ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

DISI
Dipartimento di Informatica - Scienza e Ingegneria

TESI DI LAUREA

in

Sistemi In Tempo Reale M

PROGETTO E REALIZZAZIONE DI UN MODELLO DI HMI DINAMICAMENTE RICONFIGURABILE PER MACCHINE AUTOMATICHE MULTIDOSAGGIO

CANDIDATO
Andrea Turrini

RELATORE
Chiar.mo Prof. Ing. **Eugenio Faldella**

CORRELATORE
Ing. **Enrico Pasini**

Anno Accademico 2012/2013

Sessione III

"La logica ti porterà da A a B.

L'immaginazione ti porterà ovunque."

Albert Einstein

Indice

Introduzione

Il lavoro che viene presentato è stato sviluppato nel tirocinio per tesi iniziato il 14 ottobre 2013 presso IMA Active, divisione del Gruppo IMA Pharma, che costituisce il settore farmaceutico di IMA SpA. IMA (Industria Macchine Automatiche) venne costituita a Bologna nel 1961 ed è oggi leader mondiale nella progettazione e produzione di macchine automatiche per il processo ed il confezionamento di prodotti farmaceutici, cosmetici, alimentari, tè e caffè. IMA Pharma è leader mondiale nella progettazione e produzione di macchine automatiche per il processo e il confezionamento di prodotti farmaceutici grazie ad un alto profilo tecnologico e alla capacità di offrire soluzioni personalizzate in grado di soddisfare le richieste più sofisticate del mercato. IMA Pharma include tre divisioni altamente specializzate: IMA Active (Solid Dose Solutions), IMA Life (Aseptic Processing & Freeze Drying Solutions), IMA Safe (Packaging Solutions).

La macchina automatica oggetto dell'attività di tesi è l'opercolatrice Adapta: essa è adibita al riempimento di capsule medicinali con differenti varietà di prodotto, selezionato dal cliente tra polvere, compresse e liquidi. La peculiarità di questa macchina sta nell'alta flessibilità operativa di cui è dotata: la possibilità di sostituire dinamicamente le stazioni di lavoro in cui è suddivisa la rende unica nel segmento di mercato in cui è collocata. Alla semplicità di riconfigurarsi dinamicamente dal punto di vista meccanico non corrisponde però la stessa facilità dal punto di vista software: infatti mentre risulta semplice adattare il processo di lavoro a differenti prodotti sostituendo semplicemente i gruppi di dosaggio meccanici presenti nelle stazioni di lavoro, la stessa operazione di variazione della configurazione operativa dal punto di vista del software d'interfaccia si rivela particolarmente complicata e non esente da malfunzionamenti. Il problema è legato al procedimento di generazione e selezione del tipo di interfaccia uomo-macchina corrispondente alla configurazione meccanica montata. Lato software è infatti necessario fornire al cliente a priori a tutte le possibili configurazioni di lavoro dell'applicativo di interfaccia legate ad una specifica macchina: sarà poi l'operatore a provvedere, prima dell'avvio della produzione, a sceglierne una per via manuale.

In questo senso l'obiettivo di questo lavoro di tesi si focalizza sull'analisi, la progettazione e la realizzazione di un modello di interfaccia uomo-macchina che rifletta la semplicità e l'affidabilità del cambio di configurazione del lato meccanico anche dal punto di vista del software d'interfaccia.

Da principio, nel primo capitolo, si è analizzata in maniera approfondita la macchina automatica Adapta ed il suo comportamento funzionale, soffermandosi in particolare sullo studio della sua parte software e del procedimento di variazione della configurazione operativa.

Nel secondo capitolo si è invece esaminato lo stato dell'arte relativo ai sistemi d'interfaccia uomo-macchina, ai corrispondenti modelli architetturali e alle possibili strategie di sviluppo delle applicazioni software, in modo tale da poter progettare una specifica infrastruttura di supporto alla gestione automatica della configurazione. La progettazione di questa nuova architettura trova quindi fondamento nei più recenti e solidi principi di design software che meglio si adattano ai requisiti individuati per la macchina automatica in esame.

Nel terzo capitolo si è invece passati alla fase di formalizzazione dell'infrastruttura progettata sulla base dei requisiti esposti: partendo dall'architettura proposta per il modello in esame, lo si è adattato alle tecnologie in uso nel sistema attuale, realizzandone un'implementazione sia dal punto di vista del software d'interfaccia uomo-macchina, sia dal punto di vista della gestione dei rapporti informativi.

Da ultimo, nel capitolo conclusivo, vengono tratte le dovute considerazioni sul lavoro svolto, esaminando nel dettaglio i risultati conseguiti alla luce degli obiettivi iniziali.

Come appendice al testo si può trovare invece la descrizione dello standard Opc, utilizzato in ambito industriale per consentire lo scambio di informazioni in uno stesso sistema tra componenti eterogenei.

Capitolo 1

La macchina automatica Adapta

L'Adapta (figura 1.1) appartiene alla classe delle opercolatrici, cioè macchine automatiche adibite al riempimento di capsule medicinali. Essa è frutto di un progetto estremamente innovativo, in grado di dosare entro capsule di gelatina dura fino a tre diversi prodotti su un totale di cinque disponibili. Queste sostanze differiscono per composizione e granulometrie e sono qui di seguito elencate per granulometrie crescenti:

• polvere

• cronoidi (definiti anche pellet)

• microcompresse (definite anche microtablet)

• compresse (definite anche tablet)

• liquidi

Figura 1.1: La macchina Adapta

La peculiarità della macchina risiede nel suo design estremamente flessibile: le unità di dosaggio sono state progettate per essere intercambiabili, offrendo così ai clienti la possibilità di effettuare un cambiamento tra diverse configurazioni di macchina e combinazioni di riempimento di tipo plug and play, come mostrato in figura 1.2.

Attualmente la macchina viene prodotta in due varianti, a seconda della produzione oraria desiderata:

• Adapta 100: versione da 100.000 capsule / ora

• Adapta 200: versione da 200.000 capsule / ora

La zona di lavoro della macchina è inoltre completamente accessibile, rendendo le operazioni di ispezione e pulizia molto semplici.

Figura 1.2: Gruppi di dosaggio intercambiabili

La macchina permette il controllo totale della qualità della produzione durante l'intero processo produttivo, a seconda delle necessità del cliente. Queste verifiche possono avvenire attraverso un'analisi diretta del peso del prodotto inserito nelle capsule, oppure attraverso alcuni controlli indiretti che ne prevedono solamente una sua misura mediante opportuni sensori.

Il controllo diretto del peso del prodotto inserito nelle capsule viene proposto ai clienti in tre modalità:

• Controllo statistico del peso lordo: questo tipo di esame viene effettuato campionando periodicamente una capsula per verificarne il peso.

• Controllo 100% del peso lordo: definito anche TPC (Total Production Control), questo tipo di esame viene effettuato verificando il peso di tutte le capsule prodotte.

• Controllo 100% del peso netto: definito anche TFC (Total Fill Control), questo tipo di esame viene svolto effettuando la pesatura di ogni singola capsula prima e dopo che questa sia stata riempita. Tramite una semplice operazione di differenza è quindi possibile identificare la quantità di prodotto inserita al suo interno.

I controlli indiretti, se presenti, essendo relativi al singolo dosaggio di prodotto inserito nelle capsule, avvengono all'interno della stazione di dosaggio stessa. Le modalità con cui questi controlli vengono effettuati dipendono dalla tipologia di prodotto trattato dalla stazione e verranno descritti nel seguito. In presenza di questi controlli la macchina può inoltre essere equipaggiata con un gruppo di scarto che provvede alla separazione delle capsule che rispettano i vincoli imposti per la produzione da quelle che non li rispettano.

1.1 Descrizione tecnica

L'Adapta si inserisce nella categoria delle macchine automatiche a ciclo alternato. La sua camera di lavoro (figura 1.3) ospita una tavola rotante avente un diametro di circa 70 cm, predisposta per ospitare fino a 12 stazioni per la lavorazione del prodotto, posizionate su punti angolarmente equidistanti su di un percorso circolare.

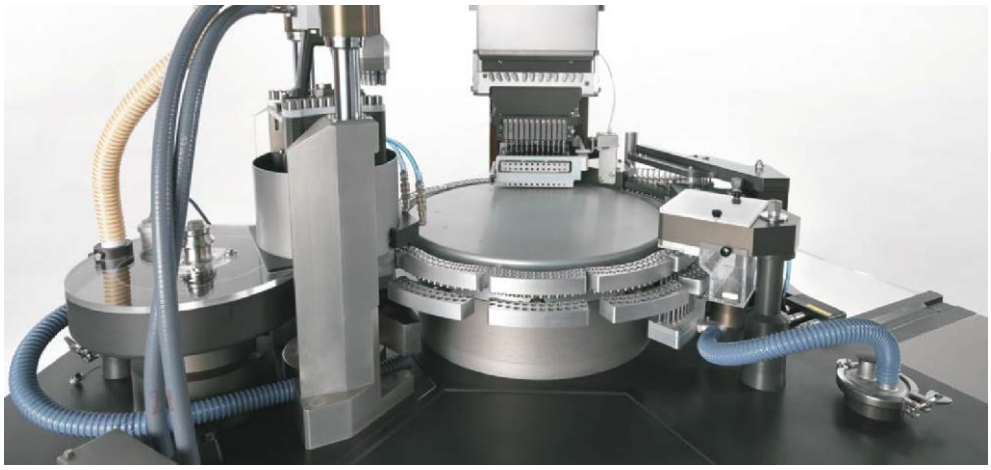


Figura 1.3: La camera di lavoro

La tavola rotante (figura 1.4) ospita al suo interno 12 unità portacapsule, in ognuna delle quali sono presenti 12 alveoli, in grado di ospitare una capsula ciascuno. Tramite poi un moto intermittente di 30°, le unità permettono la movimentazione delle capsule tra le varie stazioni, permettendone una lavorazione per fasi.

Figura 1.4: La tavola rotante

La caratteristica delle unità di trasporto è quella di movimentare contemporaneamente, ma in maniera separata, sia i coperchi che i fondelli delle capsule, dal momento che per il dosaggio delle sostanze all'interno delle capsule queste ultime devono essere aperte. Questo avviene per mezzo di due elementi meccanici (figura 1.5): in quello inferiore trovano sede i fondelli delle capsule, mentre in quello superiore i coperchi.

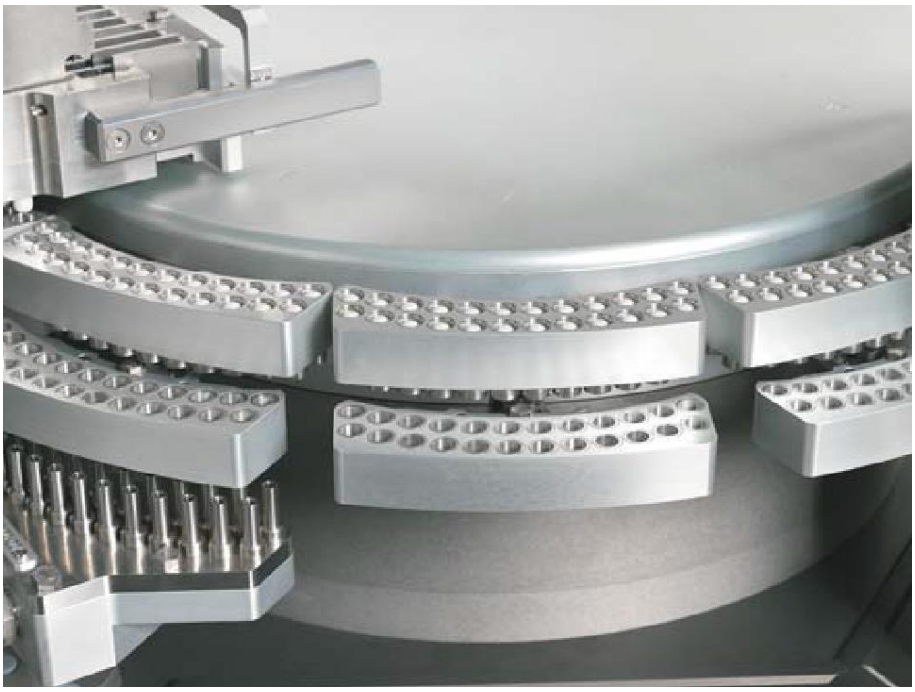


Figura 1.5: Le unità di trasporto capsule

Nella figura 1.6 viene mostrata la composizione meccanica dell'Adapta, in cui le stazioni di lavoro sono orientate secondo un normale ciclo di produzione. Nel seguito verrà illustrato l'intero flusso operativo della macchina descrivendo in maniera funzionale le singole stazioni di lavoro.

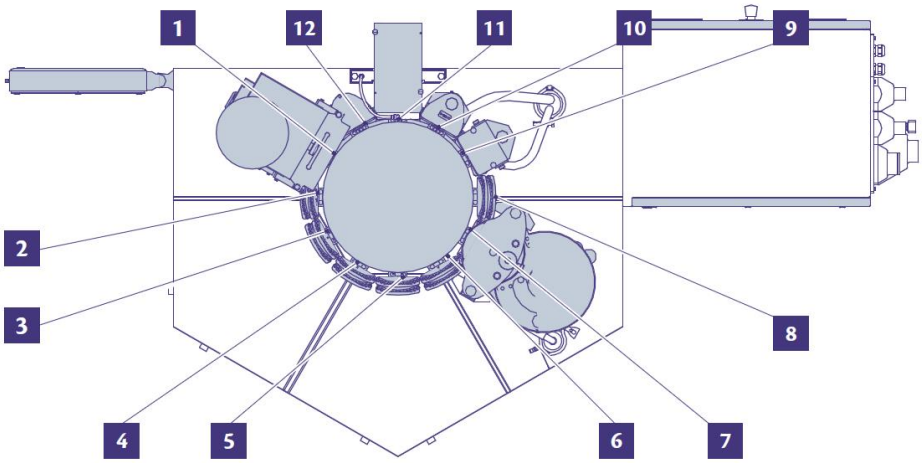


Figura 1.6: Layout della macchina

Stazione 1: in questa stazione, mostrata in figura 1.7, avviene l'inserimento delle capsule nelle unità di trasporto tramite la separazione tra coperchio e fondello. Dapprima le capsule che arrivano dalla tramoggia di carico sono accuratamente posizionate ed orientate negli alveoli, che vengono mantenuti allineati perché il fondo della capsula possa essere disposto correttamente rispetto all'alveolo presente nello strato sottostante. Successivamente, applicando una depressione sul fondello, questi si posa nell'alveolo inferiore, separandosi così dal coperchio. In questo momento il porta fondelli si muove verso una posizione più esterna rispetto al porta coperchi, in modo tale che il fondo della capsula diventi accessibile per le stazioni successive.

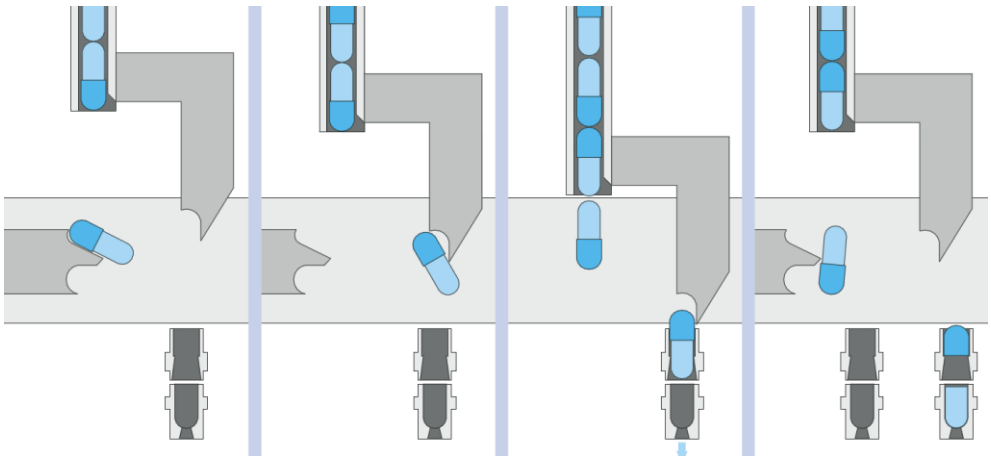


Figura 1.7: Descrizione operativa della stazione 1

Stazione 2 (PRESENZA OPZIONALE): In questa stazione avviene il controllo di presenza delle capsule: se viene rilevata l'assenza di una o più capsule, a seconda della produzione specificata dal cliente, sono previsti due modalità di funzionamento. La prima fa in modo che non venga introdotta nessuna sostanza all'interno dell'intero lotto da 12 capsule dell'unità meccanica di trasporto, che vengono così tutte scartate. La seconda prevede invece che la produzione continui effettuando il dosaggio previsto, anche se in questo modo una parte della sostanza finirà sul fondo della camera di lavoro: anche in questo caso la capsula non aperta sarà scartata. Se invece viene rilevato ripetutamente che un certo numero di capsule sono assenti dallo stesso alveolo, questo fenomeno è sinonimo dell'intasamento di uno stesso canale di alimentazione, per cui viene attivata una procedura di disintasamento dello specifico canale soffiando al suo interno aria compressa.

Stazione 3, 5, 7 (GRUPPI REMOVIBILI): Un'unità di dosaggio presente in queste stazioni può essere inserita/estratta da parte del cliente in maniera del tutto indipendente. Inoltre, le unità ospitabili rispecchiano le varie tipologie di sostanze con cui le capsule andranno riempite. In questo senso i gruppi di dosaggio acquistabili sono cinque:

- Gruppo dosaggio polvere: lo schema di funzionamento è mostrato in figura 1.8

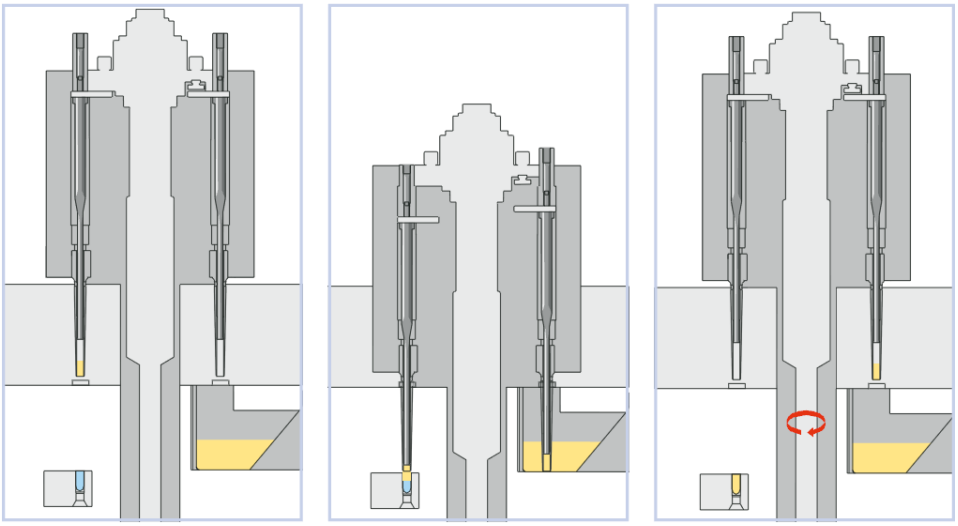


Figura 1.8: Schematico del gruppo polvere

Il gruppo è formato da una vasca contenente la polvere e da una torretta composta da 24 dosatori disposti in due gruppi da 12 dosatori ciascuno montati in posizioni opposte: la torretta segue un moto che la porta a ripetizione da una posizione alta ad una bassa e viceversa. In posizione bassa un gruppo di dosatori è immerso nella vasca contenente la polvere per prelevarne la giusta quantità da dosare,

mentre l'altro è posto in prossimità dei fondelli e si occupa di inserire in essi la polvere precedentemente prelevata. In posizione alta, invece, il gruppo di dosatori compie una rotazione di 180° al fine di portare i dosatori riempiti di polvere nella giusta posizione per effettuare il dosaggio nel fondello ed i dosatori vuoti in posizione sovrastante la vasca della polvere. Riguardo la quantità di polvere prelevata dalla vasca vengono fornite al cliente due soluzioni. La prima fa in modo che l'area della camera di raccolta della polvere all'interno del dosatore non subisca variazioni e che venga calcolata sulla base della quantità di polvere da inserire nelle capsule, specificata nella ricetta. La seconda, chiamata autoregolazione, prevede la possibilità di regolare l'area della camera di raccolta della polvere all'interno del dosatore in base ad alcune valutazioni effettuate in maniera automatica, le quali forniscono un feedback sulla quantità di polvere appena dosata. In quest'ultimo caso è richiesta la presenza di un motore ausiliario per regolare la corsa dei dosatori, che viene impostata per via meccanica attraverso alcune viti di regolazione presenti nei gruppi che non supportano l'autoregolazione. La misura indiretta della quantità di polvere dosata all'interno delle capsule può essere effettuata dotando questo gruppo di celle di carico, le quali sono installate all'interno dei gruppi di dosatori ed effettuano una misura dello sforzo di compattazione della polvere durante la fase di prelievo dalla vasca.

- Gruppo dosaggio cronoidi: lo schema di funzionamento è mostrato in figura 1.9

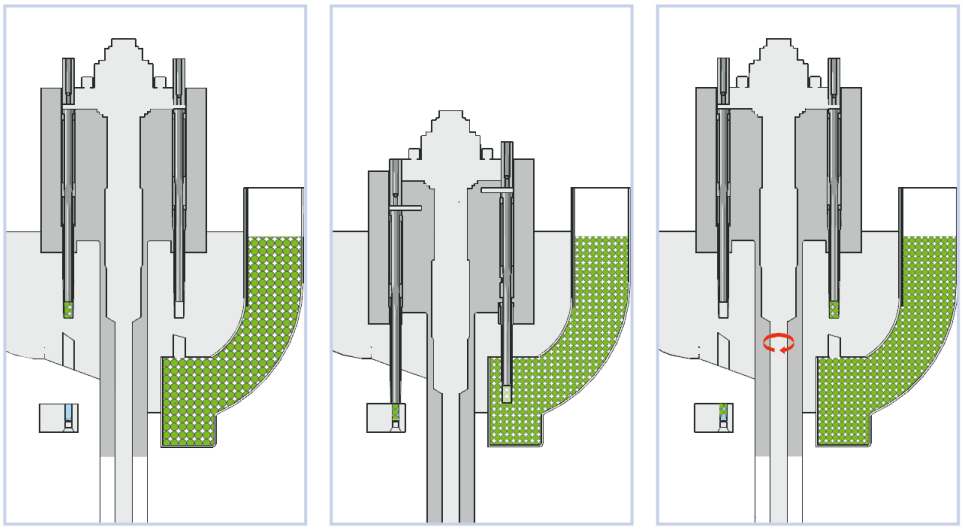


Figura 1.9: Schematico del gruppo cronoidi

Il gruppo è formato da una tramoggia contenente i cronoidi e da una torretta composta da 24 dosatori disposti in due gruppi da 12 dosatori ciascuno montati in posizioni opposte: la torretta segue un moto che la porta a ripetizione da una posizione alta ad una bassa e viceversa. In posizione bassa un gruppo di dosatori è immerso nella tramoggia cronoidi per prelevarne la giusta quantità da dosare, mentre l'altro è posto in prossimità dei fondelli e si occupa di inserire in essi i cronoidi precedentemente prelevati. In posizione alta, invece, il gruppo di dosatori compie una rotazione di 180° al fine di portare i dosatori riempiti di cronoidi nella giusta posizione per effettuare il dosaggio nel fondello ed i dosatori vuoti in posizione sovrastante la tramoggia.
Anche per questo gruppo viene data al cliente la possibilità di scegliere tra un gruppo semplice ed uno in grado di autoregolare la quantità di cronoidi prelevati dalla tramoggia.

- Gruppo dosaggio microcompresse: il gruppo (figura 1.10) è formato da una vasca che contiene le microcompresse e da un tamburo, componente a forma di corona circolare che compie un movimento rotatorio e sulla cui superficie esterna sono impressi alcuni alveoli suddivisi in gruppi e posti in zone equidistanti.

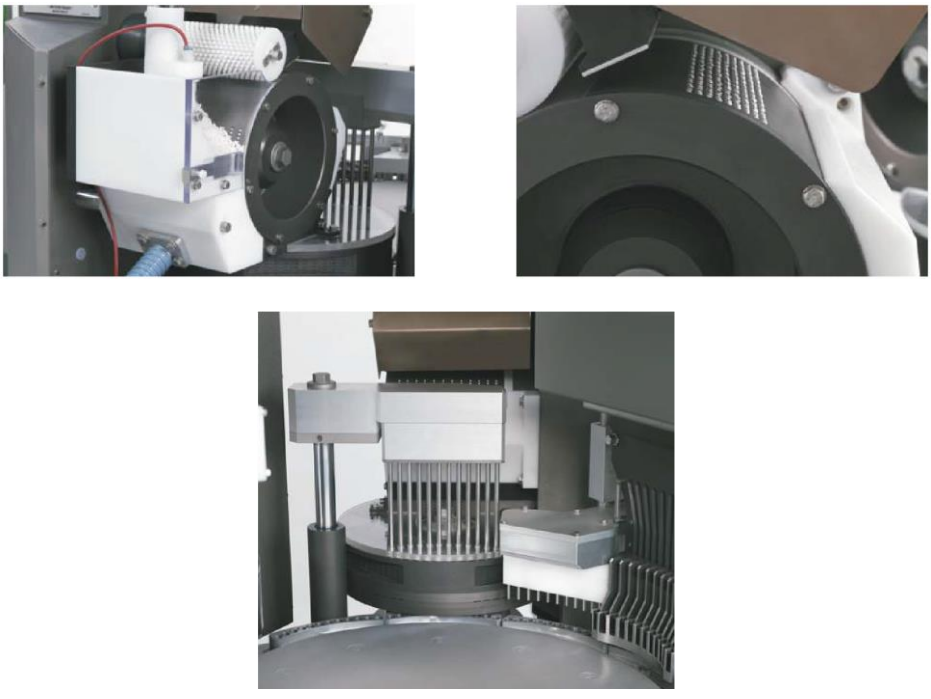


Figura 1.10: Alcune immagini del gruppo microcompresse

Il numero di alveoli impressi in ogni gruppo e la loro dimensione variano a seconda della scelta produttiva del cliente, a seconda rispettivamente del numero di microcompresse che devono essere dosate nelle capsule e dalla loro dimensione. Le microcompresse sono dosate nelle capsule dopo che il tamburo ha effettuato una certa

rotazione: la presenza di alcuni fori di aspirazione dentro gli alveoli permette di attrarre e mantenere le microcompresse al loro interno. Al fine di effettuare valutazioni in maniera indiretta sul numero di microcompresse inserite all'interno delle capsule è possibile utilizzare una telecamera per verificare la presenza di tutte le microcompresse negli alveoli prima che avvenga l'operazione di rilascio delle microcompresse provenienti dal tamburo. Un altro tipo di controllo riguarda la potenziale errata presenza di microcompresse non rilasciate, effettuato prima di rieseguire una nuova operazione di caricamento microcompresse. Entrambi i tipi di controlli sulla presenza/assenza microcompresse hanno come effetto lo scarto della singola capsula.

- Gruppo dosaggio compresse: il gruppo (figura 1.11) è formato da un vibratore pneumatico, diverse elettrovalvole, una spazzola e da una piastra di alimentazione a 12 condotti che trasporta le compresse dalla vasca fino ad una posizione corretta per il loro dosaggio all'interno dei fondelli.

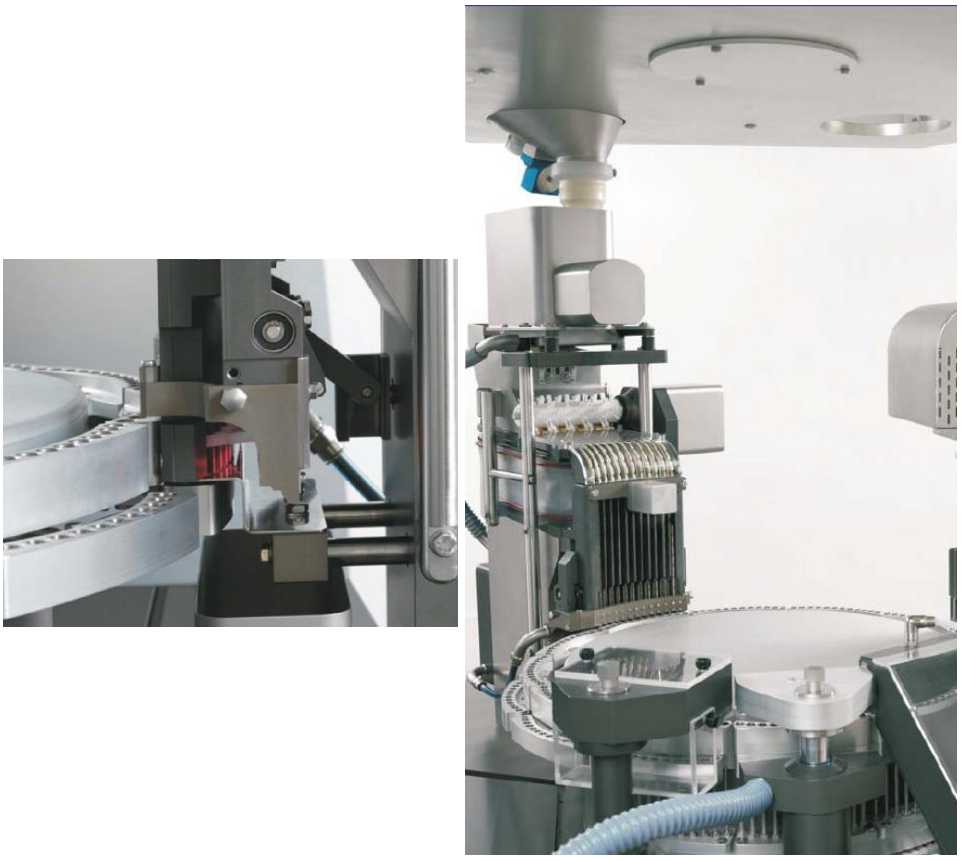


Figura 1.11: Alcune immagini del gruppo compresse

Il vibratore e la spazzola incanalano nei condotti le compresse che arrivano dalla vasca: all'ingresso dei condotti una prima elettrovalvola permette alle compresse di entrare in una camera di predosaggio, le quali poi raggiungono la fine degli stessi attraverso l'apertura di una seconda elettrovalvola, posta in posizione terminale, non appena la capsula da riempire si trova in una posizione idonea per il dosaggio. Anche per questo gruppo esistono controlli di misura indiretta sul numero di compresse introdotte all'interno delle capsule: questi avvengono tramite l'ausilio di una fotocellula per esaminare l'interno della camera di predosaggio. La fotocellula ha una doppia valenza: serve sia per calcolare il numero di compresse prima che queste vengano inserite nelle capsule, sia per controllare che siano state introdotte tutte le compresse all'apertura della seconda elettrovalvola. In entrambi i casi, la metodologia applicata porta allo scarto della singola capsula.

- Gruppo dosaggio liquidi: lo schema di funzionamento è mostrato in figura 1.12

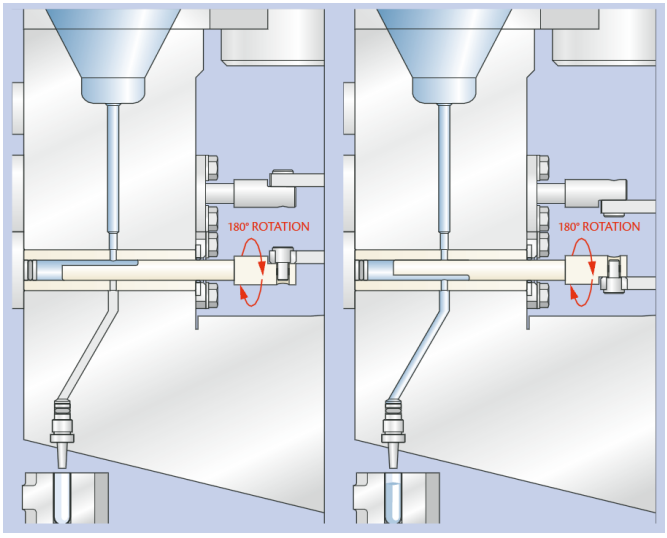


Figura 1.12: Schematico del gruppo liquidi

Il gruppo effettua un dosaggio all'interno delle capsule di tipo volumetrico, attraverso un impianto di 12 pistoni i cui steli possono sia traslare che ruotare. Il procedimento è il seguente: quando lo stelo effettua la corsa all'indietro, il liquido presente nella tramoggia della camera del pistone stesso viene aspirato, mentre quando effettua la corsa in avanti il liquido finisce nella capsula attraverso alcuni ugelli. Viene utilizzata la rotazione dello stelo quando lo si vuole impiegare come fosse una valvola: in fase di risucchio dalla tramoggia ostruisce la via verso gli ugelli, mentre in fase di invio del liquido alla capsula ostruisce la via verso la tramoggia. A discrezione del cliente può essere poi fornito un impianto di riscaldamento del liquido in

tramoggia per consentire di introdurre nelle capsule la sostanza ad una temperatura desiderata.

