ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria - DISI

Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA

in

Sistemi in Tempo Reale M

INNOVAZIONE ED UNIFICAZIONE DELLE INTERFACCE IN SISTEMI DI AUTOMAZIONE INDUSTRIALE ATTRAVERSO IL PROTOCOLLO DI COMUNICAZIONE OPC-UA

Relatore:

Prof. Eugenio Faldella

Candidato:

Matteo Ferraresi

Correlatori:

Dott. Stefano Candori

Dott. Ing. Enrico Passadore

Anno Accademico 2020/2021

Sessione Unica

"Cos'è che studi te?"

"Ingegneria informatica"

"Te…te informi anche Dio!"

Ciao nonna, grazie di tutto.

Indice

Appendice A

Indice delle immagini

Introduzione

L'Emilia Romagna è da sempre identificata come la Motor Valley grazie alla presenza di grandi marchi produttori di auto sportive, moto e componentistica varia (Ferrari, Lamborghini, Maserati, Pagani, Alpha Tauri, Ducati, Energica, Magneti Marelli sono solo alcuni esempi della ricchezza motoristica di questo territorio). Nonostante questa forte identità apprezzata e invidiata da tutto il mondo, negli ultimi anni la nostra regione è riuscita a guadagnarsi anche l'appellativo di Data Valley, tanto da poter paragonare Bologna a San Francisco e la nostra regione alla celeberrima californiana Silicon Valley[1]. Questo nome è dovuto alla presenza di poli tecnologici volti alla raccolta e l'analisi di dati di prim'ordine che sono figli di importanti investimenti economici da parte sia di enti pubblici (come la regione) che privati. Queste due facce non sono in competizione ma operano anche in sinergia per portare ancora più prestigio e clienti alle aziende stesse, crescendo sia a livello di immagine che economico.

Tra i fattori fondamentali di questa crescita va considerato il fatto che questa terra è uno dei più importanti bacini industriali a livello italiano ed europeo e, in particolar modo, lo è dal punto di vista del packaging: il tratto tra Bologna e Reggio-Emilia viene individuato come la Packaging Valley[2] italiana e, con l'avvento

[1] https://24plus.ilsole24ore.com/art/bologna-come-san-francisco-cosi-l-italia-e-diventata-data-valley-d-europa-AE58PfE?refresh_ce=1

[2] https://thepackagingvalley.com/

dell'industria 4.0, recentemente è stato coniato il nome di Automation Valley[3]. Inoltre, con lo scatenarsi della pandemia di SARS-CoV-2, negli ultimi due anni tutto questo sistema ha "subìto" una fortissima spinta. All'interno di questo contesto si trova, ben salda in una posizione da podio[4], IMA S.p.A. (acronimo di Industria Macchine Automatiche).

Immagine 1: logo di IMA

Questa posizione di assoluto rispetto è dovuta alla presenza dell'azienda all'interno di vari mercati[5] quali

- Pharma: ricerca e sviluppo di soluzioni innovative nel mercato farmaceutico a livello globale;
- Food & Dairy: soluzioni per le macchine di confezionamento economiche e innovative per l'industria lattiero-casearia e alimentare;
- Confectionary: packaging in tutte le forme e dimensioni per il settore dolciario grazie a un approccio flessibile basato sulle necessità dei clienti;

[3] https://www.packagingobserver.com/packaging-valley-perche-la/

[4] https://www.industriaitaliana.it/lindustria-del-packaging-italiano-sul-gradino-piu-alto-del-podio-globale/

[5] https://ima.it/it/il-gruppo-ima/mercati/

Tea & Beverage: decenni di esperienza nelle tecnologie di confezionamento e la diffusa presenza nei settori tè e bevande;

Coffee: esperienza, affidabilità e soluzioni complete dalla ricezione dei chicchi di caffè al fine linea, soddisfacendo tutti i requisiti di processo, confezionamento e servizio più esigenti;

Personal & Home Care: vasta gamma di attrezzature in grado di coprire tutti gli aspetti del packaging, soluzioni personalizzate e flessibilità permettono di soddisfare questo mercato esigente e orientato alla qualità;

Tissue & Nonwoven: soluzioni integrate e tecnologicamente avanzate per la realizzazione di prodotti igienici assorbenti e il loro packaging funzionale (tissue, salviette bagnate e asciutte, pannolini baby&adult e assorbenti igienici);

Automation: offre ai propri clienti un'ampia gamma di soluzioni grazie alla cinquantennale esperienza nelle tecnologie di assemblaggio;

Tobacco: produce linee di packaging innovativo per il mercato del tabacco.

Inoltre è presente il progetto IMA Digital, che dà vita a prodotti digitali, strumenti virtuali e applicazioni smart per raggiungere la piena efficienza produttiva nel mondo produttivo[6].

Grande importanza è assunta dal settore IMA Pharma che è articolato in tre divisioni: Active, Life e Safe.

[6] https://ima.it/en/ima-digital/

IMA è stata in generale in prima linea durante la fase pandemica producendo in primo luogo macchine per la produzione e l'imballaggio di mascherine[7]; inoltre molto importante nell'ultimo anno è stata la divisione Life che si occupa della produzione di impianti di lavorazione in regime asettico che comprendono liofilizzatori, isolatori, sterilizzatori oltre che sistemi di lavaggio, riempimento, tappatura e ghieratura[8]. Tra questi vi è, in differenti implementazioni, la Filler: una macchina che viene utilizzata per riempire i flaconi con prodotti medicinali come ad esempio i vaccini e che può essere considerata una delle più utili e attive in questo tempo pandemico.

Le altre divisioni del settore IMA Pharma si occupano di situazioni differenti ma non per questo meno importanti. IMA Active agisce nel campo della lavorazione e produzione di soluzioni a dose solida (come le pastiglie); IMA Safe si occupa della produzione di una gamma completa di confezionatrici blister, contacapsule e compresse, confezionatrici per bustine e stick, intubettatrici e astucciatrici e, in generale, fornisce soluzioni complete di fine linea.

Immagine 2: loghi di IMA Pharma e delle sue divisioni

[7] https://www.corriere.it/buone-notizie/20_luglio_25/ima-quei-35-milioni-mascherine-prodotte-bustine-te-1169a4f8-ccfc-11ea-83db-f973956fabb4.shtml

[8] https://ima.it/pharma/portfolio/?search=&apps=&packs=&techs=&brands=3860&view=spoiler

Anche la divisione IMA BFB collabora con le tre divisioni Pahrma in quanto progetta e realizza attrezzature per il fine linea, offrendo una vasta gamma di macchine per il confezionamento secondario (come la cellofanatura, il riempimento degli scatoloni e la palletizzazione).

Grazie a questa collaborazione è possibile produrre macchine (in realtà il termine corretto da utilizzare è linee, in quanto sono insiemi di varie macchine) in grado di eseguire un processo produttivo completo: partendo dagli elementi primari (come i singoli componenti chimici, le fialette, i tappi, etc.) ai arriva a ottenere un prodotto finito e fruibile direttamente dai consumatori (come ad esempio la scatolina del farmaco così come si può acquistare in farmacia, compresa la scatola con tutte le informazioni compresa la data di scadenza, il bugiardino piegato e il flacone riempito con il medicinale ed etichettato correttamente) o da altre aziende (come ad esempio un pallet incellofanato composto da vari scatoloni di cartone contenenti le varie scatole di medicinali).

Questa tesi è svolta proprio all'interno della divisione Life di IMA; il lavoro si focalizza nell'ambito della programmazione HMI/SCADA (che detta in parole semplici si occupa della generazione e gestione della parte software più vicina all'essere umano piuttosto che alla macchina, ambito invece legato alla programmazione dei PLC) e in particolare sulla comunicazione tra il calcolatore che governa la macchina e il terminale attraverso il quale l'operatore la supervisiona e comanda.

Scopo della tesi

Come accennato poco sopra, questa tesi è stata svolta all'interno dell'ufficio HMI-SCADA Engineering della divisione Life del settore Pharma del gruppo IMA, con sede a Castel San Pietro Terme.

L'idea è quella di tentare di sfruttare il protocollo di comunicazione OPC-UA, nato già da oltre una decina d'anni ma ancora fortemente in fase di implementazione, per migliorare la relazione, la comunicazione e l'infrastruttura attualmente presenti tra le macchine industriali e le interfacce uomo-macchina utilizzate per governare tali macchine.

Immagine 3: logo di OPC-UA

Il protocollo OPC-UA permette di snellire e velocizzare la comunicazione dei dati provenienti dalla macchina: l'attuale comunicazione prevede il passaggio dei dati attraverso l'esposizione da parte della macchina di una serie di array contenenti diversi valori dei dati e la pubblicazione di file contenti le indicazioni relative alla localizzazione delle varie informazioni all'interno di questi array; queste informazioni sono poi utilizzate per creare l'interfaccia grafica. Questa procedura di esposizione e configurazione deve essere eseguita ogni volta che la macchina subisce una modifica; si cercherà quindi di sfruttare la naturale forte strutturazione dei dati offerta da OPC-UA per evitare questa procedura che può portare a diversi errori.

Grazie a questa proprietà offerta dal protocollo si cercherà di creare una interfaccia utente innovativa in grado non solo di mostrare a un potenziale operatore una macchina e le possibili azioni da svolgere su di essa, ma in particolare sarà in grado

di navigare la struttura offerta dalla macchina grazie al protocollo OPC-UA senza la necessità di conoscerla a priori. L'interfaccia sarà in grado di autogenerarsi all'avvio andando a esplorare la struttura mostratagli dal PLC dalla macchina.

Questo è potenzialmente un enorme vantaggio: potrebbe permettere di avere un'interfaccia unica valida per tutte le macchine risparmiando la necessità di creare un'interfaccia uomo-macchina specifica per ogni macchina che l'azienda produce.

La trattazione non sarà solamente teorica ma avrà anche un forte approccio applicativo, mostrando nel concreto come utilizzare, sfruttare e implementare il protocollo di comunicazione.

L'approccio di questa tesi è sperimentale e non si possono conoscere a priori gli esiti di tale sperimentazione. Inizialmente viene affrontata una fase di studio del protocollo e degli strumenti a disposizione; successivamente è mostrata l'implementazione del protocollo assieme agli effettivi risultati raggiunti. Difficilmente questi risultati saranno prontamente sfruttabili e integrabili all'interno dei sistemi attualmente utilizzati dall'azienda, ma possono essere un ottimo punto di partenza per un'innovazione futura.

Capitolo 1

Analisi del contesto

Nel settore industriale, ogni macchina automatica è composta da elementi base come i sensori e gli attuatori e il loro insieme è chiamato campo (o field). Il lavoro coordinato dei dispositivi di campo è controllato da un computer che, però, non è paragonabile a quelli che si possono trovare nelle nostre case o nei nostri uffici. Questi computer sono detti PLC (Programmable Logic Controller) e si occupano nello specifico della gestione o controllo dei processi industriali che la macchina deve eseguire.

Per poter gestire i PLC delle varie macchine, sopra a questi troviamo un'infrastruttura software detta SCADA (Supervisory Control And Data Acquisition): un sistema centralizzato di controllo e monitoraggio utilizzato per acquisire e supervisionare in tempo reale i dati provenienti dai processi di produzione. Questo interagisce con i vari PLC

Come ultimo elemento fondamentale troviamo infine l'HMI (Human Machine Interface) che si occupa di mostrare all'essere umano un ambiente grafico che rappresenta la macchina, in cui può leggerne il funzionamento e, tramite determinati pulsanti e oggetti grafici è in grado di inviare comandi al PLC che invia a sua volta istruzioni ai dispositivi di campo.

PLC – Programmable Logic Controller

Un controllore a logica programmabile, o PLC, è un computer molto semplice basato su un microprocessore che realizza funzioni di controllo per l'automazione industriale interfacciando i dispositivi di campo con il sistema di supervisione. Contrariamente a quanto avviene nella logica cablata, dove i vari componenti sono collegati fisicamente, con l'utilizzo di un PLC tutte le funzioni logiche richieste per l'automazione di una macchina o di un processo industriale sono realizzate seguendo un determinato programma. La forza dei PLC è proprio la ciclicità con cui questa serie di istruzioni è eseguita; l'unico motivo di interruzione del ciclo può essere la ricezione di un interrupt che deve essere gestito prima di poter riprendere il normale funzionamento, ossia l'iterazione del ciclo di istruzioni.

Un buon PLC è immune ai rumori elettrici, resistente alle vibrazioni e agli impatti e lavora in regime *hard real-time*: i processi sono eseguiti con tempistiche strettamente predeterminate: se la deadline non dovesse essere rispettata, il sistema non sarebbe in grado di recuperare il tempo perduto e questo porterebbe a gravi errori in quanto la ciclicità dei processi verrebbe meno. Questo può portare a gravi danni sia sulla macchina stessa ma anche sull'ambiente circostante.

SCADA - Supervisory Control And Data Acquisition

I sistemi SCADA sono fondamentali all'interno dell'industria automatica in quanto hanno la capacità di interporsi tra l'uomo e la macchina. Da un lato recepiscono il funzionamento della macchina, i valori dei parametri e possibili errori di funzionamento. Dall'altro lato offrono una visione di tutte queste informazioni all'uomo nella maniera che più gli è congeniale e permette di far effettuare alla

macchina azioni specifiche senza la necessità di conoscere nel dettaglio tutte le istruzioni che la macchina invia agli elementi di campo e prevenendo possibili errori.

Come suggerisce il nome, un sistema SCADA ha quindi tre compiti principali:

- Supervisione → il sistema deve permettere all'operatore di osservare lo stato attuale del processo che la macchina sta eseguendo nella maniera quanto più significativa possibile;
- Controllo → il sistema deve avere la capacità di interagire con il processo supervisionato modificandone l'esecuzione in base a procedure prestabilite o in base a istruzioni impartite dall'operatore;
- Acquisizione dati → è la capacità del sistema di acquisire i dati dai PLC (per poi elaborarli inviandoli all'interfaccia utente) e dall'HMI (per inviare comandi ai PLC delle macchine); i dati sono memorizzati all'interno di un database che memorizza anche lo storico dei dati per permettere analisi e reportistica.

HMI – Human Machine Interface

L'HMI, pur essendo parte del sistema SCADA, viene analizzata a parte in quanto è l'elemento più impattante per l'essere umano; è infatti grazie a queste (e soprattutto alla loro evoluzione) che l'operatore è in grado di governare una macchina anche molto complessa.

In passato le interfacce uomo-macchina erano composte da led, relè, switch fisici e tanti altri componenti hardware che svolgevano azioni differenti; l'operatore doveva essere a piena conoscenza del funzionamento di ogni singolo elemento e della loro interazione. Negli anni questi sistemi si sono evoluti sempre di più grazie anche

alle sempre maggiori capacità e complessità acquisite dal software: oggi l'interfaccia uomo-macchina può essere modellata in maniera molto più intuitiva e user friendly, potendo anche fornire diverse visualizzazioni della macchina o del processo supervisionati secondo il livello di dettaglio desiderato (che può dipendere anche dal grado di conoscenza dell'operatore nei confronti processo o della macchina supervisionati).

Il miglioramento esponenziale delle HMI e dei sistemi SCADA più in generale, ha permesso di rendere i processi produttivi sempre più rapidi, precisi, affidabili e governabili, favorendo l'espansione dell'industria automatica.

Questa evoluzione ha portato anche l'aumento esponenziale dell'operatività degli operatori: mentre in passato l'operatore doveva interagire molto di più con la macchina (che era anche più lenta) oggi ha la possibilità di gestire più macchine contemporaneamente. L'operatore oggi ha sempre meno il ruolo di mero esecutore di operazioni meccaniche e ripetitive; la sua funzione si può parafrasare come l'estensione umana degli SCADA: è il supervisore e controllore dei processi produttivi.

Capitolo 2

Strumenti a disposizione

In questo capitoli verranno analizzati gli strumenti software e i protocolli utilizzati per lo svolgimento del progetto di tesi.

