Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingeneria Informatica

Tesi di Laurea Specialistica

Un sistema di monitoraggio e controllo remoto per dispositivi industriali

Relatore: Carlo Ferrari
Correlatore: Fabio d'Alessi

Laureando: Enrico Carlesso

14 Marzo 2011

Alla mia famiglia.

Sommario

Questa tesi ha lo scopo di introdurre e valutare strumenti tecnologicamente innovativi per il monitoraggio dei reparti di produzione in campo industriale.

Il tema è noto in letteratura con l'acronimo SCADA, **S**upervisory **C**ontrol **A**nd **D**ata **A**quisition, ovvero controllo di supervisione ed acquisizione dati. Nelle realtà delle aziende di produzione il problema di un controllo continuo e costante è di fatto una necessità. Spesso i processi produttivi si svolgono 24 ore su 24, tutti i giorni, rendendo di fatto impossibile un controllo di monitoraggio classico.

Dotare la propria catena di produzione di strumenti che permettano la visualizzazione in maniera sintetica e precisa dell'andamento della produzione è uno strumento essenziale per valutare problemi sul nascere, siano essi legati ad un'area sia più specifcatamente ad un macchinario in particolare.

Inoltre è possibile automatizzare la creazione di report, se non addirittura eliminarli, permettendo ai supervisori dei processi produttivi di avere sempre - e dovunque - accesso alla panoramica completa dei dati relativi alla produzione costantemente aggiornati.

Indice

Sommario

Capitolo 1

Introduzione

In questo lavoro di tesi vogliamo valutare lo stato attuale degli strumenti per il monitoraggio delle attività produttive e presentare le alternative tecnologicamente innovative che sono state valutate ed implementate.

1.1 Introduzione ai sistemi SCADA

L'analisi di casi esemplari di applicazioni volte alla risoluzione dei problemi di supervisione e di controllo rende evidente l'utilità, se non addirittura la necessità di sistemi SCADA. Gli esempi più lampanti provengono da ambienti produttivi caratterizzati da elementi tecnologici e organizzativi tra loro molto diversi ma che consentono di condurre un'analisi destinata a delineare gli aspetti che qualificano un sistema come SCADA.

Come prima e superficiale introduzione all'argomento si può guardare al significato dell'acronimo:

S upervisory

C ontrol

A nd

D ata

A cquisition

che si può tradurre con *Acquisizione dati, supervisione e controllo*, sintetizzando così le tre funzioni principali di questo sistema. In uno SCADA l'acquisizione dati è funzionale allo svolgimento delle funzioni di supervisione,

osservazione dell'evoluzione del processo controllato, e di controllo, attuazione di azioni volte alla gestione degli stati nei quali il processo controllato si trova e delle transizioni tra gli stati nei quali il processo può venire a trovarsi.

Questa definizione è però molto generale, e non ci permette di farci un'idea chiara delle possibili applicazioni. Presentiamo quindi due esempi classici in letteratura, ampiamente descritti e presentati in [1].

1.1.1 SCADA nella gestione di un depuratore di acque reflue

Da quando l'uomo ha iniziato a vivere in gruppo formando nuclei sociali, la necessità di mantenere i luoghi di convivenza nel migliore stato possibile è sempre stata forte. Uno dei primissimi problemi è stato lo smaltimento delle acque piovane e dei liquami prodotti, con il primario obiettivo porre controllo alla diffusione delle malattie. Un depuratore destianto al trattamento delle acque si compone principalmente di tre agenti:

- la rete di raccolta delle acque, distribuita nel territorio;
- il depuratore vero e proprio che costituisce la parte centrale del sistema di depurazione;
- la reimmissione delle acque ripulite nei fiumi o nei mari.

La parte di raccolta delle acque costituisce sicuramente la parte più articolata del sistema. Tutti i punti nel territorio ove esistono emissioni di liquami e dove viene raccolta la pioggia devono essere individuati e devono essere adeguatamente connessi e convogliati fino al depuratore centrale.

Gli impianti periferici sono quindi distribuiti nel territorio, con la forte possibilità di espansione geografica molto vasta. È quindi opportuno avere mezzi di controllo per tutti gli organi che lo compongono. Questa funzione veniva in passato svolta da esseri umani, ma con l'avvento tecnologico sono sempre più i sistemi di acquisizione dati ad eseguire questo compito. I dati acquisiti possono essere immagazzinati localmente e sincronizzati col server su richiesta, o con cadenza periodica, oppure possono essere trasmessi in tempo reale al centro di controllo, qualora sia presente un canale di comunicazione stabile.

Il depuratore che raccoglie tutte le acque provenienti dalla rete fognaria realizza la vera funzione di depurazione. Questo processo coinvolgerà una serie di apparecchiature tecnologiche tra loro interconnesse e controllate da un sistema di monitoraggio.

La concentrazione presso un unico centro di controllo di tutte le informazioni disponibili in impianto permette di effettuare una corretta gestione del depuratore. Questi sistemi rendono disponibili tutti i dati acquisiti a ulteriori livelli di controllo che realizzano le seguenti funzioni:

- visualizzazione dei valori acquisiti;
- fornitura dei dati grezzi alle logiche di gestione automatica del depuratore;
- archiviazione dei dati per analisi e elaborazioni successive.

Per rendere una visualizzazione significativa e immediata dei dati raccolti, si fa spesso uso di strumenti quali il *quadro sinottico*[1] per rendere un immediato riepilogo della situazione generale.

Con l'avanzare della tecnologia si stanno progressivamente sostituendo questi *quadri sinottici* con monitor di computer, dotati di un'espressività decisamente maggiore, fino ad arrivare ai *video wall* che arrivano alle dimensioni di pareti intere. Anche i vecchi *banchi di controllo* vengono sempre più spesso sostituiti con sistemi computerizzati.

Questa evoluzione tecnologica trae origine da diverse cause. In primo luogo è evidente la diminuzione dei costi: un computer può da solo sostituire svariati *quadri sinottici* e *banchi di controllo*, rendendo anche molto più agevole la manutenzione delle macchine di controllo. Inoltre possono meglio adattarsi all'aumentare delle informazioni che l'operatore si trova a gestire, spostando quelle che prima dovevano essere delle modifiche "hardware" con delle modifiche "software", guadagnando in termini di costo ma soprattuto in termini di scalabilità.

1.1.2 SCADA nella gestione del traffico ferroviario

Uno scalo ferroviario può essere descritto, in generale, come un sistema di gestione del flusso dei convogli dotato di un insieme di linee di ingresso, un insieme di linee d'uscita e un terzo insieme di linee interne allo scalo.

Il compito della gestione del traffico consiste nell'associazione di una linea d'ingresso e una linea d'uscita alla stessa linea interna in modo da generare continuità tra ingresso e uscita.

[1] Un quadro sinottico è una presentazione in termini visivi della situazione di un sistema. Spesso vengono sviluppati su delle mappe più o meno stilizzate che raffigurano la dislocazione geografica dei componenti, con output luminosi per indicare lo stato di ogni nodo

Nel passato queste operazioni erano eseguite da operatori sotto indicazioni dei responsabili del traffico. Il loro compito consisteva nel posizionamento corretto degli *scambi ferroviari*, che non sono altro che sistemi di deviazione che connettono un tratto di ferrovia con due tratti successvi, permettendo di decidere che strada far prendere al treno.

Con questa modalità inizialmente le comunicazioni tra i diversi operatori erano effettuate attraverso un sistema complesso di segnalazioni gestuali. La prima evoluzione di questo sistema consistette nell'installazione di semafori atti alla segnalazioni delle autorizzazioni di movimentazione.

In questo modo i responsabili del traffico hanno potuto delegare al semaforo il compito di trasmettere le informazioni, comandando gli stessi con dei quadri elettrici con degli interruttori per controllare i segnali luminosi.

Il secondo passaggio di sviluppo è stata l'introduzione di sistemi semiautomatici di gestione del traffico che possono essere considerati dei sistemi **SCADA**. Parallelamente agli attuatori di scambio fisico sui binari sono stati installati sensori per la raccolta delle informazioni relative alla posizione di ogni scambio ferroviario presente nello scalo.

Già a questo step possiamo individuare le tre componenti principali di uno **SCADA**:

acquisizione dati: acquisizione degli stati realtivi agli scambi ferroviari e al sistema di semafori

supervisione: visualizzazione dello stato di esercizio dello scalo basata su dati acquisiti

controllo: gestione centralizzata della posizione degli scambi ferroviari e delle segnalazioni del sistema di semafori

In figura 1.1 è riportata una rappresentazione schematica del sistema di controllo del traffico ferroviario, e si possono notare le due componenti principali: il quadro sinottico ed il banco di controllo.

Nel quadro sinottico abbiamo una rappresentazione visuale dello stato globale del sistema, presentando solo le informazioni utili all'operatore, il banco di controllo invece è dotato di pulsanti e leve che permettono di inviare segnali elettrici ai sistemi di manovra e di segnalazione, cioè di comandare la variazione di posizione degli scambi ferroviari.

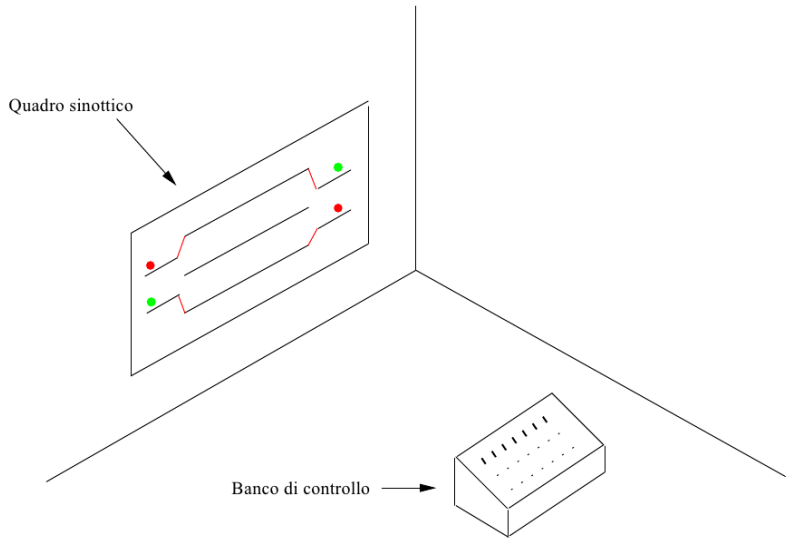


Figura 1.1: Sistema di controllo del traffico ferroviario

Capitolo 2

Architettura

Lo scopo di questo progetto di tesi è di rivedere e rielaborare l'architettura classica dei sistemi "SCADA", integrando nuovi sistemi tecnlogici. Si procederà presentando quindi l'architettura classica per poi introdurre le differenze che sono state apportate.

2.1 Architettura classica

In letteratura un sistema "SCADA" è suddiviso in una componente centrale ed una periferica. Nella componente periferica vengono posti i sistemi di acquisizione dati mentre il monitoraggio, controllo ed elaborazione dei dati viene affidato al Sistema di Elaborazione, nella componente centrale.

Tra le due componenti risiede lo strato di trasmissione dei dati che può variare a seconda dell'implementazione.

2.1.1 Acquisizione Dati

Gli agenti di acquisizione dati rappresentano lo strato dell'architettura "SCADA" più vicino al sistema che si vuole andare a controllare. Sono direttamente connessi agli attuatori del sistema, di qualsiasi forma essi siano.

Lo scopo di questi agenti è la traduzione in segnale comprensibile ed utilizzabile dal sistema "SCADA" tutte le grandezze fisiche che interessano al sistema di elaborazione (quantità di pezzi prodotti, allarmi, modalità di funzionamento...).

Questo layer, per sua natura, è fortemente legato al dispositivo al quale viene collegato, ed è impossibile delinearne una descrizione se non a livello molto generale.

A discrezione del produttore l'intefacciamento con questi dispositivi può essere di qualsiasi genere, a partire dal mezzo fisico di connessione, al protocollo di comunicazione.

2.1.2 Protocolli e tecnologie di comunicazione

Un punto cruciale, sebbene spesso non ne venga attribuita la giusta importanza, consiste nei veicoli dell'informazione tra le varie componenti del sistema.

Mentre in alcuni casi la scelta può risultare obbligata, in molti altri invece è necessario valutare approfonditamente le varie opzioni, non solamente in base ai requisiti del sistema, ma anche e soprattutto in base a quelle che possono essere le esigenze (o potenzialità) future.

Poniamo per esempio di dover realizzare un sistema che realizzi il monitoraggio di macchine di produzione. Spesso e volentieri apparecchi di questo tipo presentano un solo punto di accesso, e sovente capita che che quest'interfaccia si basi su connettori e protocolli proprietari. È quindi necessario interfacciarsi secondo le modalità previste dal produttore del macchinario, senza poter effettuare alcun tipo di scelta.

Esiste poi la connessione tra i dispositivi di accumulazione dati e la componente centrale del sistema. Numerose scelte possono essere fatte in base alla necessità di banda, alle distanze in gioco, ai luoghi e così via.

In passato era pratica comune utilizzare connessioni a banda limitata, come per esempio una connessione seriale *RS485*, che offre la possibilità di connettere più dispositivi sullo stesso cavo, come in un BUS, riducendo le spese di materiale ed installazione.

Questa soluzione presenta però almeno due grossi problemi. La *banda* è molto limitata, e non è pensabile trasmetterci quantità ingenti di dati, tantomeno trasmettere file. Inoltre deve essere scelto o, ancor più oneroso, implementato un protocollo che permetta una trasmissione concorrente, supportando l'*addressing* delle periferiche.

La strada che sta prendendo sempre più piede, e che è stata seguita anche nello svolgimento di questo lavoro, consiste nell'utilizzare l'infrastruttura tipica di una rete che segua le specifiche dell'**IEEE 802 LAN/MAN**[1] Questo permette di utilizzare una varietà di mezzi di comunicazioni che possano

[1] L'IEEE 802 LAN/MAN Standards Committee (LMSC) è una commissione dell'IEEE preposta a sviluppare standard per le reti locali (LAN) e per le reti metropolitane (MAN). Più precisamente, gli standard 802 sono dedicati alle reti che hanno pacchetti di lunghezza variabile (sono escluse le reti basate su cella (cellula) di lunghezza fissa ed anche le reti isocrone, nelle quali i pacchetti sono spediti su base temporale periodica).

adattarsi ad ogni esigenza, permettendo anche di utilizzarli contemporaneamente. Tra i vari, ricordiamo i due più diffusi: *Ethernet* [802.3] e *Wireless Local Area Network* [802.11].

