UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

PSSS: SmartLock

Anno Accademico [2018/2019]

docente
Prof. Annarita Fasolino

candidati
Federica Ventriglia
Salvatore Capuozzo
Maria Luisa Farina
Filippo Ferrandino

FRIDERICVS DEI GRA IMPERATOR ROMANORVM SEMP AVGVST

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Raccolta ed Analisi Dei Requisiti

1.1 Specifiche di Massima

L'applicazione SmartLock nasce con lo scopo di sostituire gli standard sistemi di interfono per palazzi, case e appartamenti, con una versione digitalizzata che integra l'utilizzo del riconoscimento visivo (noto anche come FaceID) di un dispositivo iOS a dei sensori IoT. SmartLock implementerà quindi, oltre alle normali funzionalità di citofono e apertura da remoto di un cancello, anche il riconoscimento facciale degli eventuali inquilini di un appartamento con conseguente apertura automatica del cancello a cui è collegato.

1.1.1 Attori

Gli attori coinvolti nell'utilizzo del sistema sono i seguenti:

- **User**: identificato come una persona fisica, tipicamente inquilino di un appartamento, il cui volto può essere registrato e mantenuto nella base di dati e utilizzato per l'apertura automatica del cancello.
- **Manager**: persona fisica con accesso a funzionalità avanzate del sistema.
- **External User**: identificato come una persona fisica, non residente all'interno dell'appartamento e richiedente l'accesso attraverso il citofono

- **Timer**: dispositivo utilizzato per il collegamento del sistema iOS alla rete IoT.
- **Vision**: framework di riconoscimento facciale
- **Factory Builder**: costruttore del dispositivo, con accesso privilegiato

1.2 Requisiti Funzionali

Andiamo ora a descrivere i requisiti funzionali dell'applicazione, tutte le funzioni che devono essere implementate affinchè il sistema realizzato svolga il suo lavoro. Il riferimento relativo a tali requisiti è indicato dall'identificativo RF*:

Riferimento	Descrizione
RF01	Il sistema permette ad un utente di accedere al palazzo con il proprio identificativo (volto).
RF02	Il sistema permette ad un utente di registrare/aggiornare il proprio identificativo (volto).
RF03	Il sistema permette ad un utente di utilizzare l'interfaccia per visualizzare le informazioni sul palazzo e citofonare ad uno specifico appartamento.
RF04	Il sistema permette di attivare manualmente il processo di scansione del volto qualora non si attivasse in automatico.
RF05	Il sistema permette di inserire un codice di accesso numerico qualora l'utente non venga riconosciuto con il sistema descritto dal RF01.
RF06	Il sistema permette ad un utente di riprendere e visualizzare chi citofona al proprio appartamento.
RF07	Il sistema permette di fornire un pass momentaneo ad utenti esterni.
RF08	Il sistema permette di interfacciarsi con un eventuale sistema di allarme già presente.
RF09	Il sistema permette al gestore di registrare nuovi utenti.
RF10	Il sistema permette al gestore di visualizzare la cronologia degli accessi.
RF11	Il sistema permette alla ditta di fabbrica di resettare il sistema, collegarlo al sistema di allarme, di gestire il gestore e di effettuare manutenzioni.

Tabella 1.1: Requisiti Funzionali

1.3 Requisiti Non Funzionali

Analogamente sono stati descritti i requisiti di tipo non funzionale, ossia relativi a richieste di performance e/o particolari scelte HW/SW necessarie per il corretto funzionamento del sistema.

Riferimento	Descrizione
NF01	L'applicazione deve essere fluida ed intuitiva, al punto di rendere gradevole il suo utilizzo a discapito della versione classica di interfono.
NF02	L'hardware su cui l'applicazione software sarà sviluppata dovrà avere un sistema operativo iOS di versione 11.0 o successiva.
NF03	L'applicazione dovrà fornire una soluzione trasparente all'utente ad eventuali anomalie al fine di non interrompere il normale utilizzo del sistema.
NF04	L'architettura Hardware del dispositivo utilizzato deve possedere tecnologia FaceID, in particolare una fotocamera TrueDepth.

Tabella 1.2: Requisiti Non Funzionali

1.4 Requisiti Architetturali

I requisiti progettuali esposte fino ad ora comportano una serie di scelte limitate dal punto di vista architetturale che portano l'implementazione a spostarsi in una particolare direzione. In particolare possiamo identificare una serie di requisiti architetturali necessari per la realizzazione di SmartLock.

Riferimento	Descrizione
AR01	L'applicazione verrà sviluppata in linguaggio Swift, orientato per applicazioni iOS.
AR02	L'ambiente di sviluppo utilizzato per l'implementazione è Xcode.
AR03	La comunicazione tra dispositivo e cancello sarà gestita dal protocollo Bluetooth, reso disponibile dalle librerie esistenti in Swift e utilizzabile grazie ad un modulo per l'Arduino collegato al relé che controlla il cancello.
AR04	L'applicazione interna al sistema verrà sviluppata seguendo lo standard delle applicazioni iOS, ovvero adottando una architettura di tipo Model-View-Controller.

Tabella 1.3: Requisiti Non Funzionali

1.4.1 Tecnologia True Depth e FaceID

Il suo scopo è quello di creare una mappa tridimensionale del volto dell'utente possessore dello smartphone, analizzando oltre 30.000 punti del viso e memorizzare tali informazioni per l'utilizzo successivo. Attraverso la presenza di numerosi sensori biometrici all'interno della fotocamera il volto di un utente è mappato su una griglia che misura le varie distanze per identificare i punti fondamentali del viso. Tutti questi punti proiettati sul viso saranno le coordinate necessarie per riprodurre fedelmente una copia del viso da memorizzare.

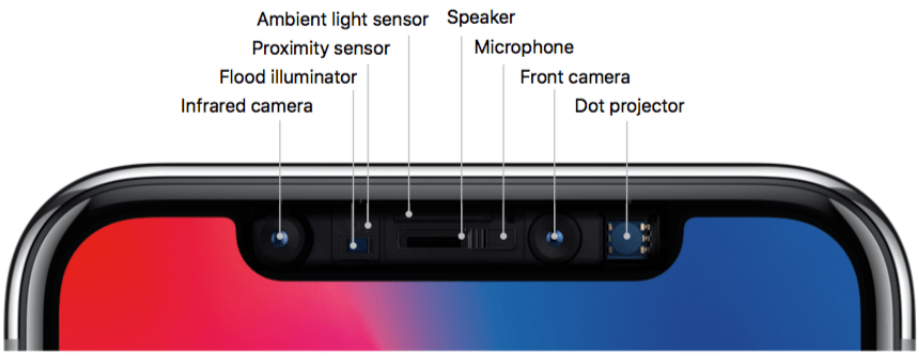


Figura 1.4.1: True Depth Camera

L'utilizzo principale della TrueDepth Camera riguarda lo sblocco del dispositivo, ossia la con-

figurazione del FaceID.

Per ottenere il riconoscimento di un volto viene utilizzato il cosiddetto Dual Sensor. Questo sensore si basa sull'integrazione di almeno due telecamere nel dispositivo, nel fare questo, garantisce che lo scatto fatto acquisiti profondità e quindi rileva se è una foto o una persona. FaceID è in grado di riconoscere il viso in meno di un secondo, raggiungendo la massima sicurezza in qualsiasi installazione ad un prezzo accessibile per qualsiasi utente.

Nel nostro progetto andiamo ad estendere tale concetto, applicandolo al caso d'uso di interesse che sfrutterà il riconoscimento del volto e la sua memorizzazione per avere accesso a funzionalità che necessitano di autenticazione, in questo caso l'apertura di un cancello.

1.4.2 Xcode

Xcode è l'ambiente di sviluppo integrato utilizzato per creare, testare, effettuare il debug delle applicazioni su piattaforma iOS.

Le principali caratteristiche di di Xcode sono le seguenti:

- **Interface Builder.** E' possibile costruire ogni vista relativa alla propria applicazione utilizzando un apposito editor chiamato Interface Builder. Questo permette di inserire oggetti nella GUI attraverso un'interfaccia semplice da utilizzare e che permette di aggiungere diverse informazioni all'oggetto (posizione, funzionalità, tipologia) e anche la loro connessione con l'applicazione stessa. Esiste comunque la possibilità di implementare tali oggetti via codice.

- **Assistant Editor.** Durante la fase di programmazione si è spesso portati a dover modificare file correlati tra loro, l'assistant editor permette al programmatore di visualizzare i documenti in questione nella stessa finestra senza bisogno di aprire più editor alla volta. Questa caratteristica velocizza di molto il flusso di lavoro, aiutando molto il programmatore.

- **Identificazione e Correzione degli errori.** Xcode ispeziona il codice in tempo reale ed identifica anche errori di programmazione. La riga errata viene evidenziata fornen-

do i dettagli relativi all'errore, talvolta vengono proposte delle possibili soluzioni per risolverli.

- **Controllo del Codice Sorgente,** permette di salvare i file di progetto in una repository (Git, Bitbucket ecc.) direttamente da Xcode, mantenendo così un controllo della versione del progetto. Grazie a questa caratteristica è possibile lavorare in team allo stesso progetto, tenendo traccia delle modifiche fatte dai vari team members.

1.4.3 Architettura iOS

L'architettura del sistema operativo iOS si articola in più livelli sovrapposti. Al livello più basso, iOS agisce come intermediario tra l'hardware sottostante e le applicazioni che appaiono sullo schermo. Ai livelli più bassi del sistema ci sono i servizi fondamentali e le tecnologie sulle quali si basano tutte le applicazioni mentre i livelli più alti contengono i servizi e le tecnologie più sofisticati. Quando si sviluppa una applicazione, dove possibile, si deve prediligere l'uso di **framework** di alto livello piuttosto che dei servizi provenienti da livelli inferiori. Infatti i frameworks di livello superiore sono stati creati appositamente per fornire astrazioni ai livelli inferiori. Queste astrazioni, in genere, rendono molto più facile la scrittura di codice, perché riducono la quantità di righe di codice da scrivere e racchiudono caratteristiche potenzialmente complesse, come i socket per le connessioni e i thread.

