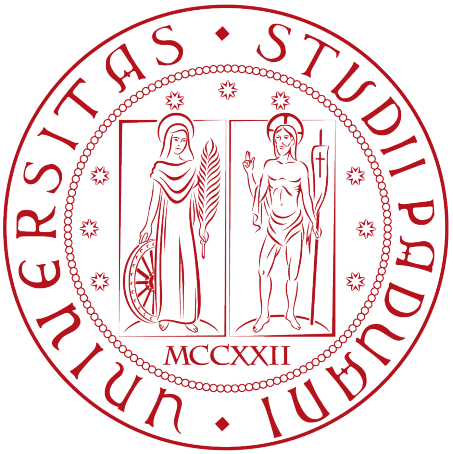
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo di un modulo middleware per la gestione di lettori RFID

Tesi di laurea

Relatore
Prof.Tullio Vardanega

Laureando
Samuele Vanini

Anno Accademico 2020-2021

Samuele Vanini: *Sviluppo di un modulo middleware per la gestione di lettori RFID,*
Tesi di laurea, © Dicembre 2021.

Computer Science is no more about computers than astronomy is about telescopes.

— Edsger Wybe Dijkstra

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 320 ore, dal laureando Samuele Vanini presso l'azienda Aton S.p.A.
L'obbiettivo principale del progetto di stage consisteva nella scrittura di un modulo software attuo a comandare un lettore RFID Kathrein in modo che questo potesse operare in autonomia e tramite l'interfacciamento con una piattaforma middleware proprietaria di Aton, chiamata da ora in poi AMP. Per fare questo, è stato necessario in primo luogo studiare parte della teoria riguardante la comunicazione RFID, per poi applicarla alle varie feature offerte dal lettore in esame. Il passo successivo ha riguardato la progettazione di un'archietettura ad-hoc che andasse ad integrare la libreria proposta dal produttore con le API del middleware AMP.
L'intero documento sarà suddiviso nei quattro capitoli di seguito introdotti:

* Primo capitolo: presenta la realtà aziendale andando a descriverne alcune delle componenti chiave, come le tecnologie utilizzate ed il mercato di riferimento.
* Secondo capitolo: presenta il progetto nel suo insieme, andando a descriverne obiettivi e vincoli.
* Terzo capitolo: presenta l'archietettura sviluppata per il modulo, illustrando le scelte implementative più rilevanti.
* Quarto capitolo: presenta una valutazione retrospettiva delle attività svolte andando ad analizzare il gap formativo tra il bagaglio di conoscenze proposte dal mondo universitario e quello richiesto dal progetto svolto.

Ringraziamenti

Padova, Dicembre 2021 Samuele Vanini

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il contesto aziendale

Nel corso di questo capitolo si presenta l'azienda in cui è stato svolto il progetto di stage, si illustrano i processi interni di questa e la sua posizione sul mercato. Vengono inoltre discusse le tecnologie utilizzate e la sua propensione nei confronti dell'innovazione.

1.1 Storia e ambiti di interesse

L'azienda ospitante prende il nome di Aton S.P.A., società del Trevigiano di importante rilevanza per le sue relazioni commerciali intraprese con aziende leader italiane e numerose multinazionali. Operante in numerosi ambiti, Aton fornisce soluzioni e servizi IT, in linea con i principi Industry 4.0, per le vendite multicanale, le catene di negozi, la grande distribuzione e la gestione degli *asset* nei settori CPG (*Consumer Packaged Goods*), Retail, Fashion ed Energy. Proprio nel contesto di gestione degli *asset* vengono introdotte due delle applicazioni tecnologiche di punta di Aton, le applicazione Internet Of Things e Machine-to-machine.

Data la grande varietà di ambiti in cui l'azienda è impiegata, anche il team che la compone presenta un cospicuo numero di professionisti, all'incirca 150. Aggiungendo a questi anche i dipendenti operanti in una rete di società partecipate, si raggiungono circa le 200 persone. Queste società, specializzate per area geografica e per soluzioni verticali, sono: Blue Mobility e Aton White in Italia, mentre, per Spagna e Portogallo si trova Aton Allspark Iberica. Non è possibile infine non citare il proficuo clima lavorativo presente all'interno dell'azienda, che in aggiunta allo spirito di innovazione e alle numerose possibilità di crescita personale offerta dall'azienda stessa, ha permesso ad Aton di ricevere per il suo terzo anno consecutivo la certificazione "Great Place To Work", che la colloca di diritto nella classifica delle 50 migliori aziende per cui lavorare in Italia.

1.2 Organizzazione dei processi aziendali

L'azienda al fine di gestire al meglio i propri processi interni, segue rigidamente un elenco ben definito di processi. All'interno del reparto RFID questa pratica viene ritenuta di primaria importanza. Questo è dovuto dalla natura stessa del reparto, lavorando su diversi progetti cliente ognuno differente dall'altro, una struttura solida con cui ogni sviluppatore possa orientarsi è l'unica possibilità per distribuire prodotti

affidabili e che soddisfino tutte le aspettative del cliente. I processi cardine citati comprendono:

* **Pianificazione e progettazione**: il reparto si occupa della creazione di flussi di lavoro coinvolgenti la tecnologia RFID sulla base delle esigenze specifiche espresse dal cliente, servendosi di un commerciale e di un tecnico specializzato. Una volta effettuato questo incontro viene creata un'offerta commerciale che viene inviata al cliente in modo che questo possa accettarla o meno. In caso di approvazione si inizia a definire in dettaglio i requisiti tecnici per lo sviluppo del *software* e sui dispositi *hardware* che dovrebbero essere utilizzati se già in possesso del cliente. In seguito si definiscono i dettagli del flusso di lavoro, questo è definito come l'insieme di tutti gli *step* che un *tag RFID* effettuerà e che eventi questi debbano scatenare. Una volta che il flusso è definito si passa all'implementazione, partendo da una piattaforma proprietaria dell'azienda vengono realizzate tutte le personalizzazioni necessarie in un determinato linguaggio di programmazione. Concluso ciò si passa alla fase di *test* e collaudo in cui si verifica che l'applicazione realizzata rispetti tutte le specifiche di progetto, così come collaudi funzionali e di performance. Infine il prodotto *software* che ha superato il collaudo viene installato all'interno dell'infrastruttura di rete del cliente e grazie ad interventi di tecnici specializzati la piattaforma viene resa utilizzabile dagli utenti.

* **Assistenza e manutenzione *software***: il reparto fornisce un servizio di assistenza e supporto al cliente nell'utilizzo e nella manutenzione di tutte le soluzioni sviluppate internamente, o da aziende terze con cui sono stati stretti appositi accordi commerciali. Il cliente nello specifico può inoltrare una richiesta di assistenza tramite l'appertura di un così detto *ticket* o tramite chiamata al centralino aziendale apposito. Il *ticket* verrà quindi preso in carico da un apposito tecnico. Una volta svolto il lavoro necessaro viene poi inviata una notifica di completamento al cliente.

* **Gestione e monitoraggio infrastuttura installata**: il reparto fornice un servizio di monitoraggio sull'infrastruttura del cliente in cui il *software* sviluppato viene installato. Questo comprende diagnostiche sullo stato della rete, sulle performance dei dispositivi *hardware* utilizzati e di tutte le possibili anomalie causate dall'errore di un operatore umano. Grazie a questo è possibile fornire *feedback* tempestivi al cliente in modo da poter intervenire sulla causa del problema, andando ad arginare le perdite economiche dovute ad un arresto della produzione.

A fare da collante ai processi descritti è l'utilizzo di un *way of working* orientato al modello Agile. Questo permette al *Project Manager* di suddividere il lavoro complessivo in molti piccoli incrementi, ognuno di questi formato da diverse fasi, queste sono illustrate in figura 1.1. Questo approccio garantisce che il processo di sviluppo sia facilmente controllabile e misurabile, ciò permette al responsabile di progetto di rispondere in maniera efficace ad imprevisti e modifiche identificate a sviluppo avviato. Olte a ciò, viene data la possibilità al cliente di seguire l'avanzamento dello sviluppo del progetto, incremento dopo incremento, potendo così dare *feedback* su quanto si è già prodotto e su cosa si andrà ad implementare.
Al fine di raggiungere questi obiettivi è di vitale importanza la comunicazione, ritenuta attività fondamentale della gestione di progetto. Il gruppo di lavoro tende ad avere più contatti possibili tra gli stessi membri, sia durante lo *smartworking* che in ufficio, così da rimanere aggiornati in modo costante riguardo il lavoro svolto.

Ad inizio di ogni giornata inoltre si effettua una breve riunione riguardante il progresso dei prodotti in sviluppo e si espongono eventuali difficoltà riscontrate nella giornata precedente. La medesima cosa viene fatta col cliente: una volta a settimana viene fatta una *call* collettiva tra tutti i partecipanti al progetto ed i referenti dell'azienda committente, in cui vengono illustrati tutti i progressi effettuati.