OPC-UA

Il protocollo di comunicazione OPC-UA nasce nel 2008 con l'idea di sostituire il precedente protocollo OPC (Open Platform Communication), sempre introdotto dalla OPC Foundation, utilizzato in ambito industriale per la comunicazione tra i dispositivi/PLC; UA significa Unified Architecture e questo vuole proprio indicare che l'idea è quella di costruire un protocollo che permetta la comunicazione tra tutti i tipi di controllori esistenti, creando uno standard capace di far dialogare anche macchine con architetture differenti. Il problema principale di OPC è che permetteva di scambiare dati solo di macchine con piattaforma basata su Windows. Con OPC-UA l'intenzione è quella di ampliare la platea di comunicazione a macchine basate anche su Linux, Mac, Android e qualsivoglia altra piattaforma o tecnologia (ad esempio il cloud) su cui è costruita una macchina. La comunicazione è quindi garantita tra tutte le macchine esistenti a patto che queste implementino il protocollo di comunicazione standard OPC-UA. Tra le caratteristiche del protocollo si possono trovare politiche di sicurezza (criptazione, controllo degli accessi e auditing), estensibilità (possibilità di

aggiungere nuove caratteristiche al protocollo senza penalizzare le applicazioni esistenti), equivalenza funzionale (per comunicare retroattivamente con OPC Classic) e, elemento di maggiore interesse in questa tesi, completa modellazione delle informazioni.

Questa modellazione trova la sua maggiore espressione nello *spazio di indirizzamento*: una collezione di informazioni che un *server* rende visibili a un *client*. L'elemento base di tale concetto è il *nodo* che può rappresentare sia un oggetto che un tipo di oggetto; un nodo è caratterizzato da vari *attributi*, alcuni comuni a tutti i *nodi*, altri specifici a seconda del tipo di *nodo*. Un *server* può quindi esporre una serie di *nodi*/oggetti organizzati in maniera gerarchica (lo spazio di indirizzamento) che un *client* può dinamicamente navigare; inoltre può offrire la possibilità di visionare non solo i dati correnti ma anche gli storici dei dati. I *nodi* all'interno dello spazio di indirizzamento sono individuabili grazie all'attributo *browse name*: la composizione dei *browse name* dalla radice della gerarchia al *nodo* desiderato permette l'individuazione univoca di tale *nodo*/oggetto. Altri *attributi* di un *nodo* possono essere il *display name*, la *descrizione*, il *livello di accesso* (lettura e/o scrittura), *l'identificativo del nodo* e tanti altri. A seconda del tipo di *nodo* da rappresentare, alcuni *attributi* sono obbligatori, altri sono opzionali mentre altri ancora non sono previsti (come ad esempio l'attributo *valore* per un nodo di tipologia *tipo di oggetto*).

Gli oggetti possono essere semplici o complessi: questi ultimi sono composti, oltre che dagli *attributi*, da un insieme di ulteriori *nodi* che possono essere identificati come *variabili* e *metodi*.

Le *variabili* sono utilizzate per rappresentare dei valori e possono essere suddivise in *proprietà* e *variabili dati*; questi differiscono nella tipologia di dato da rappresentare e nella possibilità o meno di contenere altre *variabili*. Le *proprietà*

rappresentano le caratteristiche del *nodo* (mentre gli *attributi* sono più metadati). Le *variabili dati* rappresentano invece il contenuto effettivo dell'oggetto: un oggetto edificio ha come *variabili* dati i mattoni, mentre le *proprietà* sono la data o la durata di costruzione.

I *metodi* invece sono funzioni "leggere", analogamente ai metodi di una classe nella programmazione orientata agli oggetti, sono associati a un oggetto (o a un tipo) proprietario di tale *metodo*. Un *client* può invocare un *metodo* che il *server* esegue restituendo poi al *client* il risultato; l'invocazione può prevedere sia parametri di ingresso che parametri di uscita. Affinché un *client* possa invocare un metodo deve conoscere sia il metodo specifico, sia l'oggetto cui esso appartiene. I *metodi* possono influenzare lo stato dell'oggetto cui appartengono ma non hanno stato proprio.

Oltre al modello *client server*, OPC UA offre anche un modello *publisher subscriber* (questo non verrà analizzato direttamente in questa tesi ma potrebbe offrire vantaggi differenti rispetto al modello client server). Nei più recenti sviluppi del protocollo OPC UA, alcune funzionalità tipiche del modello *publisher subscribe* sono state inserite all'interno del modello client server, dando la possibilità a un server di inviare messaggi sulla rete nei confronti di uno o più processi, tendenzialmente client, in ascolto.

Ulteriori approfondimenti e spiegazioni dettagliate del protocollo di comunicazione OPC-UA e di ciò che esso contiene non sono inseriti in questa breve introduzione in quanto l'obiettivo non è quello di creare una trattazione sul protocollo (tante già ne sono state fatte) ma quello di fornire le basi minime per comprendere ciò che seguirà nei prossimi capitoli, senza rischiare di dilungarsi troppo perdendo di vista il cuore del progetto di tesi.

GE Digital – iFIX

iFIX è una soluzione SCADA e per la gestione delle informazioni in tempo reale basato su sistema operativo Windows prodotto di GE Digital. Si occupa di raccolta dati d'impianto, gestione allarmi e distribuzione di tali informazioni a utenti e macchine collegate in rete. Il suo funzionamento può appunto essere diviso nella parte di SCADA (quando si occupa della gestione dati) e parte HMI (quando si guarda alle interfacce che permette di creare per gestire un impianto).

La relazione tra HMI e SCADA è spesso definita come client - server: lo SCADA (il server) contiene i dati di processo e ha il ruolo di raccogliere e aggiornare le informazioni degli elementi cui è connesso; l'HMI (il client) richiede al server dati e li visualizza attraverso l'interfaccia grafica (GUI) senza memorizzarli ma interagendo con i dati sul server in tempo reale.

Grazie all'utilizzo di protocolli standard per la comunicazione sia di rete che tra processi e alle molteplici possibilità di accesso ai database, iFIX può essere integrato con la maggior parte dei prodotti industriali disponibili sul mercato. Ad esempio, iFIX supporta la comunicazione di rete tramite il protocollo TCP/IP, la comunicazione con sistemi industriali tramite protocollo OPC (client e server), comunicazioni su database tramite tecnologie ADO, ODBC e OLEDB, e anche la possibilità di eseguire codice VBA per fornire interoperabilità tra i processi.

Sul nodo in cui è installato lo SCADA, solitamente sono installati anche i driver di I/O e i server OPC ognuno dei quali riserva uno spazio di memoria per memorizzare i valori correnti; per ognuno di questi elementi (driver e server) è memorizzata sullo SCADA una DIT (Driver Image Table), ovvero una tabella in cui sono contenute le immagini degli attuali valori dell'hardware (e il relativo indirizzo) che sono esposti dai driver sotto forma di record aggiornati periodicamente tramite polling.

Affinché le informazioni presenti sulle DIT siano effettivamente utili, iFIX prevede al suo interno un PDB (Process DataBase) in cui queste informazioni sono organizzate sotto forma di tag con un nome univoco, composte da vari campi quali la descrizione, l'unità di misura e tante altre (ci sono diversi tipi di tag con campi differenti, utili per descrivere al meglio ogni elemento disponibile). GE Digital offre una applicazione chiamata DataBase Manager per la gestione e il monitoraggio del PDB.

All'interno di iFIX è presente un processo che permette di mantenere il PDB aggiornato con i valori presenti sulle DIT: questo processo è nominato SAC (Scan, Alarm and Control).

Il client per poter visualizzare un valore dovrà riferirsi a esso attraverso un indirizzo che ne indica il percorso tramite dot notation: <server>.<nodo>.<tag>.<campo>; tramite un'apposita HMI il client potrà monitorare in tempo reale cosa avviene nell'impianto supervisionato.

Inoltre, iFIX permette di mantenere lo storico dei valori in un server specializzato chiamato Historian, utile per reportistica e analisi dati.

L'effettiva HMI viene creata in un framework di design sotto forma di *picture* alla quale vengono aggiunti vari oggetti grafici prestabiliti. È inoltre possibile arricchire tale interfaccia scrivendo del codice in linguaggio VBA in grado di interagire con gli oggetti appartenenti alla picture stessa o a un'altra picture aperta. Quando l'interfaccia è pronta è possibile passare a un ambiente di run-time che mostra effettivamente i valori desiderati eseguendo, ove presente, il codice scritto per arricchire le funzionalità dell'HMI (ad esempio l'evento scatenato alla pressione di un pulsante).

Altri strumenti utilizzati

Oltre al protocollo OPC-UA e allo SCADA iFIX, verranno utilizzati un gateway (IGS) e un database manager (SSMS); come linguaggi di programmazione per sviluppare il progetto si utilizzeranno C#, xml, SQL e VBA.

IGS

Per far interagire iFIX con i server si utilizzerà IGS Configurator, applicazione facente parte della suite di GE Digital ma in realtà creato da kepware. IGS è acronimo di Industrial Gatway Server e serve per poter visualizzare in tempo reale tutte i valori esposti da un server e il loro indirizzo. IGS permette di creare canali di connessione con i server (verranno utilizzati canali che implementano un client OPC-UA, però sono supportati molti altri protocolli) al cui interno vengono creati dei device per connettersi effettivamente al server e definire il periodo di scansione del server, la politica di aggiornamento (questa sarà fondamentale e prevede come possibili politiche quella basata su eccezione e quella basata su polling). Definito il canale è possibile importare come tag i campi esposti dal server che verranno visualizzati in una struttura ad albero. IGS dispone infine di un quickclient che permette di vedere i dati in tempo reale.

Nella comunicazione con iFIX, per IGS non sono previste DIT ma il collegamento è diretto sul PDB.

SSMS

SQL Server Manager Studio è un software di utilità per la gestione delle basi di dati. Al suo interno verranno create delle tabelle contenenti diverse informazioni utilizzate per comunicare dati tra i diversi processi in vari momenti dell'evoluzione dell'applicazione.

Questo database non sostituisce il Process DataBase di iFIX ma lavora in parallelo a esso in quanto composti da elementi completamente differenti (tag da un lato, qualunque tupla dall'altro) e quindi ricopriranno ruoli non ridondanti.

Capitolo 3

Il progetto

L'idea di riferimento del progetto è quella di riuscire a sfruttare le potenzialità del protocollo OPC-UA in modo da poter creare un'interfaccia grafica unificata che possa adattarsi ai PLC di qualsiasi macchina a cui ci si voglia collegare. L'obiettivo è quello di sfruttare la forte strutturazione offerta da OPC-UA affinché l'interfaccia grafica possa autogenerarsi navigando ed esplorando la struttura dati offerta dal server (il PLC) e riportandola all'utilizzatore finale.

Questo sicuramente sarà utile per poter avere un'unica soluzione in grado di adattarsi a qualunque linea richiesta dal cliente; il vero vantaggio risulterà nella misura in cui una qualsiasi linea prodotta da IMA è composta da varie macchine, ciascuna con il proprio PLC: in questo modo la stessa interfaccia grafica potrà soddisfare tutte le macchine della linea, senza quindi la necessità di creare un'HMI specifica per ogni macchina prodotta (andandole poi a comporre tra loro per creare quella finale dell'intera linea).

Oltre all'interfaccia grafica creata tramite iFIX, verrà realizzato un client (che implementi il protocollo OPC-UA) per sopperire alle mancanze del client che può offrire il gateway IGS nella versione V6.8.839.0 (quella attualmente a disposizione dell'ufficio tecnico elettrico HMI-SCADA della divisione IMA Life). In particolare, IGS non supporta la possibilità che il protocollo offre di creare delle variabili numeriche con associate le indicazioni sull'unità di misura (EUInformation) e un range di valori

che una variabile può assumere (Range); inoltre anche la possibilità di definire degli enumerativi (Enum) non è supportata da questa versione di software. In realtà guardando le specifiche tecniche sul sito di Kepware (il reale produttore di questo software che viene poi venduto a GE Digital per inserirlo nella suite di iFIX) si può notare che il supporto a Enum, Range e EUInformation è stato aggiunto a partire dalla versione V6.9.572.0.

Un'altra specifica di OPC-UA molto importante che non è supportata da IGS è la possibilità da parte del client di poter vedere e invocare i metodi offerti dal server. Questa possibilità del protocollo è molto importante e utile in quanto non richiede più una grossa sovrastruttura per poter richiamare procedure remote tra i processi che sono in gioco.

Per raggiungere l'obiettivo, il client OPC-UA che verrà generato e IGS dovranno comunicare con un server OPC-UA che simuli un PLC. Anche la progettazione e realizzazione di questo server saranno parte del progetto della tesi.

Architettura di riferimento

Per poter raggiungere gli obiettivi prefissati, il progetto di tesi sarà quindi strutturato in tre macro-sezioni:

- il server OPC-UA
- il client OPC-UA
- l'HMI.

Il server ha il compito di simulare un PLC e offrire all'esterno la sua struttura rispettando il mapping offerto da OPC-UA: una struttura ad albero con nodi, variabili, metodi, etc.

Il client ha il compito di navigare lo spazio di indirizzamento del server per memorizzarne la struttura dati, parallelamente alla capacità di effettuare le chiamate ai metodi del server.

L'HMI ha il compito, partendo dalla struttura memorizzata dal client, di creare l'interfaccia specifica relativa al server e, successivamente, attivare l'invocazione remota dei metodi.

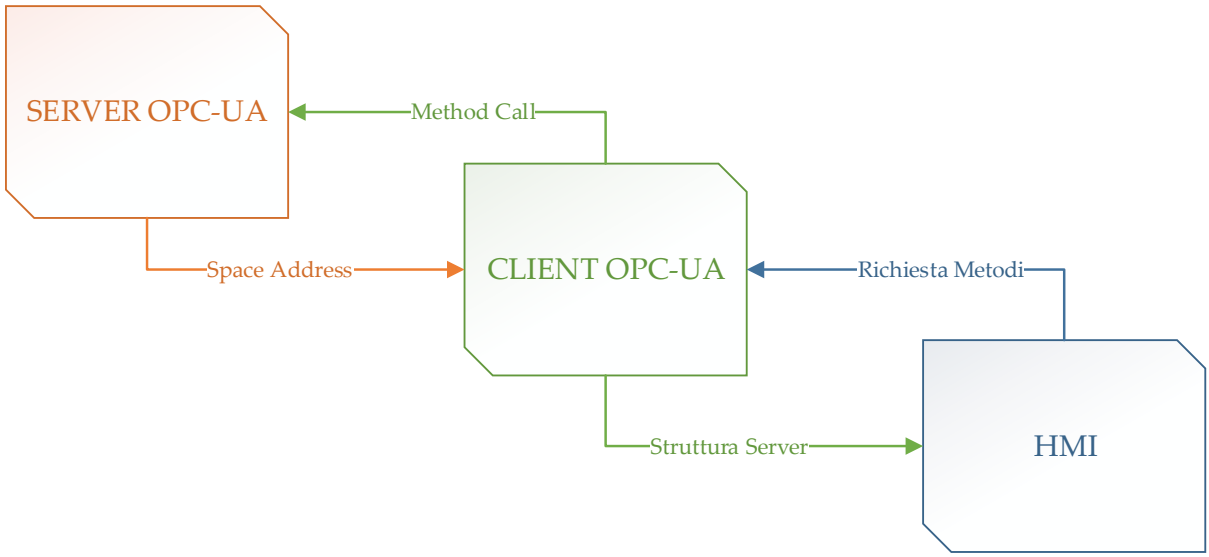

Immagine 4: struttura generica del progetto

L'implementazione della relazione tra client ed HMI può essere realizzata secondo diversi schemi. Il termine di confronto tra questi è la modalità in cui i due elementi comunicano:

- *Offline*
  Il client, scritto in codice C#, partendo dall'esplorazione dello spazio di indirizzamento del server, genera direttamente del codice in linguaggio VB corrispondente alla struttura del server; questo codice viene fornito a iFIX che genera le pagine grafiche dell'HMI corrispondenti.

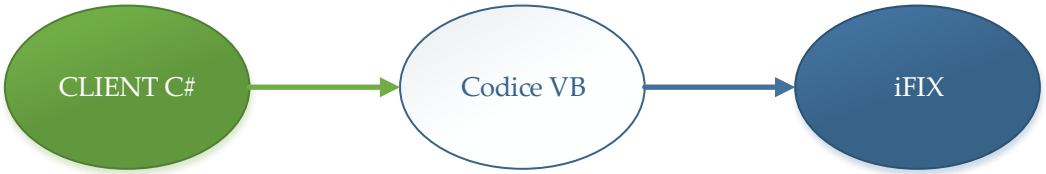

Immagine 5: modello offline communication

Questo modello prevede tale passaggio di informazioni "offline" poiché iFIX, avendo una netta separazione tra design e run-time, non permette la modifica del codice delle pagine direttamente dal run-time (bisogna passare

sempre dal design). Per questo motivo si è valutata una soluzione differente, chiamata "online", in cui il cambiamento non comporta un passaggio forzato al design.