Stazioni 4, 6, 8 (PRESENZA OPZIONALE): la presenza dei gruppi in queste stazioni è opzionale e, ove presenti, hanno il compito di attuare un controllo indiretto del dosaggio per le stazioni 3, 5, 7 rispettivamente. Il loro montaggio meccanico è direttamente collegato a quello del rispettivo gruppo di dosaggio: per cui, in caso di inserimento/rimozione di un gruppo di dosaggio, andrà riveduto meccanicamente anche il rispettivo gruppo di controllo. Nella maggior parte dei casi non sono previsti elementi in questa stazione, poiché il controllo del dosaggio può avvenire direttamente entro il gruppo stesso. Un caso particolare in cui in queste stazioni sono presenti degli elementi si verifica quando nella stazione precedente è presente un gruppo cronoidi: in questo senso potrebbero esserci degli LVDT per effettuare una verifica dei prodotti dosati ed eventualmente lo scarto della singola capsula.

Stazione 9: qui viene attuato lo scarto di tutte quelle capsule non correttamente aperte.

Stazione 10: qui si attua la chiusura delle capsule in due fasi: nella prima le due distinte unità di trasporto per coperchi e fondelli vengono riallineate, mentre nella seconda avviene la pressione del coperchio contro il fondello.

Stazione 11: qui le capsule escono dall'unità di trasporto.

Stazione 12: qui le due unità di trasporto dei coperchi e dei fondelli vengono pulite dai residui della lavorazione attraverso un canale di aspirazione coadiuvato da un soffio di aria compressa all'interno degli alveoli.

1.2 Architettura hardware

La componentistica elettrica della macchina adibita al controllo è per lo più locata all'interno di un quadro elettrico, posto a lato della macchina stessa. Altri componenti, invece, trovano alloggiamento al di sotto della camera di lavoro e all'interno della stessa: opzionalmente può essere richiesta dal cliente l'installazione di un quadro elettrico aggiuntivo per ottenere funzionalità supplementari. Di seguito verrà illustrata solo quella parte di componentistica necessaria alla comprensione del lavoro svolto.

1.2.1 Unità di elaborazione

All'interno del quadro elettrico trovano posto due personal computer di marca B&R (figura 1.13) utilizzati rispettivamente per svolgere le funzioni di controllo macchina e di gestione dell'interfaccia uomo-macchina. Le due unità sono in relazione tra loro attraverso la tecnologia Opc, la quale poggia su di un collegamento Ethernet per attuare la comunicazione: la connessione tra le unità avviene attraverso un server Opc (fornito dalla società KW Software) installato sull'unità di controllo e che implementa l'interfaccia OPC Data Access.

Figura 1.13: Unità di elaborazione B&R

L'unità per la gestione dell'interfaccia uomo-macchina è collegata ad un monitor di tipo Touch Screen per lo scambio di informazioni con l'operatore. Su di essa è installato il sistema operativo Microsoft Windows XP, il quale esegue l'applicazione Xima d'interfacciamento con l'utente. Le caratteristiche di questa unità sono le seguenti:

- CPU: Intel Core2 Duo T7400
- Ram: 2 GB
- Hard Disk Drive: 40 GB

L'unità per il controllo macchina è equipaggiata con il sistema operativo Real Time VxWorks, con le seguenti caratteristiche:

- CPU: Pentium M 1,4 GHz
- Ram: 256 MB

1.2.2 Unità di input/output

Il sistema di controllo della macchina in esame si avvale di 3 differenti tecnologie fieldbus:

- Powerlink: la più utilizzata per il comando ed il controllo del sottosistema di I/O
- Mechatrolink: impiegata per il comando ed il controllo dei motori di marca Yaskawa
- CANopen: adottata per lo scambio di informazioni con le bilance necessarie per il controllo di qualità della produzione

1.3 Architettura software

La componentistica software si compone di due parti distinte ma completamente connesse al fine di implementare un comportamento univoco relativo all'interno sistema: questa divisione riflette la struttura adottata con l'impiego di due unità di elaborazione differenti, una per ogni funzionalità. La prima parte è quella relativa al software di controllo, che risiede sull'unità specifica di controllo macchina: l'architettura (mostrata in figura 1.14) prevede alla base l'utilizzo del sistema operativo real-time VxWorks, sviluppato dalla Wind River System, per consentire l'esecuzione di applicazioni in cui è richiesto il rispetto di vincoli temporali. Nello strato superiore trova sede RTPLC, un insieme di moduli software sviluppato da IMA per la gestione dell'hardware del pc e l'esecuzione dei task real-time.

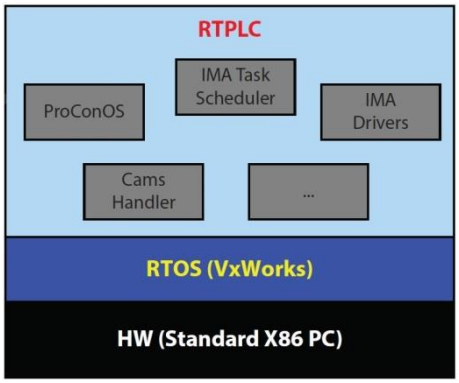


Figura 1.14: Architettura dell'unità di controllo macchina

La seconda parte, relativa alla gestione dell'interfaccia uomo-macchina, risiede sull'unità di elaborazione: l'architettura poggia attualmente sul

sistema operativo Microsoft Windows XP, su cui sono in esecuzione una serie di applicativi software, ognuno con un compito ben definito. Il primo di essi è Xima, sviluppato internamente da IMA, che ha il compito di gestire il vero e proprio interfacciamento tra la macchina e l'operatore per via grafica. Il secondo è Report Generator, un applicativo sviluppato da IMA per la gestione e la trasformazione dei rapporti informativi generati da Xima.

1.3.1 Interfaccia grafica uomo-macchina

PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

Figura 1.15: Xima in esecuzione sul monitor Touch Screen in macchina

PAGINA OMESSA A TUTELA
DELLA PROPRIETA'
INDUSTRIALE DI IMA SpA

PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

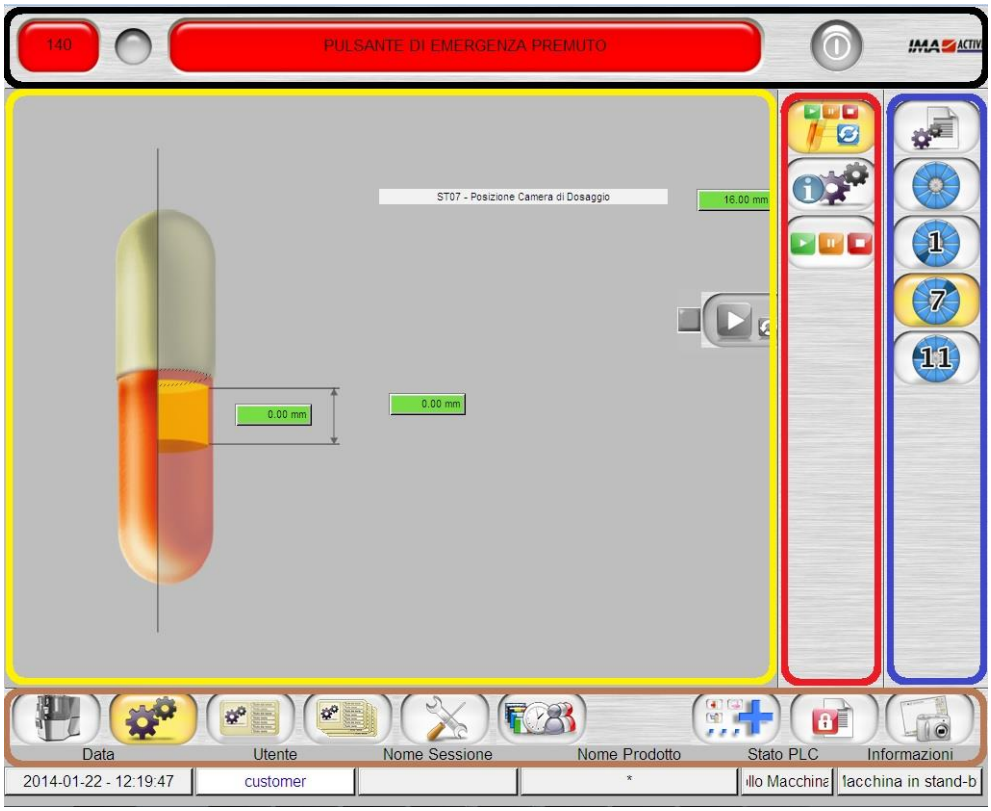
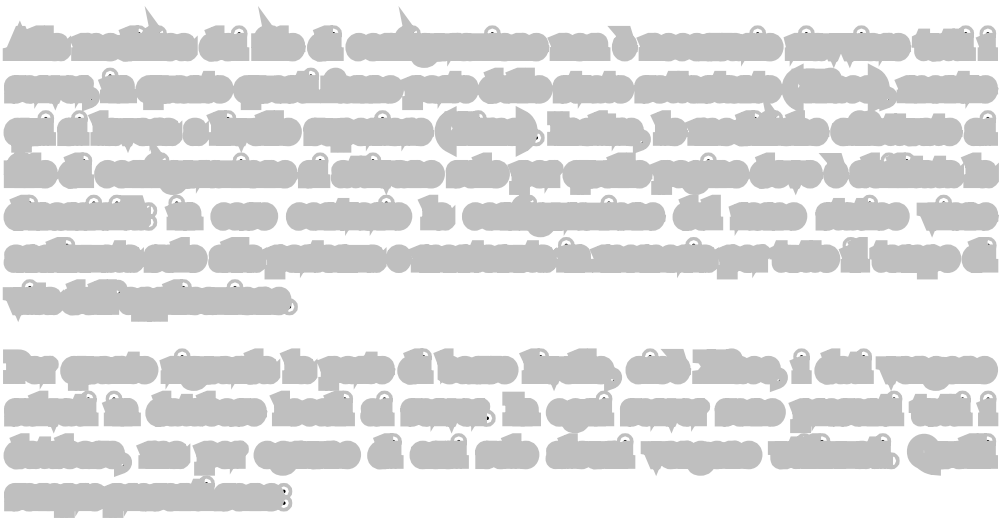


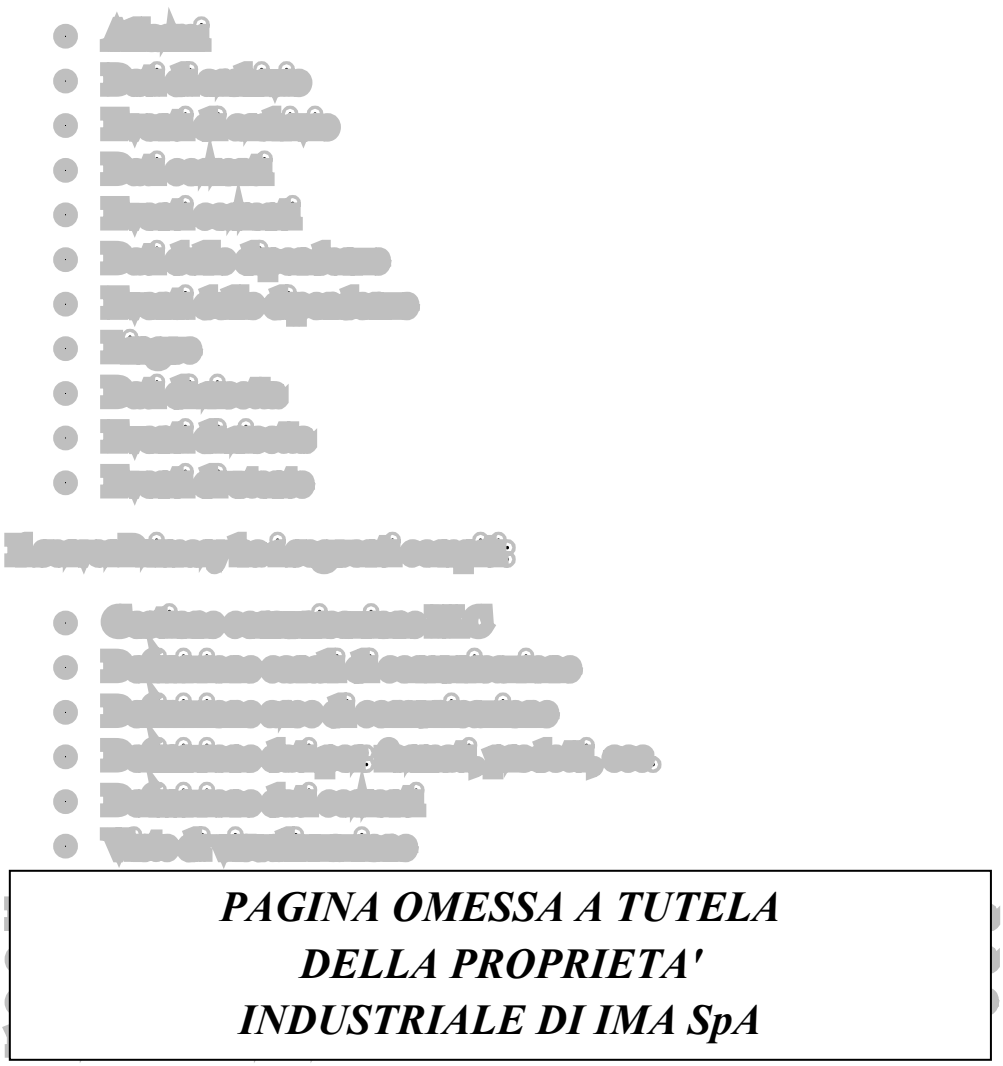
Figura 1.16: Screenshot dell'interfaccia Xima

PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

```
1  @
2      1                                                @ TIPO (1 =  TAB_FORM_TYPE)
3      TabBackground                                    @ SFONDO
4      0, 242, 242, 242                                 @ TRASPARENZA (1 Si - 0 No)
5      3                                                @ NUMERO DI BOTTONI
6  @
7      1                                                @ BOTTONE
8      010100_FrmSummary                                @ FORM ASSOCIATA
9      N                                                @ STATIC
10     "MachineSummary"                                 @ IMMAGINE BOTTONE
11     ""                                               @ DESCRIZIONE BOTTONE
12     NON_IMA                                          @ DIRITTI SPECIALI
13 @
14     2                                                @ BOTTONE
15     010200_FrmMachineInfo                            @ FORM ASSOCIATA
16     N                                                @ STATIC
17     "MachineInfo"                                    @ IMMAGINE BOTTONE
18     ""                                               @ DESCRIZIONE BOTTONE
19     NON_IMA                                          @ DIRITTI SPECIALI
20 @
21     3                                                @ BOTTONE
22     010400_FrmReport                                 @ FORM ASSOCIATA
23     N                                                @ STATIC
24     "010400_Report"                                  @ IMMAGINE BOTTONE
25     ""                                               @ DESCRIZIONE BOTTONE
26     NON_IMA                                          @ DIRITTI SPECIALI
27 *
28 @    NOTA: Non usare virgole nei commenti!!!
```

Figura 1.17: Esempio di struttura di una pagina d'interfaccia



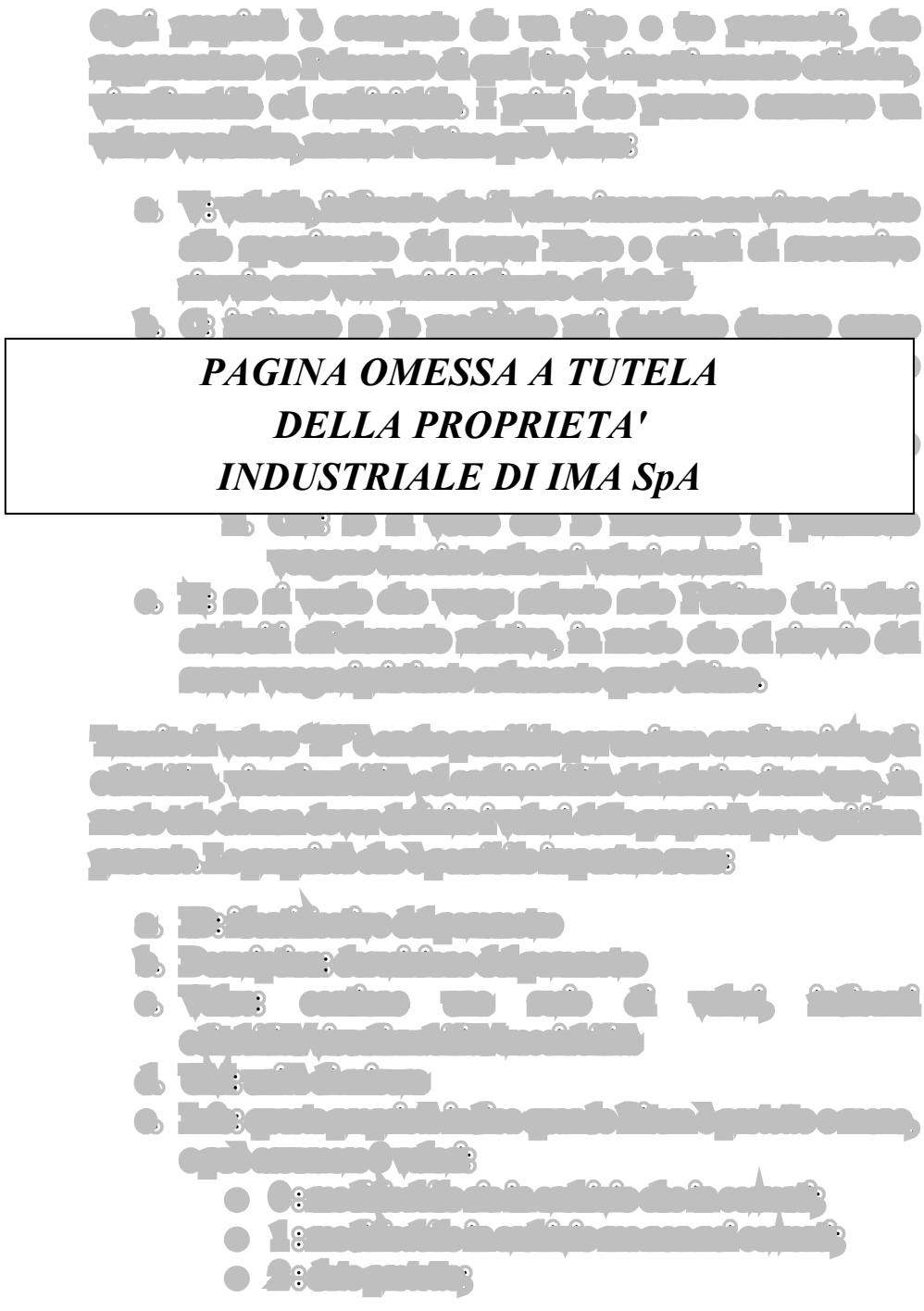



```
1274    "MachineType"
1275        "MachineTypeEnumParameter"
1276            ID                  X           X   X   "00.001"    ;
1277            Description         X           X   X   "MachineType" ;
1278            Value               X           T   N  "Channels_12"  ;
1279            UM                  X           X   X   ""            ;
1280            Info                X           X   X   0             ;
1281            Cfr                 X           X   X   Off           ;
1282            *
```

Figura 1.18: Esempio di un item definito nel server primary

```
714   MachineTypeEnumParameter
715       ID                  String              F   T   V   ;
716       Description         Linguistic          F   T   V   ;
717       Value               MachineTypeEnum     T   T   C   ;
718       UM                  Linguistic          F   T   V   ;
719       Info                Int                 F   T   V   ;
720       Cfr                 OnOff               F   T   V   ;
721       *
```

Figura 1.19: Esempio di un item type definito nel server primary



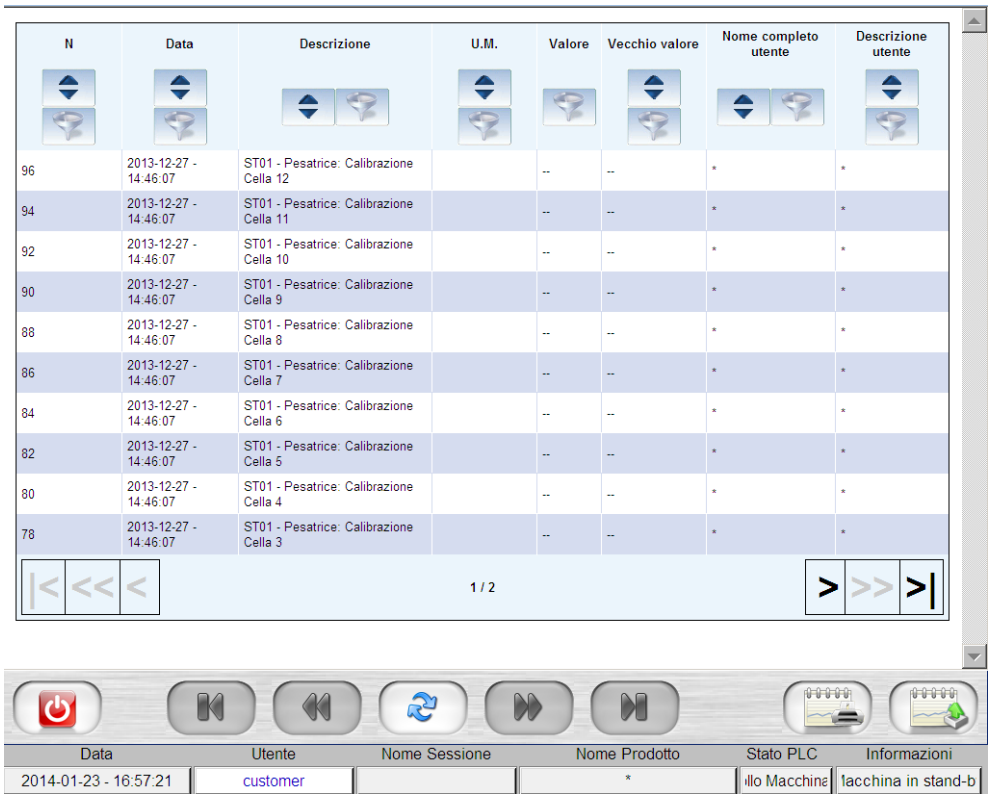

```
31184     "CurrentGenericParametersView"
31185         @SelectedProductionSpeed
31186         MachineBuzzerMode
31187         @Recipe@
31188         StopOnBatchEndEnable
31189         StopOnSubBatchEndEnable
31190         BatchSize
31191         SubBatchSize
31192         *
```

Figura 1.20: Esempio di un item view definito nel server primary

```
35052      "MotionPhasesData"
35053  @   __CRC__
35054  @   __HEADER__
35055          WeigherUnitClockPhaseCamLdEd                    Value
35056          WeigherUnitClockPhaseCamTrEd                    Value
35057          WeigherUnitClockPhaseCamCmT                     Value
35058          WeigherUnitClockPhaseCriticalPhase              Value
35059          WeigherUpperLockPhaseCamLdEd                    Value
35060          WeigherUpperLockPhaseCamTrEd                    Value
35061          WeigherUpperLockPhaseCamCmT                     Value
35062          WeigherUpperLockPhaseCriticalPhase              Value
35063          WeigherLowerLockPhaseCamLdEd                    Value
35064          WeigherLowerLockPhaseCamTrEd                    Value
35065          WeigherLowerLockPhaseCamCmT                     Value
35066          WeigherLowerLockPhaseCriticalPhase              Value
35067          WeigherSamplingPhaseCamLdEd                     Value
35068          WeigherSamplingPhaseCamTrEd                     Value
35069          WeigherSamplingPhaseCamCmT                      Value
35070          WeigherSamplingPhaseCriticalPhase               Value
35071          WeigherFlapRejectPhaseCamLdEd                   Value
35072          WeigherFlapRejectPhaseCamTrEd                   Value
35073          WeigherFlapRejectPhaseCamCmT                    Value
35074          WeigherFlapRejectPhaseCriticalPhase             Value
35075  @   __TAIL__
35076          *
```

Figura 1.21: Esempio di un item property view definito nel server primary

1.3.2 Sistema di reportistica



N	Data	Descrizione	U.M.	Valore	Vecchio valore	Nome completo utente	Descrizione utente
96	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 12		--	--	*	*
94	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 11		--	--	*	*
92	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 10		--	--	*	*
90	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 9		--	--	*	*
88	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 8		--	--	*	*
86	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 7		--	--	*	*
84	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 6		--	--	*	*
82	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 5		--	--	*	*
80	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 4		--	--	*	*
78	2013-12-27 - 14:46:07	ST01 - Pesatrice: Calibrazione Cella 3		--	--	*	*

1 / 2

Data	Utente	Nome Sessione	Nome Prodotto	Stato PLC	Informazioni
2014-01-23 - 16:57:21	customer		*	llo Macchina	lacchina in stand-b

Figura 1.22: Screenshot della visualizzazione di un report su interfaccia

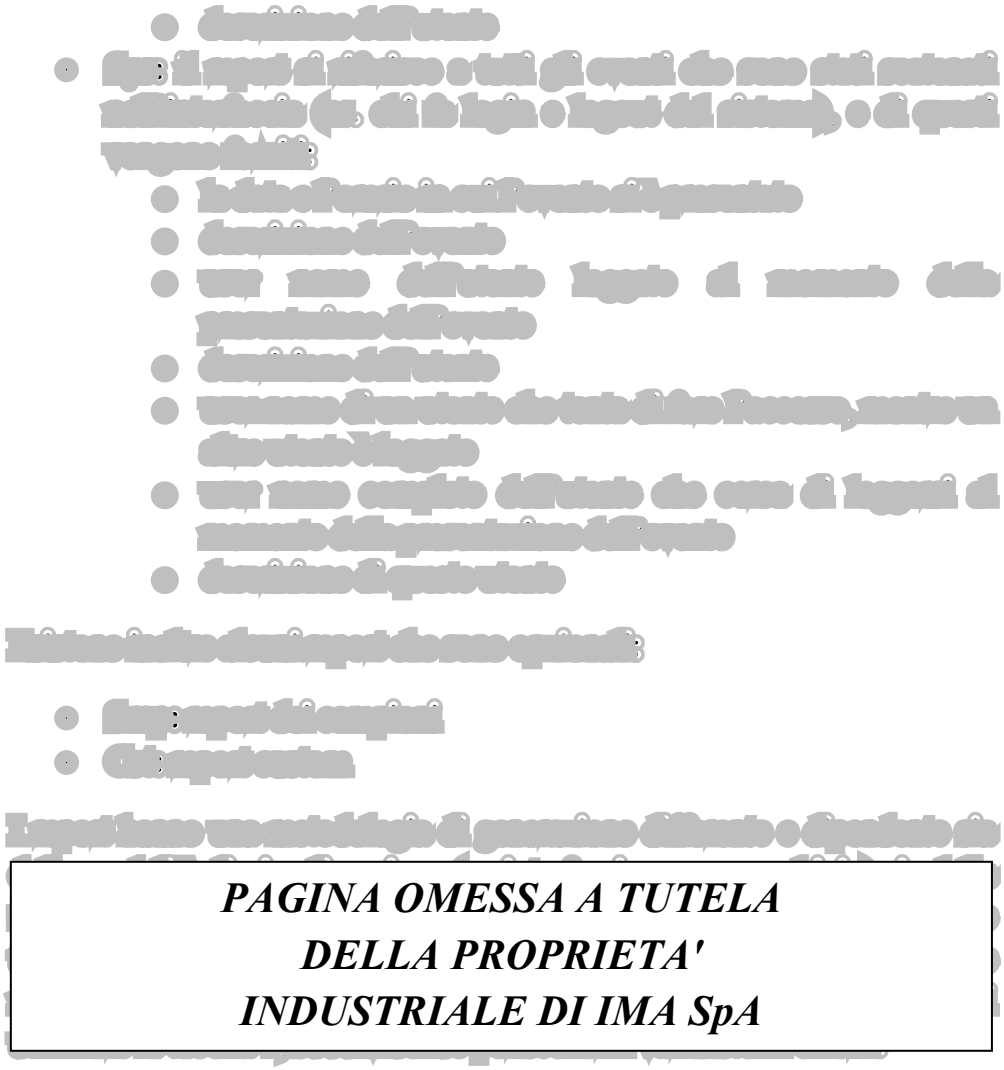
PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

[illegible]

- [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
- [illegible]
  - [illegible]
  - [illegible]
  - [illegible]

PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

- [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
- [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]



PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

	Produzione in corso	Produzione terminata
Interfaccia	Scenario A	Scenario B
Archivio	Scenario C	Scenario D

Tabella 1

PAGINA OMESSA A TUTELA
DELLA PROPRIETA'
INDUSTRIALE DI IMA SpA

[redacted]

[redacted]

- [redacted]
- [redacted]

PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

[redacted]

[redacted]

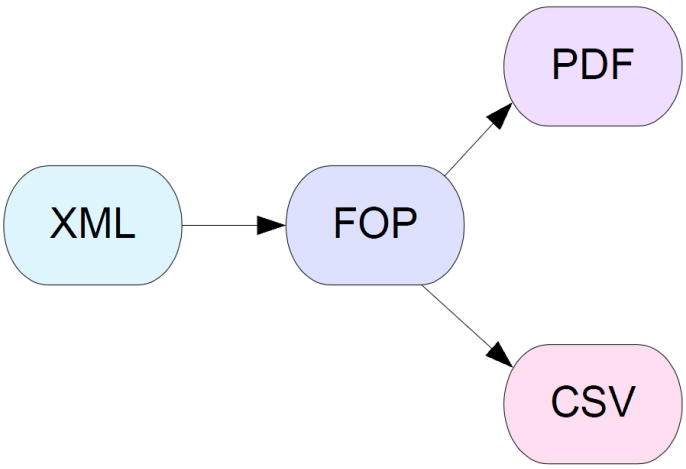


Figura 1.23: Processo di trasformazione dei report xml via Report Generator

PAGINA OMESSA A TUTELA DELLA PROPRIETA' INDUSTRIALE DI IMA SpA

```
11  <transformation>
12    <name>Generate PDF</name>
13    <input>
14      <dir>C:\Precisa\Report</dir>
15      <ext>.fop.xml</ext>
16    </input>
17    <output compression="none">
18      <dir>C:\Precisa\Report</dir>
19      <ext>.all.pdf</ext>
20    </output>
21    <stx dir="C:\WORK\ReportGenerator\stx\pdf"/>
22    <fop mode="2pass"/>
23  </transformation>
```

Figura 1.24: Estratto di file di configurazione per i report

PAGINA OMESSA A TUTELA
DELLA PROPRIETA'
INDUSTRIALE DI IMA SpA

PAGINA OMESSA A TUTELA
DELLA PROPRIETA'
INDUSTRIALE DI IMA SpA

PAGINA OMESSA A TUTELA
DELLA PROPRIETA'
INDUSTRIALE DI IMA SpA

PAGINA OMESSA A TUTELA
DELLA PROPRIETA'
INDUSTRIALE DI IMA SpA

1.4 Caratteristiche distintive

La particolarità di questa macchina automatica è relativa al fatto che la sua natura la contraddistingue come dinamicamente riconfigurabile. Lo scenario è quindi quello relativo ad una macchina automatica composta da componenti elettro-meccanici alcuni dei quali possono essere sostituiti a freddo. In questo contesto la configurazione della macchina può essere cambiata dal cliente in maniera dinamica ed individuale (una volta acquistati i componenti necessari) a seconda delle funzionalità che si vogliono aggiungere. Pertanto, per riconfigurabilità si intende la possibilità di cambiare alcuni componenti della macchina, consentendole di raggiungere una certa flessibilità operativa sui possibili prodotti. Per dinamicità invece si intende la possibilità di cambiare alcuni componenti della macchina da parte del cliente senza l'ausilio della casa costruttrice al bisogno, in un qualsiasi momento in cui esso voglia produrre lavorazioni differenti.

In questo ambito, la macchina possiede:

- alcuni componenti nativi (o di primo livello) che la rendono impostata fin dall'origine per un certo tipo di prodotto e che non possono essere cambiati durante il suo ciclo di vita

- alcuni componenti che possono essere dinamicamente aggiunti/sottratti per creare prodotti differenti

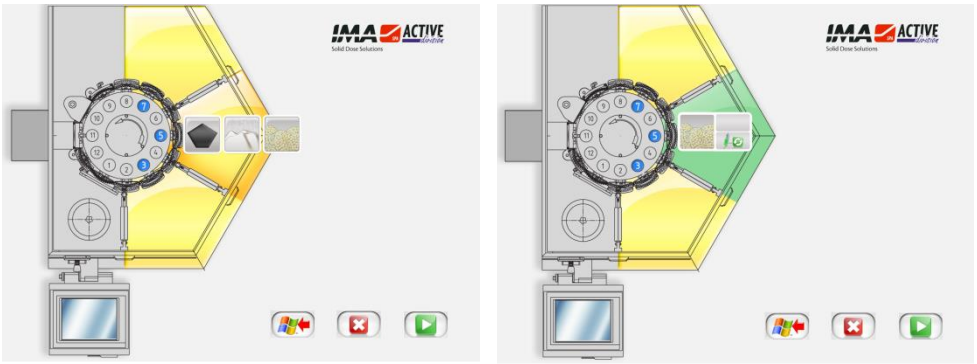
1.5 Procedura di variazione della configurazione operativa

1.5.1 Scenario presente

Attualmente il procedimento di variazione della configurazione operativa prevede che, a partire da una determinata impostazione meccanica con cui viene composta una certa macchina automatica, dapprima in ufficio vengano precompilate una molteplicità di interfacce uomo-macchina, che saranno poi successivamente locate ciascuna in una cartella separata sulla memoria di massa dell'unità di gestione dell'interfaccia. Ognuna di esse è relativa ad una specifica lavorazione di prodotto: la loro creazione viene effettuata per mezzo di un'interfaccia di configurazione chiamata "Adapta_CFG". Quest'ultima permette di comporre un'interfaccia per ogni specifica produzione tramite parametrizzazione manuale dei dati relativi ad un'interfaccia principale contenente tutti i file che è possibile selezionare per ogni configurazione che è realizzabile.

