Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Concorrenza vs distribuzione nella digital industry. Il caso Sanmarco Informatica

Relatore
Prof. Armir Bujari
Università di Padova

Tutor esterno
Alex Beggiato
Sanmarco Informatica

Candidato
Davide Barasti

Dedica..

Abstract

...

Indice

Elenco delle figure

Elenco delle tabelle

L'avvento e la crescita di internet hanno rivoluzionato il modo di fare impresa in particolare quella industriale. Ciò che prima era un impianto isolato, con operatori e responsabili, ora è parte di una catena produttrice interconnessa che genera non solo beni tangibili ma anche dati che, se ben gestiti, possono diventare informazione utilizzata per aumentare l'efficienza dell'intera impresa in un delicato ciclo di feedback.

1 Introduzione

Prima di affrontare il problema oggetto di questa tesi è doveroso fornire una panoramica del contesto aziendale che ha caratterizzato lo stage svolto. In questo modo i capitoli successivi risulteranno più chiari e inseriti in una propria logica.

1.1 Sanmarco Informatica SpA

L'azienda Sanmarco Informatica (SMI) dal 1984 si occupa di consulenza e sviluppo *software*. Si è specializzata nella progettazione, realizzazione e installazione di soluzioni a supporto dei processi aziendali di duemila aziende con numerose installazioni oltre confine.
L'azienda ha tra i suoi punti di forza quello della vicinanza ai clienti, che si traduce in diverse sedi di proprietà in Veneto, Lombardia, Emilia-Romagna e Friuli-Venezia Giulia con 450 dipendenti totali.

Il Centro Ricerca e Sviluppo (CRS) è situato a Grisignano di Zocco (VI), sede di lavoro per oltre 150 dipendenti. Questo è il fulcro dello sviluppo e della manutenzione dei prodotti *software*. Team di sviluppatori e sistemisti sono coordinati per garantire stabilità di servizio per i clienti, e un'alta personalizzazione dei prodotti offerti.
La ricerca e l'innovazione sono di grande valore per SMI, con una media del 20% di fatturato investito annualmente in ricerca e sviluppo. Sono anche numerose le

risorse impiegate nella formazione e ricerca di talenti provenienti dall'Università di Padova. Un esempio di questo è il progetto *Academy* che viene descritto nel seguente modo: "Lo scopo è quello di educare giovani allievi da inserire direttamente nel mondo lavorativo, attraverso un percorso di eccellenza che forma figure professionali altamente qualificate."*.
Il CRS è anche responsabile per l'erogazione dei servizi *cloud* forniti da SMI. È infatti presente una sala server amministrata da tecnici sistemisti.
Tutta la proprietà operativa di SMI rimane in azienda, senza il bisogno di *outsourcing* ad aziende esterne.

L'azienda è organizzata in *business unit (BU)*, un sottoinsieme dell'impresa che rappresenta un *business* specifico concentrato su una particolare linea di prodotti.

La BU interessata dallo stage è JMES, composta da 20 persone tra sistemisti e sviluppatori, che si occupa dell'omonimo applicativo utilizzato nell'ambito della *digital industry*. Questo software serve a gestire e supervisionare l'avanzamento della produzione all'interno della fabbrica.

Figura 1.1: Logo BU Jmes

Altre BU si occupano invece di *Business Process Management* (JPA), gestione contenuti aziendali come manuali e documenti della qualità (Discovery ECM), *marketing* (nextBI), sviluppo di applicazioni e siti per aziende (4words) e molto altro.

*www.sanmarcoacademy.com

Ulteriori informazioni riguardanti le BU o l'azienda in generale possono essere trovate sul sito ufficiale di Sanmarco Informatica.

1.2 La Digital Industry

Introduzione alla sezione qui?

1.2.1 L'industria 4.0

L'avvento e la crescita di internet hanno rivoluzionato il modo di fare impresa in particolare quella industriale.
Ciò che prima era un impianto isolato, con operatori e responsabili, ora è parte di una catena produttrice interconnessa che genera non solo beni tangibili ma anche dati che, se ben gestiti, possono diventare informazione utilizzata per aumentare l'efficienza dell'intera impresa in un delicato ciclo di *feedback*.
La corretta gestione di questa informazione diventa fondamentale per il successo dell'impresa poichè mantiene alta la competitività.

Questi nuovi aspetti vengono spesso riassunti nel termine *quarta rivoluzione industriale*, un altro termine comune in questo ambito è infatti *Industry 4.0*.

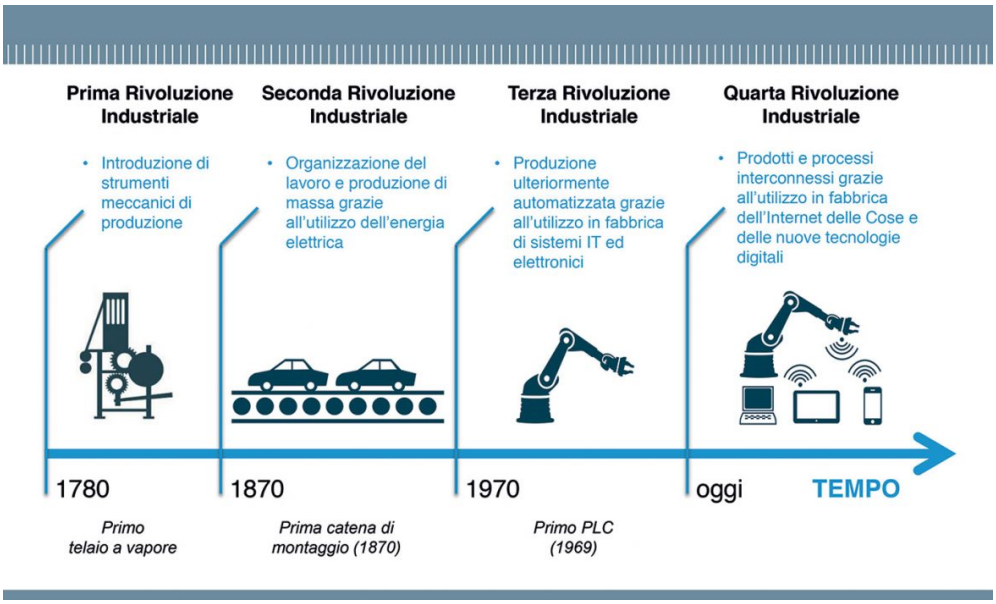

Figura 1.2: Una visione temporale delle rivoluzioni industriali fino ad oggi

Nasce quindi la necessità di un software che trasformi il dato in informazione e che faccia uso di questa per migliorare le *performance* supportando e gestendo a sua volta tutta la linea di produzione con indicazioni precise e in tempo reale.
Questo strumento prende il nome di *manufacturing execution system (MES)*.
Un software MES in pratica si occupa di raccogliere, elaborare e condividere informazioni provenienti direttamente dall'industria che normalmente restano disaggregate per diversi motivi: si trovano in formato non elettronico, sono poco precise o si trovano in ambienti isolati tra loro.