- *Online*

  L'HMI non viene generata da un codice scritto dal client ma è costituita da una serie di elementi grafici generici che vengono mostrati, modificati e nascosti tramite un opportuno codice VBA, rispecchiando la struttura dati fornita dal client. Questa soluzione è dinamica: entrambe le macro-sezioni sono a run-time e la modifica della struttura viene riportata senza dover riavviare dell'HMI. Questo modello ha due possibili implementazioni:

  - *Midpoint-through Communication*

    La comunicazione tra il client e iFIX si appoggia a un midpoint, in particolare a un database, sul quale il client scrive la struttura del server; iFIX grazie al codice VBA legge la struttura riportandola a livello grafico.

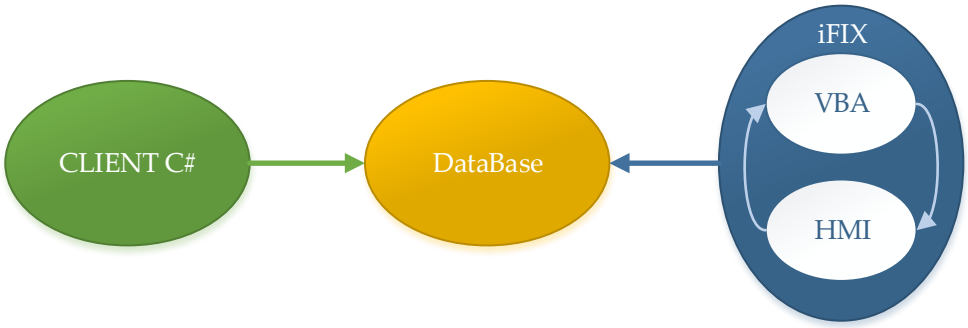

Immagine 6: modello online midpoint through communication

  - *Direct (web) Communication*

    La comunicazione tra il client e iFIX avviene in maniera diretta: la struttura dello spazio di indirizzamento del server viene comunicata dal client via rete (ad esempio tramite web api) a iFIX che mostra graficamente a run-time le informazioni

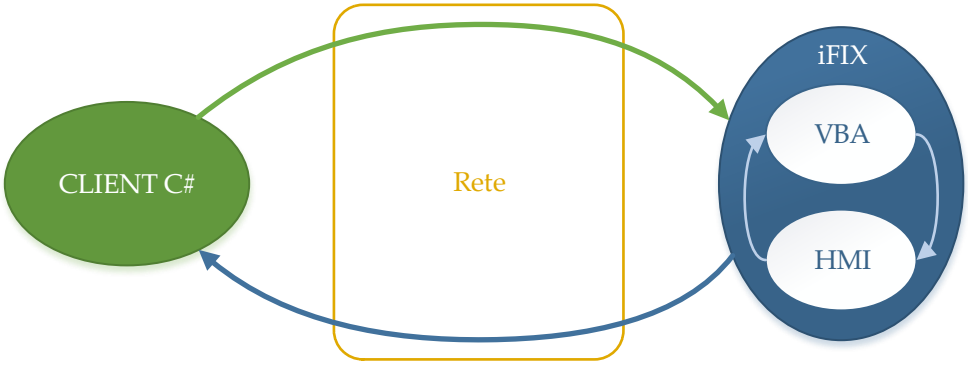

Immagine 7: modello online direct communication

La scelta implementativa ricade sul modello Online Midpoint-through Communication in quanto è leggermente più agile lato VBA, più stabile e più strutturata.

Il modello finale, dunque, si svilupperà sulla seguente struttura:

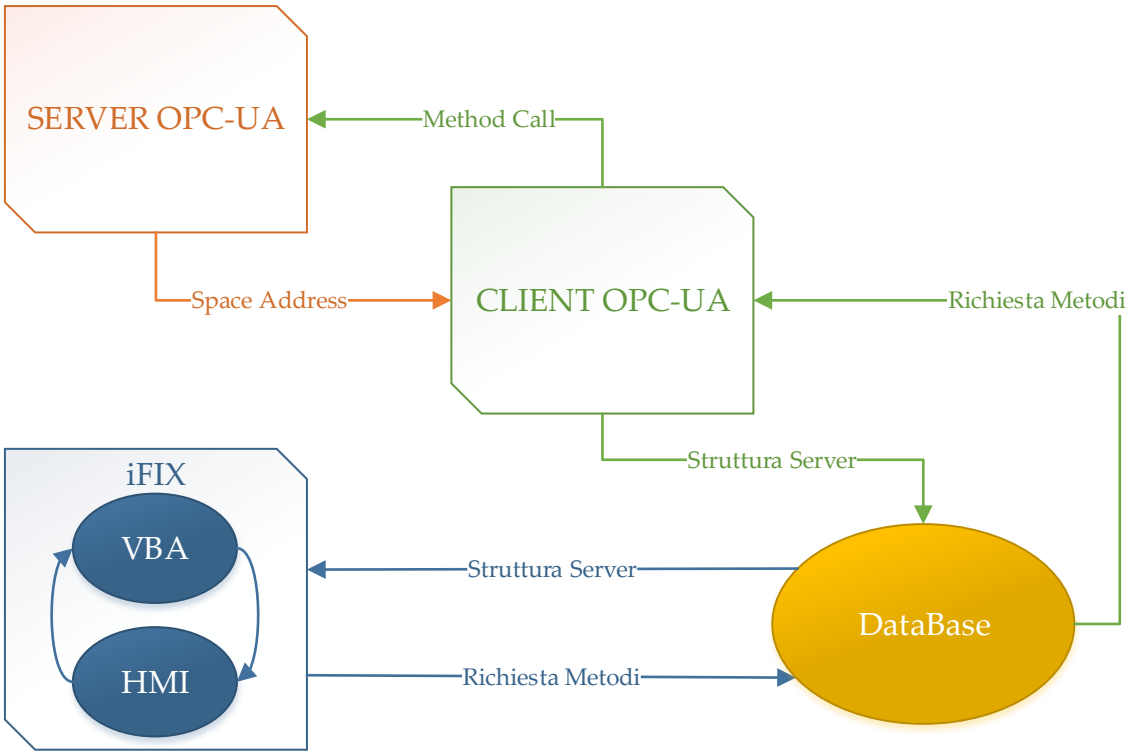

Immagine 8: modello del progetto finale

L'implementazione di questo modello verrà analizzata nei prossimi capitoli, congiuntamente a ulteriori scelte relative a situazioni specifiche.

Capitolo 4

Implementazione di OPC-UA

Sviluppo del progetto

Al fine di raggiungere gli obiettivi prefissati è necessario in primo luogo realizzare il server secondo il protocollo OPC-UA, seguito poi dal client OPC-UA. In ultima istanza sarà realizzata l'HMI.

Essendo tutto questo un ambito da esplorare, si è scelto di applicare una sorta di metodologia agile: il progetto è stato sviluppato per step progressivi, ciascuno focalizzato su una delle tre macro-sezioni e avente una sempre maggiore complessità. Ogni volta che uno step era completo, avveniva un confronto con il responsabile per delineare modifiche per lo step completato e caratteristiche di quello successivo. L'insieme degli step può essere diviso in quattro fasi, ciascuna delle quali è composta dallo sviluppo a un determinato livello di complessità delle macro-sezioni. Lo sviluppo delle fasi è da considerarsi sequenziale, mentre lo sviluppo degli step all'interno delle varie fasi è da considerarsi agile.

Per rappresentare lo sviluppo del progetto, oltre alle quattro fasi implementative possiamo considerare una fase preliminare di studio del protocollo e degli strumenti a disposizione, chiamata fase 0. Questa ovviamente prosegue, in maniera accessoria, anche durante tutto il resto dello sviluppo sia per via dei continui aggiornamenti alle librerie standard di OPC-UA sia per la necessità di approfondire la

conoscenza degli strumenti utilizzati per poterne sfruttare al massimo le potenzialità e raggiungere i risultati desiderati.

È possibile rappresentare lo sviluppo secondo un diagramma di Gant in cui sono riportate solo le WBS (work breakdown structures) rappresentanti le varie fasi del progetto e come indicatore temporale i giorni impiegati per portare a termine ciascuna fase. La suddivisione in step non è riportata in quanto sono essere considerati quasi paralleli all'interno di ogni fase; inoltre non risulta particolarmente rappresentativa l'analisi temporale dei singoli step.

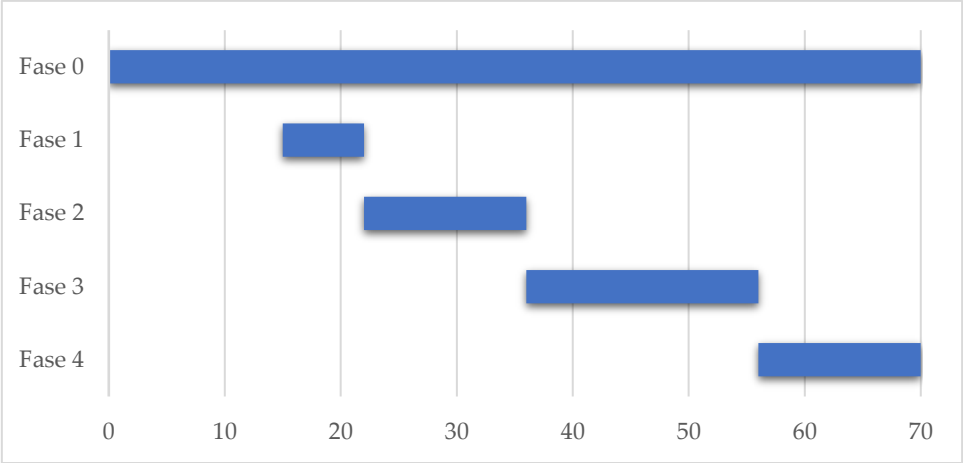

Immagine 9: diagramma di Gant del progetto

Analizziamo ora le prime tre fasi, quelle preparatorie al progetto finale, mostrandone la relazione tra gli step. L'ultima fase invece sarà affrontata in un capitolo separato (il prossimo) in quanto è quella di effettivo interesse della tesi.

Fase 1 – Primo approccio a OPC-UA

La fase 1 si concentra solamente su server e client OPC-UA, tralasciando l'HMI. Le due macro-sezioni nascono partendo da quelle fornite direttamente dalla OPC Foundation, reperite nella repository GitHub della fondazione nel progetto

denominato UA-.NETStandard. Questo progetto contiene una implementazione del protocollo mostrante tutti i possibili tipi di dati esposti e alcuni esempi specifici.

Lo sviluppo degli step segue il seguente schema

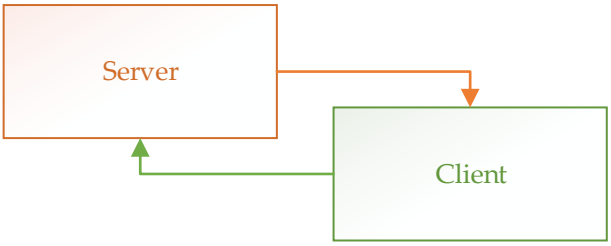

Immagine 10: step della fase 1

Il primo passo dentro al mondo OPC-UA è stato fatto attivando il Quickstart.ReferenceServer disponibile nell'SDK e collegandolo al Quickstart.ReferenceClient, anch'esso presente nell'SDK. Dal client è possibile navigare la struttura ad albero esposta dal server. Esplorando l'albero si possono notare, tra i vari nodi, diversi tipi di variabili scalari, rappresentanti semplici valori numerici. Oltre a queste sono presenti anche dati più complessi come le variabili analogiche e quelle discrete. Le variabili analogiche sono composte da un campo *Value*, corrispondente al valore effettivo della variabile, e da una serie di altri campi quali *InstrumentRange*, *EURange*, *EngineeringUnits*, *Definition* e *ValuePrecision* utili per meglio definire tutte le proprietà della variabile (riscontrando corrispondenza con il modello teorico di OPC-UA). Le variabili discrete invece possono essere principalmente di tre tipi: *TwoStateDiscreteItemVariable* (rappresentante un valore binario e avente proprietà *FalseState*, *TrueState*, *Definition* e *ValuePrecision*), *MultiStateDiscreteItemVariable* (rappresentante un valore discreto associato a un valore intero a partire da zero e avente proprietà *Definition*, *ValuePrecision* e *EnumString*) e *MultiStateValueDiscreteItemVariable* rappresentante un valore discreto associato a un valore intero arbitrario e avente proprietà *Definition*, *ValuePrecision*, *ValueAsText* e *EnumValue*).

Come prima sperimentazione si è rivolta l'attenzione al server, in particolare analizzando quello della reference e andando ad aggiungere variabili di vari tipi, valorizzandone i vari campi, modificandone i diritti di accesso direttamente dal codice C#; in questo modo si è riusciti a creare una struttura dati personalizzata che è risultata navigabile dal client.

Di seguito mostriamo come appare lo spazio di indirizzamento del server mostrato sul client (come è visibile, sono stati espansi solo alcuni nodi).

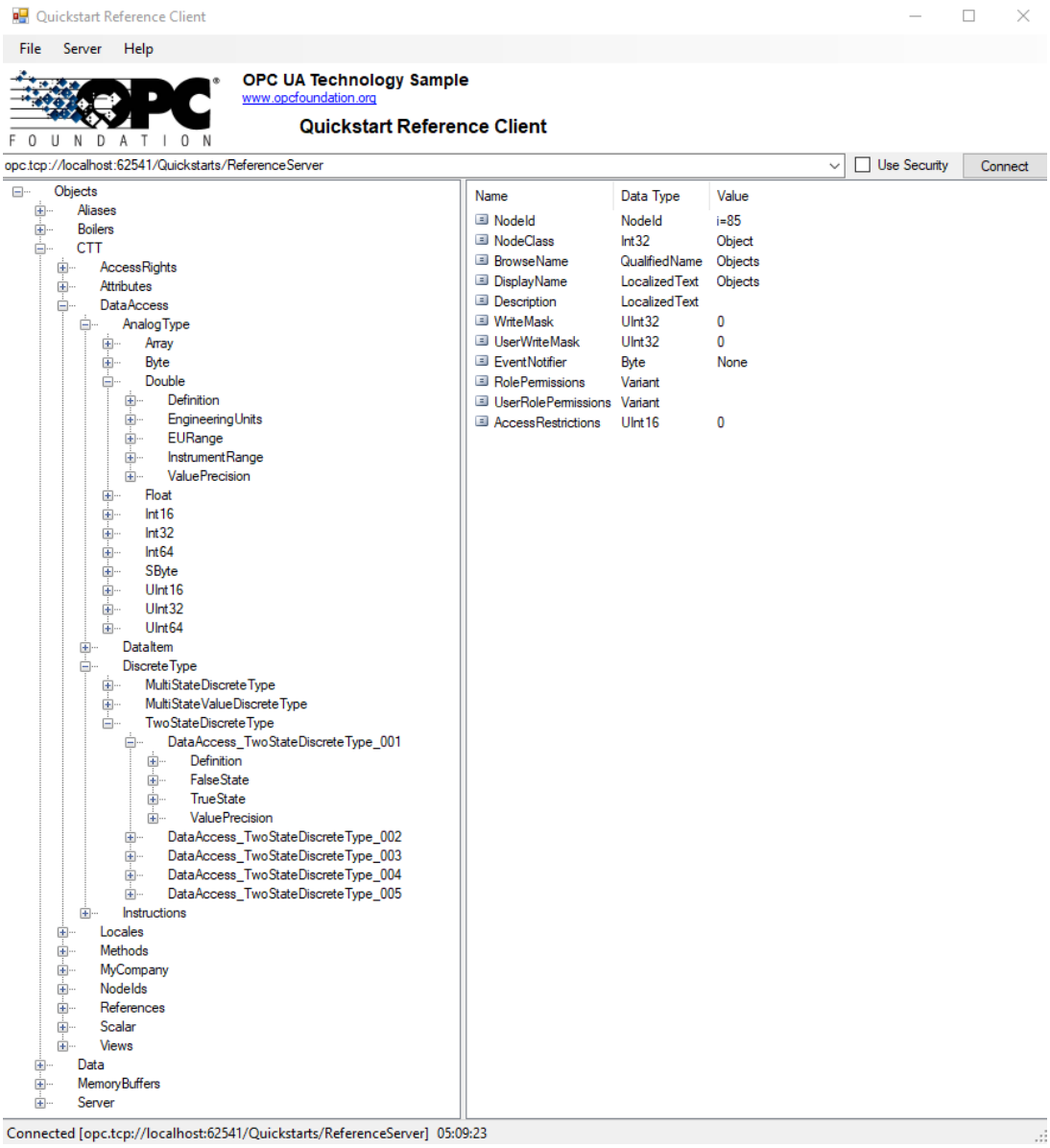

Immagine 11: spazio di indirizzamento del server della fase 1

Il ReferenceServer è stato poi collegato anche a IGS: sul gateway è stato creato un nuovo Channel di tipo OPC-UA Client ed è stato posto in ascolto all'indirizzo IP della macchina in cui è presente il server, sulla porta indicata dal server stesso. In questo canale è stato creato un Device su cui sono state importate tutte le possibili tag esposte dal server (la struttura ad albero comprendente le variabili e le proprietà). È stato notato che la versione disponibile di IGS non supporta alcuni campi, nello specifico non vengono mostrati (prendiamo in esempio una variabile analogica di tipo double e una per ogni tipo discreto):

```
(…).AnalogType.Double.EngineeringUnits
(…).AnalogType.Double.EURange
(…).AnalogType.Double.InstrumentRange
(…).MultiStateValueDiscreteType.MultistateDiscrete001.EnumValue
```

Questi problemi dovranno essere affrontati e supportati dal nostro futuro client. Come detto prima, l'attenzione si è posta principalmente sul server (il client è stato analizzato ma non particolarmente a fondo); questo perché è necessario avere dapprima un server completo e funzionante per poter implementare un client che lo sfrutti a pieno.