Una volta prodotte tutte le configurazioni relative ai prodotti richiesti dal cliente, esse vengono memorizzate sull'unità di gestione dell'interfaccia uomo-macchina in modo tale che sia possibile selezionare quella specifica legata ad una certa lavorazione per via manuale. Questo obiettivo è raggiunto per mezzo dell'applicativo chiamato StartApplication il quale, eseguito in macchina prima dell'avvio della produzione, consente all'operatore di indicizzare una tra le possibili configurazioni generate in precedenza. Questa applicazione, visualizzando il layout dell'Adapta come mostrato in figura 1.25, permette per via grafica la selezione esatta dei gruppi di dosaggio montati in macchina per una specifica configurazione di lavoro: in questo senso l'operatore che vuole effettuare una modifica deve operare sulle icone relative alle stazioni di lavoro per specificare i gruppi di dosaggio effettivamente montati. Le unità di dosaggio tra cui è possibile compiere una scelta sono specificate all'interno di un file excel: il colore giallo chiaro indica che per una certa stazione non è ancora stato selezionato un gruppo di dosaggio, mentre il colore giallo scuro indica che è in corso la scelta del gruppo di dosaggio per la stazione che è stata selezionata

(immagine 1.25(a)). Col colore verde invece si indica che la scelta è conclusa e perciò sulla stazione in esame permane l'icona del gruppo di dosaggio selezionato (immagine 1.25(b)). Unitamente a queste informazioni relative al software d'interfaccia, questa applicazione permette di gestire il cambio di configurazione anche dal punto di vista dell'unità di controllo: le informazioni relative ai gruppi di dosaggio effettivamente montati vengono inviate di conseguenza all'unità di controllo per consentire ad essa l'avvio della macchina secondo la corretta configurazione. Al termine di questa fase, con la pressione del bottone di avvio, l'operatore provoca lo spegnimento della macchina ed il suo successivo riavvio per permette ad essa di avviarsi secondo la configurazione operativa precedentemente specificata.



(a) Selezione di un gruppo di dosaggio (b) Gruppo di dosaggio selezionato

Figura 1.25: Screenshot dell'applicativo StartApplication per la variazione della configurazione operativa

Tramite questo meccanismo è quindi possibile adattare l'interfaccia visualizzata ad una specifica produzione selezionando per via manuale quale delle interfacce generate a priori eseguire secondo una determinata configurazione.

I punti deboli dell'attuale architettura sono molteplici: il primo è legato alla cardinalità del numero di interfacce che è necessario produrre, che è pari in numero a tutte le possibili configurazioni di lavoro, con conseguente necessità di collaudo di ognuna di esse. Oltre a ciò, il meccanismo stesso di generazione dell'Hmi, per come è stato progettato, è di difficile gestione ed aggiornamento: la sua elevata complessità porta sia a non permettere il riutilizzo di una stessa configurazione d'interfaccia per clienti successivi, sia alla facilità di incorrere in errori durante la fase di precompilazione delle interfacce, quando infatti sono necessari fino a 3/4 tentativi prima di arrivare ad una versione di Hmi compatibile con una configurazione di macchina specificata. Questa facilità di incappare in malfunzionamenti è legata alla difficoltà nel definire le condizioni per le quali determinati elementi di

interfaccia possono comparire o meno in un'Hmi specifica per una certa configurazione. In questa ottica il tempo impiegato nel generare e nel testare differenti versioni di interfaccia, unitamente allo spazio necessario ad ospitare ognuna di esse sulla memoria di massa della macchina automatica, comportano sia una carenza nell'affidabilità e nella manutenibilità complessiva della macchina, sia una diminuzione di produttività nella gestione complessiva del progetto.

1.5.2 Scenario futuro

La nuova infrastruttura di supporto che si andrà a progettare ha l'obiettivo di rendere dinamica l'architettura relativa al cambio di configurazione, cioè in grado di raggiungere una semplicità e flessibilità di variazione della produzione almeno pari a quelle del lato meccanico, senza però scaricare le complessità derivanti sull'operatore in macchina. In questo senso, il fine è quello di elaborare una serie di tecnologie che permettano di facilitare sviluppo e collaudo delle interfacce uomo-macchina, in modo tale da innalzare notevolmente il grado di affidabilità e manutenibilità complessivo delle macchine automatiche unitamente a permettere all'azienda di incrementarne la produzione, riducendo notevolmente i tempi legati alla loro costruzione e progettazione. La riconfigurazione di una macchina coinvolge diversi aspetti che è stato necessario trattare in maniera approfondita ma non disgiunta, in modo tale da comporre, attraverso la nuova infrastruttura, un comportamento univoco e sincrono delle varie tecniche progettate.

L'obiettivo è quindi quello di superare tutte le criticità precedentemente illustrate sull'architettura esistente, partendo dalle difficoltà incontrate per le fasi di generazione, collaudo e manutenzione dell'Hmi fino alla necessità di avere fisicamente presente in macchina una vasta molteplicità di interfacce, una per ogni specifica configurazione di lavoro. In più, s'intende affrontare i problemi legati al sempre crescente bisogno, da parte dell'infrastruttura esistente, di tempo e di spazio in memoria di massa, necessari per le fasi di precompilazione ed utilizzazione rispettivamente. Il nuovo processo di riconfigurazione, incrementando il grado d'affidabilità del sistema, si rivela dunque una piattaforma centrale nella costruzione di nuove macchine automatiche, poiché consente di abilitare sull'architettura esistente tutta una serie di specifiche legate a standard implementativi largamente diffusi in ambito ingegneristico.

Capitolo 2
Stato dell'arte nei sistemi ad interfaccia grafica uomo-macchina

2.1 Il sistema SCADA

Il sistema informatico che si occupa di monitoraggio e controllo di un sistema fisico in ambito industriale viene definito SCADA, acronimo di Supervisory Control And Data Acquisition. In un sistema Scada l'acquisizione dati è funzionale allo svolgimento delle funzioni di supervisione, cioè osservazione dell'evoluzione del processo controllato, e di controllo, cioè 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. Di seguito verranno descritte brevemente le tre funzioni principali di un sistema Scada, la cui architettura è mostrata in figura 2.1.

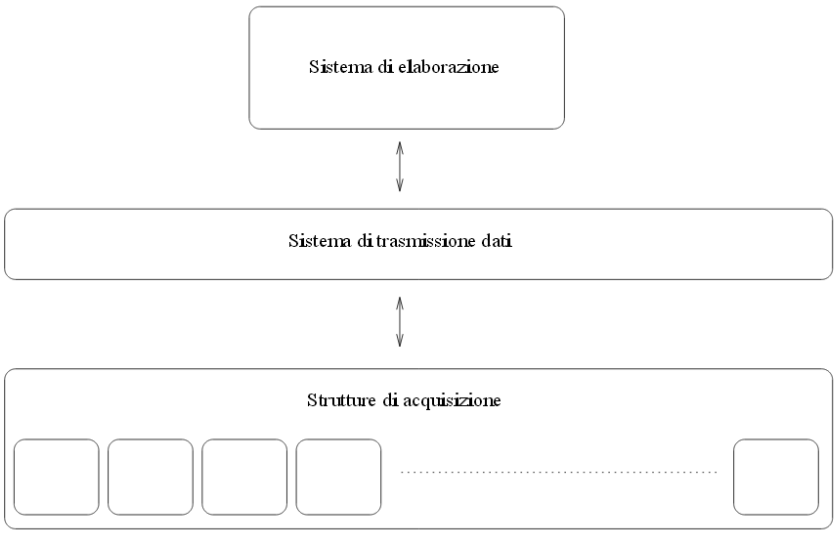


Figura 2.1: Architettura di un sistema Scada

La funzione di supervisione costituisce un fine per qualsiasi sistema Scada: essa permette l'osservazione dello stato e dell'evoluzione degli stati di un processo sotto controllo. A questa attività appartengono tutte le funzionalità di visualizzazione delle informazioni relative allo stato attuale del processo, di gestione delle informazioni passate, di gestione degli stati che costituiscono eccezioni rispetto alla normale evoluzione del processo in esame..

La funzione di controllo rappresenta la capacità di un sistema di prendere decisioni relative all'evoluzione dello stato del processo controllato in funzione dell'evoluzione del processo medesimo. La modalità tramite cui le procedure di controllo vengono realizzate all'interno dell'intera architettura del sistema dipende fortemente dal tipo di processo, essendo questo in grado imporre scelte architetturali sia hardware che software. In questo contesto i sistemi Scada sono comunemente intesi come sistemi che hanno nella funzione di acquisizione dati l'intera catena di acquisizione che dai sensori al sistema di elaborazione e archiviazione veicola informazioni che sono i dati grezzi prelevati come valori di parametri di stato del processo. Le funzionalità di controllo sono quindi concentrate nel sistema di elaborazione il quale, una volta eseguite opportune procedure di elaborazione, sfrutta il sistema di acquisizione dati in senso inverso per cambiare il valore di opportuni parametri di stato del processo controllato.

La funzione di acquisizione dati è una funzione che nella maggior parte dei casi ha un ruolo di supporto alle funzioni di supervisione e controllo poichè mette in relazione il sistema con il processo controllato, consentendo la conoscenza dello stato in cui si trova il processo e l'azione di controllo esercitata per mezzo della variazione di parametri caratteristici del processo. In questo senso con il termine acquisizione dati si intende in realtà lo scambio di dati in entrambe le direzioni: dal processo verso il sistema e viceversa. L'acquisizione dati entra nella definizione di sistema Scada per il fatto che non è possibile espletare funzioni di supervisione senza acquisire informazioni sullo stato in cui si trova il processo osservato così come non è possibile orientarne il comportamento, cioè controllarlo, senza avere la possibilità di influenzare lo stato cambiando il valore di parametri che lo caratterizzano. La funzione di acquisizione dati di un sistema Scada è considerata generalmente una funzione di scambio puro e semplice di informazioni tra la parte di sistema che realizza supervisione e controllo e processo controllato, cioè si considera assente qualsiasi processo decisionale interposto tra le strutture di supervisione e controllo e il processo controllato.

Questi sistemi sono generalmente composti da una struttura modulare, in cui ogni unità ha un compito ben definito:

- uno o più sensori, col compito di effettuare misurazioni di grandezze fisiche
- uno o più microcontrollori, che possono essere PLC o microcomputer, che, ciclicamente od ad intervalli regolari di tempo, effettuano misurazioni tramite i sensori a cui sono collegati e memorizzano i valori misurati in una memoria locale
- un sistema di telecomunicazione tra i microcontrollori e il supervisore. Può essere una rete di computer, oppure un insieme di linee seriali. Può essere basato su cavo o su radio
- un computer supervisore, che periodicamente raccoglie i dati dai microcontrollori, li elabora per estrarne informazioni utili, memorizza su disco i dati o le informazioni riassuntive, eventualmente fa scattare un allarme, permette di selezionare e di visualizzare su schermo i dati correnti e passati, eventualmente in formato grafico, ed eventualmente invia informazioni selezionate al sistema informativo aziendale
- un sistema d'interfacciamento grafico uomo-macchina, chiamato Hmi, il quale svolge i suoi compiti in maniera bidirezionale:
    - effettuare la presentazione dei dati elaborati all'operatore umano
    - permettere all'operatore umano di monitorare ed interagire col processo industriale

La realizzazione delle funzioni di un sistema di supervisione e controllo comporta sempre la realizzazione di sottosistemi responsabili dell'interazione tra gli operatori e il sistema medesimo denominati interfacce uomo-macchina (in inglese indicati con l'acronimo HMI di human-machine interface). La complessità dello sviluppo è funzione del tipo di interazione richiesta, mentre quest'ultima dipende dalle caratteristiche del processo controllato. L'interfaccia uomo-macchina può realizzare molti gradi di interazione comprendendo funzionalità di semplice osservazione dello stato di esercizio del sistema, nel caso di sistemi che realizzano procedure completamente automatizzate, o funzionalità responsabili della esecuzione di procedure manuali gestite dagli operatori.

In un sistema automatizzato, l'operatore fa da supervisore ed interagisce col sistema tramite l'Hmi. Questa deve permettere l'accesso alle variabili della base di dati con tecniche di selezione delle variabili, selezione dei campi, ordinamento, interrogazione. La comunicazione con l'operatore avviene tramite pagine grafiche che rappresentano pannelli di comando e quadri sinottici dell'impianto. Tramite i pannelli di comando viene permesso l'invio di comandi all'impianto attraverso simboli grafici.

2.2 Analisi dei requisiti di sistema

Responsabilità principale dell'interfaccia grafica uomo-macchina è quella di assistere il compito dell'operatore umano: più facile e intuitiva risulta l'interazione con il sistema Scada, migliore sarà il risultato. In questo senso, il compito principale dell'interfaccia uomo-macchina è la gestione delle interazioni tra il sistema Scada e gli operatori. Le informazioni che possono essere scambiate tra questi due attori sono:

- presentazione dei dati
- gestione dei comandi operatore

Un'interfaccia grafica utente, nota anche come GUI (dall'inglese Graphical User Interface), è un tipo di interfaccia che consente all'utente di interagire con la macchina manipolando oggetti grafici convenzionali. Il sempre maggior uso di applicazioni informatiche richiede una progettazione che sappia tenere conto dei vari possibili contesti d'uso, degli obiettivi degli utenti e delle nuove tecnologie di interazione, orientandone la realizzazione verso la comunicazione con gli utenti. Al fine di progettare architetture il più possibile aderenti a questi principi, è necessario rispettare tre criteri: accessibilità, usabilità e adattamento.

Per accessibilità si intende la caratteristica di un dispositivo, di un servizio, di una risorsa o di un ambiente d'essere fruibile con facilità da una qualsiasi tipologia d'utente. Più che riguardo alla disponibilità di questi sistemi di essere usabili dalla più grande fascia di individui, con ogni tipo di mezzo, nel contesto corrente ci si riferisce a questo termine nell'ottica di rendere la soluzione software il più facile possibile da utilizzare. Questo al fine di permettere all'utente finale di effettuare operazioni entro il suo dominio d'interesse senza la necessità di acquisire conoscenze non pertinenti ai propri obiettivi.

L' usabilità viene definita dall'ISO (International Organisation for Standardisation), come l'efficacia, l'efficienza e la soddisfazione con le quali alcuni utenti raggiungono determinati obiettivi in specificati contesti. In sostanza definisce il grado di facilità e soddisfazione con cui si compie l'interazione tra l'uomo e uno strumento software. Il termine non si riferisce a una caratteristica intrinseca dello strumento, quanto al processo di interazione tra classi di utenti, prodotto e finalità. Il problema dell'usabilità si pone quando il modello del progettista (cioè le funzionalità trasferite sul design del prodotto stesso) non coincide con il modello dell'utente finale (cioè l'idea che l'utente percepisce del prodotto e del suo funzionamento): il

grado di usabilità si innalza proporzionalmente all'avvicinamento dei due. In ogni caso, un facile accesso all'informazione non basta, poiché il sistema deve poi essere anche utilizzabile: l'accessibilità è perciò un prerequisito all'usabilità (se un sistema non è accessibile, non si può utilizzare).

Per quanto riguarda l'adattamento, si ha spesso la necessità che le interfacce utenti si possano adeguare al contesto d'uso, il quale può essere considerato relativamente a tre punti di vista: all'utente, al dispositivo e all'ambiente. Lato utente, aspetti importanti sono gli obiettivi e i relativi compiti, le preferenze e il livello di conoscenza. Nel dispositivo è importante considerare le modalità supportate, l'ampiezza e la risoluzione dello schermo, le capacità e velocità di connessione con altri dispositivi. Infine l'ambiente che ha vari aspetti che influenzano l'interazione come il livello di rumore e di luce corrente o gli oggetti che sono disponibili. L'adattamento può essere distinto in due tipologie:

- l'adattabilità, ossia la capacità di modificare aspetti su richiesta esplicita dell'utente. In riferimento ad una Gui, questa proprietà è definita a design-time
- l'adattività, ossia la capacità del sistema di modificare aspetti dinamicamente, senza esplicita richiesta dell'utente. In riferimento ad una Gui, questa proprietà è definita a run-time tramite il concetto di contesto

Gli elementi coinvolti per effettuare l'adattabilità di un'interfaccia utente sono i seguenti:

- la presentazione, cioè il layout e/o gli attributi grafici
- il comportamento dinamico, come la modalità di navigazione
- il contenuto dell'informazione fornita

Riguardo alla macchina in esame, al fine di riflettere appieno anche sullo strato software i suoi tratti distintivi, cioè la riconfigurabilità dinamica, è necessario che l'interfaccia grafica:

I. rilevi di volta in volta, all'avvio della macchina, la differente configurazione (i componenti installati) con cui è stata avviata, per permetterne un funzionamento conseguente
II. rifletta la configurazione rilevata adattando l'Hmi allo strato hardware sottostante, in maniera da poter presentare all'operatore i parametri necessari a controllare in maniera differente i vari tipi di componenti rilevati

L'adeguamento implica la modifica del layout, il che comporta:

- nascondere gli elementi grafici non utilizzati
- adattare gli elementi grafici al contesto corrente

L’adattamento è un meccanismo basato su un processo di negoziazione che coinvolge due livelli architetturali:

- i singoli elementi devono avere proprietà che li rendano adattabili
- il software deve poter gestire queste proprietà come un interprete: porzioni di interfaccia devono avere la capacità di adattarsi

Secondo queste considerazioni, sono stati definiti alcuni requisiti software che l’interfaccia utente deve rispettare nella macchina automatica in esame, elencati qui in ordine di importanza:

I. essere dotata di una flessibilità operativa almeno pari a quella della struttura meccanica, senza scaricare la complessità derivata sull’operatore. Questo aspetto implica lato software l’essere adattabile a vari tipi di prodotto, ma i cui cambiamenti permettano di offrire all’operatore un’esperienza utente quanto più possibile univoca. In questo senso, il numero di possibili configurazioni di lavoro non deve portare l’utente ad interagire con un Hmi diversa ogni volta, bensì la complessità deve essere gestita dai meccanismi software di generazione dell’interfaccia
II. possedere proprietà di usabilità ed accessibilità utente aumentata. La navigazione ed il flusso delle informazioni deve essere quanto più possibile agevolato e guidato dal software stesso, in modo tale da raggiungere tre benefici: ridurre al minimo le possibili interazioni con l’operatore, mascherare all’utente tutto ciò che è possibile gestire per via automatica ed infine rendere l’inserimento dei parametri rimanenti il più possibile facilitato. In questo modo si riducono al minimo tutti quei problemi dal punto di vista dell’operatore dovuti all’eventuale cambiamento del layout dell’Hmi
III. essere dotata di un alto tasso di manutenibilità: esiste la necessità di fornire facilità di estensione ed aggiornamento per sviluppi di future configurazioni della macchina
IV. possedere semplicità e velocità riguardo a tre aspetti: generazione, collaudo e documentazione
V. rispettare alcuni requisiti riguardo la sicurezza e la protezione nel funzionamento complessivo della macchina e la tutela di porzioni dell’architettura che devono rimanere riservate entro l’ambito della casa costruttrice, senza arrivare al cliente finale. La definizione delle configurazioni operative di macchina deve rimanere entro i confini della casa costruttrice, senza lasciare al cliente la possibilità di crearsi la propria configurazione personale. In questo modo si evitano danneggiamenti alla macchina e pericoli a persone/cose dovuti a malfunzionamenti di una non perfetta messa a punto del sistema complessivo

2.3 Modelli di architetture per la presentazione

Un'architettura viene definita nello standard ANSI/IEEE come:

"L'organizzazione basilare di un sistema, rappresentato dalle sue componenti, dalle relazioni che esistono tra di loro e con l'ambiente circostante, e dai principi che governano la sua progettazione ed evoluzione."

Avvalendosi di questa descrizione nel campo dell'ingegneria del software, la progettazione di un'applicazione rappresenta l'insieme delle attività mirate ad individuare la soluzione implementativa migliore allo scopo di centrare gli obiettivi funzionali (e quelli non funzionali) attesi dal committente e dall'utilizzatore finale. Queste attività possono essere di varia natura, possono essere svolte in tempi e modi diversi a seconda dell'approccio utilizzato, ma in generale aiutano l'architetto e il team di sviluppo a prendere decisioni importanti, spesso di natura strutturale.

La progettazione condivide con la programmazione la tendenza ad astrarre la rappresentazione delle informazioni e le sequenze logiche di elaborazione, ma il livello di dettaglio nei due casi è differente. La progettazione costruisce una rappresentazione del software che riguarda diversi aspetti, si concentra sulla struttura del sistema e sulle relazioni esistenti fra le parti costituenti, identifica le operazioni logiche che devono essere svolte, individua le modalità con cui il sistema può interagire con il mondo esterno. Il risultato della progettazione è la definizione dell'architettura del sistema, intendendo con questo termine l'organizzazione strutturale del sistema stesso, che comprende i suoi componenti software, le proprietà visibili esternamente di ciascuno di essi (l'interfaccia dei componenti) e le relazioni fra le parti.

L'architettura di un sistema software non può però essere ridotta semplicemente alla sua struttura: in realtà essa è molto di più. L'architettura include le modalità con cui le diverse parti si integrano e interagiscono a formare un tutt'uno, considera gli aspetti legati all'interoperabilità con i sistemi circostanti, rappresenta il livello con cui l'applicazione soddisfa i requisiti funzionali, comprende le caratteristiche non direttamente legate ai casi di utilizzo, ma orientate a favorire l'evoluzione nel tempo del sistema a fronte dei suoi cambiamenti strutturali e in relazione all'ambiente in cui esso è inserito (scalabilità, performance, manutenibilità, sicurezza, affidabilità, ecc.). L'architettura pertanto è una rappresentazione che permette

all'architetto di analizzare l'efficacia del progetto per rispondere ai requisiti stabiliti, di considerare e valutare le alternative strutturali in una fase in cui i cambiamenti abbiano ancora un impatto relativo sull'andamento del progetto e sul risultato finale e di gestire in modo appropriato i rischi che sono collegati alla progettazione e alla realizzazione del software.

La definizione data di architettura richiama il concetto di "componente software" nella sua forma più generale. Per componente software si intende qualsiasi entità facente parte di un sistema, a diversi livelli di dettaglio e granularità, dal semplice modulo applicativo al sottosistema complesso. Ciascun componente entra a far parte dell'architettura in funzione del ruolo che esso ricopre. L'architettura considera gli aspetti che sono inerenti la comunicazione tra le parti, si focalizza sulle modalità di interazione, tralasciando i dettagli di funzionamento interni. E' quindi possibile affermare che ogni sistema software ha una sua architettura dal momento che ciascun sistema può essere visto come un aggregato delle sue parti costituenti e delle relazioni esistenti tra loro.

Riguardo il sistema in esame, il dominio applicativo su cui si andrà ad operare è stato individuato nello strato più esterno di una tipologia di architettura denominata a livelli, ed illustrata in figura 2.2.

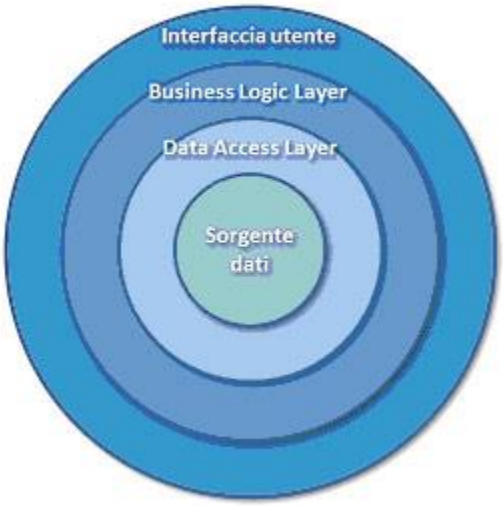


Figura 2.2: Struttura di un'architettura a livelli

Questo tipo di architettura, ospitata fisicamente su di un unico nodo macchina, prevede la strutturazione logica di un sistema software in strati sovrapposti (i layer) ma tra loro comunicanti, ciascuno caratterizzato da una forte omogeneità funzionale. La forma più nota e usata riguarda l'architettura a tre livelli composta da:

- User Interface (UI) o strato di presentazione, dove vengono gestite le interazioni dell'utente col sistema

- Business Logic Layer (BLL) o strato di business, dove sono presenti i servizi applicativi
- Data Access Layer (DAL) o strato di accesso ai dati, dove sono gestite le interazioni con il sistema di persistenza delle informazioni.

Questo lavoro di tesi si concentrerà sul livello più esterno di questa struttura, cioè quello di interfacciamento con l'utente: questo piano del sistema ha il compito di gestire la comunicazione e l'informazione con le entità esterne al sistema stesso. Esso è costituito da componenti che lavorano in maniera bidirezionale, cioè che si occupano di presentare l'informazione verso l'esterno, e che consentono al mondo esterno di interagire con il sistema per sottomettere operazioni ed ottenere risultati. In questo senso, le funzioni principali che vengono eseguite sono:

- gestione degli eventi dell'utente
- immissione di dati
- verifica coerenza fra dati e regole dell'applicazione
- risultati dell'elaborazione e visualizzazione dei dati
- notifica di eventi interni all'applicazione

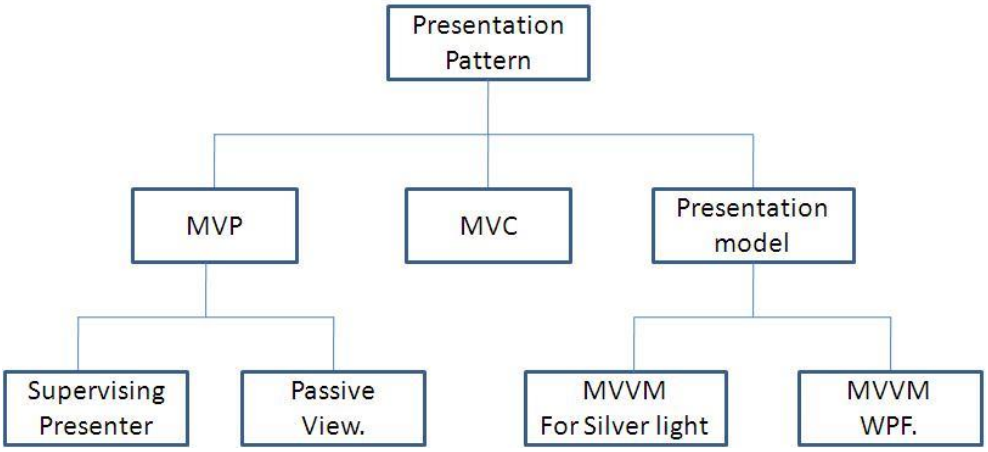


Figura 2.3: Tassonomia di pattern per lo strato di presentazione

Al fine di progettare una soluzione software per il livello indicato in precedenza, nel seguito verranno illustrati alcuni pattern architetturali specifici per lo strato di presentazione, classificati secondo lo schema illustrato in figura 2.3. Le soluzioni proposte esprimono schemi generali di base per impostare l'organizzazione strutturale di un sistema software: in questi schemi si descrivono sottosistemi predefiniti insieme con i ruoli che essi assumono e le relazioni reciproche.

2.3.1 Model-View-Controller

Questo pattern, presentato nel 1979, è uno dei primi che sono stati proposti. Fino a quel momento, la programmazione era alquanto differente dai giorni nostri: non c'erano ancora le interfacce utente di tipo grafico come le concepiamo oggi, e se ci fosse stata la necessità di averne una, si sarebbe dovuto crearla da zero, affidando la bontà del progetto solamente al talento dello sviluppatore. L'idea che viene espressa con questo pattern è quella del disaccoppiamento per responsabilità tra le entità coinvolte: il codice delegato alla visualizzazione, all'interazione, alla logica di business e all'accesso ai dati viene sviluppato in classi separate, ognuna con una distinta responsabilità, senza più mischiarlo insieme.

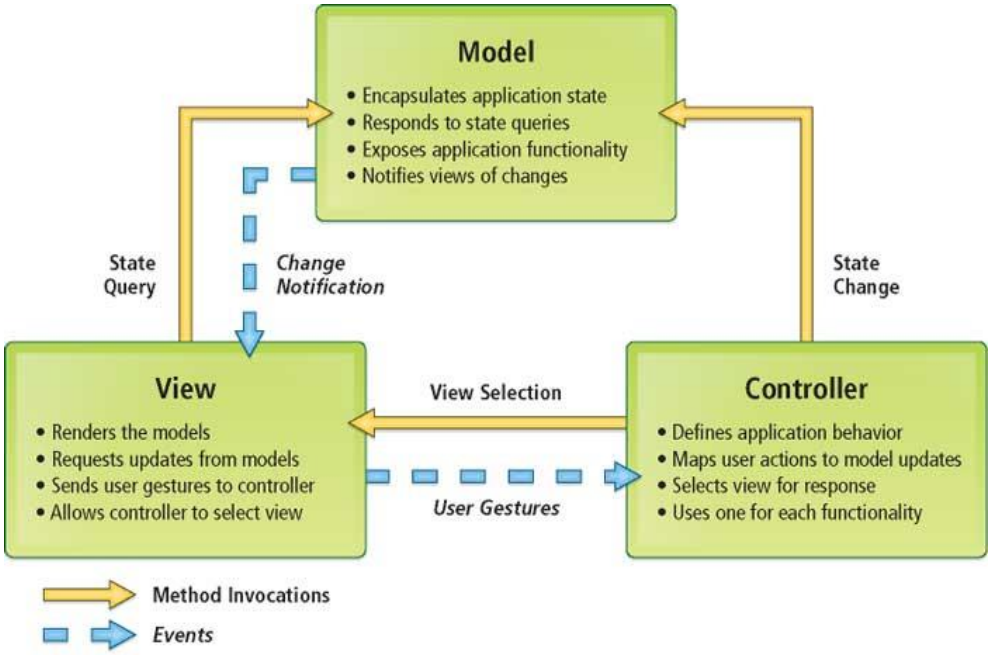


Figura 2.4: Pattern Model-View-Controller

L'architettura prevede quindi una netta separazione dei compiti tra tre componenti che si dividono i ruoli nel sistema, come illustrato in figura 2.4.

Model

Il model rappresenta tipicamente i dati presenti in un'applicazione, fornendo al contempo una logica per la loro gestione, legata alla fornitura dei servizi di accesso e persistenza. Questo componente può essere implementato in maniera differente a seconda del dominio di riferimento, talvolta come database, talvolta come web service. Esso, incapsulando lo stato dell'applicazione, definisce i dati e le operazioni che possono essere eseguite su questi. Quindi definisce le regole di business per l'interazione con i dati, esponendo alla View ed al Controller rispettivamente le

funzionalità per l'accesso e l'aggiornamento. Il Model può inoltre avere la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller, al fine di permettere alle View di presentare agli occhi degli utenti dati sempre aggiornati.

View

La View ha due responsabilità:

- visualizzare i dati contenuti nel model, aggiornandosi automaticamente non appena questi cambiano
- gestire l'interazione tra l'utente esterno e l'interfaccia

La logica di presentazione dei dati viene gestita solo e solamente dalla View. Ciò implica che questa deve fondamentalmente gestire la costruzione dell' interfaccia grafica (Gui) che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema. Ogni Gui può essere costituita da schermate diverse che presentano più modi di interagire con i dati dell'applicazione. Per far sì che i dati presentati siano sempre aggiornati è possibile adottare due strategie note come push model e pull model. Il push model adotta il pattern Observer, registrando le View come osservatori del Model. Le View possono quindi richiedere gli aggiornamenti al Model in tempo reale grazie alla notifica di quest'ultimo. Nel pull model invece la View richiede gli aggiornamenti al bisogno. Inoltre la View delega al Controller l'esecuzione dei processi richiesti dall'utente dopo averne catturato gli input e la scelta delle eventuali schermate da presentare.

Controller

Il Controller ha il compito di ricevere i comandi dell'utente per mezzo dell'interfaccia ed attuarli modificando lo stato degli altri due componenti. Questo componente ha la responsabilità di trasformare le interazioni dell'utente dalla View in azioni eseguite dal Model. Ma il Controller non rappresenta solo un semplice anello di collegamento tra View e Model. Realizzando la mappatura tra input dell'utente e processi eseguiti dal Model e selezionando la schermate della View richieste, il Controller implementa la logica di controllo dell'applicazione. L'utilizzo del pattern Strategy potrebbe semplificare la possibilità di cambiare la mappatura tra gli algoritmi che regolano i processi del Controller e le interazioni dell'utente con la View, per esempio quando è necessario cambiare il comportamento standard dell'applicazione a seconda delle circostanze.

