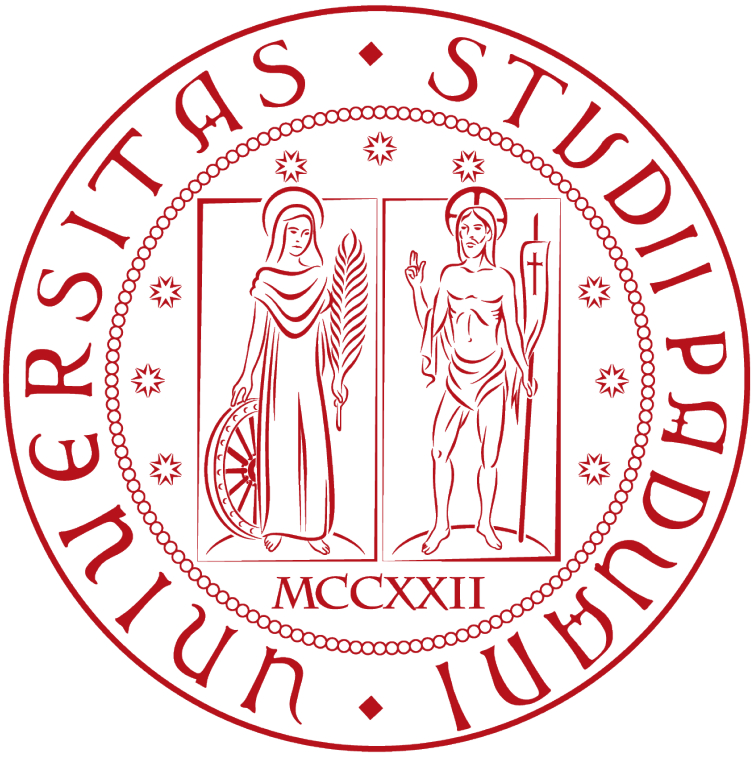

Università degli studi di Padova
Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica
A.A. 2017-2018

Analisi esplorativa software SCADA-HMI

Laureando:
Salvatore Pilò
Matricola: 1049593

Relatore:
Prof. Gilberto Filè

Alla mia famiglia che nonostante tutto non ha mai smesso di supportarmi.

A mio nonno che sarebbe orgoglioso di me.

Ad Elisa, senza la quale non ce l'avrei mai fatta.

A Marco, Matteo, Victor, Borto e Manuel senza i quali non saprei gran parte delle cose che so oggi.

A Paolo, a Valentino, a Mario, Nicola, Marzia ed Emanuele che mi hanno supportato sempre nelle tante(troppe) cose che facevo.

Ad Alessandro e Beniamino per aversi reso sempre felice.

Ad Alex, Matteo e Leo che sono riusciti a farmi innamorare di qualcosa che non ritenevo possibile.

A Gianluca, Nunzio, Maria e alla Masnada senza i quali non sarei quello che sono io oggi.

A Martina.

Ad Anakin Skywalker, senza il cui sacrificio saremmo ancora sotto il dominio dell'Impero.

Sommario

Il presente documento descrive obiettivi e attività dello stage curriculare svolto dal laureando Salvatore Pilò presso l'azienda **SanMarco Informatica S.p.A** .
La durata complessiva dello stage è stata di circa 320 ore (40 gg).
Lo scopo principale dello stage è stato quello di esplorare i sistemi SCADA ed effetturare uno studio di fattibilità, soprattutto in merito all'ambito HMI(*Human Machine Interfaces*).
La maggior difficoltà riscontrata durante lo studio di fattibilità si è dimostratata essere la ricerca di una liberia grafica che consentisse lo sviluppo del sistema in analisi.
Il passo successivo allo studio di fattibilità è stato quello di sviluppare un prototipo funzionante che includesse al suo interno le funzionalità minime richieste da un sistema di questo tipo.
Nel resto del documento saranno descritti tutti i concetti alla base di un sistema SCADA e veranno prese in analisi le tecnologie utilizzate e i problemi affrontati durante lo sviluppo.
Tutte le descrizioni dei sistemi SCADA che verranno fornite nelle successive sezioni sono da considerarsi come una panoramica generale sui requisiti di un sistema SCADA.
Tale analisi è stata utile per comprendere al meglio il tipo di software da sviluppare e le caratteristiche e funzionalità da integrare all'interno dello stesso.
Non è da considerarsi come un'analisi approfondita e specifica di tali sistemi poiché non era negli interessi di **SanMarco Informatica S.p.A** ottenere uno studio tanto specifico.
Verranno inoltre forniti alcuni esempi applicativi di un software SCADA e alcuni frammenti del codice (i più significativi) del prototitpo sviluppato durante la permanenza in azienda.

Convenzioni tipografiche

- Le parole in lingua inglese senza corrispettivo italiano saranno scritte in *corsivo*;
- Le parole che necessitano di una spiegazione esplicita sono marcate da una **g** apice per segnalarne la presenza nel **Glossario** a fine documento.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

Lo stage, svoltosi presso l'azienda **SanMarco Informatica S.p.A** e con la supervisione del tutor Alex Beggiato , consisteva nell'esplorazione dei sistemi **SCADA-HMI**[g] al fine di poter comprendere se espandersi o meno su quel particolare settore.
In particolare lo scopo dello stage era quello di esplorare le tecnologie necessarie allo sviluppo di questo tipo di sistemi e analizzarne costi e funzionalità.
In ultimo, ma non meno importante, era richiesto lo sviluppo di un prototipo embroniale che potesse mostrare l'utilità delle tecnologie selezionate dallo stagista.
Il sistema richiesto doveva avere le caratteristiche tipiche di un sistema **SCADA-HMI**[g] anche se doveva essere maggiormente orientato allo sviluppo di un Dialogo Operatore[g] più funzionale e interattivo che comprendesse al suo interno molte delle funzionalità tipiche di uno **SCADA-HMI**[g] .

1.1 L'Azienda

Figura 1: Logo SanMarco Informatica - Fonte: https://www.sanmarcoinformatica.com/

L'azienda nasce negli anni '80 con lo scopo di sviluppare software gestionali per aziende manufatturiere e, ad oggi, è una leading company sul suolo italiano nello sviluppo di soluzioni dedite alla riorganizzazioni di processi aziendali e professionali.
Partner di IBM Italia, vata 320 dipendenti circa, 13 distributori e 4 sedi: quella centrale a Grisignano di Zocco e tre filiali, a Reggio-Emilia, Udine e Monza-Brianza.
Le *bussines unit* di **SanMarco Informatica S.p.A** sono 5, **Jgalileo**, una soluzione ERP[g] per aziende manufatturiere, **4words**, *web apps marketing solutions*, **NextBI**, analisi dei dati dei clienti per un'ottimizzazione dei processi di marketing, **Discover Quality**, soluzioni in ambito gestione documentale, qualità, sicurezza e ambiente, e **SMItech**, soluzioni dedicate a progetti di infrastruttura IT e lo sviluppo di servizi gestiti di Cybersecurity.
SanMarco Informatica S.p.A è, inoltre, la prima azienda italiana entrata a far parte delll'Open Power Foundation IBM [1].

1.2 Obiettivi di stage

L'obiettivo dello stage era di effettuare uno studio di fattibilità in merito all'idea di ampliare il parco software sviluppato da **SanMarco Informatica S.p.A** con un sistema di tipo **SCADA-HMI**[g] .
SanMarco Informatica S.p.A aveva già, all'interno del suo parco software, sistemi in grado di rispondere all'esigenza di monitorare i processi di produzione all'interno di un contesto industriale. Proprio per questa sua affinità verso questa tipologia di software e vista l'esigenza espressa dai vari clienti dell'azienda di avere a che fare con applicativi sempre più complessi e completi, **SanMarco Informatica S.p.A** ha valutato la possibilità di espandersi anche nel settore degli **SCADA-HMI**[g] .
I software **SCADA-HMI**[g] per loro natura si inseriscono in maniera quasi naturale nella famiglia dei sistemi **MES** visto che condividono assieme numerosi aspetti a partire dalla supervisione dei processi di produzione. Se si fosse realizzata la fattibilità del progetto, allora questo sarebbe diventato un modulo di **JMES**, uno dei software attualmente in sviluppo

presso **SanMarco Informatica S.p.A** , e sarebbe stato focalizzato sull'ambito produttivo/metalmeccanico.
L'esigenza di questo tipo di software all'interno dell'ambito metalmeccanico, nasce, nello specifico, dalla necessità di visualizzare lo stato corrente della linea/impianto, in particolare evidenziando eventuali fermi/sospensioni/allarmi.
Una volta consolidate le conoscenze del tipo di sistema da sviluppare era richiesta una definizione dei requisiti e quindi la ricerca di una libreria grafica utile allo sviluppo del sistema analizzato, in particolare in merito all'HMI.
Lo scopo principale, dopo aver reperito tutti gli strumenti necessari, era quello di valutarne, appunto, la fattibilità in termini di costi per l'azienda.
Era necessario lo sviluppo anche di un prototipo funzionante che dimostrasse l'effettiva fattibilità del progetto e la sua possibile integrazione con il prodotto **JMES** già presente all'interno della suite aziendale.
Durante il corso dello stage si è ritenuto opportuno alleggerire, in un certo senso, i requisiti stretti imposti dai sistemi **SCADA-HMI**^{g} in favore di un sistema che ricalcasse solo i principi generalidi questi schemi.
In sintesi, il sistema risultante doveva essere un aiuto per il **Dialogo Operatore** già presente all'interno di **JMES** che mostrasse le varie macchine, e le interconessioni fra esse, i loro stati ed eventuali allarmi.
Durante il periodo di stage però ci si è resi conto di dover apportare qualche cambiamento agli obiettivi del progetto per via di alcune limitazioni hardware aziendali.
Tali criticità sono emerse durante una riunione all'interno dell'azienda di cui si parlerà nella sezione di Analisi di questo elaborato.
Gli obiettivi sono quindi stati cambiati solo per quel che rigarda il prototitpo finale che non avrebbe dovuto rispettare pienamente tutti i principi degli **SCADA-HMI**^{g} ma essere conforme solo ad una parte di questi, quelli che realizzano l'HMI.
Il progetto finale, inoltre, doveva prevedere un rudimentale editor delle forme, per permettere all'operatore di disegnare letteralmente le macchine del sistema, e un rudimentale editor di un impianto.
Ovviamente è stata prevista anche una simulazione, con input generati

manualmente, che mostrasse il corretto funzionamento dell'impianto e la corretta visualizzazione delle informazioni.
Era richiesta, inoltre, la perfetta integrazione del sistema con tutte le tecnologie di **SanMarco Informatica S.p.A** sul lato *server* e con *AngularJS*, tecnologia utilizzata da **SanMarco Informatica S.p.A** per il lato *client*.

1.3 Principali problematiche

I principali problemi emersi durante il periodo di stage sono emersi quasi del tutto al primo periodo di analisi all'interno del quale è stato piuttosto complesso riuscire a tracciare un profilo delle tecnologie da utilizzare che ben si integrassero con i sistemi già in uso in azienda. L'aiuto di Alex Beggiato è stato fondamentale all'interno del processo di ricerca vista la scarsa conoscenza dei sistemi tecologici utilizzati a **SanMarco Informatica S.p.A** .
Grazie a lui, e al team in cui ero inserito, ho imparato il corretto funzionamento degli strumenti già presenti in azienda per poi capire che tipo di tecnologie di terze parti integrare, se farlo o se fosse il caso di creare *ex novo* alcuni strumenti.

1.4 L'idea - Sistemi MES

L'Information Technology[g] , grazie agli ultimi progressi tecnologici e le ultime evoluzioni di internet e del mobile, ha acquisito un ruolo sempre più importante all'interno dell'industria divenendo quasi un aspetto fondamentale all'interno del concetto di fabbrica moderna.
La diffusione di tali tecnologie ha permesso di collegare fra loro diversi macchinari e strutture con un notevole aumento dei flussi informativi, pertanto una corretta gestione di tutte le informazioni generate diventa una condizione necessaria per il successo dell'impresa.
L'informazione diviene, infatti, un concetto fondamentale e riveste un

ruolo chiave all'interno dell'amministrazione aziendale.
Proprio per far fronte a questo nuovo tipo di industria, che presenta problemi non più risolvibili con i sistemi tradizionali, si è sviluppato un nuovo tipo di sistema che supporta e gestisce tutta la linea di produzione con informazioni precise, trasparenti e veloci, il cosidetto sistema MES (*Manufacturing Execution System*).
Un sistema MES, così come definito da MESA (*Manufacturing Enterprise Solutions Association International*) è un sistema che consente l'ottimizzazione delle attività di produzione, dal lancio dell'ordine al prodotto finito [2].
Possiamo definire un software MES, quindi, come una tecnologia in grado di raccogliere, standardizzare e organizzare informazioni provenienti dal campo che molto spesso restano isolate, cartacee, poco precise e senza nessun tipo di supporto.
Un sistema MES deve raggiungere tre principali Obiettivi:

- Fornire un modello reale per la pianificazione
- Ottimizzare le operazioni manifatturiere
- Aumentare l'integrazione con il livello di business.

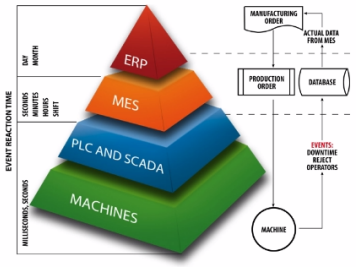

Figura 2: Ruolo MES - Fonte: http://www.info-centric.com/

Fra i moduli di **Jgalileo**, la soluzione ERP[g] di **SanMarco Informatica S.p.A** , è presente **JMES**, il sistema MES sviluppato da **SanMarco Informatica S.p.A** e completamente integrato con **Jgalileo**.
Il modulo consente un controllo efficace dell'impresa mediante la raccolta e l'analisi dei dati di fabbrica.
JMES è un sistema che controlla, in tempo reale, le attività svolte da

tutti i reparti, quindi monitora le risorse produttive, verifica lo stato di avanzamento degli ordini e analizza i consuntivi.
Il sistema viene utilizzato anche all'interno degli uffici per verificare avanzamento, produttività e stato delle macchine.
JMES si compone, essenzialmente, di due componenti:

- **Dialogo Operatore**[g]
  Utilizzato all'interno dello stabilimento per verificare e mostrare lo stato di avanzamento della produzione attraverso un *browser*, su PC, o in modalità *touch* su *tablet* e *smartphone*.
- **Gestore**
  Utilizzato per eseguire tutte le operazioni di impostazione delle anagrafiche e delle tabelle, le manutenzioni dei dati inseriti, la contabilizzazione delle rilevazioni, l'esportazione della registrazione presenze, il controllo dello stato delle risorse produttive, le interrogazioni e l'analisi delle statistiche.

Osservando nel dettaglio il **Dialogo Operatore**[g] possiamo osservare differenti funzionalità utili per fruire correttamente e agevolmente di tutte le informazioni disponili.

