di prossimità. Per rendere il diagramma maggiormente leggibile si è deciso di omettere il reset di "t" che comunque avverrebbe dopo ogni passaggio di stato.

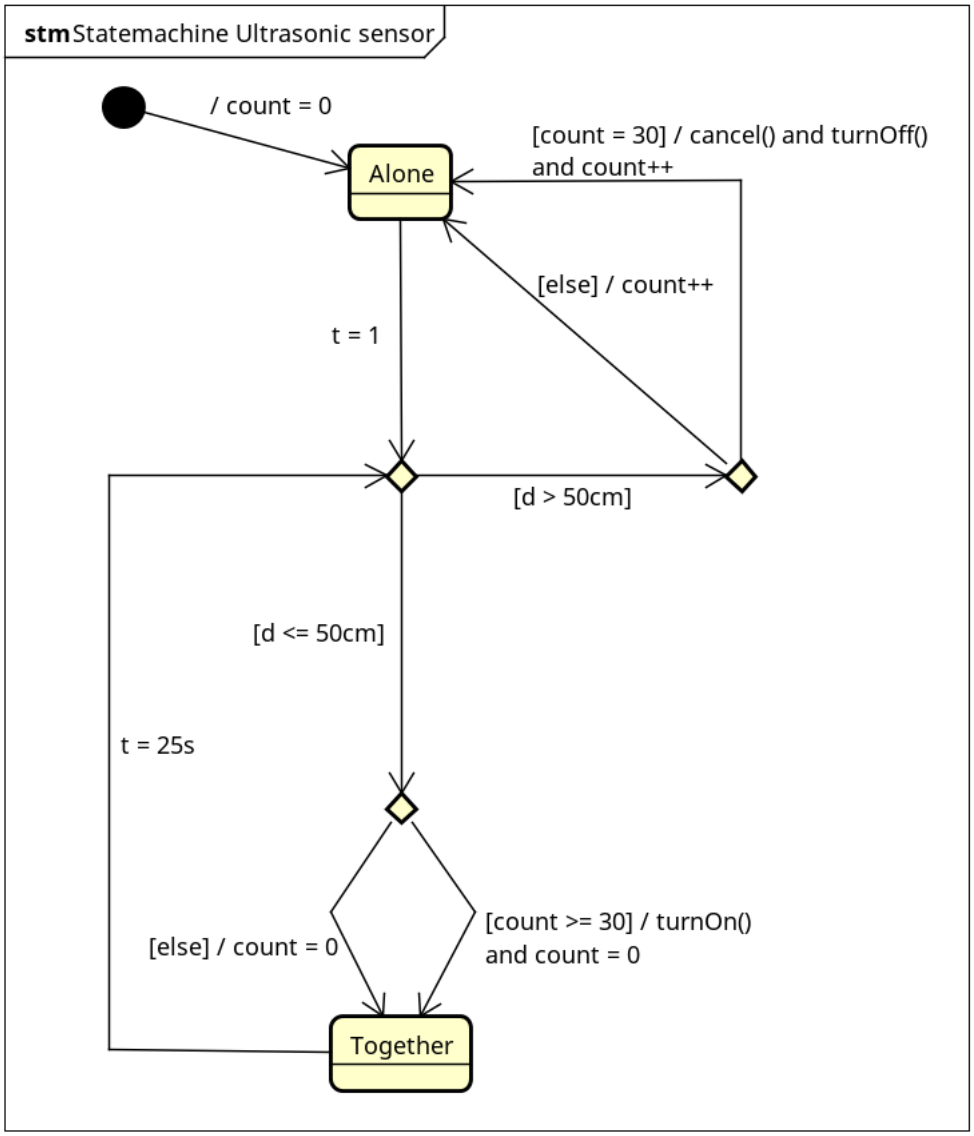

Figura 6.4: Macchina a stati finiti relativa al sensore di prossimità

6.4.2 Diagramma di sequenza

Per la rappresentazione dello scambio di messaggi relativo alla parte del sensore viene utilizzato un sequence diagram (digramma di sequenza).

Come mostrato in figura 6.5 il sensore di prossimità controlla, una volta al secondo, la distanza dell'oggetto più vicino al distributore per verificare la presenza di possibili clienti.

Se la distanza rilevata è sempre maggiore di 50 centimetri per un tempo di almeno 30 secondi allora il thread che gestisce il sensore ad ultrasuoni invia un messaggio al server per informarlo di annullare un eventuale richiesta di pagamento da questo distributore e spegne il display.

Successivamente quando viene rilevata la presenza di un potenziale cliente (distanza minore di 50 centimetri) viene riacceso il display mostrando un messaggio di benvenuto.