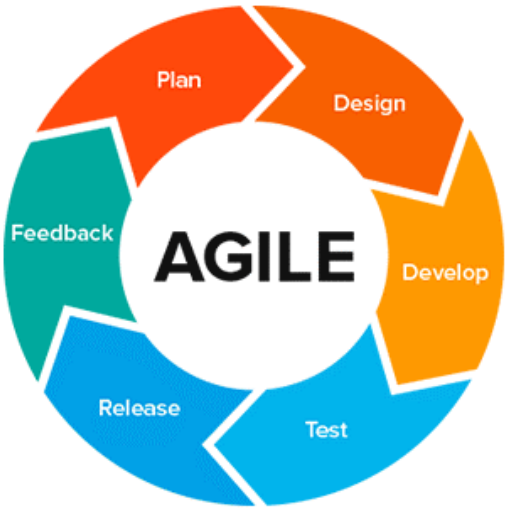


Figura 1.1: Illustrazione del modello Agile - **fonte** urly.it/3g72a

1.3 Tecnologie utilizzate

Come è facile intuire dai numerosi ambiti in cui l'azienda offre soluzioni e servizi, anche il numero di tecnologie utilizzate risulta essere consistente ed in continua evoluzione. Di seguito verrano riportate le tecnologie principali utilizzate all'interno del reparto Radio Frequency Identification con cui ho potuto venire in contatto durante il periodo di stage.

1.3.1 Sistemi operativi

Per quanto concerne i sistemi operativi, vi è una distinzione nel loro utilizzo in base al ruolo che il dipendente ricopre all'interno del reparto. Per gli sviluppatori viene adoperato Windows, in quanto, grazie alla sua robustezza e alla facile amministrazione remota permette la creazione di ambienti di lavoro omogenei per l'intero team. Per quanto riguarda invece la parte più amministrativa, la soluzione adottata è stata quella di dispositivi Apple e quindi aventi MacOS. Per ultimo ma non per importanza troviamo Linux, usato come piattaforma per ospitare le soluzioni sviluppate da Aton presso le sedi dei loro clienti.

1.3.2 Ambienti di sviluppo

Per la scrittura del codice non viene espressa una preferenza specifica, agli sviluppatori viene lasciata la possibilità di utilizzare una delle seguenti *IDE*: Eclipse, Intelij Idea o Visual Studio. La scelta di una di queste ricade in primo luogo sulle esigenze specifiche del progetto su cui lo sviluppatore si trova a lavorare, ed in secondo luogo sulla propria preferenza personale. Ognuno di questi programmi porta con se un numero molto elevanto di funzionalità, queste se combinate con ambienti di lavoro pre-configurati allestiti negli anni e disponibili agli sviluppatori, vanno a formare delle soluzioni pronte che rendono lo spostamento da un progetto ad un altro semplice ed immediato. Oltre ai prodotti già citati, non è possibile non citare la rapida diffusione negli ultimi anni di un altro strumento dalla grande popolarità, l'*editor* Visual Studio Code. Questo programma multipiattaforma nasce con lo scopo di essere un *editor* "tutto fare", di base infatti è provvisto solo di strumenti avanzati per la scrittura e la formattazione del testo. Grazie però ad un vasto e ricco market di estensioni, è possibile una sua profonda e completa personalizzazione integrando quindi il supporto a tutti i linguaggi di programmazione più rilevanti presenti sul mercato. Questo porta alla creazione di un ambiente dinamico che possa rispondere a tutte le esigenze di ogni sviluppatore.

1.3.3 Sistemi di versionamento

Dato il numeroso team di lavoro risulta indispensabile fare affidamento su un sistema di versionamento, questo tiene traccia del flusso di lavoro e di tutte le modifiche che vengono apportate al codice sorgente. Storicamente, all'interno dell'azienda, il punto di riferimento per questo compito è stato il *software open source* Subversion. Questo però sta lentamente venendo abbandonato per effettuare una massiccia migrazione verso Git, altro *software* per il controllo di versione del codice distribuito che è diventanto di fatto uno standard per il mercato odierno.

1.3.4 Suite per la produttivita e comunicazione

Per quanto riguarda l'organizzazione e la comunicazione viene fatto un ampio uso dei servizi proposti da Microsoft. Di questi, vengono usati principalmente:

* **Outlook**: servizio di posta elettronica che comprende anche un *client* desktop ed un calendario condivisibile. Viene utilizzato sia internamente tra i dipendenti che coi clienti.
* **Onedrive**: servizio *web* per la memorizzazione, sincronizzazione e condivisione di file online.
* **Office Suite**: pacchetto di applicazioni *software* che presenta numerose funzionalità, come la creazione e modifica di *file* di testo, fogli di calcolo, presentazioni e reportistica varia.
* **Teams**: piattaforma di comunicazione e collaborazione unificata che combina chat di lavoro persistente, teleconferenza, condivisione di contenuti e integrazione delle applicazioni esterne.

1.3.5 Linguaggi di sviluppo

Avendo l'esigenza di sviluppare progetti e soluzioni *software* che si interfaccino frequentemente con dispositivi *hardware* e che possano operare andando ad interfacciarsi con

un importante numero di applicativi scritti da terzi, viene fatto uso di diversi linguaggi di programmazione:

* **Java**: basato su Java Virtual Machine(JVM), ormai da svariati anni è il più utilizzato per il suo vastissimo numero di librerie. Questo, combinato alla possibilità di eseguire il codice in modo parallelo e concorrente lo rende uno strumento solido ed affidabile. Un altro vantaggio di tale linguaggio riguarda l'esecuzione del codice stesso, questo infatti risulta indipendente dall'*hardware* della macchina in quanto virtualizzato dalla piattaforma stessa. In tal modo il codice è reso portabile verso altre piattaforme con *hardware* diversi tra loro.
* **C#**: linguaggio di programmazione multiparadigma, viene sviluppato da Microsoft e presenta numerose somiglianze con Java. Negli ultimi anni ha avuto una forte crescita dovuta ad alcune sue caratteristiche uniche come la possibilità di interfacciare nativamente codice scritto in linguaggi compilati come C e C++. Questo, unito ad un solido ecosistema di librerie e dalle sue notevoli performance, ha spinto numerosi produttori ad adottarlo come strumento per interfacciarsi con i propri dispositivi *hardware*.
* **Javascript**: linguaggio di programmazione orientato agli oggetti tramite prototipi e agli eventi. Grazie alla sua grande popolarità ed alla produttività nella scrittura del codice viene utilizzato per prototipizzazione o per specifici ambiti.

1.3.6 Framework

Il principale e sicuramente più utilizzato è OSGi Knopflerfish, un *framework open-source* che si propone di implementare un modello a componenti completo e dinamico per la piattaforma Java. I concetti fondamentali dietro questa tecnologia sono essenzialmente 3:

* Definizione del concetto di modulo (*bundle*)
* Gestione automatica delle dipendenze
* Gestione del ciclo di vita del codice (configurazione e distribuzione dinamica)

Grazie a ciò viene reso possibile creare un sistema dinamico, che consente l'installazione, l'avvio, lo stop e la rimozione dei moduli a *runtime*, senza quindi necessitare di riavvii. Un esempio di come il codice venga organizzato tramite il concetto di modulo è visibile nella figura 1.2. Non a caso quindi è stato adottato per la creazione della principale piattaforma utilizzata dal reparto RFID, AMP, che offrendo la possibilità di creare nuovi moduli si rende una perfetta base di partenza per progetti orientati alla personalizzazione per il singolo cliente.