Il discrimine tra questa fase e la successiva è l'assenza di un modello che descriva la struttura del server. Tutto il server (sia la struttura creata che la valorizzazione dei dati) è definito dal codice C#. Nella fase 2 vedremo come costruire un modello di dati per la creazione di un server OPC-UA.

Fase 2 – Creazione di un semplice modello

Nella fase 2 è inserito un importante blocco, ossia la creazione di un modello di dati da cui il server si può generare in maniera più snella e dinamica. Questa possibilità

è fornita da un progetto della OPC Foundation anch'esso reperibile dalla loro repository GitHub al nome di UA-ModelCompiler che permette appunto di definire il modello del server in un file xml e, eseguendo il programma, genera automaticamente una serie di file necessari per il server.

Lo sviluppo degli step segue il seguente schema:

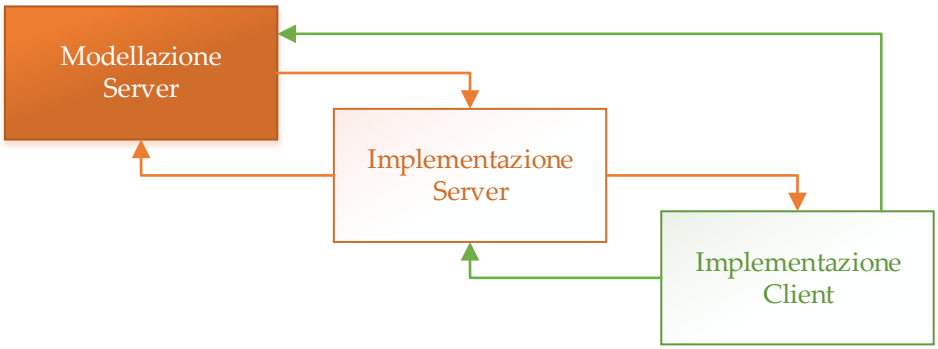

Immagine 12: step della fase 2

Il primo elemento veramente utile all'interno dell'UA-Model Compiler è il file "UA Model Design.xsd", utilizzato nello step Modellazione Server. Questo file fornisce tutte le indicazioni necessarie per poter creare in modo corretto il modello del server. Aprendo questo file all'interno del progetto di Visual Studio in cui si intende sviluppare il server, è possibile scrivere il file che definisce il modello, chiamato ad esempio "OpcUaServerDesignModel.xml", ricevendo suggerimenti, intellisense e rilevazione di errori a design time grazie al file xsd, velocizzando notevolmente il processo di stesura del codice. Questo file di modello comprende inizialmente l'indicazione delle librerie a cui fare riferimento in cui sono definiti i tipi di dati e altre indicazioni per la stesura del modello; successivamente sono definiti i namespace cui fa riferimento il server. Inizia poi la vera e propria definizione del modello tramite la creazione di tipi specifici customizzati, variabili, proprietà e istanze di oggetti che andranno a comporre il modello.

Affinché il progetto di Visual Studio possa implementare effettivamente il protocollo OPC-UA, è necessario installare le librerie ufficiali fornite dalla fondazione e disponibili sul framework .NET; per effettuare questo passaggio è sufficiente usare il NuGet Package Manager digitando il comando

```
Install-Package OPCFoundation.NETStandard.Opc.Ua
```

e l'ambiente sarà pronto per lo sviluppo del server.

Terminata la stesura del design model in formato xml, si passa allo step Implementazione Server. Per effettuare questo passaggio viene chiamato in causa l'effettivo funzionamento dell'UA-ModelCompiler: andando nella cartella in cui è presente l'eseguibile "Opc.Ua.ModelCompiler.exe", lo si esegue da PowerShell con una serie di parametri in modo tale da far generare molti dei file necessari alla creazione del server. Facendo riferimento al file del modello come <DesignModel>.xml e il namespace come <Namespace>, i file generati sono:

- <DesignModel>.csv → contiene tutti gli elementi che definiscono lo spazio di indirizzamento del server e i relativi identificativi numerici;
- <Namespace>.Classes.cs → contiene le definizioni delle classi degli oggetti del modello in linguaggio C#;
- <Namespace>.NodesId.csv → contiene tutti i tipi definiti all'interno del modello con i relativi identificativi numerici e tipi;
- <Namespace>.NodeSet.xml e <Namespace>.NodeSet2.xml → questi due file contengono l'insieme dei nodi appartenenti al server;
- <Namespace>.PredefinedNodes.uanodes → verrà utilizzato dal server per generare l'albero contenente la sua struttura, lo spazio di indirizzamento;
- <Namespace>.PredefinedNodes.xml → contiene la struttura xml estesa dello spazio di indirizzamento del server;
- <Namespace>.Types.bsd e <Namespace>.Types.xsd → altri due file di utilità.

Successivamente è necessario affiancare a questi un nuovo file chiamato <Namespace>.Config.xml contenente tutte le impostazioni di configurazione del server come i certificati di sicurezza, i tempi di timeout, l'indirizzo IP e la porta su cui verrà esposto il server, gli endpoints offerti e tante altre informazioni.

Affinché il server possa effettivamente girare in maniera corretta a run-time, è necessario impostare nelle proprietà dei file <Namespace>.PredefinedNodes.uanodes e <Namespace>.Config.xml l'opzione "Copy if newer" per il campo "Copy to Output Directory" in quanto serviranno come riferimento di tutte le configurazioni del server e per la definizione del suo spazio di indirizzamento a ogni esecuzione.

A questo punto è possibile passare alla scrittura del vero e proprio codice C# che eseguirà il server. Questo sarà suddiviso in tre diversi file:

- Program.cs

  Questo file serve principalmente per lanciare il progetto/l'applicazione. Definisce il nome che sarà analogo al namespace e carica il file di config.
- Server.cs

  Definisce il server e lo costruisce richiamando il node manager.
- NodeManager.cs

  Questo è il cuore del server: crea effettivamente lo spazio di indirizzamento richiamando il file <Namespace>.PredefinedNodes.uanodes, valorizza le variabili e implementa i metodi; in questo file si possono anche aggiungere variabili e oggetti non definiti nel modello.

  Sarà questo il file su cui si svolgerà il 65% del lavoro (il restante possiamo assegnarlo per il 30% alla modellazione e l'ultimo 5% a tutto il resto, operazioni accessorie e abbastanza sequenziali).

In questa fase, oltre al server, è stato creato anche un client. Questo, rispetto al server, è molto più semplice essendo composto da un unico file; inizialmente viene creata la sessione con il server cercando di raggiungere l'indirizzo (ip:porta) su cui è esposto e successivamente viene stabilita l'effettiva connessione con tale server. Proseguendo nell'implementazione del client possiamo trovare l'esplorazione dell'albero rappresentante lo spazio di indirizzamento del server grazie al metodo Browse invocabile sulla sessione creata. Inoltre è possibile leggere e scrivere (ove i diritti di accesso lo permettono) i valori delle variabili riferendo al loro namespace e identificativo del nodo in cui risiedono, possibile grazie ai metodi Read e Write invocabili sempre sulla sessione. Ultimo elemento presente nel client è la possibilità di invocare i metodi grazie al metodo Call invocabile sulla sessione, passando come parametri namespace e identificativo del nodo del metodo che si vuole invocare e del nodo dell'elemento su cui lo si vuole invocare. Ovviamente anche nel client è necessario installare le librerie di OPC-UA tramite il NuGet Package Manager.

In questa fase sono state implementate diverse versioni di serve con complessità crescente, mentre per il client si è utilizzato solo una versione base. Per mostrare le diverse complessità del server sfruttiamo il client della reference poiché ha un aspetto più piacevole rispetto alle stampe sulla console del nostro client.

Di seguito sono mostrate le due versioni limite del server (la più semplice e la più complessa).

In particolare, la prima versione del server ha una struttura molto semplice hce rappresenta una linea produttiva contenente una sola macchina avente due sensori e un motore con una variabile e due metodi.

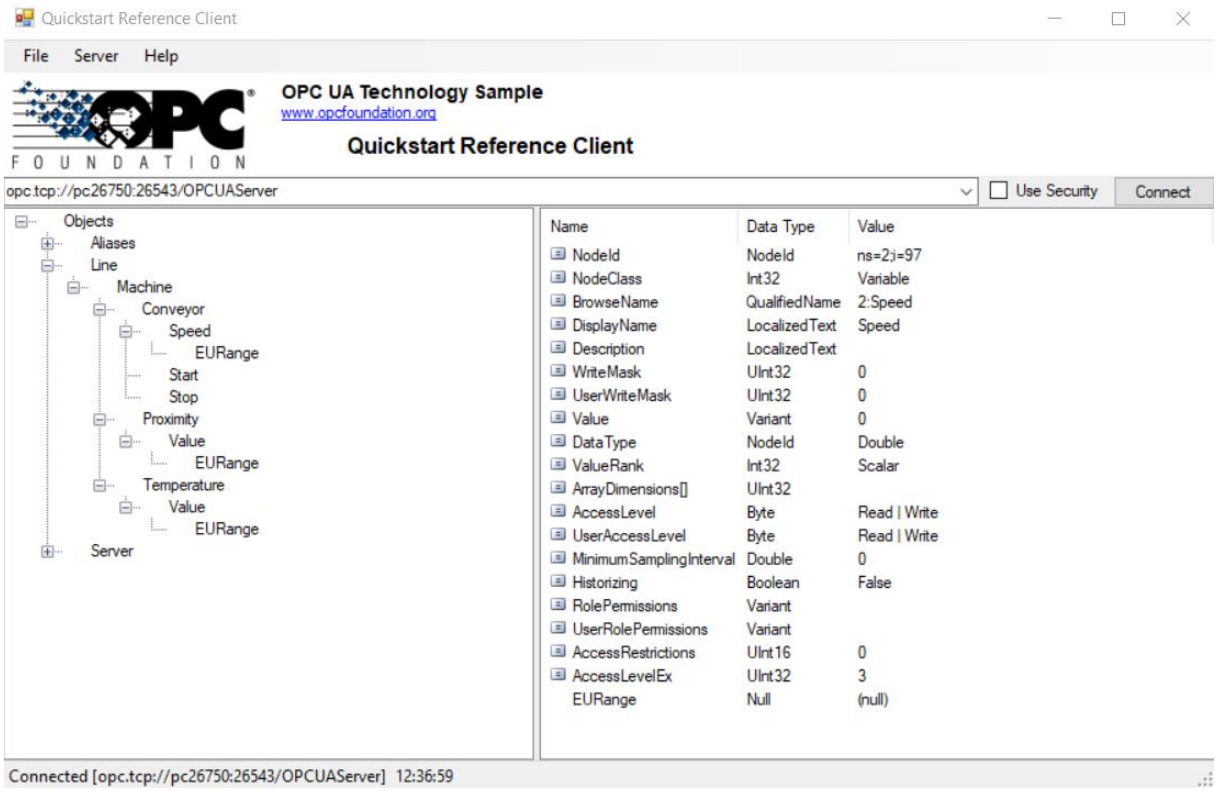

Immagine 13: prima versione del server della fase 2

Per quanto riguarda l'ultima versione, invece, la struttura è nettamente più complessa. La simulazione rappresenta un intero reparto produttivo di un'azienda: questo reparto è stato suddiviso in due aree (Area Completa e Area Composta); ciascuna area è composta da più linee (i tipi di line sono tre: Zeus, Ares e Poseidon) e ogni linea è composta da più macchine (la Zeus è composta da una Washer, una Filler e una Capper; la Ares è composta da una Filler e una Capper; la Poseidon è composta da due Washer). Infine, ciascuna macchina è composta da una serie di motori e sensori, aventi a loro volta variabili, proprietà e metodi propri.

Nella seguente immagine è possibile vedere parte della struttura sopra descritta: i nodi espansi sono solo quelli di una linea Zeus che comprende al suo interno tutte le possibili macchine presenti; i nodi non espansi hanno una struttura esattamente analoga ai nodi omonimi mostrati per la linea Zeus1 appartenente all'Area Completa.

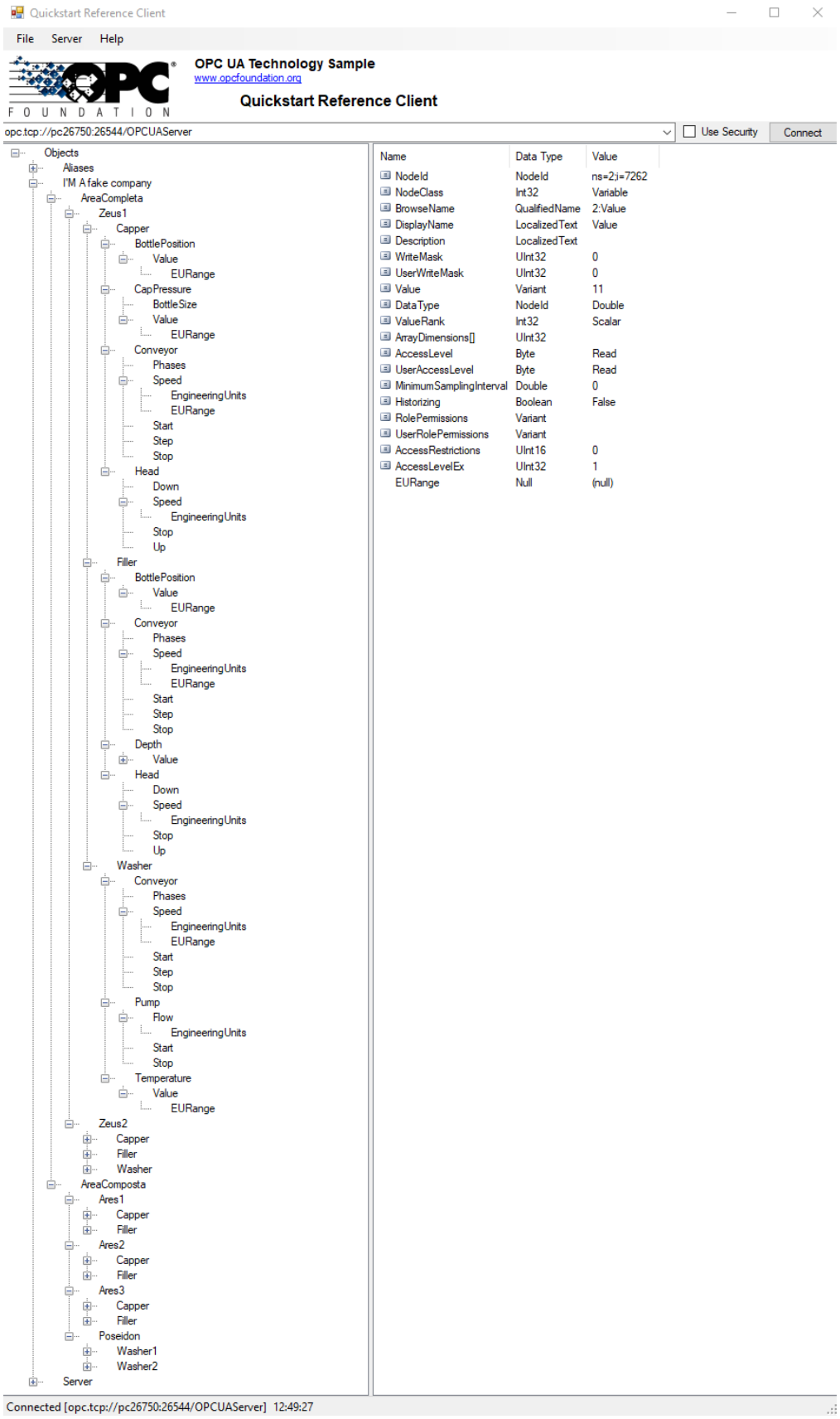

Immagine 14: seconda versione del server della fase 2

Il discrimine tra questa fase e la successiva è la presenza di una HMI che raffiguri il modello del server a un utilizzatore finale. Sarà implementato un nuovo server simile a una reale linea IMA, verrà ampliato il client per implementare anche il passaggio di informazioni verso iFIX e verrà appunto creata l'interfaccia grafica generica valida per un qualsiasi server.