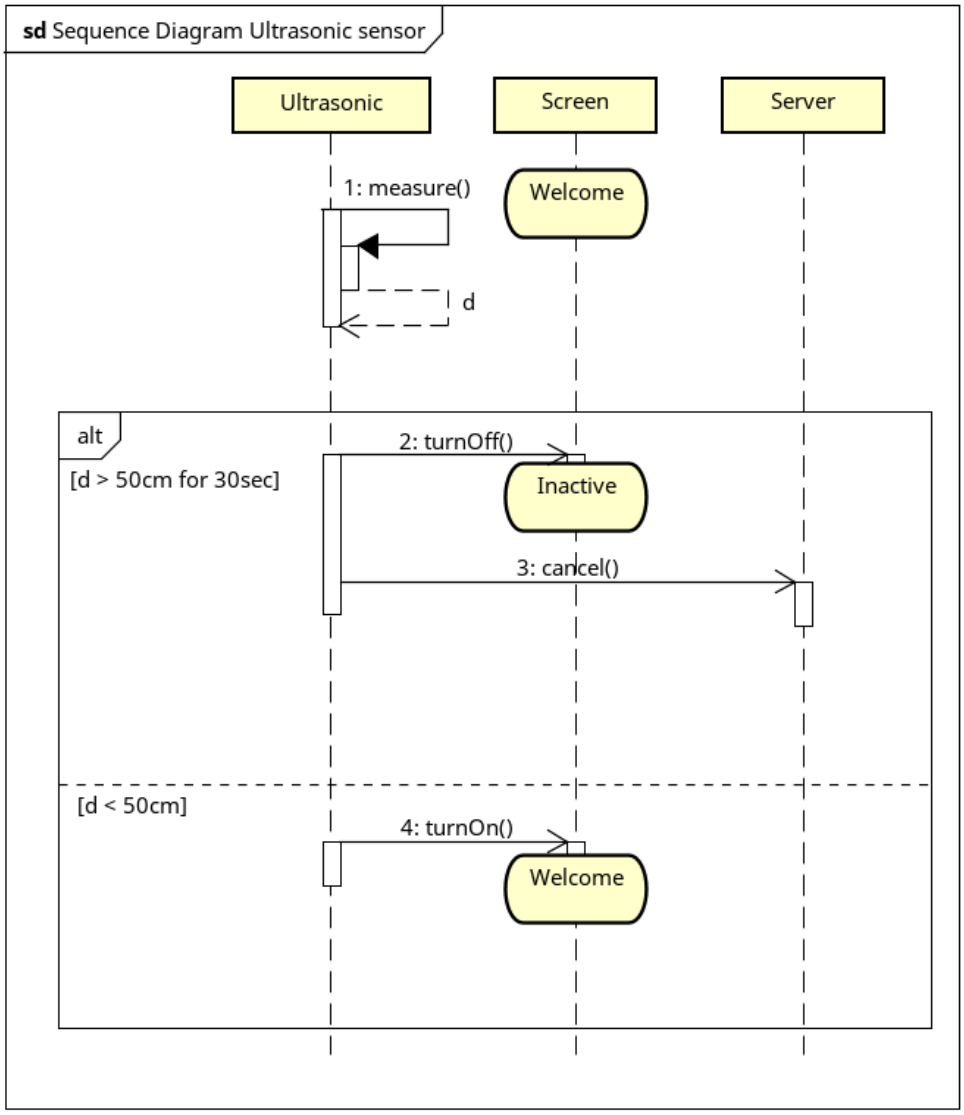

Figura 6.5: Sequence Diagram relativo al modulo del sensore di prossimità ad ultrasuoni

6.5 Progettazione server

6.5.1 Comportamento

Anche in questo caso verrà utilizzata una Finished State Machine per rappresentare il server.

Come si può notare in figura 6.6 all'avvio del server vengono creati due thread concorrenti. Nella realtà i thread che fanno pop e servono le richieste presenti sulla coda possono essere un numero maggiore. Nella macchina a stati ne è stato rappresentato uno solo perché gli altri sarebbero stati semplicemente una copia del primo.

La variabile "t" rappresenta il tempo mentre "count" è un contatore univoco per ogni elemento presente sulla coda.

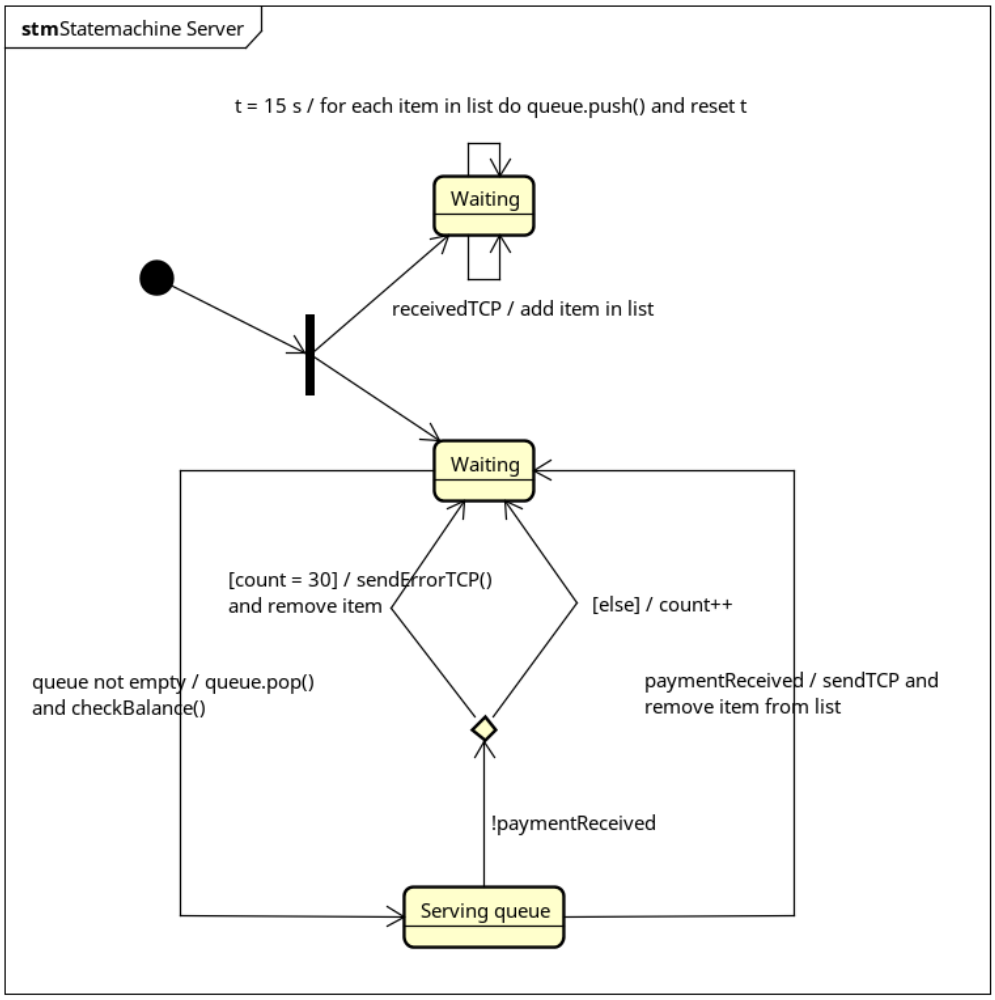

Figura 6.6: Macchina a stati finiti relativa al server

6.6 Progettazione client

6.6.1 Comportamento

Per la rappresentazione dell'architettura verrà utilizzata una Finished State Machine.