1.2.2 Manufacturing execution system

Il controllo e la gestione dell'informazione consentono non solo di mantenere un piano generale del processo produttivo aziendale, ma anche di evidenziare possibili problemi in anticipo e ottenere quindi una migliore resa di produzione.
Vediamo quali sono i principali benefici di un sistema MES[†]:

1. Riduzione di scarti e sprechi. Grazie ad una visione in tempo reale della produzione, è possibile individuare con piccoli margini situazioni in cui le unità prodotte non sono conformi, fermando la produzione e limitando gli sprechi;

2. Precisione nella definizione dei costi. Con un sistema MES, i tempi di lavoro, gli sprechi, i tempi di inattività, la manutenzione sono monitorati e registrati in tempo reale direttamente dalla fabbrica. Questo rende innanzitutto l'informazione più affidabile, e utilizzabile per riclassificare i costi e valutare criticamente situazioni con alto tasso di sprechi;

3. Ridurre i tempi di inattività. Siccome una produzione ferma difficilmente porta a un guadagno, un MES deve essere in grado di pianificare la produzione in modo che le risorse necessarie siano quelle disponibili, integrando anche i piani dei turni dei dipendenti, in modo da aumentare ulteriormente l'efficienza, riducendo quindi i costi;

4. Riduzione del magazzino. Le giacenze in magazzino costano, perché aumentano i costi logistici di gestione, oltre ai costi per produrre tali giacenze. Con un MES si ha uno stato aggiornato della nuova produzione, degli scarti e dei prodotti non conformi. In questo modo chi si occupa di acquisto, spedizione e pianificazione sa esattamente cosa è disponibile e cosa bisogna ordinare;

[†][*Five benefits of a MES*], www.industryweek.com/companies-amp-executives/five-benefits-mes

5. Riduzione di costi. Con un controllo più stretto sui tempi e i costi necessari per la produzione, è possibile snellire ulteriormente processi a supporto, come la gestione logistica del magazzino, quindi ottenendo un disimpegno di persone e attrezzature.

1.3 Le squadre JMES

Come accennato nel paragrafo 1.1 esistono due squadre che lavorano in maniera sinergica per lo sviluppo e l'installazione del prodotto JMES:

- Sviluppatori. Sono raggruppati all'interno del team di JMES e si occupano di implementare le funzionalità aggiuntive richieste dai clienti. Le richieste possono provenire da un singolo cliente o essere inserite in distribuzioni di aggiornamento per tutte le installazioni. La differenza sta nel valore monetario dell' *update*. La figura a cui fanno riferimento è lo *Scrum Master*. Ad agosto 2019 il team è composto da otto persone;
- Sistemisti. Effettuano un'analisi tecnica presso le sedi dei clienti evidenziando possibili problemi di compatibilità con le attrezzature presenti, propongono le diverse configurazioni del software JMES in base a requisiti e vincoli posti dal cliente, organizzano l'installazione e la configurazione del prodotto. La figura a cui fanno riferimento è il capo progetto. Ad agosto 2019 il gruppo è composto da dieci persone.

Per il periodo dello stage sono stato inserito nel team di sviluppo JMES.

La prima installazione di JMES risale a maggio 2018. È quindi un prodotto giovane che però è stato già apprezzato da 19 clienti attivi, ci sono inoltre 27 analisi per l'installazione in corso e altre 19 da pianificare.

1.4 Metodologia di lavoro

1.4.1 Sviluppo agile e framework Scrum

Indipendentemente dalle BU aziendali, la metodologia di lavoro per la gestione del ciclo di sviluppo del software è Agile, implementata con il *framework* Scrum.

Agile è una disciplina per lo sviluppo di software che pone al primo posto tra i suoi principi la soddisfazione e il coinvolgimento del cliente e la distribuzione di software funzionante in maniera regolare e a distanza di brevi periodi, dalle due settimane ai due mesi.

Il *framework* Scrum prevede di suddividere un periodo di lavoro, chiamato *sprint cycle* in tre fasi principali:

- *Planning*: il team comunica con gli *stakeholder* (rappresentati dal *product owner*), analizza e comprende i requisiti creando lo *sprint backlog*, composto di requisiti che il prodotto deve soddisfare entro la fine dello *sprint*. Questi prendono il nome di *story* se sono completabili entro uno sprint. Un insieme correlato di *Story* si chiamano *Epic*;
- *Implementation*: durante questa fase dello *sprint* il team crea delle porzioni complete di prodotto. Le funzionalità implementate in uno *sprint* provengono dallo *sprint backlog*. La durata di questa fase è, nel caso del team JMES, quattro settimane;
- *Review*: ci si riunisce per revisionare il lavoro svolto e pianificare ciò che non è stato possibile portare a termine nella fase di *implementation*. Ogni membro del team mostra una *demo* delle funzionalità sviluppate nella fase precedente;
- *Retrospective*: vengono analizzate in modo retrospettivo le fasi precedenti in un'ottica di miglioramento continuo dei processi al fine di renderli più efficienti negli sprint successivi. Si discute di strumenti e metodologie utilizzate e di come queste abbiano influito nello sviluppo. Se vengono ritenute poco utili, la loro pratica viene dismessa.

1.4.2 Tecnologie di sviluppo del team

Descritte le tecnologie di sviluppo del team JMES.
Principalmente si paralerà del linguaggio Java e del framework sviluppato da SMI per lo sviluppo: *Synergy*.

1.4.3 Strumenti di lavoro del team

Al centro degli strumenti per la gestione del flusso di lavoro del team JMES c'è uno strumento sviluppato da IBM, Rational Team Concert (RTC). Questo offre le seguenti funzionalità:

- versionamento del codice;
- gestione dei *work item* provenienti dallo *sprint backlog* (*issue tracking system*);
- *build tool*.

RTC si integra all'interno dell'ambiente di sviluppo del team grazie a un *plug-in* completo per l'IDE Eclipse;

1.5 Outline del documento

L'obiettivo di questo documento è di raccogliere e tradurre a parole l'esperienza fatta durante lo stage curricolare presso Sanmarco Informatica svolto nel periodo 10 giugno 2019 - 8 agosto 2019.