1.4 Propensione all'innovazione

L'azienda da molti anni ha deciso di mettere la formazione continua come uno dei suoi pilastri principali, operando in mercati dinamici e con tecnologie in continua evoluzione risulta quindi indispensabile continuare ad ampliare le proprie conoscenze, in modo da non essere superati dai vari *competitor*. A tal fine vengono messi a disposizione di ogni dipendente numerosi corsi di formazione, questi spaziano da certificazioni per tecnologie specifiche, corsi sulla gestione di progetto fino ad arrivare a corsi di lingue

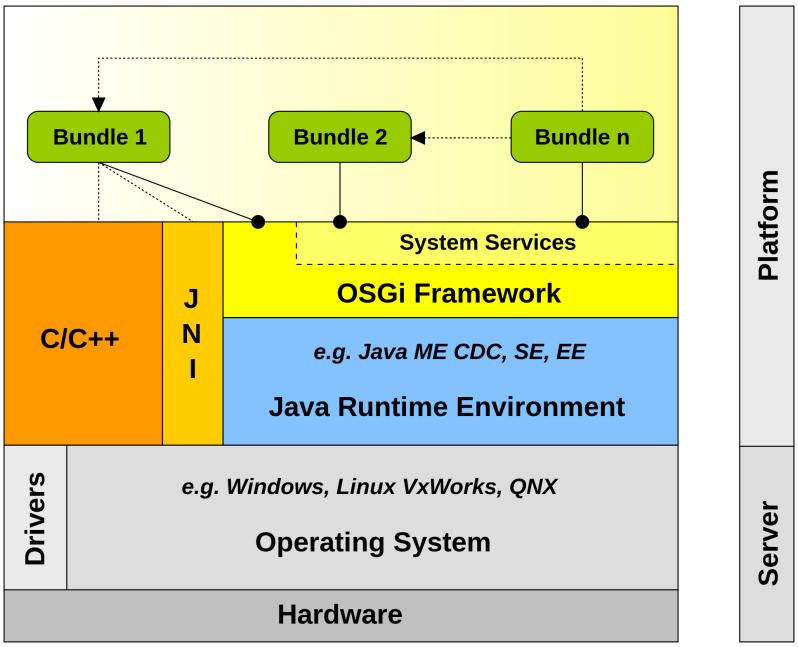


Figura 1.2: Architettura con OSGi - **fonte** urly.it/3g72y

straniere.
Il reparto RFID non fa eccezione, viene richiesto infatti a tutti gli sviluppatori di seguire percorsi formativi su un ampia scelta di tecnolgie, identificate come cruciali per il continuo rinnovamento ed espansione dell'azienda. Questi comprendono ad esempio corsi su tecnoglie riguardanti la persistenza dei dati fine all'utilizzo di *framework* di nuova generazione. Oltre a questo Aton ha dimostrato negli anni una forte connessione con l'ambiente universitario e quindi di conseguenza con la ricerca che si svolge in esso. Uno degli esempi più significati è sicuramente la collaborazione avvenuta con l'Università di Torino con cui si sono svolte delle ricerche riguardanti proprio la tecnologia RFID. Queste hanno portato alla creazione delle basi della piattaforma proprietaria per tale tecnologia, AMP, utilizzata ancora oggi come base di partenza per tutti i progetti cliente sviluppati.
Correlata alla connessione con gli ambienti universitari appena citati, troviamo l'occupazione di un dipendente Aton, che oltre al suo lavoro in azienda svolge l'attività di professore nel corso "Altri paradigmi di programmazione" all'interno della laurea in Informatica presso l'Università di Padova.

Capitolo 2

Il progetto Kathrein

Nel corso di questo capitolo si presenta il progetto di stage esponendo come queste attività siano interpretate dall'azienda. Ne vengono poi descritti gli obiettivi, i vincoli ed il futuro al termine della mia partecipazione.

2.1 Descrizione

Il progetto di stage propostomi da Aton S.P.A., mediante la figura del mio *tutor* interno, è riconducibile alla visione d'insieme che l'azienda possiede nei confronti delle collaborazioni con le università. Queste opportunità non sono nuove all'azienda, nel corso degli anni sono state svolte numerose collaborazioni con diversi atenei che hanno portato alla realizzazione di prodotti ancora oggi utilizzati.
Tali percorsi vengono interpretati come mezzo per ampliare il bagaglio di conoscenze proprio dell'azienda riguardo a nuove tecnologie. Questo scopo viene perseguito tramite l'implementazione di *software* che possa contribuire alle soluzioni utilizzate nei vari progetti cliente. Oltre a questo, gli stage proprosti si pongono come secondo fine la pubblicizzazione dell'azienda cercando di attirare nuovi sviluppatori che possano far crescere il team già esistente.
Nel mio caso specifico lo scopo del progetto era quello di studiare l'interazione con dei nuovi lettori RFID per poi applicare quanto imparato tramite lo sviluppo di un prodotto *software* apposito. Tale prodotto era quindi finalizzato a diventare un nuovo componente utilizzabile dai tecnici aziendali per la creazione di flussi di lavoro adoperanti i lettori analizzati, accrescendo di conseguenza le soluzioni offerte dall'azienda nei vari progetti cliente. Un esempio concreto della posizione di questo prodotto, all'interno di un sistema RFID classico, è rappresentato nei passi 5 e 6, riportati nella figura 2.1.

2.2 Obiettivi

2.2.1 Obiettivi aziendali

L'azienda ha posto dei macro obiettivi principali per quanto riguarda le funzionalità dell'applicativo, sia a livello utente che a livello software. Questi sono stati esplicitati nei primi giorni in cui mi è stato presentato il progretto in dettaglio dal mio *tutor* interno.

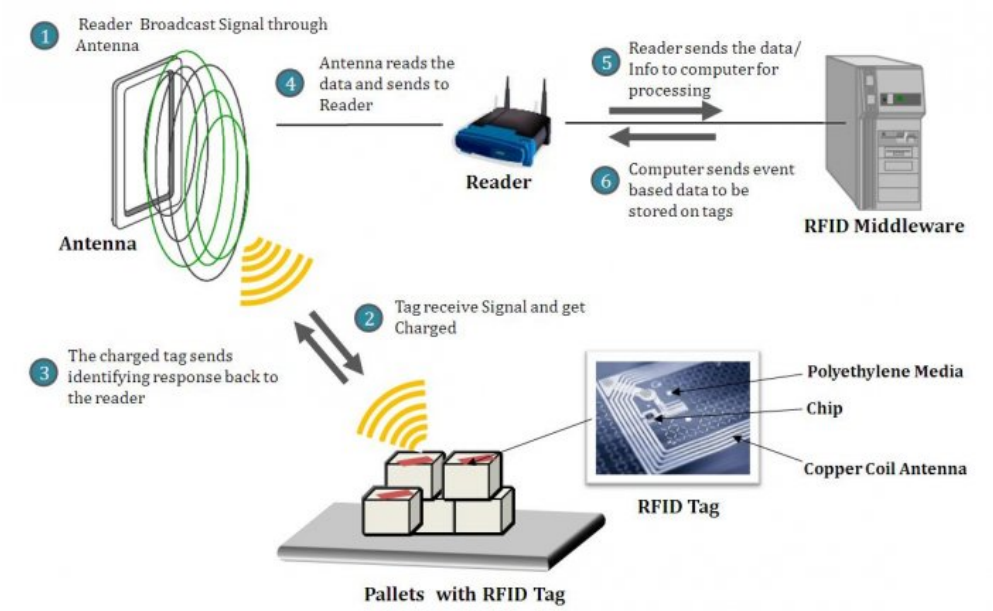


Figura 2.1: Schema di un sistema RFID - **fonte** urly.it/3gc8k

* L'applicativo avrebbe dovuto interfacciarsi con tutte le generazioni di lettori Kathrein;
* L'applicativo avrebbe dovuto funzionare in maniera autonoma, senza quindi la presenza di un operatore atto ad operare i *device*;
* L'applicativo avrebbe dovuto offrire le stesse funzionalità all'utente degli altri applicativi, dedicati ad altri marchi di lettori;
* L'applicativo avrebbe dovuto supportare la lettura di numerosi *tag* simultaneamente, nell'ordine di qualche centinaio;
* L'applicativo avrebbe dovuto essere fruibile direttamente all'interno della piattaforma AMP;
* Le anomalie dovute alla rete dovevano essere gestite e notificate se non risolvibili;
* L'applicativo avrebbe dovuto comandare altri dispositivi IoT come semafori o *buzzer* il cui operato dipendesse da quello del lettore.

2.2.2 Obiettivi personali

Tramite l'Università ho avuto accesso all'evento **Stage-IT**, offerto da AssIndustria Veneto-Centro, in cui vengono messi in diretto collegamento aziende in collaborazione con il Dipartimento di Padova e gli studenti laureandi, in modo da poter proporre stage pre-laurea. Durante l'evento numerosi progetti hanno attratto la mia attenzione, l'offerta proposta da Aton S.P.A. però è riuscita a catturare a pieno il mio interesse. Causa di questo è stata la possibilità di lavorare in un contesto a me nuovo ma in fortissima crescita nel mercato odierno. In particolare, l'utilizzo di codice nativo tramite Java, all'interno di un contesto in cui l'ottimizzazione ricopre un ruolo cruciale, mi ha

fin da subito affascianto. Questo mi ha permesso di applicare quanto imparato durante gli anni studio, garantendomi allo stesso tempo una grande occasione per apprendere nuove tecnologie. Sapevo, inoltre, che sarei stato inserito in un ambiente di lavoro giovane, dinamico e professionale dove la collaborazione viene ritenuta uno dei pilastri fondamentali per la buona riuscita di ogni progetto.

Gli obiettivi che ambivo a raggiungere a livello personale riguardavano principalmente la mia formazione riguardo tecnologie a me estranee. In particolare, ambivo a poter apprendere il funzionamento di un sistema RFID, andando poi ad applicare quanto imparato nello sviluppo di prodotto capace di essere resiliente ai guasti e che si interfacciasse con un *device* fisico. In aggiunta a questo, era mia intenzione osservare ed apprendere il più possibile le metodologie di lavoro utilizzate e le *best practicies*, consolidate e utilizzate da anni dal gruppo di *tech lead* presente all'interno dell'azienda. Questo per poter conoscere le strategie che, partendo da un'idea, portano alla concretizzazione di un prodotto finito e funzionante.