Con l'evoluzione dei paradigmi di programmazione, i singoli elementi grafici (controlli) hanno acquisito sempre più funzionalità, finendo con l'incapsulare sia proprietà di visualizzazione che di interazione con l'utente. Basta pensare ad un controllo di tipo bottone, che ha consapevolezza del

proprio compito quando viene premuto, oppure ad una textbox, che può applicare certi comportamenti non appena gli viene immesso del testo. In questa ottica si riduce notevolmente il bisogno di un controller, ma rimane stringente il bisogno di mantenere una separazione tra le logiche per la visualizzazione, il controllo e la gestione dei dati.

Grazie a questo pattern si raggiungono due obiettivi fondamentali:

- ottenere un riuso dei componenti del Model: la separazione tra Model e View permette a diverse GUI di utilizzare lo stesso Model. Conseguentemente, i componenti del Model sono più semplici da implementare, testare e mantenere, poichè tutti gli accessi passano tramite questi componenti
- avere un supporto più semplice per nuovi tipi di client. Infatti basterà creare View e Controller per interfacciarsi ad un Model già implementato

2.3.2 Model-View-Presenter

In questo pattern, derivato direttamente dal Model-View-Controller, la View riceve gli eventi dall'interfaccia utente e chiama il Presenter al bisogno. Il Presenter ha inoltre la responsabilità dell'aggiornamento della View con i nuovi dati generati del Model, come mostrato in figura 2.5. Un comportamento rilevante è che il Presenter non comunica con la View in modo diretto, bensì a mezzo di interfacce. In questo modo, il Presenter ed il Model possono essere testati in maniera separata. E' di fatto uno dei pattern più usati per realizzare sistemi a basso accoppiamento funzionale.

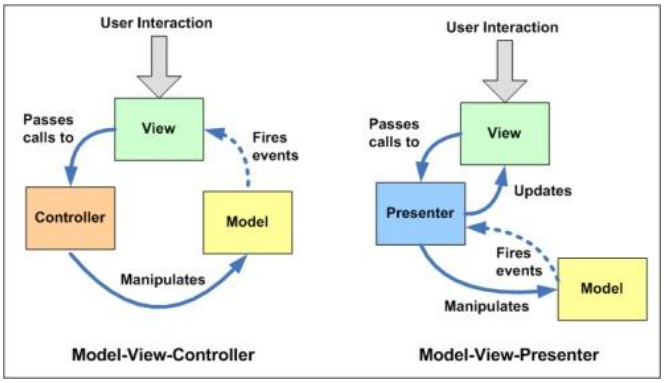


Figura 2.5: Pattern Model-View-Controller e Model-View-Presenter a confronto

Di seguito verranno illustrarti i ruoli delle tre entità presenti nel sistema.

Model

Il Model può essere visto come una sorta di interfaccia verso i dati, in cui sono definite tutte le operazioni della logica di business. Qualsiasi parte del programma che abbisogni di dati deve obbligatoriamente passare attraverso interfacce o funzioni definite per il Model dallo sviluppatore. In questo modo questa entità diviene una parte intercambiabile senza dover effettuare alcuna modifica al Presenter.

View

La View è quella parte dell'applicativo che permette all'utente finale di interagire con l'intero sistema, rappresentando lo strato di presentazione: di componenti di questo tipo ne possono essere presenti in qualsiasi numero. Le sue responsabilità sono limitate a comporre una propria visualizzazione, ricevere input dagli utenti e gestire gli eventi scatenati dall'esterno per mezzo di controlli grafici. Essa non implementa alcun tipo di logica di business, né interagisce direttamente col Model: le uniche sue invocazioni sono rivolte a metodi del Presenter. In questo senso è solo il Presenter che chiama le operazioni sul Model e che di conseguenza può impostare proprietà sulla View per riportare il risultato delle operazioni. Poiché la View viene progettata per essere totalmente intercambiabile, questo componente deve implementare un'interfaccia che definisca metodi e proprietà che poi tutte le View concrete hanno l'obbligo di implementare.

Presenter

Il Presenter è il mezzo tramite cui si attua il disaccoppiamento: deve contenere tutta la logica di business che deve essere implementata per rispondere ad un evento scatenato dall'utente. Tipicamente la View mantiene solamente metodi di gestione degli eventi e la logica per effettuare chiamate opportune verso funzioni del Presenter, in modo tale da sgravare chi sviluppa le View dal preoccuparsi di codice che non sia strettamente legato alla progettazione effettiva dell'interfaccia utente. Un altro compito del Presenter è fornire un accesso ai dati richiesti dal Model e formattarli nella giusta maniera in modo tale che la View possa visualizzarli senza alcun overhead. Questa entità non riferisce mai direttamente l'implementazione concreta della View, bensì si collega ad essa attraverso la sua interfaccia: questo permette agli sviluppatori di cambiare l'implementazione concreta della View senza riflettere modifiche sul Presenter, attuando un alto tasso di disaccoppiamento funzionale.

Il pattern in esame può essere declinato in due varianti, come mostrato in figura 2.6.

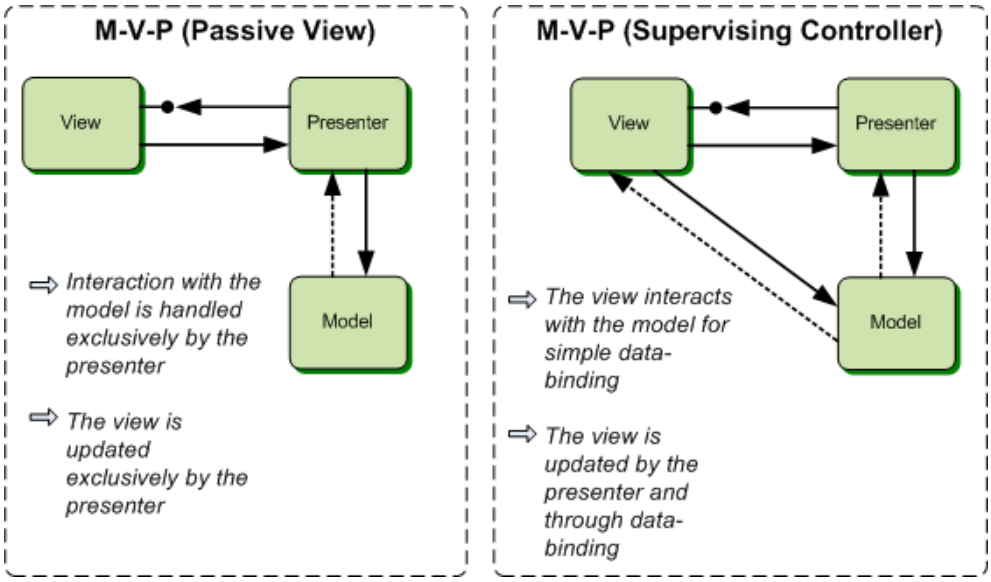


Figura 2.6: Declinazioni del pattern Model-View-Presenter

Passive View

In questo caso, la View non ha assolutamente conoscenza del Model, bensì espone proprietà per tutti i tipi di informazione che devono essere visualizzati. In questo senso il Presenter reperisce le informazioni dal Model ed aggiorna le proprietà di conseguenza.

Supervising Controller

In questa visione del pattern, la View ha conoscenza del Model ed è sua responsabilità instaurare un collegamento con esso. In questo modo la comunicazione tra Presenter e View si alleggerisce, a discapito però della possibilità di collaudo della coppia View-Presenter. Questa versione del pattern si avvicina molto a quella del Model-View-Controller, però la differenza è che il Presenter non ha tutte quelle responsabilità che invece possiede un Controller: qui infatti il suo compito è di informare il Model del compimento di quelle attività nella View che possono comportare un cambiamento dello stato. In più, il Presenter si occupa di passare un riferimento al Model in modo tale che esso possa interagire con la View direttamente ogni qual volta questi modifichi il suo stato.

2.3.3 Presentation Model

Questo pattern, proposto da Martin Fowler e ultimo uscito in ordine temporale, ha il suo punto focale nel componente Presentation Model: l'idea è quella di estrarre qualsiasi comportamento funzionale e lo stato dalla View ed inserirli nel Presentation Model. In questo modo:

- lo stato non è più contenuto nella View, bensì nel Presentation Model, che poi lo trasmetterà al Model per questioni di persistenza
- la logica di business per il coordinamento degli elementi grafici non sarà più contenuta nella View, ma nel Presentation Model

In questo pattern, illustrato in figura 2.7, i dati presenti nel Model sono arricchiti con un'informazione di stato che li mantiene sincronizzati con la View usando un modello Publish-Subscribe o un meccanismo di collegamento dei dati. Per aggiornare lo stato, la View scatena un evento che aggiorna sia lo stato nel Presentation Model, sia di conseguenza anche il proprio. Questo comportamento permette di inserire il codice legato alla sincronizzazione indifferentemente nella View o nel Presentation Model, e lascia allo sviluppatore la decisione di specificare chi tra la View ed il Presentation Model debba referenziare l'altro.

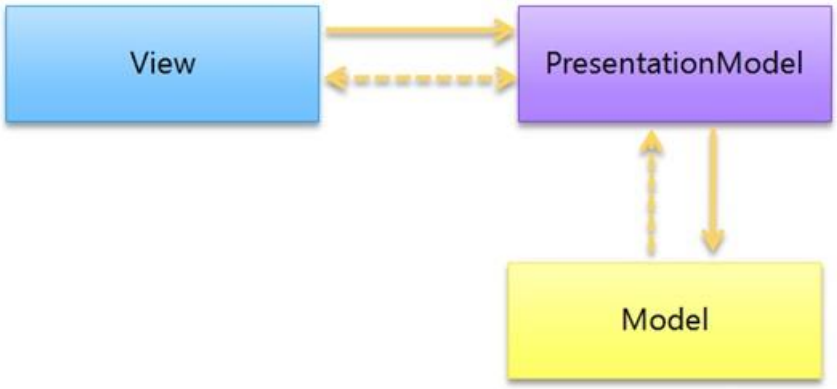


Figura 2.7: Pattern Presentation Model

Questo pattern ha trovato ampio utilizzo nel mondo Microsoft nella declinazione Model-View-ViewModel, in cui è di fatto utilizzato come una specializzazione idiomatica per le tecnologie WPF e Silverlight.

2.3.4 Comparazione

A questo punto è possibile effettuare un confronto tra i pattern illustrati, utilizzando la figura 2.8.

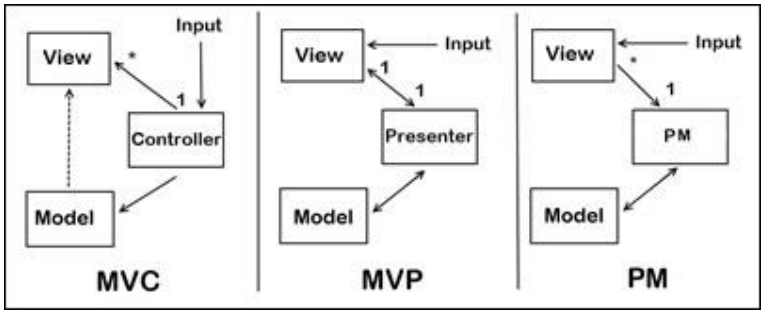


Figura 2.8: Comparativa tra i pattern

Per quanto riguarda le similarità tra MVC ed MVP, entrambi si concentrano sul concetto della separazione delle responsabilità tra i vari componenti di un sistema e promuovono l'idea di un basso accoppiamento funzionale tra le entità. Per quanto riguarda le differenze, invece, esse si concentrano sulla modalità di implementare il pattern:

- MVC
  - Il punto d'ingresso tra il sistema e l'utente esterno avviene tramite il Controller
  - Il Controller può essere condiviso da più View
  - E' il Controller che è responsabile di determinare quale delle View visualizzare: la freccia a senso unico tra i due indica che la View non è a conoscenza né ha riferimenti col Controller
- MVP
  - Il punto d'ingresso tra il sistema e l'utente esterno avviene tramite la View
  - View e Model sono più disaccoppiate tra loro: è compito del Presenter fornire un collegamento tra queste entità
  - Il collaudo dei vari componenti si rivela più semplice, anche grazie al fatto che l'interazione con la View avviene attraverso interfaccia
  - Tipicamente il mapping tra View e Presenter è di tipo 1:1: in questo senso con View di tipo complesso c'è la possibilità di trovarsi con un elevato numero di Presenter

Le differenze principali tra il pattern Model-View-Presenter ed il pattern Presentation Model invece sono:

- la cardinalità tra più View e l'unica entità di presentazione: questo comportamento, indicato dal senso unidirezionale dell'interazione, è possibile poiché la View mantiene un riferimento al Presentation Model, mentre quest'ultimo non ha conoscenza delle View che sono presenti
- mentre nel primo pattern la logica di funzionamento della View poteva essere presente entro la View stessa, nel secondo si è sicuri che ciò non possa accadere.

2.4 Strategie per il ciclo di sviluppo del software

La scomposizione dell'attività di realizzazione di un prodotto software in più sotto-attività fra loro coordinate e sincronizzate prende il nome di ciclo di sviluppo del software: nell'ambito dell'ingegneria del software, questa è una metodologia di sviluppo il cui esito finale è al contempo sia il prodotto software stesso sia tutta la documentazione ad esso associata. Questo modello di processo nasce col passaggio dello sviluppo software da attività artigianale ad approccio strutturato ed industriale, in cui il procedimento viene scomposto in fasi quali la pianificazione, il controllo e la documentazione, come avviene in tutti i settori ingegneristici. Questa forte necessità di organizzazione è stata causata sia da un costante aumento della complessità dei sistemi progettati sia dall'adozione di sempre più stringenti standard qualitativi.

2.4.1 Fasi costitutive

I modelli che è possibile utilizzare per descrivere un processo di sviluppo sono quasi tutti costituiti dalle stesse attività, che vengono elencate di seguito:

I. Analisi: è un'esplorazione preliminare del dominio applicativo in cui il prodotto andrà ad operare. Viene effettuata indagando sui requisiti e sulle caratteristiche che il prodotto dovrà possedere, sui suoi costi e su tutti gli aspetti legati alla sua realizzazione. Al termine di questa fase deve essere possibile definire nel modo più specifico possibile il problema da risolvere, elaborando un documento di specifiche funzionali utilizzando anche dati raccolti tramite incontri coi clienti.
II. Progettazione: in questa fase viene definita la struttura che dovrà assumere il software in funzione dei requisiti emersi nel punto precedente. In questo modo è possibile delineare la soluzione del problema ad un qualsiasi livello di dettaglio, tramite la stesura di un documento che definisca l'architettura di alto livello del sistema unitamente alle caratteristiche dei singoli componenti.
III. Implementazione: è la fase vera e propria di codifica del progetto, che consiste nella realizzazione di uno o più applicativi per il dominio del problema. Spesso si procede alla realizzazione della soluzione per parti, creando prima i singoli moduli, per poi integrarli a sviluppare il sistema complessivo.
IV. Collaudo: questa fase consiste nella verifica e nella validazione del prodotto realizzato nella fase precedente, nella misura in cui questo soddisfa il documento delle specifiche funzionali. Il collaudo avviene entro un certo ambiente, in modo tale da permettere agli sviluppatori di stressare in maniera approfondita il sistema: queste prove vengono solitamente condotte in maniera incrementale, esaminando prima il funzionamento dei singoli moduli e poi il sistema integrale. In caso di fallimento di questa fase, il prodotto deve tornare indietro agli sviluppatori per porre rimedio ai problemi rilevati.
V. Manutenzione: questo stadio si concentra in un momento successivo al rilascio, eseguendo attività finalizzate a modifiche. Queste possono essere provocate col fine di implementare variazioni legate ad errori, adattamento a nuovi scenari operativi o estensione di funzionalità. Ogni modificazione comporta la necessità di ripartire dalla fase precedente, al fine di verificare che le variazioni apportate non abbiano introdotto problemi nelle funzionalità preesistenti: in questo senso è facile capire come questa fase incida sui costi totali in una misura di circa il 60 %.

Contemporaneamente alle fasi di sviluppo di cui sopra, è buona norma produrre le documentazioni necessarie per le varie fasi, che saranno poi unificate alla fine dello sviluppo.

2.4.2 Modello a cascata

Questo modello è il primo in ordine di tempo che è stato proposto, ed è ampiamente utilizzato per la sua semplicità e flessibilità: le fasi sono tutte ordinate e sequenziali tra loro, in modo tale che il risultato di una sia l'ingresso per la successiva. Non sono previste retroazioni tra le fasi, come mostrato in figura 2.9: una volta che una è terminata, la sequenza non prevede retroazioni. In questa ottica questo modello può essere impiegato in progetti in cui i requisiti siano chiari e stabili fin dall'inizio, senza subire modificazioni con l'avanzare del lavoro.

Figura 2.9: Modello a cascata

2.4.3 Modello iterativo

In questo modello si fa ampio ricorso al feedback tra una fase di sviluppo e quelle successive: questo scambio continuo di informazioni nelle varie iterazioni non ha un limite predefinito, e può continuare fino al raggiungimento degli obiettivi voluti, come illustrato in figura 2.10. Questo approccio permette un approfondimento continuo della progettazione, ed è particolarmente indicato per applicazioni basate su tecnologie ad oggetti.

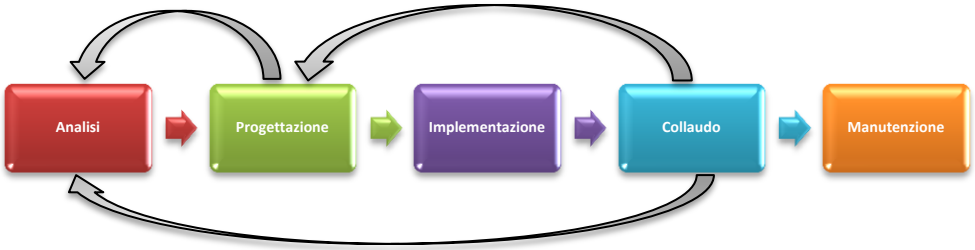


Figura 2.10: Modello iterativo

Il progetto è fortemente utilizzato per progetti di grandi dimensioni, complessi e con modifiche ai requisiti.

2.4.4 Modello a stadi

In questo modello il progetto è scomposto in componenti per un loro sviluppo individuale: ad ognuno di essi viene assegnata una funzionalità, in modo da potere poi definire le priorità di realizzazione. Gli elementi più importanti, o prerequisito di altri, vengono sviluppati nei primi stadi, mentre gli altri negli stadi più a valle, eventualmente anche in contemporanea se senza dipendenze reciproche, come mostrato in figura 2.11.

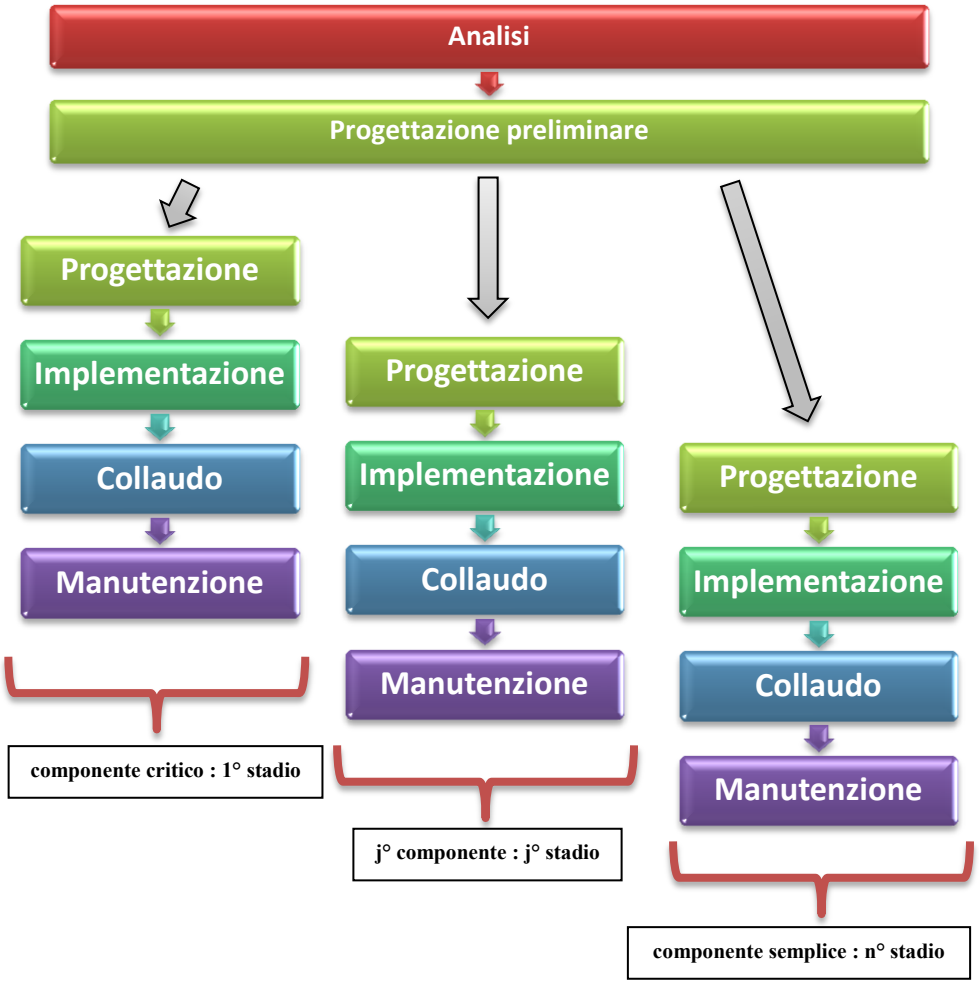


Figura 2.11: Modello a stadi

Anche per questo modello, l'intera struttura ed i requisiti devono essere stabili e definiti fin dall'inizio del progetto: esso è particolarmente adatto per applicativi che possono essere realizzati con rilasci successivi.

2.4.5 Modello prototipale

Con questo approccio si prevedono due cicli di sviluppo distinti: uno per il prototipo, ed uno per il prodotto finale, tra loro comunicanti al fine della realizzazione complessiva. I vari stadi sono evidentemente ciclici e proseguono finchè non si ottiene la precisione voluta nel sistema: non appena la fase legata al prototipo è completata, si passa a quella relativa al prodotto finale, come mostrato in figura 2.12.

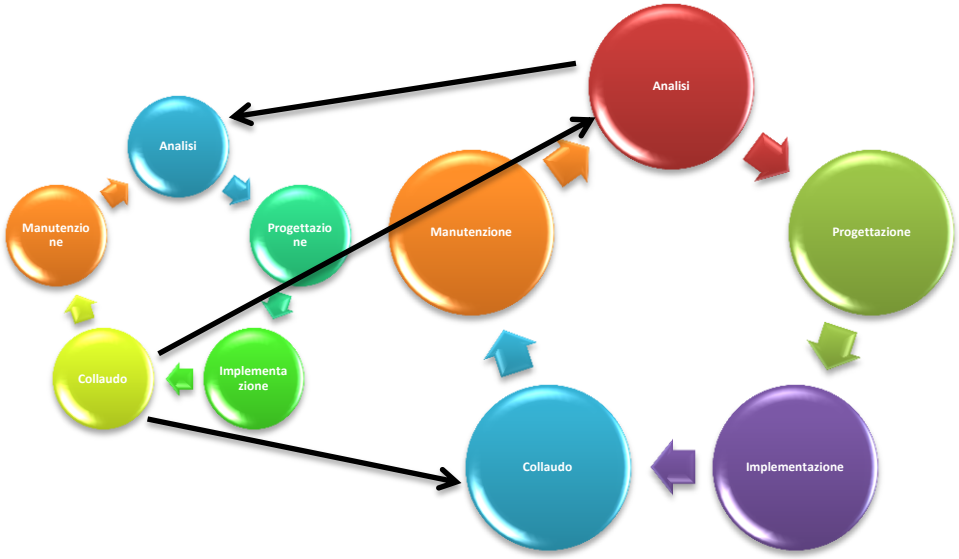


Figura 2.12: Modello prototipale. Prototipo (a sinistra) e prodotto finale (a destra)

Questo approccio è particolarmente adatto per investigare parti critiche e poco conosciute di un prodotto.

2.4.6 Modello a spirale

In questa strategia il ciclo di sviluppo è visto come la realizzazione di una serie di prototipi, ognuno più elaborato e raffinato del precedente, che poi sfociano nel prodotto finale. Alla fine di ciascun ciclo ogni prototipo è analizzato per fornire riscontri alla fase successiva, in cui sarà accresciuto nelle funzionalità, come illustrato in figura 2.13.

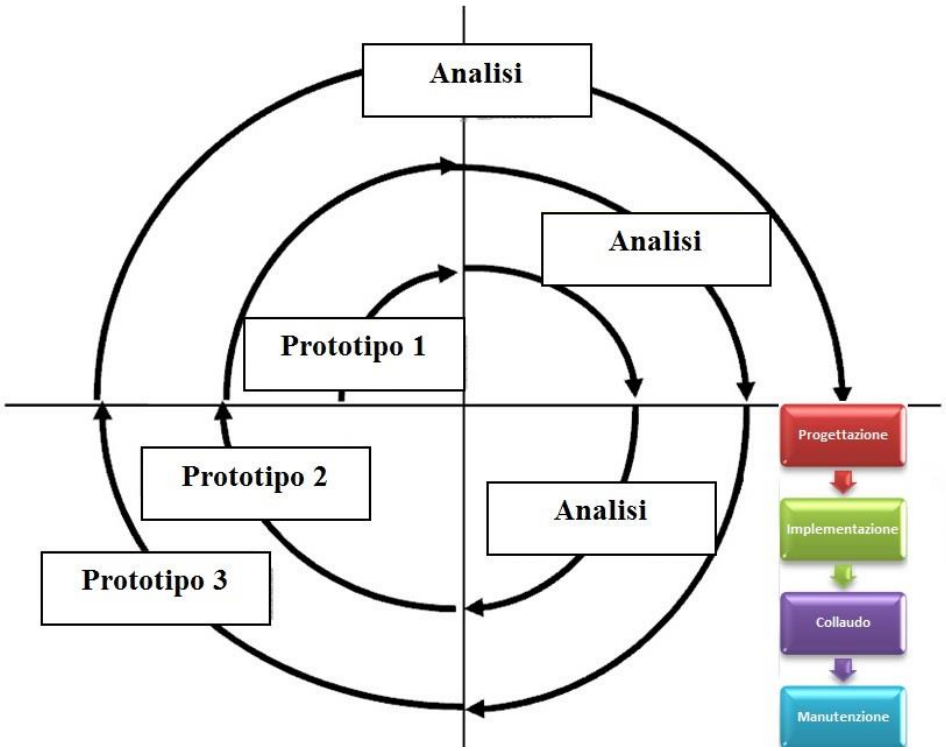


Figura 2.13: Modello a spirale

Questo modello viene adottato quando i requisiti sono poco stabili, oppure quando si utilizzano tecnologie nuove e poco conosciute.

2.4.7 Modello iterativo-incrementale

Questo modello estende e raffina quello iterativo visto in precedenza: lo sviluppo complessivo di un sistema è visto come una serie di cicli, dette iterazioni, ognuna in grado di sviluppare un insieme di funzionalità complete. Ad ogni iterazione si incrementano le funzionalità del prodotto sviluppato nel ciclo precedente: i primi due stadi del modello sono generali, cioè riguardano l'intero progetto e servono per pianificare le iterazioni successive, le quali sono invece cicli completi di sviluppo, come mostrato in figura 2.14.

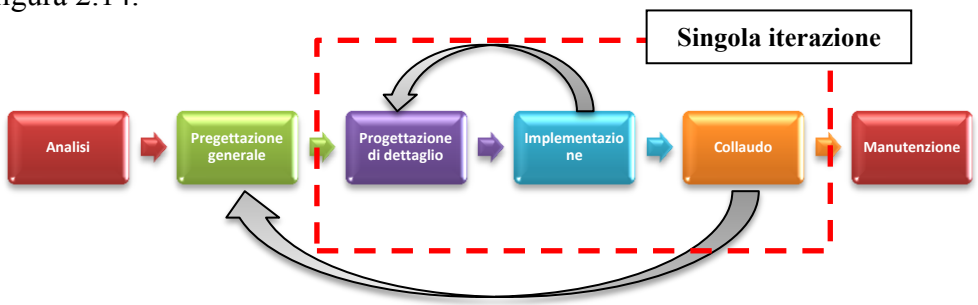


Figura 2.14: Modello iterativo-incrementale

In questo modo è possibile disporre di volta in volta di singole parti di prodotto autoconsistenti, cioè già utilizzabili dagli utenti: ad ogni completamento di una parte vengono accresciute le funzionalità del prodotto finale, che così evolve via via ad ogni iterazione. Questo modello è conveniente nei casi in cui il progetto sia di grandi dimensioni, complesso, con requisiti non noti a priori, ed in cui una serie di rilasci successivi agevolino gli sviluppatori nella consegna del prodotto finale.

2.4.8 Comparazione

Al fine di poter raffrontare al meglio i modelli di ciclo di sviluppo del software proposti per poi adottarne uno per la realizzazione dell'architettura in esame, ne viene proposta una comparativa sulla base delle loro caratteristiche, tramite la tabella 2.

	Dimesione progetto	Granularità rilascio	Definizione requisiti
A cascata	Qualsiasi	Singolo	Stabili
A stadi	Qualsiasi	Multiplo	Stabili
Prototipale	Qualsiasi	Qualsiasi	Non noti a priori
Iterativo	Grande	Qualsiasi	Instabili
A spirale	Qualsiasi	Qualsiasi	Instabili
Iterativo-incrementale	Grande	Multiplo	Non noti a priori

Tabella 2

Capitolo 3

Sviluppo di un'infrastruttura di supporto alla gestione automatica della configurazione per la macchina in esame

Al fine di progettare un'architettura software per lo scenario in esame, si è deciso di adottare come ciclo di sviluppo quello iterativo-incrementale, considerato che questo è il modello che meglio si adatta alle caratteristiche del sistema in esame. Infatti, analizzando nel dettaglio i tratti distintivi del sistema, si può notare come essi si adattino alla strategia scelta:

- Dimensioni del progetto: l'architettura da sviluppare copre un'ampia area di funzionalità del caso in esame. Esse possono essere riassunte come segue:
    - la gestione della configurazione dell'Hmi, che va dalla progettazione di una infrastruttura di supporto totalmente nuova all'instrumentazione degli elementi già presenti in Hmi per realizzare l'adattamento automatico
    - il trattamento delle informazioni nel vettore Opc destinato all'unità di controllo, al fine di comandare in maniera opportuna la macchina automatica
    - la configurazione e la gestione di tutta la parte relativa alla gestione dei rapporti informativi a seconda della configurazione selezionata, sia attraverso la modificazione automatica dei file relativi all'applicativo che produce i

report, sia attraverso un meccanismo di amministrazione e controllo dei file prodotti da quest'ultimo

- Granularità rilascio: dato che esiste la possibilità che l'infrastruttura da progettare sia composta da più applicativi, è utile poter effettuare rilasci multipli dei prodotti finali in modo tale da poterne valutare le funzionalità mano a mano che emergono nuove criticità
- Definizione dei requisiti: i requisiti del progetto in esame sono stati da subito di difficile individuazione, considerati sia i limiti della tecnologia attualmente in uso, sia la loro possibile variazione su un range molto vasto

Nel seguito, verranno illustrate nel dettaglio le fasi del ciclo di sviluppo scelto che hanno permesso di realizzare l'infrastruttura richiesta.

3.1 Realizzazione lato interfaccia grafica

La prima parte del progetto si è focalizzata sulla realizzazione di tutta quella porzione dell'infrastruttura che si occupa della gestione automatica dell'Hmi al cambio di configurazione prodotto sulla macchina. In aggiunta, questa parte deve intervenire sia sulla logica di gestione delle informazioni da mandare all'unità di controllo macchina via Opc, sia sulla configurazione di quelle informazioni che serviranno per la gestione dei rapporti informativi, dato che anche queste funzionalità dipendono dal prodotto che viene creato dalla macchina.

3.1.1 Fase di analisi

Il primo passo è stato quello di validare per il sistema corrente i requisiti generali esposti nel capitolo due. In questo senso si sono identificate una serie di metodologie di funzionamento idonee alle tecnologie attualmente in uso in azienda al fine di raggiungere i requisiti esposti, cercando di soddisfarli nell'ordine di importanza con cui sono stati esposti.

Metodologia 1: interfaccia eterogenea

In questo scenario è stato prevista la generazione di un numero di interfacce utente pari al numero di configurazioni di prodotto possibili, in modo tale che questo rapporto sia 1:1. L'idea è mutuata dalla soluzione adottata attualmente, cioè generare Hmi specifiche per ogni singola macchina (quindi per prodotto): in questo senso la cardinalità delle Hmi è pari al numero delle possibili configurazioni di produzione, come illustrato in figura 3.1.