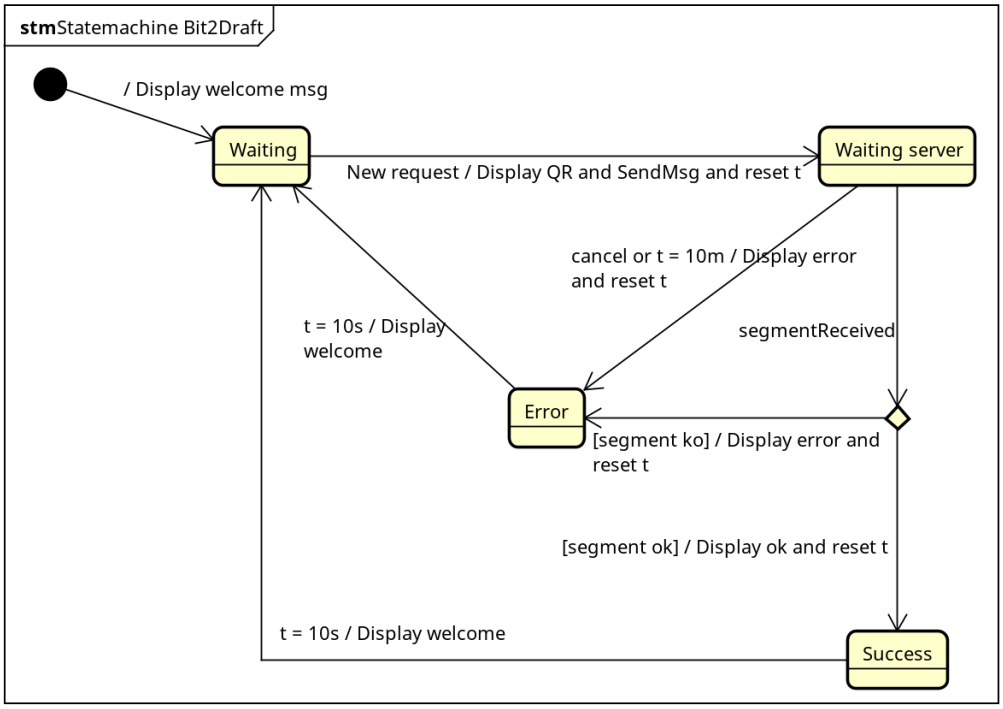

Figura 6.7: Macchina a stati finiti relativa al client

In figura 6.7 la "New request" è l'evento che indica l'intenzione di un nuovo acquisto ed è eseguito dal pulsante presente sulla breadboard. Dallo stato "Waiting server" l'azione "Cancel" può essere chiamata dal sensore ad ultrasuoni se la persona che aveva iniziato il pagamento si allontana dal distributore per diversi secondi. "DatagramReceived" è la risposta del server.

La variabile "t" rappresenta il tempo.

6.6.2 Diagramma di sequenza

Utilizzando un sequence diagram è possibile rappresentare l'interazione tra il client, il flow meter e il flow blocker.

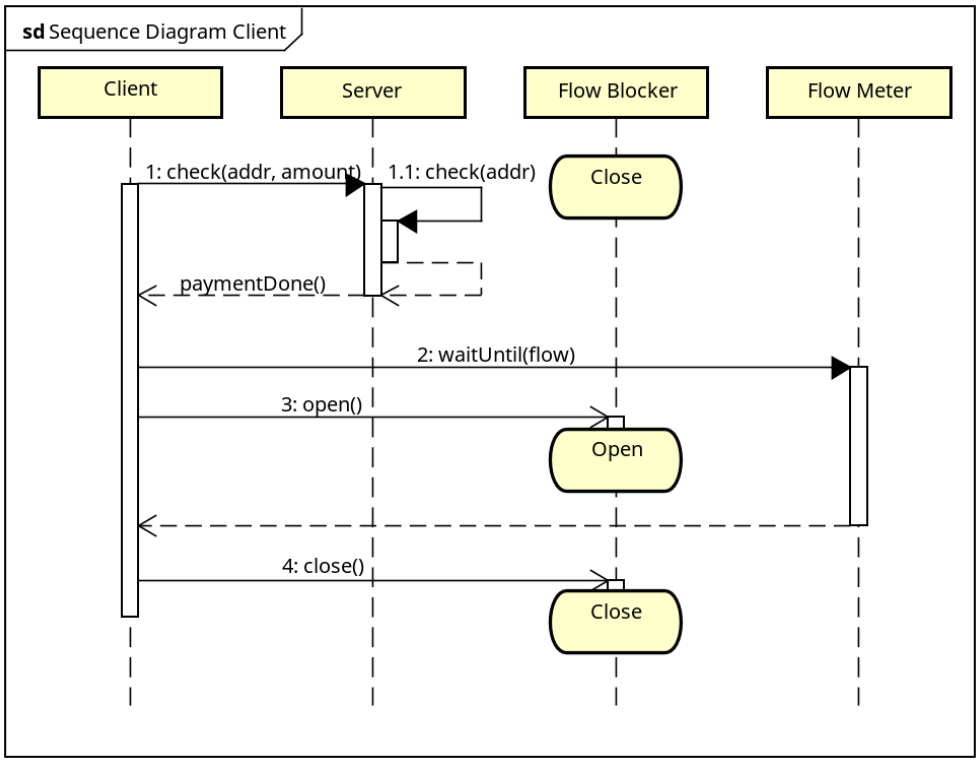

Figura 6.8: Diagramma di sequenza relativo all'interazione tra il client, il flow meter e il flow blocker

Come è rappresentato in figura 6.8, una volta che il client riceve la conferma dal server che in un determinato indirizzo Bitcoin è stato effettuato il pagamento richiesto, il client controllerà il flow meter in modo da venire a conoscenza di quando il cliente ha versato esattamente la quantità di bevanda che ha acquistato.

Il client apre il flow blocker, il quale rimarrà in questo stato fino a quando il flow meter misurerà la quantità prestabilita.

6.7 Tipica sessione di utilizzo

In una tipica sessione di utilizzo le operazioni che vengono svolte sono:

- Accensione del server, il quale si dividerà in:
  - Una parte che rimarrà in ascolto di eventuali messaggi provenienti dai client contenenti le informazioni relative alle richieste di pagamento e che ogni 15 secondi ripopola la coda delle richieste con quelle attualmente attive.
  - Una parte che rimarrà in attesa dell'arrivo di almeno un elemento sulla coda delle richieste per poterle soddisfare.
- Accensione dei client con la relativa inizializzazione della lettura dei tag NFC e del sensore di prossimità.
- Quando un utente si avvicina al distributore il sensore di prossimità rileverà una distanza ridotta e accenderà lo schermo della macchinetta mostrando un messaggio di benvenuto
- L'utente può utilizzare il proprio tag NFC per autenticarsi ed accumulare punti. In questo caso il client chiederà al server il saldo attuale dell'utente e se ha diritto ad una bevuta gratis. In alternativa l'utente può premere direttamente il pulsante presente sulla breadboard e continuare il processo senza autenticarsi.
- Il client richiede un nuovo indirizzo Bitcoin e il cambio usd-btc attuale. Successivamente crea un QR code contenente le informazioni utili all'utente per poter effettuare il pagamento.
- Il client notifica al server questa nuova richiesta di pagamento e il server controllerà circa ogni 15 secondi l'indirizzo Bitcoin per verificare se il pagamento fosse stato effettuato dall'utente.
- Quando il pagamento viene effettuato, il server invia un messaggio al client per notificarlo.
- Il client sblocca l'erogazione della bevanda fino a quando non viene versata la quantità prestabilita e, se l'utente si era inizialmente autenticato, i suoi punti verranno aumentati.