2.3 Vincoli

2.3.1 Vincoli tecnologici

Essendo la soluzione basata sulla piattaforma AMP e comprendente delle librerie native proprie del lettore, i vincoli tecnologici hanno fatto si che venissero utilizzate specifiche tecnologie:

* **Java 8**: linguaggio di programmazione utilizzato per lo sviluppo;
* **OSGi Knopflerfish**: *framework* utilizzato per la realizzazione di AMP, mette a disposizione la possibilità di creare moduli autonomi;
* **JNA**: libreria per accedere a codice nativo.

2.3.2 Vincoli temporali

Il tempo determinato dallo stage, ovvero 320 ore, è stato il vincolo temporale più rilevante, in quanto veniva incluso in esso la mia formazione autonoma riguardo: la teoria sulla tecnologia RFID, le tecnolgie utilizzate e le librerie offerte dal produttore dei lettori. In aggiunta a questo, le revisioni di progetto settimanali, di cui si parla in maniera approfondita nella sezione 3.1.2, richiedono un puntuale sviluppo di ogni incremento in modo da facilitare l'organizzazione di tali incontri.

2.4 Futuro del progetto

In connessione a quanto riportato nella sezione 2.1, è facile vedere come il prodotto sviluppato fosse interpretato come uno strumento utilizzabile in futuri progetti cliente. La visione complessiva dell'azienda, rispetto al prodotto, risultava però essere più ampia. L'applicativo non avrebbe dovuto essere fine a se stesso, ma rappresentare una base di studio per qualsiasi altro programmatore intento nello sviluppo di una soluzione basata sui lettori Kathrein o, più in generale, necessitante di interfacciarsi con del codice nativo.

Capitolo 3

Resoconto del progetto

Nel corso di questo capitolo si presenta un resoconto del progetto, descrivendone gli aspetti nel dettaglio. Si illustrano le metodologie di lavoro utilizzato e le scelte dietro lo sviluppo delle componenti ritenute più rilevanti o critiche.

3.1 Pianificazione delle attività

In accordo con il mio *tutor* interno abbbiamo stilato un piano di lavoro suddividendo gli argomenti da affrontare di settimana in settimana, indicando poi per ciascuna settimana le ore di lavoro e le attività che avrei svolto.
Tale piano viene riportato di seguito nella tabella 3.1.

3.1.1 Comunicazioni

Durante tutta la durata le progetto ho tenuto una comunicazione attiva con tutte le figure aziendali interessate al prodotto da me sviluppato.
Ho tenuto giornalmente delle riunioni con il mio *tutor* interno, della durata di circa 15 minuti in cui esponevo lo stato di avanzamento, riportando quanto svolgo durante nell'ultima giornata trascorsa, i problemi sorti e le mie proposte sullo sviluppo delle componenti richieste. Ho partecipato regolarmente alle riunioni di reparto, svolte ogni lunedì, della durata di circa un'ora in ho avuto la possibilità di rendere partecipi tutti gli sviluppatori presenti del lavoro da me svolto. Oltre agli incontri precedentemente citati, ho tenuto periodicamente delle revisioni del codice con uno dei dipendenti appartenente al gruppo di *tech lead* per discutere le scelte progettuali ed implementative da me fatte, questo verrà approfondito nella sezione 3.1.2.

3.1.2 Revisioni di progetto

La progettazione e lo sviluppo del progetto sono state intervallate da periodici controlli sulla qualità di quanto da me prodotto. Questi controlli erano effettuati da me ed un sviluppatore esperto facente parte del gruppo di *tech lead* presente in azienda. Durante questi incrontri tutto il codice sorgente è stato revisionato, commendandone le caratteristiche e le scelte fatte. Al termine di questa fase, il prodotto veniva testato tramite la simulazione di contesti reali in cui potevo effettuare test di carico e controllare l'assenza di anomalie.

Ore settimanali	Attività svolte
Prima settimana - 40 ore	Presentazione del progetto e degli attori coinvolti, predisposizione dell'ambiente di lavoro, formazione sugli apparati *hardware* adottati e inizio studio autono sulle tecnologie adottate.
Seconda settimana - 40 ore	Studio autonomo su: tecnolgie e *framework* adottati, teoria RFID, librerie offerte dal produttore degli apparati utilizzati
Terza settimana - 40 ore	Progettazione del modulo software ed implementazione delle componenti riguardanti letture tag e invio dati alla piattaforma AMP
Quarta settimana - 40 ore	Implementazione delle componenti riguardanti *sessions* e *singulation*
Quinta settimana - 40 ore	Implementazione delle componenti riguardanti *transit time* e *dwell time*
Sesta settimana - 40 ore	Implementanzione delle componenti riguardanti *kray protocol* e gestione porte *GPIO*
Settima settimana - 40 ore	Implementazione della componente riguardante il controllo del modulo dalla piattaforma AMP
Ottava settimana - 40 ore	Test approfonditi del prodotto, stesura della documentazione relativa a quanto implementato, stesura di un documento riassuntivo sulla tecnologia JNA e sulle librerie offerte dal produttore.

Tabella 3.1: Piano di lavoro

3.2 Analisi del progetto

Nonostante abbia sviluppato l'intero applicativo in autonomia, ho seguito tutti i passi adoperati dal gruppo di sviluppo per la realizzazione di un progetto interno e quindi non finalizzato ad un cliente specifico. Questi sono elencati di seguito:

* **Incontro con gli *stakeholder***: in questa fase viene fatto l'incontro tra il gruppo che andrà a sviluppare il prodotto e tutti gli attori interassati al proggetto. In questa sede vengono discusse varie idea, specificando i requisiti e gli obiettivi che il prodotto dovrà raggiungere;

* ***Scouting* delle tecnologie**: dopo aver acquisito le informazioni utili relative al progetto, il gruppo di lavoro inizia a ricercare le tecnologie che possono risultare più adatte a soddisfare le richieste poste dal committente, vagliando anche quelle più sperimentali ed innovative;

* **Studio di fattibilità**: successivamente viene fatto lo studio di fattibilità, in modo tale da poter capire se il prodotto si possa sviluppare concretamente. Viene svolta anche un'analisi dei rischi e dei costi, al fine di poter presentare, a tutti gli interessati al progetto, una chiara panoramica di quello che ci si aspetta essere il risultato finale. In questa fase il progetto può essere scartato se considerato non abbastanza vantaggioso;

* **Sviluppo**: se gli attori coinvolti accettano quanto riportato dallo studio di fattibilità, allora si inizia con la fase vera e propria dello sviluppo del prodotto.

Il lavoro viene suddiviso in tanti incrementi, come descritto nel modello Agile, così da garantirne un controllo completo. Allo stesso tempo, una suddivisione incrementale permette una facile esposizione dei risultati raggiunti al committente.

* ***Testing***: in questa fase il prodotto viene testato mediante specifici test, sia manuali che automatici, per verificarne le funzionalità e assicurarsi che il prodotto abbia soddisfatto le richieste del committente.

3.2.1 Funzionalità principali

Dopo aver impostato gli incrementi, in accordo con il mio *tutor* interno, ho redatto una lista di macro funzionalità che il prodotto avrebbe dovuto avere:

* **Funzionamento autonomo**: dovendo integrare questi lettori in contesti industriali in cui il carico di lavoro risulta essere particolarmente elevato, era necessario che i dispositivi potessero lavorare senza la presenza di un operatore umano, che gli azionasse e controllasse;

* **Tolleranza ai guasti**: operando in contesti in cui agenti fisici possono facilmente interrompere la connessione ai dispositivi, era necesserio identificare e gestire tali anomalie;

* **Configurazione a "caldo"**: necessitando di cambi di configurazione durante il suo utilizzo, il prodotto doveva poter modificare alcuni parametri di configurazione senza la necessità di riavviarsi o di interrompere il flusso di lavoro in cui inserito;

* **Integrazione con AMP**: dovendo inviare dati alla piattaforma utilizzata per la realizzazione dei flussi di lavoro e ricevere comandi da essa, era necessario implementare una strategia di comunicazione con tale *software*;

3.2.2 Obiettivi e requisiti chiave

Obiettivi

Durante la fase di stesura del Piano di Lavoro, io ed il mio *tutor* interno abbiamo prefissato degli obiettivi da raggiungere entro la fine dello stage. Di seguito sono riportati mediante una nomenclatura che li identifica univocamente:

[classificazione][numero incrementale]

dove il campo classificazione può avere uno dei seguenti valori:

* **O**: obiettivi obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;

* **D**: obiettivi desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;

* **F**: obiettivi facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Vengono di seguito riportati in forma tabellare.