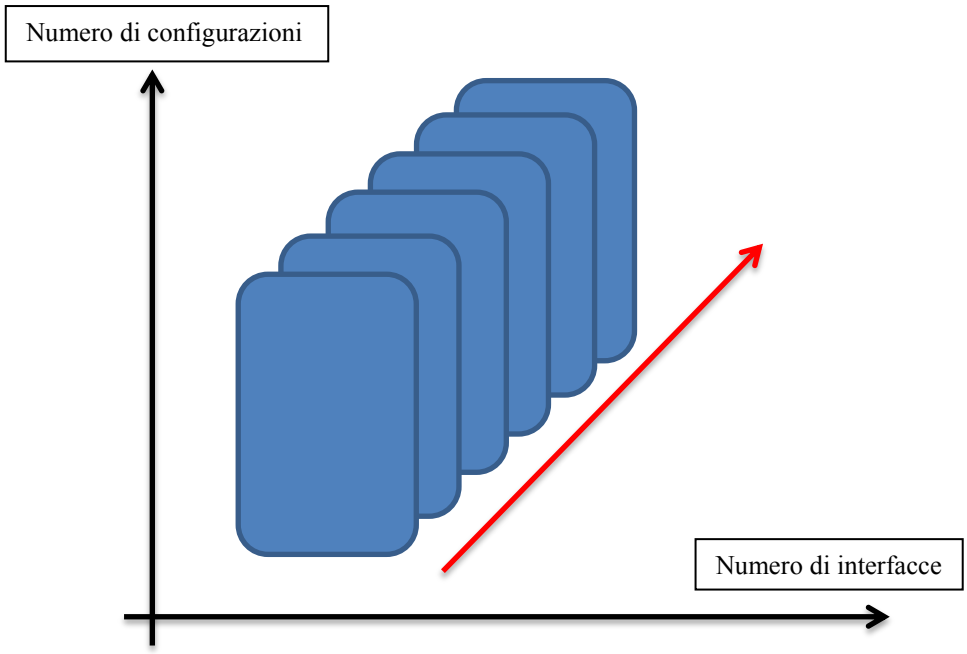


Figura 3.1: Interfacce eterogenee tra loro

Valutazione della soluzione:

- Pro
  - Layout dell'Hmi è estremamente specifico per ogni prodotto possibile
  - La soluzione garantisce la possibilità di estendere le funzionalità della macchina tramite l'aggiunta di gruppi di dosaggio futuri
  - Dato che l'interfaccia generata è specifica per ogni prodotto, non esiste la necessità di mantenere informazioni relative alle permutazioni delle varie configurazioni in base ai gruppi di dosaggio montati, al fine di gestire l'eventuale cambio di configurazione in maniera automatica
- Contro
  - Necessità di generare manualmente un Hmi per ogni specifica produzione per ogni macchina, a seconda dei gruppi

di dosaggio comprati dal cliente: questo comporta l'esplosione del numero di possibili interfacce da generare, unitamente al fatto che queste devono essere generate manualmente dall'azienda costruttrice
- Necessità di generazione di nuove Hmi su una stessa macchina per produzioni legate a gruppi di dosaggio non comprati dal cliente al momento dell'acquisto della macchina
- L'interfaccia si rivela sicuramente adattabile ai vari tipi di produzione, ma non c'è univocità tra i vari layout
- Bassa usabilità ed accessibilità utente causate dalla non omogeneità tra le differenti Hmi, che sono specifiche per prodotto
- Grande difficoltà per il mantenimento della documentazione a corredo, causa la generazione di una di esse per ogni Hmi
- Grande difficoltà nel collaudo, causa l'alta cardinalità delle interfacce, che devono essere testate tutte singolarmente

Metodologia 2: interfaccia unica

Questa metodologia prevede un'Hmi unica per tutti i vari tipi di prodotto: essa viene generata in un'unica volta e contiene gli elementi grafici di tutti i tipi di prodotti possibili, e poi personalizzata di volta in volta per singola configurazione di ogni macchina rendendo gli elementi grafici di granularità più piccola non/visibili, settando singolarmente le relative proprietà, come mostrato in figura 3.2.

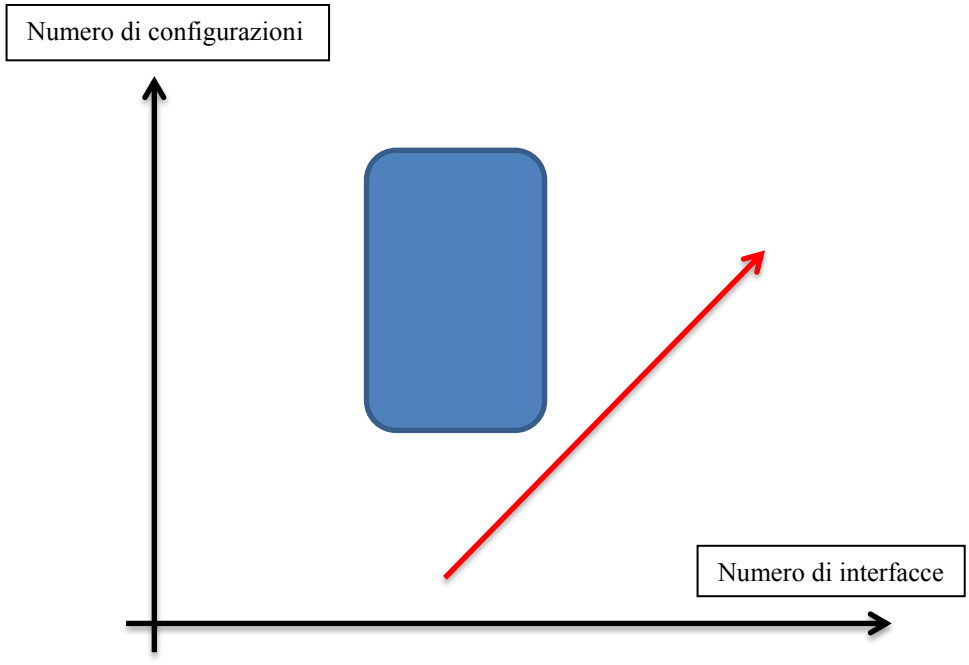


Figura 3.2: Interfacce omogenee tra loro

Valutazione della soluzione:

- Pro
    - Il layout dell'Hmi diviene sempre uguale per ogni configurazione di prodotto: si raggiunge così un'univocità d'interfaccia tra differenti produzioni
    - L'omogeneità della soluzione proposta consente di raggiungere un'alta usabilità ed accessibilità utente
    - E' sempre prevista l'espandibilità futura per nuovi gruppi di dosaggio
    - La fase di generazione dell'Hmi avviene una sola volta nella vita della macchina
    - L'omogeneità della soluzione porta a facilità di collaudo e manutenzione dell'interfaccia, che vengono fatte una sola volta in maniera completa
- Contro
    - Esiste la possibilità, da valutare nella fase di collaudo, che nella versione finale l'Hmi sia troppo carica di elementi grafici, non consentendo quindi una loro corretta visualizzazione, causa limiti della tecnologia impiegata
    - Esiste la necessità di mantenere una grande mole di informazioni relative alle possibili configurazioni dovute alle permutazioni dei gruppi di dosaggio, al fine di selezionare gli elementi grafici relativi alla specifica configurazione di produzione

Metodologia 3: interfaccia ibrida

In questa soluzione, ipotizzando di partire da un'Hmi unica, si è immaginato di ottenere per derivazione un'Hmi specifica relativa ad una configurazione basata su una serie di stazioni di lavoro categorizzate come di primo livello, cioè qualificanti per una certa produzione, come mostrato in figura 3.3. La teoria che qui si sviluppa è quella per cui alcune stazioni di lavoro siano in esclusione con altre: in questo senso, individuando le configurazioni di lavoro per stazioni mutuamente esclusive, è ancora possibile adottare la tecnica di interfaccia unica, ma senza appesantirla troppo con elementi di Hmi che si sa già a priori che non potranno mai essere inclusi in una certa configurazione che un cliente compra, causa l'esclusione reciproca. Con questo sistema, si ottiene un duplice effetto:

- si contiene la molteplicità tra differenti Hmi, poiché le configurazioni di primo livello con stazioni di lavoro incompatibili tra loro sono presenti in numero limitato

- ogni configurazione è pensata per comprendere solamente le stazioni potenzialmente compatibili con essa, riducendo così la complessità di creazione e visualizzazione dell'Hmi.

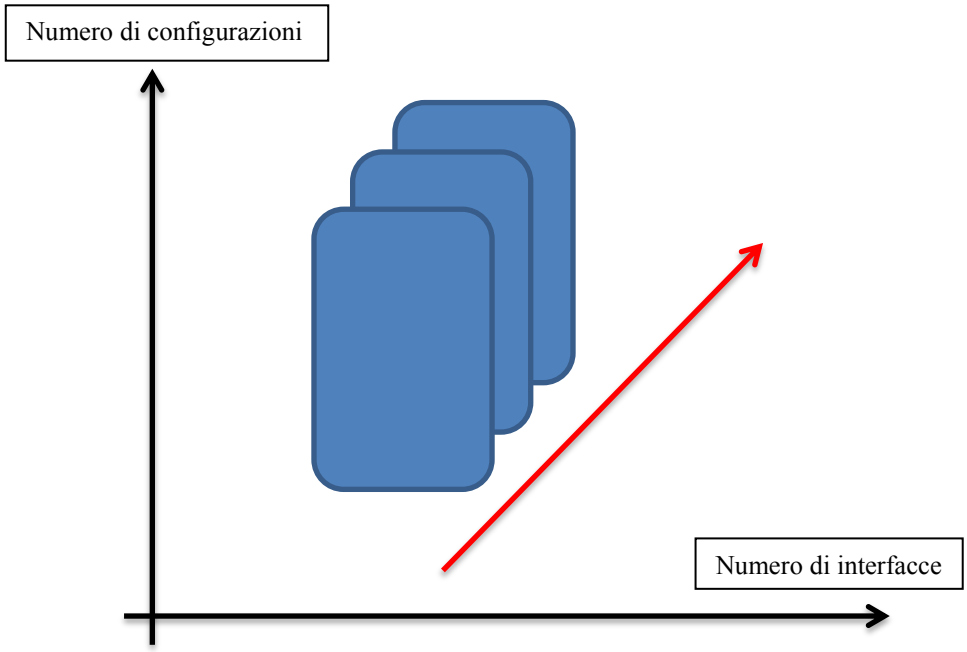


Figura 3.3: Interfacce ibride

Valutazione della soluzione:

- Pro
  - Si riesce ad ottenere univocità nella visualizzazione dell'interfaccia, limitatamente a configurazioni relative a stazioni di primo livello, conseguendo al contempo un layout di visualizzazione specifico per tipo di produzione, il quale non varia a seguito di montaggio di stazioni di lavoro a lui compatibili
  - L'Hmi è a questo modo riconfigurabile, limitatamente al montaggio di stazioni di lavoro compatibili
  - Il numero di informazioni da mantenere relativamente alle permutazioni delle differenti configurazioni si riduce, poiché si escludono a priori tutte quelle stazioni di lavoro non compatibili con la configurazione primaria
  - Buona usabilità ed accessibilità utente dell'Hmi: all'atto della riconfigurazione relativa all'inserimento di stazioni di lavoro compatibili, il layout non cambia perché gli elementi delle nuove stazioni sono già previsti nella fase di generazione dell'interfaccia
  - Collaudo e documentazione sono agevolate, poiché la molteplicità delle interfacce è di fatto limitata
  - La fase di generazione dell'Hmi viene fatta una sola volta

- Rimane la possibilità di estensione relativamente a nuove stazioni di lavoro compatibili con la configurazione primaria, tramite l'aggiunta di informazioni sulle permutazioni delle configurazioni

- Contro
  - Dato che questo modello discende da quello dell'interfaccia unica, rimane la possibilità, da valutare nella fase di collaudo, che nella versione finale l'Hmi sia troppo carica di elementi grafici, non consentendo quindi una loro corretta visualizzazione, causa limiti della tecnologia impiegata
  - Se il numero di stazioni di lavoro di primo livello accresce in maniera rilevante, esiste il pericolo che il modello collassi nella prima metodologia esposta, cioè di un'Hmi specifica per ogni configurazione, con tutte le ricadute del caso specifico

Comparativa

Al fine di valutare quale delle metodologie ideate si adatta meglio al sistema in uso, prima di passare in progettazione, si fa ricorso ai requisiti esposti in precedenza.

	Metodologia 1	Metodologia 2	Metodologia 3
Flessibilità operativa	NO	SI	SI
Usabilità / Accessibilità utente	NO	SI	SI
Manutentibilità	NO	SI	SI
Semplicità / Velocità Generazione-Collaudo-Documentazione	NO	SI	SI
Sicurezza / Protezione	SI	SI	SI
Appesantimento Layout Hmi	NO	SI	NO

Tabella 3

Come si nota dalla tabella 3, la metodologia che meglio si adatta ai requisiti che sono stati individuati in precedenza è la numero tre: essa infatti, oltre ad ereditare tutte le qualità della soluzione numero due da cui discende, riesce a risolvere il possibile problema dell'appesantimento grafico dell'Hmi in visualizzazione. Per tali motivazioni, viene scelta questa tecnica come piattaforma di partenza per la fase successiva, cioè quella di progettazione: nel caso della macchina in esame, l'interfaccia sarà dunque omogenea tra differenti produzioni, poiché la tipologia di prodotto creato è unica. La molteplicità del modello viene invece raggiunta applicando la metodologia a lavorazioni differenti, cioè altri modelli di macchina automatica.

3.1.2 Fase di progettazione

Architettura

In questa fase, adottando la metodologia individuata nel paragrafo precedente, viene specializzata l'architettura di presentazione che risulta più idonea per il dominio di riferimento. In questo senso, avendo già analizzato in precedenza i requisiti e le tecnologie impiegate in azienda, l'architettura che meglio si adatta allo scenario esposto è quella del Model-View-Presenter nella declinazione Passive View per i seguenti motivi:

- adottando un meccanismo ad interfaccia unica per tipologia, la molteplicità View-Presenter è 1:1
- imponendo i requisiti una forte semplicità di collaudo, questo pattern pone le condizioni ideali poiché impone massimo disaccoppiamento funzionale tra le entità, anche se a discapito dell'incremento di comunicazione tra View e Presenter, che comunque non è una questione critica per i requisiti

L'architettura MVP va però rivista ed estesa rispetto al modello classico esposto in precedenza: essa infatti va ridefinita e qualificata per il dominio in esame, come illustrato in figura 3.4.

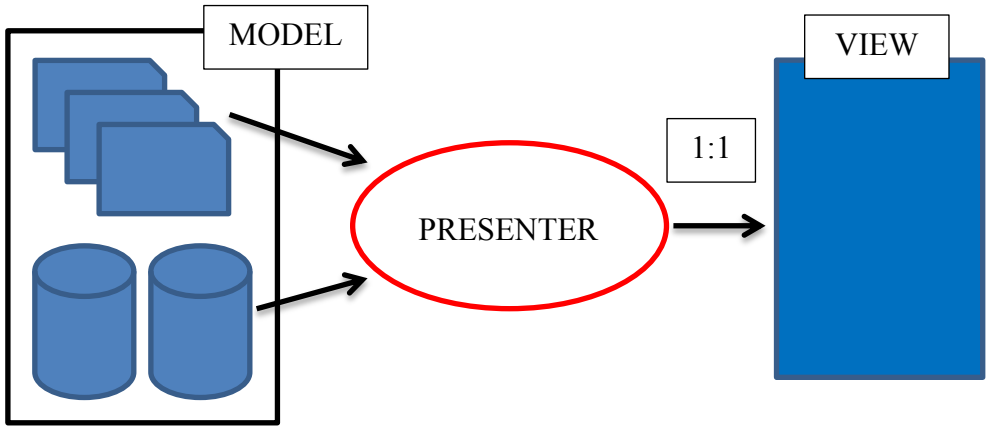


Figura 3.4: Architettura MVP specializzata per il dominio di riferimento

In questa visione il pattern assume un comportamento di tipo statico, dal momento che il Presenter viene ideato come uno o più componenti che non operano a run-time per ridisegnare l'interfaccia in qualsiasi momento, bensì operano in uno specifico punto temporale prima dell'avvio dell'unica View omogenea, cioè del software che si occupa realmente dell'interfacciamento con l'operatore in macchina. A questo modo, le competenze e le

responsabilità dell'architettura sono state ridistribuite per meglio adattarsi alle tecnologie disponibili nel seguente modo:

- Model
    - non è un base di dati unica, bensì viene scomposto in più file per meglio dividere le informazioni secondo categorie predefinite
    - più che vere e proprie basi di dati, essi sono file di instrumentazione leggeri e facilmente ispezionabili anche per via umana: essi serviranno per creare una corrispondenza tra una certa configurazione di produzione e gli elementi grafici da visualizzare
    - questi file concorrono insieme ad istruire il Presenter
- View
    - è unica poiché omogenea, secondo la metodologia individuata in precedenza
    - effettua rendering di elementi grafici e dati contenuti nel Model, sfociando nel dominio di riferimento nell'applicativo Xima
    - ha il compito di raccogliere e gestire le interazioni con l'utente, mentre nel pattern classico questo è uno dei compiti del Presenter
- Presenter
    - non gestisce gli eventi utente, poiché non c'è dinamicità di composizione a run-time, bensì ha il compito di creare l'interfaccia a partire da una serie di standard e regole precedentemente definite
    - deve riadattare l'interfaccia ad ogni avvio in maniera automatica, senza intervento manuale
    - secondo questi presupposti l'aggiornamento della View avviene passivamente (cioè staticamente in una fase precedente al caricamento)

Soluzione software a supporto dell'architettura

Dopo aver individuato un'architettura di riferimento, la fase successiva è quella di ideare due soluzioni software che potessero trattare le responsabilità attribuite al Presenter.

Le basi di partenza su cui effettuare i ragionamenti sono due:

- le pagine in cui è divisa l'interfaccia grafica vengono definite a priori e sono statiche: uno o più applicativi generano l'Hmi specifica per una determinata configurazione in modo preliminare dell'avvio di Xima, secondo le informazioni contenute nel Model

- partendo dal presupposto di un'interfaccia unica contenete gli elementi grafici corrispondenti a tutti i tipi di prodotto relativamente ad una certa tipologia, entrambe le soluzioni sono accumunate dalla caratteristica di lavorare per eliminazione di quelle parti di interfaccia superflue e non necessarie per una certa configurazione corrente.

In questo modo, analizzando gli elementi grafici su cui esso deve intervenire, cioè le pagine dell'applicativo Xima, si sono immaginati due scenari:

I. pagine fisse a popolamento proattivo in eliminazione
II. pagine fisse a popolamento reattivo in eliminazione

Nel primo scenario, il Presenter, a seconda della configurazione inserita, identifica in una base di dati le modifiche da riportare nelle pagine di Xima.

Nel secondo scenario, invece, sfruttando il fatto che le pagine di Xima sono dinamiche (cioè se a run-time cambia il contenuto, l'applicativo ridisegna l'interfaccia in automatico), si è ideata la seguente soluzione: utilizzare questo meccanismo ad eventi intrinseco dell'Hmi per implementare degli handler di tipo software che vadano a variare il contenuto dei file delle pagine di visualizzazione nel momento di pressione di alcuni pulsanti definiti, a seconda di una certa configurazione scelta.

Valutando le due soluzioni esposte, si giunge alla conclusione che quella reattiva è la meno impiegabile perché snatura il modello ad eventi del software, aggiungendo ad esso responsabilità per cui non è stato previsto. Infatti, il meccanismo ad eventi in questa tecnologia è stato pensato per comunicare dei parametri all'unità di controllo, e non per gestire responsabilità a livello di interfaccia grafica. In questo senso, quella proattiva si rivela l'unica utilizzabile nel dominio al fine di soddisfare i requisiti richiesti.

Tecnologie per la soluzione individuata

Al fine di arrivare ad implementare la soluzione individuata nello stadio precedente, sono state progettate due modalità d'intervento sul software Xima, esposte in figura 3.5.

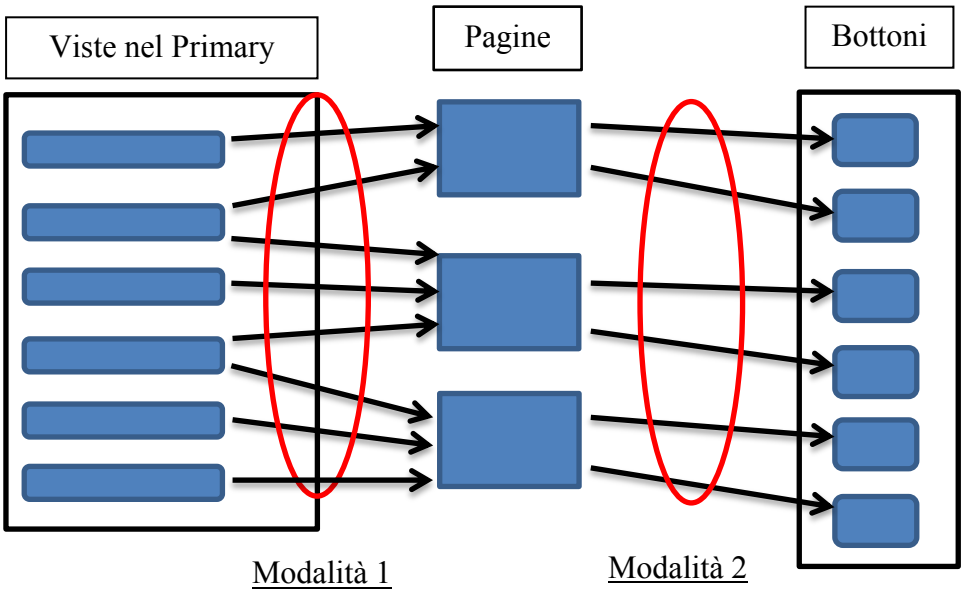


Figura 3.5: Tecnologie per la soluzione scelta a confronto

La modalità numero uno prevede di intervenire nell'associazione tra le view contenute nel primary e le pagine di visualizzazione, producendo in tal modo un numero di view pari al numero di possibili configurazioni ma mantenendo al contempo fisse le associazioni tra pagine e pulsanti dei menù.

- Pro
  - le pagine vengono modificate andando ad impostare solo le view relative ad una certa configurazione. In questo modo la loro cardinalità rimane fissa: non si ha il problema di una eventuale esplosione del loro numero in caso di espansioni future
  - il numero dei pulsanti di navigazione nei vari menù rimane fisso
- Contro
  - diviene impossibile comporre l'interfaccia unica omogenea per tipologia di produzione, causa limite della tecnologia impiegata: ogni pagina può contenere una sola view
  - c'è un incremento della cardinalità delle view nel primary: infatti vanno inserite tutte le view per ogni possibile configurazione, e di conseguenza anche tutti i relativi elementi. Il risultato è un'esplosione della dimensione del

primary, con la conseguenza di un aumento di difficoltà nella sua gestione e mantenimento

La modalità numero due opera invece sull'associazione tra le pagine ed i menù di vario livello, producendo un numero di pagine pari al numero di possibili configurazioni, mantenendo al contempo fisse le associazioni tra le pagine e le view.

- Pro
    - La dimensione del primary rimane costante, agevolandone la gestione ed il mantenimento
    - Il numero dei pulsanti di navigazione nei menù di vario livello rimane fisso
- Contro
    - Il meccanismo prevede di modificare i menù andando ad impostare le pagine volute: di conseguenza si ha un aumento della cardinalità delle pagine. Ciò può comportare una possibile difficoltà nella loro gestione dovuta a future espansioni.

Una terza modalità di lavoro prevedrebbe di intervenire sul rapporto tra view e menù, in modo tale da avere tra questi un'associazione 1:1: questo sistema non viene però preso in considerazione causa limite della tecnologia Xima, il quale non prevede di avere più di dieci pulsanti su interfaccia a qualsiasi livello di menù.

La scelta ricade perciò sulla tecnologia numero due, la quale è l'unica che consente di adottare le soluzioni individuate in precedenza sull'applicativo in uso attualmente sulle macchine automatiche. La tecnica adottata per realizzare questa metodologia prevede di rimuovere i singoli elementi dell'interfaccia grafica dai menù di vario livello che non appartengono ad una specifica configurazione di lavoro. Questa rimozione può essere di due tipi:

- fisica: gli elementi vengono rimossi in interfaccia tramite cancellazione effettiva sui corrispondenti file
- virtuale: gli elementi non coerenti con una certa configurazione vengono commentati all'interno dei file e resi non visibili in interfaccia, inibendone quindi la visualizzazione all'operatore

Al fine di rispettare l'obiettivo di avere un'interfaccia riconfigurabile, quindi pienamente modificabile in tutte le sue parti, senza introdurre ulteriore complessità nel modello, è opportuno adottare una rimozione selettiva di elementi per via virtuale, poiché deve essere possibile ad esempio abilitare un elemento grafico disabilitato in precedenza, senza per questo dover attingere a file vergini.

Identificazione delle informazioni da gestire

L'ultimo passo dello stadio di progettazione è l'individuazione delle informazioni che devono essere gestire dal Presenter, al fine di poterle trattare in maniera automatica per la riconfigurazione dell'Hmi. In questo senso, sono state definiti i seguenti dati:

- Elemento: parametro di macchina univoco di singola granularità che individua uno specifico componente fisico, il cui rapporto è 1:1
- Gruppo: definito come insieme di elementi, rappresenta una struttura logica che è mappata 1:1 con una corrispondente struttura fisica in macchina, es. un'unità di dosaggio
- Configurazione: definita come un insieme di gruppi, serve a rappresentare l'informazione sulla conformazione operativa della macchina specifica per una determinata produzione

A corredo di queste informazioni principali, vanno definite anche altri dati:

- Il tipo di dato di un certo elemento, in modo tale da poter associare, tramite un'enumerazione, un valore parlante ad un valore in grado di essere trattato da una macchina
- Una keyword associata in maniera univoca ad un determinato elemento, al fine di poter definire una semantica basata su regole
- Un parametro che indichi la presenza o meno di un certo elemento nel vettore Opc, al fine di instrumentare in maniera corretta l'unità di controllo
- Una lista dei file su cui il Presenter dovrà intervenire, per creare una sorte di base di dati su cui si effettueranno le modificazioni

Tutte queste informazioni fanno parte di una base di dati con due compiti specifici:

- Memorizzazione ed archiviazione dei dati
- Comunicazione tra i vari livelli presenti

Per attuare queste funzionalità, si è scelto di utilizzare la codifica dei dati su file xml, in cui il modello logico di archiviazione, cioè il documento xmlin questo caso coincide con l'archiviazione fisica: in questo modo si ottiene una collezione di dati raggruppati in maniera gerarchica. La soluzione individuata per il sistema in esame è l'adozione di un'architettura basata su file in formato xml che, oltre che per scambi di dati su web, si sta diffondendo per la definizione di vere e proprie basi di dati: esso ha una struttura gerarchica e non relazionale, ed è quindi paragonabile ad una base di dati gerarchica.

Questa soluzione garantisce vantaggi come:

- non necessità di implementare un'architettura client-server per i database
- gestione facilitata di una mole di informazioni non troppo elevata e dalla struttura semplificata
- beneficio di avere una base di dati gerarchica senza lo svantaggio di dover passare da un applicativo specifico per la loro gestione
- non necessità di un accesso in multiutenza
- nessun overhead prestazionale dovuto all'installazione di un DBMS
- essere in formato human-readable, al fine di agevolare eventuali modifiche da parte di tecnici/operatori

Definizione di una semantica attraverso regole booleane

L'introduzione della tecnologia di modifica dell'Hmi progettata in precedenza attraverso il commento di quelle parti di interfaccia non relative ad una determinata configurazione prevede la definizione di una semantica a struttura prefissata, dotata di alcune regole, in grado di essere interpretata in maniera automatica dal Presenter. In questo senso, si è scelto di adottare l'algebra booleana, che è un'algebra astratta che opera con i soli valori di verità 0 e 1. Questo tipo di algebra permette di definire gli operatori logici AND (prodotto logico), OR (somma logica) e NOT (negazione), la cui combinazione permette di sviluppare qualsiasi altra funzione booleana: per questo motivo tali operazioni costituiscono un insieme funzionalmente completo. Essa consente inoltre di trattare in termini esclusivamente algebrici le operazioni insiemistiche dell'intersezione, dell'unione e della complementazione, oltre a questioni riguardanti singoli bit 0 e 1, sequenze binarie, matrici binarie e diverse altre funzioni binarie.

In questo modo sono state progettate espressioni per lo scenario di riferimento col seguente formato:

```
@KEYWORD OPERATOR_RULE [ ELEMENT(VALUE) | … | ELEMENT(VALUE) ]
```

Il tag "`@KEYWORD`" serve come marcatore al Presenter per identificare l'inizio di una sezione di interfaccia che sarà sotto valutazione della regola subito successiva.

Con il termine "*`OPERATOR`*`_RULE`" si indica quale dei tre operatori dell'algebra booleana deve essere utilizzato per la valutazione dell'espressione. I casi possono quindi essere:

- AND: per definire il prodotto logico
- OR: per definire la somma logica
- !: per definire la complementazione, che viene espressa innanzi al singolo elemento, e non a livello di regola

Esistono poi una serie di termini nel formato "*ELEMENT(VALUE)*", i quali costituiscono i singoli elementi di macchina con i relativi valori, specificati nel corrispondente enumerativo. E' stato poi previsto di inserire una serie di elementi particolari, per prevedere tutti quei casi in cui una determinata porzione d'interfaccia debba essere sempre presente o sempre assente, senza per questo dover specificare una complicata espressione logica: questi vanno sotto la forma "`WILDKEY(TRUE|FALSE)`", in modo tale che il Presenter possa riconoscere facilmente se quella parte di interfaccia debba essere mantenuta a prescindere da una specifica configurazione di produzione o meno.

Il ragionamento che si è fatto progettando questo sistema è che, a seconda del valore relativo ad un certo elemento e specificato nell'espressione, ognuno di essi può valere vero o falso. Tale valore vai poi in composizione con gli altri elementi elencati per concorrere al calcolo totale dell'espressione, in un modo che varia a seconda della regola specificata.

3.1.3 Fase di implementazione

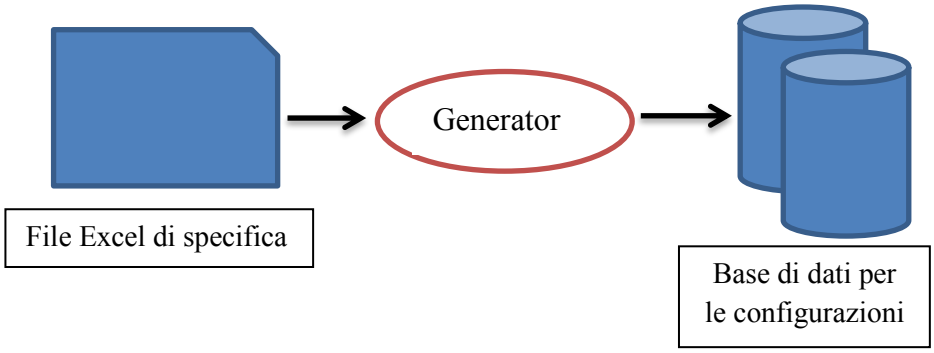
Flusso di esecuzione

Il presupposto da cui si parte è quello per cui si dà per acquisita la parte di instrumentazione di tutti i file relativi all'interfaccia unica totale: in questo senso, in una fase preliminare e in un'unica soluzione, vengono decorati con espressioni booleane per via manuale tutte quelle parti dell'interfaccia che sono soggette a modificazioni a seconda dei componenti fisici montati in macchina. Tutte le altre parti invariabili possono essere invece legate a due politiche:

- Non essere decorate, lasciandole in questo modo immodificate rispetto all'interfaccia vergine
- Essere decorate con un particolare elemento che definisca se quella parte debba essere sempre presente o meno

In questa ottica, il funzionamento del Presenter è stato diviso in due livelli, eseguiti in tempi differenti: in ognuno di essi è stata progettata una specifica applicazione che nel funzionamento complessivo costituisce l'intera infrastruttura di supporto alla riconfigurazione, come illustrato in figura 3.6.