6.7.1 Diagramma di sequenza

Dalla figura 6.9 si può notare come il processo di pagamento inizi con l'intenzione dell'utente di fare un acquisto. Il client aggiorna lo stato dello schermo e controlla se l'utente si fosse in precedenza autenticato verificando lo "State User". In seguito il client richiede un nuovo indirizzo Bitcoin e il tasso di cambio usd-btc attuale. Nella realtà le due richieste sono separate ma per mantenere una maggiore leggibilità sono state rappresentate come un singolo messaggio. Utilizzando le informazioni ricevute verrà generato un QR code che verrà mostrato a schermo per permettere all'utente di effettuare il pagamento. Il client chiederà al server di controllare quel determinato indirizzo e, quando il server informerà il client dell'avvenuta ricezione del pagamento, verrà notificato l'utente che è possibile usufruire della bibita alla spina.

Il diagramma durante il processo di comunicazione tra client e server è stato semplificato, infatti il messaggio che il client invia al server per richiedere che un determinato indirizzo venga controllato è stato inglobato in "makePayment()".

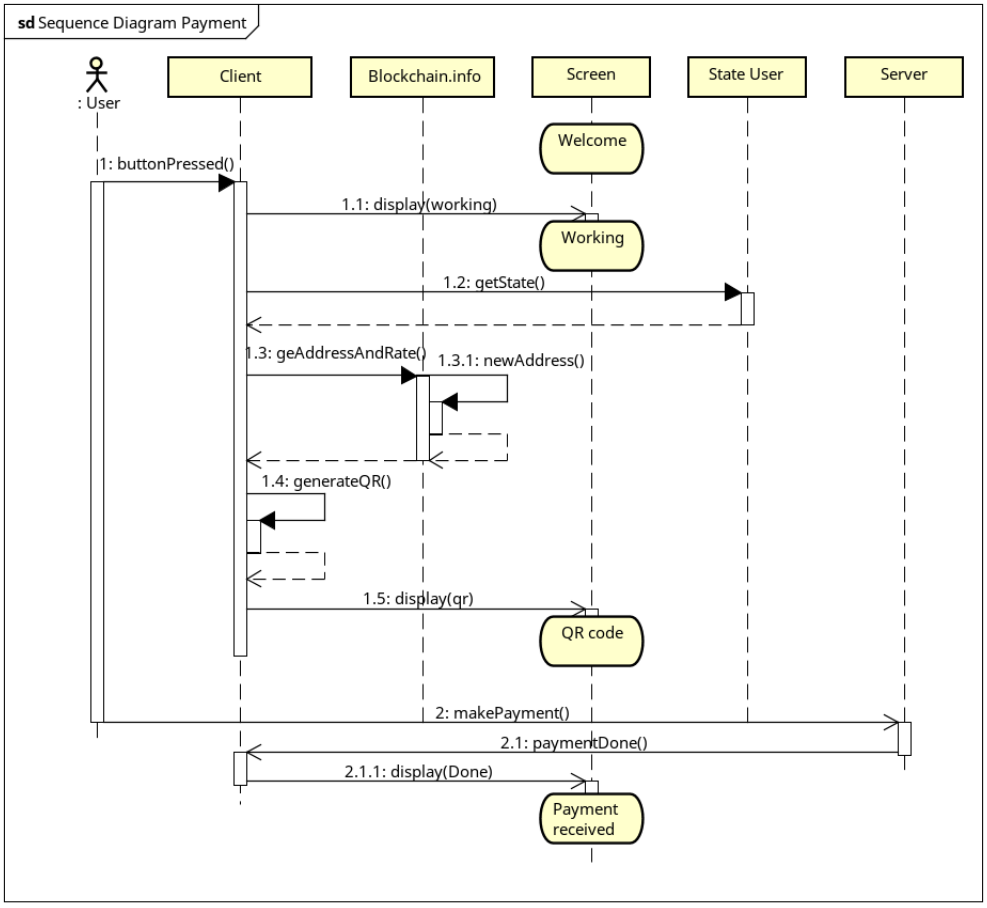

Figura 6.9: Sequence Diagram relativo ad un caso tipico di pagamento

Capitolo 7

Implementazione

In questo capitolo viene descritta l'implementazione del progetto e sono presenti le parti più significative del codice.

7.1 Prototipazione

Il progetto che verrà realizzato può essere definito un prototipo utilizzato per validare l'idea.

Come spiegato nella sezione 6.2.4 per il progetto reale bisogna utilizzare un DBMS ma per il prototipo si utilizzerà un file csv.

Lo schermo che si utilizzerà sarà il compatto Nokia 84x48. In produzione sarebbe meglio utilizzare uno schermo di dimensioni maggiori per poter permettere ai clienti una lettura più agevole delle informazioni e dei QR code.

Inoltre come flow meter sarebbe meglio utilizzarne un modello professionale con una elevata precisione.

7.2 Linguaggi di programmazione

Il linguaggio di programmazione scelto per la realizzazione del progetto è Python.

Essendo vincolati dall'utilizzo della libreria per il display Nokia si è dovuto utilizzare Python 2. Per mantenere consistenza tra la parte client e quella server si è utilizzato Python 2 per l'intero progetto.

7.3 Comunicazione Client Server

Per lo scambio di messaggi si è scelto di utilizzare TCP in quanto è necessario avere la garanzia che i segmenti vengano consegnati con successo. Inoltre l'applicazione non è suscettibile al ritardo di trasmissione tra i client e il server. Come scritto nella sezione 3.2 i segmenti TCP verranno crittografati utilizzando SSL per fare in modo che la comunicazione avvenga in modo sicuro.