È inoltre possibile interconnetere direttamente queste reti tra di loro utilizzando la rete Internet, ovviando così la necessità di interconnettere sottoreti geograficamente molto distanti.

A discrezione del mezzo trasmissivo scelto, la banda di trasmissione risulta ordini di grandezza superiore a connessioni basate su protocolli seriali.

Inoltre esistono moltissimi protocolli utilizzabili su questo tipo di reti, partedo da implementazioni di basso livello che costruiscono la comunicazione su socket TCP/IP, quindi al layer di trasporto dello stack TCP, sia implementazioni che sfruttano protocolli di alto livello, come il protocollo *HTTP*, layer applicazione.

2.1.3 Sistema di elaborazione

Finora abbiamo visto come raccogliere i dati dalle varie componenti del sistema, ma questi dati devono essere elaborati dal sistema centrale in primo luogo per il salvataggio, ma soprattutto perchè i dati devono essere messi in relazione tra loro.

La fase di elaborazione si divide in tre parti:

- Gestione dati;
- Processi di elaborazione;
- Condivisione dell'informazione.

Per gestione dati si intende il complesso di funzioni destinate allo scambio dati con le apparecchiature periferiche, al trattamento dei dati per la generazione di un insieme di dati intelligibile per le funzioni di elaborazione e di rappresentazione, all'archiviazione dei dati grezzi e dell'informazione aggregata frutto delle elaborazioni.

Per elaborazione si intende tutto quanto responsabile della corretta interpretazione dei dati visti come insieme rappresentativo dello stato di evoluzione del processo controllato e dell'attuazione delle azioni di controllo.

Il terzo elemento è la produzione dell'informazione nalizzata alla fruizione da parte di sistemi esterni e dalle strutture di interazione tra operatori e sistema.

2.1.4 Interpretazione dei dati

Una volta messi a punto gli step precedenti, ci si ritroverà con una mole ingente di dati mediante i quali è possibile ricostruire lo storico del passato.

Ma cosa ci servono tutti questi dati? Spesso, quando viene affrontanto il concetto del monitoraggio, ci si ferma alla parte tecnica, senza valutare la vera importanza di tutto il sistema, che consiste proprio nella potenzialità delle informazioni che possono essere estratte dai processi di interpretazione dei dati.

Non è immaginabile poter sempre valutare la situazione del sistema dalla totalità dei dati che vengono raccolti, è vitale che venga progettato ed implementato un buon sistema di sintesi.

Si prenda l'esempio di uno scalo ferroviario, e più precisamente lo scambio di una rotaia. I dati che sono coinvolti nella meccanica alla base di questo elemento semplice di un sistema molto complesso sono innumerevoli, dallo stato del motore, alla tensione della corrente, alla temperatura di esercizio solo per nominarne alcune.

Durante il processo di supervisione dello scalo, però, l'utilità dell'informazione per l'operatore si può limitare a tre stati mutualmente esclusivi: *Posizione A*, *Posizione B* e *GUASTO*. La totalità delle altre informazioni è determinante per desumere questa informazione aggregata ed è vitale che sia mantenuta al fine di poter valutare più in dettaglio gli andamenti dei vari componenti, ma hanno un'importanza secondaria rispetto alla situazione dello scambio.

A valle di ciò è immediato vedere come lo storico completo di una tale quantità di dati, se correttamente analizzata, possa fornire una base di vitale importanza per effettuare analisi statistiche sull'andamento del sistema.

Grazie all'evoluzione tecnologica è oggi possibile immagazzinare grandi quantità di dati a costi ragionevoli permettedo una raccolta dati completa e continua nel tempo. Potendosi concedere il "lusso" di non dover selezionare i dati da registrare a priori diventa necessario utilizzare politiche di **Data Mining** molto accurate e perfezionate.

Come presentato in [2], il processo di strutturazione di Data Mining può essere schematizzato in cinque passaggi:

1. Business Objectives Determination;
2. Data Preparation;
3. Data Mining;
4. Analysis of Results;

5. Assimilation of Knowledge.

Attraverso un'analisi così articolata è possibile raffinare algoritmi, specifici per ogni applicazione, che permettano l'estrapolazione di dati rilevanti.

L'utilizzo di queste tecniche porta benefici enormi nella vita di un'azienda. Una corretta politica di analisi statistica e di previsione dei guasti, oltre a migliorare l'immagine dell'azienda, permette di razionalizzare i costi di manutenzione e allo stesso tempo può evitare di peccare di eccesso sovrastimando le operazioni necessarie.

2.2 Architettura implementata

Nel corso di questo lavoro di tesi sono state effettuate scelte precise per ciascun blocco del sistema.

Come premesso, per quanto rigurda l'interfaccia di *ultimo livello*, le scelte sono vincolate al produttore dell'hardware ed alla necessità o meno di aggiungere uno strato di conversione dei dati. Nelle sezioni *case study* [cap. 3] verranno approfonditi questi aspetti.

Nonostante le profonde differenze tra i due progetti che verranno presentati, è possibile delineare delle linee comuni che hanno accomunato la progettazione e lo sviluppo di entrambi.

La comunicazione tra il sistema periferico ed il sistema centrale è stato basato su connessioni di rete **TCP/IP**, per il protocollo utilizzato per l'interscambio dei dati la scelta è caduta su **HTTP**.

Queste scelte ci hanno permesso di porre poca attenzione alla topologia della rete, siano connessioni *LAN* o *WAN*, con o senza livelli di *routing*.

Le comunicazioni tra i client ed il server rispettano lo standard **XML**, strutturando le API di comunicazione secondo il patterm **REST**[2].

Per i sistemi di elaborazione centrale sono state utilizzate delle web-application, che con lo stesso *core* hanno permesso di esporre degli **entry-point** per le comunicazioni in arrivo dagli elementi remoti interfacciati all'hardware, fornire un'**interfaccia web** per l'analisi dei dati e contemporaneamente fornire delle **API** per i client che sono stati creati.

Un punto sul quale è stata posta particolare attenzione è la rappresentazione dei dati. Nonostante non fosse immediatamente necessario si è preferito, da principio, avere sempre la possibilità di poter esportare i dati in un formato che astraesse dall'applicazione specifica.

[2]Representational state transfer (REST) un tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web. Sfrutta le quattro operazioni base del protocollo HTTP (GET, PUT, POST, DELETE) dando significati differenti per lo stesso *path* a discrezione del *verbo* utilizzato.

Questo accorgimento ci ha permesso di interfacciarci al sistema attreverso molteplici modalità, potendo analizzare i dati sotto diversi punti di vista al procedere di ciascun progetto.

La possibilità di mantenere monitorata l'evoluzione dei dati e la relativa interpretazione degli stessi nella durata di tutto il progetto ha favorito un processo di miglioramento continuo e di sviluppo controllato.

Questo approccio rispecchia il **B**ehaviour-**D**riven **D**evelopment. Questo approccio, così come il modello **AGILE** sul quale si basa, più che una tecnica è una filosofia che modifica sia il processo di scrittura codice sia l'interazione col cliente.

Avvalendosi di procedure di *test* precise e basate su un linguaggio elementare, è molto semplice scrivere delle *test-suite* basandosi direttamente sulle richieste del cliente.

A valle delle considerazioni tecniche sullo sviluppo sono state effettuate delle scelte progettuali che ci permettessero di produrre applicazioni scalabili, estendibili al concetto di *CLOUD*

2.2.1 CLOUD

Con il termine **CLOUD Computing** vengono raggruppati tutti i servizi tipici di un'infrastruttura tecnologica che non richiedono all'utente finale la conoscenza dell'ubicazione geografica della risorsa in esame.

Questa tendenza ha preso piede nell'ultimo decennio e risulta particolarmente vincente. La maggior parte dei servizi che si basano sul concetto di *CLOUD* sono legati strettamente al mondo del *WEB* che risulta esserne il veicolo naturale.

Tra le varie sfaccettature del concetto di **CLOUD** la più importante per la nostra anlisi consiste sicuramente nel **S**oftware **a**s **a** **S**ervice. Sotto questo termine vanno tutti i *software* che sono *deployati*[3] su server connessi ad Internet e che vengono utilizzati principalmente attraverso comunicazioni *HTTP*.

Molti di questi software presentano un'interfaccia WEB fruibile attraverso un qualiasi *browser*, ma mon si limitano a questo. Il protocollo *HTTP* è utilizzato anche per lo scambio di dati tra applicazioni, attraverso delle **API**. In un'era in cui i dispositivi *mobile* (smartphone, palmari, tablet...) sono in forte espansione è pratica comune fornire *entry-point* per l'interscambio di dati in linguaggio generico.

[3]Con l'avvento della tecnologia anche la lingua subisce forti influenze ed il termine Deploy ne è un esempio. La traduzione letterale è schierare, distribuire, ma in ambito informatico prende il significato di installazione di un software in un server.

I formati più comunemente utilizzati sono *json*[4] ed *XML*[5]. Attraverso questi linguaggi possono essere serializzate le informazioni genericamente, permettendo all'applicazione che le richiede di elaborarle a proprio piacimento.

Non è possibile prevedere quali (e quanti) dispositivi si potranno, in futuro, connettere ad una certa applicazione. Strutturare una base che permetta l'interscambio di dati in linguaggio indipendente permette agli sviluppatori di applicativi *mobile* di scrivere i programmi nel linguaggio naturale del dispositivo - che varia da produttore a produttore - e interfacciarsi al servizio scambiando solo i dati, permettendo allo sviluppatore l'elaborazione e la rappresentazione grafica degli stessi.

Il processo di sviluppo dei due progetti presentati in questo lavoro ha seguito strettamente questi dettami permettendo, come vedremo nel secondo *case-study* [**??**:tapi] di interfacciare dispositivi *mobile* senza dover apportare alcuna modifica alla *web-application* centrale.

Il tema della scalabilità è sempre stato tenuto in forte considerazione [3]. In entrambi i progetti le possibilità di ampi sviluppi futuri hanno portato ad una strutturazione del software che permettesse, con il supporto hardware adeguato, di scalare di diversi ordini di grandezza.

[4]Acronimo di **J**ava**S**cript **O**bject **N**otation, basato sul linguaggio javascript ma indipendente da esso

[5]Acronimo di e**X**tensible **M**arkup **L**anguage, standardizzato dalla W3C, è un metalinguaggio di markup, ovvero che permette di definire regole per altri linguaggi di markup. L'XHTML, per esempio, basa la propria struttura sintattica sull'XML.

Capitolo 3

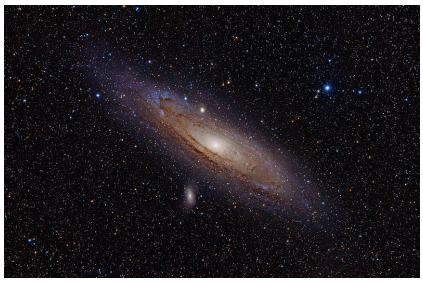
Case Study

Lo svolgimento di questa tesi è stato effettuato durante un tirocinio presso **M31 Italia Srl** con sede a Padova.

M31 è un'azienda nata nel 2006 come *spin-off* dell'Università degli Studi di Padova, è cresciuta negli ultimi quattro anni di attività. M31 nasce come un incubatore privato, con lo scopo di dar vita a nuove imprese ad elevata innovazione tecnologica.

Il nome *M31* deriva da Messier 31, ovvero il 31^o oggetto astronomico osservato da Charles Messier, più comunemente noto col nome di Galassia di Andromeda 3.1(b). Proprio al concetto di *costellazione*, richiamata anche nel logo 3.1(a), si ispira la politica dell'azienda.

(a) Logo di M31



(b) La galassia di Andromeda

L'azienda raccoglie e valuta proposte di progetti da parte di aspiranti imprenditori che si presentano con un'idea di business. Se l'idea viene accolta, M31 fornisce alla nascente *start-up* supporto finanziario, tecnologico ed amministrativo, facendola così diventare una *stella* nascente.

Lo scopo principale consiste nel togliere al gruppo operativo dell'azienda in fase di costituzione tutti i compiti che tolgono l'attenzione dallo sviluppo dell'impresa. All'interno di M31 esistono due squadre per questo scopo.

L'unità amministrativa si fa carico di tutta la burocrazia e la gestione aziendale, insegnando nel tempo ai futuri imprenditori tutto quello che è necessario sapere per una corretta gestione dell'impresa quando questa sarà autonoma e potrà staccarsi definitivamente in completa autonomia.

Il reparto tecnico, nome in codice **Zond**, è costituito da un team di 12 ingegneri altamente qualificati. Questo gruppo lavora nella ricerca e sviluppo di soluzioni tecniche e strategiche che sono al cuore di **M31**. Vengono realizzati progetti per conto terzi in diverse modalità:

Consulenza: l'esperienza maturata dal team consente di poter contribuire in maniera essenziale e molto efficacie nelle fasi di progettazione e pianificazione di soluzioni tecnologiche.

Tech Booster: tutti i membri del gruppo sono degli ottimi programmatori. Questo porta il team a molteplici richieste di affiancamento interno per le aziende che vogliono adottare o approfondire conoscenze nelle aree di forza di **Zond**:

- Qt programming
- Ruby and Python programming
- Linux system
- Advanced UI development
- Web Development (Ruby On Rails and Django)
- Web and Cloud infrastructure
- Network infrastructure
- Embedded development
- Embedded development with Qt

Partnership: molte aziende affidano lo sviluppo dei propri progetti completamente a **Zond**. Dopo un'attenta fase di raccolta dei requisiti informativi, il gruppo si fa carico di tutta la progettazione, svolgendo lavori di alta qualità, punto chiave del successo dei partner e del gruppo stesso.