Obiettivi	Descrizione
O01	Dimostrazione di una piena conoscenza della teoria legata alla tecnologia RFID
O02	Realizzazione di un modulo software pienamente funzionante che rispetti le caratteristiche desiderate dagli stakeholder
O03	Realizzazione della documentazione su quanto implementato e sulle tecnologie utilizzate
D01	Realizzazione di uno studio di fattibilità sul *porting* del modulo realizzato all'interno dei lettori RFID
F01	Esposizione tramite API dello stato fisico dei lettori

Tabella 3.2: Piano di lavoro

Analisi dei requisiti

Al fine di raggiungere gli obiettivi elencati, io ed il mio *tutor* interno, ci siamo concentrati nella raccorta di requisiti. Questi sono identificati da una nomenclatura univoca, cosí composta:

R[classificazione][tipologia][numero incrementale]

Per **classificazione** del requisito si intende:

* **O**: obbligatoro;
* **D**: desiderabile;
* **F**: facoltativo.

Per **tipologia** del requisito si intende:

* **F**: funzionale;
* **Q**: qualitativo;
* **P**: prestazionale;
* **V**: vincolo.

Nella tabella 3.3 sono riportati solo i requisiti funzionali estratti dai Casi d'Uso più significativi presentati in sezione 3.2.3. Per quanto riguarda i requisiti di vincolo,

Requisito	Descrizione	Fonte
ROF1	L'utente deve poter visionare la configurazione del lettore	UC1
ROF2	L'utente deve poter modificare la configurazione del lettore	UC2
ROF3	L'applicativo deve poter tornare in funzione in maniera autonoma a seguito di errori dovuti ala rete	UC5
ROF4	L'utente deve poter modificare lo stato delle porte GPIO connesse al lettore	UC3
ROF5	Il modulo deve inviare i dati relativi ai tag letti alla piattaforma AMP	UC4

Tabella 3.3: Tabella di tracciamento dei requisiti funzionali

consultare la tabella 3.4. I requisiti qualitativi rilevati sono elencati nella tabella 3.5: Infine, i requisiti prestazionali rilevati sono elencati nella tabella 3.6:

Requisito	Descrizione	Fonte
ROV1	Utilizzo di Java 8 per la codifica del modulo	Interna
ROV2	Utilizzo della libreria JNA per interfacciarsi con librerie native	Interna

Tabella 3.4: Tabella di tracciamento dei requisiti di vincolo

Requisito	Descrizione	Fonte
ROQ1	Il codice sorgente dev'essere versionato tramite il *software* Git	Interna
ROQ2	Dev'essere realizzato un documento riguardante le tecnologie, le scelte implementative e progettuali con relative motivazioni	Interna

Tabella 3.5: Tabella di tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
ROP1	Il modulo deve supportare la lettura di centinaia di tag simultaneamente	Interna

Tabella 3.6: Tabella di tracciamento dei requisiti qualitativi

3.2.3 Casi d'uso

UC1: Visualizzazione configurazione del lettore

Attori Principali: Utente.

Precondizioni: L'utente è entrato all'interno di AMP ed ha lanciato il comando per visualizzare la configurazione del lettore.

Descrizione: AMP mette a disposizione una finestra relativa al modulo in cui lanciare comandi verso questo tra cui visualizzazione e modifica della configurazione del lettore.

Postcondizioni: Il modulo restituisce ad AMP le informazioni riguardanti la configurazione che vengono mostrate a schermo.

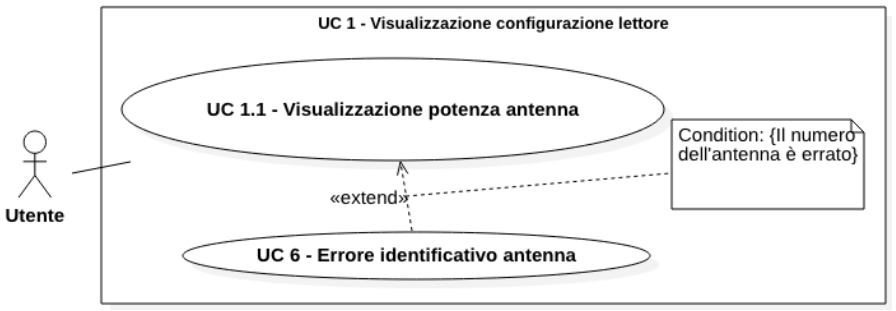


Figura 3.1: Use Case - UC1: Visualizzazione configurazione del lettore

UC2: Mofidica configurazione del lettore

Attori Principali: Utente.

Precondizioni: L'utente è entrato all'interno di AMP ed ha lanciato il comando per modificare la configurazione del lettore.

Descrizione: AMP mette a disposizione una finestra relativa al modulo in cui lanciare comandi verso questo tra cui visualizzazione e modifica della configurazione del lettore.

Postcondizioni: Il modulo restituisce ad AMP un feedback sull'esito dell'operazione e nel caso di esito positivo la configurazione inserita.

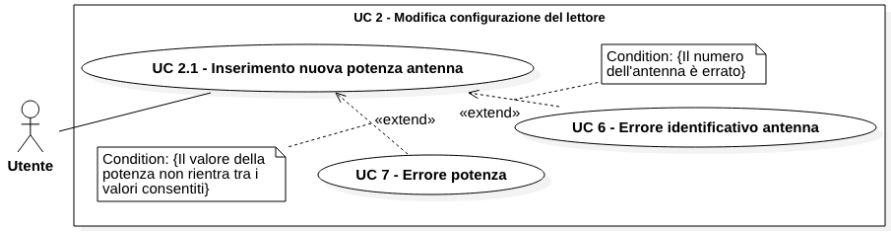


Figura 3.2: Use Case - UC2: Modifica configurazione del lettore

UC3: Modifica dello stato delle porte GPIO

Attori Principali: Utente.

Precondizioni: L'utente è entrato all'interno di AMP ed ha lanciato il comando per modificare lo stato delle porte GPIO.

Descrizione: AMP mette a disposizione una finestra relativa al modulo in cui lanciare il comando relativo alla modifica dello stato di una porta GPIO.

Postcondizioni: Il modulo restituisce ad AMP un feedback sull'esito dell'operazione.

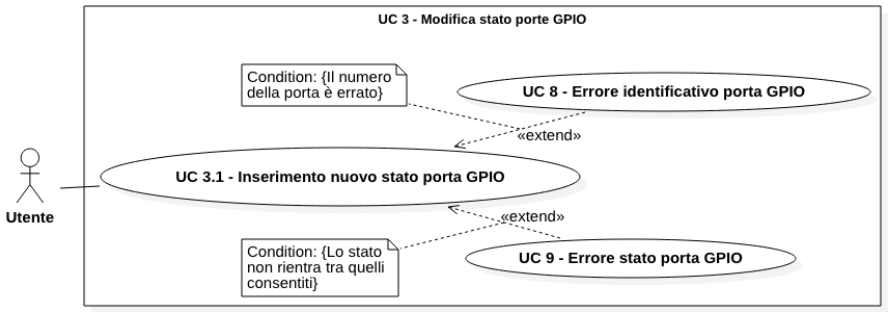


Figura 3.3: Use Case - UC3: Modifica dello stato delle porte GPIO

UC4: Visualizzazione configurazione del lettore

Attori Principali: Modulo, AMP.

Precondizioni: Il modulo ha raccolto dei dati che vuole mandare ad AMP.

Descrizione: AMP mette a disposizione un canale di comunicazione per l'invio di dati da un modulo qualsiasi ad essa.

Postcondizioni: AMP restituisce al modulo un feeback sull'esito dell'operazione.

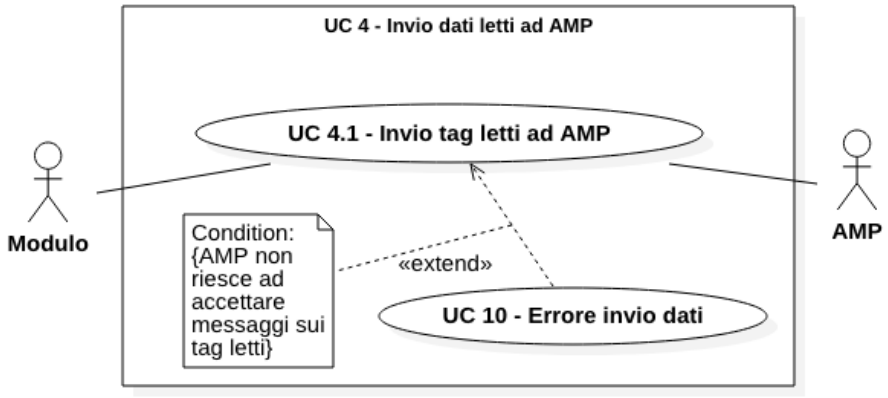


Figura 3.4: Use Case - UC4: Visualizzazione configurazione del lettore

UC5: Rispistino connessione al lettore in caso di anomalia

Attori Principali: Modulo.