I client conoscono l'IP e la porta del server per poterlo contattare, mentre il server risponde utilizzando l'IP e la porta specificate nei segmenti TCP che riceve.

Le comunicazioni tra i client e il server avvengono attraverso l'uso di Wi-Fi e/o cavi ethernet.

I client generano, in maniera random, un id univoco ad ogni richiesta di pagamento per permettere al server di gestirne un numero elevato contemporaneamente senza il rischio che si generino sovrapposizioni. È stato scelto di utilizzare un id univoco rispetto al semplice ip del distributore perché non sarebbero necessarie modifiche al codice del server nel caso in cui in un futuro si volessero ampliare i distributori per permettere erogazioni multiple di bevande.

7.4 Criticità

Durante il normale utilizzo del sistema possono avvenire diverse situazioni critiche. Le criticità che sono state gestite sono:

- Mancata connessione di rete: nel caso in cui i client non siano più in grado di contattare il server continueranno ad intervalli regolari a cercare di stabiliere una connessione. Nel frattempo verrà visualizzato un messaggio a schemo per segnalare il problema ai clienti.

- Errore nel controllo di un componente: nel caso in cui non funzioni correttamente un componente (come ad esempio il lettore NFC) il sistema segnalerà via email ai tecnici il problema e verrà visualizzato a schermo un messaggio di errore.

7.5 Implementazione della parte relativa al client

7.5.1 Organizzazione a task

Come è stato spiegato nella sezione 6.2.1 il client deve poter eseguire diverse operazione in modo concorrente. Per questo motivo utilizzando la libreria *threading* messa a disposizione da Python vengono creati i seguenti thread:

- Il main thread che rimane in attesa che l'utente prema il pulsante presente sulla breadboard per iniziare il processo di pagamento. In seguito genererà il QR code che verrà mostrato al cliente e rimarrà in attesa della conferma di pagamento da parte del server. Successivamente gestirà il flow blocker e il flow meter.
- Il thread che gestisce il lettore NFC. Rimane sempre pronto per leggere gli eventuali tag NFC tranne per il periodo in cui è già in esecuzione una richiesta di pagamento.
- Il thread relativo al sensore di prossimità che controlla continuamente la distanza. Il tempo tra i vari controlli varia in base allo stato attuale del client. Infatti, quando inizia una nuova richiesta di pagamento, il sensore smette di controllare la distanza per 25 secondi in modo da permettere all'utente di assentarsi anche per un breve periodo senza che la sua richiesta venga annullata.

```
from threading import Thread
from displayImplementation import Display
import ultrasonic
import readNFC

[...]

nfc = readNFC.CheckNfc(4, 22, 20, 18) # GPIO port used for the NFC reader

[...]

def main():
    t = Thread(target=ultrasonic.Ultrasonic(19, 26).start_measuring)
    t.daemon = True
    t.start() # run the ultrasonic check in a separated thread
    Display.instance().turn_on()
    t = Thread(target=nfc.start_reading)
    t.daemon = True
    t.start() # run the nfc thread in a separated thread
    t = Thread(target=wait_loop)
    t.daemon = True
    t.run() # run in the same main thread
```

7.5.2 Lettura della distanza

Per il calcolo della distanza viene utilizzato il sensore ad ultrasuoni HC-SR04. Per mitigare eventuali errori grossolani nella misurazione vengono effettuate sempre due misurazioni calcolando successivamente la media.

Il thread che si occupa del sensore di prossimità utilizza un secondo thread per inviare il segnale ad ultrasuoni. Grazie a questa scelta si evita che il thread principale del sensore vada in busy waiting attendendo l'invio e il ritorno del segnale ad ultrasuoni.

Inoltre si è utilizzata la funzione *wait_for_edge* che permette di fermare l'esecuzione del thread fino a quando non viene rilevata un'oscillazione, consumando una quantità irrisoria di cpu.

```
def send_input(self):
    while True:
        self.e.wait()
        self.e.clear()
        while True:
            with self.lock:
                if not self.blocked:
                    break # exit the last while and wait for the event
            GPIO.output(self.GPIO_TRIGGER, True)
            time.sleep(0.00001)
            GPIO.output(self.GPIO_TRIGGER, False)

def measure(self):
    with self.lock:
        self.blocked = True
    self.e.set()
    GPIO.wait_for_edge(self.GPIO_ECHO, GPIO.RISING)
    start = time.time()
    GPIO.wait_for_edge(self.GPIO_ECHO, GPIO.FALLING)
    stop = time.time()
    with self.lock:
        self.blocked = False
    # Time multiplied by the speed of sound, go and return (half)
    return ((stop-start) * 34300) / 2

def _measure_average(self):
    distance1 = self.measure()
    time.sleep(0.1)
    distance2 = self.measure()
    return (distance1 + distance2) / 2

def start_measuring(self):
    counter = 0
    t = threading.Thread(target=self.send_input)
    t.daemon = True
    t.start()
    try:
        while True:
            distance = self._measure_average()
            if distance > 50:
                counter += 1
                # In this way only one tcp segment is sent until the
                    count will be set to 0
                if counter == 30:
                    clientTCP.cancel()
                    Display.instance().turnOff()
                time.sleep(1)
            else:
                if counter >= 30:
                    Display.instance().turnOn()
                counter = 0
                time.sleep(25)
```

7.5.3 Lettura tag NFC

Il lettore NFC utilizzato per il progetto è un PN532 e i tag NFC che si ipotizzano essere utilizzati per l'autenticazione sono i Mifare Classic. La loro capienza per i dati è di 16 byte ma, per scelta progettuale, vengono utilizzati solo i primi 8 byte per salvare l'id dell'utente.

Per controllare che l'id sia valido viene verificato che il primo byte corrisponda esattamente a *b1*.

```
def start_reading(self):
    pn532 = PN532.PN532(cs=self.CS, sclk=self.SCLK, mosi=self.MOSI,
        miso=self.MISO)
    pn532.begin()
    ic, ver, rev, support = pn532.get_firmware_version()
    pn532.SAM_configuration()
    while True:
        time.sleep(1)
        self.e.wait()
        uid = pn532.read_passive_target()
        if uid is None:
            continue
        if not pn532.mifare_classic_authenticate_block(uid, 4,
                PN532.MIFARE_CMD_AUTH_B, [0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
                0xFF]):
            continue
        data = pn532.mifare_classic_read_block(4)
        if data is None:
            continue
        if binascii.hexlify(data[:1]) == b'b1':
            Display.instance().show_image(Image.open("images/ReadNFC.png"))
            with self.lock:
                self._user = binascii.hexlify(data[:8])
        else:
            Display.instance().show_image(Image
                                          .open("images/WrongNFC.png"))
        time.sleep(5)
```

7.5.4 Generazione QR code

Per la generazione del QR code si utilizza la libreria *qrcode*.