Fase 3 – Impostazione di un prototipo più complesso con HMI

Nella fase 3 l'attenzione è posta maggiormente sulla realizzazione dell'interfaccia grafica tramite iFIX. Questo strumento permette di generare oggetti grafici ed eseguire script di codice, scritto in linguaggio VBA, per modificare gli oggetti rappresentati o per eseguire funzioni allo scatenarsi di alcuni eventi come il click su un oggetto. Per comunicare lo space address del server, il client scriverà su una tabella SQL le informazioni necessarie all'HMI per poter ricreare la struttura e mostrare l'esplorazione del server in modo dinamico.

Lo sviluppo degli step segue lo schema qui riportato:

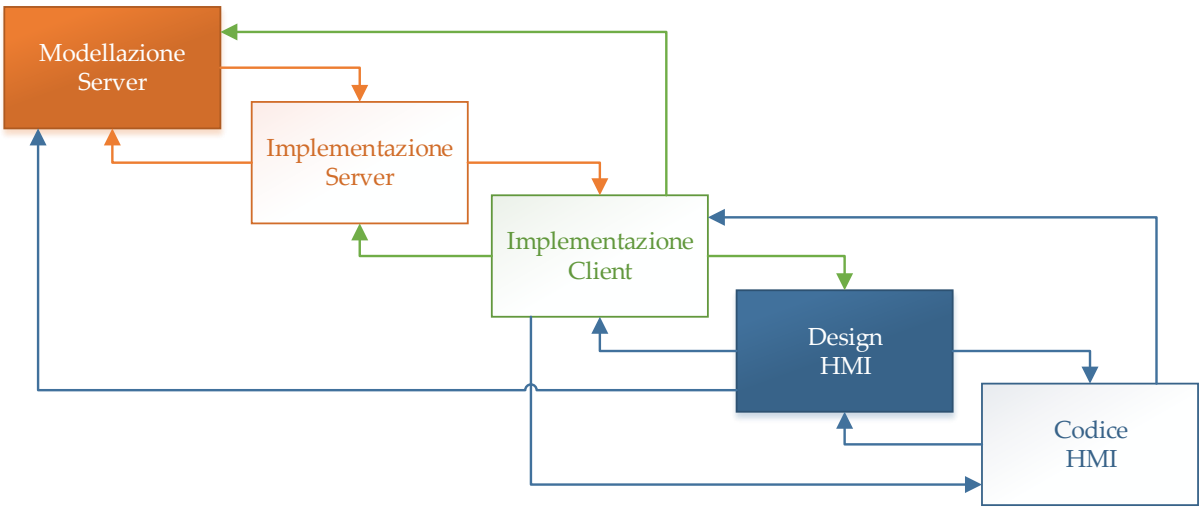

Immagine 15: step della fase 3

La modellazione del server prende spunto da quella più complessa della fase precedente ma cerca di rifarsi a una possibile linea che IMA può fornire a un cliente. Per quanto questo modello sia ancora lontano da una simulazione reale (in quanto una linea non è formata da un solo server ma ogni macchina della linea è governata da un PLC, senza considerare inoltre la reale complessità delle macchine), la gerarchia tra gli elementi rispecchia quella di una vera linea IMA: la linea è formata da più macchine, ciascuna macchina è formata da vari gruppi macchina e ogni gruppo macchina è formato da un insieme di sensori e motori, ognuno dei quali dispone di valori, proprietà e metodi. Questa è la struttura su cui verrà costruita l'interfaccia grafica generica, la quale sarà in grado di adattarsi a qualunque server (che in questo caso rappresenta un'intera linea) che abbia come modello base l'albero linea-macchina-gruppo macchina-sensore/motore.

Nello specifico, il server simulerà la linea chiamata Efas9, costituita dalle seguenti macchine: una Washer (W1), una Tunnel (T1), una Filler (SL1), una Capper (ALU1), una External Washer (E1), una Tray Loader (L1), due Isolatori standard (SN1 e SN2) e un Isolatore MTI (MTI1). La composizione dei gruppi macchina di queste macchine è stata inventata, come anche la composizione in sensori e motori di ciascun gruppo macchina. L'analisi della struttura completa non verrà analizzata in quanto abbastanza complessa (e di conseguenza piuttosto lunga) e di poco interesse in questa fase della trattazione del progetto. L'attenzione è focalizzata maggiormente sui concetti e sulle relazioni tra i vari elementi; parte della struttura verrà analizzata nel prossimo capitolo poiché il progetto finale si basa sulla stessa linea (con qualche miglioramento dal punto di vista della simulazione di una linea reale).

Unico elemento da menzionare in questa analisi (che possiamo definire più concettuale/relazionale) è una variabile che non compare nell'albero di esplorazione ma che appartiene al server generale, nella quale viene memorizzato il numero di

versione attuale del server. Questo servirà per gestire gli aggiornamenti del server lato client e, soprattutto, HMI.

Anche l'implementazione del server migliora nettamente rispetto alla fase 2 nella misura in cui tutte le variabili e tutte le varie proprietà (nello specifico facciamo riferimento ai campi EngineeringUnits ed EURange) vengono valorizzate e i metodi hanno una piccola implementazione che permette di vedere quando questi effettivamente vengono invocati.

Passando al client, abbiamo una modifica più significativa rispetto a quella del server (che certamente è stato modificato e arricchito, ma sostanzialmente non ha grosse funzionalità aggiuntive rispetto a quello della fase 2). Questa riguarda principalmente l'interazione con il database nel quale memorizzare la struttura dati del server. Nel database viene creata una tabella denominata dbo.OPUAMachineConfiguration contenente delle tuple corrispondenti ai vari elementi del server identificati dai seguenti campi:

- *ID*: è l'intero identificativo della tupla; è la chiave primaria
- *parent*: indica l'ID del nodo padre di questo elemento
- *DisplayName*: è il DisplayName del nodo secondo il protocollo OPC-UA
- *BrowseName*: è il BrowseName del nodo secondo il protocollo OPC-UA
- *NodeNamespaceIndex*: è il NamespaceIndex del nodo secondo il protocollo OPC-UA
- *NodeId*: è l'identificativo del nodo all'interno del namespace sopra indicato
- *TypeNamespaceIndex*: è il NamespaceIndex del tipo di nodo
- *TypeDefinitionId*: è l'identificativo del tipo di nodo all'interno del namespace sopra indicato

- *NodeClass*: è la classe del nodo (Object, Varibale o Property)
- *NodeType*: è il tipo specifico del nodo, definito dal client analizzando la classe e la valorizzazione dei campi
- *Value*: rappresenta il valore che assume l'elemento (se è variabile o proprietà)
- *nChildren*: indica il numero di nodi figli del nodo corrente

Oltre a questi elementi viene creata una tupla all'interno della tabella dbo.Settings che serve a memorizzare il numero di versione del server attualmente disponibile sul database; questo valore sarà utile nel momento in cui il client esplorerà il server, leggendo anche la versione attuale del server a cui si è collegato: se la versione memorizzata non è quella aggiornata, il client riscriverà la tabella sul database e l'HMI sarà in grado di adattarsi a tale cambiamento.

Nel momento in cui il vecchio client andava semplicemente a stampare su terminale le informazioni relative al nodo che stava esplorando, il client implementato in questa fase va a creare una istanza della classe DBElement (nata appositamente per questo scopo) valorizzandone in maniera appropriata tutti i campi. Una volta creato questo elemento, esso verrà poi scritto sul database tramite opportuni metodi definiti nella classe DatabaseIO (realizzata per interfacciarsi al database in maniera semplice). Rispetto al vecchio client, avviene inoltre la lettura effettiva di tutti i valori di variabili e delle proprietà che vengono poi inseriti sul database e stampate anche sul terminale all'interno della ricerca esplorativa del server.

L'ultimo punto di aggiornamento del client è la possibilità di invocare metodi da terminale indicando gli identificativi del nodo relativo al metodo che si vuole chiamare e del nodo su cui si vuole chiamare quel metodo (sarebbero necessari anche i namespace ma questi sono individuati direttamente dal client); questi identificativi

sono individuabili poiché anch'essi sono stampati a terminale nell'albero rappresentate lo spazio di indirizzamento del server.

Come nel caso della modellazione del server, anche per questi elementi aggiornati non forniamo ulteriori dettagli in quanto saranno presenti nel prossimo capitolo (con qualche miglioria per adattarsi alle specifiche del progetto finale).

L'unico aspetto implementativo che viene riportato in questo capitolo è un comportamento inspiegabile riscontrato nell'utilizzo della funzione Call per la chiamata dei metodi: presupponendo di avere a disposizione i valori numerici dei NodeId del motore e del metodo rispettivamente nelle variabili `motor` e `method`, la signature del metodo Call prevede due oggetti NodeId, uno per il motore e uno per il metodo. Il problema si focalizza nell'utilizzo dei costruttori di un'oggetto NodeId; i costruttori presi in considerazione sono due (in generale ce ne sono una decina):

```
public NodeId(string value, ushort namespaceIndex);
public NodeId(string text);
```

Con entrambi è possibile eseguire i comandi che creano correttamente i NodeId in maniera analoga; utilizzando però il primo costruttore, all'invocazione della Call si genera un errore che non permette di eseguire tale istruzione. È necessario quindi utilizzare il secondo costruttore al fine di avere una corretta chiamata:

```
NodeId mot = new NodeId("ns=2;i=" + motor.ToString()); // ok
NodeId met = new NodeId("ns=2;i=" + method.ToString()); // ok
session.Call(mot, met); // ok
```

oppure in forma abbreviata:

```
session.Call( new NodeId("ns=2;i=" + motor.ToString()),
              new NodeId("ns=2;i=" + method.ToString()) ); // ok
```

Passiamo infine all'HMI che è stata creata basandosi sul modello linea – macchina – gruppo macchina – motori/sensori. Partendo dal design, sono state create su iFIX tre picture che rappresentano le tre schermate principali: BaseLine, BaseMachine e BaseMachineGroup, ognuna rappresentante gli elementi del livello sottostante.

BaseLine è una schermata che rappresenta tutte le macchine all'interno della linea, fino a un massimo di 15, e in alto presenta il nome della linea. Il codice di questa pagina prevede, all'avvio del run-time, di eseguire una query nei confronti della tabella del database andando a estrapolare tutti gli elementi che hanno il campo *parent* uguale a uno. Questa proprietà identifica appunto le macchine all'interno della linea poiché, lato client, viene inserita come primo elemento la linea con *ID* pari a uno e *parent* uguale a NULL. Di conseguenza tutte le macchine avranno *parent* pari a uno. Una volta ottenute le tuple aventi tale proprietà, il codice va a nascondere le macchine non utili (ad esempio con la Efas9 ne mostra nove e nasconde le restanti sei) e, per le macchine individuate, ne scrive il nome sopra l'elemento grafico rappresentate la macchina. In ambiente run-time, cliccando su una macchina viene chiusa l'attuale picture BaseLine e viene aperta la picture BaseMachine, memorizzando quale macchina è stata selezionata.

BaseMachine è una schermata che rappresenta tutti i gruppi macchina all'interno della macchina selezionata, fino a un massimo di 15. In alto è presente, come prima, il nome della linea e, in aggiunta, il nome della macchina selezionata seguendo uno schema bread-crumb; cliccando sul nome della linea è possibile tornare alla picture BaseLine chiudendo la picture BaseMachine. Inoltre, nella zona inferiore dell'interfaccia, sono presenti due frecce che permettono la navigazione tra le macchine, ricaricando la stessa picture BaseMachine ma riferendosi alla macchina precedente o successiva all'interno della linea. Il codice di questa pagina prevede,

all'apertura di essa, di eseguire una query nei confronti della tabella del database, andando a estrapolare tutti gli elementi che hanno il campo *parent* uguale all'identificativo della macchina attualmente selezionata. Questa proprietà identifica, per costruzione delle informazioni lato client, i gruppi macchina all'interno della macchina. Una volta ottenute le tuple aventi tale proprietà, il codice va a nascondere i gruppi macchina non utili (ad esempio visualizzando la *Washer W1* ne mostra tre e nasconde i restanti dodici) e, per i gruppi macchina individuati, ne scrive il nome sopra l'elemento grafico rappresentante il gruppo macchina. A livello grafico questa picture è molto simile alla precedente: la rappresentazione dei gruppi macchina è leggermente differente rispetto a quella delle macchine e vengono aggiunti i tre elementi rappresentanti la macchina selezionata (in alto), la macchina precedente e la macchina successiva (in basso). In ambiente run-time, cliccando su un gruppo macchina viene chiusa l'attuale picture BaseMachine e viene aperta la picture BaseMachineGroup, memorizzando quale gruppo macchina è stato selezionato; inoltre, come già accennato, cliccando sul nome della linea o sul nome della macchina precedente o successiva è possibile navigare tra gli elementi della linea.

BaseMachineGroup è una schermata che rappresenta tutti i sensori e i motori all'interno del gruppo macchina selezionato, fino a un massimo di 10 elementi per ciascun tipo. In alto sono presenti, come prima, il nome della linea e il nome della macchina; in aggiunta vi è anche il nome del gruppo macchina selezionato, continuando il percorso con schema bread-crumb. Cliccando sul nome della linea è possibile tornare alla picture BaseLine chiudendo la picture BaseMachineGroup mentre cliccando sul nome della macchina è possibile tornare alla picture BaseMachine chiudendo l'attuale BaseMachineGroup. Inoltre, nella zona inferiore dell'interfaccia, sono presenti due frecce che permettono la navigazione tra i gruppi macchina appartenenti alla macchina attualmente esplorata, ricaricando la stessa picture

BaseMachineGroup ma riferendosi al gruppo macchina precedente o successivo. Il codice di questa pagina prevede, all'apertura di essa, l'esecuzione di una query nei confronti della tabella del database, andando a estrapolare tutti gli elementi che hanno il campo *parent* uguale all'identificativo del gruppo macchina attualmente selezionato. Questa proprietà identifica, per costruzione delle informazioni lato client, i sensori e i motori all'interno del gruppo macchina. Una volta ottenute le tuple aventi tale proprietà, il codice va a nascondere i sensori e i motori non utili, esaminando anche il campo *NodeClass* per differenziare i motori dai sensori. In base alle informazioni ricevute, i sensori mostreranno il proprio nome, il nome del loro campo (che può essere numerico o di testo a seconda del tipo di sensore) e il valore che esso assume; se i sensori sono di tipo analogico, vengono mostrate anche l'unità di misura e il range in cui il sensore opera. Per quanto riguarda i motori invece viene rappresentato il nome del motore, il nome del campo numerico al suo interno, il valore di questo, l'unità di misura e il range in cui esso opera; oltre a questi dati, se presente, viene visualizzata anche una proprietà che identifica le fasi che il motore può assumere; infine, sempre all'interno del motore, vengono visualizzati dei bottoni che rappresentano i metodi invocabili su quel motore con i relativi nomi. A livello grafico questa picture è abbastanza differente rispetto alle precedenti: la rappresentazione dei sensori e dei motori è differente e lo è anche il posizionamento (in alto sono rappresentati i sensori, in basso i motori), considerando proprio una diversa rappresentazione dei due diversi elementi. Come nelle picture precedenti, tutto ciò che non presente all'interno del gruppo macchina selezionato viene nascosto. Similarmente alla picture precedente, sono presenti altri quattro elementi interattivi (cliccabili): in alto la linea e la macchina corrente (l'ultimo elemento del menu di navigazione bread-crumb, ovvero il gruppo macchina, non è cliccabile in quanto indica la posizione attuale di esplorazione), in basso il gruppo macchina precedente e quello successivo. In ambiente run-time è possibile appunto navigare all'interno della linea cliccando sul nome della linea, della

macchina, del gruppo macchina successivo o quello precedente; infine cliccando sui metodi è possibile invocarli e, per alcuni elementi, è possibile scrivere direttamente i valori desiderati (ad esempio la Speed di un certo motore).