A partire dal prossimo capitolo si toccheranno tutti gli argomenti tecnici trattati. Iniziando con un chiarimento sulle tecnologie adottate per il progetto di stage, si passerà poi alla motivazione che ha spinto SMI a fare questa proposta di stage. Infine nel capitolo 4 verranno sintetizzate le attività di analisi, progettazione e sviluppo che sono iniziate a fine giugno 2019 e sono terminate agli inizi di agosto 2019.

Nonostante il tempo limitato, durante lo stage ho cercato di esplorare diversi aspetti dell'ingegneria del software che durante il percorso triennale ho potuto analizzare solo dal punto di vista teorico.
L'esperienza mi ha infatti permesso di trascorrere del tempo in un team che utilizzava la disciplina *Agile*, ho sperimentato la tecnica del *pair programming* durante una fase dello sviluppo e ho potuto mettere in uso il modello del *Test Driven Development (TDD)*, anche in questo caso in maniera limitata rispetto all'intero progetto.

Nel capitolo Background Tecnologico vengono analizzate più nel dettaglio le tecnologie già citate in questo capitolo, come il software JMES. Sono introdotti i linguaggi di programmazione utilizzati e gli strumenti per lo sviluppo e la gestione del codice.

Il capitolo Le Problematiche intende presentare i problemi che hanno portato alla proposta di stage. Il prodotto sviluppato durante lo stage mira a riprodurre le funzionalità principali di un software già presente in SMI, il cui sviluppo è terminato a metà 2018, che presentava diversi problemi sul piano architetturale. Questi hanno minato la stabilità e l'affidabilità del prodotto, che necessitava quindi di un rifacimento.

Nonostante il fallimento della prima versione di JDI, la seconda versione da me sviluppata doveva riprodurre le funzionalità di base per poter essere una solida base di partenza per il resto del team JMES che una volta terminato lo stage avrebbe preso

in carico il progetto. Il capitolo 4 riassume l'analisi della prima versione e la raccolta dei requisiti. L'attività di progettazione ha ricoperto un ruolo particolarmente importante poichè doveva evitare gli errori commessi durante la progettazione del primo JDI. Verrà qui messa in luce la contrapposizione tra l'architettura distribuita della prima versione e quella concorrente della seconda.

Infine nel capitolo 5 raccolgo le impressioni dell'esperienza di stage all'interno di SMI, di quale preparazione ho sentito la mancanza e per quali aspetti ho più apprezzato il corso di studi.

2
Background Tecnologico

In questo capitolo si fornirà una introduzione alle tecnologie che risulterà utile per comprendere al meglio i capitoli successivi.
Gli argomenti trattati saranno principalmente JMES, JDI e alcuni standard della digital industry (e.g. protocollo Modbus)

2.1 JMES

Nella sezione 1.2.2 sono stati individuati i principali benefici che l'utilizzo di un *software* MES porta.
Vediamo quali sono i benefici che JMES, l'implementazione MES di SMI intende portare ai clienti con il suo prodotto:

1. Avanzamento processi produttivi. L'informazione sull'avanzamento della produzione è disponibile in tempo reale, permettendo una gestione più flessibile del lavoro. Ad esempio gli uffici commerciali che si trovano in sede distaccata dal centro produttivo possono avere informazioni dettagliate sullo stato di avanzamento di un ordine in breve tempo, senza nemmeno interpellare gli operatori o i responsabili di produzione;
2. Gestione materiali. La disponibilità in magazzino dell'ordine completato è immediatamente rilevata, permettendo un proseguimento di processo più reattivo. Si pensi ad esempio alla possibilità di richiedere una spedizione appena l'ordine risulta saldato;

3. *Monitoring* dei processi. Oltre all'avanzamento del singolo ordine, di interesse specialmente per operatori e impiegati, è possibile effettuare il monitoraggio al fine di supportare la *business intelligence* aziendale, cioè l'insieme delle strategie usate dall'impresa per analizzare i dati di produzione (stato dell'impianto, individuazione dei problemi, macchinari che portano spesso a rallentamenti);

4. Consuntivazione. Il beneficio per cui i sistemi MES nascono: rilevare i tempi di processo per valutare il discostamento da quanto preventivato. Impiegare più tempo del previsto significa ridurre il margine di guadagno che all'estremo può portare ad una perdita. Rilevare gli scostamenti permette di arrivare principalmente a due conclusioni:

   - Lo standard di valutazione è errato, ottenendo dalla serie storica dei rilevamenti una conferma che il processo produttivo non è in grado di rispettare i tempi preventivati;
   - Il processo è migliorabile. Ci sono delle casistiche che portano ad una degradazione occasionale dei tempi di produzione. Ad esempio fermi macchina ricorrenti o operatori non abbastanza efficienti.

   La possibilità di rilevare queste informazioni permette di accorgersi in tempo di situazioni critiche in cui ad esempio i rilevamenti effettuati si allontanano dal piano di budget;

5. Progetto carta zero. Grazie alla gestione software della produzione, si riduce la carta circolante che molto spesso veniva utilizzata per tracciare le fasi degli impianti, per registrare le interazioni operatore-macchina e per le stampe di documenti tecnici di assemblaggio.

L'applicazione JMES presenta un'architettura a *layer* schematizzata in figura 2.1. Si tratta di un'applicazione che utilizza *AngularJS* per l'interfaccia grafica e si affida a servizi web esposti da un'applicazione Java su web server *Apache Tomcat* per gestire i flussi di dati *front-end - back-end*.
Poiché l'applicazione oggetto di questa tesi non è logicamente legata al software JMES, non risulta utile ai fini della comprensione analizzare dettagliatamente la sua composizione.
Dalla prossima sezione si sottolineerà la mancanza in JMES di un collegamento alle macchine industriali, che fino ad ora è stato dato per scontato. Scopriremo invece che questo elemento

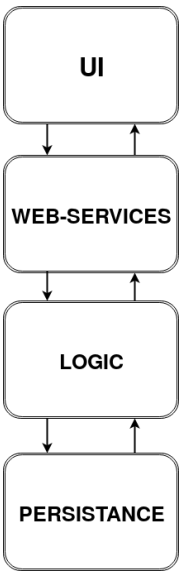

Figura 2.1: Rappresentazione dei *layer* di JMES

non è fondamentale in un sistema MES, e non è quindi sempre presente, ma può aumentarne l'efficienza e l'usabilità.

2.2 Controllori a Logica Programmabile

PLC sta per *Programmable Logic Controller*, controllore a logica programmata. Si tratta di un computer ideato per il mondo industriale. Controlla differenti processi ed è programmato in base ai requisiti del processo a cui viene applicato.