- **Gestione della presenza del personale** che lavora lungo le linee di produzione e coordinazione delle squadre dinamiche con la possibilità di conoscere e impostare quando il turno lavorativo inizio e termina, considerando anche i vincoli e requisiti richiesti dalle necessità del piano di produzione
- **Dichiarazione e documentazione delle attività** svolte da operatori, da macchinari e da squadre e possibilità di tenere traccia quando un operatore inizia e finisce l'attrezzaggio, del ciclo della macchina, delle pause, sospensioni e fermi impianto; il tutto è visualizzabile mediante una cronologia degli eventi
- **Dichiarazione quantità prodotte**, scartate, non conformi e dichiarazione dell' avanzamento al solo fine di attestare le quantità prodotte a riempimento del contenitore

- **Visualizzazione e gestione del piano lavori** e delle attività da svolgere presso un macchinario, in funzione delle vendite effettuate o di target specifici, il tutto elaborato con l'ausilio ERP Jgalileo e ordinato per data-ora
- **Dichiarazione su bolle aggraffate** (e lotto di produzione) e gestione e conteggio della produzione di pezzi e lotti appartenenti anche a ordini e progetti diversi
- **Gestione di attività indirette** (manutenzione, pulizia, etc.) e di attività non pianificate (come rilavorazioni e fasi aggiuntive) con possibilità di creazione automatica di fasi non previste nell'ordine di produzione
- **Dichiarazione interattiva dei componenti prelevati e dei prodotti versati** (quantità effettiva prelevata/versata, lotti, UDC) con relativa stampa dell'etichetta del contenitore
- **Visualizzazione delle informazioni tecniche e documentali** (disegni, fotografie, video, istruzioni di lavoro, specifiche tecniche, schemi di attrezzatura, etc.)
- **Messaggistica interna ed esterna** (via mail) per comunicazioni tra operatore, capoturno, caporeparto, manutentore, etc.

1.5 L'idea - Sistemi SCADA [3]

1.5.1 Introduzione

I sistemi **SCADA-HMI**^{g} nascono come una risposta all'esigenza, per le industrie, di monitorare l'impianto nella sua totalità, di poter controllare in remoto (solo per le operazioni più sempici e basilari) le macchine e poter gestire rudimentalmente gli allarmi generati dagli stessi.
Prima dell'avvento di **SCADA-HMI**^{g} , questo tipo di attività era svolta grazie all'ausilio di circuiti hardware imponenti sui quali erano posti vari LED (figura 3).
L'operatore incaricato di osservare l'impianto era tenuto, quindi, ad osservare un sistema molto complesso, lontano (nella forma) dall'impianto

reale e a dover ricordare, in qualche modo, tutta la storia degli allarmi e degli stati assunti dal sistema nel tempo.

Figura 3: Plant Montoring System - Fonte: https://www.shutterstock.com

Per far fronte all'esigenza di supervisionare in maniera più semplice gli impianti, per avere accesso ad uno storico preciso e puntuale degli stati dello stesso e per poterlo controllare in modo più efficace nascono i sistemi **SCADA-HMI**[g] .
Un sistema **SCADA-HMI**[g] si compone, essenzialmente, di tre attività:

- **Acquisizione Dati**
  Ha un ruolo di supporto.
  Si tratta di una semplice funzione di scambio di dati puro, senza nessun potere decisionale da parte del sistema sulla struttura dalla quale di acquisiscono le informazioni.

- **Supervisione**
  Si tratta della funzionalità "core" di un sistema **SCADA-HMI**[g] che permette, appunto, di monitorare un processo e l'evoluzione degli stati dello stesso. Tutte le funzionalità di visualizzazione degli stati, di gestione degli stessi, di visualizzazione degli storici dei dati e gestione degli stessi sono da relegare all'interno di questa attività. L'attività di supervisione è caratterizzante per un software **SCADA-HMI**[g] che non può essere definito tale senza di essa.

- **Controllo**
  Si tratta delle funzionalità che permettono di agire su un dato processo alterandone lo stato.

Queste funzionalità utilizzano lo stesso canale utilizzato dai processi di acquisizione. La differenza sta nella direzione del flusso informativo.
Se nel primo caso "leggiamo" semplicemente dei dati, nel caso di un'attività di controllo utilizziamo il canale per inviare degli ordini così da alterare lo stato dei processi che si stanno monitorando modificando uno o più parametri degli stessi.

1.5.2 Architettura

Possiamo ricondurre l'architettura di un sistema **SCADA-HMI**[g] a tre livelli fondamentali.
In basso abbiamo tutte le strutture di acquisizione dati che inviano informazioni al livello superiore, il sistema di trasmissione dati, che invia a sua volta l'informazione al livello più in alto che comprende tutto il sistema di elaborazione che processa l'informazione e, una volta elaborata può spedirla nuovamente ai livelli inferiori.
Un esempio concreto di questo funzionamento è quello del monitoraggio di un impianto di conta dei pezzi prodotti.
La macchina, il conta-pezzi, ad ogni pezzo prodotto "attiva" un PLC[g] che genera un'informazione compatta che viene spedita al livello più basso dell'architettura, il sistema di acquisizione.
Questo invia l'informazione (una stringa che indica il nome della macchina, il tempo di acquisizione, l'identificativo del pezzo prodotto) al livello superiore, il sistema di trasmissione.
Questo livello è semplicemente un livello "di transito" che non fa altro che inviare l'informazione al livello superiore, il sistema di elaborazione.
Qui la stringa compatta viene decodificata ed elaborata in modo tale da generare un dato comprensibile all'operatore. Tale dato viene rispedito in basso fino a tornare al livello di acquisizione all'interno del quale il dato verrà mostrato, elaborato, all'utente finale.
Si noti che nel caso di sistemi **SCADA-HMI**[g] che presentano una **HMI** in grado semplicemente di mostrare gli stati dei processi del si-

stema osservato, il sistema di elaborazione ha il solo scopo di gestire la visualizzazione degli stati perdendo, quindi, la funzionalità di elaborazione delle informazioni per alterare lo stato dei sistemi osservati.

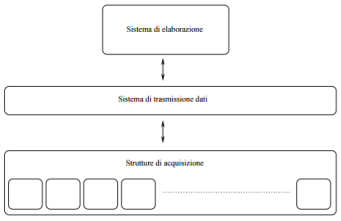

Figura 4: Architettura SCADA - Fonte: http://www.apogeonline.com/libri/88-503-1042-0/scheda

Apparato di acqusizione dati

Si tratta, essenzialmente, dell'apparato che consente al sistema SCADA di comunicare con il mondo reale circostante. In sintesi "traduce" le informazioni derivanti dal mondo reale in informazioni riconoscibili e processabili dal sistema stesso. Le informazioni gestite possono essere divise in tre macrocategorie, anche se ognuna di esse contiene svariati sottogruppi. Tale divisione è molto semplicistica ma mira a comprendere al meglio il tipo di informazione che può essere gestita dal sistema.
La tipologia di queste può essere classificata quindi in tre gruppi:

- **Direzione delle informazioni**
  Possiamo suddividere questo tipo di informazione in due macrogruppi, in entrata o in uscita
- **Qualità delle informazioni**
  Natura delle informazioni ricevute e/o inviate.
  Possiamo operare un'ulteriore distinzione in:
  - **Informazioni digitali**
    Sono una rappresentazione della grandezza attraverso stati digitali ben distinti fra loro e riconducibili a degli "stati logici". Un esempio banale è dato dallo stato di un motore che può essere acceso (ON) o spento (OFF). Tali informazioni possono

essere estrapolate grazie a delle porte logiche che, attraverso una carica elettrica, informano il sistema sullo stato dell'apparecchio.
È possibile combinare più stati logici fra di loro per avere delle informazioni più dettagliate e specifiche. Ad esempio, avere l'informazione del motore spento non ci da nessuna informazione sulla causa del fermo. È possibile, infatti, che lo stesso sia in funzione e che si sia verificato semplicemente un guasto alla connessione al sistema su quella particolare porta.
Con l'aggiunta di ulteriori collegamenti è possibile ricreare più combinazioni per ottenere quindi informazioni più specifiche a riguardo, che possono segnalarci, ad esempio, il guasto descrito sopra.
Le informazioni digitali possono essere sia in ingresso al sistema, telesegnali(TS) o *digital input*(DI), oppure in uscita, telecomandi (TC) o *digital output*(DO)

– **Informazioni analogiche**
Ovviamente non tutte le informazioni sono interpretabili come digitali. Una grandezza fisica come la temperatura non è riconducibile a nessun tipo di stato logico pertanto è necessario un sistema di traduzione che traduca l'informazione arrivata alla porta in ingresso applicando una certa tensione, in un dato memorizzabile e processabile dal sistema.
Tali informazioni possono essere sia in ingresso al sistema, telemisure(TM) o *analog input*(AI), oppure in uscita, teleregolazioni(TR) o *analog output*(AO)

– **Informazioni impulsive**
Si tratta di un tipo di informazione che necessita di una continuità nel tempo per poter essere correttamente elaborata. Un esempio banale è un contatore elettrico che necessita di un'analisi su un certo periodo per poter estrapolare il risultato corretto della sua informazione.
Tali informazioni vengono immagazzinate tramite un registro di accumulo degli impulsi che viene tradotto ed elaborato dal

sistema **SCADA-HMI**[g] .
Anche questo tipo di informazioni può essere utilizzato sia in ingresso che in uscita

– **Informazioni complesse**
A volta per la comunicazione e comprensione del mondo esterno, un sistema SCADA non può semplicemente interfacciarsi con il sistema monitorato per avere a disposizione le informazioni che cerca. Se si immagina un sistema molto complesso che può dare in uscita parecchie informazioni risulta sconveniente creare delle porte di uscita (e di entrata) per ognuna di queste. Per questo esistono particolari apparecchiature che fanno da "ponte" fra il sistema monitorato e il sistema **SCADA-HMI**[g] .
Tali tipi di informazioni, prodotte da queste apparecchiature di mezzo, portano a particolari configurazioni sia dei "ponti" che del software di monitoraggio stesso. Questa loro particolare natura determinata dalla mole di informazioni da passare e dalla configurazione necessarie le rende differenti da tutte le altre, pertanto prendono il nome di "complesse".

- **Caratteristiche elettriche**
Ovviamente ogni informazione è trasmessa attraverso un segnale elettrico che può avere caratteristiche diverse in modo da rappresentare l'informazione in modo appropriato.
Le caratteristiche elettriche dell'informazione sono quindi necessarie sia per la traduzione che per l'elaborazione della stessa.

Protocolli di comunicazione

Le **infrastrutture di comunicazione** sono, chiaramente, differenti a seconda del sistema che si vuole sviluppare. Sistemi differenti necessitano di protocolli e infrastrutture di comunicazione particolari in grado di gestire al meglio le esigenze del sistema stesso.
Fra i parametri da tenere in considerazione per quel che riguarda le infrastrutture di comunicazione abbiamo la **velocità** che serve per gestire al

meglio svariate esigenze. Generalmente ci sono vari fattori, fissi, da tenere in considerazione come l'instaurazione di un canale di comunicazione fra i vari sistemi. Un fattore determinate è l'interazione con l'operatore e l'HMI che, nei cosiddetti sistemi *real-time* deve consentire comunque una lettura agevole dei parametri a schermo.
La velocità è un fattore che va tenuto in analisi, quindi, per necessità così da modellare un sistema in grado di rispondere perfettamente alle esigenze di comunicazione.
I **protocolli di sicurezza** nelle comunicazioni nascono dall'esigenza di rendere un sistema "sicuro" sia dal punto di vista delle intrusioni dolose sia dal punto di vista dell'errore umano che può influire sui dati letti e analizzati.
La gestione della sicurezza va applicata non solo alle comunicazioni fra il sistema ed eventuali apparati periferici ma anche, e soprattutto, alle comunicazione fra il sistema e le unità di controllo che hanno il potere di manipolare dati e informazioni.
Nell'analisi delle tecnologie da utilizzare per progettare il sistema nell'ambito dei protocolli di comunicazione, è necessario valutare con cura anche i servizi supportati dagli stessi sopratutto quelli riguardanti la qualità della comunicazione.
Se in un primo momento questo tipo di servizi possono risultare accessori, risulta palese quanto sia importante avere a che fare con "sistemi" automatizzati che garantiscano la qualità della comunicazione fra le varie infrastrutture del sistema.

Sistema di elaborazione

I tre elementi fondamentali in cui si organizza il sistema di elaborazione (figura 5) di un sistema SCADA sono la gestione dei dati, l'elaborazione e la disponibilità dell'informazione. La gestione dei dati, di cui si discuterà approfonditamente nella sezione in basso, racchiude tutte quelle funzionalità che si occupano di creare un insieme di dati intelligibili e affidabili pronti per poter essere archiviati.
L'elaborazione racchiude tutte quelle funzionalità che si occupano di in-

terpretare correttamente i dati come un insieme rappresentativo dello stato di un processo.

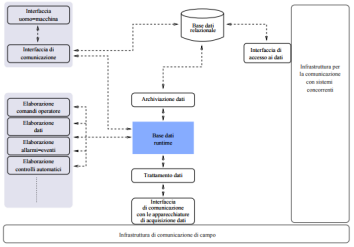

Figura 5: Sistema di elaborazione - Fonte: http://www.apogeonline.com/libri/88-503-1042-0/scheda

Per disponibilità dell'informazione si intende la produzione di informazioni destinate ad essere fruite da altri eventuali sistemi o dall'operatore stesso.

Gestione dei dati

La gestione dei dati è una delle funzionalità "core" di un sistema di supervisione. Il suo funzionamento è piuttosto "semplice" poiché prende dei dati in ingresso, arrivati dai vari sistemi periferici, e li processa per uniformare l'informazione agli standard interni del sistema **SCADA-HMI**[g] in modo da uniformarne la lettura e la gestione. Un esempio di tale funzionamento è il seguente.

Il PLC[g] i-esimo legge i vari dati in entrata, da tutti i suoi *slave*, che saranno inseriti all'interno di una sequenza di bit che sono caratterizzati da un codice univoco della macchina a cui si riferisce l'informazione e poi da una sequenza di informazioni in merito alla macchina osservata. Il sistema di elaborazione, tramite la funzionalità di gestione dei dati, deve processare tali sequenze di bit e uniformarle al sistema. Se, ad esempio, ogni sequenza di bit contiene l'orario di inizio delle sue funzionalità e il sistema immagazzina il dato con un preciso formato orario, queste verranno uniformate a tale formato.

La gestione dei dati è soggetta però ad un vincolo piuttosto stretto che è

definito dai tempi di accesso limitati: l'elaborazione dei dati deve essere compatibile con le evoluzioni costanti del processo quindi deve essere in grado di servire le richieste adattandosi alle tempistiche dettate dal sistema.
Utilizzando impropriamente il termine *realtime* possiamo dire che la funzionalità di gestione dei dati del processo di supervisione deve agire in tempo reale rispetto al sistema osservato per garantire una corretta fruizione delle informazioni all'operatore e per mantenere l'integrità delle stesse all'interno del sistema **SCADA-HMI**$^{\mathbf{g}}$.
Tale rappresentazione di questa base di dati è detta *database runtime* che rappresenta una sorgente di informazione costante per tutti i processi di elaborazione e rappresentazione dell'informazione. Una fra le soluzioni tecnologiche adottabili è un'area di memoria fisica condivisa dai processi che insistono sui dati in essa contenuti.
All'interno di un sistema **SCADA-HMI**$^{\mathbf{g}}$ quindi, si preferisce un sistema di base di dati relazionale per via della sua particolare natura. Una base dati relazionale presenta una grande razionalità nell'organizzazione dei dati, la semplicità di interrogazione al database, la completezza dell'informazione e la possibilità di associarla ad informazioni di sistemi concorrenti. Questo consente di associare, quindi, informazioni fra di loro slegate ma che è importante mettere in relazione per una corretta analisi.
Il database relazionale diventa quindi il canale di comunicazione fra il sistema di controllo e il mondo esterno.