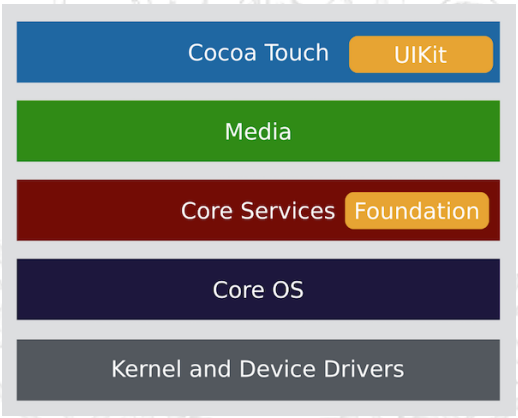


Figura 1.4.2: iOS Architettura a Strati

1.4.4 Collegamento al Cancello: Arduino Project

Per la realizzazione del componente di sensoristica relativo all'apertura del cancello abbiamo sfruttato un controller Arduino collegato ad un modulo Bluetooth ed un Relay per l'invio del segnale di apertura al cancello. Di seguito riportiamo i dettagli del microcontrollore e del modulo bluetooth scelti:

- Arduino Uno
- Modulo DSD TECH HM-10 Bluetooth 4.0 BLE IBeacon UART

Per la prima realizzazione del sistema si è sostituito al relè che è connesso al cancello un collegamento ad un LED per testare la bontà del collegamento bluetooth. Lo schema di funzionamento è il seguente:

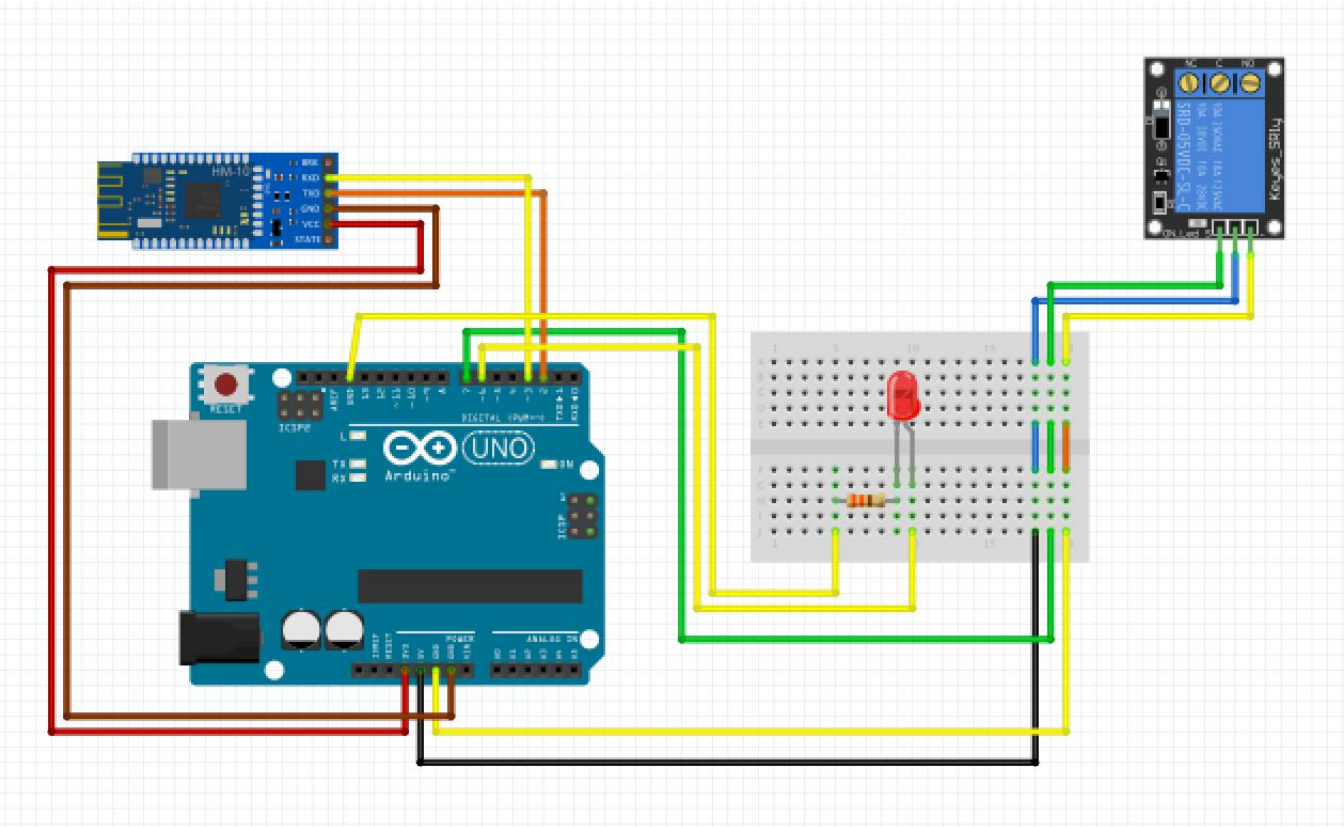


Figura 1.4.3: Schema Arduino

1.4.5 iOS SDK: Model View Controller Pattern

Una qualsiasi applicazione per iOS o per OS X che utilizza gli strumenti messi a disposizione dai vari framework, se progettata correttamente, è basata su un modello Model View Controller (MVC). MVC non solo realizza la separazione tra dati e interfaccia utente ma svincola anche la logica di controllo dell'applicazione dai dati. Il modello è basato sulla separazione dei compiti fra tre componenti software che interpretano tre ruoli principali:

- **Model**: Gli oggetti che costituiscono il modello rappresentano parti- colari conoscenze e competenze, contengono i dati di un'applicazione e definiscono la logica di utilizzo di tali dati. Infatti una applicazione con MVC ben progettata ha tutti i dati importanti incapsulati in oggetti del modello. Tutti i dati che fanno parte dello stato persistente dell'applicazione devono risiedere negli oggetti del modello una volta che i dati vengono caricati nell'applicazione. Idealmente, un oggetto del modello non ha alcun collegamento esplicito con l'interfaccia utente. In generale un oggetto del model non dovrebbe occuparsi di compe è realizzata la grafica relativa ad esso.

- **View**: Un oggetto di tipo vista è in grado di visualizzar e a volte modificare i dati del modello. La view non dovrebbe essere responsabile per la memorizzazione dei dati che presenta, ma può memorizzare nella cache alcuni dati per motivi di prestazioni. Il framework UIKit definisce oggetti che possono essere riutilizzati assicurando lo stesso funzionamento in diverse applicazioni, garantendo un elevato livello di coerenza per aspetto e comportamento tra le applicazioni.

- **Controller**: Un oggetto di controllo funge da intermediario tra gli oggetti della view dell'applicazione e gli oggetti del modello. In un tipico modello MVC, quando gli utenti immettono un valore o indicano una scelta attraverso un oggetto di visualizzazione, il valore o la scelta viene comunicato ad un oggetto di controllo. L'oggetto di controllo può interpretare l'input dell'utente in alcune applicazioni specifiche o può indicare ad un oggetto del modello cosa fare con questo ingresso.

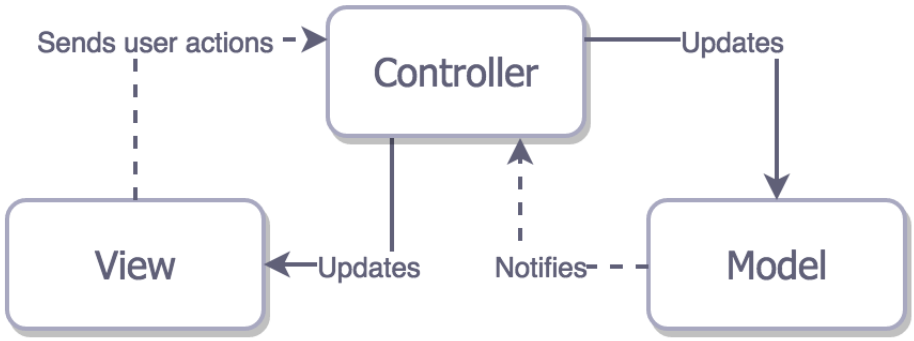


Figura 1.4.4: MVC in iOS

1.5 Analisi in Dettaglio dei Requisiti

La logica generale di funzionamento dell'applicazione riguarda appunto il riconoscimento automatico di un utente autorizzato con il conseguentemente accesso al sistema. In particolare sono previste due modalità di autenticazione: la prima, coincidente anche con il principale metodo di utilizzo del sistema, è il riconoscimento automatico tramite il volto di un Utente ed il conseguente accesso a quella che è la funzionalità di ingresso, ossia l'apertura di un cancello automatico; la seconda modalità è invece inserita come metodo secondario (o di backup) da utilizzare nell'eventualità che il primo metodo fallisca oppure se l'attore che sta utilizzando il sistema non dispone dei requisiti necessari per poter sfruttare il riconoscimento facciale, questa prevede l'utilizzo di un codice numerico univoco che svolge il ruolo di chiave di accesso.

Per migliorare la descrizione sono stati utilizzati vari diagrammi UML per modellare il funzionamento precedentemente descritto testualmente. In particolare presentiamo:

- Diagramma di Contesto con Boundary
- System Domain Model
- Diagramma dei Casi D'Uso.

1.5.1 Diagramma dei Casi D'Uso

La logica di funzionamento è anche descritta attraverso il seguente **Diagramma dei Casi D'uso** in cui sono raccolti i casi d'uso relativi al sistema, ricavati analizzando i requisiti descritti in precedenza. Questo evidenzia quindi le modalità di utilizzo dell'applicazione da parte dei fruitori e quindi le interazioni tra il sistema e la componente umana e/o hardware in termini di funzionalità richieste.

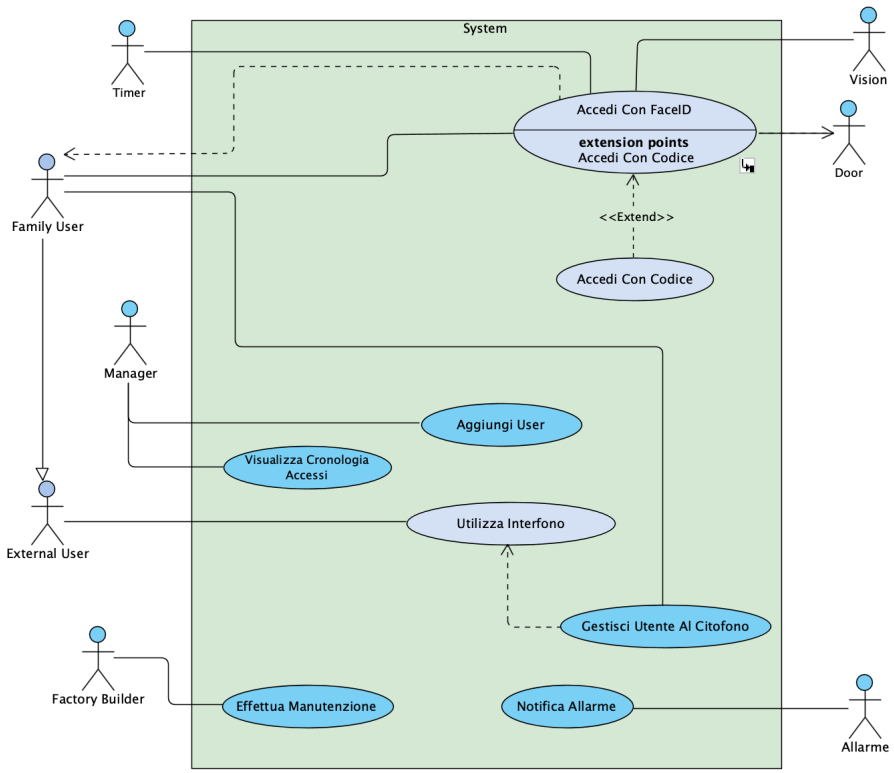


Figura 1.5.1: Diagramma dei Casi D'Uso

Descrizioni Brevi

Di seguito riportiamo delle descrizioni testuali dei casi d'uso in formato breve. Sono contemplati i soli casi d'uso che non saranno implementati nell'iterazione descritta nei capitoli successivi.