Viene usata la minima correzione possibile sul QR code per poter generare un'immagine che possa essere visualizzata sullo schermo Nokia 84x48.

Prima della generazione è necessario richiedere un nuovo indirizzo Bitcoin e il tasso di cambio USD-BTC corrente.

Sia per la generazione di indirizzi Bitcoin, che per ottenere il cambio Dollari-Bitcoin, vengono utilizzate le api messe a disposizione dal sito *Blockchain.info*. Le richieste a Blockchain.info avvengono in modo sincrono perché il main thread nel frattempo non deve compiere altre operazioni. Vengono effettuati tre tentativi di connessione prima che venga segnalata la presenza di un errore.

```
1  import qrcode
2  import requests
3  import time
4  from PIL import Image
5  import Adafruit_Nokia_LCD as LCD
6
7
8  class _QrCodeGenerator(object):
9      def __init__(self, username, password):
10         self.username = username
11         self.password = password
12
13     def generate(self, address, total):
14         qr = qrcode.QRCode(
15             version=5,
16             error_correction=qrcode.constants.ERROR_CORRECT_L,
17             box_size=1,
18             border=4,
19             )
20         qr.add_data('bitcoin:' + address + '?amount=' + str(total) +
               '&label=PaymentSendNFC')
21         qr.make(fit=True)
22         img = qr.make_image()
23         resized_im = Image.new('RGB', (LCD.LCDWIDTH, LCD.LCDHEIGHT),
               "white")
24         resized_im.paste(img, (20, 1))
25         return resized_im
26
27     def toBTC(self, value, currency='USD'):
28         payload = {'currency': currency, 'value': value}
29         url = 'https://blockchain.info/tobtc'
30         btc_total = -1
31         try:
32             r = requests.get(url, params=payload, verify=True, timeout=10)
33             if r.status_code == 200:
```

```
34                btc_total = float(r.text)
35            except:
36                pass
37            return btc_total
38
39        def getNewAddress(self):
40            payload = {'password': self.password}
41            url = 'https://blockchain.info/merchant/' + self.username +
                  '/new_address'
42            address = ""
43            try:
44                r = requests.get(url, params=payload, verify=True, timeout=10)
45                address = eval(r.text, {}, {})['address']
46            except:
47                pass
48            return address
49
50
51    def new_payment(amount):
52        qr = _QrCodeGenerator(username, password)
53        # Try 3 times to reach blockchain.info
54        i = 3
55        total = qr.toBTC(amount)
56        while not i == 0:
57            if not total == -1:
58                i = 0
59            else:
60                i -= 1
61                time.sleep(0.5)
62                total = qr.toBTC(amount)
63        address = qr.getNewAddress()
64        if not total == -1:
65            i = 3
66            while not i == 0:
67                if address:
68                    i = 0
69                else:
70                    i -= 1
71                    time.sleep(0.5)
72                    address = qr.getNewAddress()
73        if total == -1 or not address:
74            return
75        else:
76            image = qr.generate(address, total)
77            return [image, address, total]
```

7.5.5 Gestione delle richieste di pagamento

Il main thread del client rimane in attesa fino a quando non viene rilevata la pressione del pulsante sulla breadboard da parte di un cliente. Per lo scopo si è utilizzata la funzione *add_event_detect* che permette di tenere traccia di eventuali oscillazioni su un pin di GPIO. Il controllo sull'evento deve essere effettuato manualmente con la funzione *event_detected*, la quale viene chiamata ogni secondo per mantenere una certa reattività e al tempo stesso non consumare cicli di cpu. La funzione *event_detected*, a differenza del polling, non perde l'evento anche se viene chiamata dopo che l'oscillazione è terminata.

Non si è potuto utilizzare *wait_for_edge* perché la libreria non permette a un programma di aspettare su due pin di GPIO contemporaneamente, anche se su thread differenti.

```
from threading import Thread
from displayImplementation import Display
import ultrasonic
import readNFC

[...]

nfc = readNFC.CheckNfc(4, 22, 20, 18) # GPIO port used for the NFC reader

[...]

def main():
    t = Thread(target=ultrasonic.Ultrasonic(19, 26).start_measuring)
    t.daemon = True
    t.start() # run the ultrasonic check in a separated thread
    Display.instance().turn_on()
    t = Thread(target=nfc.start_reading)
    t.daemon = True
    t.start() # run the nfc thread in a separated thread
    t = Thread(target=wait_loop)
    t.daemon = True
    t.run() # run in the same main thread
```

7.5.6 Invio Email di notifica

Per l'invio dell'email per segnalare che un determinato distributore è prossimo all'esaurimento si è utilizzata la libreria *smtplib*

```
import smtplib


def send_gmail_email(username, passwd, to, subject, body):
    msg = """\From: %s\nTo: %s\nSubject: %s\n\n%s
    """ % (username, to, subject, body)
    try:
        server_ssl = smtplib.SMTP_SSL("smtp.gmail.com", 465)
        server_ssl.ehlo()
        server_ssl.login(username, passwd)
        server_ssl.sendmail(username, to, message)
        server_ssl.close()
    except:
        logger.info("An error occurred sending the email")
```

7.5.7 Invio richieste TCP

Per inviare segmenti TCP viene utilizzata la libreria *twisted* e le richieste avvengono sempre in asincrono.

```
def check_payment(address_btc, amount):
    address = address_btc
    total = amount
    with lock:
        completed = False
        current = base64.standard_b64encode(os.urandom(8))
    ss.sendMsg(NEW_REQUEST + "," + str(current) + "," + str(address) +
        "," + str(total), server_ip, server_port)
    e.wait(600)   # 10 minutes
    e.clear()
    with lock:
        current = ""
    return completed


def has_user_enough_points(user):
    with lock:
        enough = False
        user_id = user
    ss.sendMsg(CHECK_USER + "," + str(user), server_ip, server_port)
    e_short.wait(20)  # 20 seconds
    e_short.clear()
    with lock:
        user_id = 0
        return enough


def are_nearby_not_empty(zone):
    with lock:
        nearby = False
    ss.sendMsg(CHECK_NEARBY + "," + str(zone), server_ip, server_port)
    e_short.wait(20)  # 20 seconds
    e_short.clear()
    with lock:
        return nearby


def cancel():
    if current:
        with lock:
            if current:
                ss.sendMsg(CANCEL + "," + str(current), server_ip,
                    server_port)
                current = ""
                e.set()
```

7.6 Implementazione della parte relativa al server

7.6.1 Organizzazione a task

Il server deve essere in grado di ricevere e gestire le richieste provenienti dai client.