Processi di elaborazione

Questa è una delle funzionalità più importanti dopo quella della gestione dei dati poiché risponde ad un'esigenza piuttosto particolare che è dovuta all'impossibilità da parte di un operatore di processare ed elaborare tutte le informazioni derivate dal sistema osservato e di inviare, allo stesso tempo, dei comandi che ne alterino lo stato dei processi. Nel dettaglio, i processi di calcolo ed elaborazione rispondono a delle esigenze precise e particolari che mirano non solo ad aiutare l'operatore ma a garantire

anche, e soprattutto, una gestione ottimale del sistema osservato che prescinda dall'intervento umano.
I processi di elaborazione mirano, quindi, a:

- Generare rappresentazioni sintetiche dello stato di un processo
- Generazioni di anomalie all'interno dell'evoluzione di uno stato di un processo
- Interpretazione del comando dell'operatore
- Produzione di informazioni aggregate che rappresentano la tendenza del processo
- Generazione di procedure automatiche di controllo.

Queste ultime sono proprio quelle che mirano a gestire il sistema osservato a prescindere dalle operazioni "esterne" dell'operatore che potrebbero non essere tempestive quanto servirebbe in talune situazioni in cui, l'automatismo di certe forme di controllo, rende l'intero sistema più sicuro e affidabile.

Condivisione dell'informazione

Proprio al fine di limitare gli interventi umani allo stretto indispensabile, quindi limitarli a quelle operazioni che non sarebbero possibili altrimenti, è necessario fornire una funzionalità di condivisione dell'informazione. La condivisione dell'informazione, all'interno di questi sistemi, si riferisce alla possibilità di controllo da parte di altri sistemi automatici che, per poter esercitare una funzione di controllo, necessitano di avere condivise le informazioni. Devono poter accedere, quindi, ad una base dati condivisa.

Elaborazione dati

Le informazioni che arrivano ad un sistema **SCADA-HMI**$^{\mathbf{g}}$ sono molteplici e di svariata natura. Ciò comporta l'esigenza di uniformare tali

dati al alcuni tipi da dati predefiniti e conosciuti dal sistema di supervisione. Da qui appare lampante che un sistema **SCADA-HMI**^{g} opera indipendentemente dal sistema da osservare poiché le sue funzionalità standard esulano dal tipo di periferica osservata e dal tipo di informazioni da elaborare che dovranno uniformarsi necessariamente a tipi di dato processabili dal sistema.

Trattamento dei dati

Un sistema **SCADA-HMI**^{g} , opera sui seguneti tipo di dato:

- **Dati Discreti**
  I dati discreti rappresentano tutte le informazioni che possono assumere SOLO stati logici distinti. Un esempio banale è rappresentato da singolo bit 0/1 che rappresenta ON/OFF, quindi due stati distinti anche se non n bit a disposizione è possibile rappresentare n stati distinti e separati

- **Dati Analogici**
  I dati analogici rappresentano tutti quei valori analogici che sono rappresentati per mezzo di un valore intero o reale. Generalmente si parla di misurazioni che vengono rappresentate mediante un range. Ad esempio la pressione può essere indicata come un intervallo che va a 0 a 300

- **Dati Stringa**
  I dati stringa rappresentano tutte quelle informazioni rappresentate da una sequenza di caratteri che, spesso e volentieri, va interpretata nella lingua dell'operatore. L'esempio più semplice è la stampa a video di un'operazione da compiere. Piuttosto che stampare "compiere operazione 2" è più semplice sostituire quel "2" con il compito da svolgere.

In tutti gli altri casi i dati fanno riferimento a informazioni complesse che il sistema di elaborazione deve processare senza nessun cambiamento di forma, quindi esattamente così come li riceve.

Database Runtime

Tutti i dati gestiti da un sistema **SCADA-HMI**$^{\mathbf{g}}$ devono risiedere in un unico database detto database runtime, la cui caratteristica principale è fornire l'accesso a tali dati privilegiando la velocità. Tale caratteristica è fondamentale per garantire un'elaborazione compatibile con la velocità di cambiamento dello stato di un processo. La struttura di tale database deve essere pertanto monolitica. Un ulteriore vincolo per tale database è rappresentato dall'accesso concorrente di più processi ai dati in esso contenuti. All'interno di un sistema **SCADA-HMI**$^{\mathbf{g}}$ dobbiamo distinguere, necessariamente, le informazioni in due tipi, **statiche** e **dinamiche**. Le prime sono informazioni che non variano nel corso dell'elaborazione, che non necessitano quindi di rimanere costantemente all'interno del database.

Le seconde sono informazioni che mutano nel corso dell'elaborazione e rappresentano quindi quel tipo di informazione che va contenuta all'interno del database dinamico.

All'interno di un database dinamico *non* possono mancare:

- **L'identificativo del dato**
  Si tratta di un codice univoco e interno al sistema che identifica il dato. Può essere il nome del dato stesso, o può essere assegnato da chi configura il sistema anche se, generalmente, è generato e assegnato automaticamente dal sistema stesso. Tramite l'identificativo il sistema riconosce il tipo di dato e conosce tutte le sue caratteristiche

- **Etichetta di tempo relativa all'ultima acquisizione**
  Si tratta dell'informazione che tiene traccia dell'ultima acquisizione effettuata del dato. Serve quindi a mantenere una certa integrità all'interno del sistema e per far sì che anche altri sistemi esterni che possano accedere al database siano in grado di conoscere il momento in cui è avvenuto l'ultimo *fetch* per quel che riguarda quel particolare dato.
  Ciò porta anche ad una qualità migliore dell'informazione poiché sappiamo perfettamente quando il sistema osservato ha "generato" quel particolare stato e quindi l'informazione che si va ad analizzare

- **Valore corrente**
  Si tratta dell'ultimo valore assunto dal dato nel momento dell'ultima acquisizione dello stesso

- **Codice di qualità**
  Si tratta di un attributo che contribuisce a completare l'informazione espressa dal dato e che ne aumenta significativamente il valore. Un esempio semplice è quello di un dato analogico che va sopra o sotto l'intervallo definito dal sistema. Avere l'estremo superiore o inferiore è un'informazione riduttiva, mediante questo attributo siamo in grado di sapere subito se quel valore è nel range dell'intervallo o se sta fuori per eccesso o per difetto. Si tratta di un'informazione importante per ottenere un'informazione quanto più completa e corretta possibile.
  I valori generalmente associati a questo attributo sono i seguenti:

  - **Fuori scansione** : il valore dato non è stato correttamente acquisito
  - **Fuori range** : il valore è fuori dall'intervallo
  - **Valore manuale** : il valore del dato è stato impostato manualmente
  - **Invalido** : il valore del dato non è valido.

Configurazione dei dati

La gestione di un singolo dato necessita di un ulteriore livello di analisi. Un dato è, infatti, definito da un ulteriore gruppo di informazioni chiamato "configurazione".
Generalmente un dato, all'interno di un sistema **SCADA-HMI**^{g} , attraversa le seguenti operazioni:

- **Processo di acquisizione**
  Per essere definito c'è bisogno di:

  - **Il canale di acquisizione**
    Definisce essenzialmente a quale canale di acquisizione è as-

sociato il dato e permette l'instaurarsi di un flusso di comunicazione con lo stesso

  - **Identificativo del dato nel canale**
    Si tratta di un ID che identifica univocamente il dato all'interno del flusso di informazioni che si è instaurato. La definizione di tale identificativo è definita dal protocollo di comunicazione utilizzato

  - **Caratteristiche dato da acquisire**
    Definisce la tipologia del dato da acquisire. Un sistema **SCADA-HMI**[g] definisce in maniera differente dati analogici o digitali, pertanto è fondamentale conoscere a priori la tipologia del dato che deve essere acquisito e successivamente elaborato.

- **Processo di conversione ingegneristico**
  Una volta acquisito il dato è necessario effettuare una conversione dello stesso per far sì che possa essere elaborato correttamente. Ad esempio l'acquisizione di un segnale può essere (e in genere è così) convertita in un dato logico. In questo caso "semplicemente" si associa un dato segnale ad uno stato logico che lo identifica.
  Più complesso è il caso di una misura analogica in cui il dato acquisito è rappresentato da un certo numero di byte compresi del bit di segno che rappresentano il valore numerico del dato all'interno di un intervallo ben definito che va dall'estremo negativo all'estremo positivo del range di accettazione.
  Tale valore, definito "grezzo", non ha nessun significato per un operatore se privo del suo "senso ingegneristico", pertanto è necessario un processo di conversione che trasformi il valore analogico in un valore ingegneristico. Quello che accade per effettuare questa conversione è quindi l'associazione di un'**unità ingegneristica** (banalmente l'unità di misura) e di una **funzione di conversione** che non è altro che l'algoritmo da utilizzare per convertire il valore letto.
  Gli algoritmi che si utilizzano, in genere, sono:

– **Conversione lineare** :

$$y = ax + b \tag{1}$$

– **Conversione quadratica** :

$$y = ax^2 + bx + c \tag{2}$$

– **Conversione quadratica inversa** :

$$y = a + b * \sqrt{x} \tag{3}$$

Per essere definiti, tali algoritmi necessitano dei parametri di conversione che generalmente coincidono con i valori di massimo e minimo grezzi dell'unità ingegneristica.

Processo di rilevamento allarmi

Quando un dato viene acquisito questo può assumere un valore che rappresentano condizioni normali di funzionamento oppure valori che identificano una qualche sorta di malfunzionamento del sistema osservato.
In quest'ultimo caso è necessario dare all'operatore un segnale d'allarme che lo porti a riportare il sistema in una condizione di funzionamento ottimale.
Un allarme può essere generato solamente da segnali o misure analogiche ed esiste un trattamento dello stesso differente a seconda del tipo di dato che lo ha generato.
Un segnale, che assume il valore di uno stato logico, non fa altro che associare ad ogni suo stato una condizione di funzionamento o di malfunzionamento.
Ci sono anche eccezioni definiti da casi particolari come ad esempio la posizione intermedia di un interruttore che potrebbe essere vista come corretto funzionamento se data semplicemente dal tempo necessario al passaggio da ON ad OFF, o malfunzionamento se tale posizione eccede

dal limite di tempo fissato per il passaggio da uno stato all'altro. In casi del genere c'è da definire anche un range di accettazione che possa definire la generazione, o meno, di un allarme.
Per quel che riguarda le misure analogiche, invece, abbiamo a che fare con una pool più ampia di tipologie di allarme possibili anche se le più comuni possono essere:

- **Allarme per superamento di un limite**
  Viene generato quando la misura di riferimento è fuori dal range di accettazione, per eccesso o per difetto. In questo caso vengono definiti 4 valori che definiscono il limite, due per quello superiore e due per quello inferiore, così da definire un range di accettabilità e un range di allarme

- **Allarme di deviazione**
  Viene generato quando si sta osservando un valore che deve mantenersi costante, come la temperatura di un liquido che deve mantenersi costante. In questo caso vengono definiti due valori, uno di variazione minima e uno di variazione massima in modo tale da definire, ancora una volta, un range di accettabilità e uno di allarme

- **Allarme di velocità di variazione**
  iene generato quando un valore deve mantenersi costante nel tempo ed eccede se viene superato il limite fissato. Per definire questo valore di allarme vengono definiti un valore massimo di variazione del limite, percentuale o "analogico", e il tempo nel quale questa può avvenire.

Calcoli in linea

Una delle funzionalità che un sistema **SCADA-HMI**^{g} dovrebbe offrire è quello di restituire una tipologia di dato che sia frutto di calcoli automatici in grado di offrire un aiuto all'operatore.
Parliamo di informazioni che elaborano ulteriormente i dati primari per trarne particolari conclusioni senza lasciare l'onere di queste all'operatore in modo tale da limitare, per quanto possibile, l'intervento e l'errore

umano.
Tutto questo insieme di operazioni che permettono all'operatore di avere una serie di dati aggiuntivi per valutare lo stato di funzionamento del sistema viene definito **calcoli in linea**.
Generalmente, le informazioni che un sistema SCADA deve essere in grado di fornire sono di tre tipi:

- **Aggregato**
  Sono tutte quelle informazioni che derivano da un insieme di dati primari o aggregati. Un esempio banale è dato da un impianto elettrico composto da più generatori. E' ragionevole pensare di avere a disposizione la somma di tutte le potenze elettriche derivate dai singoli generatori in modo tale da avere sott'occhio, in un unico dato, un'informazione che ci da una panoramica generale sul funzionamento dell'impianto.
  Le operazioni matematiche che un sistema **SCADA-HMI^{g}** deve utilizzare per generare questo tipo di informazione sono essenzialmente quelle elementari, elevazioni a potenza ed estrazione di radici

- **Informazioni logiche**
  Queste informazioni corrispondono ad una serie di dati digitali che riuniscono in una serie di stati logici la valutazione degli stati. In parole povere si tratta di informazioni che condensano in una sola informazione una serie di informazioni riassumendone il significato.
  Le operazioni che le generano sono *AND*, *NOT* e *OR*

- **Informazioni statistiche**
  Si tratta di informazioni utili per ottenere informazioni in merito all'evoluzione temporale di un certo valore.
  I calcoli statistici più comuni all'interno di un sistema **SCADA-HMI^{g}** sono: calcolo del valore medio, l'integrale, la derivata, la distribuzione, in numero di transizione in un determinato stato logico e il tempo di permanenza in un determinato stato logico.

1.5.3 HMI

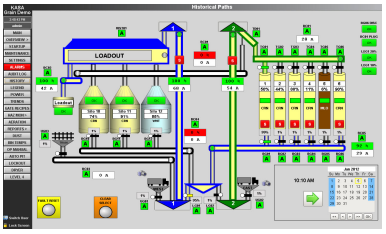

Figura 6: Esempio HMI - Fonte: http://www.kasacontrols.com/automation/hmi

L'interfaccia uomo-macchina deve avere come prima caratteristica quella dall'**usabilità** e dell'**ergonomicità**. Deve aiutare il lavoro dell'operatore facilitandone i compiti di controllo e deve essere quanto più semplice ed intuitivo possibile in modo da agevolare l'uso dello stesso.
Le funzioni principali dell'HMI sono la **presentazione dei dati** e la **gestione dei comandi dell'operatore**, entrambe le quali devono rispondere ai principi di semplicità e usabilità. La presentazione dei dati può avvenire in tre differenti modi, tramite rappresentazioni schematiche, tabellari o tramite diagrammi temporali.
Nel primo caso, quindi con **rappresentazioni schematiche**, si permette all'operatore di avere completa visione dello stato del processo osservato. Il metodo utilizzato è quello di una rappresentazione schematica degli impianti utilizzando simboli più o meno universali, riconducibili ai vari macchinari, ai quali vengono associati tutti i dati del caso. Si permette all'operatore di avere una vista generale, di effettuare dei *ZoomIn* o *ZoomOut*, o di concentrarsi su una sezione specifica dell'impianto.
Le **rappresentazioni tabellari** invece sono utilizzate per rappresentare tutti quei dati che difficilmente si possono visualizzare all'interno di una rappresentazione schematica. Generalmente si utilizzano per rappresentare relazioni fra valori prelevati da punti diversi dall'impianto in modo da offrire, in un'unica schermata, più informazione possibile.
Si deve tenere a mente che solo gli operatori più "esperti" prediligono un tipo di rappresentazione tabellare dei dati mentre una rappresentazione schematica, quando e se possibile, risulta sempre più fruibile e apprezza-

ta.

I **diagrammi temporali** invece sono fra le rappresentazioni più apprezzate dagli operatori poiché permettono, a colpo d'occhio, di osservare l'evoluzione dello stato di un processo nel tempo così da permettere anche delle predizioni sul futuro avanzare del processo osservato.