Molte industrie mettono in pratica processi produttivi specifici, ad esempio per la creazione di un certo bene. Modificare questo processo richiede il rifacimento di gran parte dell'apparato produttivo utilizzato, ad un costo estremamente elevato.

Per superare questo problema una prima versione del PLC fu inventata da *Dick Morley*, che al tempo lavorava per *Modicon*, nel 1968 *. Un PLC può essere in breve descritto come un sistema di controllo che contiene la definizione di una sequenza programmata.

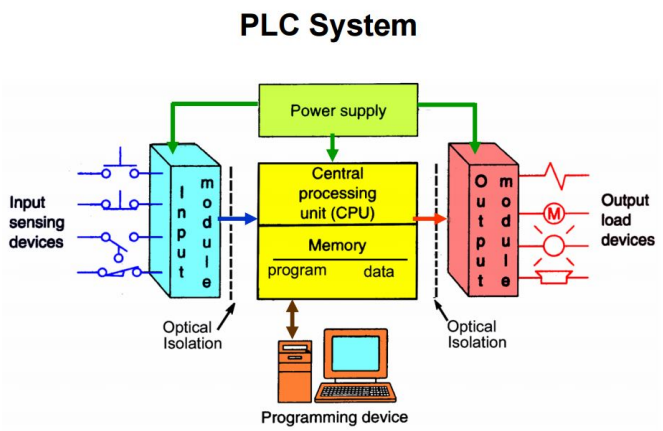

Figura 2.2: Schema di un classico PLC e dei suoi moduli. (Dall'articolo *Engineering Essentials*. Disponibile su https://www.machinedesign.com/engineering-essentials/engineering-essentials-what-programmable-logic-controller)

I PLC sono modulari, quindi componibili in diverse configurazioni. I moduli fondamentali sono quelli di *input* e di *output*. Entrambi sono in grado di gestire segnali analogici e digitali. Sono progettati per essere robusti e per resistere a forti condizioni atmosferiche.

Sono programmabili con linguaggi di programmazione ad alto livello che sono facilmente comprensibili. Il linguaggio più comune è *Ladder Diagram*. I programmi sono scritti su normali computer per poi essere

*[https://library.automationdirect.com/history-of-the-plc/]

trasferiti ai PLC via cavo o via rete.

Questi controllori sono stati ideati per sostituire i componenti elettrici (soprattutto relé e *timer*) con collegamenti fissi che caratterizzavano i vecchi impianti elettrici nei processi industriali, rendendo più semplice riconfigurare un impianto.

I PLC hanno anche degli svantaggi. Non sono generalmente in grado di gestire dati complessi, non sostituiscono quindi i computer. Hanno inoltre bisogno di moduli ulteriori per permettere la visualizzazione di dati.

2.3 L'interazione macchina-MES: JDI

Con un sistema MES tradizionale esistono due tipi di interazione: uomo-macchina e uomo-MES. La prima consiste nell'insieme di azioni che l'operatore svolge sulla macchina per avanzare nel processo produttivo, la seconda consiste nella dichiarazione delle azioni svolte dall'operatore al sistema MES.

Si consideri ad esempio l'azione che un operatore può svolgere su una macchina per il taglio laser: l'operaio o l'operaia effettua una o più operazioni di taglio. Una volta terminate deve dichiarare al sistema MES utilizzato nello stabilimento che ha terminato il lavoro e ha portato a termine X tagli validi, ha prodotto Y scarti e che il macchinario si è bloccato Z volte.

Questa approccio può generare un bias che consiste nella differenza tra quanto effettivamente svolto durante la prima interazione e quanto dichiarato nella seconda, in quanto l'operatore può accidentalmente compiere degli errori durante le dichiarazioni manuali.
Per questo il mercato ha richiesto a SMI una soluzione in grado di permettere una maggiore precisione delle rilevazioni di produzione.

Figura 2.3: Interazioni tra operatore e JMES

L'elaborazione di questo bisogno ha portato a considerare l'introduzione di una terza interazione ai fini di ridurre quando possibile l'interazione operatore-MES. In una visione semplicistica dello scenario questa interazione può essere definita come tra la macchina e il sistema JMES. Nella pratica però si tratta di inserire un terzo attore che si occupi di fornire a JMES tutti i dati ricavati dalla macchina (o meglio, come vedremo, dal PLC collegato alla macchina). Questo attore si chiama JDI, acronimo ideato da SMI che significa *Java Digital Industry*.

Figura 2.4: Rappresentazione di come JDI si integra nei sistemi di produzione e con JMES

Alle macchine industriali moderne è quasi sempre collegato un PLC che si occupa di comandare la macchina a cui e connesso. Inoltre elabora e immagazzina i dati che vengono generati relativi ad esempio al numero di pezzi prodotti, lo stato degli allarmi o l'interruzione di funzionamento della macchina.

JDI è un prodotto a se stante, indipendente da JMES. L'idea alla base di questo prodotto è un software che si interfacci con i PLC nelle aziende dei clienti per intercettare i dati immagazzinati. Questi dati sono accuratamente gestiti e resi disponibili oltre a JMES, ad altre applicazioni della famiglia SMI.

2.4 Gli standard della Digital Industry

Esistono decine di produttori di PLC e spesso ognuno implementa i propri protocolli[†]. Le aziende clienti di SMI presentano una grande variegatura di tipologie di PLC e relative politiche di comunicazione. Di seguito una lista con degli esempi di protocolli supportati da SMI per i suoi clienti:

- S7;
- Modbus TCP/IP;
- OPC UA;
- MTConnect;
- MQTT.

Ogni protocollo definisce specifiche proprie di comunicazione con una certa complessità. Dato il tempo limitato a disposizione durante lo stage ho circoscritto il dominio dell'applicazione per gestire una sola tipologia di protocollo: Modbus TCP/IP. È stata posta particolare attenzione durante lo studio dell'architettura in modo che sia possibile estendere le capacità dell'applicazione per gestire ulteriori protocolli.

2.4.1 Modbus

Modbus è un protocollo di comunicazione seriale sviluppato inizialmente da *Modicon*. È stato concepito per operare con i PLC. È un protocollo del livello applicativo, che opera al settimo strato della scala OSI e permette una comunicazione *client-server* su differenti tipi di rete. Definisce un modo di accedere e controllare un dispositivo tramite un altro senza dipendere dalla rete fisica coinvolta nella comunicazione.

Il protocollo descrive la modalità con cui un dispositivo accede ad un altro, come l'informazione è ricevuta e come devono essere strutturate le risposte alle *query*. In caso di errore, il protocollo definisce un meccanismo per inviare il corretto comando a chi ha richiesto l'operazione.