Nome	Aggiungi User
Descrizione	L'attore può aggiungere al log dei condomini un nuovo utente, associandovi nome, cognome, privilegi e codice di accesso univoco.
Attori	Manager User

Tabella 1.4: Caso D'Uso Breve: Aggiungi User

Nome	Utilizza Interfono
Descrizione	L'attore tramite la GUI può utilizzare la funzionalità interfono per citofonare all'appartamento desiderato.
Attori	External User/Family User/ Manager/Factory Builder

Tabella 1.5: Caso D'Uso Breve: Utilizza Interfono

Nome	Gestisci Utente al Citofono
Descrizione	L'attore principale può comunicare l'esito della richiesta generata nel caso d'uso Utilizza Interfono per aprire il cancello.
Attori	Family User

Tabella 1.6: Caso D'Uso Breve: Gestisci Utente al Citofono

Nome	Visualizza Cronologia Accessi
Descrizione	L'attore principale può accedere al log degli accessi del sistema attraverso privilegi di amministratore concessi tramite l'autenticazione con codice.
Attori	Manager User

Tabella 1.7: Caso D'Uso Breve: Visualizza Cronologia Accessi

Nome	Effettua Manutenzione
Descrizione	L'attore effettua modifiche al sistema accedendo con privilegi di factory builder.
Attori	Factory Builder

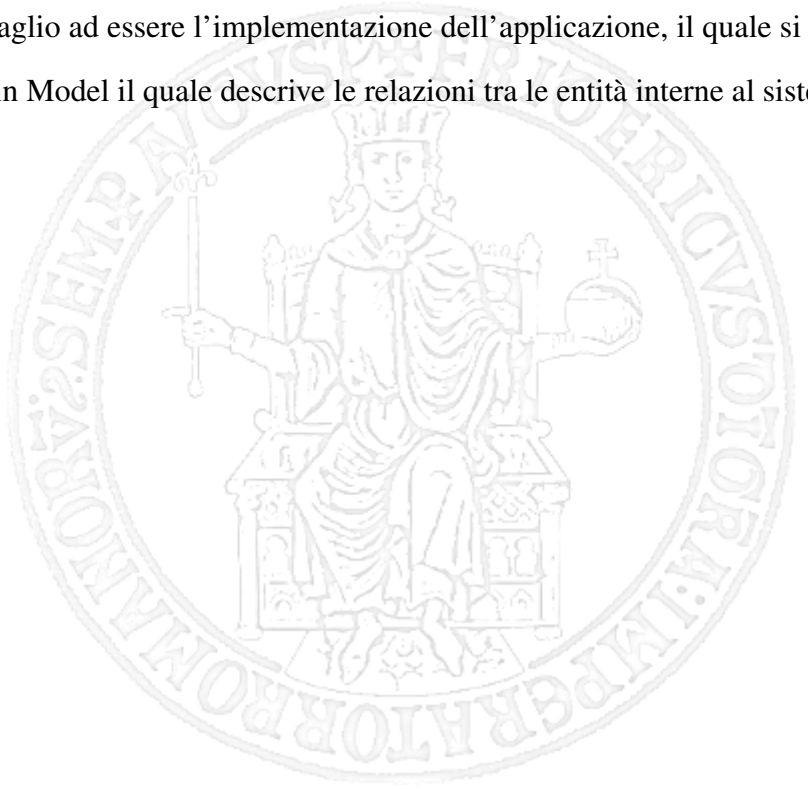
Tabella 1.8: Caso D'Uso Breve: Effettua Manutenzione

1.5.2 Diagramma di Contesto

Lo scopo è mostrare il contesto di una **View**. Il context rappresenta l'ambiente con cui il sistema o una specifica parte del sistema interagisce. UML non fornisce stereotipi per la realizzazione del diagramma di contesto pertanto sono stati utilizzati degli stereotipi custom per descrivere le entità coinvolte. Le entità principali che sono evidenziate nel diagramma di contesto sono:

1. Vision Framework: (framework) utilizzato per il riconoscimento del volto
2. GUI: (interfaccia utente) utilizzato per permettere l'interazione dell'utente con l'applicazione tramite interfaccia grafica
3. Alarm System: (sistema esterno) gestore di allarme da poter collegare al sistema per implementare eventuali sistemi di detection
4. Relay: (device esterno di output) interfaccia con il sistema di apertura del cancello

Le interazioni tra queste entità ed il nostro sistema sono gestite dal GRASP Controller, che andrà poi in dettaglio ad essere l'implementazione dell'applicazione, il quale si interfaccia con il System Domain Model il quale descrive le relazioni tra le entità interne al sistema.



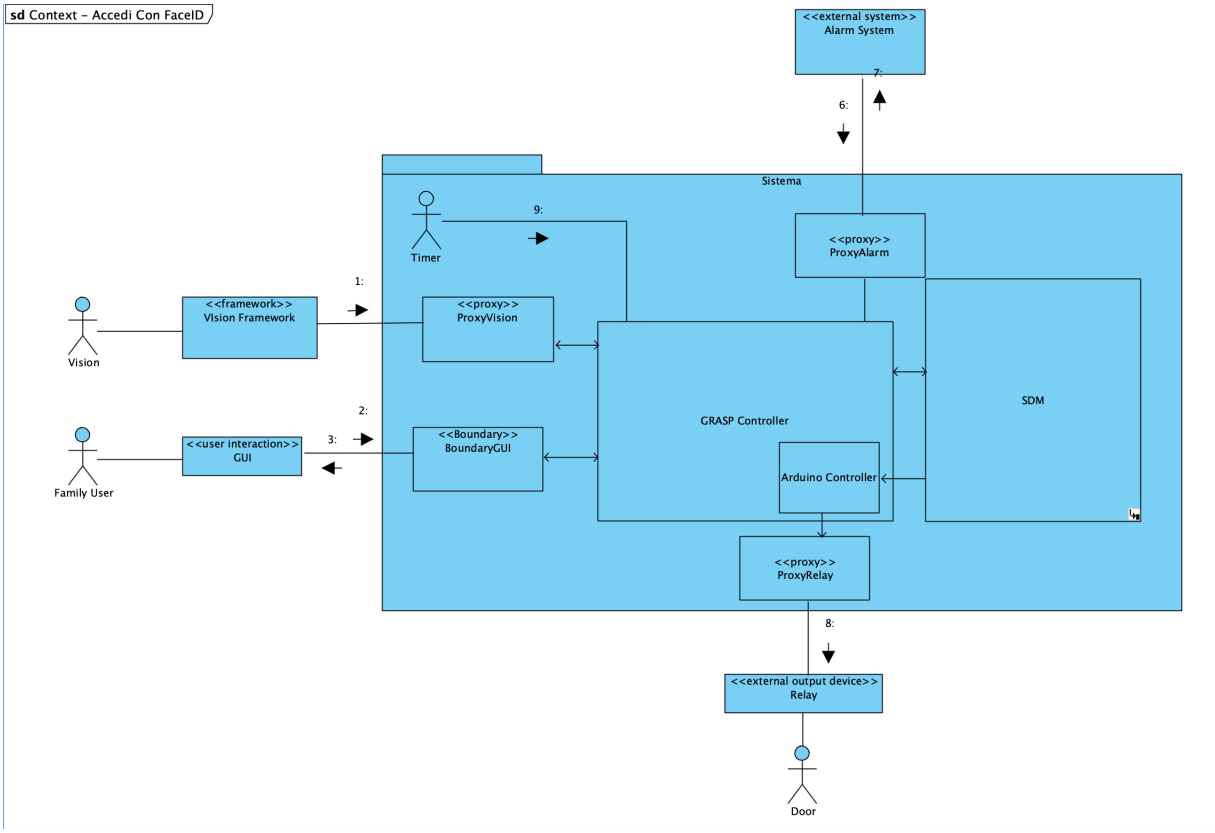


Figura 1.5.2: Diagramma di Contesto Caso D'Uso

1.5.3 System Domain Model

Il modello di dominio invece descrive il modello concettuale del sistema, in particolare del caso d'uso a cui facciamo riferimento. Nel seguente domain model sono descritte le entità che compongono il progetto con maggiore rilevanza ed evidenziate le relazioni tra esse. La realizzazione in UML si basa su un Class Diagram, dato che non sono forniti diagrammi specifici per un modello di dominio in Visual Paradigm. Nel SDM presentato ogni classe fa riferimento ad un'entità in gioco nel sistema, in particolare le relazioni non trattegiate fanno riferimento a relazioni che sono di interesse per il caso d'uso che si sta studiando.

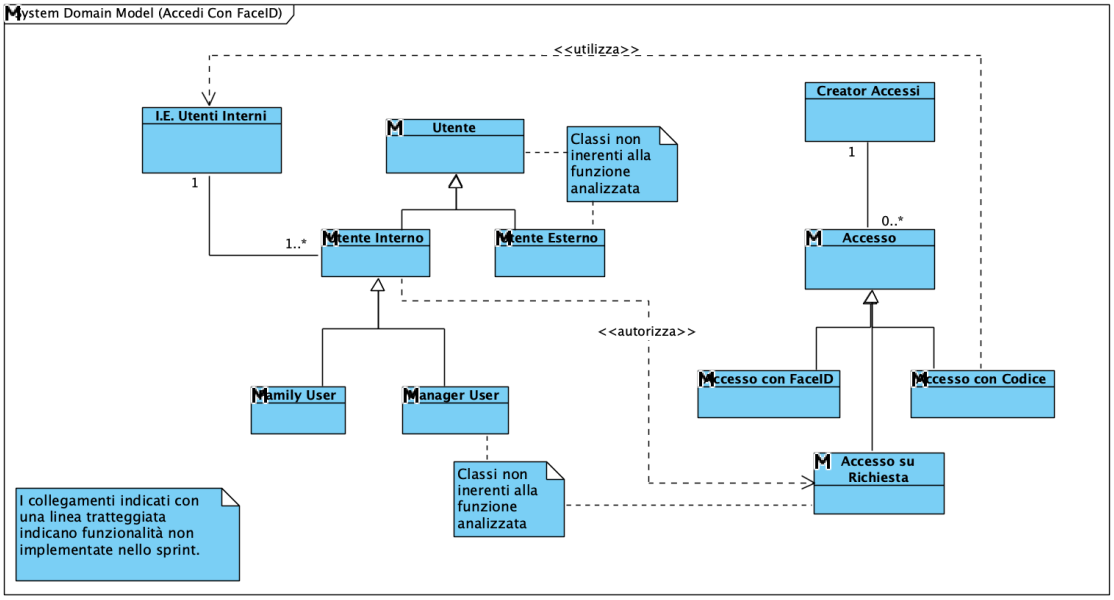


Figura 1.5.3: System Domain Model Caso D'Uso

In particolare nei capitoli successivi faremo riferimento al caso d'uso **Accedi con Face ID** che è stato scelto per essere implementato applicando un'approccio Agile, quindi incrementale.

Capitolo 2

Funzionalità Implementata

Verrà introdotto con maggior dettaglio il requisito funzionale in questione, ovvero quello che permette ad un utente di utilizzare il proprio identificativo (volto) per accedere al palazzo, per poter poi accedere al proprio appartamento.

2.1 Descrizione del Caso D'Uso

Come abbiamo già detto il caso d'uso che andremo a realizzare è **Accedi con FaceID**, ossia la richiesta al sistema di poter essere attivato all'avvicinamento dell'utente che vuole effettuare l'accesso all'appartamento, con conseguente scansione del volto automatica o su richiesta e il confronto di tale scansione con la base di dati del sistema per poter verificare la validità. Il sistema invierà una notifica relativa all'esito dell'operazione e, in caso di successo, una richiesta sarà inviata conseguentemente al modulo Bluetooth che procederà a sbloccare il relè a cui è collegato il cancello automatico permettendone l'apertura.

In caso di fallimento del riconoscimento visivo per motivi quali:

- volto non presente in database,
- scansione non avvenuta con successo dopo 3 tentativi consecutivi,
- indisponibilità della fotocamera ad eseguire la scansione

il sistema provvederà a proporre la modalità di accesso secondaria, richiamando quindi il caso d'uso **Accedi Con Codice**.

Di fatto quindi l'implementazione si concentrerà sui due casi d'uso già descritti, inoltre è prevista anche l'implementazione relativa alla **memorizzazione di un utent**e nel database, necessaria per poter realizzare le funzionalità di accesso.