Le curve rappresentate non sono riconducibili a nessuna funzione matematica nota ma, agli occhi di un operatore che ha una profonda conoscenza del processo, tali curve sono incredibilmente utili per rendersi conto dello stato di un processo.

La gestione dei comandi operatore rappresenta, appunto, la possibilità di inviare comandi da parte di un operatore tramite l'HMI. Poiché l'invio di un comando di controllo è un momento critico per l'operatore per via della crescente pressione psicologica che un ipotetico comando errato potrebbe causare errori in tutto il sistema, l'HMI deve supportare l'intero processo di invio del comando facendo sì che l'operatore si senta il più a suo agio possibile.

Generalmente, ogni volta che il sistema può arrangiarsi da solo e prendere quindi una decisione autonomamente è bene che l'operatore non venga interpellato in merito alla decisione da prendere e, in ogni caso in cui ciò può avvenire, il sistema dovrebbe dare dei suggerimenti in merito all'azione da compiere all'operatore.

Se possibile è bene anche lasciare all'operatore solo il **potere esecutivo** relegando al sistema quello decisionale. Per esempio il sistema potrebbe autonomamente decidere che è meglio, per via di un valore fuori norma, fare uno *swtich* su un determinato componente. Potrebbe chiedere all'operatore conferma di questa manovra snellendo notevolmente la pressione psicologica dello stesso.

L'**attivazione di un comando** deve avvenire in modo molto semplice e intuitivo attraverso l'utilizzo di bottoni presenti sull'HMI stessa che devono ricordare, in qualche modo, l'oggetto reale aiutando l'operatore anche mediante l'utilizzo di animazioni che ne simuli l'effettivo comportamento.

Per facilitare i compiti di un operatore sarebbe il caso di preparare l'HMI in modo tale che ci sia un profilo assegnato ad ognuno di loro. Tramite il profilo si può abilitare l'operatore a svolgere certi tipi di funzioni e disa-

bilitarne altre, così da evitare che si possano verificare errori dovuti alla scarsa attenzione o a pressioni errate. Sarebbe anche il caso di richiedere, ad ogni comando inviato, anche una conferma dello stesso in modo tale da scoraggiare qualsiasi azione impulsiva.
Ovviamente il design di questo genere di funzionalità è guidato dall'operazione da svolgersi che ne determina il livello di sicurezza richiesto.
Una volta attivato un comando, l'HMI deve fornire anche la visualizzazione dell'esecuzione che possa in qualche modo informare l'operatore che il suo comando è stato eseguito, che è ancora in corso e mostrandone gli effetti a livello grafico. Si tenga sempre a mente che un operatore sul campo è abituato a ricevere un feedback istantaneo dalla macchina sotto forma di suono, movimento o simili. L'HMI deve simulare in qualche modo tale comportamento dando un risultato in tempo reale all'operatore che può osservare l'esecuzione del suo comando.
Gli **eventi e gli allarmi** rappresentano un'ulteriore importante caratteristica da rappresentare all'interno dell'HMI essendo essi rappresentazioni di anomalie da parte del sistema. Se inizialmente può sembrare che eventi ed allarmi rappresentino la stessa cosa, un'analisi più approfondita mostra quanto essi siano profondamente differenti e quanto necessitino, quindi, di un modo differente di essere trattati.
In sintesi, l'evento è scatenato dal passaggio dello stato osservato da una fascia ad un'altra mentre l'allarme è associato ad ogni banda, tranne quella normale.
Ogni qualvolta che si verifica un'anomalia, l'operatore è chiamato a prenderne visione in qualche modo.
È importante tenere traccia dello storico di eventi e allarmi all'interno di una lista nella quale non dovrebbe essere presente una funzionalità di cancellazione automatica. Se al momento della scomparsa dell'anomalia il sistema nota che l'operatore ne aveva preso atto allora è autorizzato a cancellarlo, altrimenti l'evento non deve scomparire.
È importante che l'operatore possa sempre prenderne atto e che abbia sempre a disposizione, in forma tabellare, tutto lo storico degli allarmi con, magari, una segnalazione luminosa o simile, che segnali che un allarme non è ancora stato riconosciuto.
Di grande utilità sarebbe anche una caratterizzazione temporale di que-

sta informazione in modo tale da poter ricostruire tutta la storia di un malfunzionamento dal momento in cui si è scatenato il primo evento anomalo.
Ulteriori funzionalità utili che possono essere messe a disposizione dal sistema sono l'associazione di una categoria, di una priorità, di una regola di disabilitazione dinamica o di permessi operatore.
Tali funzionalità servono a facilitare l'organizzazione degli eventi o degli allarmi e, di conseguenza, ne migliorano notevolmente la consultazione.

Archiviazione dei dati

Un sistema **SCADA-HMI**[g] dovrebbe dare anche la possibilità di archiviare i dati relativi all'intero sistema e questo può portare a svariate soluzioni di progettazione che devono tenere conto del modo in cui si vogliono salvare le informazioni e, sopratutto, della mole delle informazioni da archiviare.
Una cosa che sicuramente non può mancare è la messa in relazione di valori e informazioni fra di loro differenti in modo tale da tenere traccia di tutte le relazioni fra gli stati del sistema osservato.
A questo scopo vengono in aiuto i database relazionali anche se è possibile pensare anche a differenti soluzioni di progettazione.

Analisi dei dati

Un'ulteriore funzione che un sistema SCADA dovrebbe offrire in merito ai dati estrapolati è quello di fornire strumenti di analisi.
Fra i più comuni, il più utile è il cosiddetto **trend storico** che mette in relazione un dato al tempo, o ad altri tipi di dati, in modo tale da mostrarne l'evoluzione all'interno di un unico strumento. Tale funzione è generalmente inserita all'interno dell'HMI in modo tale da fornire all'operatore uno strumento di analisi molto potente al fine di aiutarlo nell'interpretazione dei dati.
Possono essere utilizzati anche altri tipi di grafici o strumenti che mettano in relazioni dati (diagrammi a torte, istogrammi etc.) che fungano da strumento di analisi dei dati trattati. La loro implementazione e funzione

è da definire in fase di progettazione a seconda dei casi.

2 Analisi e Requisiti

Il primo passo svolto durante la durata dello stage è stato quello di ampliare la conoscenza in merito ai software **SCADA-HMI**[g] con uno sguardo ravvicinato ai *competitor* in modo tale da comprendere pienamente quello che sarebbe stato il sistema sviluppato in modo tale da riuscire a reperire le librerie più adatte allo scopo.
Al fine di rendere il lavoro più agevole è stata effettuata una riunione in **SanMarco Informatica S.p.A** così da capire al meglio la direzione da far prendere ai lavori e per definire i requisiti da dover soddisfare.

2.1 Riunione

Durante la riunione in azienda si è analizzato un rudimentale quadro sinottico sviluppato nel passato da **SanMarco Informatica S.p.A** che si componeva di alcune sezioni interessanti per il nuovo sistema da sviluppare che sembrava avere molto in comune con il vecchio progetto.
Il vecchio progetto di **SanMarco Informatica S.p.A** prevedeva la visione di un quadro sinottico[g] all'interno del quale disporre a proprio piacimento alcune immagini, precedentemente caricate all'interno del sistema, in modo da definire il proprio impianto.
Il sistema offriva la possibilità di piazzare anche gli operatori vicino ad ogni macchinario in modo tale da tenere traccia anche dell'operatore al lavoro sulla particolare macchina in un dato periodo di tempo. Inoltre era previsto un elaborato sistema d'allarme che metteva in mostra gli stati alterati degli impianti oltre che mettere a disposizione dell'operatore uno storico degli stessi in modo da avere sempre sotto controllo lo stato dell'impianto.
Sono poi state analizzate alcune interfacce HMI per comprendere meglio di cosa aveva bisogno **SanMarco Informatica S.p.A** e quindi per fissare i requisiti del sistema che sarebbe stato poi sviluppato.
Alla fine della riunione è stato deciso che, per via di alcune limitazioni hardware non sarebbe stato possibile sviluppare un vero e proprio siste-

ma **SCADA-HMI**$^{\mathbf{g}}$ poiché i PLC presenti in **SanMarco Informatica S.p.A** non riuscivano a garantire una velocità di *fetch* delle informazioni tale da poter garantire un monitoraggio ottimale del sistema.
Si è pensato quindi di sviluppare il sistema sulla base di uno **SCADA-HMI**$^{\mathbf{g}}$ senza addentrarsi troppo all'interno dei requisiti di base di quest'ultimo.
In sintesi, il bisogno di **SanMarco Informatica S.p.A** si è manifestato nella forma di un componente dell'attuale **JMES** che integrasse in qualche modo il **Dialogo Operatore** offrendo la possibilità all'operatore di definire e salvare un impianto, di monitorarlo e offrire un rudimentale sistema d'allarme.

2.2 Analisi del mercato

Durante la riunione già menzionata nella sezione 2.1 si è osservato con occhio critico anche l'attuale mercato **SCADA-HMI**$^{\mathbf{g}}$ cercando di capire in che direzione si sono spinti i vari competitor.
Nel dettaglio si sono analizzati i software :

- WinCC
- Wonderwave
- Movicon-11
- Zenon
- Ignictiion

Dall'analisi di tali software e soprattutto dal *bussiness plan* delle *software house* sviluppatrici ci si è resi conto che **SanMarco Informatica S.p.A** non poteva entrare in competizione con tali sistemi.
Da qui l'esigenza di modificare leggermente i requisiti del progetto proseguendo su una strada parallela che consisteva nello sviluppare un sistema non appartenente alla famiglia degli **SCADA-HMI**$^{\mathbf{g}}$ ma che ne riprende molti aspetti, come la visione dell'impianto, la realizzazione di un quadro generale e quella di un rudimentale sistema di allarmi che tenga

traccia dello storico degli stati dell'impianto.

2.3 Requisiti e Database

L'analisi dei requisiti è partita dal lato *server* dell'applicativo e ha previsto un considerevole numero di ore dedicato allo studio del sistema già presente in **SanMarco Informatica S.p.A** .
L'aiuto di Alex Beggiato e di tutto il team di sviluppo è stato fondamentale in questa fase al fine di velocizzare la comprensione della strumentazione da utilizzare e per la stesura dei requisiti riguardanti il lato *server* della web-app[g] .

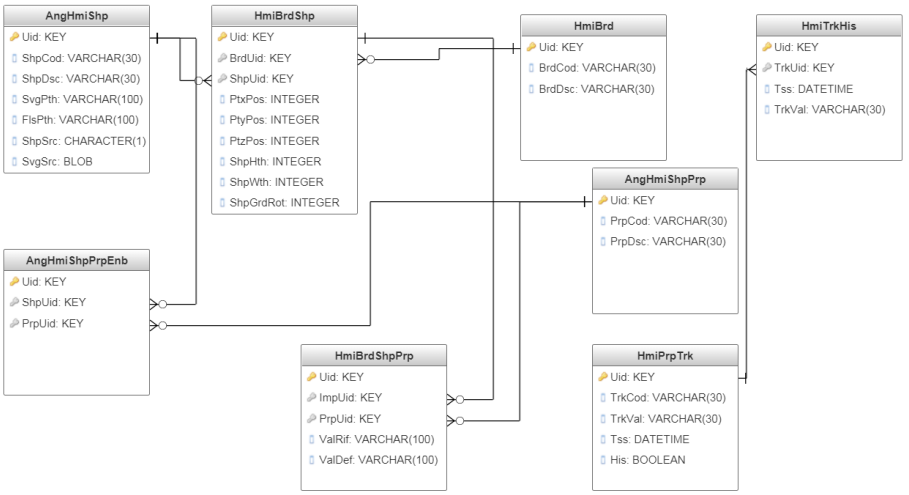

Figura 7: ER Database

La nomenclatura delle tabelle rispetta le norme di sviluppo di **SanMarco Informatica S.p.A** che prevede parole, anche in sequenza, di tre lettere che descrivano, in maniera generale, l'utilizzo della tabella in questione.
Tutte le tabelle presentano una cancellazione logica[g] e solo la tabella **HmiBrdShpPrp** presenta la possibilità di Audit Log[g] .
Le tabelle presenti all'interno del database, come visibile in figura, sono le seguenti e hanno il seguente comportamento:

- **AngHmiShp**
  Si tratta della tabella di anagrafica delle forme, quindi della tabella che tiene traccia di tutte le forme, equivalenti alle macchine di un impianto, presenti all'interno del sistema

- **HmiBrd**
  Si tratta della tabella che deinisce e tiene traccia di una *board*, ovvero di un impianto

- **HmiBrdShp**
  Si tratta della tabella che mette in relazione una forma e una *board* definendo quindi un'implementazione dell'impianto

- **HmiBrdShpPrp**
  Si tratta della tabella che definische una *dashboard*, ovvero l'interfaccia con cui costruire la propria implementazione dell'impanto

- **AngHmiShpPrp**
  Si tratta della tabella che definisce l'anagrafica delle possibili proprietà di una forma

- **AngHmiShpPrpEnb**
  Si tratta di una tabella che mette in relazione le forme e le proprietà definendo, per ogni forma, tutte le proprietà che sono abilitate per la stessa

- **HmiPrpTrk**
  Si tratta della tabella he definisce le variabili di *tracking*, ovvero tutte quelle variabili il cui valore cambia dinamicamente l'aspetto di un forma all'interno dell'implementazione di un impianto.
  La tabella contiene il valore temporale dell'utimo *fetch* dell'informazione e un campo *boolean* che definisce l'intenzione di storiciz-

zare il dato

- **HmiTrkHis**
  Si tratta della tabella che tiene traccia dello storico dei valori assunti dalle proprietà storicizzate in modo tale da avere sempre a disposizione l'andamento nel tempo dell'impianto.

2.3.1 Requisiti database

Nella tabella di seguito è possibile osservare tutti i requisiti riferiti alle tabelle del *database* visto nella sezione precedente, ovvero tutti quei requisiti definiti dalle operazioni di *insert*, *delete* e *update*.
Si tratta quindi di requisiti da rispettare ogni qualvolta viene svolta una delle operazioni precedentemente elencate.
Tutti i requisiti elencato di seguito sono da considerarsi come obbligatori e per ognuno di essi sono stati definiti alcuni test automatici al fine di verificarne il corretto funzionamento.