Grazie a queste prerogative, negli anni è stato possibile supportare attivamente tutte le aziende che sono nate all'interno di **M31**:

Adant: sviluppa sistemi d'antenna adattativi che migliorano le prestazioni delle comunicazioni wireless di ultima generazione;

Adaptica: progetta e produce componenti e sistemi di ottica adattativa, elementi ottici deformabili e device optoelettronici ad alte prestazioni;

CenterVue: progetta e realizza avanzati sistemi diagnostici inteconnessi al web che permettono lo screening di malattie sistemiche e specialistiche che si manifestano nell'organo-occhio;

Si14: sviluppa e progetta computer embedded, sistemi e soluzioni di elettronica integrate dai componenti software e hardware personalizzabili;

Uqido: sviluppa servizi web-based che mirano a facilitare la gestione della risorsa tempo per le attività commerciali, gli enti ed i professionisti.

Durante lo svolgimento del progetto all'interno di M31 è stato vitale avere a disposizione il supporto di questo team, le cui competenze vanno del campo dell'elettronica, alla realizzazione di software di alto livello, alla creazione e configurazione di reti complesse fino alle metodologie di *problem-solving*.

Un altro punto di forza consiste sicuramente nel fatto di far convivere tutte queste *start-up* sotto lo stesso tetto, fatto questo che permette uno scambio di idee e competenze completamente trasversale. Si può trovare la persona giusta a cui chiedere consigli di meccanica, di ottica, di gestione aziendale facendo pochi passi.

Come è naturale, esiste una forte collaborazione tra le aziende. Per il secondo *case study* che affronteremo, il progetto Tapì [3.2], le schede hardware sono state realizzate da **Si14**.

Questo è stato fondamentale sia nella fase di progettazione, sia nella fase di *deploy* del sistema, permettendo di ricevere risposte a domanda tecniche in minuti anzichè in giorni.

In questo ambiente sono state affrontate due implementazioni del concetto di **SCADA** per due aziende italiane: DATA4 e Tapì.

I due progetti, nonostante vertano sullo stesso argomento, presentano prerogative completamente diverse e si sono distinti molto per scelte implementantive, architettura e risultati.

3.1 DATA 4

DATA4 è un'azienda nata nel 2004 con alle spalle una storia ed un'esperienza ventennale nella progettazione, produzione e distribuzione di sistemi self-service per il pagamento di beni e di servizi con denaro contate ed in forma elettronica.

La missione aziendale di DATA4 è riassunta nelle seguenti attività. Progettare, produrre e distribuire hardware e software per:

- sportelli automatici dedicati alla riscossione dei ticket sanitari e delle prestazioni in libera professione in ambienti non presidiati;
- sportelli self-service dedicati alle pubbliche amministrazioni locali per la riscossione diretta delle utenze (acqua, gas, multe, ICI, diritti di segreteria...);
- casse automatiche per la gestione ottimale dei servizi universitari in ambienti non presidiati;
- sistemi di cambiavalute e bancomat;
- chioschi informatici erogatori di informazioni, certificati e documenti.

Lo scopo del progetto consiste nella realizzazione di piattaforme generiche adibite alla riscossione ed erogazione pagamenti. Per questo progetto abbiamo dovuto interfacciarci ad un "Totem" con le seguenti periferiche:

- touchscreen;
- erogatore di banconote;
- erogatore di monente;
- incameratore di banconote;
- incameratore di monete;
- lettore POS;
- lettore RFID.

In figura 3.1 è possibile vedere uno dei totem prodotti da DATA4.

DATA4 ha inoltre sviluppato un'architettura software per esporre tutti i dispositivi sopraelencati con un'unica interfaccia di comunicazione basta su standard XML.

Figura 3.1: DATA4: Blu Cash

Quello che si voleva realizzare era la possibilità di generalizzare il "totem", in modo che fosse possibile configurare e modificare i servizi offerti dopo la sua installazione.

Altro requisito importante consiste nel prevedere il fatto che i "totem" non siano direttamenre raggiungibili, quindi tutte le comunicazioni client/server devono necessariamente effettuate dal client al server, che dovrà quindi accodare tutti i comandi da inviare al client ed attendere la comunicazione spontanea successiva da parte del client.

3.1.1 Strumenti software

Da specifiche di progetto il sistema doveva funzionare sotto Windows XP Embedded. Il sistema di interfacciamento, denominato WinMacc5, era stato appositamente scritto per questa configurazione.

I requisiti richiedevano che ogni client comunicasse con un server centrale appositamente studiato con lo scopo di accumulare informazioni e fare

da interfaccia esecutiva con i vari "totem", ma potessero altresì lavorare in maniera autonoma in caso di assenza di connettività.

Dal punto di vista dell'usabilità il server ed il client risultano praticamente identici, con la sola differenza che dal client è possibile controllare solo la macchina stessa, mentre dal server si possono controllare tutti i totem prodotti e attivamente collegati al sistema.

Col procedere dello studio iniziale si è reso sempre più evidente che non esisteva alcun motivo per differenziare completamente la parte server e la parte client del progetto. Si è deciso di strutturare il software come un'applicazione web, potendo così ridurre lo stato corrente dell'applicazione allo stato del database e potendo semplicemente trasferire lo stato - e lo storico - di un "totem" sul server replicandone il contentuto in database.

Il sistema "WinMacc5" si basa su un database "MsSQL 2005", vincolando così la scelta del back-end database per il progetto. Per esigenze di portabilità, visto che client e server dovranno girare su sistemi operativi differenti, si è quindi scelto di utilizzare un linguaggio intepretato ad alto livello che rendesse possibile uno sviluppo "AGILE".

Si è ridotta la scelta tra "Python" ed il relativo web-framework "Django" contro "Ruby" ed il framework "Ruby on Rails". La decisione è caduta sulla prima alternativa principalmente per la semplicità di interfacciamento col database "MsSql" preesistente.

3.1.2 Architettura

L'interfaccia all'hardware avviene attraverso "WinMacc5" col quale la comunicazione viene effettuata via comandi strutturati con la sintassi XML inviati su un Socket locale. Il progetto necessita quindi di implementare ed esporre tre servizi principali:

1. WebServer per il servizio principale, denomianto "Scada4";
2. TCPSocket per la comunicazione con WinMacc5, denominato "WinMaccFactory";
3. PeriodicalJob per la sincronizzazione col server, denomianto "Poller".

Tutti questi agenti devono comunicare tra loro, è quindi una scelta naturale utilizzare uno strumento che possa soddisfare tutte le necessità raggruppandole sotto un unico macroprocesso.

Tra i vari framework per applicazioni network la scelta è caduta sul framework *event-driven* "Twisted"[4], decisamente maturo e versatile, permettendo alle diverse parti del progetto di comunicare direttamente tra loro.

Scada4

Sotto il nome di "Scada4" va tutto l'applicativo web-based per la gestione della macchina e la sincronizzazione tra server e client. Grazie alla struttura ad applicazioni di "Django" si è potuto suddividere il progetto in sottoprogetti, mantenendo ordine e separazione logica tra i vari componenti.

Il nucleo fondamentale di tutto il progetto è il concetto di **Operazione** ed **Azione**. Le Operazioni rappresentano le i possibili comandi che è possibile inviare al sistema, mentre le Azioni rappresentano un'esecuzione di un'Operazione, mantenendone parametri e risposte del sistema.

Ogni modifica allo stato del sistema è conseguente ad un'Azione istanza di un'Operazione. Possono essere suddivise in quattro categorie principali, che sono figlie di una Operazione generica che descrive l'interfaccia specificando le azioni che devono essere esposte.

Interfaccia Operazione

Un'Operazione consiste nei seguenti attributi e funzioni di istanza:

name: il nome dell'operazione;

description: la descrizione dell'azione;

signature: la firma dell'azione, ovvero un nome univoco che la intentifica e ne descrive i parametri;

can_be_done(User, Totem): restituisce la possibilità o meno, per un determinato utente di poter eseguire l'operazione;

Interfaccia Esecuzione

totem: il riferimento al Totem sul quale deve essere eseguita l'Operazione;

executor: riferimento all'utente che sta eseguendo l'operazione;

executor_ip: l'IP dal quale viene ricevuta la richiesta di esecuzione;

is_local: valore booleano che determina se l'azione è eseguita direttamente sul totem o proviene dal server;

to_be_executed: specifica l'ora ed il giorno nel quale si vuole venga eseguita l'azione;

status: riferimento allo stato in cui è l'azione, descritta dal modello *ExecutionState*;

created: data di creazione dell'entry in database;

updated: data di ultima modifica dell'entry in database;

repeatability: specifica se l'Operazione deve essere ripetuta nel tempo;

frequency: assieme a rrule specifica i termini di ripetitività in maniera conforme all'RFC2445[5];

rrule: stringa nel formato RFC2445;

son_of: nel caso di azione ripetuta, viene generata un'azione che non genera eventi, ma ha il solo compito di generare altre azioni. Le azioni generate in tal modo avranno questo campo che punta all'azione generatrice;

mark_sent(): imposta lo stato a inviata, ovvero in attesa di risposta da WinMacc;

mark_executed(): imposta lo stato a eseguito con stato positivo, in seguito ad una risposta da WinMacc;

mark_failed(): segnala lo stato di esecuzione fallita, in seguito ad un errore, all'impossibiltà di eseguire l'operazione o dovuta ad un timeout;

remove(): rimuove l'Operazione dalla coda delle operazioni da effettuare;

get_next(after?): da invocare su un'azione generatrice di azioni periodiche, restituisce la prossima esecuzione in base alla ripetitività impostata, da ora o dopo after se passato come parametro;

pretty_repeat(): restituisce una versione *human-friendly* dell'intervallo di ripetizione, e.g. "Ogni giorno alle 14:35";

generate_next(): invocato su un'azione periodica, genera la prossima Esecuzione;

is_executable(): ritorna se l'azione in questione è eseguibile in quell'istante;

execute(): genera le ripetizioni delle Esecuzioni. Ogni implementazione di questa classe che sovrascriverà questo metodo dovrà esplicitamente richiamarlo.

Operazioni ed Azioni WinMacc

Sotto le azioni WinMacc vengono raggruppate tutte le funzioni che hanno a vedere con l'hardware del Totem. Per esempio la richiesta di lettura di un barcode avrà la seguente sintassi:

Listato 3.1: Data4: Xml per la richiesta di lettura barcode

```
<request-message sender="frontend">
  <DATA command="PERIF_REQUEST">
    <PARAMETER name="peripheral" value="BARCODE"/>
    <PARAMETER name="command" value="READ">
      <ITEM name="timeout" value="15"/>
    </PARAMETER>
  </DATA>
</request-message>
```

Come si può vedere, è un'operazione di tipo PERIF_REQUEST, e tutte le richieste di questo tipo contengono il nome della periferica da interrogare e il comando con gli eventuali parametri.

Tutte le Operazioni da inviare a WinMacc sono di questa struttura, il che ci ha permesso di modellizzare in maniera lineare il database per serializzare le informazioni.

Le implementazioni quindi delle classi di Operazione ed Esecuzione ricalcano la seguente struttura:

Funzioni: implementazione Operazioni WinMacc

needs_admin: specifica se questa Operazione può essere fatta mentre la macchina è in *Admin Mode*;

device: riferimento al tipo di periferica sulla quale si vuole eseguire l'operazione;

command: nome del comando richiesto (in [3.1] è "READ");

expected_replies: definisce il numero di messaggi che ci si aspetta da WinMacc in risposta;

get_form(): genera il form HTML che presenta le opzioni corrette per l'esecuzione di questa operazione in base ai parametri che è possibile fornire;

Azioni: implementazione Esecuzioni WinMacc

function: riferimento alla funzione che si vuole eseguire;

waiting_replay: stato che specifica se questa esecuzione sta aspettando o meno risposte da WinMacc;

generate_from_post(post_data): prende i parametri generati da un POST di un form relativo all'Esecuzione e genera la relativa "entry" in database;

generate_xml(): genera l'xml da inviare a WinMacc;

execute(): esegue l'Operazione, ovvero genera l'xml corretto e lo invia a WinMacc invocando WinMaccFactory;

Tutte le Funzioni eseguite necessitano dei parametri relativi. Come nell'esempio, sono sempre di forma PARAM ← ITEM. Sono stati quindi generati due modelli, Item e Param che assieme alla relativa entry in Function danno la descrizione globale del comando.

ActionParam e ActionItem mantengono la stessa struttura, ma sono relativi all'Esecuzione della Funzione, mantengono quindi anche il valore del parametro.

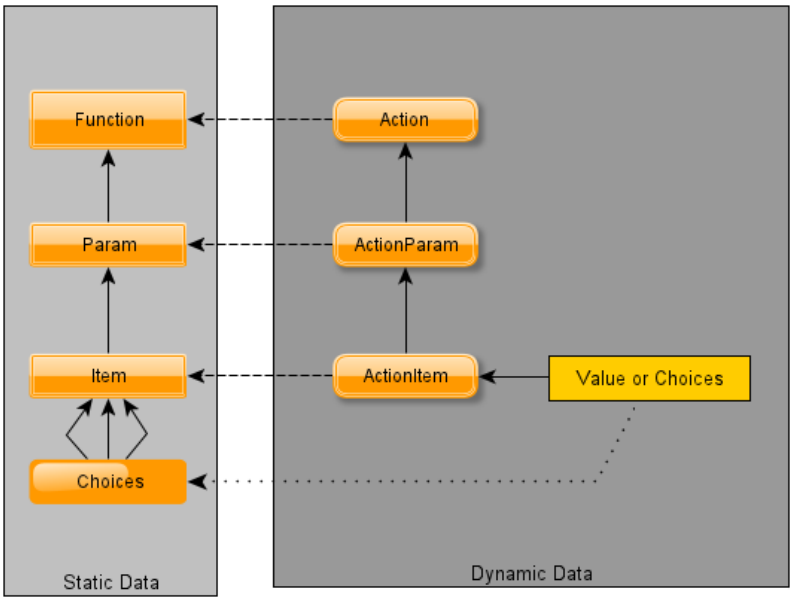


Figura 3.2: Data4: Schema di Operazioni e Funzioni per WinMacc

In figura 3.2 è schematizzata questa struttura, che ha la potenzialità di ricalcare una qualsiasi configurazione di comando XML secondo il 'protocollo' WinMacc.