Nome	Accedi Con FaceID
Descrizione	L'utente utilizza il riconoscimento facciale per sbloccare l'apertura del cancello.
Attori	Family User, Timer
PreCondizioni	Il Family User è registrato tra gli utenti con accesso Il Timer è predisposto al riconoscimento
Post Condizioni	Il Sistema riconosce l'utente e in caso di successo apre il cancello. L'accesso è registrato nel log di sistema.
Flow	1. L'utente si predispone di fronte alla camera del dispositivo 2. Il dispositivo riconosce la presenza di un volto ed effettua una scansione per riconoscerlo 3. Il dispositivo effettua 3 tentativi di riconoscimento e conferma il successo (se avvenuto) inviando un segnale al sistema che gestisce il cancello 4. Il cancello è aperto e l'utente può entrare 5. Il dispositivo ritorna allo stato di attesa di riconoscimento.
Flow Alternativi	Nel caso in cui il dispositivo non sia in uno stato di polling continuo l'utente può attivare il riconoscimento attraverso l'interazione touch con l'interfaccia L'utente può scegliere di autenticarsi con codice d'accesso
Eccezioni	Al passo 3 del normale flusso di esecuzione, se il dispositivo non riesce a riconoscere il volto dell'utente dopo 3 tentativi l'utente può attivare l'apertura del cancello tramite l'inserimento del codice univoco nella GUI. Successivamente il flow prosegue come nel caso standard, al passo 4.
Requisiti	RF01 RF04 RF05

Tabella 2.1: Caso D'Uso Dettagliato: Accedi con FaceID

2.2 Viste Dinamiche

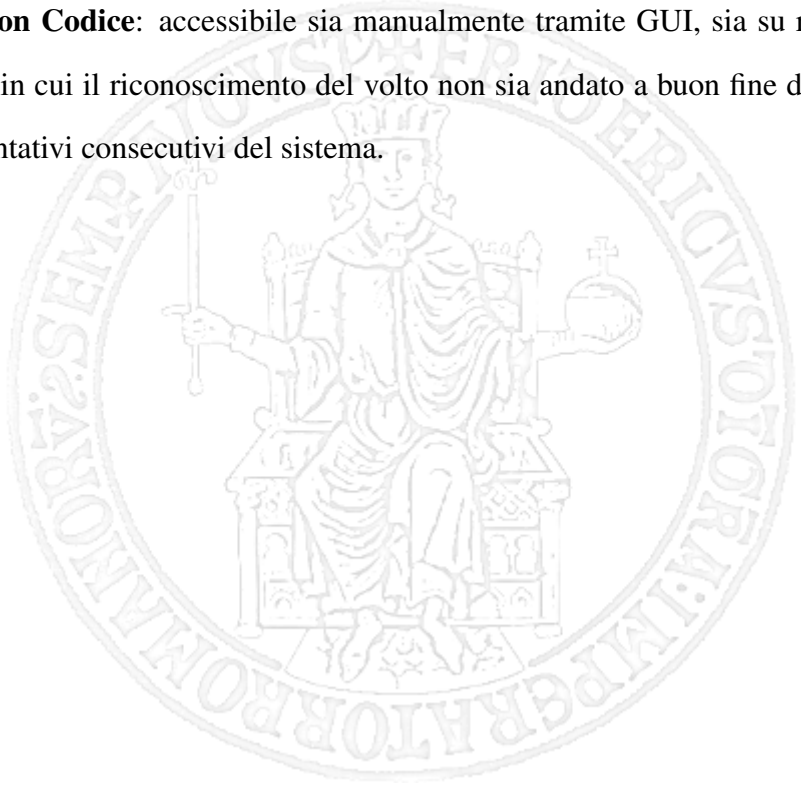
2.2.1 Sequence Diagram per le funzionalità dell'app SmartLock

Per descrivere più dettagliatamente il flow delle attività da implementare si scelto un Sequence Diagram, in questo diagramma si possono evidenziare chiaramente tre modalità di accesso, ossia tre possibili casi in cui l'utente può autenticarsi nel sistema.

1. **Accesso Automatico**: si presuppone che l'applicazione sia in uno stato di polling, in attesa di riconoscere un volto dal sensore della fotocamera che prevede un'interrogazione continua dello stato;

2. **Accesso Manuale**: nell'eventualità in cui la modalità di "polling" del sistema sia inattiva per guasti o preferenze dell'utilizzatore, l'utente può attivare manualmente il sensore facciale attraverso la GUI dell'applicazione, che provvederà ad eseguire le stesse operazioni conseguenti al caso automatico di accesso.

3. **Accesso con Codice**: accessibile sia manualmente tramite GUI, sia su richiesta nell'eventualità in cui il riconoscimento del volto non sia andato a buon fine dopo un numero pari a 3 tentativi consecutivi del sistema.