Livello I:



Livello II:

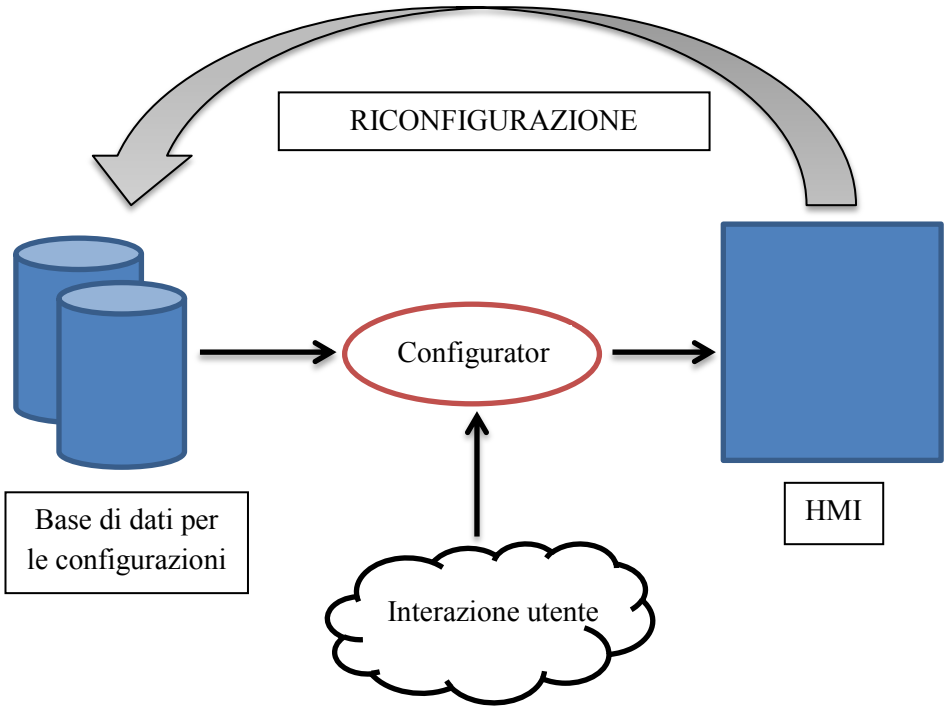


Figura 3.6: Divisione a livelli del funzionamento del Presenter

Nel primo livello vengono specificate, attraverso un file excel, tutte quelle informazioni relative alle configurazioni comprate dal cliente a seconda dei componenti fisicamente montati in macchina. Nella composizione di questa serie di informazioni è stato previsto il fatto che le configurazioni specificate non coincidano con tutte le permutazioni possibili legate ai componenti di macchina comprati, bensì sia possibile specificare liberamente un sottoinsieme di queste perché non è detto che un cliente sia interessato a tutti i tipi di produzioni legati ai componenti comprati. Un'altra considerazione è stata fatta modellando la composizione delle informazioni: poiché è previsto nei requisiti che la piattaforma implementata sia estendibile in futuro con nuovi componenti, a qualsiasi livello, è stata definita una struttura dei dati che riflette una composizione gerarchica delle

informazioni, in modo tale che sia facile un domani adattare questa infrastruttura per nuovi componenti od addirittura nuove macchine. In questo senso non esistono vincoli di composizione tra i vari elementi, i gruppi di appartenenza e le configurazioni, poiché queste considerazioni sono lasciate alle valutazioni dei tecnici sulla base del comportamento meccanico dei componenti: l'unica condizione da rispettare è che gli elementi inclusi in un foglio di specifica devono essere unici relativamente ai vari gruppi definiti. Questo livello di configurazione è previsto che venga effettuato internamente dall'azienda produttrice, in modo tale da rispondere al punto cinque dei requisiti sulla sicurezza e protezione nel funzionamento della macchina automatica: solo tecnici qualificati possono imporre il comportamento funzionale della macchina, al fine di evitare che personale esterno o poco esperto possa portare a malfunzionamenti o comportamenti pericolosi. Al termine della specifica su file excel di queste informazioni, l'applicativo chiamato Generator crea una base di dati in formato xml, i quali saranno poi utilizzati nel livello successivo per andare a comporre a tutti gli effetti l'interfaccia secondo una determinata configurazione di produzione.

Nel secondo livello, invece, viene eseguito sulla macchina automatica un applicativo chiamato Configurator che, alimentato dalla base di dati prodotta nella fase precedente, va ad adattare l'interfaccia in tutte le sue parti secondo una configurazione di produzione impostata dall'operatore per via manuale.

Il vantaggio di questo doppio livello di configurazione è che, nel caso un cliente compri nuovi componenti di macchina per diversificare la produzione, dal punto di vista Hmi non solo si troverà un layout ancora familiare, ma sarà semplificata anche la parte di adattamento dei nuovi componenti con la precedente interfaccia: infatti basterà che l'azienda costruttrice rigeneri la base di dati relativa alle nuove configurazioni comprate perchè il cliente possa immediatamente cominciare ad utilizzarle per la produzione.

Realizzazione

Entrambe le applicazioni che verranno illustrate in seguito sono state realizzate per mezzo del linguaggio C#, in tecnologia .NET, tramite l'editor Visual Studio.

Generator

L'applicazione Generator, dotata di interfaccia grafica per offrire un layout visivo all'utente, riflette l'impostazione della modellazione delle informazioni effettuata a partire dal documento excel, ed ha il compito di comporre una base di dati in formato xml per il livello di configurazione successivo. Al fine di realizzare questi obiettivi, essa include le classi Collection, HmiFile, Configuration, Group ed Element che definiscono attributi e metodi per gestire le relative informazioni inserite nei corrispondenti fogli del file excel di specifica. Queste classi vengono poi coordinate dalla classe principale Form1, di tipo Form, che richiama nella maniera opportuna i metodi delle altre classi, gestendone informazioni e tempistiche. In figura 3.7 viene mostrata l'immagine che rappresenta la modellazione relazionale effettuata per il sistema in esame. L'associazione logica tra le istanze degli oggetti raffigurati è quella di aggregazione in molteplicità uno a molti, ed indica che la distruzione dell'insieme contenitore non implica l'eliminazione dei singoli oggetti contenuti, poiché essi continuerebbero ad esistere di vita propria.

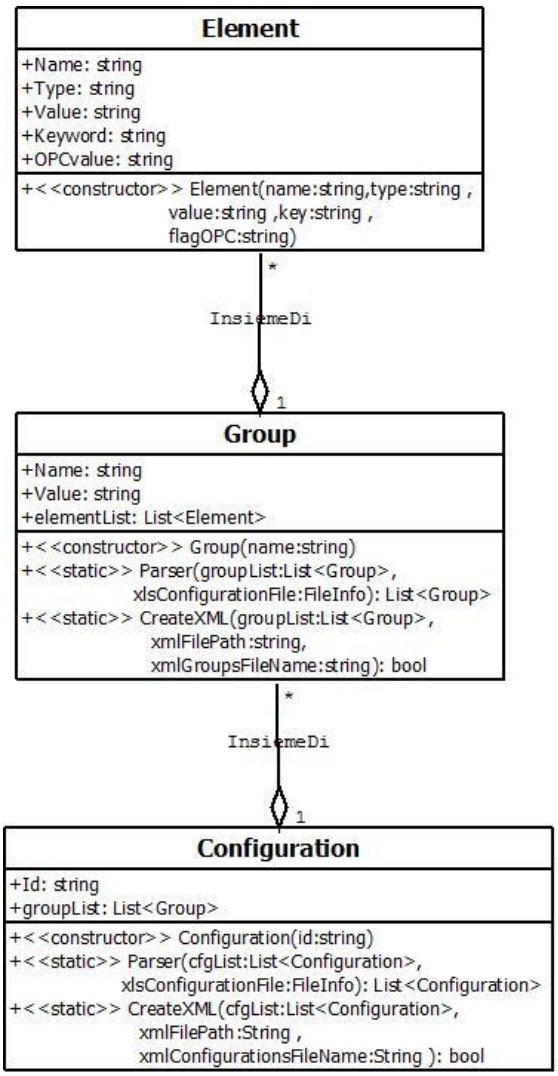


Figura 3.7: Relazione di aggregazione tra le classi rappresentate

L'idea con cui è stata realizzata l'applicazione è quella per cui, tramite l'interfaccia grafica di cui è dotata, all'utente venga permesso di specificare in ingresso due informazioni:

- il percorso del file excel di specifica della configurazione di macchina
- la directory in cui dovranno essere ospitati i file xml relativi alla base di dati che viene prodotta

Una volta inserite queste informazioni, la pressione del tasto Start Exporting scatena l'evento di processamento del file excel precedentemente immesso: esso viene parsato, le informazioni contenute processate ed incapsulate in oggetti, ed infine vengono prodotti, a partire da questi, una serie di file xml rappresentati la base di dati necessaria al successivo livello di configurazione per operare. L'elaborazione del file excel è stata effettuata utilizzando la libreria indipendente chiamata EPPlus, che utilizza il formato aperto Open Office Xml, col fine di essere il più possibile compatibili col formato xml ed al contempo di rendere il processamento del file il più semplice ed efficiente possibile. Nel seguito verranno illustrate tutte le classi realizzate per conseguire le finalità esposte.

Classe Collection

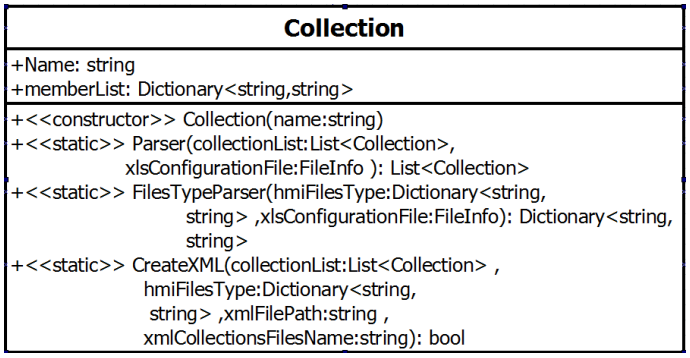


Figura 3.8: Rappresentazione UML per la classe Collection

Questa classe, rappresentata in figura 3.8, modella l'informazione corrispondente a tutti i tipi di collezioni inserite nel corrispondente foglio excel: essi sono definiti da un nome che ne contraddistingue il tipo, e per ognuno di essi è prevista una lista di coppie membro-valore rappresentate nel sistema come una coppia chiave-valore di un dizionario.

Il costruttore ha il compito istanziare una nuova collezione a partire dal nome del tipo contenuto nel file excel, creando contemporaneamente un nuovo dizionario associato ad essa. I due parser progettati, di tipo statico, servono rispettivamente per acquisire le informazioni di due tipi diversi di collezioni: quelle che corrispondono ai singoli elementi relativi alla macchina e quelle legate alla definizione dei tipi di file di Hmi da trattare. Questa differenziazione dei parser si è resa necessaria poiché, mentre le

collezioni relative agli elementi sono strutturate come nome ed una lista di coppie membro-valore, quelle relative ai file di Hmi sono puramente una coppia tipo di file-valore, in modo tale da stabilire un comportamento di processamento dei file differente a seconda dello specifico tipo. In questo modo essi prendono entrambi in ingresso il file excel da processare unitamente alla lista di collezioni da riempire con i valori contenuti nel file il primo, mentre il secondo un dizionario contenente i tipi di file. Col metodo CreateXML(), anch'esso statico, si compone l'operazione duale: a partire da una lista di collezioni e da un dizionario dei tipi di file viene composto un file di tipo xml contenente tutte le informazioni estratte da file excel, in modo da produrre una parte della base di dati relativa alla fase di configurazione successiva.

Classe Element

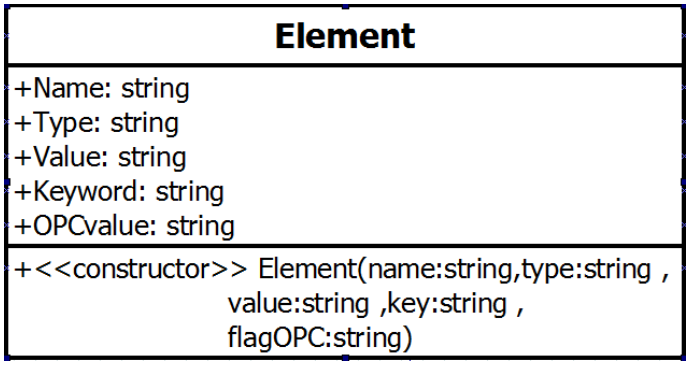


Figura 3.9: Rappresentazione UML per la classe Element

La classe Element, in figura 3.9, funge da rappresentazione per i singoli elementi di macchina: questo è il componente di granularità base per il sistema. Esso è rappresentato dai seguenti elementi:

- Name: nome dell'elemento
- Type: il tipo di collezione di cui fa parte, al fine di poter esprimere uno dei valori corrispondenti
- Value: uno dei valori appartenenti al tipo della collezione relativa, al fine di rendere l'elemento parametrico
- Keyword: il nome della keyword per la valutazione, nel livello successivo di configurazione, dell'elemento corrispondente nell'espressione booleana presente nei file di Hmi
- OPCvalue: il valore presente/assente dell'elemento rispetto al suo inserimento nel vettore Opc, destinato poi all'unità di controllo

Questa classe contiene un unico metodo costruttore per istanziare l'elemento corrispondente, a partire dal file excel di specifica della configurazione.

Classe Group

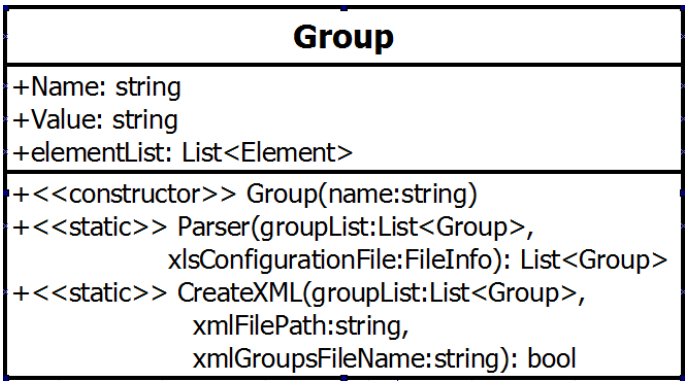


Figura 3.10: Rappresentazione UML per la classe Group

La classe Group, rappresentata in figura 3.10, è stata progettata per contenere una lista di elementi, i quali vengono specificati ognuno in un diverso foglio excel e che devono essere univoci in tutto il file: in questo modo essa è identificata da un nome e da un valore che esprime la presenza/assenza dello specifico gruppo in una determinata configurazione. I metodi che vengono definiti sono il costruttore ed i due metodi statici di ingresso ed uscita dei dati dal sistema, implementati rispettivamente da un parser e da un metodo responsabile della scrittura dei dati su file xml.

Classe Configuration

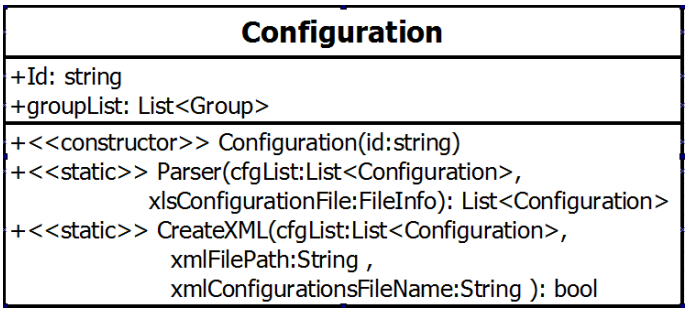


Figura 3.11: Rappresentazione UML per la classe Configuration

La classe Configuration, in figura 3.11, modella l'informazione complessiva di configurazione di una macchina: essa è composta da un identificativo che la individua univocamente nel sistema, unitamente ad una lista di gruppi che la compongono. Anche per questa classe, oltre il costruttore, vengono definiti due metodi statici per gestire ingresso ed uscita dei dati dall'applicativo.

Classe HmiFile

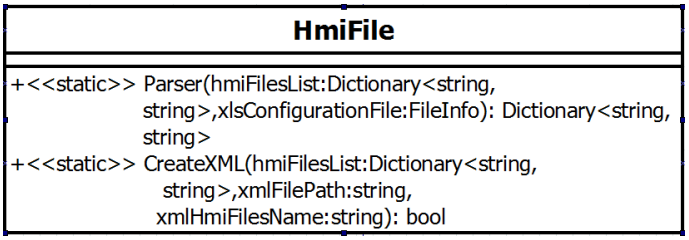


Figura 3.12: Rappresentazione UML per la classe HmiFile

La classe HmiFile, in figura 3.12, non è invece dotata di alcun attributo e di alcun costruttore, poiché serve solamente per gestire le informazione dei file di Hmi che andranno modificati in configurazione. In questo senso vengono definiti unicamente i due metodi statici necessari a trattare l'ingresso e l'uscita dei dati corrispondenti dall'applicativo.

Classe Form1

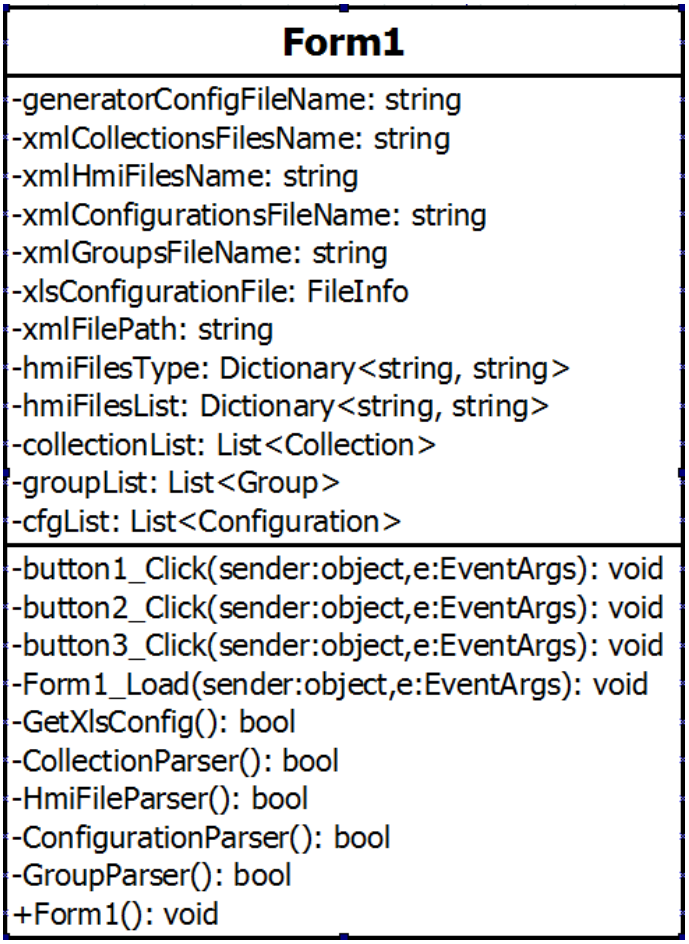


Figura 3.13: Rappresentazione UML per la classe Form1

La classe Form1, in figura 3.13, è di tipo Windows Form ed ha due compiti:

- gestire l'interazione per via grafica tra l'applicazione e l'utente
- coordinare le altre istanze al fine di comporre un comportamento funzionale corretto dell'intero sistema

La progettazione prevede l'utilizzo di un file xml per l'instrumentazione dell'applicazione: è infatti necessario alimentarla con i nomi dei file che si andranno a creare, relativi ad ogni parte della base di dati, in modo tale da rendere queste informazioni parametriche a discrezione dei tecnici. Questo file, chiamato Generator-config.xml e contenuto nell'attributo generatorConfigFileName, contiene una serie di nodi chiamati element che rappresentano ognuno il nome da attribuire al file relativo alla parte della base di dati corrispondente: queste informazioni andranno a valorizzare gli attributi di classe chiamati xml*Type*FilesName. I due ulteriori attributi xlsConfigurationFile e xmlFilePath mantengono rispettivamente il file excel d'ingresso per la specifica di configurazione (con relativo percorso) e la directory di uscita dei file relativi alla base di dati, entrambi immessi via interfaccia utente dell'applicazione, come mostrato in figura 3.14.

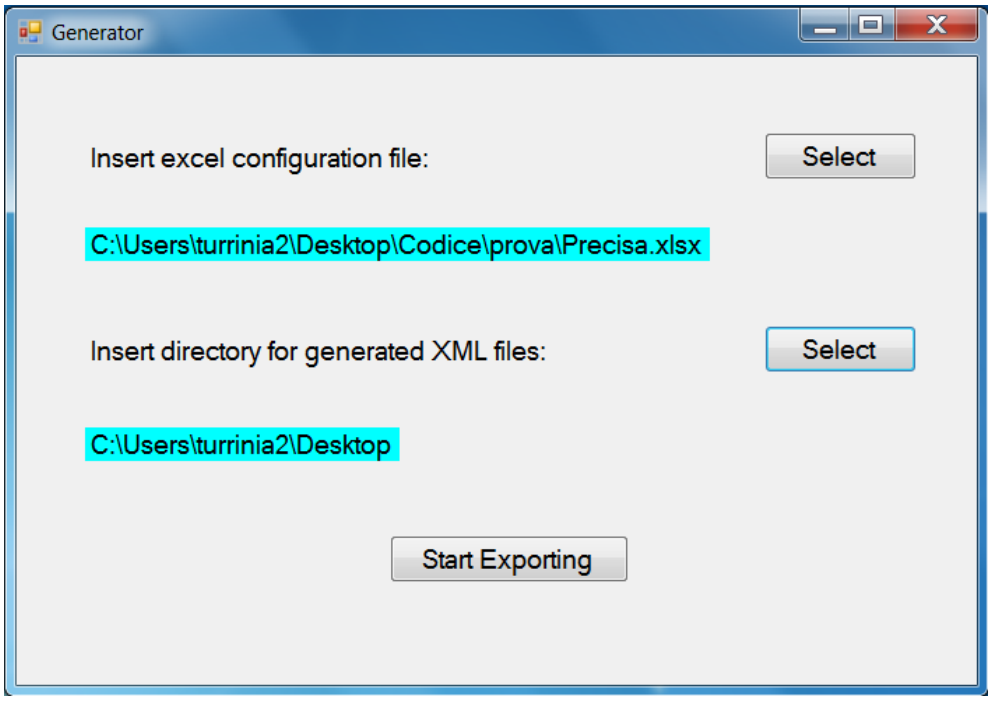


Figura 3.14: Applicativo Generator in funzione

I successivi attributi servono per mantenere in strutture dinamiche le informazioni contenute nel file excel:

- hmiFilesType è un dizionario per mantenere le coppie tipo di file-valore, modellando la collezione di dati relativa ai tipi di file

- hmiFilesList è un dizionario per mantenere le coppie file di Hmi da modificare-tipo di file, modellando il foglio excel della base di dati relativa ai file che cambiano alla variazione della configurazione
- collectionList è la lista di tutte le collezioni presenti nel sistema
- groupList è la lista di tutti i gruppi definiti nel file excel
- cfgList è l'insieme delle configurazioni specificate per una determinata macchina

La classe è stata poi corredata di metodi per la gestione coordinata di tutti gli oggetti presenti nel sistema, unitamente alle strutture dati precedentemente esposte. Tramite i tre handler denominati button*N*_Click() è possibile gestire l'interazione con l'utente:

- il primo si occupa di un controllo di tipo OpenFileDialog per l'immissione del file excel di specifica
- il secondo si occupa di un controllo di tipo FolderBrowserDialog per l'immissione del percorso assoluto dei file xml costituenti la base dati della configurazione
- il terzo si occupa di richiamare, alla pressione del bottone Start Exporting, una serie di metodi nella giusta sequenza per elaborare il file excel immesso e creare i file della base di dati

Il metodo GetXlsConfig() si occupa, attraverso un approccio Dom al documento xml di configurazione dell'applicativo, di riempire gli attributi relativi al nome dei file xml della base di dati. I quattro metodi parser, progettati ognuno per una parte specifica della base di dati, sono stati realizzati come wrapper. Il fine è infatti quello di richiamare nella giusta sequenza i parser relativi ad una certa porzione della base di dati unitamente ai relativi metodi per la creazione dello specifico file xml tramite un'unica chiamata logica: essi restituiscono tutti un valore booleano che definisce l'esito delle operazioni invocate.

Configurator

L'applicazione Configurator rappresenta il fulcro dell'intera infrastruttura di supporto alla riconfigurazione dinamica di Hmi della macchina in esame: alimentata dalla base di dati presente nei file xml prodotti nella fase di configurazione precedente, essa offre un'interfaccia grafica agli utenti al fine di poter selezionare con quale configurazione avviare la produzione in macchina. I presupposti da cui si parte nello sviluppo dell'applicativo sono quelli precedentemente esposti, cioè da un lato quello di avere già instrumentato alcune porzioni dei file di Hmi attraverso regole booleane per poterne permettere un riadattamento in maniera automatica, e dall'altro di aver creato una base di dati da cui poter attingere per gestire la riconfigurazione della macchina. In questa ottica l'idea di funzionamento prevede di poter far scegliere all'utente, via interfaccia grafica

dell'applicazione, una possibile configurazione di lavoro tra quelle specificate nella fase precedente, come mostrato in figura 3.15: una volta selezionata quella con cui si andrà in produzione, l'infrastruttura si occupa di individuare i file specificati e quindi modificarli secondo la configurazione immessa, in modo tale da permettere un adattamento automatico di tutta la parte inerente all'Hmi.

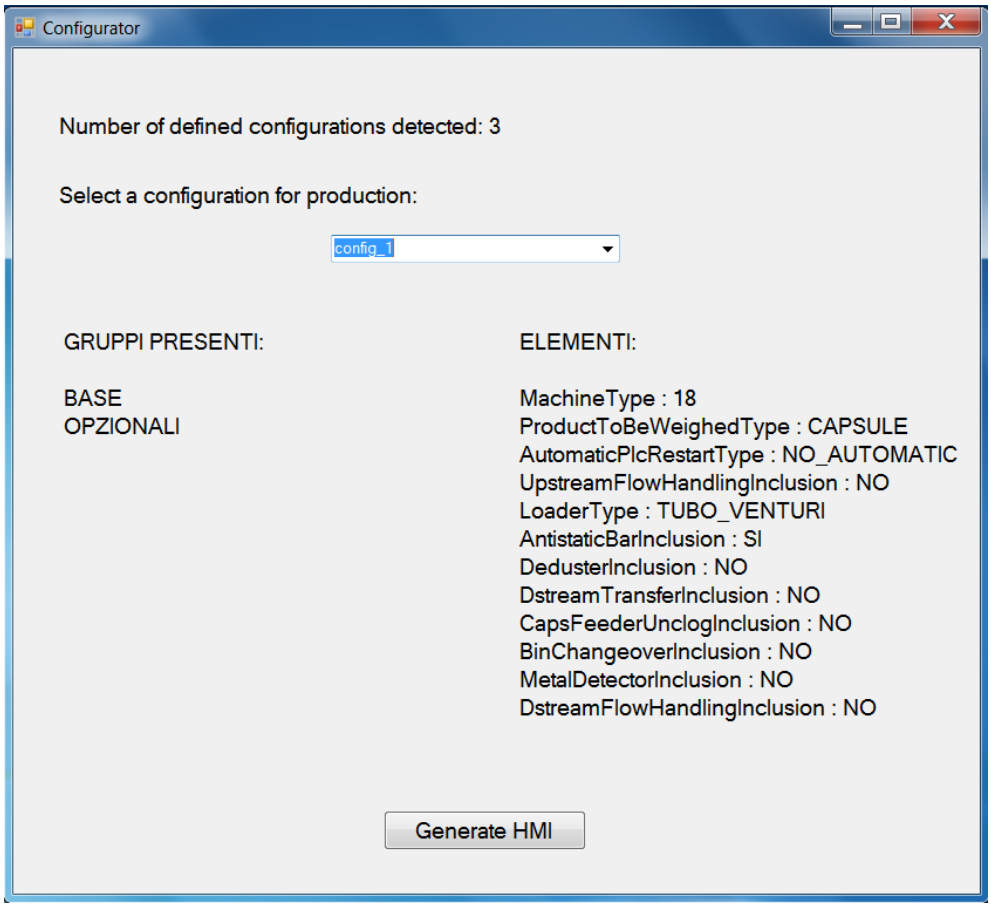


Figura 3.15: Applicativo Configurator in funzione

Lo schema delle classi di questo sistema poggia su quello effettuato nella fase precedente: in questo senso sono state importate dall'applicativo Generator le classi Element, Group, Configuration, Collection: esse sono state depurate dai metodi statici Parser() e CreateXML(), poiché non più necessari, ma ne sono stati mantenuti attributi e costruttori per poter incapsulare nel software i dati corrispondenti.

Il diagramma delle classi collegato alle parti di valutazione delle regole booleane e di parsing dei file di Hmi da adattare secondo una specifica configurazione di lavoro viene mostrato in figura 3.16: tramite la classe HmiFile, che si comporta come un front-end, è possibile differenziare il comportamento di modifica dei file a seconda del loro tipo, fornendo al contempo un approccio univoco agli specifici metodi. A loro volta questi parser fanno utilizzo della classe ExpressionEvaluation al fine di

determinare l'esito vero/falso relativo alla regola booleana con cui è decorata una determinata porzione di codice.

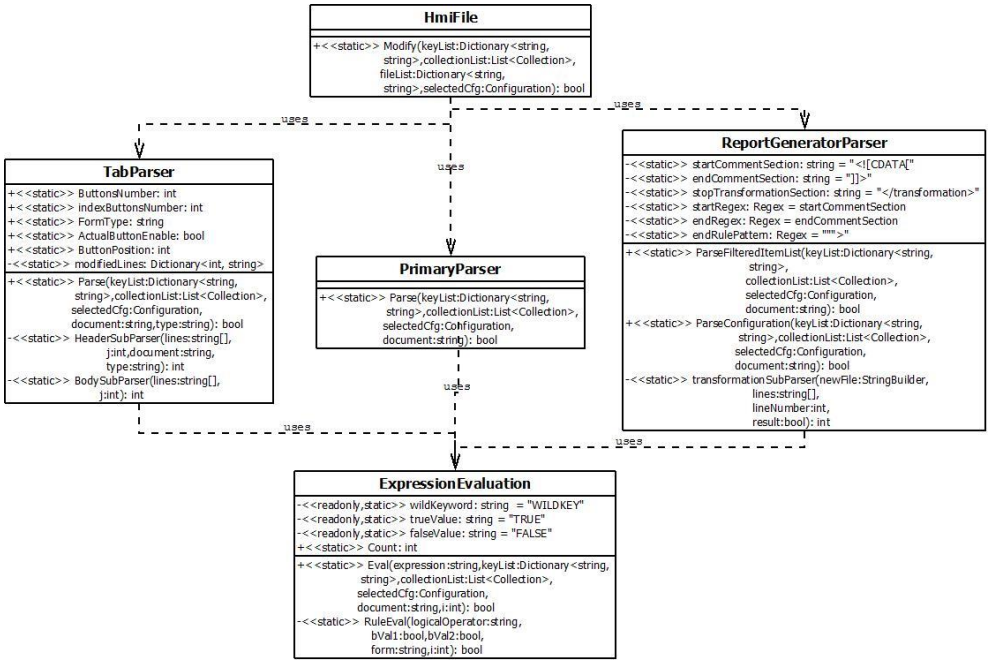


Figura 3.16: Diagramma delle classi relativo alla modifica dell'Hmi

Oltre alle modifiche dei file di Hmi, è necessario che l'infrastruttura di configurazione istruisca opportunamente l'unità di controllo al fine di poterle permettere un funzionamento coerente con una determinata configurazione di lavoro. In questo senso le informazioni inerenti una specifica configurazione scambiate tra questo applicativo e l'unità di controllo sono state codificate in formato Json: questa sintassi di tipo testuale è stata adottata perché indipendente da specifici linguaggi di programmazione e perchè permette una comunicazione leggera senza eccessivo overhead sul canale di trasmissione. Per realizzare questa funzionalità è stata impiegata la libreria indipendente chiamata Json.NET, in modo tale da poter serializzare/deserializzare in maniera efficiente gli oggetti nel sistema coinvolti nella comunicazione. In tal modo, all'unità di controllo verrà inviato un vettore di dati chiamato OpcArray il quale contiene una serie di elementi chiamati OpcItem, rappresentanti le singole informazioni da scambiare: la modellazione di questa parte del sistema è mostrata in figura 3.17.

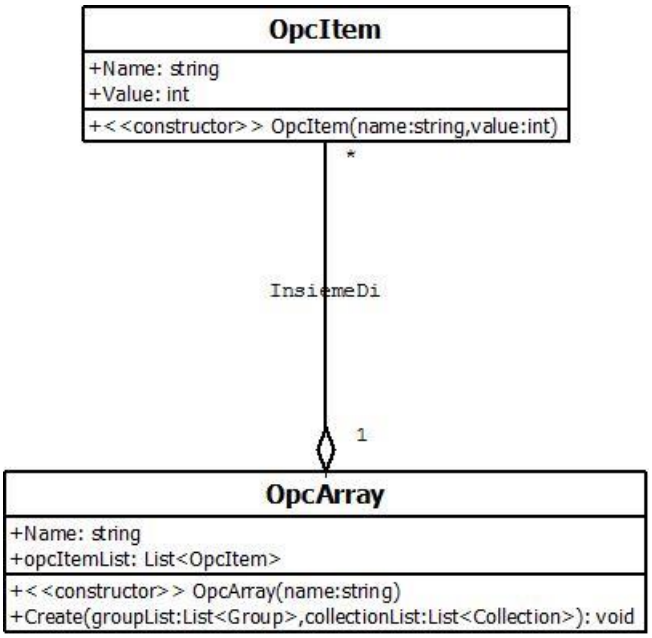


Figura 3.17: Relazione di aggregazione tra le classi relative alla comunicazione con l'unità di controllo

Di seguito verranno esposte le varie classi, con relativo comportamento, in cui è stato diviso il sistema al fine di raggiungere i requisiti funzionali citati in precedenza.