Operazioni SCADA

Sotto il gruppo di operazioni denominate *operazioni SCADA* sono state inserite tutte le operazioni della gestione del sistema stesso.

Allo stadio implementativo attuale sono state implementate cinque operazioni:

go_admin_mode e leave_admin_mode: con queste due operazioni si imposta semplicemente il timeout della chiamata a Poller, come mostrato in 3.3. A livello pratico viene cambiato un valore nel database del server e del totem relativo, la flag che determina appunto lo stato del "Totem".

wdb_sync: questa funzione ha lo scopo di sincronizzare, in maniera incrementale, i database interni dell'applicativo "WinMacc". A tal scopo, sul server è replicata tutta la struttura di uno di questi database, aggiungendo una *foreign_key* per specificare il database al quale ci si riferisce e un *timestamp* temporale per poter verificare lo stato dell'ultima sincronizzazione.

wdb_full_resync: come l'azione precedente, anche questa riguarda la sincronizzazione dei database interni, ma non lo fa in maniera incrementale. Semplicemente serializza completamente il contenuto del database e lo invia al server, operazione che richiede un forte impiego di banda. Nella progettazione si è quindi deciso di implementare questa funzione e chiamarla una volta alla settimana per verificare il corretto allineamento dei dati tra totem e server.

vnc_request: un altro requisito fondamentale consisteva nella possibilità di effettuare connessioni vnc per amministrare i totem in maniera remota. Non essendo i totem direttamente raggiungibili abbiamo quindi dovuto effettuare delle connessioni *reversed*, ovvero istruendo il server *VNC* del totem a connettersi al client in ascolto dell'operatorore.

WinMaccFactory

Tutte le azioni che vengono intraprese dal client web, arrivino da interazione diretta o dal server, generano una richiesta che deve essere girata a "WinMacc", attraverso il LocalSocket esposto. Aprire e richiudere la connessione ogni volta che si presenta la necessità di questo tipo di connessione sarebbe uno spreco di risorse immotivato, non permetterebbe una corretta gestione della concorrenza e soprattutto non permeterebbe di rimanere in ascolto dei

messaggi spontanei provenienti da "WinMacc5", che può scrivere dei messaggi in risposta ad input dell'utente nelle varie periferiche (inserimento di monete, di banconote, presenza di badge RFID...).

Si sfruttano quindi le potenzialità del framework "Twisted" per creare un servizio ad eventi che permetta sia l'invio che la ricezione di comunicazioni in maniera *safe*.

Come accennato, è necessario avere un canale di comunicazione con "WinMacc" che permetta di inviare dei messaggi al software di interfacciamento con l'hardware, ma tutte le comunicazioni avvengono in maniera asincrona.

Questo significa che deve essere perennemente verificata la presenza di nuovi messaggi in arrivo, che devono quindi essere comunicati a Scada4.

Twisted mette a disposizioni due *utility* per gestire questo tipo di comunicazioni, ClientFactory e LineReceiver, che permettono - nella loro combinazione - di lasciare al framework il compito di monitorare il socket, istruendolo a chiamare delle *callback* al verificarsi di determinati eventi.

L'asincornia del sistema "WinMacc" richiede che tutte le comunicazioni effettuate siano *cieche*, ovvero non è possibile sapere se il comando è stato o meno preso in carico dal sistema.

Listato 3.2: Data4: Winmacc Protocol

```
class WinmaccProtocol(basic.LineReceiver):
    def connectionMade(self):
        self.factory.prot = self
        self.setRawMode()

    def rawDataReceived(self, data):
        recv = etree.fromstring(data)
        msg = WinmaccMessage()
        msg.totem = Totem.objects.get()
        try:
            msg.content = etree.tostring(recv, pretty_print = True)
        except:
            return
        msg.save()
        if msg.link_action():
            print "Request and reply correctly linked"
        else:
            print "Unable to link this request and reply"
```

Nel listato 3.2 si può vedere la semplicità del codice per la ricezione dei messaggi. Questo è reso possibile delegando tutta la logica a Scada4.

Alla ricezione di ogni messaggi viene creato in `WinmaccMessage` che viene popolato col contenuto liscio dell'*XML* in ricezione, viene salvato e quindi sincronizzato sul database.

Nonostante la natura spontanea dell'invio di un messaggio da parte di "WinMacc", alcuni messaggi derivano da eventi esterni al sistema, altri in-

vece sono risposte dei messaggi inviati da Scada4. A causa di questa asincronicità e alla mancanza di identificativi per le richieste scambiate, non è possibile assicurare un match prefetto tra richiesta e risposta, bisogna fare una *guess* tracciando le richieste fatte che attendono una risposta e le risposte in ingresso, verificando la compatibilità dei messaggi.

Lo sviluppo del progetto andava di pari passo con lo sviluppo di "WinMacc" da parte di DATA4, rendendo quindi alcune parti delle comunicazioni non affidabili o non ben precise.

Nel listato 3.3 si può vedere il componente Factory. Il componente interno `_wprotocol` è un'istanza di WinmaccProtocol 3.2, che viene collegata direttamente dal Protocol.

Listato 3.3: Data4: Winmacc Factory

```
class WinmaccFactory(protocol.ClientFactory):
    protocol = WinmaccProtocol
    _wprotocol = None

    def send(self, line):
        if not self._wprotocol:
            print "Protocol not available, calling in 3 seconds"
            reactor.callLater(3, self.send, line)
        else:
            pad = chr(0)*(2048-len(line))
            self.prot.sendLine(line+pad)

    def clientConnectionLost(self, connector, reason):
        print 'Lost connection.  Reason:', reason
        print 'Retrying in 5 seconds'
        reactor.callLater(5, connector.connect)

    def clientConnectionFailed(self, connector, reason):
        print 'Connection failed. Reason:', reason
        print 'Retrying in 5 seconds'
        reactor.callLater(5, connector.connect)
```

Il programma attende quindi che il protocollo sia correttamente connesso, ed è robusto a fallimenti e perdite di collegamento. L'errore viene loggato e il sistema viene riconnesso.

Poller

Un'applicazione web, per sua natura, non ha una *vita* propria. Ogni richiesta che viene fatta al server ha una propria vita, che inizia alla connessione dal client, fa tutte le operazioni necessarie alla costruzione della pagina di risposta e termina la propria esistenza quando risponde al client.

Questa architettura non permette un approccio di programmazione di tipo classico. Non si può, per esempio, mettere in sleep delle porzioni di

codice ed effettuare delle operazioni ritardate. Non fa parte dell'architettura delle applicazioni web.

Internamente, quindi, non è nemmeno possibile eseguire un comando in maniera ripetuta con una specifica cadeza.

Non c'è quindi alcun modo *nativo* per eseguire una porzione di codice ad un certo istante temporale.

Nella relizzazione di questo progetto però avevamo questa necessità in almeno due occasioni: le comunicazioni periodiche col server e l'esecuzione di comandi *schedulati*.

Per ovviare a questo inconveniente è stata realizzata la classe `Poller`.

Listato 3.4: Data4: Poller per sincronizzazione e delayed jobs

```
class Poller:
# Tempo tra una chiamata e l'altra, a discrezione se in modalita' admin o meno
    ts = 30
    ta = 1
    def __init__(self):
        self.loop = LoopingCall(self.poll)

    def start(self):
        self.loop.start(1, now = False)

    def stop(self):
        try:
            self.loop.stop()
        except:
            pass

# Fa una chiamata forzosa al ciclo, dovuta ad interazione diretta dell'utente
    def force(self):
        if self.loop.running:
            i = self.loop.interval
            self.loop.stop()
            self.loop.start(i)
        else:
            self.poll(True)

    def poll(self, one_shot = False):
        try:
# Viene cercato il totem installato relativo alla macchina.
# Se non esiste l'installazione non e' ancora avvenuta
            t = Totem.objects.get()
        except Totem.DoesNotExist:
            print "No Totem installed yet"
            return
        try:
# Viene chiamto il metodo poll() del Totem in Scada4
            t.poll()
# Cicla continuamente finche' non e' esaurita la sincronia
# col server
            while t.execute():
                t = None
```

```
                t = Totem.objects.get()
                t.poll()
43              t = None
                t = Totem.objects.get()
        except:
            pass
        if one_shot:
48          return True

        t = None
        t = Totem.objects.get()
# Verifica il delay con il quale viene invocata ogni iterazione
53 # del programma in funzione dello stato del Totem
        if t.admin_mode and t.last_activity() > self.ts:
            t.admin_mode = False
            t.save()
            t.poll()
58
        if t.admin_mode and self.loop.interval != self.ta:
            self.loop.stop()
            self.loop.start(self.ta)
            print "Rescheduled in admin mode"
63
        elif not t.admin_mode and self.loop.interval != self.ts:
            self.loop.stop()
            self.loop.start(self.ts)
            print "Rescheduled in normal mode"
68
# Rigenera le istanze delle azioni ripetute future
        t._regenerate_periodic()
```

In figura 3.3 è possibile vedere il ciclo di vita del `Poller`. Il primo passo è sempre la sincronizzazione col server, che può avere delle operazioni da far eseguire al totem. Quando vengono copiate sul database locale il totem verifica se ha operazioni da fare.

Se ne sono presenti le esegue ed invia la risposta al server, che nel frattempo può averne accumulate altre. Finchè il *burst* di azioni non termina il client rimane sempre responsivo.

Quando non ci sono più azioni da effettuare, il Poller determina se deve attendere un secondo o cinque minuti, a discrezione se sia o meno attiva la modalità *admin*[1].

3.1.3 Peculiarità del sistema

Nello svolgimento del progetto abbiamo dovuto tener conto della rigidità del software "WinMacc" col quale abbiamo dovuto interfacciarci. Questo ha reso

[1] A causa della comunicazione che parte dai totem verso il server, questi potrebbero essere molto poco responsivi, nel caso peggiore la risposta potrebbe arrivare dopo un ciclo intero del Poller. Per ovviare questo problema è possibile mettere il Totem in modalità *admin* o *superpoller* che riduce il tempo di attesa a un secondo.

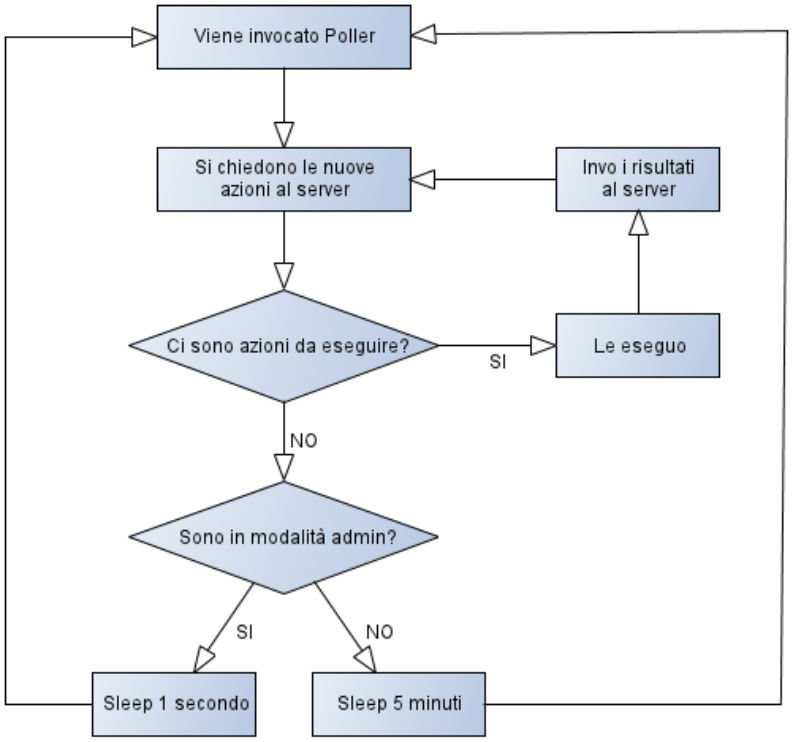


Figura 3.3: DATA4: Poller Life cycle

molto complessa la fase di progettazione del database e la modellizzazione delle strutture dati.

Ciò nonostante il risulato consiste in un'infrastruttura solida nella gestione delle comunicazioni e della sincronizzazione tra i vari agenti in gioco.

Il punto fondamentale consiste nella similarità della componente client e della componente server, che permette di replicare in maniera consistente i dati.

Dopo quattro mesi di sviluppo, però, DATA4 ha deciso di abbandonare il progetto. Durante lo sviluppo del progetto l'azienda ha subito una forte ristrutturazione interna che ha portato dei cambi del reparto di progettazione.

Nella revisione di questo progetto DATA4 ha deciso di cambiare strategia. Hanno pensato ad una nuova archiettura basata completamente sul concetto di *cloud*, mantenendo il client completamente ignorante.

A valle di questa decisione, ci è stata chiesta una proposta per la realizzazione del nuovo progetto. Il *team* di DATA4 ha però valutato migliore l'offerta fatta da un'altra azienda, chiudendo quindi il progetto in essere.

3.2 Tapì

Tapì nasce nel 1998 per rispondere alle esigenze del mercato per tappi sintetici d'alta qualità. Dal 2005 Tapì Group è riconosciuto come leader indiscusso nella produzione di tappi a "T"

Nel corso degli anni, Tapì Group ha ampliato le proprie strutture con stabilimenti in Italia, Argentia ed America settentrionale.

(a) Logo dell'azienda

(b) Esempio di tappo prodotto

È quindi comprensibile l'esigenza di poter tenere sotto controllo l'andamento della produzione in tutti gli stabilimenti in maniera centralizzata, così come mantenere sotto controllo la produzione nell'andamento della produzione nelle ore notturne, quando il reparto produttivo è monitorato con risorse ridotte.