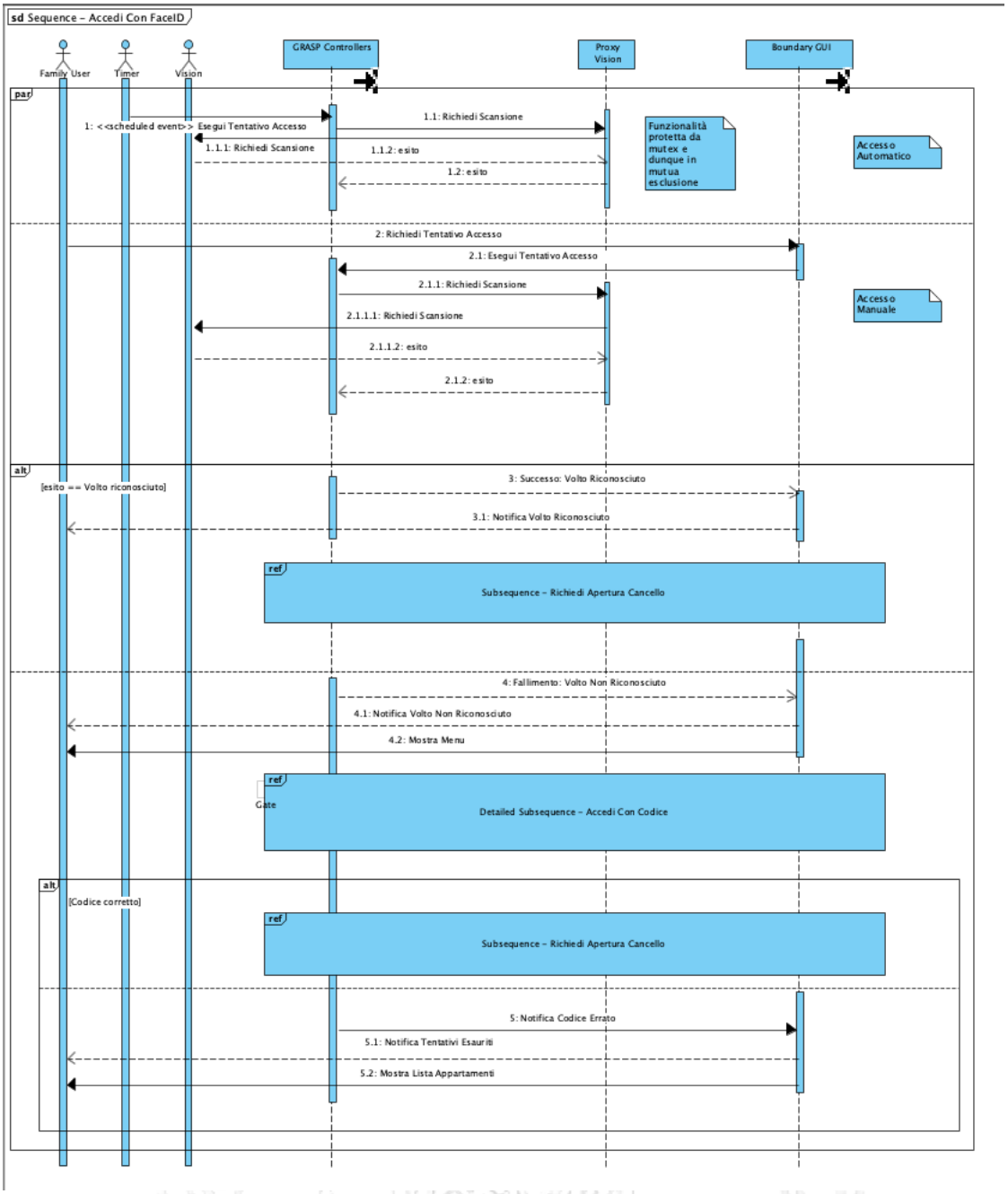


Figura 2.2.1: Sequence Diagram Caso D'Uso

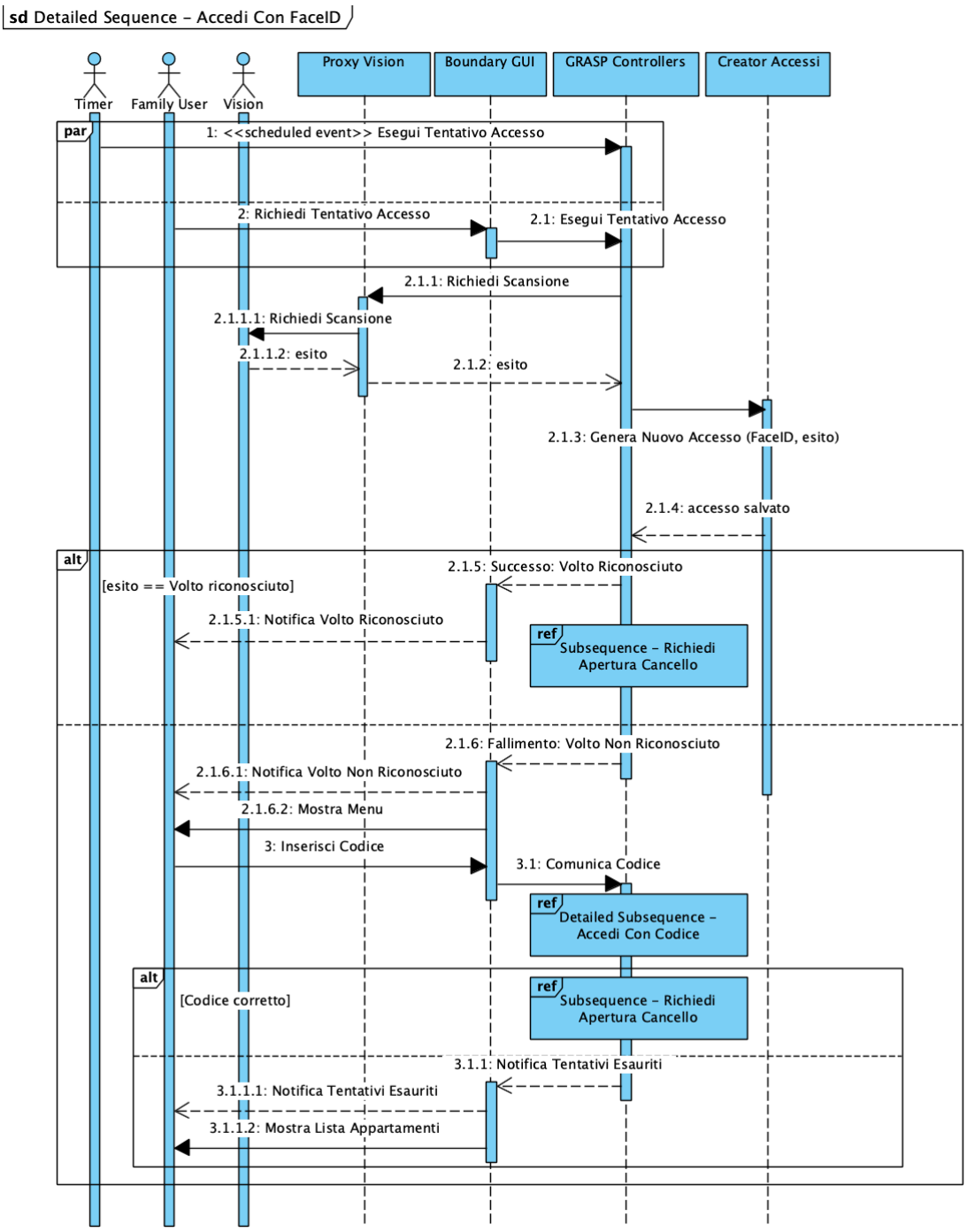


Figura 2.2.2: Detailed Sequence Diagram Caso D'Uso

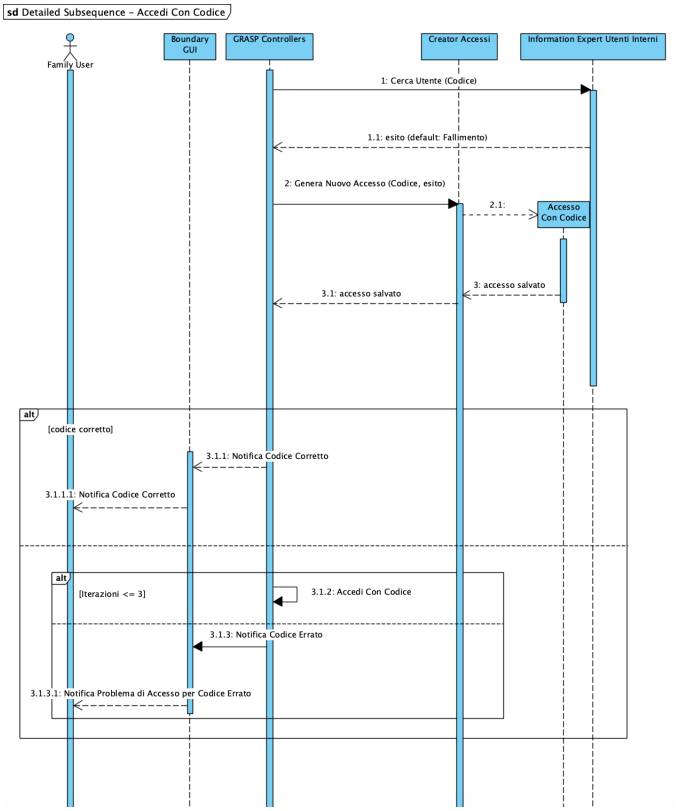


Figura 2.2.3: Sub Sequence Diagram Accedi Con Codice

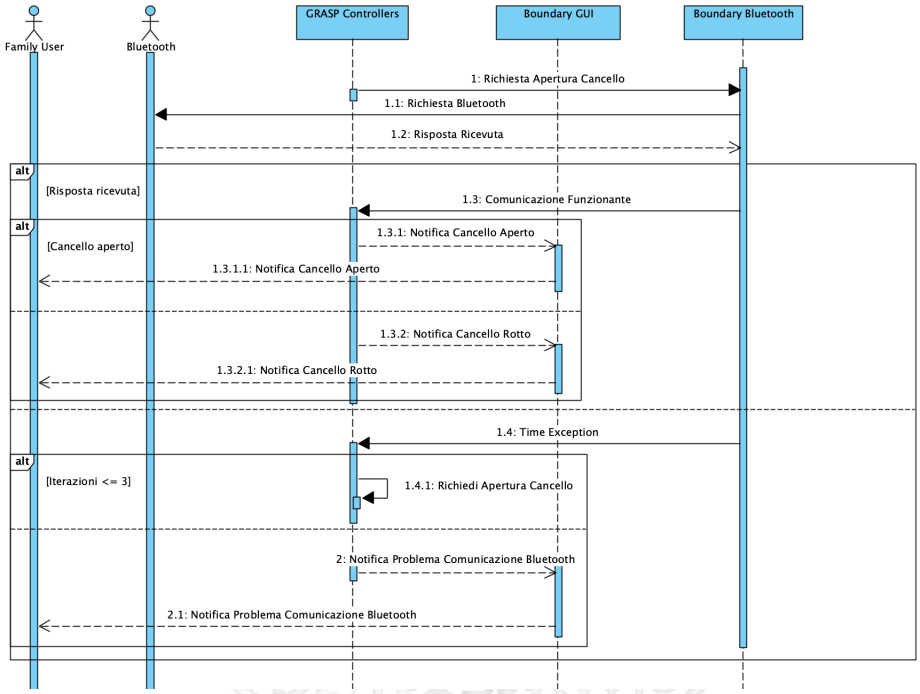


Figura 2.2.4: Sub Sequence Richiedi Apertura

2.2.2 Sequence Diagram per la funzionalità dell'Arduino

Finora abbiamo descritto le funzionalità presenti in app, tuttavia c'è da considerare il fatto che il sistema è costituito non solo da questo, ma anche dal componente Arduino, che permette l'effettiva comunicazione tra l'app e il relay collegato alla porta del condominio. Pertanto, tramite un'ulteriore grafico della dinamica, anche in questo caso un Sequence Diagram, abbiamo rappresentato la funzionalità che si avvia alla ricezione di un determinato segnale Bluetooth da parte dell'app, ricordando che intanto la funzionalità generata nell'Arduino è sempre in attesa di essere attivata dal momento che si trova in una funzione iterativa loop() equivalente al main dei programmi classici in Java.

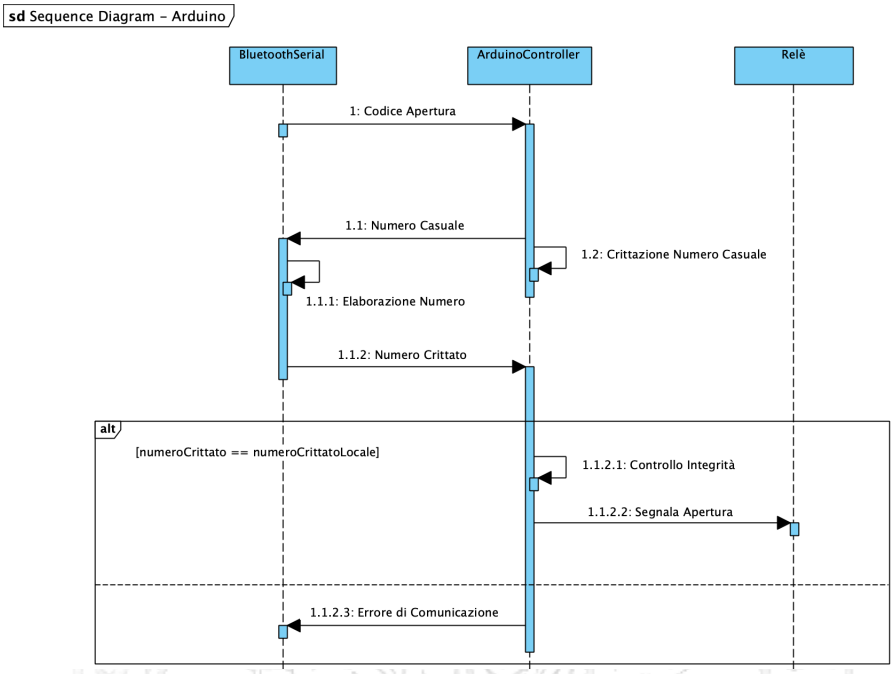


Figura 2.2.5: Sequence Arduino

Capitolo 3

Implementazione

3.1 Component & Connector Diagram

Riportiamo di seguito il diagramma dei Componenti e Connettori che permette di descrivere il sistema con una vista dei componenti. In particolare evidenziamo i due componenti principali quali SmartLock (applicazione sviluppata sul device, iPad e iOS) e Arduino Controller (sistema di interfacciamento con il cancello automatico). I due componenti comunicano tra loro attraverso un'interfaccia Bluetooth.

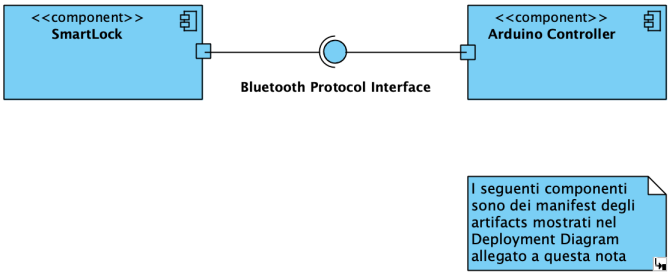


Figura 3.1.1: Package Diagram Smart Lock

3.2 Package Diagram: MVC

Per una descrizione semplificata della struttura del sistema si è scelto un Package Diagram, questo ci permette di mostrare l'applicazione del pattern MVC applicato per lo sviluppo.

E' possibile identificare infatti le 3 componenti principali:

Model contenente i dati del sistema, collegamenti con il database e procedure di accesso al database;

Controller contenente gli oggetti Controller utilizzati per gestire l'interfacciamento tra UI e dati dell'applicazione. In particolare distinguiamo 3 macro controller per la gestione della UI (ViewControllers), comunicazione Bluetooth con il modulo HM-10 (Bluetooth) e Controller di Utilities utilizzati per supporto nella definizione della GUI;

View contiene le viste dell'interfaccia grafica, UIViews ed elementi Custom richiamati nella GUI dell'applicazione e delle funzioni di supporto per la loro creazione.

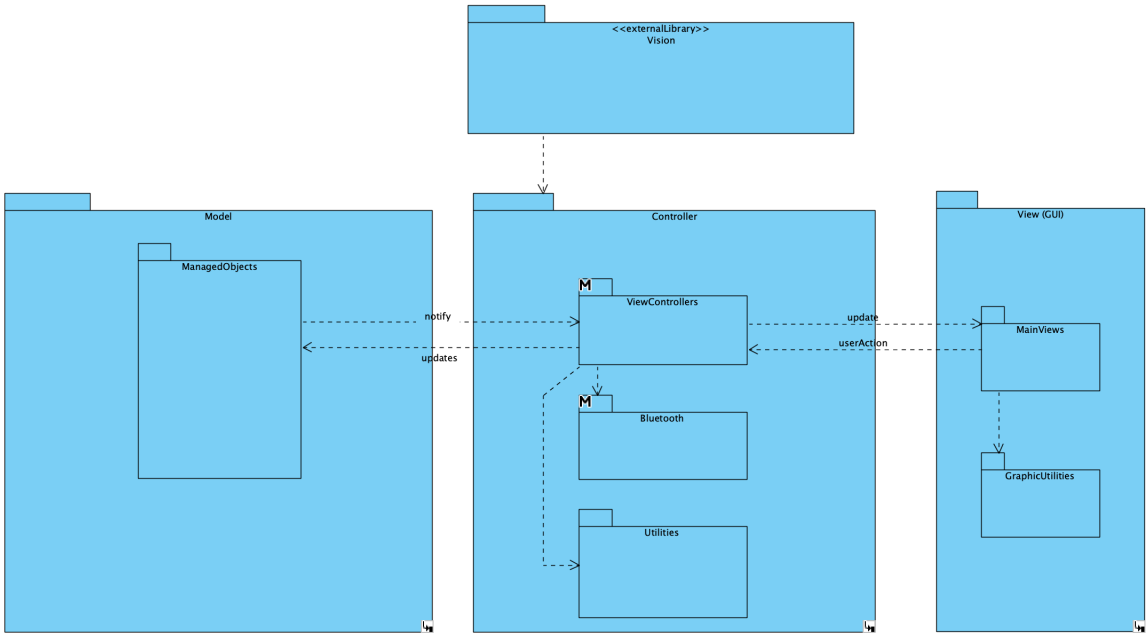


Figura 3.2.1: Package Diagram Smart Lock

Questo diagramma è collegato ai vari Class Diagram che descrivono nel dettaglio Model, View e Controller, mostrati nel paragrafo successivo.

3.3 Dettagli Implementativi: Design Patterns

Per l'implementazione di SmartLock sono stati utilizzati diversi Design Pattern tra cui:

- Singleton
- Creator
- Information Expert
- Observer
- Polimorfismo

3.3.1 Singleton

Il design pattern singleton, presentato dalla Gang of Four, è una soluzione a problemi ricorrenti relativi alla programmazione ad oggetti. In particolare è un Pattern Creazionale utilizzato per ottenere un'unica istanza di un oggetto, fornendo un unico punto di accesso globale. La classe StyleManager implementata tramite Singleton Pattern prevede la scelta di diversi Temi per l'UI dell'applicazione. Le funzioni implementate all'interno di tale classe riguardano la creazione di elementi dell'User Inferface, riutilizzati nei relativi controller per creare gli oggetti dell'interfaccia.