Tabella 1: Requisiti Database

Tabella	Evento	Requisito
AngHmiShp	Insert	Se il campo dati ShpSrc è 'f' allora il campo dati FlsPth deve essere valorizzato
AngHmiShp	Insert	Se il campo dati ShpSrc è 's' allora il campo dati SvgPth deve essere valorizzato
AngHmiShp	Insert	Se viene inserito un record con il campo dati SvgPth verifico che ci sia anche un SvgSrc valorizzato
AngHmiShp	Update	Se viene aggiornato il campo FlsPth, e questo diventa nullo, il flag ShpSrc deve essere 's'
AngHmiShp	Update	Se viene aggiornato il campo SvgPah, e questo diventa nullo, il flag ShpSrc deve essere 'f'
AngHmiShp	Delete	Se viene eliminata una forma, allora vengono eliminate tutte le implementazioni che la utilizzano
AngHmiShp	Insert	Se viene inserita una forma con codice "link" allora in AngHmiShpPrpEnb devono esserle associate le proprietà di from, to e fillColor

HmiBrd	Delete	Se esiste un'implementazione dello schema non è possibile cancellarlo
HmiBrdShp	Insert	Verifico che BrdUid esista
HmiBrdShp	Insert	Verifico che ShpUid esista
HmiBrdShp	Insert	Verifico che PtxPos non sia negativo
HmiBrdShp	Insert	Verifico che PtyPos non sia negativo
HmiBrdShp	Insert	Verifico che ShpHth non sia negativo
HmiBrdShp	Insert	Verifico che ShpWth non sia negativo
HmiBrdShp	Insert	Verfico che PtzPos non sia negativo
HmiBrdShp	Update	Verifico che BrdUid esista
HmiBrdShp	Update	Verifico che ShpUid esista
HmiBrdShp	Update	Verifico che PtxPos non sia negativo
HmiBrdShp	Update	Verifico che PtyPos non sia negativo
HmiBrdShp	Update	Verfico che PtzPos non sia negativo
HmiBrdShp	Update	Verifico che ShpHth non sia negativo
HmiBrdShp	Update	Verifico che ShpWth non sia negativo
HmiBrdShp	Delete	Se elimino un'implementazione allora elimino tutte le dashboard che la utilizzano
AngHmiShpPrp	Delete	Se la proprietà è riferita non posso cancellarla
HmiBrdShpPrp	Insert	Verifico che ImpUid, se non null, esista
HmiBrdShpPrp	Insert	Verifico che PrpUid, se non null, esista
HmiBrdShpPrp	Insert	Verifico che PrpUid esista per la forma dell'implementazione se questa ha almeno una proprietà
HmiBrdShpPrp	Update	Verifico che PrpUid esista per la forma dell'implementazione se questa ha almeno una proprietà
HmiBrdShpPrp	Update	Verifico che PrpUid, se non null, esista
HmiBrdShpPrp	Update	Verifico che ImpUid, se non null, esista
HmiPrpTrk	Insert	Se His è true allora verifico che esista il record anche dentro HmiTrkHis
HmiTrkHis	Delete	His viene settato a false su HmiPrpTrk

2.3.2 Requisiti di Sistema

Nella tabella di seguito vengono riportati tutti i requisiti di sistema definiti con il tutor aziendale Alex Beggiato .
La notazione per ogni requisito è la seguente:

- **RxTy** dove **x** sta per:
  - 0 : Obbligatorio
  - 1 : Desiderable.

  T sta per:
  - F : Funzionale
  - V : Vincolo
  - Q : Qualità

  e **y** indica il numero progressivo del requisito

Alla fine del periodo di stage sono stati coperti il 99%.

Tabella 2: Requisiti di Sistema

Requisito	Tipologia	Descrizione
R0F1	Funzionale-Obbligatorio	L'applicazione deve offrire un servizio di creazione di una forma
R0F2	Funzionale-Obbligatorio	L'applicazione deve offrire un servizio di editing di una forma
R0F3	Funzionale-Obbligatorio	L'applicazione deve fornire un servizio di creazione di un impianto
R0F4	Funzionale-Obbligatorio	L'applicazione deve fornire un servizio di editing di un impianto
R0F5	Funzionale-Obbligatorio	L'applicazione deve fornire un servizio di salvataggio di una forma
R0F6	Funzionale-Obbligatorio	L'applicazione deve fornire un servizio di caricamento di una forma
R0F7	Funzionale-Obbligatorio	L'applicazione deve fornire un servizio di salvataggio di un impianto
R0F8	Funzionale-Obbligatorio	L'applicazione deve fornire un servizio di caricamento di un impianto
R0F9	Funzionale-Obbligatorio	L'editor delle forme non deve permettere il disegno di quadrati non adiacenti fra loro
R0F10	Funzionale-Obbligatorio	L'editor delle forme non deve permettere la modifica del nome di una forma
R0F11	Funzionale-Obbligatorio	L'editor delle forme deve permettere l'inserimento di un nome

R0F12	Funzionale-Obbligatorio	Il nome, all'interno dell'editor delle forme, deve essere obbligatorio
R0F13	Funzionale-Obbligatorio	L'editor delle forme deve permettere l'inserimento di una descrizione
R0F14	Funzionale-Obbligatorio	La descrizione, all'interno dell'editor delle forme, deve essere modificabile
R1F15	Funzionale-Desiderabile	L'editor delle forme deve permettere la creazione di forme complesse
R0F16	Funzionale-Obbligatorio	L'editor di un impianto non deve permettere la modifica del nome di un impianto
R0F17	Funzionale-Obbligatorio	L'editor di un impianto deve permettere l'inserimento di un nome
R0F18	Funzionale-Obbligatorio	L'editor di un impianto deve consentire l'utilizzo di tutte le forme definite dall'utente
R0F19	Funzionale-Obbligatorio	L'editor di un impianto deve consentire l'utilizzo di tutte le forme inserite nel sistema
R0F19	Funzionale-Obbligatorio	L'utente può modificare tutte le proprietà attive di una forma all'interno dell'editor di un impianto
R0F20	Funzionale-Obbligatorio	L'utente può decidere quali saranno le variabili di tracciamento per ciascuna proprietà
R0F21	Funzionale-Obbligatorio	L'editor dell'impianto deve consentire di creare un collegamento fra più forme
R0F22	Funzionale-Obbligatorio	L'editor dell'impianto deve consentire di creare un collegamento che parte da una forma e finisce su se stessa
R0F23	Funzionale-Obbligatorio	L'editor dell'impianto deve consentire la singola modifica di una singola forma al suo interno
R0F24	Funzionale-Obbligatorio	L'editor dell'impianto deve consentire l'eliminazione di una singola forma al suo interno
R0F25	Funzionale-Obbligatorio	Se viene eliminata una forma all'interno dell'editor di un impianto devono essere eliminati tutti i collegamenti che la riferiscono
R0F26	Funzionale-Obbligatorio	Deve essere presente, all'interno del sistema, un modulo simulativo

R0F27	Funzionale-Obbligatorio	Deve essere presente, all'interno del sistema, un modulo che consenta la modifica dei valori delle variabili di tracciamento definire dall'utente
R1F28	Funzionale-Desiderabile	Deve essere presente, all'interno del sistema, un modulo che funga da storico per i valori delle variabili di tracciamento
R0V1	Vincolo-Obbligatorio	Il sistema deve essere compatibile con TOMCAT 9
R0V2	Vincolo-Obbligatorio	Il sistema deve essere compatibile con Jersey 2
R0V3	Vincolo-Obbligatorio	Il sistema deve essere compatibile con AngularJS 1.5.5 e inferiori
R0V4	Vincolo-Obbligatorio	Il sistema deve essere compatibile con Java 8
R0Q1	Qualità	Il lato client dell'applicativo deve essere valido secondo gli standard del W3C

3 Libreria grafica

Il primo *step* effettuato per la fase di sviluppo del sistema è stato quello di analizzare l'offerta delle libreria grafiche, compatibili con **AngularJS**, disponibili sul mercato in modo tale da riuscire a coprire tutti i requisiti decisi in fase di **Analisi**.
Il mondo di *Javascript* è costellato di librerie grafiche, sotto licenza e non, che possono essere utili allo scopo quindi si è resa necessaria un'analisi accurata dell'offerta in modo tale da scegliere quella più adatta allo scopo e, allo stesso tempo, più adatta a **SanMarco Informatica S.p.A** , quindi con un costo non troppo elevato e soprattutto con un buon livello di supporto.
Di seguito è possibile osservare una tabella riassuntiva dei benchmark[g] delle librerie analizzate al fine di comprendere al meglio la scelta finale.
Il resto del capitolo si occuperà di analizzare meglio le singole librerie e, infine, di analizzare nel dettaglio **GoJS**, quella selezionata per lo sviluppo del prototipo.

Tabella 3: Benchmark Liberie grafiche

	Costo*	Documentazione	Supporto	Libertà	Compatibilità AngularJS	HMI Prefabs
BonsaiJS	Free (MIT)	Sufficiente	Buono	Buona	Ottima	No
Mango Automation	$2495	Scarsa	Ottimo	Scarsa	Nessuna	Sì
MbLogic	Free (MIT)	Sufficiente	Scarso	Scarsa	Nessuna	Sì
PerfectWidget	$2999	Sufficiente	Sufficiente	Sufficiente	Sufficiente	Sì
Draw2d	399 €	Buona	Sufficiente	Buona	Ottima	No
GoJS	$8,985.00	Ottima	Ottimo	Ottima	Ottima	Sì

* Il costo è da considersi per una licenza completa (infinite installazioni/infinite applicazioni/supporto completo)

3.1 BonsaiJS

Figura 8: Logo libreria BonsaiJS - Fonte:https://bonsaijs.org/

Si tratta di una libreria gratutita (licenza MIT) che offre un servizio di disegno e renderizzazione di qualsiasi tipo di figura geometrica base. Al suo interno sono, infatti, integrati alcuni oggetti che riferiscono ciascuno ad una forma particolare.
BonsaiJS mette a disposizione dello sviluppatore varie funzioni che permettono di disegnare la forma, renderizzarla, colorarla come si preferisce e personalizzarla in modo piuttosto dettagliato.
A tutto questo si aggiunge un sistema molto basilare e semplice per l'animazione di qualsiasi elemento disegnato sullo schermo.
Purtroppo il prodotto non si è rivelato adatto allo scopo del progetto di stage per via di alcune sue limitazioni.
Il suo difetto principale risiede in una documentazione non troppo approfondita che necessita di uno studio più approfondito e dedito al testing da parte dello sviluppatore per riuscire a compiere tutte le azioni complesse offerte dalla libreria.
Ioltre, e questo difetto si è rivelato discriminante all'interno della scelta, **BonsaiJS** non consente una manipolazione efficace ed efficiente di forme caricate dall'esterno.
Si rendono necessarie, infatti, alcune conversioni mediante le funzionalità avanzate della libreria.
La buona libertà d'azione offerta dagli strumenti presenti si traduce in un eccessivo dispendio di tempo e risorse che ha portato, quindi, alla decisione di scartare di tale libreria per lo sviluppo del progetto.

3.2 Mango Automation

Figura 9: Logo Mango Automation - Fonte:https://infiniteautomation.com/data-acquisition

Più che una libreria, in questo caso, parliamo di un complesso sistema di acquisizione e manipolazione dati sviluppato e offerto da **Infinite Automation**.
Mango in particolare è la loro soluzione end-to-end[g] per lo sviluppo di dashboard e interfaccie per **SCADA-HMI**[g] .
Il sistema da loro offerto permette una personalizzazione piuttosto ampia di ogni componente e si integra perfettamente con **Angular 2** e permette anche l'importazione di *component* personali.
Il sistema permette anche l'*override* di ogni sua funzionalità permettendo allo sviluppatore, di fatto, di utilizzare solo lo scheletro architetturale sviluppato da **Infinite Automation** pensato ad hoc per i **SCADA-HMI**[g] e cucirci addosso la propria *dashboard* personale, pienamente compatibile con **Angular 2**.
Il limite dell'applicativo risiede nella sua compatibilità con **Angular 2** mentre a **SanMarco Informatica S.p.A** il sistema necessita di un prodotto pienamente compatibile con **Angular 1.5** come definito all'interno dei requisiti.

3.3 MbLogic

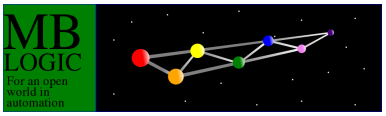

Figura 10: Logo MbLogic - Fonte:http://mblogic.sourceforge.net/

Questo sistema, distribuito sotto licenza MIT, presenta numerosissimi vantaggi per lo sviluppo del progetto di stage grazie a tutti gli strumenti integrati pensati appositamente per le interfacce HMI e i **SCADA-HMI**[g] .
Il vantaggio maggiore risiede proprio nella mole di strumenti utili allo sviluppo e nelle potenzialità degli stessi che permettono, infatti, di creare **SCADA-HMI**[g] anche piuttosto complessi ed efficienti.
D'altra parte abbiamo a che fare, purtroppo, con una documentazione

quasi inesistente, con un supporto minimo al software e, difetto maggiore per il progetto di stage, nessuna compatibilità con il *framework* **AngularJS**.
MbLogic, infatti, è scritto interamente in *Python* e non presenta nessn tipo di compatibilità, a meno di utilizzo di ulteriori librerie, al tipo di sistemi utilizzati a **SanMarco Informatica S.p.A** .

3.4 PerfectWidget

Questa libreria è una fra le più interessanti fra quelle prese in analisi per via della sua altissima compatibilità con qualsivoglia applicazione web.
Presenta il grandissimo vantaggio di essere scritta interamente in **Javascript** *vanilla* il che la rende, potenzialmente, compatibile con qualsiasi *framework* fra cui **AngularJS** al patto di integrare manualmnete tutte le funzionalità necessarie al proprio progetto.
La libreria offre una soluzione semplice ed efficace per la creazione (e solo questa) di widget[g] per qualsiasi tipo di applicazione web basata su **HTML5**.
È possibile con pochissime righe di codice importare le proprie immagini, renderizzarle in *canvas* e utilizzarle a piacimento all'interno della pagina web.
La manipolazione avviene attraverso le funzionalità standard di **Javascript** anche se questo rappresenta un'arma a doppio taglio per via delle scarse performance che ne derivano.
Tali difetti, e la scarsa libertà nel riuscire a comporr un'interfaccia più complessa di un singolo widget[g] hanno portato allo scarto di questa libreria nonostante le sue evidenti potenzialità.

3.5 Draw2D

Questa libreria, assieme a **GoJS**, è una di quelle più indicate per il tipo di progetto di stage che si sarebbe dvuto sviluppare.
Fra i vantaggi troviamo un'ottima documentazione, un ottimo prezzo,

un ottimo supporto, la piena compatibilità con **AngularJS** e una grandissima libertà d'azione per lo sviluppatore.
La libreria mette a disposizione dell'utente, infatti, numerosissime funzioni per il disegno e la manipolazione di semplici forme, funzioni per il caricamnto e la manipolazione di immagini SVG e numerosi funzioni per la costruzione di una *dashboard*.
È possibile creare applicativi piuttosto complessi e avanzati ed è possibile avere pieno controllo su tutto quello che viene disegnato tramite la libreria.
Il pieno supporto a **AngularJS** rende la libreria pressoché perfetta per il tipo di progetto da sviluppare.
La scelta finale è stata determinata da due ulteriori fattori, meno importanti, che hanno segnato la differenza.
All'interno dei *prefabs* di **Draw2D** non sono presenti forme utili ad interfacce HMI e la libreria non è compatibile, assolutamente, con **Angular 2** o superiori.
Un eventuale aggiornamento dei sistemi di **SanMarco Informatica S.p.A** avrebbe portato all'abbandono di **Draw2D** e al dover cercare e sviluppare nuovamente il prodotto finale con un notevole dispendio di tempo e risorse.

3.6 GoJS

Figura 11: Logo GoJS - Fonte:https://gojs.net/

GoJS è una libreria grafica avanzata che, con le giuste conoscenze e il giusto tempo di apprendimento, consente, fondamentalmente, di creare qualsiasi tipo di elemento grafico all'interno di un'applicativo web e di manipolarlo con estrema libertà ed estrema facilità.