3.2.1 Struttura del reparto produttivo

Come primo passo per la realizzazione di sistemi di controllo per tutti gli impianti dell'azienda, si è partiti con un progetto pilota nella sede di Massanzago (PD). In questa sede sono presenti undici presse che producono tappi, le quali sono connesse ad un PLC. Questi plc offrono una porta seriale attraverso la quale è possibile comunicare per interrogare lo stato della macchina.

Queste presse sono dislocate in tutta l'estensione dell'azienda, ed i PLC sono alloggiati all'interno di quadri elettici. Questi quadri, per normative sulla sicurezza, possono essere aperti solo se la pressa è spenta.

Nella sede è inoltre presente un ufficio amministrativo, al quale tutti i quadri elettrici sono connessi.

3.2.2 Infrastruttura implementata

Per monitorare tutte le presse si è deciso di utilizzare dei moduli hardware basati su processore ARM, uno per plc, che sono stati quindi connessi ad ogni plc con un cavo seriale RS232 e con un cavo di rete sono stati interconnessi in una Virtual Lan con un server centrale installato nell'ufficio amministrativo.

Le schede in questione sono dei moduli basati sul chip iMX27, che permettono di utilizzare la distrubuzione GNU/Linux OpenEmbedded, mentre per il server è stato acquistato un server HP [Intel Pentium G6950 (2 core, 2,8 GHz, 3 MB, 73W)], nel quale è stato installato il software accumulazione e elaborazione dei dati.

Moduli iMX

Per l'interfacciamento con i PLC sono stati scelti dei moduli **iMX27** [3.4], prodotti da *Si14*, che presentano le seguenti caratteristiche:

- basso costo;
- connettore SODIMM standard;
- Cpu prestante (200MHz);
- design pronto all'uso;
- nessuna necessità di connessioni JTAG;
- Windows CE 6.0 e Linux BSP disponibili;
- Consumo a regime inferiore a 1.5 Watt;
- Ampio range di temperatura di lavoro (da $-20°C$ a $+85°C$);
- 128MB di RAM disponibili;
- 128MB di spazio disponibile su disco a stato solido.

Per la realizzazione del sistema di controllo è stato utilizzato il linguaggio *Python*, che ha permesso tempi di realizzazione brevi e processi di debug estremamente rapidi.

I PLC forniscono, attraverso la porta seriale, un accesso diretto alle aree di memoria. Attraverso una sintassi relativamente semplice è possibile richiedere il contenuto di una specifica area di memoria, ad ognuna delle quali corrisponde un relativo valore dello stato della macchina. Ogni area di memoria è composta da una *word* di 16 bit.

Sono distinti tre categorie principali di informazioni disponibili:

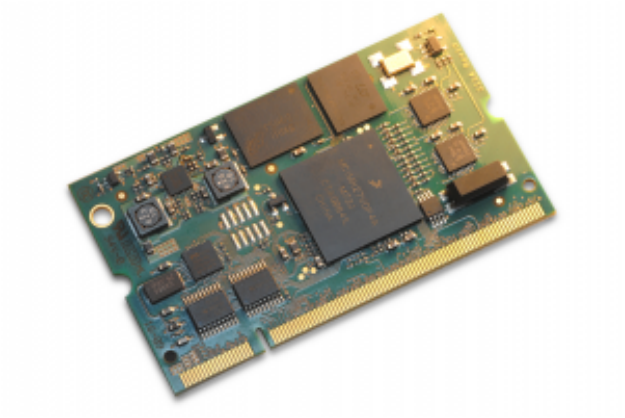


Figura 3.4: Tapì: Un modulo iMX27

Allarmi: Sono rappresentati con dei bit, banalmente allarme attivo/allarme non attivo;

Stati: Anch'essi rappresentati da dei bit, che indicano o meno il fatto di essere in un preciso stato. Sono mutalmente esclusivi in gruppi logici, lo stato del selettore, per esempio, può variare tra Manuale, Semiautomatico o Automatico. Uno ed uno solo di tre stati può essere acceso;

Contatori: Descrivono numericamente lo stato di un contatore, per esempio "Pezzi realizzati", "Pezzi da produrre", "Produzione tappi/ora in base al tempo di ciclo medio".... Può essere rappresenta da una singola Word o da una doppia Word.

Per non prevedere alcun tipo di specializzazione sui moduli, tutte le configurazioni delle aree di memoria (che differiscono da pressa a presa) sono mantenute sul server centrale, mentre i moduli sono tutti uguali. Vengono solo forniti di una configurazione di base (indirizzo IP del server, nome del modello...). Ogni volta che vengono riavviati chiedono al server la propria configurazione e solo in base ai dati restituiti dal server inizializzano le proprie richieste di lettura.

Riportiamo un esempio tipico di configurazione debitamente commentato:

Listato 3.5: File di configurazione del Core Tapì

```
#!/usr/bin/env python2

import os
```

```
# Set logging level. Set to False in production
DEBUG = False

# Set to True if want to daemonize by direct call
DAEMON = True

# Set to True if want log to be printend to stdout. Useless if DAEMON = True
LOG_TO_STDOUT = True

# If we want to emulate the serial behavior for testing pourpose
EMULATE_SERIAL = True

# Watchdog file
WATCHDOG_FILE = "/tmp/tapi.wd"

# Time in seconds that represent Watchdog life cycle
WATCHDOG_INTERVAL = 10

# Threshold sets for how many ALIVE_NOT_RESPONDING cycle hw watchdog should be updated
WATCHDOG_THRESHOLD = 5

# Log file for TaPi application
LOG_FILE = "/var/log/tapi.log"

# Executable core.py for respawn
CORE_PATH = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'core.py')

# Define the timeout of every check interaval
CHECK_INTERVAL = 3

# Update interval with respect to CHECK_INTERVAL (2 means every 2 CHECK_INTERVALS)
UPDATE_INTERVAL = 2

# Machine Unique ID
MACHINE_UID = ''.join(open('/sys/class/net/eth0/address').readlines()).replace('\n', '').replace(':', '-')

# Machine ID
MACHINE_IDENTIFICATION = "0204/03"

# Data for Tapi WebServer
TAPI_PORT = '3000'
TAPI_HOST = '172.18.0.1'
TAPI_REQUESTS_PATH = 'requests.xml'

# Api for authenticaton on Tapi WebServer
TAPI_API_KEY = 'FDt1WFp8dbPn065d6SUD'

# Serial port attached to PLC
SERIAL_PORT = '/dev/ttymxc0'
SERIAL_TIMEOUT = 2

# File where the configuration for the machine is stored
MACHINE_CONFIGURATION = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'configuration.xml')

# name => "Roto T1/24 EXL"
```

```
# name => "Roto T2/48"
# name => "Roto T2/96"
# Machine Model
MODEL_NAME = "Roto T1/24 EXL"

URLS = {
  'CONFIGURATION': 'http://%s:%s/machine_models.xml?name=%s' % (TAPI_HOST,↩
      TAPI_PORT, MODEL_NAME.replace(' ', '%20')),
  'PUSH_DATA': 'http://' + TAPI_HOST + ':' + TAPI_PORT + '/installations/↩
      current/machines/' + MACHINE_UID + '/' + TAPI_REQUESTS_PATH
}
```

Ad ogni ciclo di esecuzione del sistema vengono lette tutte le aree di memoria relativa specificate nella configurazione. Tutti i dati vengono immagazzinati in una struttura dati gestita come una coda che ad ogni *UPDATE_INTERVAL* viene condificata e serializzata secondo lo standard XML ed inviata al server.

Particolare attenzione è stata posta nel monitoraggio della vita del processo. Questi moduli non presentano interfaccie di input esterne, quindi se qualcosa va storto, che sia l'applicazione di acquisizione dati, il sistema operativo o problemi di rete, deve essere il sistema stesso a tornare ad uno stato di operatività.

Per assicurare il corretto funzionamento è stato realizzato un Watchdog[2] software con il compito di controllare i punti cruciali del sistema. Durante lo svolgimento del progetto lo sviluppo e la verifica del corretto funzionamento di questo componente ha causato diverse problematiche che verranno approfondite in appendice A.

Viene anzitutto verificata la data di modifica del file *WATCHDOG_FILE* verificando che sia entro una certa soglia.

Nel caso non sia così viene eseguito un check più approfondito, verificando nella lista di processi la presenza o meno del processo principale.

Nel corso dei primi test si è notato che capitava spesso che il controller ethernet andasse in blocco senza generare però dei blocchi globali. Abbiamo quindi aggiunto un test di ping verso il server centrale.

Se tutti questi controlli vanno a buon fine viene scritto il watchdog hardware, disponibile nelle distribuzioni linux in `/dev/watchdog`.

Questo watchdog viene inizializzato al momento del boot con un tempo di check di 120 secondi. Se il suo valore non viene aggiornato almeno una volta nei 120 secondi successivi ad una scrittura, il sistema subisce un reboot hardware. In questa maniera, avendo il sistema basato su una configurazione esterna e senza scritture locali, non si presentano inconsistenze nell'integrità

[2]Con il termine Watchdog si intende un sistema di temporizzazione hardware che permette alla CPU la rilevazione di un loop infinito di programma o di una situazione di deadlock

del disco, e possiamo quindi assicurare un tempo di uptime molto vicino al 100%.

Server - Elaborazione dati

Come premesso è stato installato un server centrale che facesse da accumulatore dei dati e potesse mantenere lo storico degli stati di tutte le presse.

Abbiamo scelto di implementare una web-application basata sul web framework *Ruby on Rails*. Tutte le comunicazioni dai moduli iMX27 avvengono attraverso il protocollo HTTP sfruttando la sintassi REST. Le richieste di configurazioni della pressa vengono effettuate con una chiamata GET, mentre gli invii di informazioni sullo stato avvengono con una chiamata POST.

Listato 3.6: Esempio di invio via XML di dati verso il server

```
<request id="0" identification="0204/03" version="0.1" created_at="↩
    2011-01-19 09:58:04 UTC" uid="00-15-f2-05-6a-a1" machine_model="Roto ↩
    T1/24 EXL" plc_online="True">
  <alarms>
    <reading count="0">
      <alarm code="45.00" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.01" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.02" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.03" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.04" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.05" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.06" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.07" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.08" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.09" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.10" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.11" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.12" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.13" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.14" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="45.15" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.00" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.01" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.03" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.04" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.05" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.06" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.07" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.08" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.09" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.10" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.11" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.12" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.13" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.14" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="46.15" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="47.00" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="47.01" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="47.04" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
      <alarm code="47.05" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
```

```
      <alarm code="47.06" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
    </reading>
  </alarms>
  <counters>
    <counter code="d70" value="42101251" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d72" value="772" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d350" value="44588124" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d530" value="8414" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d531" value="1808" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d74" value="21511580" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d76" value="1694" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d352" value="88114012" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d565" value="8200" logged_at="2011-01-19 09:58:04 UTC"/>
    <counter code="d566" value="804" logged_at="2011-01-19 09:58:04 UTC"/>
  </counters>
  <statuses>
    <status code="20.0" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
    <status code="20.1" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
    <status code="20.2" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
    <status code="60.0" value="1" logged_at="2011-01-19 09:58:04 UTC"/>
    <status code="60.1" value="0" logged_at="2011-01-19 09:58:04 UTC"/>
  </statuses>
</request>
```

Ogni volta che il server riceve un aggiornamento provvede a loggare ogni dato nel relativo modello di interesse. Dei task in background ad intervalli regolari provvedono ad aggiornare dei contatori aggregato, come i tappi prodotti nell'ultima ora, ultimo giorno ed ultimo mese.

Così facendo nè il server in risposta a richieste dei moduli iMX27, nè le richieste dalle interfaccie di monitoraggio generano grandi quantità di calcoli, ma prelevano e visualizzano nella maniera più consona i dati richiesti.

Presentiamo la struttura dei modelli che compongono il database.

Machine

name: Stringa identificativa della pressa;

uid: Identificatore univocamente assegnato alla macchina;

last_request_at: Rappresenta l'istante temporale in cui è avvenuta l'ultima comunicazione da parte di questa pressa;

plc: Identificativo del modello del plc al quale è connesso il modulo iMX27;

code: Identificativo per mappare l'ordine delle macchine secondo la disposizione in azienda;

identification: Identificativo relativo al modulo iMX27, specificato nel file di settings.

Vengono rappresentate le istanze delle macchine da monitorare. Dal lato server, col termine "Macchina" si intendono indifferentemente Modulo iMX27, pressa o plc.

MachineModelStatus, MachineModelCounter e MachineModelAlarm

code: Codice identificativo del dato;

description: Descrizione dello stato;

param_type: Tipo del parametro;

category: Categoria di appartenenza.

Ogni allarme, stato o contatore sono rappresentati come entry nel database, ed attraverso una relazione molti-a-molti son legati alle istanze delle macchine. Ad ogni aggiornamento da parte di Moduli viene ricevuto un XML che viene valutato, vengono estratte le informazioni e vengono generate le entry nel database relative.

Queste sono le strutture principali che ci permettono di presentare la struttura del database.

Interfaccia di monitoraggio - WEB

Per permettere di monitorare l'applicazione in ogni situazione è stato creato un front-end WEB semplice ed intuitivo che permette di arrivare velocemente a visualizzare le informazioni più utili.

Dalla schermata di login [3.5], autenticandosi, è possibile vedere la mappa degli stabilimenti [3.6]. Si può poi vedere in un'unica schermata lo stato di tutte le macchine presenti, permettendo di vedere la velocità di produzione in tappi/ora. Un'icona sulla destra chiarisce anche lo stato della macchina, verde nessun problema, rosso spenta o in stand-by, arancione allarmi attivi [3.7].

Da questa visualizzazione è possibile conoscere in maniera più approfondita lo stato di ogni macchina. Cliccando sul segnale sulla destra verrà presentato un riepilogo delle informazioni sulla macchina come in [3.8].

Per ogni macchina è inoltre disponibile il grafico giornaliero [3.9], settimanale [3.10] e mensile [3.11].