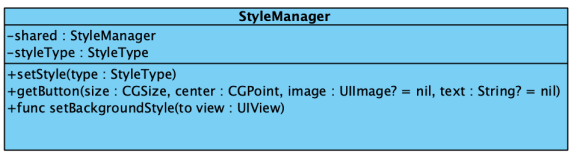


Figura 3.3.1: StyleManager con Singleton Pattern

Tramite la costante shared definiamo un unico punto di accesso alla classe StyleManager, che sarà richiamata nei ViewController AccessScreenViewController e CodeAccessViewController.

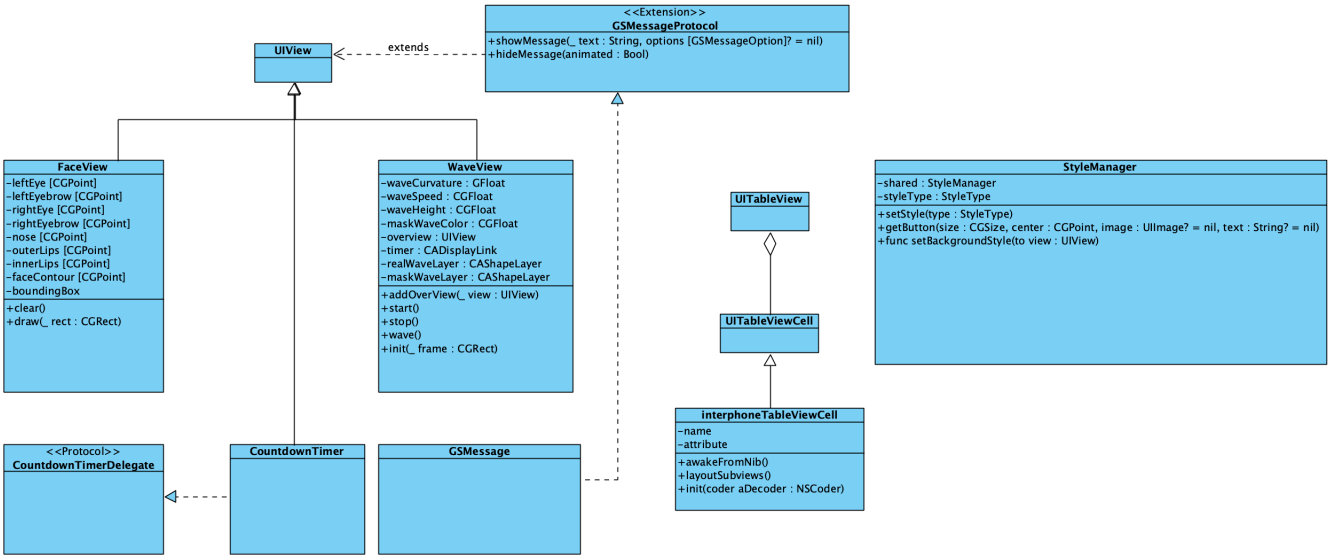


Figura 3.3.2: Class Diagram - Views

3.3.2 Creator

Il Pattern Creator fa parte dei Pattern GRASP i quali forniscono in generale delle linee guida per l'assegnazione delle responsabilità ad oggetti che collaborano all'interno del sistema. In particolare il pattern creator definisce chi crea un oggetto e chi deve crearne un'istanza. La classe DataController è stata implementata utilizzando il pattern Creator per l'accesso alle query del database Core Data. All'interno dell'oggetto DataManager sono contenute le funzionalità di accesso al database (fetch, delete, insert) e tale oggetto è chiamato all'interno delle classi che implementano la logica di business per accedere al metodo richiesto. I dettagli implementativi sono descritti nel Class Diagram relativo al Model del sistema (Figura 3.3.4).

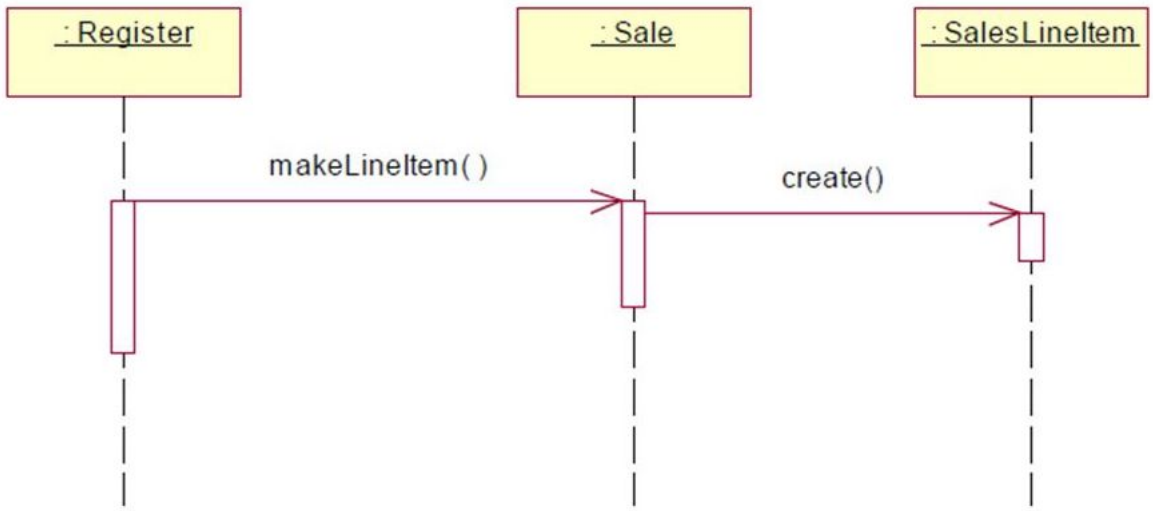


Figura 3.3.3: Pattern Creator

3.3.3 Information Expert

Dato un oggetto il pattern Information Expert consigna di assegnare ad esso solo le responsabilità per cui ha le informazioni necessarie a soddisfare la richiesta. Nell'applicazione è stato utilizzato il pattern IE all'interno del DataController per far riferimento alla gestione delle classi Utente e Accesso del database, questo perchè le informazioni relative a tali oggetti sono contenute all'interno delle richieste gestite dal DataController e vengono risolte solamente da esso, isolando il resto dell'applicazione dalla responsabilità di doverle implementare.

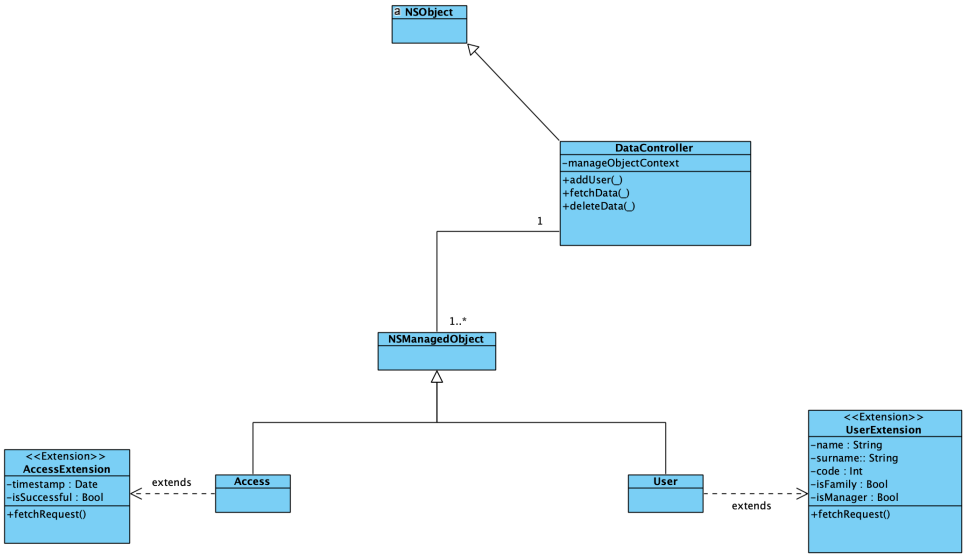


Figura 3.3.4: Class Diagram - Model

3.3.4 Observer

Assegna all'oggetto monitorato il ruolo di registrare al suo interno un riferimento agli altri oggetti che devono essere avvisati dei cambiamenti (osservatori concreti) tramite l'invocazione di un loro metodo. Nell'applicazione si è applicato il pattern Observer per in due casi, infatti AppViewController monitora due eventi:

- "receivedMessage" : per ottimizzare la comunicazione bluetooth si è utilizzata un'architettura ad eventi, si riceve una risposta solo all'invio di un particolare messaggio (apri, chiudi..).
- "deviceConnected": per notificare l'applicazione dell'evento relativo alla presenza del dispositivo Bluetooth a cui collegarsi.

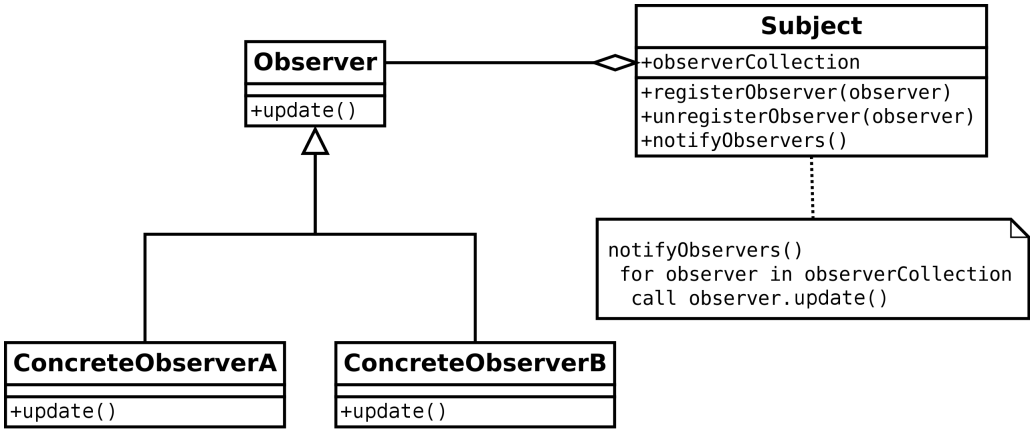


Figura 3.3.5: Observer Pattern

3.3.5 Polimorfismo

E' stato poi utilizzato il Pattern GRASP relativo al Polimorfismo per la realizzazione dei View-Controller dato che questi condividevano una stessa struttura di base relativa all'interfaccia grafica, da estendere in entrambi in base alle esigenze. Questo si può osservare anche nel Class Diagram dei controller dove AccessScreenViewController e CodeAccessViewController ereditano AppViewController e in particolare estendono il metodo setupUserInterface() per estendere la GUI.