Al suo interno sono racchiusi degli strumenti che si sono, con il senno di poi, rivelati fondamentali all'interno dello sviluppo del progetto di stage e che hanno semplificato di molto lo sviluppo di moltissime funzionalità.
La libreria presente un'ottima documentazione, un ottimo supporto, con il quale ci si è interfacciati molto drante il periodo di stage per la spiegazione di alcune funzionalità criptiche del prodotto, e piena compatibilità con **AngularJS**.
La libreria, inoltre, presenta piena compatibilità anche con le versioni attuali del *framework* oltre che il supporto a tutti i *framework* più utilizzati oggi per lo sviluppo di applicativi web.
All'interno di **GoJS** sono presenti anche numerose forme base tipiche delle interfacce HMI che hanno svolto un ruolo chiave nello sviluppo del sistema.
Una delle *feature* più interessanti del prodotto è, però, la funzionalità di caricamento e gestione degli SVG.
GoJS mette a disposizione, infatti, una funzione di parsing (di cui è possibile fare l'*override* a piacere per migliorarne le performance) che data in input una tipica stringa nel formato standard per gli SVG ne restituisce l'immagine.
Questa funzionalità si è rivelata fondamentale per la scrittura del modulo di caricamento di una forma esterna e ha reso possibile il soddisfacimento di molti dei requisiti analizzati.

4 Sviluppo

All'interno di questa sezione sarà trattato nel dettaglio lo sviluppo dell'applicativo.
Nella prima parte del capitolo verranno trattati nel dettaglio i casi d'uso dell'applicativo mentre nella seconda parte verranno mostrate le sezioni di codice più interessanti e importanti.

4.1 Casi D'uso

Tutti i casi d'uso di seguito sono stati disegnati rispettando lo standard UML 2.0 e utilizzando il software **Astha Professional**.
Tutti i casi d'uso dell'applicativo prevedono un singolo attore, l'utente loggato, poiché l'utente effettuerà l'autenticazione all'interno del modulo principale del sistema quindi, una volta arrivato all'interno del modulo sviluppato per il progetto di stage, sarà stato autenticato.

4.1.1 UC 0 - Overview dell'applicativo

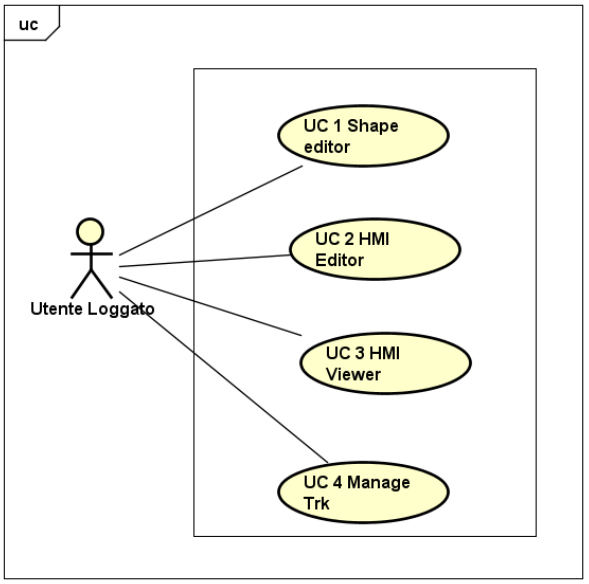

Figura 12: UC 0 - Overview dell'applicativo

L'applicativo sviluppato prevede al suo interno 4 moduli differenti, accessibili ognuno da una voce di menù che permettono all'utente di entrare all'interno delle singole applicazioni.

- **Shape Editor**: Si tratta del modulo mediante il quale è possibile accedere all'editor delle forme all'interno del quale l'utente può disegnare la sua forma personalizzata e salvarla all'interno del sistema
- **HMI Editor**: Si tratta del modulo mediante il quale è possibile accedere all'editor di una board HMI mediante la quale l'utente è in grado, mediante le forme disegnate da lui stesso o mediante quelle già presenti nel sistema, di disegnare un impianto
- **HMI Viewer**: Si tratta del modulo che consente di visionare l'impianto disegnato

- **Manage Trk**: Si tratta di un modulo simulativo all'interno del quale è possibile dare dei valori alle variabili di tracciamento in modo tale da simulare il funzionamento di un impianto.

4.1.2 UC 1 - Shape Editor

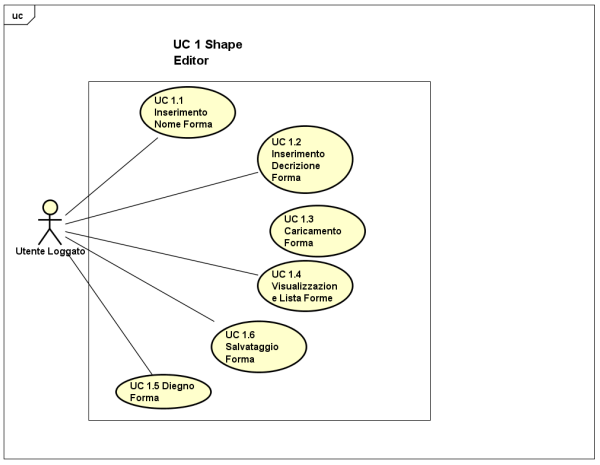

Figura 13: UC 1 - Shape Editor

Il modulo che si occupa dell'editing delle forme si è rivelato uno dei lavori più ardui da portare a termine poiché si è reso necessario lo sviluppo di un algoritmo particolare per la manipolazione dei *path* SVG.
Un file SVG presenta, infatti, la medesima struttura di un XML. Uno dei nodi, <*path*> per l'esattezza è quello che contiene al suo interno una stringa in una formato particolare che indica il "perimetro" della forma rappresentata.
Si è rivelato necessario, quindi, riuscire a ricreare questo tipo di stringa a partire da un'immagine disegnata all'interno dell'editor delle forme, così da consentire alla libreria di poterla renderizzare correttamente.
A livello puramente *client* abbiamo deciso di implementare un'interfaccia piuttosto semplice e scarna che consentisse tutte le operazioni di base per il disegno e il savataggio di una forma personalizzata all'interno del database interno.
L'utente, una volta entrato all'interno del modulo di editing delle forme

si troverà davanti una sezione in cui, tramite dei semplici *click* potrà disegnare e modificare una forma.
Avrà poi a disposizione un semplicissimo *form* mediante la quale potrà caricare una vecchia forma, sceglieno da una lista di tutte le forme presenti sul sistema, modificare nome e descrizione della forma disegnata/caricata, e inserire la forma appena disegnata/modificata all'interno del sistema.
Potrà, quindi, aggiornarne una già esistente o crearne una *ex novo*.
Nella figura 14 è possibile osservare l'interfaccia del modulo. Con un *click* sulla griglia è possibile disegnare un quadrato purchè questo sia adiacente ad un altro quadrato così da disegnare un poligono regolare composto da tanti piccoli quadrati.
Nella parte alta è invece disponibile il *form* con cui dare un nome e una descrizione alla forma, salvarla o caricarne una fra quelle già esistenti.

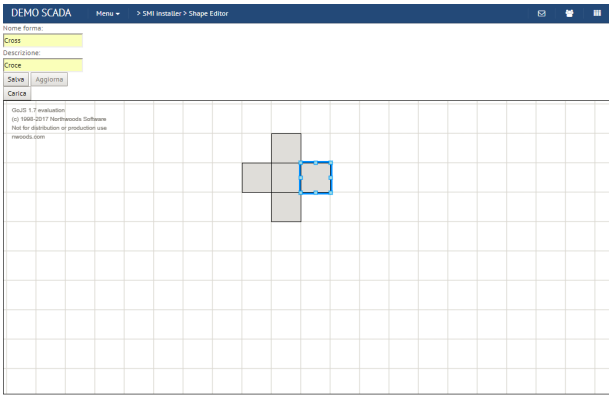

Figura 14: Shape Editor - Overview

4.1.3 UC 2 - HMI Editor

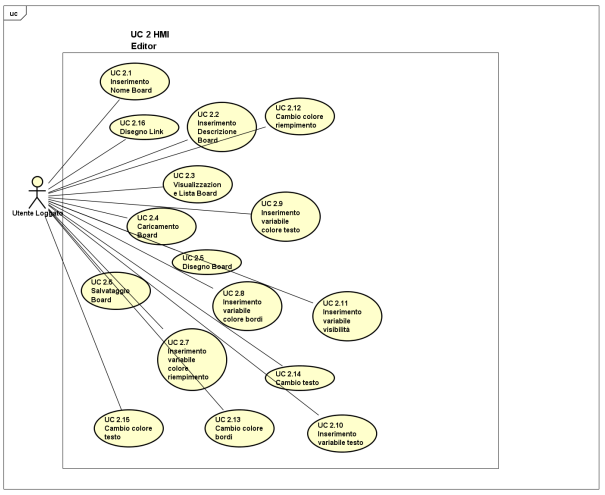

Figura 15: UC 2 - HMI Editor

Il modulo che si occupa dell'editing di un'implementazione è stato pensato per essere il più semplice possibile da utilizzare per l'utente finale. L'interfaccia si compone di 5 sezioni distinte ognuna delle quali si occupa di una funzione particolare dell'editor, come visibile nell'immagine di seguito.

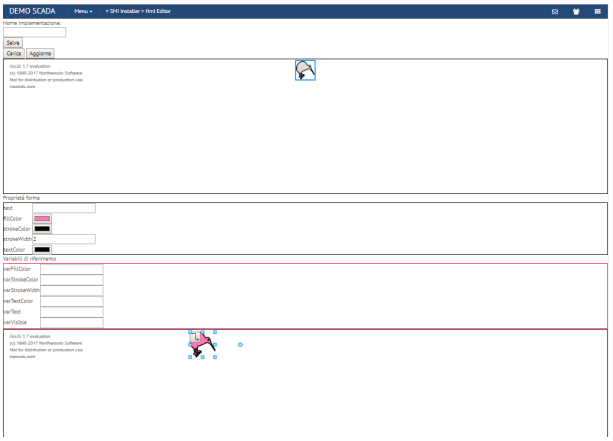

Figura 16: HMI Editor - Overview

L'utente può, tramite il semplice *form* iniziale, modificare/inserire l'anagrafica di un'implementazione, salvarla, aggiornarla oppure caricare la lista di tutte quelle salvate in modo tale da caricarne una esistente per poterla modificare.
La seconda sezione invece vede la lista di tutte le *palette* disponibili, ovvero tutte le forme disponibili per il disegno dell'impianto, che siano esse disegnate dall'utente stesso o che siano già disponibili all'interno del sistema.
Una volta trascinata una forma all'interno dell'editor appariranno due ulteriori sezioni che consentono la modifica degli elementi estetici di una forma (colore di sfondo, colore dei bordi, spessore dei bordi, testo e colore del testo) e la definizione di una variabile di tracciamento legata a queste singole proprietà.
Tali variabili saranno utilizzate dal sistema per definire il valore in tempo reale delle proprietà estetiche di una forma. Ad esempio se la variabile che tiene traccia del colore di sfondo è **d1123f**, il valore acquisito da questa variabile nel tempo determinerà il colore di sfondo della forma.
Questo sistema serve per la gestione, a video, degli allarmi legati alle singole macchine che compongono l'impianto.
Nell'ultima sezione l'utente può disporre a piacere le forme all'interno di un impianto, modificarne la grandezza oppure creare un *link* fra due forme.

4.1.4 UC 4 - Manage trk

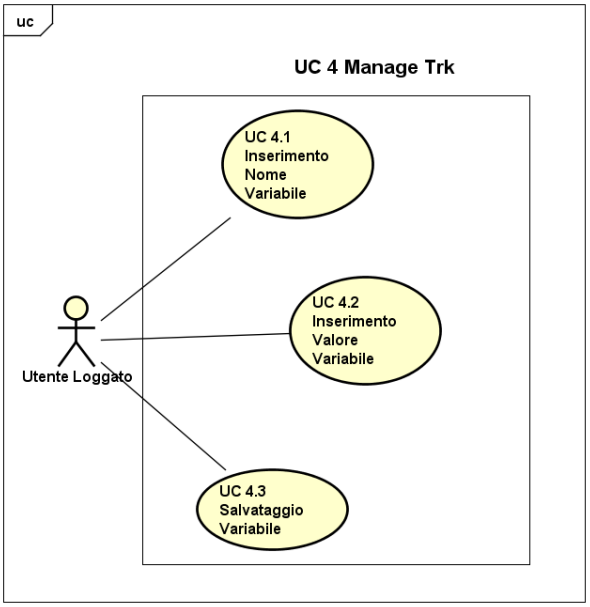

Figura 17: UC 4 - Manage Trk

Questo modulo è stato pensato solo ed esclusivamente per il testing interno inerente alle simulazioni del funzionamento di un impianto.
Il modulo presente una semplice *form* in cui l'utente può inserire il nome della variabile, il suo valore e salvarla nel sistema.
All'interno del modulo di visualizzazione potrà verificare il risultato della simulazione in corso.