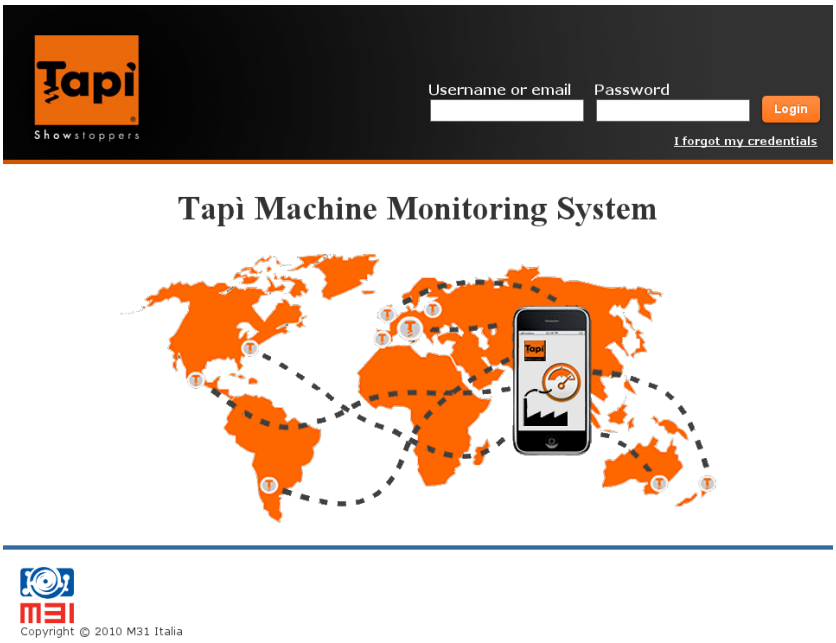


Figura 3.5: Tapì: Schermata di Login

Interfaccia di monitoraggio - iPhone/iPad

Per consentire di poter monitorare in qualsiasi occasione l'andamento della produzione è stata sviluppata anche un'applicazione per iPhone, poi portata anche su iPad. Questa applicazione sfrutta le api esposte dall'applicazione *Ruby on Rails* attraverso la sintassi *REST*.

Parallelamente all'applicazione web sono presenti le schermate di login [3.12(a)], di schermata generale delle presse [3.12(b)] e di specifiche per ogni macchina [3.12(c)].

3.2.3 Peculiarità del sistema

Il sistema realizzato risulta molto stabile, molta attenzione è stata posta al *fault-tolerance*. Dopo la realizzazione del prototipo abbiamo fatto l'installazione pilota nell'impianto produttivo a Massanzago.

Il *deploy* del sistema ha comportato una lunga fase di monitoraggio stretto. Nel primo periodo sono sorti molti problemi non prevedibili a priori per i quali spesso e volentieri è stata necessario la trasferta per poter correggere bug e malfunzionamenti.

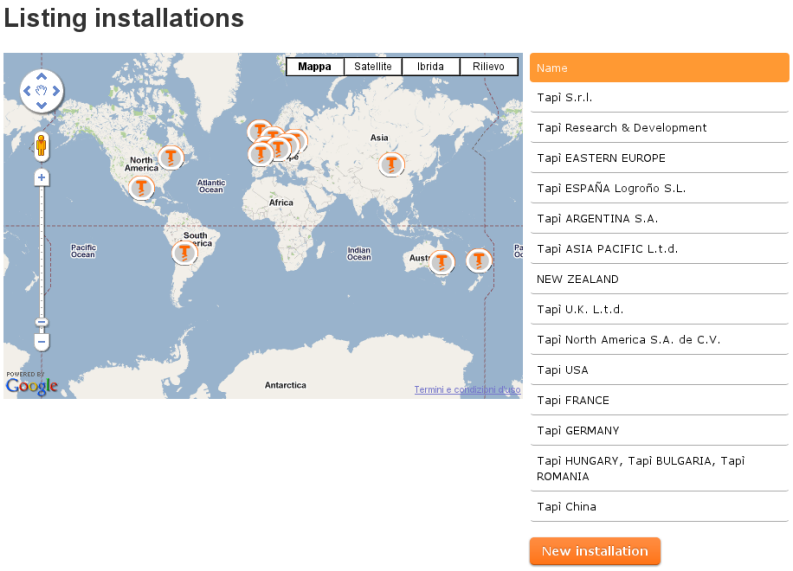


Figura 3.6: Tapì: Home Page

In primo luogo si è dovuto procedere a raffinamenti successivi per capire correttamente la comunicazione seriale con i PLC. Il formato di rappresentazione dei dati poi non è sempre correttamente spiegato nella documentazione delle presse, le configurazioni delle aree di memoria sono quindi dovute essere corrette a mano.

Inoltre si è scoperto che spesso e volentieri le macchine venivano tenute spente per un certo periodo di tempo. Questo causava un reset dell'ora dei moduli iMX27, che si riportavano al 01/01/2010. Questo generava dei log con date temporali sbagliate che il server non registrava considerandole errate. Si è quindi provveduto ad installare un server *NTP*[3].

Abbiamo istruito i moduli iMX27 in modo che sincronizzino la propria ora ad ogni avvio col server, per poter mantenere una consistenza tra i vari agenti del sistema.

Un server ntp, però, richiede un certo tempo - alle volte svariate decine di minuti - per essere pronto a rispondere alle richieste che gli arrivano. A valle di un'interruzione di corrente di tutto il sistema, quindi, i moduli risultano ancora non sincronizzati, in quanto la richiesta di aggiornamento fallisce.

Abbiamo quindi preferito, con evidente eccesso di zelo, effettuare una

[3]NTP sta per **N**etwork **T**ime **P**rotocol. Un server ntp aggiorna in continuazione la propria ora di sistema con altri server ritenuti affidabili, e risponde alle richieste dei client che vogliono sincronizzare la propria ora

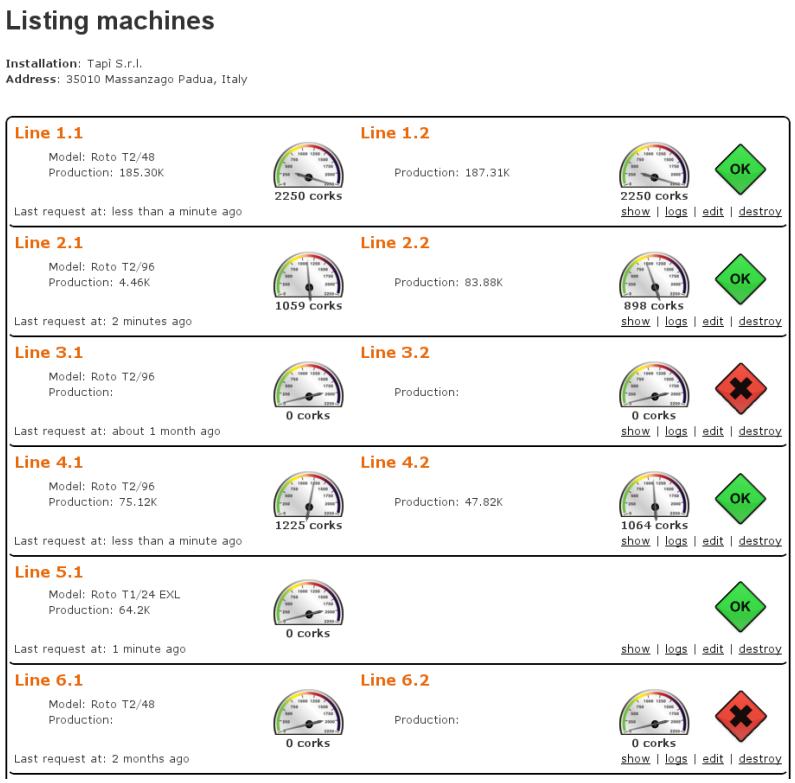


Figura 3.7: Tapì: Lista delle macchine installate

richiesta di sincronizzazione ogni cinque minuti, così da diminuire il più possibile ogni desincronizzazione.

Un altro problema cruciale consisteva nel fatto che molto spesso, dopo un riavvio, i moduli iMX27 non avviavano il sistema operativo, quindi non veniva fatta la prima scrittura sul watchdog e rimanevano in una situazione di stallo.

Per capire quale fosse il problema alla base di queste situazioni sono state valutate tutte le possibilità e sono state seguite decine di strade.

Si è poi scoperto che il firmware di boot dei moduli, denomianto uBoot, permette di effettuare delle operazioni in fase di preboot. Per accedere al menu di configurazione basta premere un qualsiasi tasto nei primi due-tre secondi di accensione del modulo.

Come spesso accade in un sistema *nix e nei moduli embedded soprattutto, l'interfaccia primaria di connessione al sistema è rappresentato dalla porta seriale.

Abbiamo quindi verificato l'ipotesi per la quale il PLC, connessi appunto

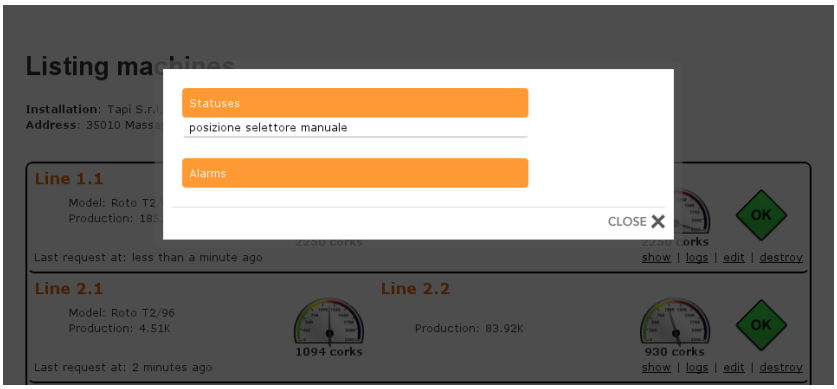


Figura 3.8: Tapì: Descrizione dello stato per ogni macchina

via seriale ai moduli iMX27, scriverssero dei dati "sporchi" verso il modulo, inibendo quindi il processo di boot.

Configurando uBoot ad interrompere il boot solo con una configurazione particolare di tasti abbiamo quindi risolto il problema.

A valle di queste correzioni in corso d'opera abbiamo dato vita ad un prodotto stabile e molto scalabile per la monitorizzazione remota degli impianti.

Questo strumento permette alle persone preposte alla sorveglianza del processo produttivo di avere sempre, in maniera immediata, una visione completa dell'andamento del lavoro.

Inoltre è possibile utilizziare i dati raccolti costantemente per poter verificare le attese di produzione e poter meglio pianificare l'effettiva produttività degli impianti. Con questi dati è possibile fare delle stime sulla produzione futura molto più corrette, potendo contare su andamenti effettivi e non ipotetici.

3.2.4 Sviluppi futuri

Il risultato di questo progetto, oltre ad un'infrastruttura hardware e software completa, stabile e robusta, ci ha permesso di sviluppare un *know-how* molto approfondito nel monitoraggio dei processi industriali di produzione.

Una delle possibili evoluzioni che sono in fase di valutazione è l'integrazione del progetto Tapì con il framework di domotica **SAN**, sviluppato da M31.

SAN è un progetto volto alla gestione della domotica, permettendo di integrare tutte le componenti elettronice o elettriche in un unico sistema. Il punto forte del sistema consiste nella completa apertura e versatilità del

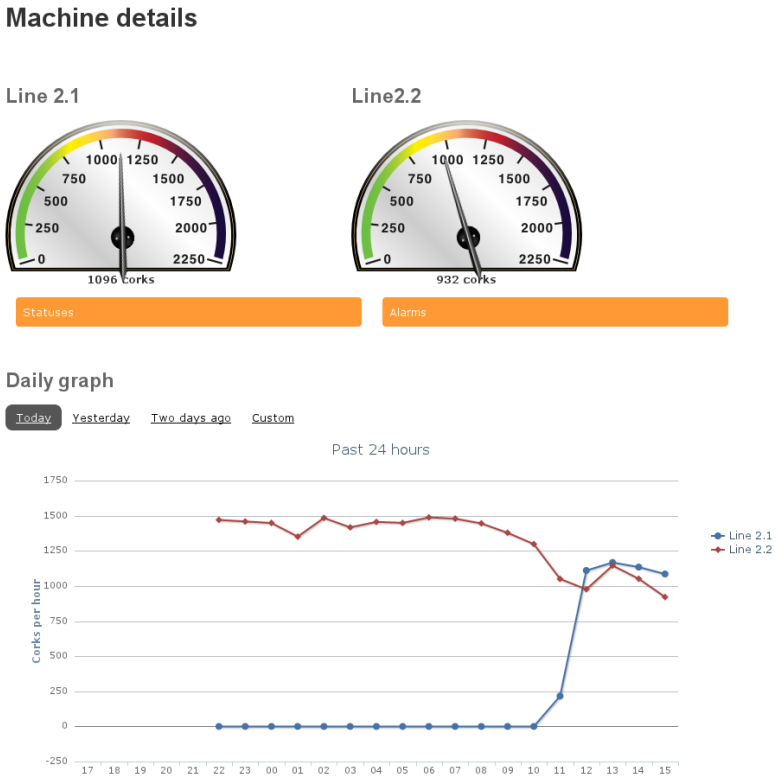


Figura 3.9: Tapì: Tachimetro e grafico giornaliero

sistema di basso livello, che permette ad ogni periferica di autodescriversi all'interno del sistema e ricevere ed inviare comandi e notifiche.

È quindi possibile intefacciare direttamente i moduli iMX27 con questo sistema di gestione permettendo eventualmente di descrivere delle regole complesse basate - ad esempio - sul verificarsi di determinati allarmi in una pressa.

Questo sistema permetterebbe di rendere ancora più generica l'infrastruttura sviluppata e permetterebbe un'elaborazione molto più articolata dei dati storici, seguendo la strategia nota come *Web of Things*[4].