In questa fase di prototipazione l'invocazione dei metodi non è stata implementata al 100%: cliccando su un metodo viene mostrata una message box contenente i valori di namespace e identificativi dei nodi necessari per invocare il metodo; nell'ultima implementazione del progetto verrà effettivamente implementata l'invocazione dei metodi con i relativi effetti e messaggi di ritorno. Altro discrimine nei confronti della versione finale è l'organizzazione della linea e la relativa gestione a livello client: nella realtà un server rappresenta un solo PLC, quindi la linea sarà composta da molti server; di conseguenza il client dovrà gestire una struttura multi-server e dovrà anche essere in grado di affrontare le possibili connessioni/disconnessioni da parte dei vari server. Infine, la versione finale del progetto prevede di avere i valori degli elementi in campo (le velocità dei motori, i valori dei sensori, etc) non derivanti da campi statici all'interno del database ma valori reali presenti sul server in real-time, sfruttando il gateway IGS e il database manager interno di iFIX.

Capitolo 5

Simulazione di una linea IMA

Questo è il vero e proprio progetto finale, la fase 4. Lo sviluppo degli step è analogo a quello della fase 3. Ciò che differenzia la fase 4 sono due principali punti:

- la simulazione di una linea IMA come insieme di diversi server ciascuno rappresentante una macchina (e la relativa gestione lato client)
- l'effettiva implementazione dei metodi e delle chiamate da HMI

Modello

Iniziando ad analizzare il primo punto, ogni server a runtime ha necessità di avere il proprio file di configurazione <server>.Config.xml e il file contenente la struttura base dei nodi <server>.PredefinedNodes.uanodes; il client legge il file infoServer_Remote.csv per ottenere le informazioni necessarie per potersi collegare ai vari server della linea. Nella tabella Settings del database è possibile trovare una tupla per ogni server, contenente l'informazione relativa alla versione attualmente caricata sul database. Questa informazione è poi mantenuta sul client in una struttura contenente tutte le informazioni di ciascun server, comprendendo anche la sessione aperta con ciascuno di essi che sarà particolarmente utile per le chiamate dei metodi. Volendo effettuare una vera e propria simulazione a run-time dei server, nella tabella PT_PLCs del database il client non scriverà più il valore dei campi del server (come le

velocità dei motori, il valore dei sensori), ma andrà a scrivere nel relativo campo del db il nome della tag di iFIX legata a quella variabile. Le informazioni sulle tag che verranno poi importate su iFIX (quindi il nome, l'indirizzo IGS, l'unità di misura, i limiti etc) sono definite all'interno del client e scritte su un file chiamato PT_v1_tagImport.csv (il quale cambierà nome aggiornando la versione ogni volta che avrà necessità di essere rigenerato). Importando questo file nel database manager di iFIX, le tag saranno collegate tramite IGS ai veri valori del server. Anche l'HMI subisce una modifica in questi termini: prima per visualizzare i valori delle variabili erano presenti dei campi di testo riempiti staticamente, mentre ora sono stati inseriti dei DataLink che puntano dinamicamente alle tag relative ai valori delle variabili dei sensori e motori presenti nel gruppo macchina attualmente rappresentato nella HMI. Questo passaggio di dati è possibile rappresentarlo con il seguente modello:

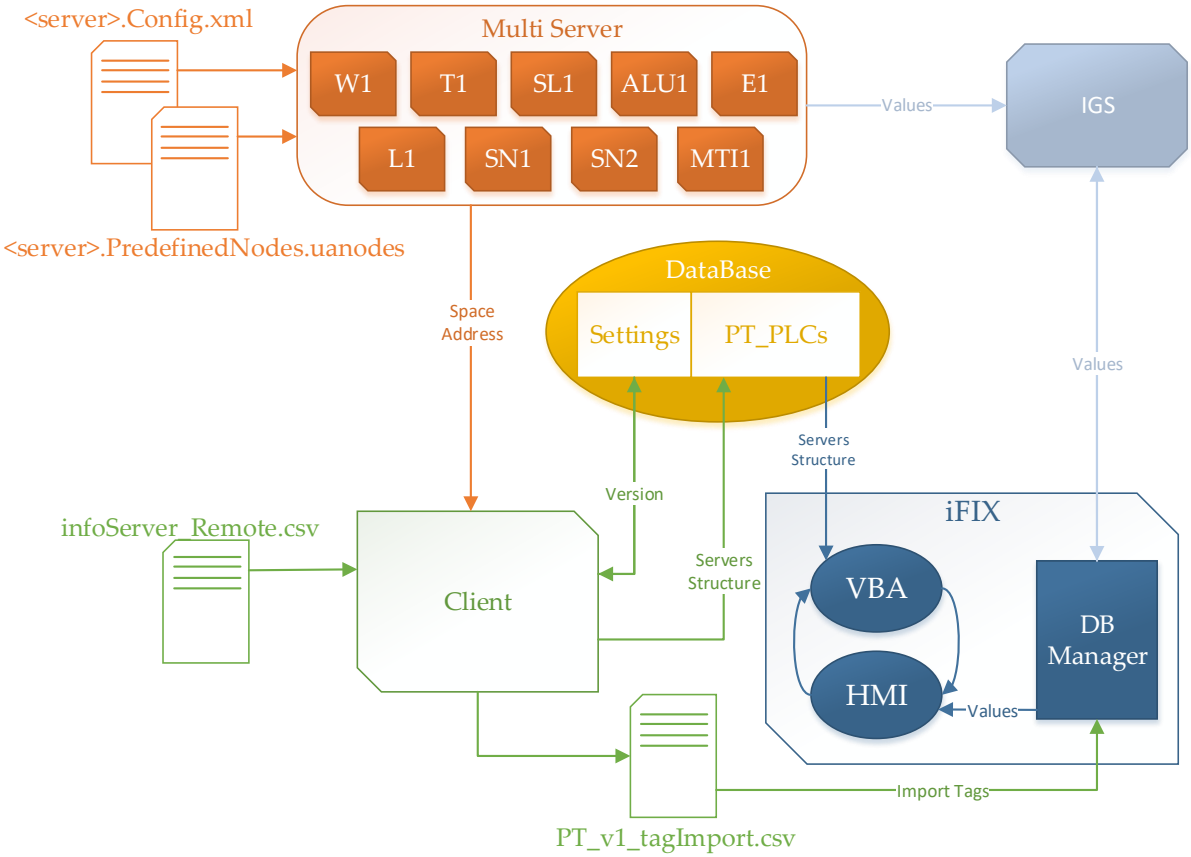

Immagine 16: modello del passaggio dello space address

Per quanto riguarda i metodi, invece, è stato necessario introdurre una qualche forma di comunicazione tra l'HMI e il client. Le possibilità analizzate sono tre: un servizio di web api, l'utilizzo di una tabella sul database come punto di appoggio e, infine, una soket tra il codice VBA dell'HMI e il client C#. È stata scelta, per semplicità e per conformità con gli elementi già utilizzati per lo sviluppo fino a questo momento, la seconda opzione: quando un utente preme un bottone relativo a un metodo, il codice VBA scrive su una tabella le informazioni necessarie al client per chiamare il metodo (macchina, motore e metodo); il client periodicamente andrà a leggere questa tabella ed effettuerà la Call del metodo. Il server eseguirà il metodo e l'HMI riceverà l'esito in due modi che possiamo definire paralleli e complementari:

- da un lato i valori aggiornati del server arriveranno all'HMI sui DataLink mostrati a run-time: questi sono collegati a delle tag che sono associate a degli indirizzi su IGS e, ricordando che IGS è un gateway utilizzato per leggere in tempo reale i dati del server, il giro delle informazioni è presto concluso in direzione server to HMI. L'unico accorgimento particolare da tenere in considerazione è che il device su IGS deve avere tra le impostazioni di Subscription il campo Update Mode non con valore Exception ma con valore Poll (e il relativo campo Registered Read/Write con valore Enable).
- dall'altro lato il server invia l'esito del metodo al client che lo scriverà sul database nella tupla corrispondente a quella chiamata. Il codice dell'HMI va poi a leggere tale informazione riportandone l'esito in una messagebox. Infine, elimina dalla tabella la entry corrispondente alla chiamata di cui ha letto l'esito.

Questo duplice scambio di informazioni dai vari server all'HMI non è sempre necessario ma è risultato molto utile implementarlo nella misura in cui si è appresa la possibilità per l'HMI di avere non solo le informazioni aggiornate a run-time ma anche

la possibilità di ricevere eventuali messaggi di errore o risposte particolari che possono avere una struttura qualsiasi, senza doversi limitare a semplici aggiornamenti di valori.

Tutto questo passaggio di informazioni per la chiamata dei metodi è rappresentabile con il seguente modello:

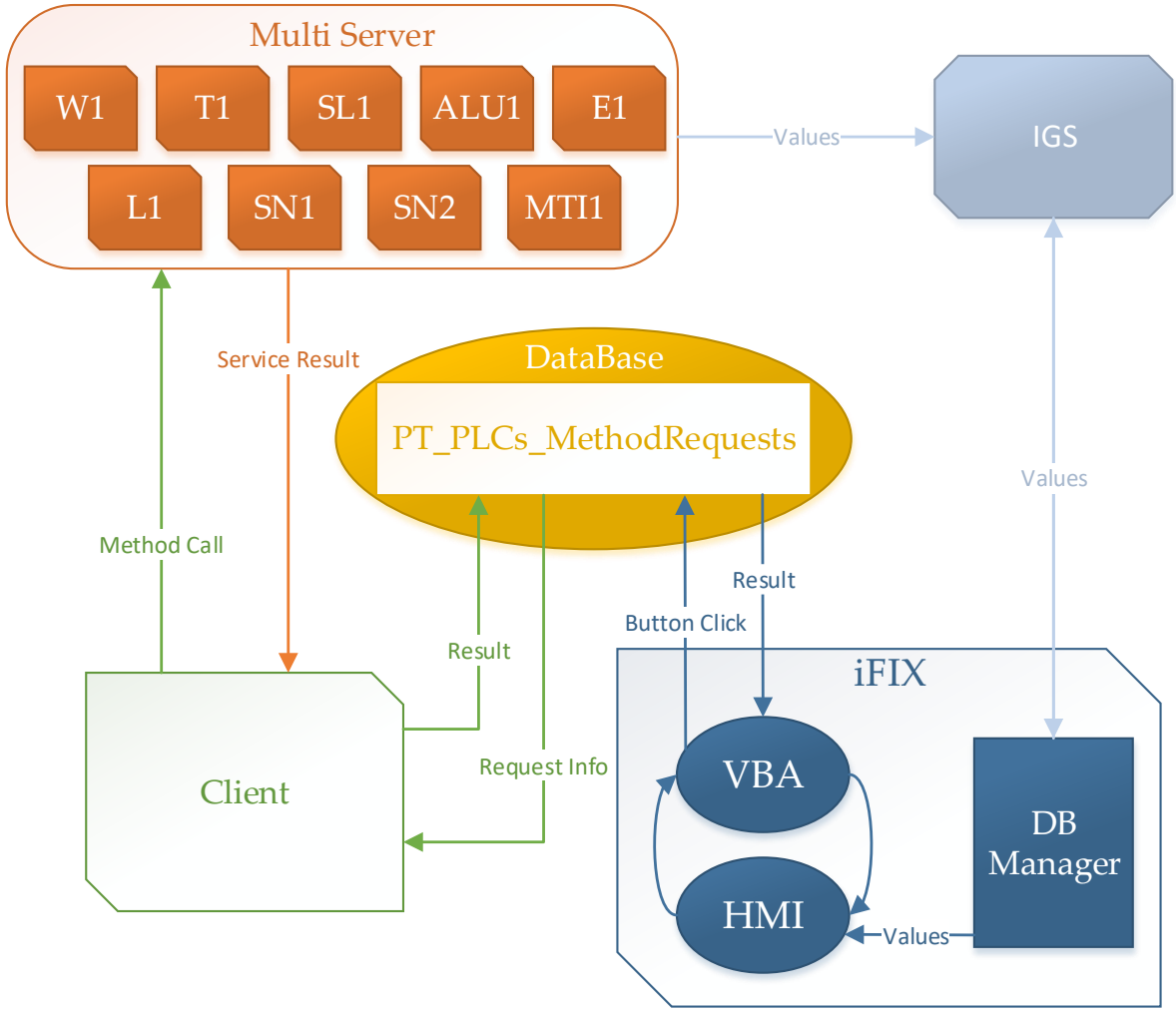

Immagine 17: modello della chiamata dei metodi da HMI

Server

Guardiamo ora nel dettaglio la creazione del primo server che vuole simulare il PLC di una macchina *Washer W1* appartenente alla linea *PiTi*. Vediamo i passaggi concreti da effettuare per riuscire ad avere un server pienamente funzionante.

In primo luogo è necessario creare un progetto di Visual Studio usando il template Console Application, impostare il nome PT_Washer e scegliere .NET 5.0 come framework. Appena creato il progetto è buona norma installare subito le librerie di OPC-UA necessarie per lo sviluppo del server: dal menu Tools, selezionare il NuGet Package Manager e scegliere la voce Package Manager Console. Si aprirà un terminale interattivo nel quale sarà necessario scrivere il comando per installare le librerie:

```
Install-Package OPCFoundation.NETStandard.Opc.Ua
```

Giunti a questo punto, nel progetto creiamo due cartelle nominandole Data e Model. In quest'ultima andiamo a creare il file che definirà il modello del server chiamandolo PT_WasherDesignModel.xml. Per avere supporto durante la stesura del codice di questo file, è utile aprire (senza necessariamente importarlo) all'interno del progetto il file UA Model Designer.xsd presente nel progetto disponibile nella repository GitHub della OPC Foundation chiamato UA-ModelCompiler. Ora è possibile creare il file del modello inserendo nuovi tipi di oggetto, istanze di oggetti, variabili, proprietà e tutto ciò che il protocollo OPC-UA prevede.

L'unico elemento offerto dal protocollo che pare non supportato dalla modellazione tramite model compiler è la possibilità di inserire una lingua desiderata e le possibili traduzioni per quanto riguarda, ad esempio, le descrizioni dei vari nodi.

Importante notare come la variabile rappresentante la versione del server è valorizzata direttamente dentro a questo file e non a livello C# come avviene per tutte

le altre variabili. Questa scelta è stata fatta perché questa informazione non dipende dal runtime (come invece è per le velocità dei motori e i valori dei sensori) ma è un valore che cambia solo al variare della composizione e della struttura del server; è quindi questo il luogo più consono in cui definire e valorizzare questa informazione.

L'effettivo codice in cui è possibile percepire la struttura della Washer è riportato nell'appendice A, sezione server.

Per uniformare la modellazione dei vari server, si è scelto di inserire in tutti i file <server>DesignModel.xml gli stessi elementi base (sensori e motori) anche se non utilizzati da quella specifica macchina. In questo modo si avranno sempre gli stessi identificativi corrispondenti ai vari tipi di sensore o motore su tutta la linea. L'unica informazione presente in tutti i server che però cambia nodo di residenza a ogni server è la versione; per questo motivo l'informazione del nodo su cui risiede la versione verrà fornita al client in maniera statica.