Per ricevere i segmenti TCP viene utilizzato il thread principale del server che, attraverso la libreria *twisted* offerta da Python, rimane sempre in ascolto per ricevere eventuali pacchetti evitando comunque il busy waiting.

Per le richieste viene utilizzato un thread per accodare ad intervalli regolari tutte le richieste e un pool di thread per gestirle.

I thread necessari al funzionamento vengono creati utilizzando la libreria *threading*.

```
1  from twisted.internet import protocol, reactor, defer
2  [...]
3
4  if __name__ == '__main__':
5      num_worker_threads = 2
6      q = Queue.Queue()
7      for i in range(num_worker_threads):
8          t = threading.Thread(target=_wait_for_request)
9          t.daemon = True
10         t.start()
11     t = threading.Thread(target=_send_requests_loop)
12     t.daemon = True
13     t.start()
14     ss = SegmentSender()
15     ss.start()
16     reactor.run() # @UndefinedVariable
```

7.6.2 Gestione lista clienti

Come specificato nella sezione 6.2.4 prima che il server aggiorni il file csv viene generata una copia temporanea per sicurezza. Per lo scopo è stata utilizzata la funzione *NamedTemporaryFile* presente nella libreria *tempfile*.

```
from tempfile import NamedTemporaryFile

def _get_user_points(user):
    with open(filename, 'a+') as f:
        f.seek(0)
        reader = csv.reader(f, delimiter=',')
        for row in reader:
            if user == row[0]:
                return row[1]
    return 0


def _increase_user_points(user, increase):
    with open(filename, 'a+') as csvFile, temp_file:
        csvFile.seek(0)
        reader = csv.reader(csvFile, delimiter=',', quotechar='"')
        writer = csv.writer(temp_file, delimiter=',', quotechar='"')
        found = False
        for row in reader:
            if user == row[0]:
                row[1] = int(row[1]) + increase
                found = True
            writer.writerow(row)
        if not found:
            writer.writerow([user, 1])
    shutil.move(temp_file.name, filename)
```

7.6.3 Servire le richieste

Per servire le richieste si è scelto di utilizzare due liste:

- Una lista contenente tutte le richieste attualmente in corso
- Una coda con le richieste che bisogna soddisfare il prima possibile

Esattamente come spiegato nelle sezioni 6.2.4 e 6.2.5 un thread accoderà le richieste presenti sulla lista ad intervalli regolari utilizzando la funzione *_send_requests_loop()*. I thread nel pool invece restano in attesa, su quella coda, dell'arrivo di richieste da servire.

Nel codice seguente è presentata la funzione chiamata quando la richiesta corrisponde alla verifica della ricezione di un pagamento.

```
def _send_requests_loop():
    while True:
        time.sleep(SLEEP)
        for key in list_active:
            try:
                q.put(list_active[key])
            except:
                pass


def _new_request(received_list, tcp_object):
    # received_list[1] = unique number, [2] = addressBTC, [3] = amount
    global lock
    with lock:
        if received_list[1] in list_active and
            list_active[received_list[1]][2] >= LIMIT:
            list_active.pop(received_list[1]) # The empty message will be
                sent with the next if...else
    if received_list[1] in list_active:
        sc = serverCheck.ServerCheck(received_list[2], received_list[3])
        if sc.is_payment_arrived():
            ds.sendMsg("True," + str(received_list[1]), tcp_object[1][0],
                tcp_object[1][1])
            with lock:
                if received_list[1] in list_active:
                    list_active.pop(received_list[1])
        else:
            with lock:
                if received_list[1] in list_active:
                    list_active[received_list[1]][2] += 1
                else:
                    ds.sendMsg("False," + str(received_list[1]),
                        tcp_object[1][0], tcp_object[1][1])
    else:
        ds.sendMsg("False" + str(received_list[1]), tcp_object[1][0],
            tcp_object[1][1])
```

Capitolo 8

Testing

In questo capitolo vengono descritti i test che sono stati effettuati per verificare la correttezza dell'implementazione.

8.1 Test utilizzando unittest

Utilizzando la libreria unitteset si è potuto creare dei casi di test automatici per verificare le seguenti funzionalità:

- Lettura tag NFC
- Generazione del QR code
- Aggiornamento del proprio stato per un distributore (pieno o esaurito), con richiesta di controllo nella zona di appartenenza
- Gestione del saldo punti per i clienti

```
import unittest
from threading import Thread
import readNFC
import qrgenerator
from PIL import Image
import clientTCP


class TestMethods(unittest.TestCase):

    def test_nfc_read_tag(self):
        nfc = readNFC.CheckNfc(4, 22, 20, 18)
        t = Thread(target=nfc.start_reading)
        t.daemon = True
        t.start() # run the nfc thread in a separated thread
        self.assertEqual(nfc.user, 0)
        test_nfc = "b1dc1337cdaa1122"
        message = "Tap the NFC tag {0} and press Enter to continue..."
        raw_input(message.join(test_nfc))
        self.assertEqual(nfc.user, "b1dc1337cdaa1122")

    def test_generate_qr(self):
        payment_data = qrgenerator.new_payment(2)
        self.assertIsInstance(payment_data[0], Image)
        self.assertNotEqual(payment_data[1], "")
        self.assertNotEqual(payment_data[2], -1)

    def test_tcp(self):
        clientTCP.update_status(2, "123", True)
        self.assertTrue(clientTCP.are_nearby_not_empty(2))
        clientTCP.update_status(2, "123", False)
        self.assertFalse(clientTCP.are_nearby_not_empty(1))
        self.assertFalse(clientTCP.are_nearby_not_empty(2))
        self.assertFalse(clientTCP.has_user_enough_points("1234"))
        clientTCP.increase_points("1234")
        self.assertFalse(clientTCP.has_user_enough_points("1234"))
        clientTCP.increase_points("1234")
        clientTCP.increase_points("1234")
        clientTCP.increase_points("1234")
        clientTCP.increase_points("1234")
        # With 5 points the user get a free drink
        self.assertTrue(clientTCP.has_user_enough_points("1234"))
        clientTCP.increase_points("1234")
        self.assertFalse(clientTCP.has_user_enough_points("1234"))


if __name__ == '__main__':
    suite = unittest.TestLoader().loadTestsFromTestCase(TestMethods)
    unittest.TextTestRunner(verbosity=2).run(suite)
```

8.2 Test effettuati manualmente

I test che non si è potuto automatizzare sono stati effettuati manualmente.