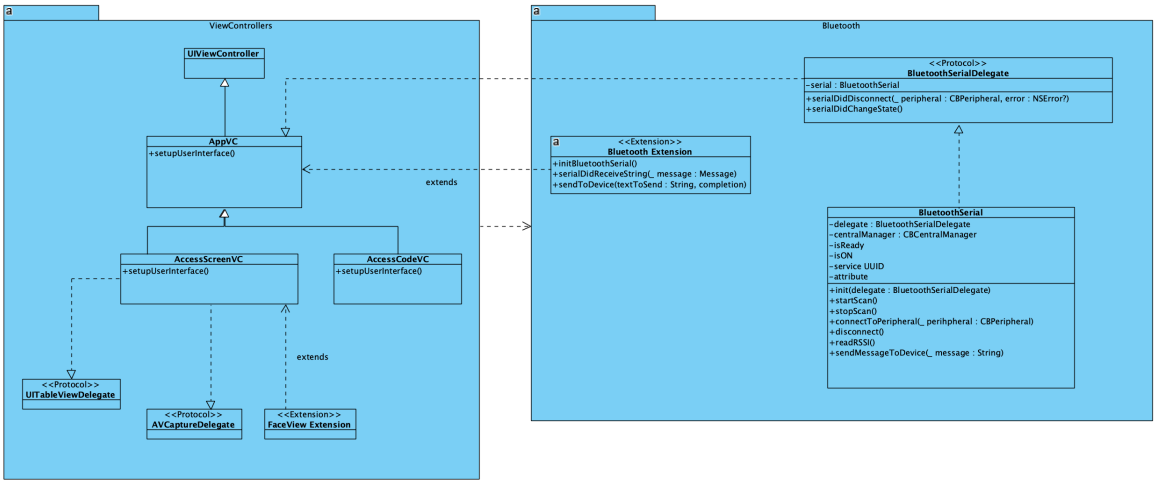


Figura 3.3.6: Class Diagram - Controllers

3.4 Progetto Arduino

Il sottosistema che gestisce l'apertura del cancello automatica è stato realizzato, come descritto nel Capitolo 1, utilizzando un progetto Arduino che prevede l'utilizzo dei seguenti componenti:

- Arduino Uno Board
- Modulo Bluetooth HM-10
- Relay da collegare al Cancello Automatico

I programmi di Arduino vengono caricati tramite l'Arduino IDE. Il codice caricato viene preso in gestione da un **bootstrapper** che però, a differenza degli altri sistemi comuni, rimane in esecuzione per tutto il tempo. Infatti i programmi per Arduino non hanno main, ma hanno una funzione di loop che viene chiamata ripetutamente dal bootstrapper. Dalla definizione di Sistema Operativo ricordiamo che: un sistema operativo è un programma che viene avviato al momento del boot e rimane in esecuzione per tutto il tempo. Notiamo quindi che nel caso dell'Arduino il bootstrapper potrebbe essere considerato una sorta di sistema operativo, sebbene i programmi abbiano accesso diretto all'hardware. Il bootstrapper quindi alcune funzioni di un sistema operativo ma non tutte. Puo' essere assimilato a un "monitor residente" storicamente in uso nei sistemi batch.

Di fatto l'implementazione di questo sottosistema è stata ottenuta applicando il pattern architetturale Sense Compute Control schematizzato nella seguente figura.

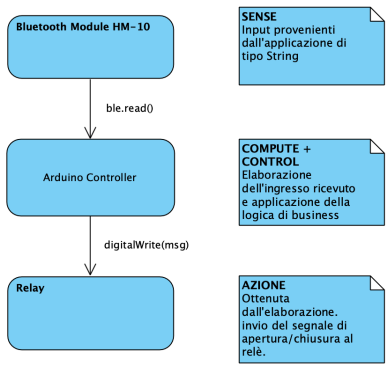


Figura 3.4.1: Arduino Sense Compute Control Pattern Schema

In particolare distinguiamo i tre componenti principali dell'architettura:

Sense — Lettura dei dati provenienti dal modulo Bluetooth HM-10. La lettura avviene interrogando il modulo per verificare che questo sia attivo e in caso positivo viene letto il valore attraverso la chiamata ble.read(). Il dato letto è una sequenza di caratteri che devono essere manipolati.

Compute — Il dato letto dal modulo bluetooth viene modificato ottenendo il messaggio di apertura o chiusura, quindi componendo una stringa con i caratteri ricevuti. Tale stringa può essere di due tipi "apri" oppure "chiudi".

Control — Ai dati opportunamente modificati per essere utilizzati viene applicata la logica di business che nel nostro caso si traduce nell'invio al relè del segnale conforme con i dati letti (apertura o chiusura).

Di seguito riportiamo un Sequence Diagram che descrive il processo di crittazione del messaggio di apertura del cancello tramite scambio di messaggi tra il microcontrollore e l'applicazione, tramite il modulo bluetooth citato.

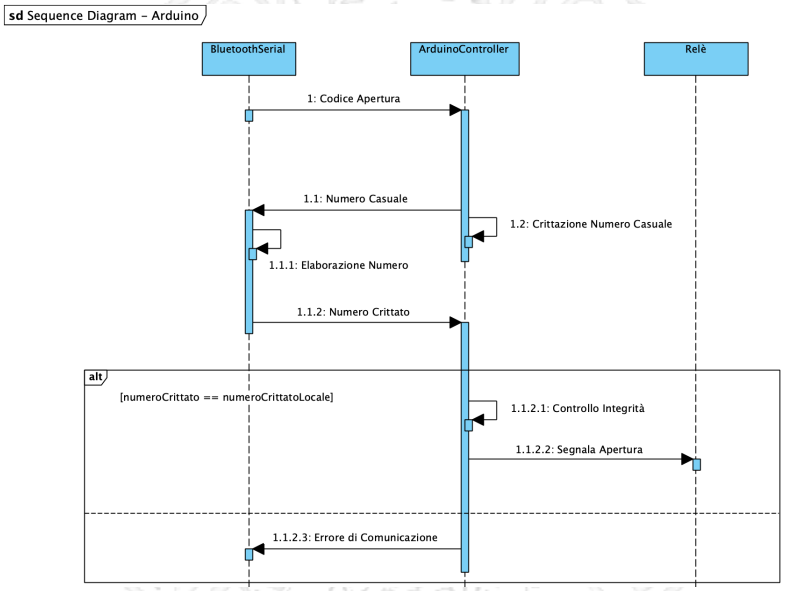


Figura 3.4.2: Sequence Diagram - Comunicazione Bluetooth

3.5 Deployment

Di seguito descriviamo le procedure di deploy dell'applicazione, ossia i metodi con cui avverrà la consegna del prodotto al cliente. Per fare ciò ci si è avvalsi dell'utilizzo di un Deployment Diagram.

3.5.1 Deployment Diagram

Tale diagramma descrive le associazioni tra i nodi componenti il sistema. In paticolare evidenziamo tre tipi di device:

- iPad su cui sarà installata l'applicazione SmartLock
- Arduino sul quale sarà installa la logica di controllo per l'apertura del cancello.
- Relay sul quale troveremo l'aggancio fisico con il cablaggio del cancello.

Le comunicazioni tra i 3 device avverrano secondo un protocollo bluetooth per iPad e Arduino, mentre per il collegamento tra Arduino e Relay saranno utilizzati connettori fisici.

Per quanto riguarda invece i nodi relativi all'ambiente di esecuzione, distinguiamo due tipologie, relative ai device Arduino e iPad, questi infatti supportano due ambienti di esecuzione distinti i quali sono iOS e Bootstrapper, e su tali ambienti di esecuzione saranno eseguiti gli artifacts rappresentati.

L'artefatto SmartLock è relativo all'applicazione per dispositivo iPad, è un eseguibile che manifesta il componente SmartLock, contenente l'implementazione del sistema. L'artefatto prova_Hm10.hex invece manifesta il modulo relativo al controller della logica implementata su Arduino.

Per maggiori dettagli su questi componenti manifestati si può far riferimento al C&C Diagram esposto nella Sezione 3.1 e ai dettagli implementativi che lo seguono.

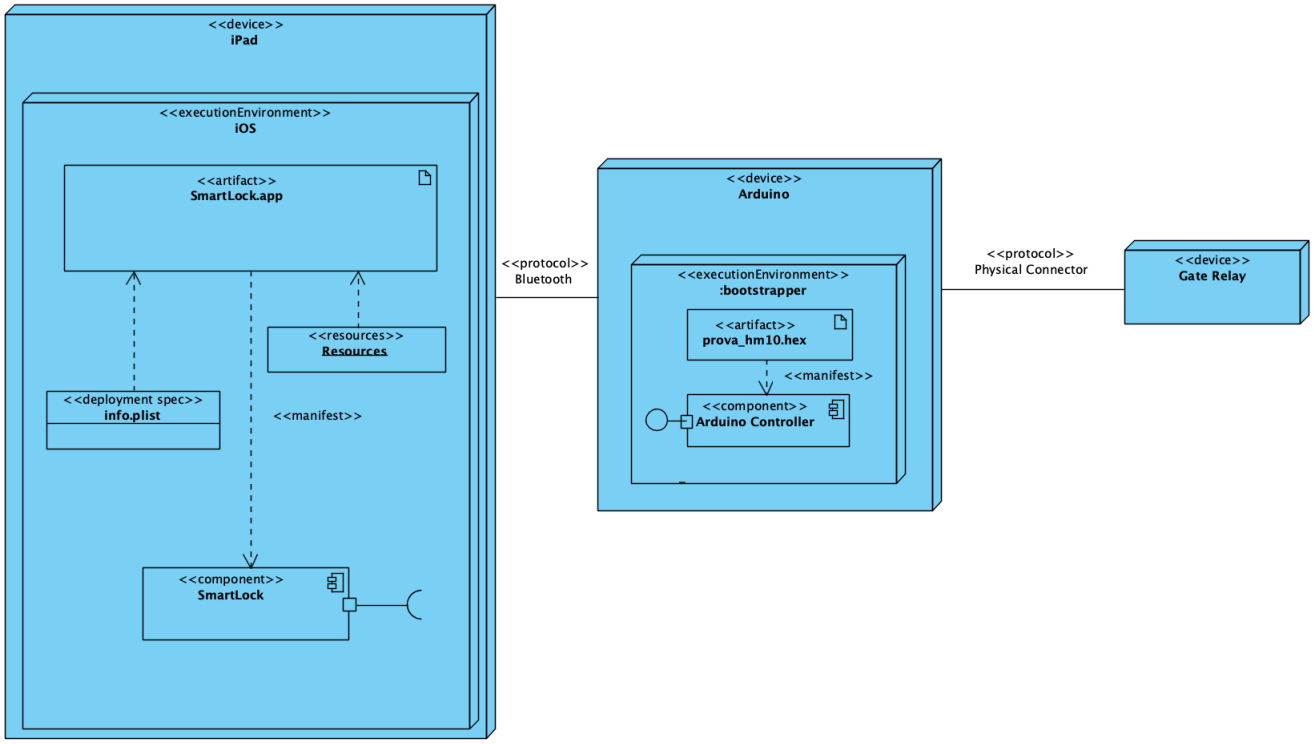


Figura 3.5.1: Deployment Diagram - SmartLock

3.6 Funzionalità Aggiuntiva: Core ML Visual Recognition

Al termine dello sprint finale dello sviluppo, avvenuto in contesto Agile, è stato possibile testare una nuova funizonalità aggiunta al sistema che prevede l'integrazione di un algoritmo di Image Recognition basato sulla libreria di Machine Learning CoreML. Grazie a questa libreria è stato possibile integrare un modello istriuto di machile learning nella nostra applicazione. In particolare abbiamo utilizzato il modello Visual Recognition di Watson.

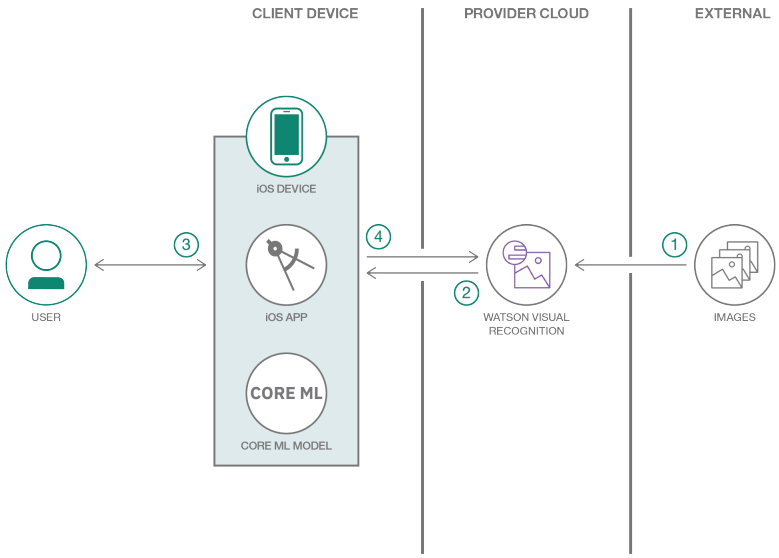


Figura 3.6.1: CoreML Image Recognition

Per istruire il modello sono state utilizzate immagini degli utenti.