Classe Xml

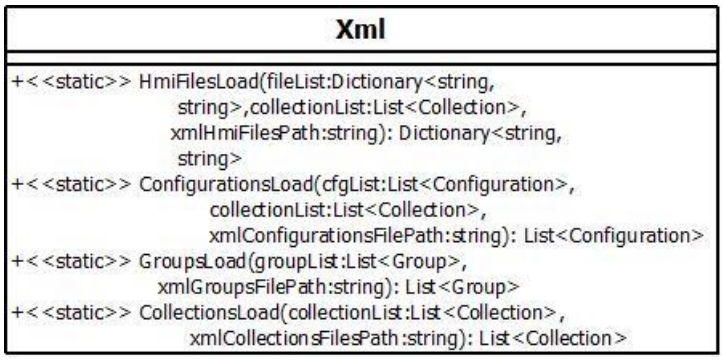


Figura 3.18: Rappresentazione UML per la classe Xml

Questa classe, rappresentata in figura 3.18, ha il compito di caricare in memoria le informazioni contenute nella base di dati prodotta nel precedente livello di configurazione, al fine di poterle utilizzare per comporre una specifica configurazione di lavoro. Attraverso un approccio di tipo Dom i quattro metodi statici elaborano, ognuno per la propria area di competenza, i file relativi ad una certa porzione della base di dati: il loro comportamento prevede la creazione di opportuni oggetti per incapsulare queste informazioni, unitamente ad una serie di vettori (sotto forma di liste/dizionari) in cui essi vengono memorizzati per poterli trattare in maniera efficiente.

Classe HmiFile

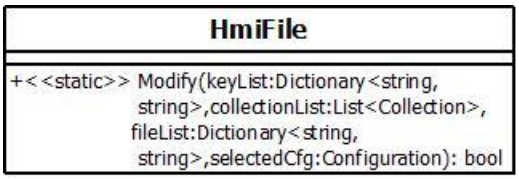


Figura 3.19: Rappresentazione UML per la classe HmiFile

La classe HmiFile, mostrata in figura 3.19, funziona come front-end per i parser sottostanti, i quali sono specifici ognuno per una determinata tipologia di file relativi all'Hmi: il fine è quello di offrire univocità per tutti i vari tipi di chiamate. In questo senso, il comportamento dell'unico metodo Modify(), che è implementato come statico, è quello di esaminare nel dizionario fileList ogni file presente (contenuto come chiave), ed a seconda del corrispondente tipo (contenuto come valore) invocare il parser opportuno. I parser che è possibile invocare sono di tre tipologie:

I. Tab: opera sui file relativi alle pagine di Xima
II. Primary: opera sul descrittore di Xface relativo allo specifico database
III. ReportGenerator: opera sia sul file di configurazione dell'applicativo di gestione dei report (Report Generator), sia sul file contenente la lista di elementi da filtrare nel report finale specifici per una determinata configurazione

Nel caso la tipologia di un file non fosse compresa in uno di questi tre casi, il sistema è stato progettato per lanciare un'eccezione, permettendo così all'operatore di individuare con precisione il problema. Ogni metodo di parsing ha come uscita un valore booleano che rappresenta l'esito positivo/negativo della modifica di un determinato file: il risultato di ogni singola chiamata viene poi messo in somma logica condizionale con tutti gli altri, in modo tale da comporre lo stato finale dell'operazione di modifica del dizionario dei file.

Classe TabParser

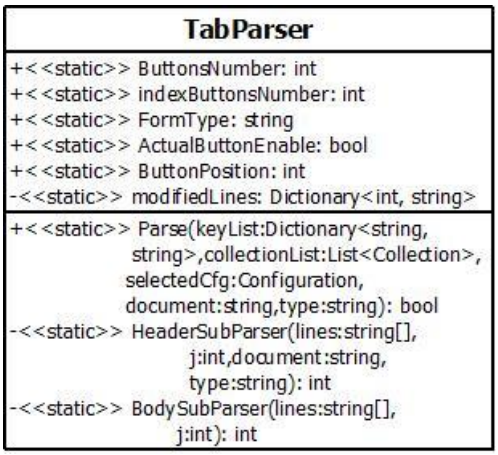


Figura 3.20: Rappresentazione UML per la classe TabParser

Questa classe, il cui diagramma UML è mostrato in figura 3.20, è stata realizzata per trattare i file relativi alle pagine di Xima attraverso il metodo statico Parse(): la loro struttura è stata divisa in due parti, header e body, per poter operare in maniera differente a seconda della sezione del file in cui ci si trova tramite la chiamata ad uno dei due relativi sotto parser. L'idea di progetto che si è adottata è quella per cui un certo file viene elaborato sezione per sezione: alla lettura di ogni riga, in caso di modifica dovuta alla valutazione di una determinata regola booleana, essa viene memorizzata nel dizionario modifiedLines per permetterne una gestione agevole ed efficiente, altrimenti essa viene copiata non trasformata nel file di destinazione. Qui la chiave è rappresentata dal numero di riga, poiché univoco per ogni file, mentre il valore corrisponde alla riga modificata. La trasformazione dipende dalla sezione di pagina in cui ci si trova:

- per l'header riguarda solamente il numero totale di bottoni contenuti nel file
- per il body può riguardare più elementi:
  - il commento/decommento dell'intera sezione di codice che rappresenta il bottone, a seconda del suo stato attuale e della valutazione dell'espressione booleana che lo decora
  - l'incremento/decremento del numero progressivo del bottone entro la pagina, a seconda dell'operazione svolta sul bottone precedente

Il metodo di funzionamento è il seguente: il file che viene passato al metodo Parse() in ingresso sotto il nome di document viene caricato in memoria e diviso in array di righe col metodo ReadAllLines(): questo fa parte della classe File, la quale viene qui impiegata perché a basso livello utilizza un oggetto StreamReader, che si rivela molto efficiente per leggere file anche di notevole dimensione. Contemporaneamente viene istanziato il nuovo file d'uscita come StringBuilder e svuotato il dizionario statico contenente le

righe modificate. A questo punto il file in ingresso viene elaborato una prima volta: all'interno di questa porzione di codice il comportamento differisce a seconda della sezione in cui ci si trova. Se la sezione incomincia con la stringa "@#" allora viene invocato il metodo statico privato HeaderSubParser(), che lavora sul blocco di codice contando per quattro righe a meno di linee vuote o commenti. Esso ha i compiti di:

- effettuare un controllo di conformità sulla tipologia di file, pena lancio di un'eccezione
- individuare il numero totale di bottoni presenti in pagina ed il numero di riga contenete questa informazione, assegnando un valore agli attributi ButtonsNumber e indexButtonsNumber rispettivamente, al fine di mantenere una coerenza complessiva tra le varie operazioni di abilitazione/disabilitazione pulsanti

Se invece la sezione incomincia con una regola booleana, allora viene invocato il metodo statico privato BodySubParser(): esso lavora su sei righe a meno di linee vuote, e successivamente la regola booleana non è possibile introdurre commenti dato che si opera su file dirty. L'attributo ActualButtonEnable contiene il risultato vero/falso della valutazione della regola per un certo bottone in esame, effettuata tramite la classe ExpressionEvaluation, mentre l'attributo ButtonPosition funge da contatore per incrementare/decrementare il numero progressivo legato ad ogni bottone. Il metodo in esame opera in maniera differente a seconda dello stato in cui si trova il blocco di codice e dell'esito della valutazione della regola booleana posta in testa, come schematizzato in tabella 4.

	Sezione commentata	Sezione non commentata
Bottone abilitato	Comportamento A	Comportamento B
Bottone non abilitato	Comportamento C	Comportamento D

Tabella 4

Comportamento A

Se il bottone deve essere abilitato perché la corrispondente espressione booleana viene valutata come vera, ma al contempo la corrispondente sezione in pagina è commentata, allora il bottone deve essere attivato. Questo implica l'eliminazione dalla sezione dei caratteri di inibizione delle relative righe, unitamente all'aggiornamento del numero progressivo del bottone stesso.

Comportamento B

Se il bottone è già abilitato perché la corrispondente espressione booleana è stata valutata in precedenza come vera, ma al contempo la corrispondente

sezione in pagina non è commentata, allora il bottone è già attivo. Questo implica solamente l'aggiornamento del numero progressivo del bottone stesso, a seconda di quelli attivati che lo precedono in maniera posizionale nel file.

Comportamento C

Se il bottone non deve essere abilitato perché la corrispondente espressione booleana viene valutata come falsa, ed al contempo la corrispondente sezione in pagina è già commentata, allora il bottone non deve essere soggetto a modifiche poiché risulta già inibito.

Comportamento D

Se il bottone non deve essere abilitato perché la corrispondente espressione booleana viene valutata come falsa, ma al contempo la corrispondente sezione in pagina non è commentata, allora il bottone deve essere disattivato. Questo implica l'introduzione in testa a tutte le righe della sezione interessata del carattere di commento al fine di inibire il bottone stesso.

Entrambi i metodi privati di parsing restituiscono come risultato al chiamante il numero di riga in cui è terminata l'elaborazione di un determinato blocco di codice, in modo tale da permettere al ciclo di processamento più esterno di continuare l'elaborazione dell'intero file dal punto in cui si è arrivati senza ripetizioni. Una volta effettuata questa prima fase di trasformazioni, il risultato che si ottiene è il dizionario modifiedLines completamente riempito di tutte quelle righe di pagina modificate secondo una determinata configurazione rispetto al file originale. A questo punto, il file in ingresso viene processato una seconda volta al fine di sostituire le linee originali con quelle modificate: una volta terminata questa seconda fase, il file originale viene eliminato e sostituito con quello prodotto per la configurazione immessa.

Classe PrimaryParser

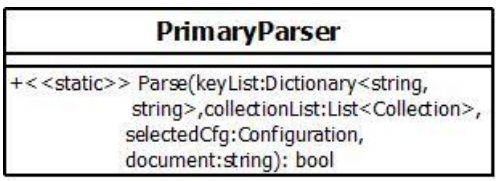


Figura 3.21: Rappresentazione UML per la classe PrimaryParser

Questa classe, illustrata in figura 3.21, è dotata di un solo metodo statico Parse() con il compito di elaborare il relativo descrittore di Xface opportunamente instrumentato di regole booleane per ogni linea. Il criterio di progetto che si è adottato è quello per cui il metodo lavora riga per riga a meno di quelle vuote o che non contengono il tag "@KEYWORD": in

questi casi ricopia la linea originale nel file di destinazione senza operare alcun cambiamento. Il funzionamento prevede di commentare o decommentare la riga a seconda del risultato dell'espressione logica con cui è decorata.

Classe ReportGeneratorParser

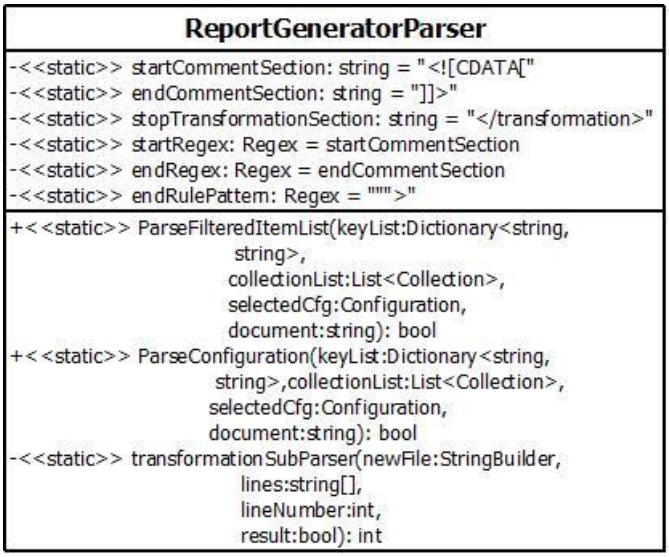


Figura 3.22: Rappresentazione UML per la classe ReportGeneratorParser

La classe ReportGeneratorParser, mostrata in figura 3.22, contiene due metodi statici di parsing per i due file che riguardano la gestione dei report:

- ParseFilteredItemList() elabora un documento xml in cui ogni nodo "<filtered-item>" è un elemento che può essere attivo o meno a seconda che esso sia incluso in una configurazione di lavoro selezionata. Ogni elemento xml è stato quindi dotato di un attributo chiamato keyword il cui valore è rappresentato da una regola booleana che permette di definire se il corrispondente nodo deve essere abilitato o meno nel filtraggio dei report
- ParseConfiguration() è il metodo che processa il file xml di configurazione dell'applicativo Report Generator, il quale gestisce tutti i rapporti informativi generati da Hmi. Esso è costituito da una serie di elementi xml "<transformation>" che sono stati corredati di attributi keyword il cui valore è rappresentato da una regola booleana che permette di definire se il corrispondente nodo deve essere abilitato o meno

Al fine di rendere il più possibile sicura e protetta da malfunzionamenti la gestione di questi file da parte degli applicativi che li utilizzano, l'inibizione delle parti di file non valide è stata progettata utilizzando come commento la sezione xml "<![CDATA[*valore*]]>" in luogo del normale commento xml "<!-- *valore* -->", in modo tale che in caso di regola booleana falsa fosse impossibile per il motore xml il parsing del nodo disabilitato. In questo

senso, al fine di elaborare questi file, non è stato possibile utilizzare un parser xml, ma è stato necessario gestirli come file di teso: il motivo è che, per questioni di protezione e sicurezza, quando un nodo non è valido, è necessario disabilitare l'intero nodo, e non solo il corrispondente valore.

Il funzionamento dei due metodi è differente: mentre quello che opera sul file contenente gli elementi da filtrare lavora riga per riga, a meno di linee vuote o non contenenti le stringhe "keyword" e "filtered-item", quello che opera sulla configurazione di Report Generator lavora per blocchi, a meno di linee vuote o non contenenti le stringhe "keyword" e "trasformation". In questo secondo caso, al fine di agevolare il processamento del file di configurazione, l'elaborazione è stata divisa su due chiamate. La prima, attraverso il metodo principale, si occupa della valutazione della regola booleana e della gestione della parte iniziale della sezione di commento. La seconda invece, col nome di transformationSubParser(), viene richiamata dalla prima una volta entrati nel nodo "<trasformation>" e si occupa di elaborare gli elementi contenuti al suo interno unitamente alla parte terminale della sezione "CDATA".

Classe ExpressionEvaluation

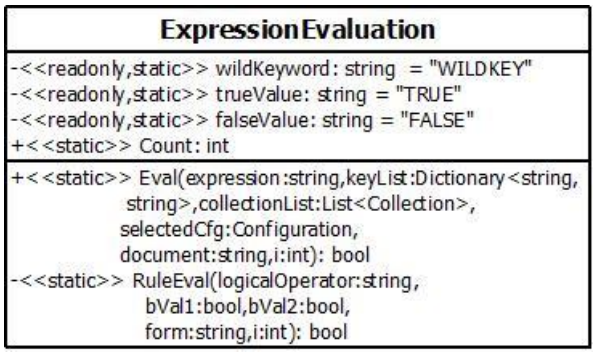


Figura 3.23: Rappresentazione UML per la classe ExpressionEvaluation

Questa classe, la cui rappresentazione UML è illustrata in figura 3.23, è stata progettata per permettere la valutazione delle regole booleane incluse nei vari file da processare. L'idea è che essa venga richiamata al bisogno ogni qual volta venga incontrata una regola nei file: ad essa viene passata l'intera stringa contenente l'espressione e tutte le strutture dati necessarie a valutarla, e restituisce, dopo una serie di controlli ed elaborazioni, un valore booleano vero/falso. La chiamata statica di alto livello Eval() prepara la stringa contenente l'espressione distinguendo l'operatore logico dai singoli elementi ed i relativi valori, ed effettua tre controlli di conformità in successione per:

I. verificare correttezza sintattica degli elementi, dei relativi valori e dell'operatore logico inclusi nella regola, pena eccezione
II. verificare che l'elemento non appartenga ad un gruppo che non è compreso nella configurazione di lavoro selezionata: in questo caso

il valore di questo singolo elemento è considerato falso, e concorre con questo esito alla valutazione complessiva dell'espressione

III. verificare che il valore legato ad un certo elemento contenuto nell'espressione coincida col valore indicato nel documento excel di specifica, a seconda che la logica sia positiva oppure negativa

Una volta passati questi controlli, viene chiamato il metodo statico di basso livello RuleEval() che permette di comporre l'esito complessivo dell'espressione come sequenza di valutazioni logiche parziali, a seconda dell'operatore contenuto nella regola e passato come parametro.

Classe OpcItem

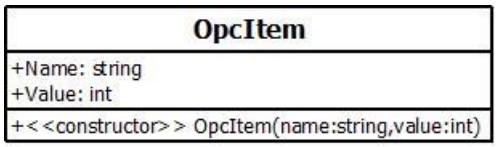


Figura 3.24: Rappresentazione UML per la classe OpcItem

Questa classe, mostrata in figura 3.24, serve per incapsulare le informazioni relative agli elementi che è necessario trasmettere dal primo livello di configurazione verso l'unità di controllo. In questo senso, essa è dotata di due attributi, entrambi impostati dal costruttore al momento della creazione dell'elemento:

- Name di tipo string per inserire il nome di un determinato elemento, in modo tale da poterlo identificare nel sistema
- Value di tipo int per inserire il valore del relativo elemento secondo la codifica definita per l'unità di controllo

Classe OpcArray

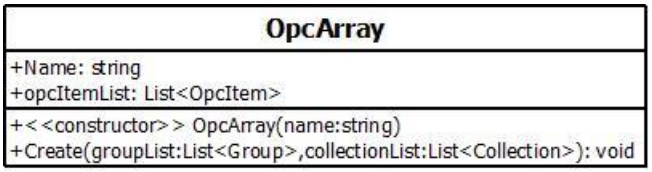


Figura 3.25: Rappresentazione UML per la classe OpcArray

La classe OpcArray, illustrata in figura 3.25, ha il compito di creare una struttura dati per gestire le informazioni da inviare all'unità di controllo. Essa è dotata di un nome, che identifica l'unità di contenimento di queste informazioni, ed una lista di oggetti OpcItem, che sono la vera e propria informazione da scambiare.

La funzione Create() serve per creare il vettore di comunicazione col controllo come insieme di elementi appartenenti a tutti i gruppi specificati nel file excel: l'unico requisito che questi elementi devono avere è quello

per cui il corrispondente campo di presenza in questo vettore, rappresentato dall'attributo OPCvalue, sia impostato al valore vero.

Classe Form1

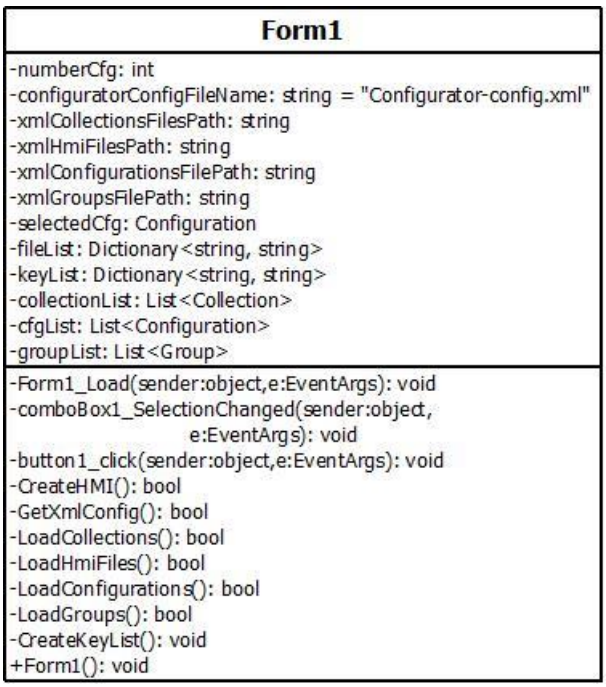


Figura 3.26: Rappresentazione UML per la classe Form1

La classe Form1, in figura 3.26, è di tipo Windows Form ed ha due compiti:

- gestire l'interazione per via grafica tra l'applicazione e l'utente
- coordinare le altre istanze al fine di comporre un comportamento funzionale corretto dell'intero sistema

L'event handler comboBox1_SelectionChanged() gestisce l'evento di selezione di nuova configurazione da parte di un utente a partire da quelle mostrate attraverso il controllo grafico di tipo combo box, inserito nella windows form di figura 3.15. Ad ogni nuova selezione, il metodo si occupa di popolare le label sottostanti la combo box, suddivise per mostrare separatamente gruppi ed elementi definiti nella base di dati, solamente con i dati presenti per la configurazione selezionata in un determinato momento. In questo modo vengono mostrati all'utente solo i gruppi presenti per una specifica configurazione, unitamente ai relativi elementi di cui sono composti: questa scelta è stata effettuata per dare modo all'operatore in macchina di poter conoscere i dettagli realizzativi della configurazione con cui poter avviare la produzione.

Tramite l'event handler button1_click() è possibile gestire l'evento di pressione del bottone Generate HMI, il quale scatena il processamento di tutti i file relativi all'Hmi inclusi nella base di dati. La logica d'implementazione prevede che l'hander richiami il metodo CreateHMI(), il

quale è realizzato come wrapper per invocare in un'opportuna sequenza i vari metodi di elaborazione dei file. Una volta che questo metodo è terminato, viene visualizzato a video un controllo grafico di tipo message box per informare l'utente sul completamento delle operazioni di adattamento dell'Hmi: questo feedback visuale è necessario per garantire un certo livello di qualità del software, dato che in questo modo l'operatore è sempre informato sull'andamento delle operazioni. Alla conclusione di queste azioni, previa conferma dell'utente, l'applicazione viene quindi chiusa in automatico.

Il metodo CreateHMI(), realizzato per invocare in un giusto ordine i metodi di elaborazione dei file di Hmi, richiama il metodo HmiFile.Modify() passandogli tutte le strutture dati popolate con la base di dati unitamente alla configurazione selezionata dall'operatore, in modo tale da adattare i vari file ad una specifica configurazione. Una volta terminate le operazioni di modifica dell'Hmi, la parte conclusiva si occupa di creare la struttura dati per la comunicazione con l'unità di controllo e di serializzarla su file in formato Json.

Col metodo GetXmlConfig(), tramite approccio Dom, viene caricato il documento xml di configurazione dell'applicativo: in questo file, il cui nome è Configurator-config.xml, sono contenuti i file ed i corrispondenti percorsi assoluti relativi alla base di dati. Queste informazioni sono perciò incapsulate negli attributi corrispondenti col nome xml*FileType*FilesPath, e successivamente caricate via Dom dai rispettivi metodi contenuti nella classe Xml, al fine di popolare le strutture relative alla porzione della base di dati interessata. I metodi di caricamento contenuti nella classe Xml non vengono invocati in maniera diretta, bensì ognuno di essi è stato incapsulato in una specifica chiamata: il fine è quello di uniformare le invocazioni alle specifiche chiamate attraverso i metodi di alto livello Load*FileType*().

La chiamata CreateKeyList() serve per creare la struttura dati dizionario contenente la lista di keyword per ogni elemento, unitamente al relativo valore: essa è stata progettata per offrire un accesso efficiente e veloce alle keyword presenti nel dominio del sistema e permettere un più agevole processamento delle regole booleane presenti nei vari file.

Infine, il metodo Form1() serve per richiamare in sequenza i metodi di caricamento della base di dati ed istanziare le strutture dati che dovranno ospitarli.

3.2 Realizzazione lato rapporti informativi

La seconda parte del progetto si è concentrata su quella sezione di infrastruttura che si interessa della gestione dei rapporti informativi nella misura in cui questa parte è coinvolta dal cambiamento di configurazione operativa.

3.2.1 Fase di analisi

L'adozione della nuova metodologia di progettazione dell'Hmi, cioè quella relativa all'introduzione di un'interfaccia ibrida legata ad una serie di stazioni di lavoro categorizzate come di primo livello, provoca un effetto indesiderato per quanto riguarda la produzione dei report. Dal momento in cui il software di Hmi viene progettato per essere unico a livello di componenti di macchina qualificanti per un certo insieme di configurazioni di lavoro, la conseguenza che si viene a creare è che, in tutti gli scenari significativi esaminati in tabella 1, il file xml di reportistica conterrà non solo i parametri tracciati specifici per una determinata configurazione, bensì tutti quelli relativi all'insieme delle configurazioni specificate. In questo modo al cliente verrebbe visualizzato l'andamento dei valori anche per quei parametri non relativi la configurazione di lavoro che aveva selezionato precedentemente per una determinata produzione.

Un secondo problema riguarda l'utilizzo della stessa cartella sia come destinazione dei file xml dei report sia per il contenimento degli script per il caricamento e la visualizzazione dei dati di report. In questa ottica, dato che la cartella è liberamente accessibile al cliente per poter gestire i file, è possibile che questi cancelli la cartella degli script, dato che non è a conoscenza del funzionamento del sistema sottostante, andando così ad inficiare il meccanismo complessivo della struttura di reportistica.

In questo senso, i requisiti che sono stati individuati per adattare il sistema di reportistica al nuovo scenario operativo sono elencati di seguito in ordine di importanza:

I. rendere il meccanismo complessivo di reportistica pienamente riconfigurabile al pari dello strato software, in modo tale da

permettere anche a questa porzione di Hmi di adattarsi dinamicamente alla configurazione operativa

II. assicurare protezione in caso di malfunzionamenti dell'applicativo di Hmi oppure dell'unità di gestione dell'interfaccia uomo-macchina, evitando di produrre file di report non corretti relativamente ad una determinata configurazione di lavoro

III. garantire la sicurezza nel funzionamento del sistema, prevenendo dal punto di vista architetturale possibili malfunzionamenti dovuti a comportamenti non corretti da parte del cliente nella gestione dei file dei report

3.2.2 Fase di progettazione

L'idea generale per garantire il rispetto del primo requisito è introdurre nell'architettura un archivio sotto forma di file xml in cui immettere la lista completa dei parametri presenti nel sistema tramite i corrispondenti identificativi. In questo modo viene creata una base di dati da cui attingere per filtrare in una fase successiva il file di report completo prodotto dall'Hmi. E' su questa base di dati che si concentra la configurabilità del meccanismo dei report: questo archivio unico, chiamato filtered-item-list.xml, viene reso adattabile al fine di permettere all'applicativo di gestione a valle di filtrare il file xml mantenendo solamente i parametri validi. In questo senso al suo interno, ad ogni nodo xml corrispondente ad un parametro da tracciare, viene associato un attributo "keyword" contenente una regola booleana formattata come esposto in precedenza: per tutti quei parametri non presenti in una determinata configurazione, l'applicativo di configurazione procede a disabilitarli, non permettendone perciò il tracciamento nel file di report finale per il cliente. Un secondo aspetto di adattamento della reportistica riguarda il file di configurazione dell'applicativo Report Generator per la gestione dei report: dal momento che il file contiene una serie di nodi xml "<trasformation>", ognuno dei quali si riferisce ad uno specifico file stx di trasformazione del file xml di report prodotto dall'Hmi, è opportuno inserire anche qui un meccanismo che renda possibile variare i tipi di trasformazione utilizzati secondo un determinato valore. In questo senso, è stato introdotto nel nodo xml relativo ad ogni trasformazione un attributo "keyword" per poter effettuare la valutazione, tramite l'applicativo di configurazione precedentemente progettato, di una regola booleana espressa come valore corrispondente.

Di seguito viene mostrata la struttura del file xml di archivio dei parametri filtered-item-list.xml:

```
<?xml version="1.0"  encoding="UTF-8"?>

<filtered-item-list>

          <filtered-item membership="description(value)" keyword="
          AND_RULE [ BARRE_ANTISTATICHE(SI) ]">02.008</filtered-
          item>

          <filtered-item membership="description(value)" keyword="
          AND_RULE [ DISINTASAMENTO(SI) ]">02.009</filtered-item>

          <filtered-item membership="description(value)" keyword="
          AND_RULE [ DISINTASAMENTO(SI) ]">02.080</filtered-item>

          <filtered-item membership="description(value)" keyword="
          AND_RULE [ DISINTASAMENTO(SI) ]">02.010</filtered-item>

          <filtered-item membership="description(value)" keyword="
          AND_RULE [ DISINTASAMENTO(SI) ]">02.011</filtered-item>

          <filtered-item membership="description(value)" keyword="
          AND_RULE [ MACCHINA_MONTE(SI) ]">02.012</filtered-item>

</filtered-item-list>
```

L'attributo "membership" relativo ad ogni singolo nodo viene introdotto nel rispetto del punto due dei requisiti. Infatti poiché è opportuno prevedere sia il caso di blocco o terminazione improvvisa dell'applicativo di filtraggio e gestione dei report che sta a valle, sia il caso di riavvio o caduta dell'unità di gestione dell'interfaccia uomo-macchina, si è reso necessario pensare ad un meccanismo di protezione dell'architettura in caso di malfunzionamenti. In questo senso l'attributo in esame serve per istruire l'applicazione di filtraggio e gestione dei report a valle che un determinato parametro, il quale è abilitato in una specifica configurazione di lavoro, deve corrispondere ad una particolare stazione di lavoro fisicamente presente in macchina. Grazie a questo collegamento tra il parametro tracciato ed una specifica caratterizzazione della macchina, anche in caso di malfunzionamento e successivo riavvio dell'unità/applicativo con conseguente cambio di configurazione di lavoro, si è sicuri che i file relativi alla reportistica non verranno filtrati considerando una lista sbagliata dei parametri da tracciare.

Riguardo il punto tre dei requisiti, è necessario introdurre modifiche sostanziali all'architettura generale del sistema di reportistica: per risolvere il problema della cartella di destinazione unica relativa ai file di report prodotti dall'Hmi, è stato necessario che l'applicativo a valle progettato per il filtraggio e la gestione dei report si occupasse anche di creare più sotto cartelle, entro quella sorgente, in cui inserire copie multiple dei file di report filtrati. In questo modo, in un secondo momento tramite l'applicativo Report Generator, sarà possibile spostare i file qui inseriti in altre cartelle distinte accessibili per il cliente in maniera facilitata e controllata.

Lo schema di funzionamento generale dell'architettura che sarà progettata per il filtraggio e la gestione dei file di report è divisa in due fasi, come illustrato in figura 3.27. Nella prima fase, di competenza dell'applicativo Report Handler, viene operato il filtraggio sui alcuni file di report completi inseriti in una cartella specificata come parametro d'ingresso: successivamente, una volta create le sotto cartelle necessarie, prima i file di report filtrati vengono spostati al loro interno e poi i file di report completi vengono eliminati. Il metodo di lavoro prevede un funzionamento a polling con timeout: Report Handler una volta avviato rimane sempre in funzione, e si attiva ogni volta dopo un determinato periodo di tempo impostato come parametro d'ingresso. Nella seconda fase, invece, è compito dell'applicativo Report Generator spostare i file prodotti nello stadio precedente in cartelle precedentemente definite in moda tale da consentire un accesso facilitato ma controllato al cliente, intervenendo opportunamente sul suo file di configurazione per istruirlo di conseguenza.

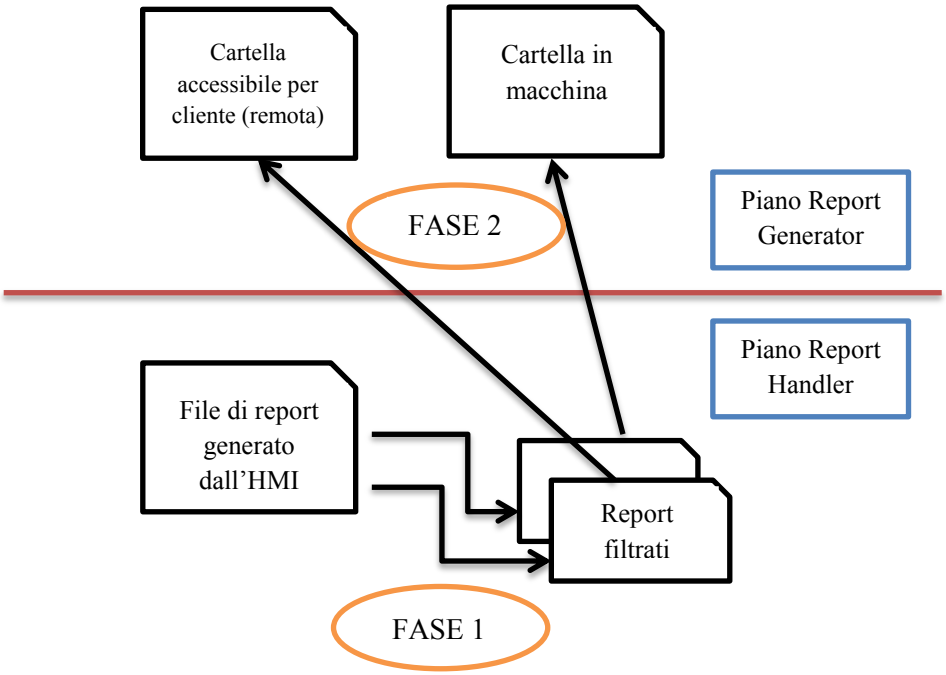


Figura 3.27: Schema di funzionamento dell'architettura di reportistica

Tramite questa nuova infrastruttura di reportistica sarà possibile integrare i file prodotti nell'architettura esistente differenziando le tecnologie utilizzate secondo gli scenari di funzionamento esposti in tabella 1.