[†]https://en.wikipedia.org/wiki/List_of_automation_protocols

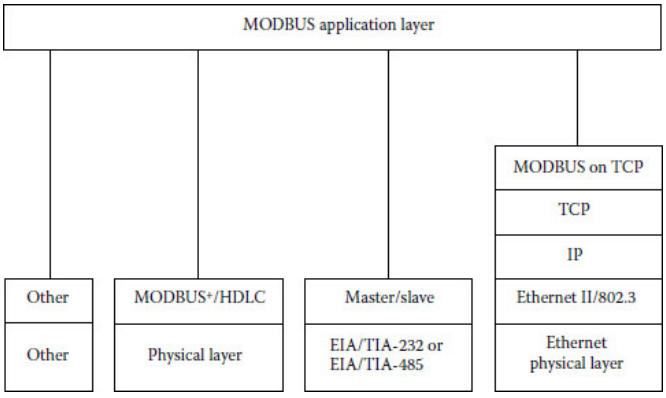

Figura 2.5: *Stack* di comunicazione di Modbus. (Da MODBUS, Application Protocol Specification, vol. 1.1b, 28 dicembre 2006. Disponibile su www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf.)

La comunicazione può avvenire su diversi tipi di rete (ad esempio *Ethernet*) incorporando il protocollo Modbus in pacchetti dati nel protocollo che si intende utilizzare. Ci sono quindi diversi modi di implementare Modbus. Uno di questi è con *TCP/IP over Ethernet*.

2.4.2 Modbus TCP/IP

La specifica Modbus TCP/IP è stata introdotta nel 1999. Ci sono dei vantaggi nell'utilizzare questo protocollo, tra cui la semplicità di utilizzo, l'uso di *Ethernet* e il fatto che sia una specifica aperta.

Modbus TCP/IP è un protocollo internet. Consiste nel protocollo Modbus inserito in un contenitore TCP. In pratica quindi i dispositivi Modbus possono comunicare su Modbus TCP/IP. L'unico vincolo è quello di un *gateway* che effettui le conversioni adatte per passare i dati dallo strato fisico a quello *Ethernet* e da Modbus a Modbus TCP/IP.

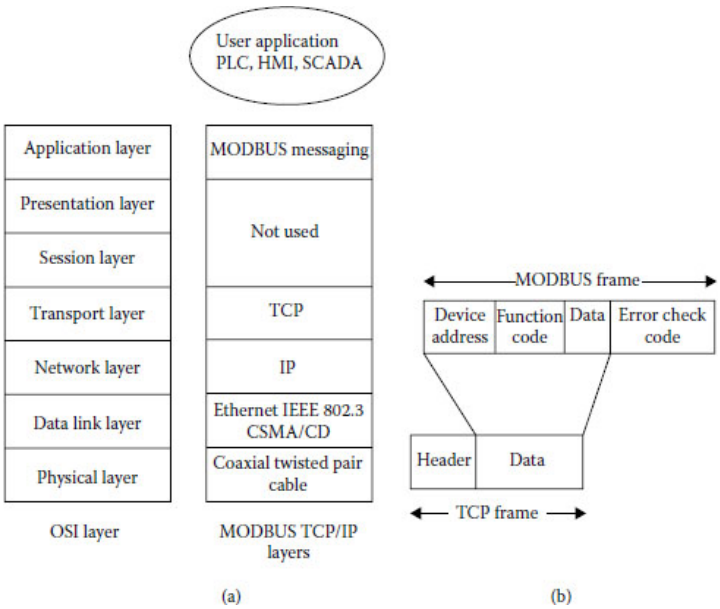

Figura 2.6: (a) *layer* di Modbus TCP/IP; (b) *frame* Modbus TCP/IP. (Da *Fieldbus and Networking in Process Automation*, S.K. Sen. CRC Press, 2017)

La figura 2.6 mostra gli strati del protocollo Modbus TCP/IP affianco allo standard OSI e il *frame* Modbus contenuto nel *frame* Modbus TCP/IP.

2.5 Tecnologie e strumenti impiegati

Saranno qui elencate e descritte le tecnologie e gli strumenti scelti per lo sviluppo del progetto e altri a supporto delle diverse attività.

Saranno giustificate alcune scelte, come la riduzione quasi a zero di librerie esterne, e l'approccio "chiuso" di SMI verso il fare affidamento a prodotti esterni.
Ho notato infatti che un ragionamento spesso fatto dai responsabili tecnici è "preferiamo reinventare la ruota sviluppando internamente librerie anche già esistenti, per avere pieno controllo sullo sviluppo e sulla manutenzione". Cercherò di analizzare criticamente questo ragionamento.

Come accennato in 2.3, il lavoro da me svolto non è integrato in JMES ma è un modulo esterno che permette di ampliare le sue funzionalità. Questo mi ha permesso di avere un certo grado di libertà nella scelta degli strumenti di lavoro, prediligendo tecnologie che ero curioso di conoscere più nel dettaglio.

I principali strumenti per lo sviluppo da me utilizzati sono stati i seguenti:

- Intellij IDEA. Un ambiente di sviluppo per Java prodotto da *JetBrains* che mi ha permesso, grazie anche a numerosi *plug-in*, di integrare in un solo ambiente lo sviluppo *software*, i test automatici, la *build* e il versionamento del codice;
- Gradle. Un sistema per l'automazione di *build* che prende spunto da i classici Apache Maven e Apache Ant che permette di specificare la configurazione del progetto con un linguaggio specifico basato su Groovy;
- Git. Sistema di versionamento distribuito utilizzato in questo progetto assieme a GitHub che ha fornito il servizio di *repository* remoto.
  Il suo uso è stato fondamentale in quanto sono state create, sopratutto nel primo periodo di sviluppo, diverse possibili soluzioni al problema. Git ha permesso di mantenere in parallelo ogni soluzione sviluppata, grazie all'uso di *branch*;
- GitHub ITS. *Issue tracking system* integrato in GitHub, ha permesso di tenere traccia dei cambiamenti durante il progetto. Le *issue* sono state inserite nel *kanban board* integrata in GitHub per avere ben visibile la situazione del progetto in ogni momento della progettazione e dello sviluppo.

Per quanto riguarda invece gli strumenti e le tecnologie non prettamente legate allo sviluppo ma che hanno supportato alcune attività abbiamo:

- Wireshark: strumento per la cattura e l'analisi di pacchetti utilizzato per la comprensione della comunicazione con protocollo Modbus TCP/IP, di cui si è parlato in 2.4.2, tra PLC e computer;
- Modbus PLC simulator: programma di simulazione di PLC Modbus che supporta diversi protocolli, tra cui TCP/IP. Utilizzato per i test di scalabilità e di consumo risorse dell'applicazione realizzata;
- SoMachine Basic: gran parte dei test sono stati eseguiti con PLC fisici. Questo software ha permesso di visualizzare e modificare la configurazione interna dei PLC utilizzati;
- Ladder Diagram: linguaggio grafico per la programmazione di PLC. Utilizzato quando necessario per modificare il comportamento di un PLC durante i test.