Precondizioni: Il modulo riscontra una disconnessione dal lettore.

Descrizione: Il modulo periodicamente controlla l'esistenza di una connessione attiva e valida verso il lettore in operazione.

Postcondizioni: Il modulo entra in uno stato di *recovery* in cui cerca di ripristinare la connessione e notifica l'utente dell'accaduto.

3.2.4 Tecnologie coinvolte

Per lo sviluppo del modulo, come accennato in sezione 2.3.1, sono state utilizzate le seguenti tecnologie:

* **OSGi Knopflerfish**: si tratta di un *framework open-source* sviluppato da Makewave che rispecchia gli standard della OSGi Alliance. Questa fondazione, sovvenzionata a sua volta dalla Eclipse Foundation, si pone l'obiettivo di formalizzare l'evoluzione di tecnolgie industriali per la creazione di soluzioni modulari sulla piattaforma JVM. Knopflerfish offre molte *feature* per agevolare lo sviluppo dell'applicativo, tra cui:
  - Componenti per la creazione di un *bundle* integrabile direttamente all'interno di AMP;
  - Strumenti per l'accesso ad una *thread pool* gestista dalle grandi performance;
  - Un canale di cominicazione per lo scambio di dati tra i *bundle*.

* **JNA**: è una libreria *open-source* che permette di accedere a codice nativo, ovvero scritto in un linguaggio compilato come C e C++, direttamente dal codice scritto in Java. Il suo utilizzo è particolarmente adatto a contesti industriali dove si vuole permettere l'interazione con codice che possa operare direttamente sull'*hardware* del dispositivo. Le principali *feature* che offre per raggiungere questo intento sono:
  - Conversione o rappresentazione di tipi di dati nativi nelle controparti utilizzate dalla JVM;
  - Astrazione del paradigma di programmazione procedurale in programmazione ad oggetti;
  - Gestione autonoma della memoria utilizzata dal codice nativo.
* **Java 8**: linguaggio di programmazione orientato agli oggetti, scelto per lo sviluppo dell'applicativo per la sua integrazione con Knopflerfish. La scelta relativa alla versione è stata presa a valle di alcune considerazioni. La *release* 8 infatti presenta numerose migliorie come l'introduzione dei metodi lambda, conservando però, tutte le aggiunte alla libreria di base riguardante il parallelismo e la concorrenza introdotte nella versione 7. In aggiunta a questo, l'utilizzo di questa versione avrebbe impedito il versificarsi di conflitti fra JVM differenti all'interno della stessa soluzione.

3.3 Sviluppo delle componenti critiche

Durante lo sviluppo del modulo sono state identificati e risolti numerosi problemi, i più importanti tra questi sono stati illustrati di seguito.

3.3.1 Design del modulo

Problematica

La prima difficoltà riscontrata consisteva nella scelta dell'archietettura del modulo. Questo infatti, come descritto precedentemente, doveva potersi integrare all'interno della piattaforma AMP.

Soluzione adottata

Ho deciso di strutturare il modulo in 2 *package* principali: source e driver. Così facendo, ho potuto inserire tutto il codice relativo all'integrazione con il *framework* OSGi e con la piattaforma AMP in un'unico posto. Questo ha permesso di poter astrarre il funzionamento del lettore, implementato all'interno del *package* driver, rendendolo di fatto invisibile ad ogni suo utilizzatore. Il vantaggio principale di questa soluzione è la portabilità del modulo stesso, questo potrà essere riutilizzato con semplicità in futuri progetti cliente, indipendenti dalla piattaforma attualmente in uso. Una visione d'insieme di questa organizzazione è visibile nella figura 3.5. In questa si può notare che il *package* driver ne contenga a sua volta altri, specializzati per compito, completamente invisibili ad ogni possibile utilizzatore del modulo.

Al fine di rendere l'utilizzo del *package* driver il più semplice possibile, ho deciso di utilizzare un *pattern* progettuale apposito, il *command pattern*. Questo, come riportato in figura 3.6, permette di identificare le varie funzionalità e capacità di una

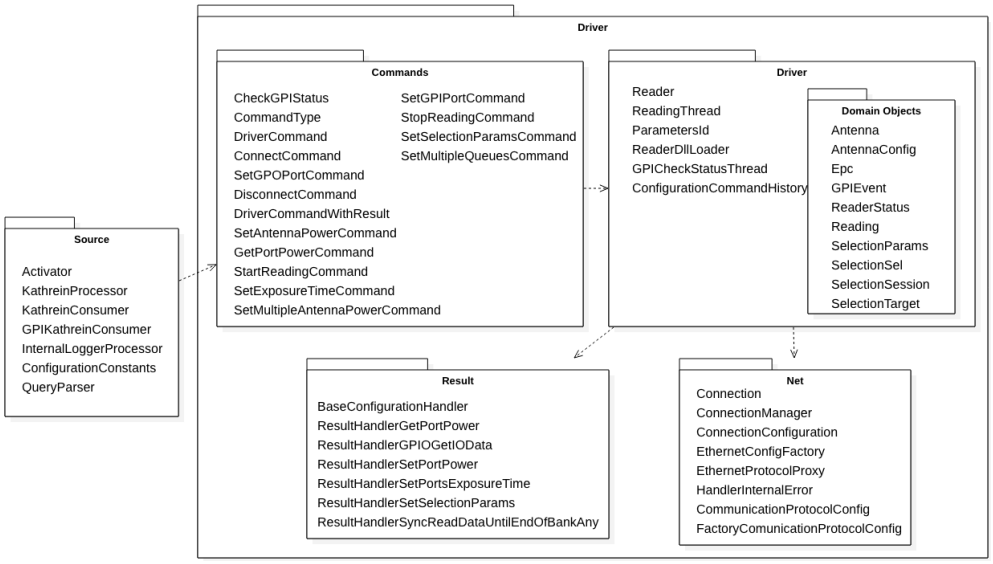


Figura 3.5: Diagramma dei package del modulo

componente del codice come azioni o comandi analoghi a quelli che possiamo trovare sul telecomando di un televisore. Ciò permette di utilizzare tutte le caratteristiche offerte dal lettore senza avere la necessità di conoscerne il funzionamento specifico, semplificando di conseguenza l'applicazione della *business logic* desiderata. Oltre a ciò, tale *pattern* è stato particolarmente adatto per la risultizione di altri problemi, uno di questi viene descritto approfonditamente nella sezione 3.3.4.

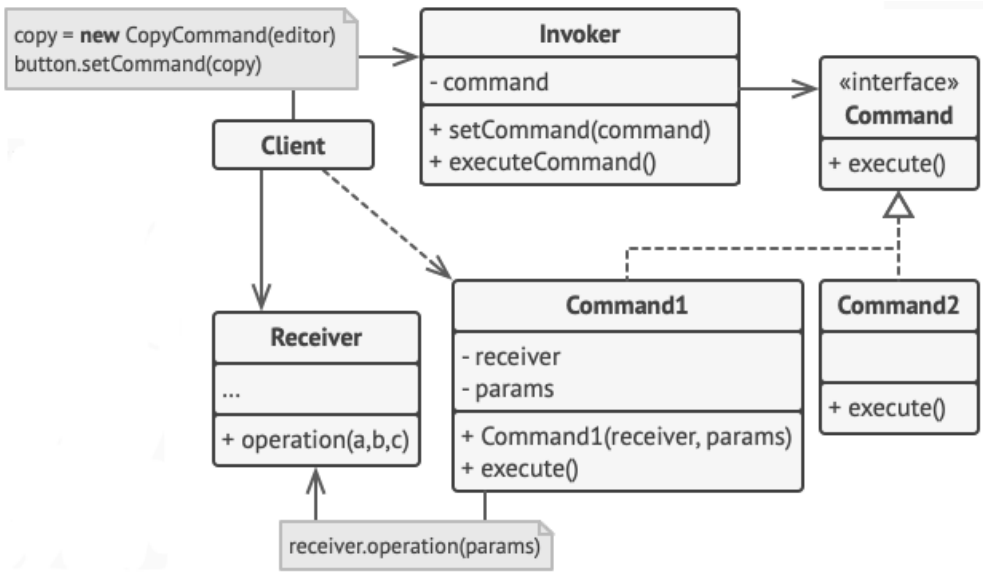


Figura 3.6: Diagramma UML del Command Pattern - **fonte** urly.it/3gn4p

3.3.2 Interfacciamento con librarie native

Problematica

La seconda difficoltà riscontrata consisteva nell'utilizzo delle librerie native offerte dal produttore dei lettori. Questo era dovuto principalmente a 2 fattori. In primo luogo vi era l'accesso a tale codice direttamente dal modulo Java. In secondo luogo vi era la differenza tra i paradigmi di programmazione utilizzati, ovvero, il paradigma procedurale ed il paradigma orientato agli oggetti.