[4]Con il termine Web of Things si intende la visione ispirata al concetto di "Internet of Things" che consiste nell'esporre tutti i dispositivi di utilizzo comune attraverso il WEB

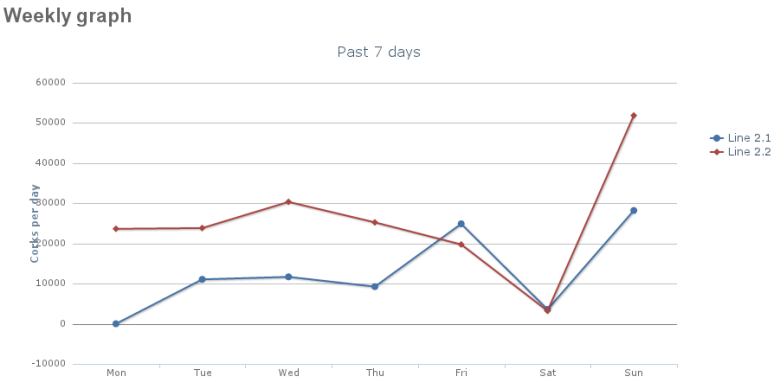


Figura 3.10: Tapì: Grafico settimanale

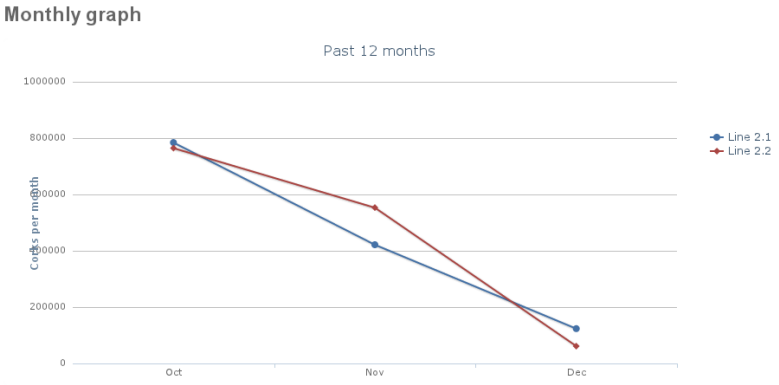
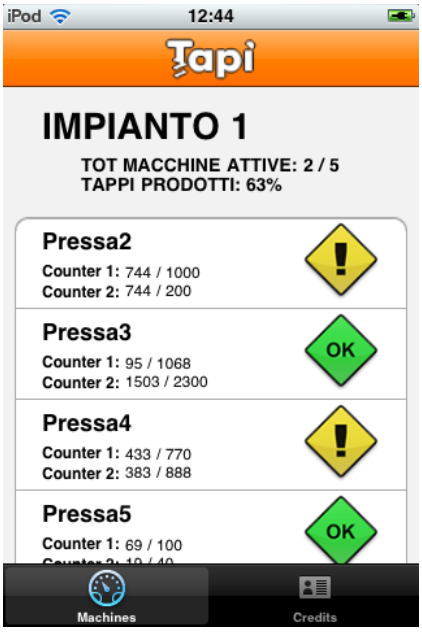


Figura 3.11: Tapì: Grafico mensile

(a) Tapì for iPhone: Login



(b) Tapì for iPhone: Overview macchine

(c) Tapì for iPhone: Stato di una macchina

Capitolo 4

Conclusioni

A valle delle conclusioni sui singoli progetti, è possibile tracciare una valutazione globale sulla situazione attuale dei sistemi **SCADA**.

È indubbio che in questo settore, come tutti i settori legati in qualche modo alla tecnologia, l'ultimo decennio abbia portato a grandi rivoluzioni in tutti i campi collegati.

Si pensi all'avvento massivo e completamente ramificato della rete Internet ad ampia banda e con costi ridotti. Il mondo delle comunicazioni è cambiato radicalmente. Utilizzando la struttura esistente è possibile connettere due punti distantissimi al mondo con costi estremamente contenuti.

La proliferazione di linguaggi di programmazione interpretati ad alto livello ha permesso la nascita dell'**Agile Development**[1] [6], riducendo i costi ma soprattutto i tempi di sviluppo del software.

L'evoluzione dell'hardware, intesa come diminuzione dei costi e aumento delle prestazioni, permette ora di equipaggiare anche apparati di scarsa rilevanza con schede elettroniche con le medesime potenzialità di normali Personal Computer.

Risulta quindi evidente come tutte le aree tecnologiche, in questo periodo, siano in fase di rimodernazione. Nella pratica comune, le innovazioni tecnologiche si riflettono sempre con qualche anno di ritardo su impianti esistenti.

Le politiche comuni consistono nel non modificare un impianto funzionante in produzione a meno che non sia strettamente necessario o che se ne possa trarre un grosso vantaggio che possa giustificarne il costo. Anche in questo caso, però, le innovazioni non possono stravolgere l'infrastruttura esistente.

[1] Con l'accezione Agile Development si delineano un gruppo di strumenti di sviluppo software basati sullo sviluppo iterativo ed incrementale. Il processo consiste nell'utilizzare linguaggie e strumenti che possano portare a delle modifiche immediate e visibili, valutando in maniera continuativa l'evoluzione del software.

Negli impianti nuovi, invece, si parte dallo stato dell'arte e della tecnologia in quell'istante. È quindi naturale trovare grosse innovazioni tecnologiche in impianti completamente nuovi.

Nel campo degli **SCADA** le prospettive sono differenti. Concettualmente uno scada costituisce uno strato sopra l'implementazione esistente.

Nel caso di Tapì, ad esempio, non è stata intaccata la struttura preesistente ma si è costriuto un sistema *sopra*. Questo rappresenta una differenza enorme rispetto ad altri ambiti.

Sistemi **SCADA** possono essere ancora meno invasivi di quelli proposti in questa tesi, possono addirittura non entrare mai in contatto fisico con il processo che si sta analizzando.

Si pensi per esempio a dei sistemi di controllo del traffico in ingresso/uscita collocati presso i porti. Potrebbe essere semplicemente effettuato con l'installazione di telecamere e degli algoritmi di visione per valutare il flusso delle navi.

Indubbiamente questo periodo di crisi economica ha portato una forte consapevolezza sulla necessità di razionalizzare i costi, e uno dei metodi più efficaci è la prevenzione dei guasti. Parallelamente si cerca di massimizzare la produttività di ogni singolo elemento, il che richiede un'analisi approfondita dello storico produzione per poter individuare punti di forza o di debolezza, trovando così i metodi giusti per ottimizzare la produzione.

Questo contesto determina un terreno estremamente fertile per i sistemi SCADA, che stanno proliferando e riceveranno sempre più attenzione nei prossimi anni.

Una volta ultimati i progetti è stato svolto un profondo lavoro di analisi delle problematiche e delle *feature* del lavoro svolto. Questo ha permesso la raccolta delle peculiarità di entrambi i progetti per realizzare un framework *general purpose* per applicativi **SCADA**.

Questo progetto, che va sotto il nome **Innesca**, fornisce una base avanzata per la realizzazione del sistema centrale di sistemi di monitoraggio. Basato sulla versione 3 di *Ruby on Rails*, è stato strutturato di modo da essere un ottimo punto di partenza che permetta una forte verticalizzazione sulle richieste del cliente in maniera veloce e mirata.

Indipendentemente da come saranno strutturati i dati di una qualsiasi applicazione futura per la quale verrà utilizzato, sono già presenti le schermate di riepilogo e di monitoraggio in real-time, utilizzando tecniche all'avanguardia come i *Web-Socket*.

Questo strumento permette di essere molto competitivi, diminuendo decisivamente il costo della realizzazione di un sistema SCADA dalla progettazione e realizzazione del centro di accumulazione dati, potendo concen-

trarsi solamente sul sistema periferico, focalizzandosi sull'integrazione con i dispositivi che il cliente vorrà monitorare.

Capitolo 5

Ringraziamenti

Doverosi, ovviamente.

Questa sezione è sempre la pi piacevole da scrivere, permette di scorrere con la mente tutti gli anni (tanti) passato nel mondo unversitario e non, riportando alla mente splendidi ricordi e importanti esperienze.

Scontato e banale, ma ovviamente le prime persone che voglio nominare sono i membri della mia famiglia. Sappiatelo, ho una famiglia splendida e se son arrivato qui gran poco merito è mio. Grazie **papà Mauri** per avermi insegnato a stare al mondo con l'esempio. Mi sei sempre stato vicino nei momenti difficili, mi hai lasciato sbagliare, me lo hai fatto capire e poi mi hai insegnato come fare. Grazie alla **mamma Stefi** per essere anzitutto un'amica con cui ridere e scherzare, ma soprattutto per avermi inculcato, senza saperlo, il modo di pensare adatto ad un ingegnere. Grazie alle mie sorelle, **Elena** e **Laura**, perchè sono le persone che ho sempre avuto più vicino, che mi hanno sopportato, coccolato, sorretto, aiutato, di più in assoluto. Sono veramente fortunato, ho una famiglia piena di amore, vi voglio bene.

Ma non finisce qui, perchè la mia famiglia è un bel po' allargata. Ho altri due persone che mi son state molto vicine soprattutto nel periodo universitario, sorreggendomi, consolandomi ed ascoltando i miei sfoghi... Le mie splendide nonne, ovviamente. **Nonna Vittoria** grazie per tutti i caffè, tutti i pomeriggi, tutte le chiacchierate che abbiamo fatto, per tutte le parole giuste che hai sempre avuto per me, incitandomi a proseguire nei momenti difficili e facendomi sentire speciale nei momenti di gioia. Grazie **Nonna Silvana**, grazie per i caffè a tutte le ore (si, nella mia famiglia si bevono un sacco di caffè!), grazie perchè so che ci sei e mi fai sentire importante ogni volta che ci vediamo.

Un grazie a tutti gli **zii**, **zie** e **cuginanza** completa. L'unione che ci lega è una cosa preziosa, la gioia ongi volta che abbiamo occasione di ritrovarci assieme è una cosa alla quale non rinuncierei mai al mondo. Un ringraziamento

particolare a mia cugina **Marta**, la prima laureata della famiglia. Grazie per i preziosi consigli che mi hai dato in questi anni.

Arriva poi la seconda famiglia, quella con cui ho condiviso sudore, pasti frugali, imprecazioni, soddifazioni e delusioni giorno per giorno nella mia carriera universitaria. Primo tra tutti, grazie **Salvatore** Orso **Brundo**. Grazie dal profondo del cuore per aver condiviso con me i momenti peggiori di questa avventura, grazie per aver ascoltato i miei (tanti) sfoghi, grazie per essere stato un ottimo coinquilino e grazie per essere una delle persone che stimo di più. Assieme ne abbiam passate di pessime, veramente, ma abbiamo sempre trovato il lato positivo di tutte le esperienze e assieme siamo cresciuto personalmente e professionalmente.

Grazie poi al buon **Trevi**, compagno instancabile di splendide serate, grazie per avermi accolto in famiglia. Anche se sei scappato un po' lontano resti un fratello. Un grazie particolare anche a **ghedamat**. Abbiamo avuto alti e bassi, ma ora siamo qui, e resti una delle persone più importanti. Consulente tecnico preziosissimo, ma soprattutto amico-fratello-fratello-amico, grazie per tutto, veramente. Grazie per la compagnia nei momenti di scazzo, grazie per i mezzi, per i pomeriggi interminabili a parlare di tutto e niente. E tra gli amici-fratelli-fratelli-amici, grazie a **Enrico Longo** e **Francesco Puglierin**, (più a longo che al puglie...) per aver condiviso degli splendidi viaggi e per restare comunque dei punti fermi.

Come non ricordare lo **zoppo** poi... Grazie per avermi accolto in casa, per le birrette, i tranci di pizza e i gossip che nemmeno le casalinghe dal parrucchiere. Grazie a **Marco Papacizza** per gli stessi motivi, forse meno birre e più gossip, ma il senso è quello.

Un grazie a tutti i coinquilini che ho avuto in questi anni, grazie per avermi fatto compagnia nelle serate tristi pre-esame. Grazie **Gabry** truzzo **Meneghetti**, per essere un amico di lunga data. Grazie a **Prezze**, al **Porde**, alla **Federicuccia**, alla **Vero**, **Davide**, alla **Jè**, a **Kappa**.

Grazie a tutti gli amici del dei, **Mauro Tubiana**, anche se scrivi codice con l'acetta discutere dei massimi sistemi algebrici con te è sempre un piacere. Un grazie a tutto il gruppo, **Mauro Donadeo**, **Samuele Stefanoni**, **Riccardo Bonetto** e tutti quelli che stupidamente mi sarò dimenticato.

Impossibile non ringraziare la mia terza famiglia ora. Un grazie profondo, pieno di gratitudine, al gruppo **M31**. Un anno fa sono stato accolto e qui ho trovato un ottimo posto dove imparare e dove mettere a frutto le mie capacità, ma soprattutto ho trovato un'accoglienza calorosa. Un grazie particolare va a **Zond**.

Mio caro **Fabio**, grazie per avermi accompagnato e sostenuto in questo anno. Grazie per sempre avuto la parola giusta per farmi sentire importante, grazie grazie e grazie ancora davvero. **Rudi**, vabbè, lo sai, e per tutto que-

sto grazie. **Stefano**, mio caro *pair-programmer* preferito, grazie per avermi insegnato una marea di cose senza presunzione. Un grazie a tutto il gruppo, **Luca**, **Alberto R.**, **Alberto S.**, **Giorgio**, **Mattia**, **Arrigo**, **Lorenzo** e **Francesco**. **TullioTech**, grazie per essere una carissima amica, per tutti gli incoraggiamenti e le consolazioni, grazie. **Sicolo**, grazie per avemi battezzato, quasi un'anno fa, con una lavata di birra e avermi fatto sentire a casa. Per evitare di scrivere più di ringraziamenti che di tesi evito di nominare tutti i più di settanta elementi del gruppo, ma grazie ad ognuno di voi. Grazie infinite perchè ho avuto la possibilità di conoscervi tutti e di imparare da **ciascuno** qualcosa.

Tra gli ultimi, ma non certo per importanza, un ringraziamento veramente sentito al buon **Pier**. Coinquilino quasi per caso, ho trovato un prezioso amico. Grazie per la pazienza che hai portato nei periodi più stressanti e soprattutto grazie per le interminabili chiacchierate serali. Grazie per infondermi ogni giorno fiducia in mé stesso.

Un ultimo grazie alle donne che hanno avuto la pazienza di accompagnare alcuni frangenti della mia vita. I miei ricordi migliori sono quasi tutti legati a voi. Grazie per avermi fatto sentire felice, a dispetto di quello che mi succedeva attorno.