Per la comunicazione interna al team JMES è stato utilizzato lo strumento di messaggistica SLACK, mentre per la comunicazione più formale interna ed esterna all'azienda mi è stato fornito un account di posta elettronica.

La prima versione di JDI era organizzata a nodi router-link che potenzialmente potevano permettere di distribuire il prodotto in zone fisiche diverse, comunicando poi tramite WebSocket. Nella realtà però gli stakeholder di JMES hanno constatato che gli svantaggi introdotti dalla soluzione sviluppata superavano possibili vantaggi commerciali del prodotto.

3 Le Problematiche

In questo capitolo si analizzerà brevemente la situazione che ha portato al primo sviluppo di JDI nel 2018, gli errori commessi durante lo sviluppo e la progettazione, il sorgere dei primi problemi arrivati in produzione e i propositi per la nuova versione di JDI, oggetto della tesi.

3.1 Il primo sviluppo di JDI

Dai bisogni sottolineati in 2.3 SMI ha deciso di iniziare lo sviluppo di un *middleware* (JDI) che fosse in grado di interfacciarsi con i PLC utilizzando i giusti protocolli di comunicazione. Lo sviluppo iniziò a maggio 2018 e venne assegnato alla BU JMES. Terminò circa sei mesi dopo.

Per sottolineare ulteriormente la separazione logica tra JMES e JDI è utile ricordare che JMES serve all'operatore per determinare come organizzare il proprio turno lavorativo. JDI invece mira ad aiutare l'operatore nelle operazioni manuali che possono facilmente portare a errori. Ad esempio la dichiarazione dei pezzi prodotti da un macchinario o la segnalazione dell'interruzione di funzionamento di una macchina e di tutte le operazioni *error-prone*.

Lo sviluppo è stato eseguito da un solo programmatore a cui non erano stati posti vincoli precisi per la progettazione dell'applicazione, se non l'utilizzo del linguaggio

Java. Questo ha dato piena libertà sulla scelta delle tecnologie di sviluppo.

Architetturalmente JDI si presentava nel seguente modo: un software a servizi con compatibilità Windows. I servizi rappresentavano nodi che potevano essere *Router* o *Link*. Ogni nodo veniva avviato come servizio Windows. I *Link* erano servizi che utilizzavano un file di configurazione per effettuare la connessione con i PLC. Ognuno stabiliva una connessione con i PLC utilizzando il protocollo adatto (ricavato dal file di configurazione) e una connessione verso il nodo *Router* tramite *WebSocket*. Quest'ultima serviva a comunicare i risultati delle operazioni svolte dai nodi *Link* su i PLC. Nella figura 3.1 sono rappresentate queste connessioni.

Il nodo *Router* effettuava delle chiamate http verso JMES per comunicare le variazioni dei dati rilevati dai nodi *Link*. A quali dati fosse interessato JMES era specificato nei file di configurazione che ogni link utilizzava per determinare le richieste da fare ai PLC.

Le richieste erano principalmente

- Lettura di registri *general purpose* su cui il PLC manteneva dati relativi al funzionamento della macchina a cui era collegato. Ad esempio il numero di pezzi prodotti fino a quel momento;
- Scrittura di registri *general purpose*, per resettare determinati stati, o per attivare/disattivare funzioni sulla macchina.

La soluzione sviluppata aveva un primo visibile vantaggio: la struttura a nodi poteva essere distribuita in punti fisici diversi. L'importante era che la comunicazione tra nodi *Link* e *Router* potesse avvenire. Questo era garantito dalla connessione *WebSocket* stabilita, che era indipendente dalla posizione dei nodi.

3.2 I problemi con JDI

La soluzione sviluppata presupponeva un utilizzo di JDI che facesse uso della possibilità di distribuire i suoi nodi su più punti. Nella realtà però gli *stakeholder* di JMES hanno constatato che le esigenze dei clienti non prevedevano lo sfruttamento di questa caratteristica. Gli svantaggi che come vedremo ha portato la soluzione sviluppata erano molto maggiori dei possibili vantaggi commerciali del prodotto.

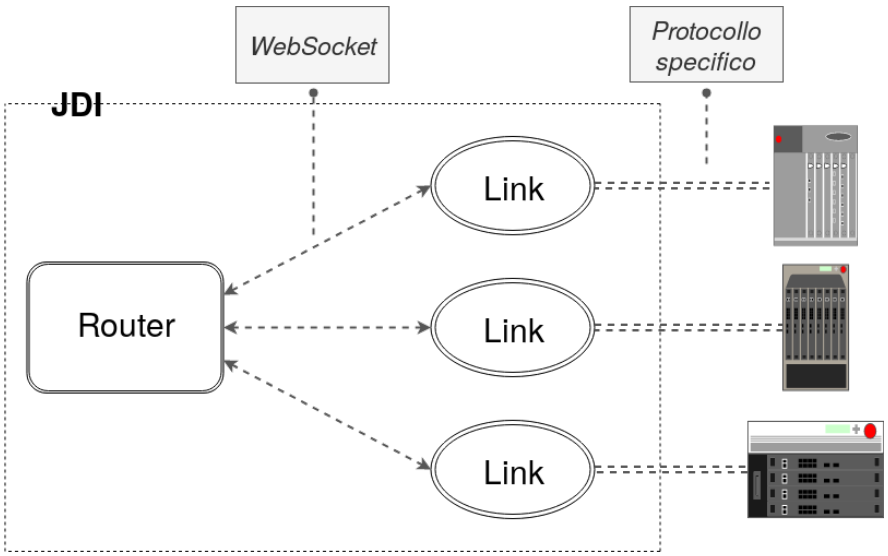

Figura 3.1: Architettura a nodi *Router*/*Link* della prima versione di JDI

A seguito dello sviluppo, sono iniziati i primi test del prodotto. Questi test dovevano verificare la stabilità del sistema e sopratutto la sua affidabilità nel lungo termine. Una volta installano in ambiente di produzione sarebbe infatti dovuto rimanere attivo per lunghi periodi, facilmente oltre l'anno.
I test prevedevano invece sessioni troppo brevi e con un numero di dati coinvolti nella trasmissione non realistici. Non è mai stata effettuata un'analisi dinamica sul consumo di risorse.