Soluzione adottata

Per risolvere questo problema ho ricorso all'utilizzo della libreria JNA. Questa, tramite l'implementazione di interfacce ed oggetti appositi, permette di effettuare un *wrap* del codice nativo andando a rendere disponibili tutti i tipi di dati primitivi all'interno dell'ecosistema JVM. Due esempi di come questo venga effettuato sono riportati nei frammenti di codice di seguito ripotati. Il primo frammento mostra la conversione di un tipo enumeratore denominato CommunicationStandard da nativo a tipo JVM e la conversione di una funzione nativa che utilizza tale tipo come parametro.

```
// RRU4JavaLibrary.java
public abstract interface RRU4JavaLibrary extends Library {
    public abstract tReaderErrorCode SetCommunicationStandard(Pointer
        readerConnection,
        tCommunicationStandard paramtCommunicationStandard);

    public static enum tCommunicationStandard {
        CS_None((byte)0), CS_Autodetect((byte)1),
            CS_ETSI_EN302208((byte)2), CS_ETSI_EN302208_LBT((byte)3),
        CS_FCC((byte)4), CS_Special((byte)5), CS_China((byte)6),
            CS_Thailand((byte)7), CS_Brazil((byte)8),
        CS_SouthKorea((byte)9), CS_Peru((byte)10), CS_Singapore((byte)11),
            CS_Australia((byte)12), CS_Japan((byte)13),
        CS_Last((byte)14);

        private final byte value;

        private tCommunicationStandard(byte paramByte) {
            this.value = paramByte;
        }

        public byte getValue() {
            return this.value;
        }
    }
}
```

Il secondo frammento, invece, mostra come venga effettuato il caricamente della libreria nativa a seconda della piattaforma in cui il modulo viene eseguito.

```
// ReaderDllLoader.java
public class ReaderDllLoader {

    public static RRU4JavaLibrary load(String pathToLib) {
```

```
        Map<String, Object> options = new HashMap<String, Object>();
        options.put(Library.OPTION_TYPE_MAPPER, new RRU4TypeMapper());
          NativeLibrary.addSearchPath("RRU4", pathToLib);
          return (RRU4JavaLibrary) Native.loadLibrary(
            Platform.isWindows() ? "RRU4" : "ReaderLib",
            RRU4JavaLibrary.class, options);
      }
  }
```

Questo però non è stato sufficente a risolvere i problemi relativi ai differenti paradgmi di programmazione utilizzati. La libreria, infatti, era strutturata in modo che ogni funzione richiamata ritornasse solo un codice di errore relativo alla libreria stessa. Ogni dato rilevante che ci si sarebbe potuti aspettare come risultato, viene invece ricevuto come parametro di una funzione di *callback*. Questo comporta una interruzione nel normale flusso di chiamate dei metodi, con la conseguente necessità di gestione dei dati e degli errori. Per ovviare a ciò ho provveduto ad utilizzare un *Exchanger* fornito dalla libreria standard di Java. L'utilizzo di questo è visile nel frammento di codice di seguito riportato.

```
// ResultHandlerGetPortPower.java
public class ResultHandlerGetPortPower extends BaseConfigurationHandler
    implements tDllResultHandlerGetPortPower {
        @Override
        public void resultHandlerGetPortPower(tResultFlag enResultFlag,
             byte ubPort,
                byte ubPortPower, Pointer rru4Handle, Pointer pTag) {
            if (enResultFlag != tResultFlag.RF_NoError) {
                exchangeData(new ReaderStatus.StatusBuilder()
                        .withException(new ReaderException(
                                "Error occured getting port power: " +
                                    enResultFlag,
                                enResultFlag))
                        .build());
            } else {
                exchangeData(new ReaderStatus.StatusBuilder()
                        .withPortConfig(new AntennaConfig(new Antenna[] {
                                new Antenna(ubPort, ubPortPower / 4, 0) }))
                        .build());
            }
        }
}

// Reader.java
public class Reader {

    public Antenna getPortPower(byte portNumber) throws
         InterruptedException {
      tReaderErrorCode errorCode = libraryWrapper
             .GetPortPower(connection.get().getHandler(), portNumber);
      String MSG = "Error getting antenna power: " + errorCode;
      checkErrors(errorCode, MSG);
      ReaderStatus newStatus = exchanger.exchange(null);
      checkCallBackErrors(newStatus);
      AntennaConfig config = newStatus.getPortConfig();
```

```
        Antenna port = config.getPort(portNumber);
        return port;
    }
}
```

Nello specifico, all'arrivo di una *callback*, la libreria JNA istanzia un nuovo *thread* per elaborarla. Il primo processo che durante l'esecuzione raggiunge l'*exchanger* viene messo in attesa finchè anche il secondo non raggiunge il punto desiderato, instaurando di conseguenza una sincronia fra essi. Una volta che questa sincronia viene creata avviene uno scambio di dati tramite oggetti che permette ai *thread* di continuare con la loro normale esecuzione.

3.3.3 Gestione delle operazioni di lettura

Problematica

Nonostante la sincronizzazione delle *callback*, descritta nella sezione 3.3.2, fornisca molteplici vantaggi, pecca in quanto a prestazioni ed autonomia. Questo è dovuto in primo luogo alle aspettative sul funzionamento del modulo, questo infatti deve poter essere utilizzabile, per esempio: dare informazioni sullo stato delle antenne, notificare dati ad AMP, modificare delle configurazioni del lettore, ecc. Nonostante stia eseguendo delle operazioni di lettura. Il secondo problema riscontrato è di natura invece più tecnica. Per come è stata implementata la libreria offerta dal produttore, anche le operazioni di lettura restituiscono i dati desiderati tramite il meccanismo delle *callback*. Questo comportamento obbliga la creazione di un nuovo *thread* ogni volta che viene effetuata una lettura, rischiando quindi di saturare i *thread* istanziabili dal sistema operativo in uso.

Soluzione adottata

Per ovviare a questo problema ho deciso di trattare le *callback* di lettura dati in modo diverso dalle altre. Nello specifico, il *thread* che viene creato ad ogni chiamata, viene liberato il più velocemente possibile grazie all'utilizzo di un "consumatore". Questo è un processo incaricato di raccogliere i dati delle letture tramite l'utilizzo una struttura sincronizzata, lasciando quindi che la *callback* muoia il più velocemente possibile. Periodicamente questo processo provvederà l'invio di tali dati alla piattaforma AMP. Quanto descritto è visibile nel frammento di codice di seguito riportato.

```
// ResultHandlerSyncReadDataUntilEndOfBankAny.java
public class ResultHandlerSyncReadDataUntilEndOfBankAny
    implements tDllResultHandlerSyncReadDataUntilEndOfBankAny {

    @Override
    public void resultHandlerSyncReadDataUntilEndOfBankAny(
            tResultFlag enResultFlag, byte ubExtendedResultFlag,
            tEPCListEntry pEPCList, Pointer rru4Handle, Pointer pTag) {
        Builder readingBuilder = new Reading.Builder()
                .withEPC(epc.toString())
                 .withReaderTimestamp(pEPCList.udwTimestamp)
                 .withAntenna(pEPCList.ubPort)
                 .withRSSI(pEPCList.ubRSSI);
        consumer.accept(readingBuilder.build());
```

```
    }
}

// KathreinConsumer.java
public class KathreinConsumer {

    private final BlockingQueue<Reading> readQueue = new
        LinkedBlockingQueue<Reading>();
    private ConsumerAction thread;

    public void addToQueue(Reading t) {
      readQueue.add(t);
   }

    private class ConsumerAction extends TerminableYarn {
      @Override
      protected long doCycle() {
         Collection<Reading> buffer = new ArrayList<>(readQueue.size());
         readQueue.drainTo(buffer);
         for (Reading item : buffer) {
            lambda.accept(item.ampFormatMessage(ip, procName));
         }
         return reportPeriod;
      }
   }
}
```

3.3.4 Linguaggio di dominio specifico

Problematica

La quarta difficoltà riscontata consisteva in come gli utenti del modulo avrebbero potuto interfacciarsi con esso. Questi sono composti dai tecnici che effettuano la configurazione dei lettori, e da altri moduli che ne modificano il funzionamento in base a precise regole di *business*. Per i primi è disponibile una finestra di dialogo all'interno di AMP che permette di inviare stringhe di testo al modulo. Per gli altri moduli è possibile inviare sempre del testo tramite delle apposite code all'interno di AMP. Entrambe le tipologie di interazione portano quindi alla chiamata del medesimo metodo.

Soluzione adottata

Per creare un metodo di interazione che fosse univoco e facilmente mantenibile, ho quindi deciso di formalizzare un DLS, ovvero un linguaggio di dominio specifico. Questo è un linguaggio di specifica utilizzabile per comandare il lettore, specificando uno o più comandi concatenabili.