Terminata la modellazione è necessario compilarla con il model compiler al fine di ottenere tutti i file necessari per la realizzazione del server. Per svolgere questo passaggio è necessario avere scaricato e compilato il progetto dell'UA-ModelCompiler; entrando nel percorso UA-ModelCompiler-master\build\bin\Debug\net5.0 aprire una PowerShell in cui digitare il comando

```
.\Opc.Ua.ModelCompiler.exe
      -d2 "<path>\PT_Washer\PT_Washer\Model\PT_WasherDesignModel.xml"
      -cg "<path>\PT_Washer\PT_Washer\Model\PT_WasherDesignModel.csv"
      -o "<path>\PT_Washer\PT_Washer\Data"
```

Questo comando genera nella cartella Model il file PT_WasherDesignModel.csv contenente tutti i nodi del server, il loro identificativo e il tipo del nodo (ObjectType, Object, Variable, Method). Nella cartella Data invece vengono generati tutti gli altri file

necessari (già citati nel capitolo 4 Implementazione di OPC-UA, Fase 2 – Creazione di un modello); in particolare il file PT_Washer.Classes.cs sarà molto utile nella creazione dei file che andranno effettivamente a creare il processo server.

Prima di passare a questi file creiamo, sempre all'interno della cartella Data, il file di configurazione PT_Washer.Config.xml; questo file, come accennato in precedenza, contiene le informazioni relative all'indirizzo a cui si può trovare il server, la porta, i timeout, il file di log e altre informazioni necessarie per l'avvio del server. L'unico elemento di concreto interesse, per quanto tutte le informazioni siano importanti e necessarie, è la porta su cui verrà attivato il servizio: questa informazione va modificata per ogni server poiché al momento tutti i server girano sullo stesso computer e quindi se si affacciassero tutti sulla stessa porta non sarebbe possibile attivare tutti i servizi contemporaneamente. Nello specifico, le porte utilizzate sono: 40059 per la PT_Washer, 40060 per la PT_Tunnel, 40061 per la PT_Filler, 40062 per la PT_Capper, 40063 per la PT_ExternalWasher, 40064 per la PT_TrayLoader, 40065 per la PT_Isolator1, 40066 per la PT_Isolator2 e 40067 per la PT_IsolatorMTI.

Per capirne meglio i dettagli, è possibile visionare il codice di questo file nell'appendice A, sezione server.

Definito anche il file PT_Washer.Config.xml, è necessario impostare per questo file e per PT_Washer.PredefinedNodes.uanodes la proprietà "Copy to Output Directory" con valore "Copy if newer" in modo tale che, quando vengono aggiornati, vengano riportati nella cartella di destinazione del progetto una volta compilato.

I prossimi passaggi riguarderanno la stesura del codice C# per l'avvio e la costruzione del server.

Attualmente nel progetto è presente solo il file Program.cs; il codice da scrivere in questo file serve solo per avviare il server nella sua completezza caricando a runtime il file PT_Washer.Config.xml; per avviare il server viene creata un'istanza di tipo Server, che non è un tipo base di Opc.Ua o di C#.

Dunque è necessario creare un nuovo file Server.cs che implementa questa classe che è una sottoclasse dello StandardServer definito all'interno della libreria Opc.Ua.Server. In questo ridefiniamo un paio di metodi: uno di questi serve per impostare le proprietà del server quali il produttore, il nome esteso del prodotto, l'uri e altre informazioni simili; l'altro, invece, serve per istanziare il NodeManager che è il cuore del server in cui vengono effettivamente valorizzate le variabili e gestiti i metodi. Anche questo elemento, come il server, va definito in un nuovo file.

Il file NodeManager.cs è una classe che implementa la super-classe Opc.Ua.Server.CustomNodeManager2. Al suo interno viene creato lo spazio di indirizzamento del server in primo luogo caricando a runtime il file PT_Washer.PredefinedNodes.uanodes, andando poi a valorizzare tutti i campi degli elementi base e, infine, implementando i metodi. Per quanto riguarda l'inizializzazione degli elementi base è stato scelto di ciclare sui gruppi macchina e per ciascun elemento presente nel gruppo macchina andare a impostare i valori in base al tipo di elemento analizzato. In questo modo è stato possibile impostare per ogni tipo di motore o sensore le proprietà specifiche in maniera coerente con la tipologia di elemento base (ad esempio l'*EngineeringUnits* e l'*EURange* per quanto riguarda le variabili analogiche e il *TrueState* e il *FalseState* per quelle discrete binarie); le inizializzazioni dei valori effettivi delle variabili (come la velocità del motore o la distanza di un sensore) sono state invece create in maniera casuale all'interno di un range sensato per quello specifico tipo di elemento base. Per alcuni motori, ove previsto dal modello, è valorizzata anche una proprietà chiamata Phases che

rappresenta le fasi che un motore può assumere, sotto forma di array (questa proprietà è definita come array nel modello). Come ultimo punto sono stati implementati i metodi specifici di ciascun motore: alla chiamata del metodo è associata una routine di esecuzione che va a modificare i valori o delle variabili interne al motore o dei sensori appartenenti al gruppo macchina in cui risiede quel motore.

Il codice di questi tre file è reperibile nell'appendice A, sezione server.

La modellazione e la creazione degli altri server segue la falsa riga di quanto visto in merito alla *Washer W1*. Ciò che cambia in maniera evidente tra i vari server è la loro struttura, il loro spazio di indirizzamento e nello specifico, quindi, è il design model di ogni singola macchina della linea *PiTi* che riporta le maggiori differenze; in questa trattazione questi file non sono mostrati perché non influenti rispetto agli esiti del progetto e di notevole lunghezza. Ci sono poi differenze "minori" (sono comunque elementi di grossa divergenza tra i vari server e anche di discreto impegno implementativo) che consistono in qualche dettaglio di configurazione, nei namespace, nelle valorizzazioni delle variabili e nell'implementazione dei vari metodi. Queste non sono state riportate esplicitamente in quanto non ritenute rilevanti in questa analisi e facilmente derivabili da tutte le altre informazioni fornite; nell'appendice A, sezione server, non sono quindi riportati tutti i file autogenerati con il model compiler (in quanto facilmente ottenibili seguendo i passi indicati), i file <server>.Config.xml degli altri otto server (poiché facilmente deducibili da quello della Washer già inserito), i file <server>.Program.cs e <server>.Server.cs (quasi identici per tutti i server) e i file <server>.NodeManager.cs (in quanto derivabili dal file PT_Washer.NodeManager.cs in combinazione ai vari file di design model e, inoltre, ritenuti di poco interesse in questa trattazione).

Client

Guardiamo ora nel dettaglio la creazione del client che deve gestire tutti i server che simulano i PLC delle varie macchine. Vediamo i passaggi concreti da effettuare per riuscire a ottenere un client pienamente funzionante.

In primo luogo è necessario creare un progetto di Visual Studio usando il template Console Application, impostare il nome ClientMultiPLC e scegliere .NET 5.0 come framework. Appena creato il progetto è buona norma installare subito le librerie di OPC-UA necessarie per lo sviluppo del server: dal menu Tools, selezionare il NuGet Package Manager e scegliere la voce Package Manager Console. Si aprirà un terminale interattivo nel quale sarà necessario scrivere il comando per installare le librerie:

```
Install-Package OPCFoundation.NETStandard.Opc.Ua
```

La stesura del client è più diretta e lineare rispetto a quella del server (ha sicuramente necessità di meno file per essere creato) ma non per questo più banale.

Partendo direttamente dal file Program.cs, inizialmente sono stati definiti alcuni elementi accessori:

- L'enumerativo GenType per poter definire in modo standard i tipi generici dei nodi secondo un'utilità di nostro interesse a livello grafico. Gli elementi potranno essere di tipo generico Line, Machine, MachineGroup, ASensor, DSensor, Motor, Value, Property o Method.
- La struttura dati TypeStruct utile per poter definire in maniera ancora più scrupolosa il tipo del nodo. Questa struttura contiene i campi nsIndex (indice del namespace), nodeTypeId (identificativo del tipo di nodo), genericType (enumerativo definito poco sopra) e specificType (una stringa

che indica il tipo specifico di ciascun elemento, come ad esempio TemperatureSensorType o PumpMotorType).

- La classe interna ServerInfo utile per andare a memorizzare le varie informazioni relative a ciascun server: il nome, l'indirizzo ip, la porta, l'id del nodo in cui trovare il numero di versione, il numero di versione attualmente disponibile sull'HMI, il numero di versione aggiornato, il canale IGS cui è collegato il server, il device associato al canale appena indicato e la sessione correntemente aperta con il server.

Passando ora all'effettivo codice di funzionamento del client, inizialmente vengono valorizzati i parametri di connessione al database; la scelta attuale è stata quella di cablarli direttamente a codice, ma se il progetto dovesse continuare a essere sviluppato probabilmente queste informazioni sarebbero passate come parametri oppure recuperate da un file esterno.

Passaggio successivo è il caricamento di un file di configurazione, ClientMultiPLC.Config.xml, contenente alcuni parametri utili al funzionamento del client come informazioni sui certificati di sicurezza, timeout, file di log e altre informazioni di questo genere. Questo file è visionabile all'appendice A, sezione client.

Giunti a questo punto è necessario caricare le informazioni relative ai vari server. Leggendo un file chiamato "infoServer_Remote.csv" opportunamente creato in precedenza, si ricavano le informazioni quali: il nome, l'indirizzo ip, la porta, l'id del nodo in cui trovare il numero di versione, il canale IGS legato al server e il device del canale. Leggendo queste informazioni viene popolata una lista contenente delle istanze della classe ServerInfo. Successivamente, per ogni elemento di questa lista, viene provata ad aprire una connessione; in caso di server attivo vengono confrontate le versioni del server attualmente up e quella, se presente, già memorizzata sul

database (se assente viene inserita una nuova tupla nella tabella). In caso di mancata connessione invece viene memorizzato che tale server non è attivo.

Se almeno un server risulta non aggiornato, vengono avviate due procedure: la prima di aggiornamento del database e la seconda di creazione del file delle tag da importare sull'HMI.

La procedura di aggiornamento del database come prima operazione svuota la tabella PT_PLCs presente sul database (tutte le operazioni di lettura e scrittura del database vengono effettuate tramite una classe opportunamente creata contenuta nel file DatabaseIO.cs visionabile nell'appendice A, sezione client). Questa tabella è strutturata esattamente come OPUAMachineConfiguration analizzata nella fase 3 dello sviluppo: contiene i campi *ID*, *parent*, *DisplayName, BrowseName, NodeNamespaceIndex*, *NodeId, TypeNamespaceIndex, TypeDefinitionId, NodeClass, NodeType, Value* e *nChildren*. Nella tabella vuota viene inserito come primo elemento un oggetto rappresentante la linea; questo oggetto è istanziato tramite una classe esterna definita nel file DBElement.cs (visionabile nell'appendice A, sezione client). A questo punto inizia l'esplorazione dello spazio di indirizzamento dei server attivi e l'aggiornamento, se necessario, del numero di versione nella tabella Settings del database. La navigazione nello space address viene effettuata da un'ulteriore funzione esterna che esegue in maniera ricorsiva l'analisi dei nodi: si analizza il nodo, ne si scrivono le informazioni nel database e si procede analizzando i figli di tale nodo. Se il nodo è di tipo Variable ne viene valorizzato il campo Value ma, a differenza del client della fase 3, questo campo può contenere due tipi di informazioni diverse:

- per i nodi che identificano proprietà quali EngineeringUnits, EURange, TrueState e FalseState viene indicata l'effettiva valorizzazione di questi elementi sotto forma di stringa;

- per i nodi che identificano il valore di una variabile (sia scalare come per i sensori o alcune proprietà dei motori come la velocità, sia vettoriale come ad esempio le fasi di alcuni motori), viene creato il nome univoco della tag corrispondente al nodo analizzato che verrà importata su iFIX.

Questo nome generato non sarà utile solo per la scrittura degli elementi sul database ma anche per la creazione di una stringa che servirà per creare le tag da importare nel DataBase Manager di iFIX.

Le tag saranno create solo per gli elementi appartenenti al secondo gruppo di tipi di dato analizzato poco fa. Oltre al nome della tag, viene ricostruito l'indirizzo di IGS a cui sarà possibile accedere per ottenere il valore reale della variabile (possibile memorizzando i *BrowseName* di tutti i nodi dalla radice al nodo corrente). Queste due informazioni, assieme a molti altri parametri che non analizzeremo nel dettaglio, vanno a comporre una stringa che verrà inserita in una lista contenente tutte le tag necessarie (in realtà sono presenti due liste diverse: una per le variabili analogiche e l'altra per quelle digitali poiché le informazioni di creazione della tag sono differenti).

Se l'elemento analizzato appartiene invece al primo gruppo di tipi di nodo, questo non identifica una variabile analogica o discreta ma una proprietà di una di queste. Per questo motivo non viene creata una tag ma viene modificata la stringa che identifica la tag della variabile a cui corrispondono queste proprietà e ne vengono aggiunti o modificati alcuni campi come l'unità di misura, i limiti minimo e massimo entro cui la variabile può variare e i valori rappresentativi corrispondenti ai due possibili stati alle variabili discrete binare.

La procedura di creazione delle tag invece è molto più semplice: viene creato un file csv, nominato in base al nome della linea e alla versione, in cui vengono scritte le due liste contenenti le tag analogiche e discrete, ciascuna delle quali preceduta da

due righe di intestazione (contenenti ad esempio il nome dei campi delle tag) necessarie affinché il DataBase Manager possa interpretare in maniera corretta le informazioni per creare le tag presenti nelle righe successive.

Terminate queste due procedure, il client procede con un'analisi periodica delle connessioni di tutti i server, aggiornandone gli stati in caso di caduta e riconnessione.

Ultimo elemento, non certo per importanza, presente sul client è la chiamata periodica di una routine per percepire le chiamate dei metodi effettuate tramite HMI: collegandosi a un'ulteriore tabella del database chiamata PT_PLCs_MethodRequests, il client verifica se sono presenti nuove tuple corrispondenti a nuove richieste. Questa tabella è strutturata con i seguenti campi:

- *ID*: l'identificativo (la chiave primaria) della tupla
- *Machine*: il nome della macchina sulla quale risiede il motore su cui si vuole invocare il metodo (questa informazione è utile perché per ogni macchina abbiamo un server e quindi una sessione differente)
- *MotorNodeId*: l'identificativo del nodo rappresentante il motore su cui si vuole chiamare il metodo
- *MethodNodeId*: l'identificativo del nodo rappresentante il metodo che si vuole chiamare
- *Result*: una stringa che serve per inviare risposte dal server all'HMI tramite il client (utile in caso di errori e quando il metodo non deve semplicemente modificare un valore)

Il client per identificare una richiesta ancora da servire, effettua una query richiedendo le tuple con *Result* pari a NULL. Ottenuta una richiesta, il client analizza i dati letti dal database per effettuare una Call sulla sessione del server corrispondente alla macchina indicata nella tupla, passando come parametri gli identificativi dei nodi

del motore e del metodo. Ricevuta dal server una risposta di tale invocazione, la scrive nel campo *Result* della tupla precedentemente analizzata in modo da far arrivare tale informazione all'HMI.

HMI

Guardiamo ora nel dettaglio la creazione dell'HMI per visualizzare a monitor la linea *PiTi*, evidenziandone i passaggi concreti da effettuare per ottenere un sistema pienamente funzionante.

In primo luogo è necessario creare tre picture su iFIX rappresentanti, come già evidenziato nella fase 3, tre livelli di dettaglio differenti della linea: una rappresentante le macchine della linea, una rappresentante i gruppi macchina all'interno di una macchina e una rappresentante i motori e i sensori (con le relative variabili e metodi) appartenenti a ogni gruppo macchina.

Nel capitolo precedente (Capitolo 4: Implementazione di OPC-UA, Fase 3 – Impostazione di un prototipo più complesso con HMI) l'HMI è stata analizzata in maniera abbastanza approfondita. Ora verrà fornito qualche ulteriore dettaglio implementativo, senza però scendere nel dettaglio come è stato fatto per server e client (soprattutto dal punto di vista del codice).

La prima picture, denominata BaseLine, è composta da sedici elementi complessi, ciascuno formato da una forma geometrica e un campo di testo. Di questi elementi uno rappresenta l'inizio di un menu di navigazione in stile bread-crumb e identifica la linea di produzione; questo elemento è un gruppo, chiamato "Group_LineName" contenente un poligono a forma pentagonale e un campo di testo chiamato "LineName". Gli altri quindici rappresentano le macchine all'interno della

linea e sono copie di quello che possiamo definire elemento base che non è altro che un gruppo “Group_M1” al cui interno sono contenuti un poligono di forma quadrata “Machine1” e un campo di testo “Name_M1”; i restanti elementi sono copie di questo elemento base con il numero progressivo che varia da 1 a 15.