Scenario A

Dato che in questo scenario l'applicativo di interfaccia grafica uomo-utente genera un file di report con cui alimentare il file preview.html ad ogni richiesta da parte dell'utente, non è possibile sostituirlo con quello filtrato prodotto da Report Handler, poiché esso sarebbe sovrascritto ad ogni invocazione dell'utente. In questo senso si è risolto il problema

intervenendo sul file xsl che processa il file di report: progettandolo nella maniera opportuna, si fa in modo che esso attinga al file filtered-item-list.xml per visualizzare a video solamente i parametri in esso abilitati.

La funzionalità voluta è stata raggiunta tramite il template di codice illustrato: ogni volta che il parser xml incontra un nodo dal nome "record" si fa in modo che esso controlli nella lista dei parametri da filtrare, contenuta nel file filtered-item-list.xml, se il corrispondente identificativo è presente o meno.

```
<xsl:template match="record">

   <xsl:variable name="it-IdTemp" select="./it_id"/>

          <xsl:if test=
"document('file:///C:/Work/ximascada/ReportGenerator/ filtered-item-
list.xml')/ filtered-item-list[filtered-item=$it-IdTemp]" >

                 <tr id="{n}">

                        <xsl:apply-templates/>

                 </tr>

          </xsl:if>

</xsl:template>
```

In caso positivo, tramite la condizione di test espressa dal comando "if test", il nodo viene visualizzato a video nel formato opportuno, altrimenti esso viene scartato.

Scenario B

In questo scenario è possibile alimentare l'applicativo di Hmi con i file di report filtrati prodotti da Report Handler: grazie alla sua configurabilità legata alla possibilità di specifica della cartella da cui alimentarsi, modificando opportunamente il file LngDatX.par nella sezione relativa al LocalArchivePath, è possibile fornirgli i file da visualizzare a video.

Scenario D

In questo contesto, dato che non è possibile intervenire in alcun modo sulla trasformazione dei file di report completi effettuata da Report Generator, è stata progettata la nuova applicazione Report Handler che opera a valle di tutto il sistema di reportistica. La sua realizzazione verrà esposta nel paragrafo successivo, ed i suoi compiti sono i seguenti:

- filtraggio dei file di report completi prodotti da Xima
- spostamento dei file filtrati
- rimozione dei file completi
- creazione e gestione delle sotto cartelle

3.2.3 Fase di implementazione

L'implementazione della nuova architettura riguardo la gestione dei rapporti informativi prevede anche l'introduzione nel sistema dell'applicativo chiamato Report Handler che lavora a valle dell'intero meccanismo. Esso è stato realizzato come applicativo di tipo Console poiché per offrire le sue funzionalità non richiede un'interfaccia di tipo grafico: il relativo diagramma delle classi è mostrato in figura 3.28.

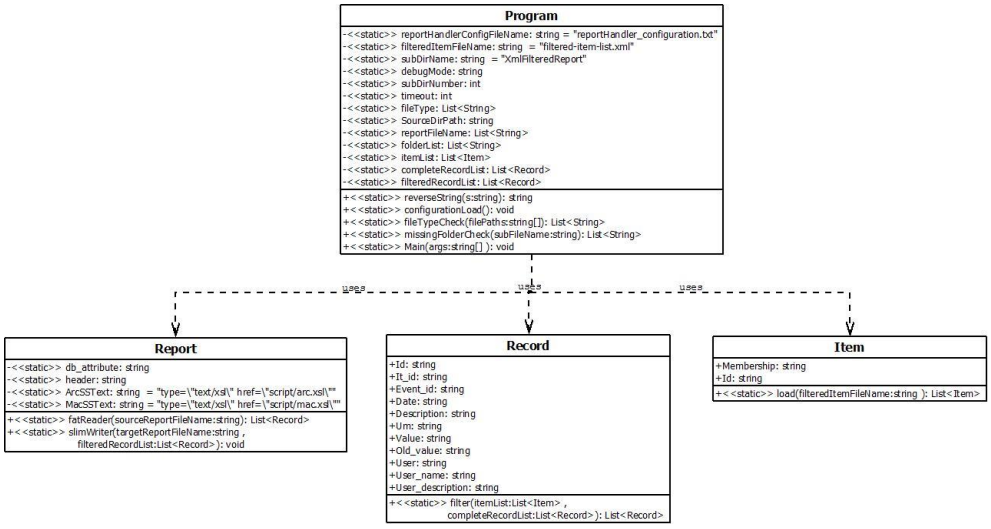


Figura 3.28: Diagramma delle classi relativo all'applicativo Report Handler

L'applicativo in esame è stato progettato per funzionare secondo varie modalità di funzionamento, in modo tale da essere il più flessibile possibile, offrendo in questo modo varie modalità di lavoro. In questo senso, in fase di avvio, esso abbisogna di un file di configurazione, chiamato reportHandler_configuration.txt, in cui vanno specificate le seguenti informazioni:

- se si lavora in modalità debug, cioè se deve essere mantenuto o meno il file di report completo originale
- il numero di sotto cartelle da creare
- il tempo di ciclo (timeout in millisecondi)
- la lista di tipologie di report da trattare, ad esempio parametri d'archivio e/o correnti
- il percorso della directory di lavoro in cui sono prodotti i file di report completi da analizzare

Classe Item

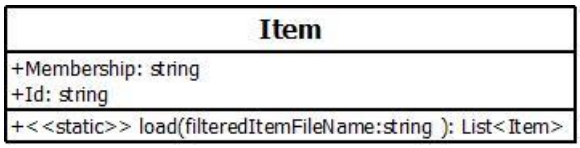


Figura 3.29: Rappresentazione UML per la classe Item

La classe Item, il cui diagramma UML è illustrato in figura 3.29, serve per incapsulare l'informazione presente nel file dei parametri da filtrare filtered-item-list.xml. Essa contiene due attributi:

- Membership: per contenere l'informazione legata al meccanismo di protezione dai malfunzionamenti
- Id: per rappresentare il valore di ogni nodo attivo presente nel file di filtraggio, relativo all'identificativo del parametro da tracciare non disabilitato per una determinata configurazione di lavoro

L'unico metodo presente, di tipo statico e denominato load(), è stato realizzato per deserializzare, attraverso un oggetto di tipo XmlSerializer, il file filtered-item-list.xml in maniera veloce, decorando opportunamente la classe ed i relativi attributi con i tipi xml corrispondenti appartenenti alla classe Xml.Serialization.

Classe Record

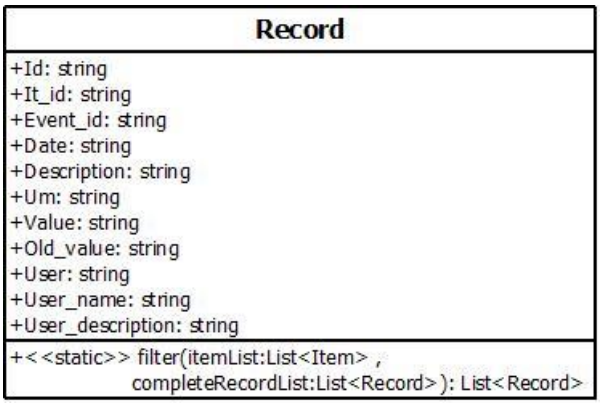


Figura 3.30: Rappresentazione UML per la classe Record

Questa classe, raffigurata in figura 3.30, modella l'informazione contenuta nei file di report completi prodotti da Xima, basandosi sulla granularità del singolo record. In questo senso essa è corredata con attributi che corrispondono in rapporto 1:1 ai nodi contenuti nel nodo xml radice "record". Il singolo metodo presente filter(), realizzato come statico, ha il compito di filtrare la lista completa dei record, passata come secondo parametro, sulla base della lista degli elementi da tracciare per una determinata configurazione, passata come primo parametro. Il filtraggio viene eseguito scorrendo prima la lista dei parametri di filtraggio e poi la

lista completa dei record: per ogni record viene controllato che la "Description" ed il "Value" coincidano entrambi con quelli specificati per il parametro con il valore "It_id" selezionato. In caso positivo il record sotto esame viene aggiunto ad una nuova lista di record filtrati, altrimenti si passa ad esaminare il record successivo.

Classe Report

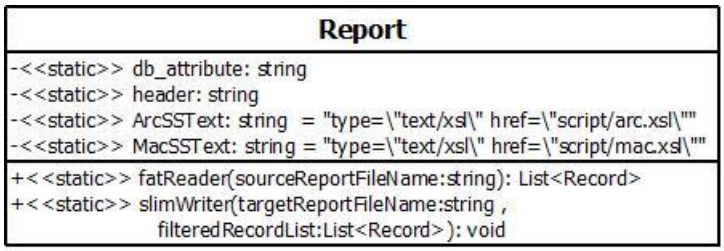


Figura 3.31: Rappresentazione UML per la classe Report

La classe Report ha il compito di realizzare le funzionalità di consumatore e produttore dei file di reportistica attraverso rispettivamente i due metodi statici esposti nel diagramma UML di figura 3.31. Tramite il metodo fatReader() viene letto il file di report completo ed incapsulati tutti i record nelle opportune strutture dati: esso è stato realizzato utilizzando la classe XmlReader che utilizza un parser Sax al fine di essere il più leggeri ed efficienti possibile. Tramite questo approccio si evita di caricare in memoria l'intero documento xml da elaborare, poichè si impiega un lettore che fornisce un accesso veloce, non in cache e di tipo forward-only ai dati xml. L'utilizzo di un parser Dom con il caricamento dell'intero documento xml in memoria è stato invece scartato dopo un'accurata analisi dei documenti di report prodotti da Hmi: infatti, considerando la quantità di dati prodotti dalla nuova architettura ad interfaccia ibrida comprendente tutti i parametri da tracciare per un determinato tipo di macchina, l'impiego di questa tecnica avrebbe introdotto nel sistema un consistente overhead prestazionale per un applicativo che è destinato a lavorare costantemente per tutto il tempo di funzionamento della macchina. Col metodo slimWriter() si compone invece l'operazione duale: a partire da una lista di record precedentemente filtrati, secondo i dati contenuti nel file filtered-item-list.xml, viene prodotto il file xml di report d'uscita attraverso un oggetto XmlWriter.

Classe Program

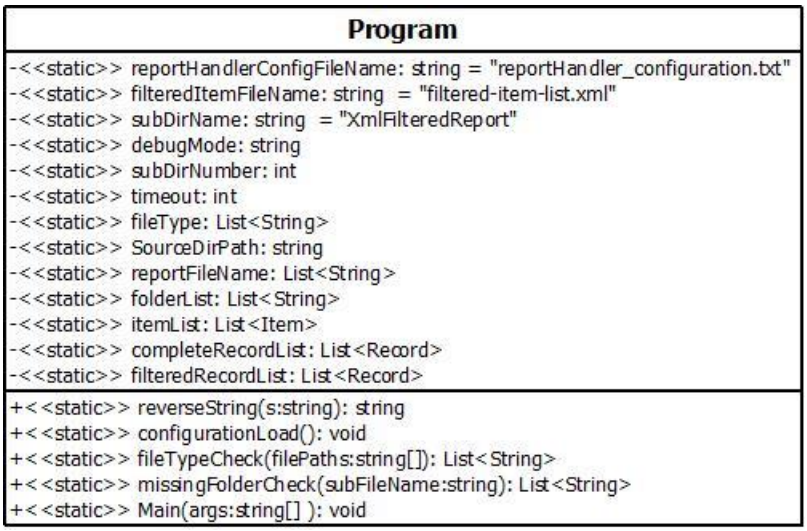


Figura 3.32: Rappresentazione UML per la classe Program

La classe Program di figura 3.32 serve per orchestrare il funzionamento complessivo di tutti gli oggetti nel sistema. Col metodo statico configurationLoad() viene inizializzata l'applicazione con le informazioni necessarie per raggiungere una certa flessibilità operativa: in questo senso vengono valorizzati gli attributi corrispondenti ai dati di interesse. Il metodo statico reverseString() è una chiamata di utilità generale che, a partire da una stringa d'ingresso, la restituisce rovesciata in uscita. La chiamata statica fileTypeCheck() serve per controllare la tipologia dei file di report completi presenti nella cartella di lavoro specificata: se uno dei file appartenenti alla lista passata come parametro possiede un'estensione coerente con le tipologie di report immesse nel file di configurazione dell'applicativo, allora esso viene aggiunto alla lista dei file da elaborare. Con l'invocazione statica missingFolderCheck() si controlla l'esistenza nelle sotto cartelle di lavoro dei file filtrati corrispondenti a quelli completi presenti nella cartella radice: in caso negativo, la sotto cartella viene aggiunta ad una lista contenente tutte le cartelle in cui dovranno essere inseriti i file filtrati. Infine, il metodo Main() viene realizzato per gestire tutte le varie funzionalità precedentemente esposte. La metodologia di lavoro è la seguente: dapprima viene caricato il file di configurazione dell'applicativo e il file contenente la lisa dei parametri di filtraggio. Successivamente, entrati in un ciclo infinito, ad intervalli regolari specificabili attraverso il timeout di configurazione, l'applicazione controlla dapprima la mancanza di report filtrati nelle sotto cartelle di lavoro relativamente alla tipologia specificata in configurazione, ed in caso positivo esegue in successione i tre metodi corrispondenti alla lettura del file di report completo, relativo filtraggio e composizione del file filtrato. Da ultimo, essa si occupa si spostare i file prodotti nelle sotto cartelle in cui essi risultano mancanti.

Conclusioni e sviluppi futuri

Il lavoro di tesi che è stato presentato affronta il problema legato alla riconfigurazione dinamica legata al software d'interfacciamento uomo-macchina per macchine automatiche multidosaggio dotate di una alta flessibilità operativa. Dapprima si è effettuata un'analisi approfondita della macchina automatica Adapta al fine di comprenderne a fondo il funzionamento complessivo sia dal punto di vista meccanico che dal punto di vista hardware/software. Questa fase è servita a raccogliere informazioni per definire in maniera puntuale quali fossero le problematiche legate allo scenario di cambio di configurazione relative alla produzione e, di conseguenza, per individuare gli obiettivi da seguire nella progettazione e nella realizzazione della nuova architettura. I problemi principali che si sono identificati nell'infrastruttura esistente sono legati al fatto che alla flessibilità operativa della macchina del lato meccanico non corrisponde la stessa flessibilità del lato del software d'interfacciamento uomo-macchina. Questo porta a scaricare le criticità e le complessità del sistema sia sui tecnici che devono approntare le macchine sia sul cliente finale: in questo senso si hanno ripercussioni sia dal punto di vista della creazione delle macchine automatiche, sia dal punto di vista della loro gestione e della loro manutenibilità.

Seguendo quindi gli obiettivi individuati alla luce delle criticità precedentemente esposte per il sistema esistente, si sono passate in rassegna tutte quelle metodologie di progettazione che coinvolgono la realizzazione di infrastrutture software allo stato dell'arte, prendendo a riferimento design pattern che fossero universalmente riconosciuti come standard ingegneristici. Individuato quindi quello che per il sistema in uso meglio si confacesse agli obiettivi specificati in precedenza, lo si è dapprima adattato secondo una metodologia identificata, e poi esso è stato inserito nella nuova architettura complessiva che è stata progettata per andare a sostituire quella esistente, al fine di raggiungere le funzionalità volute.

Riferendosi quindi all'architettura progettata nella fase precedente, si è passati quindi alla sua completa realizzazione sia dal punto di vista del software d'interfacciamento uomo-macchina, sia da quello dei rapporti informativi. Da ultimo, l'intera infrastruttura è stata collaudata su macchine automatiche reali al fine di verificarne il rispetto dei requisiti iniziali, in modo tale da attuare in maniera completa le fasi di analisi, progettazione, implementazione e collaudo che costituiscono gli stadi di sviluppo del software.

Si è perciò dimostrato che tale lavoro può essere impiegato fin da subito come futura piattaforma di partenza per la realizzazione dal punto di vista software di nuove macchine automatiche dotate di flessibilità operativa, in maniera tale da introdurre nel loro modello di progettazione un alto tasso di adattabilità congiuntamente ad un'alta affidabilità e manutenibilità. In questo senso sono stati superati tutti i vincoli presenti nell'architettura preesistente, semplificando drasticamente le fasi di generazione, collaudo e manutenzione di un'Hmi: in questo modo, da un lato si sono eliminati la maggior parte dei malfunzionamenti intrinseci legati ad eventuali errori manuali, e dall'altro si sono significativamente ridotti i tempi relativi alla configurazione di una determinata versione di interfaccia. Oltre a ciò, si è risolto completamente il problema relativo all'occupazione di spazio su memoria di massa nell'unità in macchina, spostando l'intera piattaforma di Hmi verso un modello omogeneo e non più eterogeneo per composizione.

La naturale prosecuzione di questo lavoro di tesi sta nella realizzazione delle stesse funzionalità di adattamento dinamico alla produzione anche dal punto di vista dell'unità di controllo macchina: l'implementazione di un meccanismo di riconoscimento plug and play della configurazione operativa, tramite rilevazione automatica dei gruppi di dosaggio effettivamente montati, potrebbe ulteriormente semplificare le attività di progettazione delle macchine automatiche. Questa ipotesi di funzionamento comporterebbe non solo la capacità di adattare in maniera automatica l'unità di controllo all'effettiva produzione, riducendo in questo modo tutte le problematiche legate ad una loro attuazione manuale, ma richiederebbe anche una nuova gestione per l'interfacciamento con l'architettura qui proposta, dal momento in cui sarebbe necessario comunicare al software d'interfaccia le informazioni relative all'effettiva composizione della macchina al fine di permettergli di adattarsi automaticamente. Estensioni di questo lavoro di tesi possono essere invece identificate con sistemi a corredo di questa piattaforma che consentano di incrementarne le funzionalità: possibili sviluppi possono quindi riguardare la progettazione di tecnologie che permettano di automatizzare il primo livello di configurazione qui proposto tramite l'utilizzo di un applicativo grafico che consenta di specificare, eventualmente a diversi livelli di utenza, le caratteristiche di composizione di una macchina automatica. Un altro indirizzo di estensione che viene proposto è l'ideazione di un meccanismo che permetta la connessione per via remota ad una macchina automatica al fine di realizzarne una riconfigurazione, andando ad interfacciarsi con la base di dati che è stata progettata in questo lavoro per effettuare modifiche al software deputato all'interfacciamento con l'operatore. Infine, potrebbe risultare utile dotarsi di un sistema di tracciamento delle modifiche operate durante una fase di riconfigurazione unitamente ad una tecnologia per il backup automatico delle stesse: poter analizzare queste informazioni

permetterebbe di gestire con più facilità i casi legati a possibili malfunzionamenti.

Una volta che tutte queste attività saranno state svolte ed integrate fra loro per comporre un meccanismo unico, esse consentiranno ad IMA non solo di semplificare drasticamente la progettazione di nuove macchine automatiche, incrementando di conseguenza la produttività, ma anche di garantire un corretto funzionamento delle stesse in merito all'adattabilità, riducendo notevolmente le fasi di generazione, collaudo e manutenzione.

Appendice A
Lo standard OPC

OLE for Process Control (OPC) è una tecnologia utilizzata per semplificare il trasferimento dei dati tra i sistemi di controllo industriali, le interfacce operatore, i sistemi di supervisione e sistemi software aziendali come i database. Fu sviluppato con l'intenzione di fornire una tecnologia standard che consentisse a sistemi di controllo differenti di interagire tra loro.

Prima dell'avvento di Opc, gli sviluppatori di applicazioni software erano costretti a sviluppare un driver di comunicazione per ciascun sistema di controllo con cui intendevano scambiare dei dati; Opc, invece, ha messo a disposizione un'interfaccia comune per interagire con differenti prodotti di controllo industriale, indipendentemente dall'hardware e dal software utilizzati nel processo, come mostrato in figura A.1.

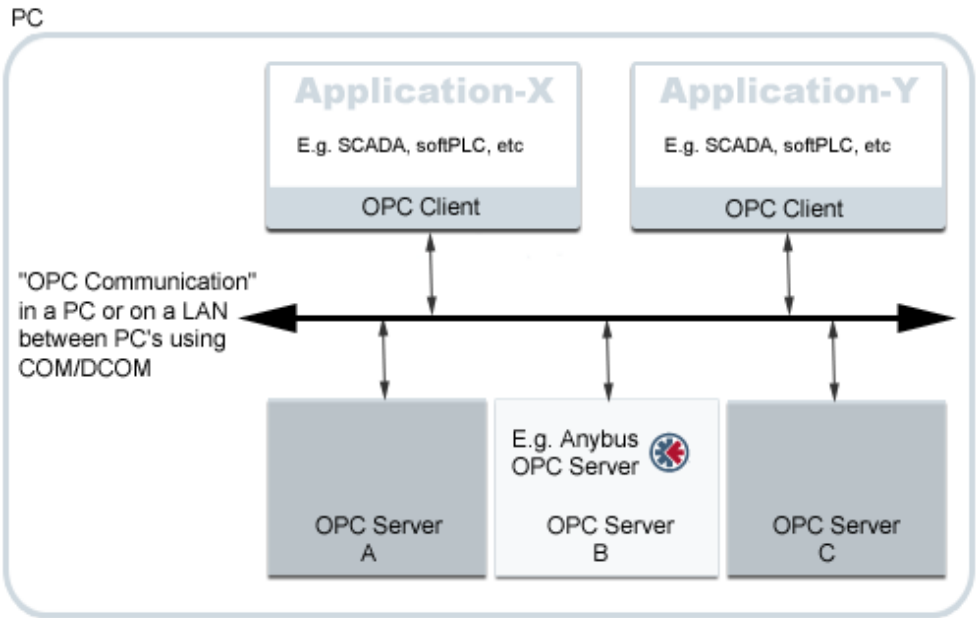


Figura A.1: Schema di funzionamento Opc

Il principio di base della specifica Opc è un modello client-server nel quale un qualsiasi processo (client) basato su Opc può accedere a qualsiasi sorgente di dati (server) dotata di interfacce Opc.

In particolare:

- un server Opc consente ai fornitori hardware di offrire ai propri acquirenti dei servizi che permettono a qualsiasi client di accedere alle loro apparecchiature
- l'applicazione client controlla i dispositivi e gestisce i relativi utilizzando i metodi standard di accesso ad un oggetto Opc

Opc segue perciò un'architettura client-server, come mostrato in figura A.2: un server Opc è un'applicazione software che raccoglie le informazioni dai dispositivi (PLC, DCS, ecc.) tramite protocolli nativi (Modbus, Profibus, ecc.), il server poi fornisce l'accesso ai dati collezionati tramite gli oggetti COM; i client Opc leggono e scrivono i dati sul dispositivo di campo tramite il server Opc.

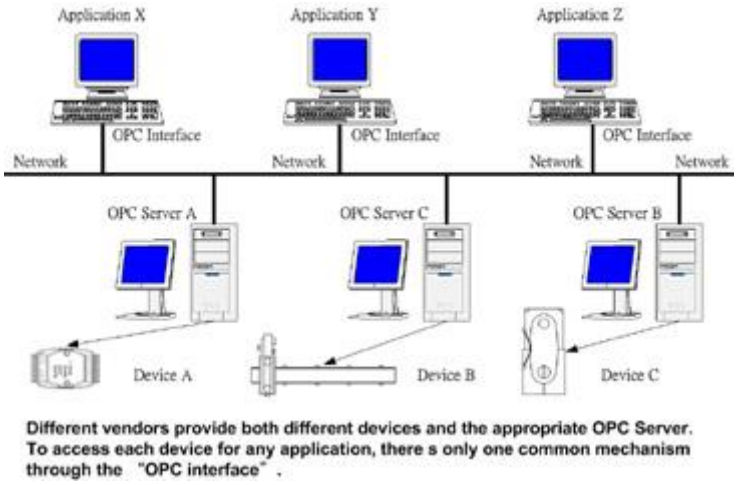


Figura A.2: Modello client/server per Opc

Le informazioni disponibili tramite un server Opc sono organizzate in gruppi secondo criteri di efficienza e i gruppi possono essere:

- pubblici (disponibili per qualunque client Opc)
- locali (accessibili solo dal client che li ha creati)

Uno dei vantaggi più significativi nell'utilizzo di Opc, consiste nel fatto che l'applicazione non necessita di conoscere i dettagli dell'architettura interna del dispositivo con cui scambia i dati, ma esclusivamente i nomi dei gruppi e degli elementi a cui è interessata.

Bisogna comunque sottolineare il fatto che Opc non elimina la necessità dei driver di comunicazione: il costruttore sviluppa il server Opc specifico per il proprio prodotto, utilizzando il protocollo di comunicazione necessario per il funzionamento del suo dispositivo; a questo punto diventa più semplice interfacciare il sistema con altro software compatibile con Opc.

La prima formulazione dello standard nacque dalla collaborazione di alcuni fornitori di prodotti di automazione, riconosciuti a livello mondiale, e Microsoft; fu chiamata semplicemente "Opc Specification" ed era basata su COM e DCOM di Microsoft. Le specifiche definiscono un insieme di oggetti, interfacce e metodi per semplificare l'interazione delle applicazioni di controllo di processo e di automazione della produzione.

E' importante capire che Opc non è un protocollo di comunicazione alla stessa stregua di Ethernet o TCP/IP, ma rappresenta un livello di astrazione più alto: è costituito da un insieme di API che nascondono la rete di trasporto sottostante e la codifica utilizzata per lo scambio dei dati.

Allo stato attuale l'attenzione dei progettisti dello standard Opc si è focalizzata sulle aree comuni a tutti i venditori. Funzionalità aggiuntive verranno definite nelle successive versioni. La versione attuale è focalizzata su tre differenti server:

- Online DataAccess: offre metodi per uno scambio efficiente di dati (read-write) tra un'applicazione ed un device per il controllo di processo.
    - Write sincrona/asincrona
    - Read polling, report by exception, cache and device read
- Allarmi e Gestione degli Eventi: meccanismi, per i client Opc, che permettano la notifica della occorrenza di un evento specifico o di condizioni d'allarme
- Historical Data Access: strumenti per la lettura e il processamento dei dati per un'analisi storica.

Gli obiettivi di Opc sono:

- semplicità di implementazione
- flessibilità per soddisfare le necessità di più venditori
- fornire un livello alto di funzionalità
- efficienza

A.1 Architettura

Le specifiche includono due tipi di interfacce:

- Custom Interface per l'utilizzo da parte degli sviluppatori dei server e dei client
- Automation Interface come riferimenti ad un insieme di interfacce di OLE automation per supportare clienti sviluppati con applicativi business ad alto livello come Excel, Visual Basic, etc

Queste specifiche Opc non danno informazioni sull'implementazione delle interfacce ma solo sulle loro caratteristiche: tutte le funzionalità si basano su Application Programming Interfaces (APIs). Poiché è difficile provvedere ad una singola API che lavora bene sia per C++ che per VB l'interfaccia OPC Custom è ottimizzata per l'uso di Client C++, mentre l'interfaccia Automation è ottimizzata per l'uso di Client VB. Tipicamente Opc fornisce l'interfaccia Custom come sua primitiva mentre l'interfaccia Automation è costruita come strato superiore della Custom Interface, come mostrato in figura A.3.

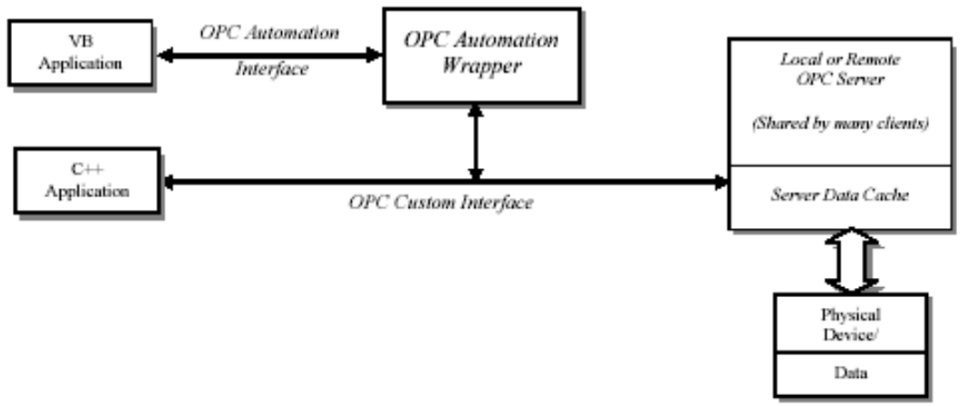


Figura A.3: Tipologie di interfacce

Il Server Opc è costituito da diversi oggetti quali: il Server, il Gruppo e l'Item, tra loro in logica di aggregazione, come mostrato in figura A.4.

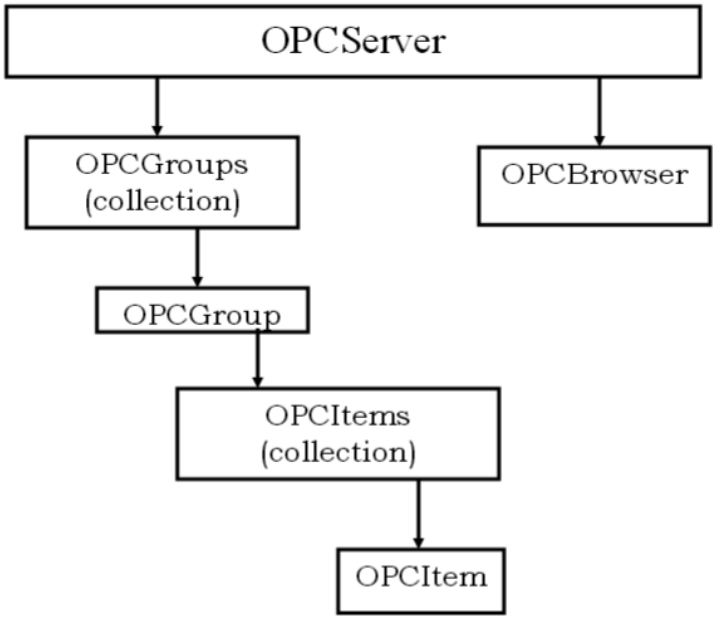


Figura A.4: Modello ad oggetti per il server Opc

I gruppi sono un meccanismo conveniente per i client per organizzare i dati: gruppi differenti possono essere usati da porzioni differenti di una applicazione. Gli aggiornamenti al loro interno sono multipli ed essi contengono informazioni sugli elementi per provvedere alla loro organizzazione. Questo tipo di oggetto accede ai dati in modo sincrono e asincrono: essi si distinguono in pubblici e privati. I gruppi pubblici sono condivisi da più client e possiedono delle particolari interfacce, mentre quelli privati sono gruppi esclusivi.

Gli item definiscono il punto I/O di un dispositivo: l'item Opc non è accessibile come un oggetto da un client Opc e non possiede una interfaccia esterna. L'accesso avviene tramite il gruppo Opc in cui l'item è contenuto o in cui è definito. Ad ogni item è associato:

- valore corrente
- qualità/status
- timestamp

Bisogna comunque tenere presente che gli elementi non sono dati, ma sono ad essi connessi. Gli elementi Opc devono essere pensati come una specifica sull'indirizzo dei dati.

Riferimenti bibliografici

Passadore E., Triossi P., *Manuale utente XIMA*

Bimbo S., Colaiacovo E. (2006), *Sistemi SCADA: Supervisory control and data acquisition*, Apogeo

Gamma E., Helm R., Johnson R., Vlissides J. (2002), *Design Patterns - Elementi per il riuso di software ad oggetti*, Pearson Education Italia

Colonese E. (2006), *Strategia di sviluppo e ciclo di vita del software*

OPC: OLE for Process Control, http://www.studiolevi.com/lii/pub_web/le10-opc.pdf

OPC, http://www.dia.uniroma3.it/autom/Reti_e_Sistemi_Automazione/PDF/OPC.pdf

Json.NET, http://json.codeplex.com/

EPPlus, http://epplus.codeplex.com/

Materiale sulla programmazione C#,

http://stackoverflow.com

http://msdn.microsoft.com

Ringraziamenti

Desidero innanzitutto ringraziare il professor Faldella per l'opportunità e la disponibilità dimostratemi nello sviluppo di questo lavoro di tesi.

Un sentito ringraziamento va a tutte le persone dell'Ufficio Tecnico Elettrico di IMA Active per la calorosa ospitalità rivoltami: in particolar modo voglio ringraziare Enrico e tutto il suo team per le preziose indicazioni fornitemi nello svolgimento di questa attività, senza le quali essa non sarebbe stata possibile. Un ringraziamento particolare lo devo a Giorgio per avermi concesso l'occasione di produrre questo lavoro.

Ringrazio anche tutti gli amici ed i parenti che mi sono stati vicini lungo questo percorso, grazie ai quali il peso e la fatica degli studi sono divenuti sopportabili.

Da ultimo, ma primo per valore, va uno speciale ringraziamento alla mia famiglia, ed in particolar modo ai miei genitori, a cui questa opera è dedicata, per i numerosi sacrifici sostenuti al fine di permettermi di giungere a questo traguardo e per il costante sostegno che essi mi hanno profuso durante tutti gli anni di studio.