I primi problemi sono emersi in produzione, quando dopo pochi giorni di utilizzo il sistema presentava dei cali prestazionali e delle interruzioni impreviste.

A seguito di questi eventi avvenuti presso la sede di un cliente sono intervenuti i tecnici sistemisti del team JMES che hanno tentato di effettuare il *debug* del codice scoprendo numerosi problemi legati all'utilizzo di librerie esterne che rendevano inoltre difficoltoso analizzare le zone del codice in cui si verificavano degli errori.

Le problematiche emerse hanno sottolineato carenze che si riassumono in:

- basse prestazioni (velocità di trasmissione);
- scarsa affidabilità (stabilità del sistema a lungo termine);
- alto consumo di risorse;

- bassa manutenibilità.

3.3 Il nuovo sviluppo

Data l'esperienza precedente, la nuova versione di JDI, oggetto di questa tesi, è stata sottoposta a vincoli più rigidi.

A differenza della versione precedente, che avviava un servizio Windows per ogni nodo, il programma doveva essere eseguibile su un *web server* Apache Tomcat, limitare in consumo di risorse impiegate sopratutto dopo lunghi periodi di esecuzione, poter gestire un numero ragionevolmente alto di flussi di dati.

Inoltre visto lo scarso successo della versione distribuita di JDI, la nuova versione doveva sfruttare la concorrenza su singola JVM per adempiere ai compiti di gestione dei diversi flussi da e verso i PLC.

Un importante punto per JDI è che potesse essere facilmente manutenibile e che facesse uso il meno possibile di librerie esterne che invece regnavano nella prima versione.

Lo scopo in pratica era di raggruppare le funzionalità di base che vecchia versione offriva sotto forma di servizi Windows, in un ambiente *multi-threadded* e:

- garantirne la scalabilità;
- valutare quantitativamente le sue caratteristiche, attraverso *benchmark* di prototipi;
- rendere disponibile i dati ricavati con servizi web. Principalmente a JMES ma in generale a chi ne possa trarre vantaggio;
- storicizzare i dati raccolti.

Gli ultimi due punti sono obiettivi più a lungo termine per JDI, che non rientrano nel dominio dello stage svolto.

4

Analisi, Progettazione e Sviluppo

4.1 Studio della versione precedente

Nonostante il primo sviluppo di JDI non avesse avuto un risultato positivo una volta messo in produzione, le funzionalità di base di questo dovevano essere presenti anche nella nuova versione.
Per raccogliere i requisiti del progetto è stato utile avere un contatto diretto con lo sviluppatore che realizzò la prima versione di JDI. Mi è stato dato accesso al codice sorgente del progetto, che ho utilizzato principalmente per comprenderne il funzionamento interno. Questo mi ha permesso di:

- ricavare informazioni sull'uso delle tecnologie utilizzate;
- individuare componenti riutilizzabili.

Oltre a ciò sono avvenuti numerosi colloqui con le persone chiave per raccogliere i vecchi e i nuovi requisiti del progetto di stage. Come visto nel capitolo precedente, i problemi stavano nella scarsa soddisfazione di requisiti non funzionali, e non nel soddisfacimento di quelli funzionali.

Lo studio e l'analisi della versione precedente hanno impegnato le prime due settimane di lavoro.

Il codice sorgente nelle sue componenti principali constava di:

- otto progetti contenenti i driver per la comunicazione con i PLC. Ogni driver è un'implementazione diversa di un protocollo;
- librerie di base per JDI, contenenti definizioni di interfacce e classi astratte, strutture dati, ecc;
- progetti per i nodi *Router* e *Link*.

Dall'analisi del codice sono emersi due principali attori che hanno caratterizzato lo stile del codice:

- ReactiveX: una libreria per la programmazione asincrona e *event-based* in Java* che estende il *pattern Observer*. Largamente utilizzata per lo sviluppo di applicazioni Android e in generale dove esiste una gestione delle *UI*. Aggiunge astrazione alla gestione a basso livello di *thread*, sincronizzazione e strutture dati concorrenti;
- jawampa: come detto la comunicazione tra i nodi *Link* e *Router* avveniva tramite *WebSocket*. La libreria jawampa è l'implementazione di Web Application Messaging Protocol (WAMP), un protocollo secondario di WebSocket. WAMP unifica in un protocollo due *pattern* per lo scambio di messaggi: Remote Procedure Call (RPC) e *Publish & Subscribe*. Per l'implementazione di questo sub-protocollo è stata utilizzata la libreria open-source jawampa†.

Problemi con la libreria ReactiveX: i vantaggi di questa libreria sono sentiti se si gestiscono eventi provenienti da interfacce grafiche (e.g. la pressione di un pulsante) o se si fa uso di *Callback*. L'utilizzo rilevato nella prima versione di JDI non aveva a che fare con questi casi. Non è infatti prevista alcuna interfaccia grafica per JDI. Inoltre molte porzioni di codice sono risultate complicate dall'uso forzato della libreria. Un altro aspetto che non è stato considerato quando si è fatta la scelta di utilizzare tale libreria è che quasi mai chi scrive un *software* è poi colui che lo mantiene. Quando si sono verificati i primi problemi con JDI il codice non era facilmente interpretabile da chi non aveva mai scritto del codice ReactiveX.

Problemi con la libreria jawampa: durante le riunioni effettuate con i responsabili e gli sviluppatori del team JMES è emerso che i problemi dovuti alle basse prestazioni nella comunicazione tra i nodi erano causati da questa implementazione di WebSocket. Jawampa è inoltre una libreria non più mantenuta. Una conferma di ciò si può trovare nella *repository* pubblica su GitHub.

*Più informazioni su http://reactivex.io/intro.html
†Disponibile su https://github.com/Matthias247/jawampa

4.2 I requisiti del progetto

In questa sezione vengono raccolti i requisiti estrapolati dall'analisi della vecchia versione e dagli incontri svolti con i responsabili del team JMES. Particolare attenzione sarà posta sui requisiti non funzionali. Come detto sono stati il punto debole della versione precedente.

I requisiti possono essere espressi con diversi livelli di dettaglio, a seconda della loro destinazione. Ian Sommerville nel testo *Software Engineering* suggerisce due definizioni per effettuare una distinzione nel livello di dettaglio:

- *User requirements* (requisiti utente): definizioni con linguaggio naturale e con eventuali diagrammi più formali di cosa il sistema debba essere in grado di fare, dal punto di vista dell'utente. Questo sottintende una definizione ad alto livello della funzionalità. Queste definizioni sono adatte ad un lettore delle specifiche non interessato ai dettagli tecnico del requisito;
- *System requirements* (requisiti di sistema): sono descrizioni più dettagliate delle funzionalità, dei vincoli e dei servizi del sistema. Definiscono nel dettaglio cosa deve essere sviluppato (non definiscono comunque il "come").