4.2 Codice

Di seguito saranno riportate alcune sezioni di codice ritenute interessanti e importanti all'interno dell'economia del progetto.
Ogni pezzo di codice sarà accompagnato da una breve spiegazione della funzionalità in questione.

```
1 /**
```

```
2   * Ritorna l'elemento piu' in alto e piu' a sinistra dell'
        array
3   * @param {[]} A - Array contenente tutti i quadrati
        appartenenti al poligono da disegnare
4   */
5   var getTopLeft = function (A) {
6     let x = y = Number.MAX_VALUE;
7     let aux;
8     A.forEach(function (e) {
9       let eX = parseInt(e.pos.split(' ')[0]);
10      let eY = parseInt(e.pos.split(' ')[1]);
11      if (eY < y) {
12        y = eY;
13        x = eX;
14        aux = e;
15      }
16      else if (eY == y) {
17        if (eX <= x) {
18          x = eX;
19          aux = e;
20        }
21      }
22    })// forEach
23    return aux;
24  }// getTopLeft
25
26  /**
27   * Dato un array A di nodi, restituisco un array di elementi
         ordinati per riga
28   * @param {shape[]} A - Array contenente tutti i quadrati
        appartenenti al poligono da disegnare
29   */
30  var rowSort = function (A) {
31    let sortedArray = [];
32    while (A.length > 0) {
33      let node = getTopLeft(A);
34      let index = A.indexOf(node);
35      sortedArray.push(node);
36      A.splice(index, 1);
37    }// while
38    return sortedArray;
39  }// rowSort
40
```

```
41 /**
42  * Costruisco l'array dei lati dei signoli nodi. Passato un
        nodo e la dimensione del lato, inserisco dentro l'array
        i segmenti di riferimento di ogni lato
43  * @param {single shape} node - Nodo attuale
44  * @param {obj} container - contenitore dell'array dei lati
45  * @param {int} cell_size - larghezza di lato
46  */
47 var pushEdges = function (node, container, cell_size) {
48   let x = parseInt(node.pos.split(' ')[0]);
49   let y = parseInt(node.pos.split(' ')[1]);
50   let horizontalEdgeTop, horizontalEdgeBottom,
        verticalEdgeLeft, verticalEdgeRight;
51   let h = x + cell_size;
52   let v = y + cell_size;
53   horizontalEdgeTop = "M," + x + ',' + y + ",H," + h + ' ';
        // "M "+x+' '+y+" H "+h+' ';
54   horizontalEdgeBottom = "M," + x + ',' + v + ",H," + h + '
        '; // "M "+x+' '+v+" H "+h+' ';
55   verticalEdgeLeft = "M," + x + ',' + y + ",V," + v + ' ';
        // "M "+x+' '+y+" V "+v+' ';
56   verticalEdgeRight = "M," + h + ',' + y + ",V," + v + ' ';
        // "M "+h+' '+y+" V "+v+' ';
57   container.edges.push(horizontalEdgeTop);
58   container.edges.push(horizontalEdgeBottom);
59   container.edges.push(verticalEdgeLeft);
60   container.edges.push(verticalEdgeRight);
61 }// pushEdges
62
63 /**
64  * Dato un array ordinato di nodi, restituisce l'array dei
        lati dei singoli nodi
65  * @param {sortedShape[]} A - Array dei nodi ordinati per
        riga
66  * @param {int} cell_size - Dimensione del lato
67  */
68 var searchEdges = function (A, cell_size) {
69   let straightLines = [];
70   let edgesContainer = {
71     edges: straightLines
72   }
73   A.forEach(function (e) {
74     pushEdges(e, edgesContainer, cell_size);
```

```
75    })// forEach
76    return edgesContainer.edges;
77  }// searchEdges
78
79  /**
80   * Dato il container con l'array dei lati, elimina tutti i
         lati adiacenti
81   * @param {[]} container - Contenitore dei lati dei singoli
         nodi
82   */
83  var deleteAdjEdges = function (container) {
84    let A = container.edges;
85    //costruisco una mappa contenente tutti gli elementi che
         hanno un duplicato
86    let uniq = A
87      .map((edge) => {
88        return { count: 1, edge: edge }
89      })// map
90      .reduce((a, b) => {
91        a[b.edge] = (a[b.edge] || 0) + b.count
92        return a
93      }, {})// reduce
94    let duplicates = Object.keys(uniq).filter((a) => uniq[a] >
         1);
95    //elimino i duplicati
96    while (duplicates.length > 0) {
97      let index = container.edges.indexOf(duplicates[0]);
98      while (index != -1) {
99        container.edges.splice(index, 1);
100       index = container.edges.indexOf(duplicates[0]);
101     }// while
102     let dupIndex = duplicates.indexOf(duplicates[0]);
103     duplicates.splice(dupIndex, 1);
104   }// while
105 }// deleteAdjEdges
106
107 /**
108  * Cerco il lato da ordinare
109  * @param {edges[]} A - Array dei lati del poligono
110  * @param {int} x - Posizione x dell'ultimo lato ordinato
111  * @param {int} y  - Posizione y dell'ultimo lato orindato
112  * @param {char} dirS  - Direzione di disegno dell'ultimo
         lato oridnato
```

```
113  * @param {int} endP - Posizione di chiusura dell'ultimo
         lato ordinato
114  * @param {char} dir - Direzione di scrittura del nuovo lato
         da ordinare
115  * @param {bool} first - Determina se e' il primo lato da
         inserire e gestisce l'ordinamento dei lati paralleli
116  * @param {bool} reverse - Determina se sto leggendo la
         direzione contraria a quella convenzionale (senso
         orario)
117  */
118 var getEdge = function (A, x, y, dirS, endP, dir, first,
        reverse) {
119   if (dirS == 'H') {
120     for (let i = 0; i < A.length; i++) {
121       if ((parseInt(A[i].split(',')[1]) == endP) && (
              parseInt(A[i].split(',')[2]) == y) && (A[i].split(
              ',')[3] == dir)) { // A[i].x == endP && A[i].y ==
              y && A[i].dir == dir
122         if (dir == 'H' && y == (parseInt(A[i].split(',')[2])
                ) && !first && reverse) {
123           return null;
124         } else {
125           return A[i];
126         } // if - else
127       } // if
128     } // for
129     return null;
130   } else if (dirS == 'V') {
131     for (let i = 0; i < A.length; i++) {
132       if ((parseInt(A[i].split(',')[1]) == x) && (parseInt(A
              [i].split(',')[2]) == endP) && (A[i].split(',')[3]
               == dir)) { // A[i].x == x && A[i].y == endP && A[
              i].dir == dir
133         if (first && dir == 'H') {
134           return null;
135         } else {
136           return A[i];
137         }// if - else
138       } // if
139     } // for
140     return null;
141   } // if - else
142 }// getEdge
```

```
143
144 /**
145  * Cerco il prossimo lato da inserire nell'array degli
        ordinati in senso orario
146  * @param {edges[]} A - Array dei lati del poligono
147  * @param {int} x - Posizione x dell'ultimo lato ordinato
148  * @param {int} y  - Posizione y dell'ultimo lato orindato
149  * @param {char} dirS  - Direzione di disegno dell'ultimo
        lato oridnato
150  * @param {int} endP - Posizione di chiusura dell'ultimo
        lato ordinato
151  * @param {char} dir - Direzione di scrittura del nuovo lato
         da ordinare
152  * @param {bool} first - Determina se e' il primo lato da
        inserire e gestisce l'ordinamento dei lati paralleli
153  * @param {bool} reverse - Determina se sto leggendo la
        direzione contraria a quella convenzionale (senso
        orario)
154  */
155 var getNextEdge = function (A, x, y, dirS, endP, dir, first,
        reverse) {
156   return getEdge(A, x, y, dirS, endP, dir, first, reverse);
157 } // getNextEdge
158
159 /**
160  * Questa funzione si occupa di inserire il lato giusto nell
        'array ordinato e di eliminarlo da quello disordinato
161  * @param {obj} coord - Oggetto contenente le coordinate del
         lato in analisi
162  * @param {obj} data - Oggetto contenente tutte le variabili
         utili
163  * @param {obj} edge - Il lato da inserire
164  * @param {obj} arrays - Oggetto contenente gli array di
        lati ordinati e disordinati
165  * @param {char} dir - H se il lato e' orizzontale, V se
        verticale
166  * @param {int} cell_size - DImensione del lato
167  */
168 var insertAndSplice = function (coord, data, edge, arrays,
       dir, cell_size) {
169   data.first = false;
170   if (data.reverse) {
171     coord.endP = parseInt(edge.split(',')[4]) - cell_size;
```

```
172     coord.x = parseInt(edge.split(',')[1]);
173     coord.y = parseInt(edge.split(',')[2])
174     let newEdge = "M," + coord.x + ',' + coord.y + ',' + dir
            + ',' + coord.endP + ' ';
175     arrays.sortedEdges.push(newEdge);
176     data.reverse = false;
177     if (dir == 'V') {
178       data.nullFromReverse = false;
179     }// if
180   }
181   else {
182     arrays.sortedEdges.push(edge);
183   }// if - else
184   let index = arrays.edges.indexOf(edge);
185   arrays.edges.splice(index, 1);
186   if (dir == 'H') {
187     data.nullFromReverse = false;
188   }// if
189 }
190
191 /**
192  * Dato l'array di tutti i lati (esclusi gli adiancenti) li
        ordina in senso orario
193  * @param {edges[]} edges - Array dei lati del poligono
194  * @param {int} cell_size - Dimensione di lato
195  */
196 var sortEdges = function (edges, cell_size) {
197   let data = {
198     reverse: false,
199     first: true,
200     nullFromReverse: false
201   };
202   let arrays = {
203     sortedEdges: [],
204     edges: edges
205   };
206   arrays.sortedEdges.push(edges[0]);// pusho sicuramente il
        primo dei miei lati, che e' certamente il primo da
        inserire
207   arrays.edges.splice(0, 1);// elimino dall'array dei lati
        quello appena inserito
208   let coord = {
209     x: null,
```

```
210       y: null,
211       endP: null
212     };
213     while (arrays.edges.length > 0) {
214       if (!data.reverse) {
215         coord.x = parseInt(arrays.sortedEdges[arrays.
              sortedEdges.length - 1].split(',')[1]);
216         coord.y = parseInt(arrays.sortedEdges[arrays.
              sortedEdges.length - 1].split(',')[2]);
217       }// if
218       if (!data.nullFromReverse) {
219         coord.endP = parseInt(arrays.sortedEdges[arrays.
              sortedEdges.length - 1].split(',')[4]);
220       }
221       else {
222         coord.x = parseInt(arrays.sortedEdges[arrays.
              sortedEdges.length - 1].split(',')[1]);
223         coord.y = parseInt(arrays.sortedEdges[arrays.
              sortedEdges.length - 1].split(',')[2]);
224         data.first = true;
225       }// if - else
226       let dirS = arrays.sortedEdges[arrays.sortedEdges.length
            - 1].split(',')[3];
227       let edge = getNextEdge(arrays.edges, coord.x, coord.y,
            dirS, coord.endP, 'H', data.first, data.reverse);
228       if (edge) { // abbiamo un lato H da inserire
229         insertAndSplice(coord, data, edge, arrays, 'H',
              cell_size);
230       }
231       else { // non abbiamo un lato H da inserire, interrogo
            per V
232         edge = getNextEdge(arrays.edges, coord.x, coord.y,
              dirS, coord.endP, 'V', data.first, data.reverse);
233         if (edge) { // abbiamo un lato V da inserire
234           insertAndSplice(coord, data, edge, arrays, 'V',
                cell_size);
235         }// if
236         else if (!data.reverse) {// Non abbiamo ne' un H ne'
              un V, quindi ribaltiamo
237           if (dirS == 'V') {
238             coord.x = coord.x - cell_size;
239             coord.y = coord.y + cell_size;
240           }
```

```
241         else {
242           coord.y = coord.y - cell_size;
243           coord.x = coord.x + cell_size;
244         }// if - else
245         data.reverse = true;
246       }
247       else {
248         coord.endP = coord.endP - cell_size;
249         data.nullFromReverse = true;
250       }// if - else
251     }// if - else
252   }// while
253   return arrays.sortedEdges;
254 }// sortEdges
255
256 /**
257  * A partire dall'array dei miei lati ordinati, costruisco l
        'SVG path della forma finale
258  * @param {*} A
259  */
260 var createString = function (A) {
261   let M = A[0].split(',')[0];
262   let Mx = A[0].split(',')[1];
263   let My = A[0].split(',')[2];
264   let dir = A[0].split(',')[3];
265   let endSpot = A[0].split(',')[4];
266   let path = "F " + M + ' ' + Mx + ' ' + My + ' ' + dir + '
        ' + endSpot;
267   A.splice(0, 1);
268   A.forEach(function (e) {
269     let dir = e.split(',')[3];
270     let endSpot = e.split(',')[4];
271     path = path.concat(' ' + dir + ' ' + endSpot);
272   })// forEach
273   return path;
274 }// createString}
275 }
```

Listing 1: PolygonPath

La sezione di codice (Listing 1) è la rappresentazione dell'algoritmo sviluppato per la conversione di una forma disegnata tramite il modulo di

editor in una stringa SVG accettata dalla libreria GoJS.
Una stringa SVG molto semplice è composta dai seguenti elementi:

- **F**: se presente, la forma ha un colore di riempimento, non lo ha in caso contrario
- **M x y**: posizione iniziale su cui si inizia a disegnare la riga
- **H h**: va disegnato un segmento orizzontale che arriva in posizione h y
- **V v**: va disegnato un segmento verticale che arriva in posizione x v;

L'algoritmo implementato si occupa quindi di creare un array con tutti i quadrati che compongono la forma e riordinarlo cercando sempre il quadrato in posizione *top-left*.
Successivamente il passo è quello di identificare ed eliminare tutti i quadrati che fanno da riempimento alla forma e ottenere così solo le stringhe che definiscono il perimetro del poligono.
Una volta identificati, quindi, i lati di ogni quadrato ed eliminati i segmenti inutili (quelli che fanno parte "dell'interno" della figura) si passa alla costruzione della stringa vera e propria che verrà ritornata alla funzione chiamante.
Purtroppo per forme molto grandi l'algoritmo risponde molto lentamente poiché ci sono tanti quadrati, e quindi tanti segmenti, da processare.
In fase di analisi è stato stabilito però che non saranno disegnate mai forme troppo complesse o troppo grandi quindi per tutti i casi d'uso dell'applicativo l'algoritmo in questione risponde perfettamente.

```
1  /**
2   * Cerco la posizione della cella da riempire
3   * @param {*} size
4   * @param {*} pos
5   */
6  let getPosition = function (size, pos) {
7    let divX = Math.floor(pos.x / size.width);
8    let modX = pos.x % size.width;
9    let divY = Math.floor(pos.y / size.height);
```

```
10      let modY = pos.y % size.height;
11      return new go.Point(
12        (divX * size.width) + size.width / 2,
13        (divY * size.height) + size.height / 2
14      );
15    }// getPosition
```

Listing 2: getPosition

La sezione di codice (Listing 2) è un piccolo pezzo del modulo che gestisce il disegno dei singoli quadrati, che rappresentano una forma, all'interno della griglia.
Ci si è preoccupato di capire quanta tolleranza ci sarebbe dovuta essere nel punto di *click* da parte dell'utente per decidere all'interno di quale cella disegnare il quadrato.

```
1   /**
2    * Ricaricamento di un model
3    */
4   function reloadModel () {
5     let indexImp = 0;
6     let x = { key: 0, loc: "0 0", visible: false }; // dumb
          shape
7     $scope.main.model.addNodeData(x);
8     $scope.main.board.brdImp._BrdShpLst.forEach(function (
          imp) { // scorro per tutte le implementazioni di una
           forma
9       if (imp._Shp.shpCod != "Link") { // sto ricostruendo
            una forma
10        let key = imp.uid;
11        let modelUid = imp.shpUid;
12        let geo = imp._Shp.svgPth;
13        let loc = imp.ptxPos + " " + imp.ptyPos;
14        let angle = imp.shpGrdRot;
15        let height = imp.shpHth;
16        let width = imp.shpWth;
17        let shape = { key: key, geo: geo, loc: loc, angle:
              angle, height: height, width: width }; // creo
              la forma base per poi aggiungerle le varie
              proprieta'
18        let trk = { modelUid: modelUid };
```

```
19          $scope.main.shapeVarTrk[key] = [trk];
20          $scope.main.model.addNodeData(shape);
21          let node = $scope.main.model.findNodeDataForKey(key)
                ;
22          let indexPrp = 0;
23          imp._BrdShpPrpLst.forEach(function (prp) {
24            switch (prp._Prp.prpCod) { // controllo i valori
                  delle proprieta' per settarle correttamente (
                  comprese di variabili di riferimento)
25              case "fillColor":
26                $scope.main.model.setDataProperty(node, "
                      fillColor", prp.valDef);
27                $scope.main.model.setDataProperty(node, "
                      varFillColor", prp.valRif);
28                $scope.main.shapeVarTrk[key].push({ varName: "
                      varFillColor", value: prp.valRif });
29                break;
30              case "strokeColor":
31                $scope.main.model.setDataProperty(node, "
                      strokeColor", prp.valDef);
32                $scope.main.model.setDataProperty(node, "
                      varStrokeColor", prp.valRif);
33                $scope.main.shapeVarTrk[key].push({ varName: "
                      varStrokeColor", value: prp.valRif });
34                break;
35              case "strokeWidth":
36                $scope.main.model.setDataProperty(node, "
                      strokeWidth", parseInt(prp.valDef));
37                $scope.main.model.setDataProperty(node, "
                      varStrokeWidth", prp.valRif);
38                $scope.main.shapeVarTrk[key].push({ varName: "
                      varStrokeWidth", value: prp.valRif });
39                break;
40              case "text":
41                $scope.main.model.setDataProperty(node, "text"
                      , prp.valDef);
42                $scope.main.model.setDataProperty(node, "
                      varText", prp.valRif);
43                $scope.main.shapeVarTrk[key].push({ varName: "
                      varText", value: prp.valRif });
44                break;
45              case "textColor":
46                $scope.main.model.setDataProperty(node, "
```

```
                    textColor", prp.valDef);
47                $scope.main.model.setDataProperty(node, "
                    varTextColor", prp.valRif);
48                $scope.main.shapeVarTrk[key].push({ varName: "
                    varTextColor", value: prp.valRif });
49                break;
50              case "visible":
51                $scope.main.model.setDataProperty(node, "
                    visible", prp.valDef);
52                $scope.main.model.setDataProperty(node, "
                    varVisible", prp.valRif);
53                $scope.main.shapeVarTrk[key].push({ varName: "
                    varVisible", value: prp.valRif });
54                break;
55              default:
56                alert("ERRORE. IMPOSSIBILE RICOSTRUIRE LA
                    FORMA");
57                break;
58            }// switch
59            indexPrp++;
60          })// foreach
61          indexPrp = 0;
62        } else if (imp._Shp.shpCod === "Link") { // devo
            ricostruire un link
63          let from, to, fillColor;
64          let link;
65          imp._BrdShpPrpLst.forEach(function (prp) {
66            switch (prp._Prp.prpCod) { // scorro per le
                proprieta' per poter ricreare un link
67              case "from":
68                from = prp.valDef;
69                break;
70              case "to":
71                to = prp.valDef;
72                break;
73              case "fillColor":
74                fillColor = prp.valDef;
75                break;
76              default:
77                alert("ERRORE. IMPOSSIBILE RICOSTRUIRE IL LINK
                    ");
78                break;
79            } // switch
```

```
80          }) // foreach
81          link = { from: from, to: to };
82          $scope.main.model.addLinkData(link);
83        }// if - else
84        indexImp++;
85      })// foreach
86      $scope.model = $scope.main.model; // assegno il nuovo
            modello al vecchio cosi' da sovrascriverlo
87      $scope.main.board.lnkNum = $scope.model.linkDataArray.
            length; // tengo traccia del numero di link del
            modello appena caricato
88      $scope.main.board.shpNum = $scope.model.nodeDataArray.
            length; // tengo traccia del numero di forme del
            modello appena caricato
89      $scope.main.board.brdImpCopy = $scope.main.board.brdImp;
             // mi salvo la copia dell'implementazione prima di
            eliminarla
90      $scope.main.board.brdImp = []; //riazzero l'array per
            permettere caricamenti in sequenza
91    }// reloadModel
```