Per quanto riguarda il sensore di prossimità, si è provato ad allontanarsi dal distributore durante una richiesta di pagamento e, una volta passato il tempo preimpostato, lo schermo si è spento correttamente ed è stata inviata la notifica di annullamento al server.

Successivamente si è provato a completare un normale processo di pagamento con successiva erogazione della bevanda. Sia autenticandosi con tag NFC che senza.

Si è provato a richiedere una bibita alla spina senza effettuare il pagamento per i successivi 10 minuti e il client ha annullato con successo la richiesta automaticamente.

Infine è stata simulata la mancata connessione ad Internet da parte del server, o un malfunzionamento al sito Blockchain.info, e sullo schermo del distributore è stato visualizzato il messaggio che informava il cliente di riprovare successivamente.

Capitolo 9

Conclusioni

9.1 Futuri sviluppi

Il progetto, per come è stato implementato, prevede l'utilizzo di client con un singolo distributore alla spina ciascuno. A questo proposito si potrebbe pensare di avere diverse postazioni per l'erogazione alla spina con vari tipi di bevande tutti collegati ad un singolo client.

Attualmente la quantità di bibita che viene erogata è fissa, ma aggiungendo una pulsantiera si potrebbe dare la possibilità ai clienti di scegliere la quantità che desiderano comprare.

Inoltre per rendere i distributori ancora più indipendenti si potrebbe pensare ad un sistema per aggiornare il software da remoto evitando di dover mandare un tecnico in loco.

9.2 Valutazioni finali

Il processo di sviluppo è stato abbastanza lineare. Le maggiori difficoltà sono state riscontrate nella mancanza totale di documentazione e esempi per alcuni componenti utilizzati nel progetto come per esempio il flow meter.

La scelta del linguaggio di programmazione da utilizzare per il progetto è ricaduta su Python perché è il linguaggio principale del Raspberry Pi. D'alto canto usare Python mi ha rallentato molto durante il progetto visto che, essendo un linguaggio nuovo per me, in diverse occasioni ho dovuto fermare lo sviluppo per studiare il linguaggio di programmazione.

I Bitcoin, utilizzati in un mondo in cui sempre più oggetti semplici di uso comune diventano intelligenti entrando a far parte del così detto Internet of Things, offre sicuramente dei vantaggi sia per gli acquirenti che per i venditori. D'altro canto però la sua attuale diffusione è ancora molto limitata e la maggior parte delle persone non solo non possiede Bitcoin ma ne ignora del tutto la loro esistenza. Inoltre negli ultimi anni il valore dei Bitcoin rispetto agli euro o i dollari è variato molto velocemente e per alcuni questa volatilità può essere un deterrente, infatti come disse il fondatore della piattaforma *BTC.SX Joseph Lee*: *"This is damaging for the huge array of businesses who simply want to be involved in the space by accepting bitcoins for their products, whether its food or hardware, but unknowingly end up playing a long speculative bet on the price"* [5].

Ringraziamenti

Ringrazio la mia famiglia che mi è stata vicino e mi ha sempre sostenuto.

Ringrazio il mio relatore il prof. Alessandro Ricci per tutti i consigli che mi ha dato per la stesura della tesi.

Infine ringrazio gli ottimi amici che ho trovato in questa facoltà.

Elenco delle figure

Bibliografia

[1] "le nuove regole sull'alcol". http://www.fipe.it/files/pubblicazioni/GUIDA_PRATICA_ALCOl.pdf.

[2] "p2413 - standard for an architectural framework for the iot". http://standards.ieee.org/develop/project/2413.html.

[3] "welcome to the european vending association!". http://www.vending-europe.eu/eva/home.html.

[4] K. Ashton. "that 'internet of things' thing". http://www.rfidjournal.com/articles/view?4986, 2009.

[5] D. Bradbury. "can bitcoin's price ever be stable?". http://www.coindesk.com/can-bitcoins-price-ever-stable/, 2014.

[6] J. Canter. Quick response (qr) code generation in vending machines or kiosks for customer engagement. https://www.google.com/patents/US20130035787, 2013. US Patent App. 13/565,620.

[7] W. Dai. "b-money". http://www.weidai.com/bmoney.txt, 1998.

[8] D. Evans. "the internet of things: How the next evolution of the internet is changing everything". http://www.iotsworldcongress.com/documents/4643185/0/IoT_IBSG_0411FINAL+Cisco.pdf, 2011.

[9] Intel and ADLINK. "improving vending machine profitability with the iot".

https://www-ssl.intel.com/content/dam/www/program/embedded/internet-of-things/blueprints/iot-vending-machine-adlink-blueprint.pdf, 2014.

[10] E. Jaffe. "old world, high tech". http://www.smithsonianmag.com/science-nature/old-world-high-tech-141284744/?page=2, 2006.

[11] D. Leason and S. Sullivan. Vending machine. *https://www.google.com/patents/US6575363*, 2003. US Patent 6,575,363.

[12] S. Nakamoto. "bitcoin: A peer-to-peer electronic cash system". https://bitcoin.org/bitcoin.pdf, 2008.

[13] N. Negroponte. MIT.

[14] M. Powers. "how iot is revolutionizing vending machine capabilities". https://networkingexchangeblog.att.com/enterprise-business/iot-revolutionizing-vending-machine-capabilities/, 2015.

[15] W. Reade and J. Lindsay. Rfid system and method for vending machine control. *https://www.google.com/patents/US7490054*, 2009. US Patent 7,490,054.

[16] E. R. T. Dierks. Rfc 5246 - the transport layer security (tls) protocol, version 1.2. https://tools.ietf.org/html/rfc5246, 2008.