L'astrazione dai dettagli fornita dai requisiti utente è utile quando la documentazione delle specifiche è indirizzata a lettori non esperti. Nell'ambito di questo stage invece ho collaborato esclusivamente con sviluppatori e responsabili tecnici che non necessitavano di filtri nella descrizione delle specifiche. I requisiti raccolti di seguito saranno quindi da considerarsi *system requirements*.

I requisiti descritti nel resto della sezione sono raccolti in tre categorie:

- Requisiti funzionali: descrivono cosa il sistema dovrebbe fare;
- Requisiti qualitativi;
- Requisiti di vincolo.

Per tenere traccia dei requisiti, per ognuno verrà usata la seguente codifica:

$$\mathrm{R[F \mid Q \mid V][Codice]}$$

dove la prima opzione specificata indica un requisito rispettivamente funzionale, qualitativo o di vincolo. La seconda opzione è un identificativo numerico intero crescente.

Codice requisito	Nome	Descrizione
RF1	Configurazione del sistema	Il sistema deve ricavare le informazioni riguardanti la configurazione delle macchine ad esso collegate da un file JSON. Questo file conterrà le seguenti informazioni: • indirizzo IP, porta e ID dei PLC con cui il sistema deve interagire; • per ogni PLC, le locazioni di memoria con cui interagire; • come interpretare i dati presenti nelle locazioni di memoria (numero intero, numero reale, stringa ecc..); • le modalità di accesso alle locazioni di memoria: lettura e scrittura.
RF2	Riconfigurazione	L'interfaccia delle API del sistema deve presentare un comando per scatenare la riconfigurazione. questa deve poter essere effettuata senza riavviare l'applicazione
RF3	Operazioni di lettura	Il sistema deve essere in grado di eseguire operazioni di lettura verso i PLC collegati. La lettura deve avvenire a intervalli regolari. I tempi di lettura possono variare per ogni PLC collegato. Questa informazione deve essere presente nel file di configurazione

RF4	Operazioni di scrittura	Il sistema deve essere in grado di eseguire operazioni di scrittura verso i PLC collegati. La scrittura può avvenire solo se l'area di memoria interessata ha i permessi di lettura, specificati nel file di configurazione.
RF3	Richiesta scrittura	le API del prodotto devono permettere di richiedere la scrittura di un valore su una specifica area di memoria di uno specifico PLC. La richiesta deve rispettare i permessi definiti nel file di configurazione
RF3	Salvataggio dati	Il sistema deve mantenere in memoria i valori aggiornati dei dati letti ad ogni ciclo di lettura.
RF3	545	778
RF3	Rilevazione malfunzionamento (watchdog)	Per rilevare il malfunzionamento di un PLC il sistema deve gestire un particolare bit all'interno di ogni PLC con cui è collegato: prima di effettuare un'operazione di lettura, il sistema verifica (con una lettura) che il bit si trovi al valore 1. In tal caso il sistema provvede a resettare tale bit a 0. In caso contrario invece il sistema segnala la situazione anomala
RV1	Linguaggi	Il linguaggio per lo sviluppo del sistema JDI deve essere Java. Non sono posti vincoli sulla versione
RV2	Concorrenza	Deve essere utilizzato un approccio concorrente per realizzare il prodotto.

RQ1	Reattività scrittura	Quando avviene una richiesta di scrittura, il sistema deve prendere in carico la gestione di questa nel più breve tempo possibile
RV3	Richieste multiple di scrittura	Se vengono richieste nel breve periodo multiple scritture, il sistema deve soddisfarle tutte (quando sono disponibili i giusti permessi) mantenendo l'ordine di arrivo delle richieste
RV3	Librerie	Le dipendenze del progetto devono limitarsi allo stretto necessario per portare a termine gli obiettivi fissati. L'introduzione di una libreria deve essere approvata da un responsabile
RQ4	Scalabilità sistema	Il sistema deve presentare un aumento lineare nel consumo delle risorse fino
RQ5	Consumo memoria	Il sistema deve occupare non più di 2GB di memoria in una situazione reale: 5 PLC collegati con diverse operazioni di lettura e scrittura che occorrono a intervalli di pochi secondi
RV5	Compatibilità PLC	Il sistema deve poter funzionare con dispositivi *Schneider* che utilizzano il protocollo *ModbusTCP*
RV6	Svolgimento test #1	Il sistema deve essere testato con almeno un dispositivo PLC *Schneider* M221
RV6	Svolgimento test #2	Per i test di scalabilità del sistema devono essere simulati dei dispositivi PLC con protocollo *ModbusTCP* con l'uso del software *Modbus PLC Simulator*

4.3 Metodologia di lavoro

Way of working utilizzato durante il progetto, interazione con il team JMES di cui ero parte ecc..

4.4 Tracer bullet development

Definire questo progetto un prototipo sarebbe errato, infatti spesso questi sviluppi servono solo a dimostrare la fattibilità di un progetto e il codice utilizzato viene quasi sempre scartato.
Il lavoro svolto invece è stato mirato alla creazione e all'implementazione di un'architettura solida, scalabile e affidabile della nuova versione di JDI, l'opposto quindi di un prototipo!
Questo tipo di sviluppo mira a utilizzare la maggior percentuale possibile di tecnologie per stendere una prima strada verso lo sviluppo completo. Questa metodo prende il nome di *tracer bullet development*.

Si parlerà principalmente di:

- lavoro di integrazione iniziale delle tecnologie e degli strumenti utilizzati durante lo sviluppo.
- la progettazione di prototipi e l'analisi di prestazione con benchmark per testare le prestazioni prima ancora di iniziare lo sviluppo intensivo;
- la scelta del prototipo
- lo sviluppo
- i test.

5
Retrospettiva

Glossario

business unit Un sottoinsieme dell'impresa che rappresenta un business specifico concentrato su una particolare linea di prodotti.. 2

digital industry Relativamente all'industria 4.0, ci si riferisce al concetto di fabbriche dove le macchine sono potenziate con connettività *wireless* e sensori, connesse a un sistema che può tenere sotto controllo l'intera filiera produttiva e compiere delle decisioni sulla base dei dati raccolti autonomamente.. 2

outsourcing L'appalto a una società esterna di determinate funzioni o servizi, o anche di interi processi produttivi.. 2

Acronimi

CRS Centro Ricerca e Sviluppo. 1, 2

SMI Sanmarco Informatica. 1, 2