Attraverso l'aggiunta di questo modello è stato possibile creare una relazione tra Utente ed Accesso effettuato con successo, prima nascosta dall'utilizzo automatico del FaceID che, per motivi di privacy, non permette di salvare informazioni sul volto riconosciuto. Utilizzando CoreML e Watson l'applicazione è in grado di autenticare l'utente tramite FaceID e consentire l'accesso e contemporaneamente ottenere informazioni sull'accesso da poter aggiungere nel log di sistema.

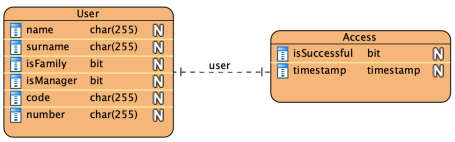


Figura 3.6.2: Database Entiry Relationship Diagram

Capitolo 4

Conclusioni

4.1 Vantaggi dello sviluppo Agile

Per la realizzazione del progetto Smart Lock sono state applicate le best practices di Agile, in un team di sviluppo composto da 4 sviluppatori ed in un tempo totale di 2 mesi. I vantaggi di utilizzare questo tipo di approccio allo sviluppo software sono notevoli, in particolare quelli che sono stati i riscontri maggiori nel nostro Team sono:

- adattamento al cambiamento: il continuo interfacciamento con il cliente per la realizzazione del progetto ha fatto si che applicare le tecniche agile di sviluppo iterativo facilitassero i vari rework avvenuti nelle fasi di analisi e descrizione dell'architettura del progetto. Infatti grazie alla produzione veloce di output da presentare al cliente, il feedback è stato raccolto in modo estremamente veloce e continuamente sfruttato per migliorare il prodotto con ogni iterazione del processo di sviluppo.

- suddivisione del lavoro facilitata dalle iterazioni: l'utilizzo di un approccio iterativo ed evolutivo di tipo Agile ha consentito al team di poter collaborare in modo efficiente e giornaliero per la risoluzione di problemi e la realizzazione delle funzionalità scelte per l'implementazione.

- utilizzo di sistemi di version management distribuiti (GitHub): la scelta di utilizzare la piattaforma GitHub per la condivisione del progetto è stata dettata non solo dalla familiarità dei team members ma anche dai vantaggi di poter lavorare su una clonazione locale del progetto, in modo che ogni membro del team avesse pieno accesso all'intero sistema e potesse testarne l'intero funzionamento prima di eseguire commit e push sulla repository master. Inoltre sistemi come Github (distribuiti) permettono di mantenere sempre una copia di backup del sistema da utilizzare in casi di perdita di informazione.

- utilizzo di sistemi di comunicazione virtuali e face-to-face: nonostante la maggior parte delle decisioni principali del progetto siano state ottenute da meeting face-to-face tra i team members, l'implementazione ha fatto anche molto uso di moderni software di comunicazione e collaborazione disponibili per lo sviluppo software i quali hanno permesso al team di restare in continuo contatto virtuale, aumentando la produttività e le ore di lavoro.

In particolare ogni milestone/meeting con il cliente è stato riportato in uno Spreadsheet condiviso e aggiornato con i task da completare e i task portati a termine e approvati dal cliente.

Data	07/06/2019	10/06/2019	12/06/2019	18/06/2019	26/06/2019	02/07/2019	17/07/2019	23/07/2019
	(Data di inizio)							(Data di consegna)
Task done		PASS:	PASS:	PASS:	PASS:	PASS:	PASS:	
		-documento requisiti	-use case	-sequence diagram	-sequence diagram dettaglio	-c&c	-project diagram	
					-dcb	-package diagram	-implementazione	
					-system domain model		-deploy	
		REDO:	REDO:	REDO:	REDO:	REDO:	REDO:	
			-activity diagram	-communication diagram	-c&c arch. proposal	-deploy		
			-dcb	-dcb				
			-fsm	-system domain model				
			-system domain model					
Task to do	-documento requisiti	-use case	-sequence diagram	-sequence diagram dettaglio	-package diagram	-project diagram	-rifinimenti	
		-activity diagram	-communication diagram	-dcb	-deploy	-implementazione		
		-dcb	-dcb	-c&c arch. proposal	-c&c	-deploy		
		-fsm	-system domain model	-system domain model				
		-system domain model						

Figura 4.1.1: Work Managment Spreadsheet

Tramite Github è possibile anche valutare le performance del team sulla base di statistiche relative a commit e code frequency basati sui collaboratori della repository come mostrato di seguito.

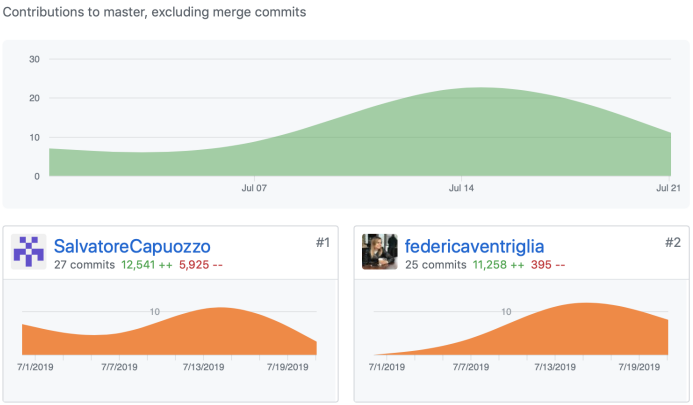


Figura 4.1.2: Github Insights - Commits

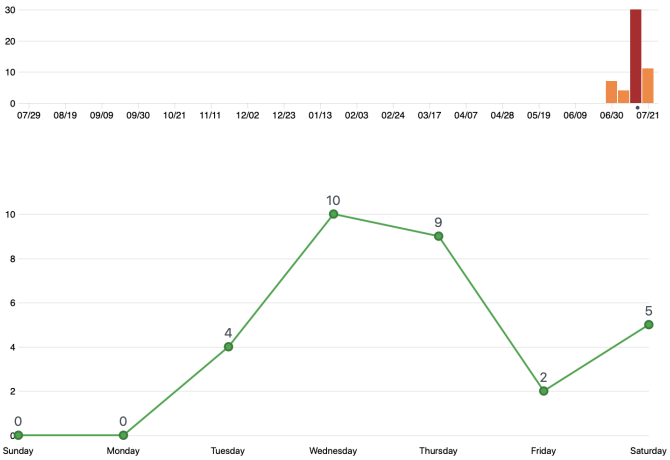


Figura 4.1.3: Github Insights - Weekly Commits

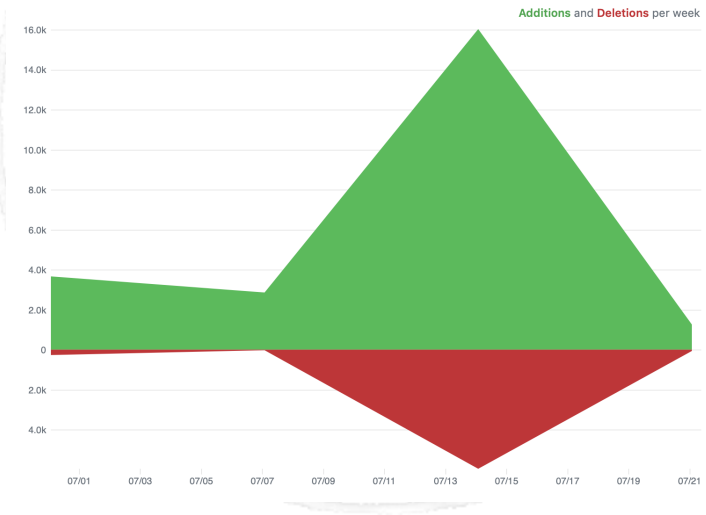


Figura 4.1.4: GitHub Insights - Code Frequency

Tool Utilizzati

- Slack
- Visual Paradigm Enterprise Edition: Teamwork
- GitHub (Private Repository)
- Google Spreadsheet

4.2 Implementazioni Future

Tra le funzionalità ancora da implementare per il progetto SmartLock evidenziamo quella di utilizzo del citofono. La realizzazione di tale caso d'uso si baserà sul sistema pre-esistente nell'appartamento, interfacciandolo con il microcontrollore in modo da poterlo controllare tramite la GUI dell'applicazione, senza dover re-implementare la parte hardware. Questa proposta prevede un'ottimizzazione di sistemi già esistenti, favorendo il **riuso** di tecnologie funzionanti e testate e riducendo i tempi di produzione che si avrebbero dovendo realizzare l'intera architettura ex-novo.

L'idea principale di sviluppo consisterebbe nell'aggiungere un ulteriore componente che si interfaccia direttamente con l'Arduino, ovvero una GSM Shield, un componente che permette all'Arduino di poter effettuare da codice tra le tante cose chiamate a numeri telefonici. Lo scopo principale difatti sarebbe quello di telefonare gli utenti richiesti dall'utente che utilizza l'interfono, così da permettere al condomino di rispondere ed eventualmente aprire la porta con le tecnologie preesistenti, ma anche quello di notificare il manager dell'applicazione e dunque del condominio in caso di numero di tentativi massimo raggiunto nell'inserimento del codice per accedere.

In tale maniera, l'unica cosa sufficiente da compiere a livello di codice è inserire una casistica nella funzionalità dell'Arduino di ricezione di un numero telefonico ed implementare una chiamata di funzione a livello di app che invii tale numero con i protocolli Bluetooth già esistenti ed implementati.

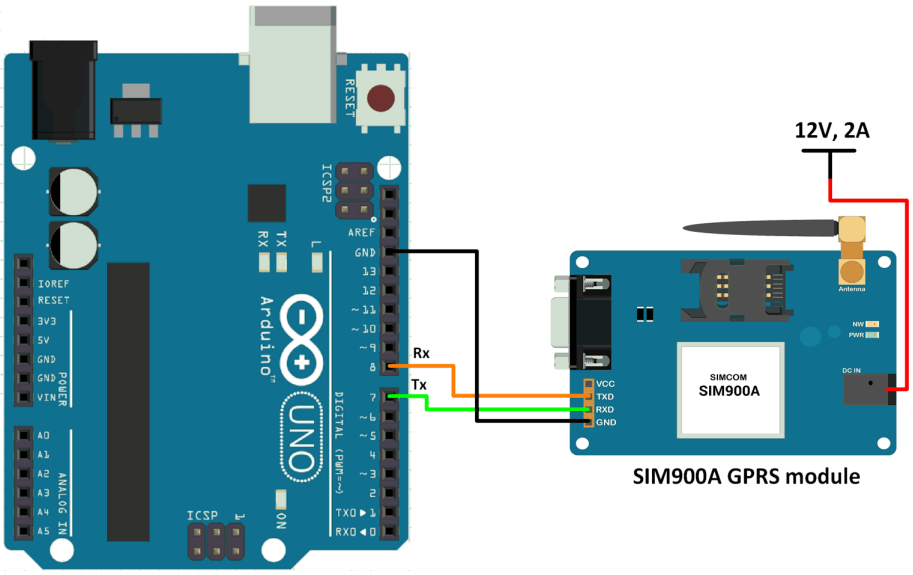


Figura 4.2.1: Esempio di collegamento Arduino - GSM Shield

Ovviamente non è da escludere un futuro sistema Client-Server dove il condomino, oltre ad essere “citofonato” dall’utente, può anche visualizzare in tempo reale l’utente all’esterno mediante la fotocamera del device stesso installato come interfono, che considereremmo il Server, tramite il proprio device personale in cui verrebbe installata una versione Client della stessa.

Le prospettive future possono potenzialmente essere tanto estese quanto risulterà solida la base, pertanto ci siamo garantiti di aver scritto già a monte un codice quanto più riusabile, modulare a livello di classi ed adattabile possibile.