Listing 3: realoadModel

La sezione di codice (Listing 3) è stata una fra le più delicate poiché si tratta della funzione che si occupa di ricostruire un modello a partire da tutti i dati memorizzati sul *database*.
Purtroppo **GoJS** non riesce a processare molto bene questo tipo di operazione poiché il *JSON* sorgente che accetta è molto particolare e pieno di dati non presenti all'interno della base di dati.
Si è cercato quindi di sfruttare qualche comportamento anomalo della libreria al fine di riuscire a ricostruire correttamente tutta l'implementazione e di risalire ai valori di tutte le variabili.
In particolare si è reso necessario il disegno di una *"dumb shape"* che non viene visualizzata all'interno dell'impianto ma che serve per far ricreare a **GoJS** tutti quei campi dato altrimenti impossibili da ricreare senza il *JSON* sorgente.

```
1  /**
2     * Servizio speciale si set per le variabili che tracciano
          le proprieta'
```

```
3     * @param token
4     * @param tokenHead
5     * @param safe
6     * @param fromVersion
7     * @param input
8     * @param languageUid
9     * @param resSubUid
10    * @return
11    */
12   @Path ("/HmiPrpTrk/set/")
13   @PUT
14   @Produces (MediaType.APPLICATION_JSON)
15   public Response setPrpTrk (@QueryParam("token") String
         token, @HeaderParam ("token") String tokenHead,
         @QueryParam("safe") Boolean safe, @QueryParam("
         fromVersion") Integer fromVersion, RestInput input,
         @QueryParam("lngUid") Long languageUid, @QueryParam("
         resSubUid") Long resSubUid) {
16       ExecutionContext context = null;
17       try {
18           context = SessionManager.getContext (token,
                 tokenHead).getInstance (languageUid, resSubUid
                 );
19       } catch (Exception e) {
20           return Response.status (Status.UNAUTHORIZED).build
                 ();
21       }// try - catch
22       RestOutput output = new RestOutput ();
23       try {
24           GsonConverter.convertToModel (input, HmiPrpTrk.
                 class);
25           output.result = HmiPrpTrkWs.get ().set ((HmiPrpTrk
                 ) input.entity, fromVersion, null, context,
                 null);
26           return Response.ok ().entity (output).build ();
27       } catch (SException e) {
28           DebugUtil.error (e);
29           return Response.status (Status.NOT_ACCEPTABLE).
                 entity (e.buildRestObject ()).build ();
30       } // try - catch
31   } // setPrpTrk
```

```
1  /**
```

```
2     * Metodo set speciale che fa un insert in caso il record
          non esista, un update in caso contrario
3     * @param toSet
4     * @param fromVersion
5     * @param params
6     * @param context
7     * @param connection
8     * @return
9     * @throws SException
10    */
11    public HmiPrpTrk set(HmiPrpTrk toSet, Integer fromVersion,
          HashMap<String, Object> params, ExecutionContext
          context, Connection connection)  throws SException {
12        Connection con = null;
13        try {
14            con = ConnectionsUtil.checkOpenConnection (
                  connection);
15            CacheLogic cache = CacheLogic.get ();
16            HmiPrpTrk exist = HmiPrpTrkDao.get ().getByCode (
                  toSet, context, con); // faccio il get per
                  vedere se esiste il record che voglio settare
17            HmiPrpTrk toReturn = null;
18            if(exist == null) { // non esiste quindi devo fare
                  un insert
19                SDateTime tss = new SDateTime ();
20                toSet.setTss (tss);
21                cache.setHmiLastUpdate (tss); // setto il
                      valore di tss nella cache
22                toReturn = HmiPrpTrkDao.get ().safeInsert (
                      toSet, context, con);
23            } else { // esiste quindi devo fare l'
                  aggiornamento
24                if(exist.getHis ()) { // se il record e' da
                      storicizzare allora lo inserisco anche
                      nella tabella degli storici
25                    HmiTrkHis newHis = new HmiTrkHis();
26                    newHis.setTrkUid (exist.getUid ());
27                    newHis.setTss (exist.getTss ());
28                    newHis.setTrkVal (exist.getTrkVal ());
29                    newHis = HmiTrkHisDao.get ().safeInsert (
                          newHis, context, con);
30                } // if
31                SDateTime tss = new SDateTime ();
```

```
32              exist.setTss (tss);
33              exist.setTrkVal (toSet.getTrkVal ());
34              cache.setHmiLastUpdate (tss); // setto il
                    valore di tss nella cache
35              toReturn = HmiPrpTrkDao.get().safeUpdateAll (
                    exist, context, con);
36          } // if - else
37          ConnectionsUtil.commit (con, connection);
38          return toReturn;
39      } catch (SException e) {
40          ConnectionsUtil.rollback (con, connection);
41          throw e;
42      } finally {
43          ConnectionsUtil.safeClose (connection == null ?
                con : null);
44      } // try - catch - finally
45  } // set
```

La di codice precedente indica invece un tipico servizio di gestione di una richiesta *rest* da parte del *server*.
La struttura del codice rispetta lo standard di **SanMarco Informatica S.p.A** e quindi opera seguendo tutte le chiamate effettuate dalle classiche applicazioni sviluppate in loco.
In questo caso specifico abbiamo una richiesta che arriva sull'indirizzo **/spec/hmi/HmiPrpTrk/set/** che si occupa di richiamare la classe del *web service* che, a sua volta, richiama la classe della logica dove risiede la seconda funzione che è possibile visionare.
All'interno di questa classe vi sono tutte le funzioni che richiamano direttamente il **Dao** che effettua le interrogazioni al *database*.
Se nessuna delle chiamate ha generato eccezioni si procede a restituire un oggetto al *client*. in caso contrario vengono effettuate delle *rollback* che annullano tutte le operazioni, chiudono la connessione e generano un errore che viene restituito al client.

5 Valutazione Retrospettiva

5.1 Obbiettivi raggiunti

5.1.1 Requisiti imposti dall'azienda

Gli obbiettivi imposti dall'azienda erano di due tipi:

- **Formativi**:
  - **Mimini**: Lo studente dovrà acquisire le competenze di base sulla famiglia di software denominati "SCADA"
  - **Massimi**: Lo studente dovrà acquisire competenze avanzate sulla suddetta famiglia di software e i relativi campi di utilizzo.
- **Produttivi**:
  - **Minimi**: Analisi e documentazione
  - **Massimi**: Sviluppo prototipo al fine di testare le librerie selezionate e mole di accessi al database.

Per quel che riguada quelli **formativi**, entrambi, massimi e minimi, sono stati raggiunti nella prima settimana e mezzo circa di stage nella quale, soprattutto grazie alla riunione interna, sono riuscito a documentarmi pienamenamente e in maniera esaustiva su questa particolare famiglia di sfotware maturando una conoscenza piuttosto ampia su tutti i requisiti degli stessi.
Per quel che riguarda quelli **produttivi**, quelli minimi sono stati raggiunti con la stesura della tabella dei *benchmark* delle librerie e con la stesura dei reqsuisiti e casi d'uso.
Entrambe le attività si sono rivelate utili e fondamentali al fine di riuscire a sviluppare l'applicativo finale con in mente, in maneira piuttosto chiara, lobbiettivo di sviluppo.
Quelli massimi sono stati raggiunti con lo sviluppo e l'effettivo test dell'applicativo che, al netto di qualche errore di forma e di implementazione, si è dimostrato funzionale e funzionante.

5.1.2 Requisit personali

I miei requisiti, nonché obbiettivi, personali durante la permanenza a **SanMarco Informatica S.p.A** per il periodo di stage sono stati:

- Sviluppare una certa dimestichezza con le tecnologie *front-end*
- Familiarizzare con le *best pratice* in merito allo sviluppo di applicazioni web lato client
- Riuscire a inserirmi all'interno di una realtà aziendale complessa in modo da interfacciarmi al meglio con il futuro mondo del lavoro e dello sviluppo.

Alla fine dellostage posso affemrare di essere riuscito in tutti i miei obbiettivi completando con un discreto successo i compiti assegnatomi e acquisendo tutte le conoscenze che volevo acquisire al momento dell'inizio del progetto.

5.2 Bilancio Formativo

Questo stage mi ha portato a comprendere al meglio il mondo dello sviluppo di un applicativo web sul lato del *front-end* e mi ha dato modo di impratichirmi molto all'interno di un mondo che non è di certo il mio.
Mi ha, inoltre, concesso la possibilità di addentrarmi all'interno del mondo dell'industria 4.0 e di scoprire e affinare le mie capacità in merito ai **SCADA-HMI**[g] .

5.3 Distanza tra formazione acquisita e formazione necessaria

All'inzio dell'esperienza di stage le mie abilità erano prettamente collegate agli studi universitari e a qualche progetto personale che mi ha affinato le conoscenze del mondo *back-end*.
In particolare avevo qualche base di *Java* dovuta al corso di Programmazione Concorrente e Distribuita e numerose nozioni di *HTML5* dovute

al corso di Tecnologie Web.
Mi sono tornate utili anche le conoscenze acquisite all'interno del corso di Basi di Dati nonostante il tipo di interrogazioi al database fosse molto differente e distante da quello studiato durante il corso di laurea.
Infine il corso di Ingegneria del Software, e in particolare il progetto, mi hanno dato modo di arrivare allo stage abbastanza preparato sulle tecnologie da utilizzare (in particolare **Angular**).
Il tutor aziendale e tutto il team di lavoro mi sono stati molto d'aiuto nel riuscire a comprendere tutto il sistema con il quale andavo ad interfacciarmi. Mi hanno, inoltre, permesso di affinare tutte le conoscenze pregresse mostrandomi un *way of working* completamente nuovo che mi ha portato a migliorare di molto il mio modo di approciarmi a questo tipo di progetti.

5.4 Conclusioni

Lo stage in questione mi ha permesso di affinare tutte quelle conoscenze basilari che avevo in ambito di analisi e sviluppo sfotware e mi ha permesso di interfacciarmi ad un mondo, quello del *front-end*, a me del tutto nuovo e con il quale, per preferenze personali, non ho mai avuto troppo a che fare.
È stato interessante applicarmi ad un ambito differente da quello del *back-end* con il quale mi trovo molto più a mio agio ed è stato molto istruttivo imparare a gestire tuttu i requisiti del caso.
Nel futuro prossimo prevedo di prolungare la mia esperienza in **SanMarco Informatica S.p.A** al fine di continuare a migliorare le conoscenze ottenute durante il periodo di stage.
In seguito mi piacerebbe poter seguire un corso da *game designer* per potermi spostare su un campo a me molto più affine e nel qale avrei molto piacere ad iniziare una carriera.

6 Glossario

Benchmark: Con il termine benchmark si intende un insieme di test (prova) software volti a fornire una misura delle prestazioni di un computer per quanto riguarda diverse operazioni.
Dialogo Operatore: Applicativo con il quale un operatore di macchina di interfaccia al fine di comunicare con l'impianto e di ricevere informazioni dallo stesso.
End-to-end: Il principio end-to-end afferma che, se si hanno due applicazioni che comunicano tramite una rete, tutte le funzioni e le operazioni specifiche richieste da tali applicazioni, come il controllo di errori, devono essere realizzate ed eseguite in modo completo nei nodi terminali (o end point) e non nei nodi intermedi (o intermediate node) della rete. In caso contrario, cercando di ottenere delle funzioni operando sul canale di trasmissione invece che sui nodi estremi, si potrebbero non soddisfare i requisiti delle applicazioni oppure si potrebbe penalizzare le applicazioni presenti su altri nodi della rete.
ERP: Enterprise resource planning (letteralmente "pianificazione delle risorse d'impresa", spesso abbreviato in ERP)[1] è un sistema di gestione, chiamato in informatica "sistema informativo", il quale integra tutti i processi di business rilevanti di un'azienda (vendite, acquisti, gestione magazzino, contabilità ecc.).
Information technology: 'espressione tecnologia dell'informazione, in acronimo TI (in inglese information technology, in acronimo IT), indica l'utilizzo di elaboratori e attrezzature di telecomunicazione per memorizzare, recuperare, trasmettere e manipolare dati,[1] spesso nel contesto di un'attività commerciale o di un'altra attività economica.
PLC: Il Controllore a Logica Programmabile o Programmable Logic Controller (PLC) è un controllore per industria specializzato in origine nella gestione o controllo dei processi industriali. Il PLC esegue un programma ed elabora i segnali digitali ed analogici provenienti da sensori e diretti agli attuatori presenti in un impianto industriale.
Quadro sinottico: Quadro di riepilogo che mostra, in maniera sintetica, più informazioni fruibili dall'utente.
Web app: n informatica l'espressione applicazione web, ovvero web ap-

plication in inglese, indica genericamente tutte le applicazioni distribuite web-based.

Widget: Un widget, in informatica, nell'ambito della programmazione, è un componente grafico di una interfaccia utente di un programma, che ha lo scopo di facilitare all'utente l'interazione con il programma stesso. In italiano è detto congegno[1] (o elemento) grafico; può essere una vera e propria miniapplicazione.

Riferimenti bibliografici

[1] SanMarco Informatica S.p.A [Descrizione],
2017
http://www.info-centric.com/

[2] Manufacturing Enterprise Solutions [MESA definition],
Wikipedia, 2017b
https://en.wikipedia.org/wiki/Manufacturing_Enterprise_Solutions_Association

[3] Sistemi SCADA, Supervisory control and data acquisition,
Stefano Bimbo - Enrico Colaiacovo,
APOGEO, 2006
http://www.apogeonline.com/libri/88-503-1042-0/scheda