Come potete vedere, alla fine della fiera, è più merito d'altri che mio se sono arrivato fino a qui. "Sono ciò che sono per merito di ciò che siamo tutti", o no?

E ora fatemi fare un po' di demagogia spicciola, o buonismo, chiamatelo come vi pare. Per i coraggiosi che sono arrivati fin qui in fondo, vi prego, fate un pensiero serio sul diventare **donatori di sangue**. Non vi costa nulla, vi fa sentire meglio e aiutate veramente chi ne ha bisogno. Stop, fine parentesi buonista.

E ora, col vostro permesso, vado a stampare...

Appendice A

Appendice - I Watchdog

Un Watchdog è un sistema temporizzato di sicurezza dei dispositivi dotati di CPU. Lo scopo consiste nel rilevare indirettamente un blocco del sistema e provvedere ad un riavvio del dispositivo.

Si definisce rilevamento indiretto perchè è compito del software *pingare* il watchdog per confermare l'attività del software.

Più precisamente quello che avviene è questo. All'avvio del sistema viene abilitato il watchdog, che è connesso ad un timer esterno alla CPU. Viene impostato il tempo limite di aggiornamento all'interno del quale il software deve aggiornare il timer. Se scade il *timeout* senza che venga aggiorato il watchdog, il sistema subisce un riavvio hardware.

Nell'implementazione per il progetto *Tapì* avevamo a disposizione dei moduli iMX27 con OpenEmbedded come sistema operativo.

Come in tutti i sistemi basati sul sistema *GNU/Linux*, l'*entry-point* del watchdog a livello software consiste nella periferica a caratteri `/dev/watchdog`.

Nella prima implementazione lo schema di funzionamento rifletteva il diagramma in A.1.

Questo funzionava correttamente nei test effettuati in azienda, ma dai primi *deploy* in produzione si è notato un inusuale periodo di downtime dei moduli.

Attraverso un processo di analisi per sintesi, sono state valutate tutte le possibilità di blocco del modulo, grazie anche alla preziosa consulenza degli ingegneri di *Si14*.

Anzitutto sono stati aggiunti delle verifiche al Watchdog software. Si è notato che alle volte capitava che il controller *ethernet* non fosse correttamente inizializzato al boot, il che portava il sistema a comportarsi correttamente, senza però riuscire ad inviare i propri dati al boot. Abbiamo quindi aggiunto un controllo sulla raggiungibilità del server, non aggiornando il watchdog hardware in caso di mancata connessione.

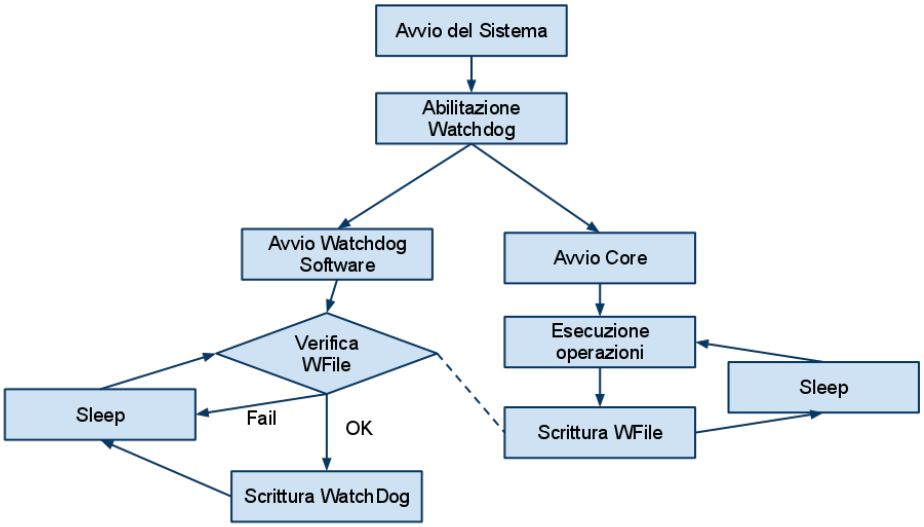


Figura A.1: Watchdog: primo schema

Il watchdog così modificato risulta come nel listato A.1, dove si possono notare i controlli di connettività.

Listato A.1: TapiDog: Il Watchdog Software per Tapì

```
#!/usr/bin/env python2

import os
import time
import datetime
import settings
import vendor.commands as commands

INITIALIZE = 0
RUNNING = 1
ALIVE_NOT_RESPONDING = 2
NOT_ALIVE = 3

class TapiWatchdog:
    """Watchdog for Tapi Core program."""
    status = INITIALIZE
    counter = 0

    def run(self):
```

```
        while True:
            last_modification = False
            while not last_modification:
                try:
                    last_modification = os.path.getmtime(settings.↩
                        WATCHDOG_FILE)
                    # Catching OSError if file is not present, core not ↩
                        initialized.
                except OSError:
                    self.deep_check()
                    time.sleep(2)
                    continue

            # Check for the delta time, 150% of CHECK_INTERVAL for ↩
                tollerance
            if time.time() - last_modification > (settings.CHECK_INTERVAL ↩
                * 1.5):
                self.deep_check()
            else:
                self.hw_watchdog()
                self.status = RUNNING
                pass
            time.sleep(settings.WATCHDOG_INTERVAL)

    def deep_check(self):
        """Makes a deep check if mtime check fails.
        Calls ps aux to check for core.py. If not present respawn the ↩
            process """
        ps = commands.getoutput('ps aux | /bin/grep core.py | /bin/grep -v↩
             grep')
        if ps:
            self.status = ALIVE_NOT_RESPONDING
            self.counter += 1
            # If under threshold, update the hw watchdog
            if self.counter <= settings.WATCHDOG_THRESHOLD:
                self.hw_watchdog()
        else:
            self.status = NOT_ALIVE
            self.respawn()

    def respawn(self):
        """Runs main application."""
        os.popen('%s &' % settings.CORE_PATH)
        self.counter = 0

    def ping_test(self):
        ping_server = commands.getoutput('ping -W1 -c1 %s' % settings.↩
            TAPI_HOST)
        return 'ttl=' in ping_server

    def hw_watchdog(self):
        """Update the hardware watchdog."""
        ### Check connection with server
        if self.ping_test():
            hw_wd = open('/dev/watchdog', 'w')
            hw_wd.write('1')
            hw_wd.close()

if __name__ == '__main__':
    a = TapiWatchdog()
    a.run()
```

Nonostante questi accorgimenti capitava ancora spesso che i moduli fossero *down*. Analizzando approfonditamente il processo di avvio, ci siamo accorti che il *boot-loader* dei moduli, l'*UBoot*, come tutti i boot-loader, al boot permetteva di fermare il boot e modificare l'*environment* di avvio.

Bastava premere un tasto per interrompere quindi il processo di boot. Ma questi moduli sono per loro natura sprovvisti di periferiche di input classiche, tutta la gestione normalmente avviene attraverso la porta seriale che noi utilizziamo per la comunicazione col PLC.

Abbiamo quindi ipotizzato e verificato riavvii nel quale il plc scriveva qualcosa sulla seriale, a causa di richieste pendenti, interrompendo così il boot. Abbiamo quindi modificato *UBoot* per entrare in modalità amministrazione solo con una specifica combinazione di tasti (`Ctrl+C`).

Questa scoperta ci ha ricorato che una dei terminali virtuali del sistema veniva inizializzata per la comunicazione seriale. Per evitare quindi qualsiasi inconveniente abbiamo rimosso da `/etc/inittab` la configurazione per aprire la shell sulla seriale di sistema. Avere questa shell però può essere molto utile in fase di analisi successiva. Abbiamo quindi configurato il sistema per aprire una shell di debug attraverso un connettore seriale/USB.

Così facendo è possibile configurare il modulo mantenendolo connesso al PLC. Nel listato A.2 si possono vedere le modifiche effettuate, si nota che nei moduli embedded le seriali vengono enumerate con `/dev/ttymxcX` a differenza di un sistema classico, dove sono `/dev/ttySX`.

Listato A.2: /etc/inittab modificato per i moduli iMX27

```
2 #tapi
S:2345:respawn:/sbin/agetty -L 115200 ttyUSB0 vt100

#debug
#S:2345:respawn:/sbin/agetty -L 115200 ttymxc0 vt100
```

Grazie a queste modifiche abbiamo drasticamente ridotto il numero di eventi di stallo dei sistemi. Nonostante questo importante miglioramento, il numero di eventi di stallo non e' stato portato a zero, ma si verificano comunque sporadici casi di blocco. Ad un'analisi ancora più approfondita abbiamo notato che la tensione di alimentazione dei moduli subiva delle forti oscillazioni, probabilmente dovute alle inteferenze presenti nel quadro elettrico A.2.

Gli alimentatori utilizzati avevano un cavo di circa un metro, arrotolato all'interno dell'alloggiamento. Abbiamo quindi provveduto a tagliare il cavo in eccesso di modo da evitare di fare da antenna, risolvendo così anche questo problema.

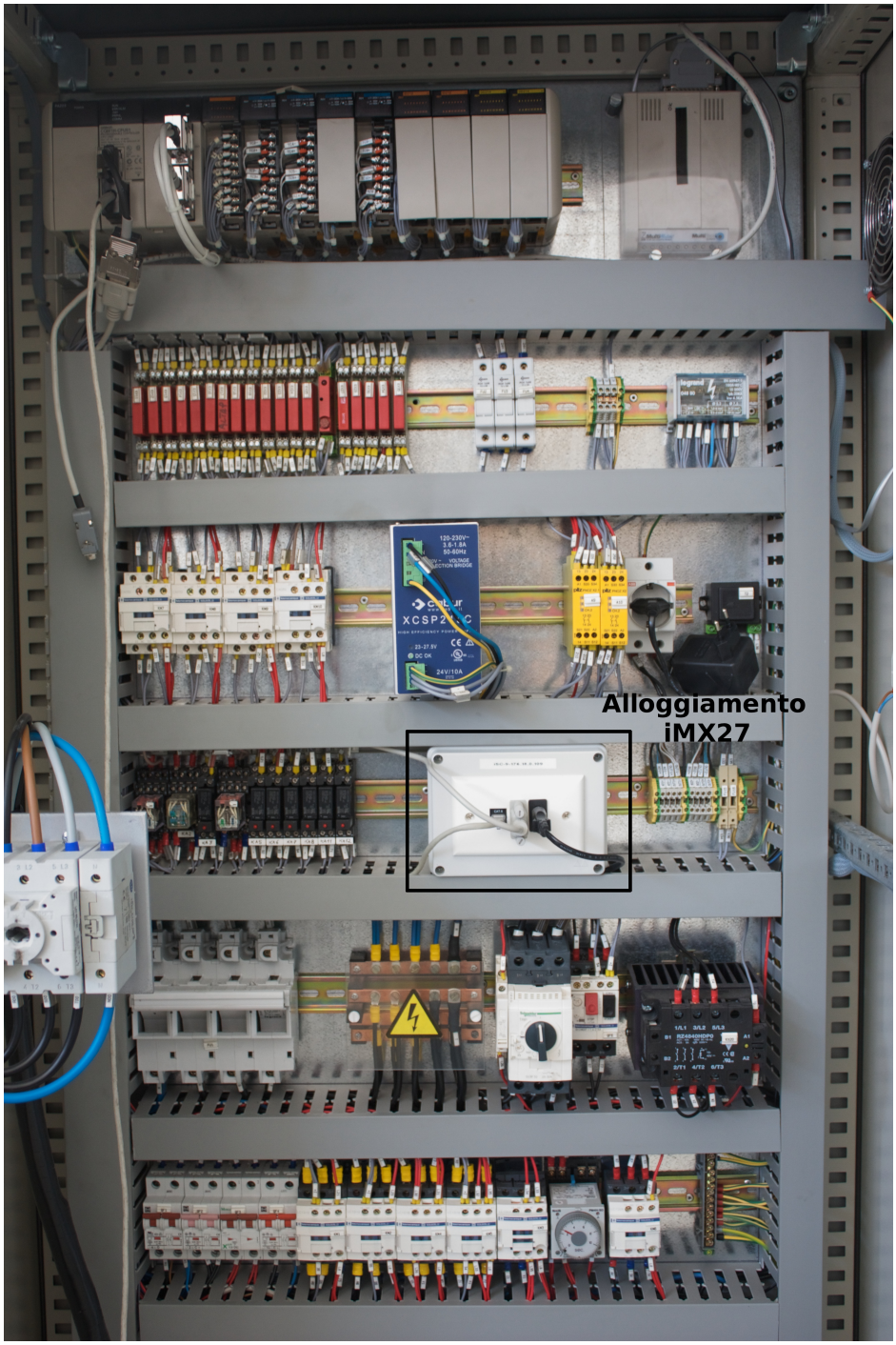


Figura A.2: Un quadro elettrico con il componente iMX27

Acronyms

SCADA Supervisory Control And Data Aquisition

REST Representational state transfer

LAN Local Area Network

MAN Metropolitan Area Network

TCP Transimssion Control Protocol

HTTP Hypertext Transfer Protocol

XML Extensible Markup Language

API Application Programming Interface

PLC Programmable Logic Controller

Bibliografia

[1] Stefano Bimbo; Enrico Colaicovo, *Sistemi SCADA; Supervisory control and data acquisition.* Apogeo, 2006.

[2] P.Cabena; P.Hadjinian; R.Stadler; J.Verhees; A.Zanasi, *Discovering data mining from concept to implementation.* Prentice Hall PTR, 1997.

[3] Jeff Barr, *Host your Web Site in the Cloud.* SitePoint, 2010.

[4] Abe Fetting, *Twisted; Network Programming Essentials.* O'Reilly, 2005.

[5] F. Dawson; D. Stenerson, *RFC2445: Internet Calendaring and Scheduling Core Object Specification (iCalendar).* IETF, November 1998.

[6] Sam Ruby; Dave Thomas; David Heinemeier Hansson, *Agile Web Development with Rails.* Pragmatic Bookshelf, 2011.

Elenco delle figure

Elenco dei listati