Per la creazione di tale linguaggio ho utilizzato la libreria *open source* Parboiled. Questa fornisce tutto il necessario per l'implementazione di un *parser* PEG, Parsing Expression Grammar, direttamente all'interno del codice Java. In aggiunta a ciò viene fornita un'implementazione di un *parser* ricorsivo discendente ed il supporto per la costruzione di: *abstract syntax tree*, report sugli errori di analisi e procedure di recupero in caso di errore. Simile ad un CFG, PEG è una grammatica formale

analitica deterministica, che descrive un linguaggio formale in termini di un insieme di regole per riconoscere le stringhe che appartengono al linguaggio. Nel frammento di codice, di seguito riportato, è possibile vedere come la libreria venga utilizzata per per la creazione di una parte della grammatica. Questo, unito all'utilizzo del *Command Pattern* descritto in precedenza, permette la creazione di comandi eseguibili partendo esclusivamente da testo.

```
// QueryParser.java
@BuildParseTree
public class QueryParser extends BaseParser<Object> {

    public Rule input() {
      return OneOrMore(Command(), FirstOf(Ch('\n'), EOI));
    }

    public Rule Command() {
      return FirstOf(StartReading(), StopReading(), SetPortPower(),
             GetPortPower(), SetGPO());
    }

    public Rule SetPortPower() {
      return Sequence(QRY_SET_POWER, whiteSpaces(), Arg(), whiteSpaces(),
             Arg(), Optional(whiteSpaces()),
             Boolean.valueOf(push(new SetAntennaPowerCommand(reader, kpl,
                    CommandType.CONF, ((Integer) pop(1)).intValue(),
                    ((Integer) pop()).intValue()))));
    }

    public Rule Arg() {
      return number();
    }

    public Rule number() {
      return Sequence(OneOrMore(CharRange('0', '9')),
             Boolean.valueOf(push(new Integer(match()))));
    }

    public Rule whiteSpaces() {
      return OneOrMore(Ch(' '));
    }
}
```

3.3.5 Tolleranza ai guasti

Problematica

La quinta difficoltà consisteva nella gestione dei guasti dovuti alla disconnessione del lettore dal modulo. Il problema è scomponibile in 3 problemi più piccoli, quali:

* Identificazione del guasto
* Procedura di arresto delle operazioni di lettura
* Ripristino dello stato del lettore in caso di nuova connessione

Soluzione adottata

Per agevolare la gestione della connessione ho quindi provveduto ad implementare un *thread* indipendente, chiamato ConnectionManager, avente il solo scopo di effettuare connessione, disconnessione e controllare che questa sia ancora valida e attiva. Un frammento di codice relativo al funzionamento di questo è di seguito riportato.

```
// ConnectionManager.java
public class ConnectionManager {

    private final AtomicReference<Connection> connection;
    private ReconnectionThread reconnectionThread;
    private final Runnable reconnectionSignal;

    public boolean isConnectionAlive() {
      return connection.get().isValid();
    }

    private boolean attemptConnection() throws Exception {
      Thread.sleep(reconnectionSleep);
      try {
         boolean connected = connect().get().isValid();
         return connected;
      } catch (ConnectException ex) {
         throw new ConnectException("Re-connection failed");
      }
    }

    private class ReconnectionThread extends TerminableYarn {
        @Override
        protected long doCycle() {
            if (isConnectionAlive())
                return 1000;
            try {
                boolean reconnected = attemptConnection();
                if (reconnected)
                    reconnectionSignal.run();
            } catch (Exception e) {
                kpl.info("In attempting (re)connection: " + e.getMessage());
                return reconnectionSleep;
            }
            return 1000;
        }
    }
}
```

Come è visibile, ogni secondo il *thread* controlla che l'oggetto relativo alla connessione sia ancora valido. Nel caso in cui questo sia stato invalidato da una *callback* apposita, andrà a tentare una riconnessione, che nel caso avesse successo, segnalerà al modulo la necessità di riconfigurare il lettore. La configurazione viene effettuata eseguendo nuovamente i comandi di configurazione che sono stati lanciati dall'avvio del modulo. Affinchè questa lista di comandi non presenti ripetizioni inutili, prima di ogni inserimento viene effettuata una riduzione di tale lista. Di seguito viene riportato un frammento del codice responsabile di queste operazioni.

```java
// ConfigurationCommandHistory.java
public class ConfigurationCommandHistory {

    private final List<DriverCommand> commandsHistory = new ArrayList<>(3);

    public void addCommand(DriverCommand command) {
        if (command.getType() != CommandType.CONF)
            return;
        addCommandAndreduceHistory(command);
    }

    public void executeConfHistory() throws Exception {
        Iterator<DriverCommand> it = commandsHistory.iterator();
        while (it.hasNext()) {
            it.next().execute();
        }
    }

    private void addCommandAndreduceHistory(DriverCommand command) {
        for (int i = 0; i < commandsHistory.size(); i++) {
            DriverCommand cur = commandsHistory.get(i);
            if (command.isSameCommand(cur)) {
                commandsHistory.set(i, command);
                return;
            }
        }
        commandsHistory.add(command);
    }
}

public class Reader {

    private void reconfigureReader() {
     history.executeConfHistory();
     kpl.info("Reader re-configured and ready to use");
  }
}
 }
```

3.4 Risultati raggiunti

3.4.1 Requisiti soddisfatti

Un elemento oggettivo, per descrivere la completezza del codice richiesto, riguarda la copertura dei requisiti maturati durante lo stage. L'azienda, avendo già esperienza con progretti analoghi, non ha ritenuto necessario richiedere funzionalità desiderabili o opzionali. Considerando infatti, tutte le richieste fatte come necessarie affinchè il prodotto sviluppato potesse essere utilizzabile all'interno delle loro soluzioni aziendali. Conseguentemente a ciò, i requisiti funzionali, di vincolo e qualitativi, sono risultati essere tutti **obbligatori**. In sezione 3.2.2 sono stati elencati solamente i requisiti degni di nota che però non rappresentano la totalità di quelli effettivamente da soddisfare.

La tabella 3.7 a seguito, invece, rappresenta la totalità dei requisiti individuati con l'azienda.

Tipologia requisiti	Numero requisiti	Classificazione	Soddisfatti
Funzionali	26	Obbligatori	26
Vincolo	5	Obbligatori	5
Qualitativi	2	Obbligatori	2

Tabella 3.7: Piano di lavoro

3.4.2 Prodotti creati

Un altro elemento oggettivo sul lavoro svolto è sicuramente la quantità di prodotti creati. Nel caso specifico ho prodotto diversi file sorgente per un totale di circa 3600 righe di codice, ognuno di essi corredato con l'opportuna documentazione sotto forma di Java-doc. In aggiunta a questo ho redatto 3 documenti, uno riguardante l'utilizzo della libreria JNA e due riguardanti le parti più rilevanti del modulo.

Capitolo 4

Valutazione retrospettiva e conclusioni

Nel corso di questo capitolo si presenta una valutazione retrospettiva sullo stage. Vengono descritti gli obiettivi raggiunti, i progressi personali e il gap tra la formazione offerta dall'Università e quella richiesta dallo stage.

4.1 Obiettivi raggiunti

4.2 Difficoltà riscontrate

4.3 Progressione personale

4.4 Gap formativo tra Università e stage

Glossario

CFG Con *Context Free Grammar* si intente una grammatica formale in cui ogni regola sintattica è espressa sotto forma di derivazione di un simbolo a sinistra a partire da uno o più simboli a destra.. 23, 33

IoT Internet delle cose, nelle telecomunicazioni è un neologismo riferito all'estensione di Internet al mondo degli oggetti e dei luoghi concreti.. 1, 33

M2M Con *Machine-to-machine*, in generale ci si riferisce a tecnologie e applicazioni di telemetria e telematica che utilizzano le reti wireless. Machine-to-machine indica anche un insieme di software e applicazioni che migliorano l'efficienza e la qualità dei processi tipici di ERP, CRM e asset management.. 1, 33

RFID Identificazione a radiofrequenza, in elettronica e telecomunicazioni, è una tecnologia di riconoscimento e validazione e/o memorizzazione automatica di informazioni a distanza.. 3, 33

Acronimi

CFG Context Free Grammar. 31

IoT Internet Of Things. 31

M2M Machine-to-machine. 31

RFID RFID. 31

Bibliografia

Riferimenti bibliografici

Goetz, Brian. *Java Concurrency in Practice*. Addison-Wesley Professional, 2006.

Siti web consultati

Context Free Grammar. URL: https://it.wikipedia.org/wiki/Grammatica_libera_dal_contesto.

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/.

Parboild. URL: https://github.com/sirthias/parboiled/wiki/.

Parsing Expression Grammar. URL: https://it.wikipedia.org/wiki/Parsing_expression_grammar.