Graficamente la picture si presenta nel seguente modo:

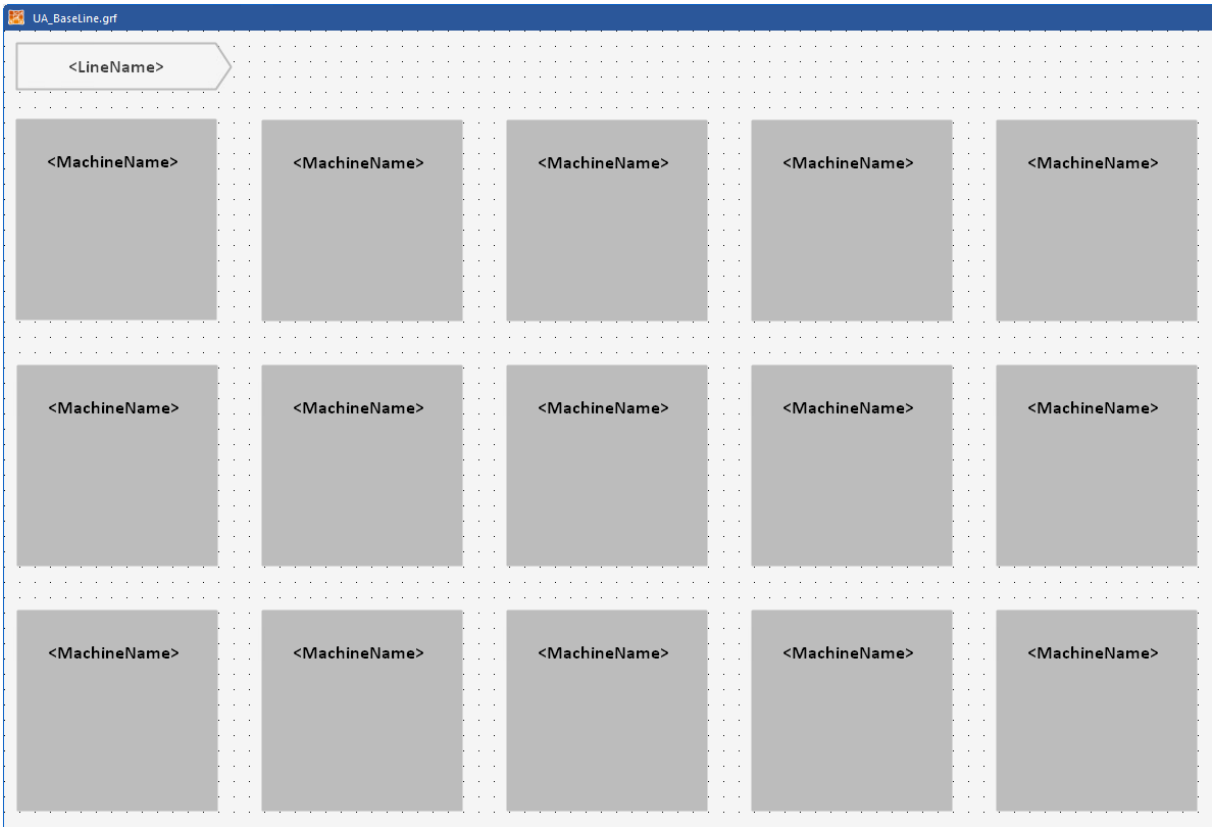

Immagine 18: HMI - BaseLine

Il codice VBA legato a questa pagina inizia con una routine all’inizializzazione della pagina eseguendo due query, utilizzando la libreria ADODB, nei confronti della tabella PT_PLCs:

- La prima query va a ricercare la tupla con campo *parent* uguale a NULL. Questo è l’elemento, per costruzione della tabella lato client, corrispondente

alla linea. Da questa tupla viene preso il nome e inserito nell'apposito elemento grafico nella barra di navigazione in alto.

- La seconda query va a ricercare tutte le tuple con campo *parent* uguale all'ID della tupla restituita dalla query precedente. Per costruzione questa query restituirà tutte le macchine appartenenti alla linea.

Successivamente viene fatto un ciclo iterativo sui record restituiti dall'ultima query descritta e si ricerca a livello grafico un elemento "Group_M*k*" (dove *k* è un numero da 1 a 15) andandone a valorizzare l'elemento "Name_M*k*" con il campo *DispalyName* del record che si sta analizzando. A livello globale viene memorizzato un elenco delle macchine, ciascuna avente una struttura analoga a DBElement vista nel client.

Per quanto riguarda gli elementi grafici che non corrispondono ad alcuna macchina, questi vengono nascosti.

Infine, viene associata una routine per gli eventi "click" di ogni elemento "Group_M*k*" visibile: viene memorizzata globalmente la macchina sulla quale si è cliccato (chiamata d'ora in poi macchina corrente) e successivamente viene aperta una picture di tipo BaseMachine, chiudendo l'attuale picture BaseLine.

La seconda picture è, appunto, BaseMachine ed è composta da diciannove elementi complessi, ciascuno formato da una forma geometrica e un campo di testo. Di questi elementi due rappresentano un menu di navigazione in stile bread-crumb, aumentando il livello di profondità di quello della picture precedente; questi elementi identificano la linea di produzione e la macchina corrente; questi elementi sono due gruppi, chiamati "Group_LineName" e "Group_Machinename", contenenti un

poligono a forma pentagonale o esagonale e un campo di testo chiamato rispettivamente “LineName” e “Machine_Name”; questi due elementi appena citati vengono valorizzati grazie alle variabili globali valorizzate durante il passaggio dalla picture precedente a quella attuale. Nella parte bassa della schermata invece sono presenti altri due gruppi, “Group_PreviousM” e “Group_FollowingM”, composti sempre da un poligono pentagonale e un campo di testo; questi due elementi servono per identificare la macchina precedente e quella successiva. Il procedimento appena descritto avviene all’inizializzazione della schermata (se la macchina corrente è la prima o l’ultima viene nascosto l’elemento di navigazione non utile).

I restanti quindici elementi rappresentano i gruppi macchina all’interno della macchina corrente e sono analoghi agli elementi rappresentanti le macchine nella picture BaseLine (cambia solo leggermente la forma estetica).

Graficamente la picture si presenta nel seguente modo:

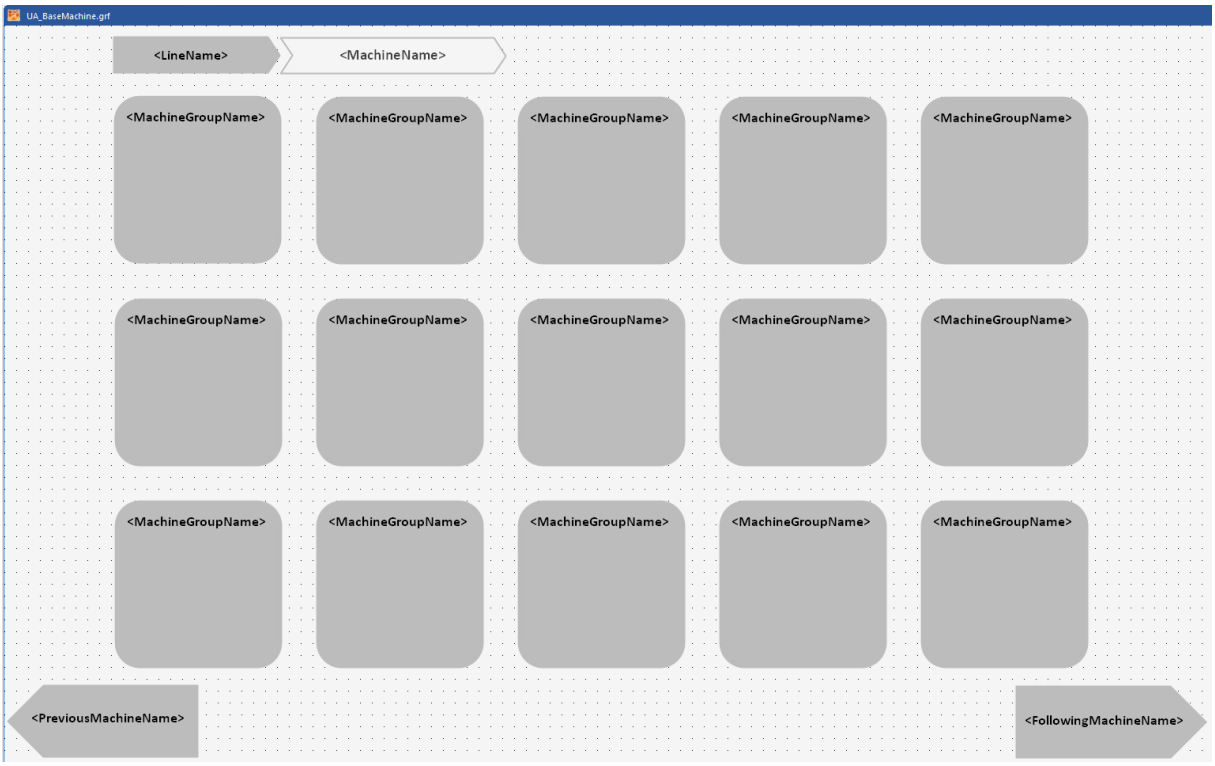

Immagine 19: HMI - BaseMachine

Anche in questa picture, all'atto dell'inizializzazione, viene effettuata una query per ottenere la lista dei gruppi macchina relativi alla macchina corrente. Il tutto avviene seguendo lo stesso principio del caso di BaseLine, cambiando però il valore del campo *parent* degli elementi che si desidera ottenere.

La novità di questa picture è la possibilità di navigare all'interno della linea:

- Cliccando sul "Group_LineName" viene chiusa l'attuale picture BaseMachine e aperta, ripartendo dall'inizializzazione, la picture BaseLine.
- Cliccando invece, quando presenti, sulle frecce di navigazione "Group_PreviousM" e "Group_FollowingM" viene re-inizializzata la picture BaseMachine modificando il riferimento alla macchina corrente. In questo modo si può navigare tra le macchine della linea senza passare forzatamente dalla picture BaseLine.

Infine, viene associata una routine per gli eventi "click" di ogni elemento "Group_MG*k*" visibile: viene memorizzato globalmente il gruppo macchina sul quale si è cliccato (chiamato d'ora in poi gruppo macchina corrente) e successivamente viene aperta una picture di tipo BaseMachineGoup, chiudendo l'attuale picture BaseMachine.

L'ultima picture è BaseMachineGroup ed è composta da venticinque elementi complessi:

- Tre elementi composti da poligono pentagonale o esagonale e un campo di testo che realizzano il menu bread-crumb come nelle picture precedenti ma avendo un livello di profondità maggiore. Questi sono "Group_LineName",

"Group_MachineName" e "Group_MachineGroupName" e funzionano esattamente come presumibile.

- Due elementi "Group_PreviousMG" e "Group_FollowingMG" che, analogamente a quelli della picture BaseMachine, identificano i gruppi macchina precedente e successivo (internamente alla singola macchina).
- Dieci elementi "Group_S*k*" (dove *k* è un progressivo da 1 a 10) che rappresentano dei sensori. Questi sono composti da un rettangolo "Sensor*k*" con gli angoli stondati contenente i campi di testo
    - "Name_S*k*" in cui verrà inserito il nome del sensore,
    - "Range_S*k*" in cui verrà inserito il range del valore del sensore,
    - "EngUnits_S*k*" in cui verrà inserito l'unità di misura del valore del sensore.

  Oltre ai campi di testo è presente anche un DataLink "rValue_S*k*" che verrà collegato dinamicamente a una tag del DataBaseManager.
- Dieci elementi "Group_M*k*" (dove *k* è un progressivo da 1 a 10) che rappresentano dei motori. Questi sono composti da un rettangolo "Mensor*k*" contenente i campi di testo
    - "Name_M*k*" in cui verrà inserito il nome del motore,
    - "Value_M*k*" in cui verrà inserito il nome della variabile del motore,
    - "Range_M*k*" in cui verrà inserito il range del valore della variabile del motore,
    - "EngUnits_M*k*" in cui verrà inserito l'unità di misura del valore della variabile del motore,
    - "Property_M*k*" in cui verrà inserito il nome, se presente, della proprietà del motore (nel nostro framework si riferisce alle fasi).

  Oltre ai campi di testo sono presenti anche i DataLink

- “rValue_M*k*” che verrà collegato dinamicamente a una tag del DataBaseManager,
- “rProperty1_M*k*”, “rProperty2_M*k*”, “rProperty3_M*k*” che verranno collegati dinamicamente a delle tag che rappresentano le possibili fasi del motore.

Infine, sono presenti tre pulsanti “Method1_M*k*”, “Method2_M*k*” e “Method3_M*k*” che saranno associati ai metodi invocabili su ogni motore.

Graficamente la picture si presenta nel seguente modo:

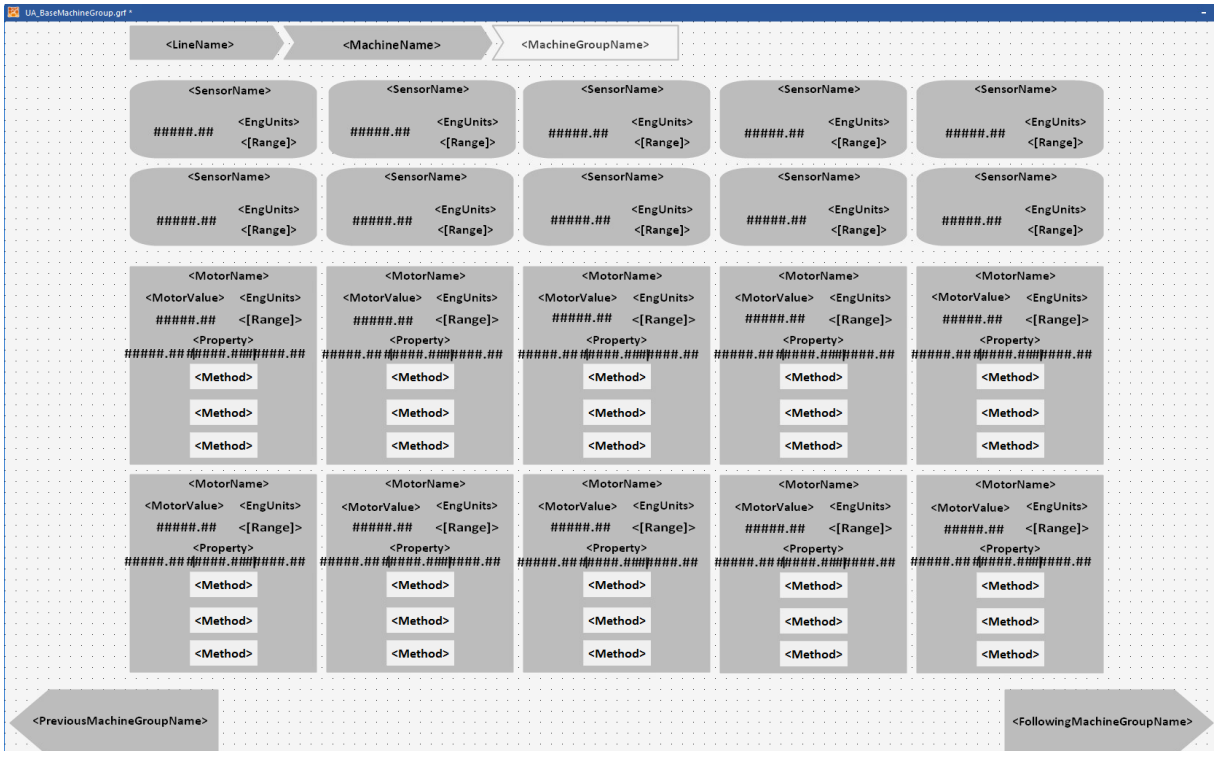

Immagine 20: HMI - BaseMachineGroup

Tutte le valorizzazioni dei campi di testo e i collegamenti ai DataLink avvengono all’inizializzazione della schermata facendo delle query al database e ciclando sui record ottenuti come risposta. Ovviamente tutti gli elementi non disponibili per il gruppo macchina corrente verranno nascosti.

Cliccando sul menu di navigazione in alto e sulle frecce in basso è possibile navigare all'interno della linea in totale libertà.

Interessante può essere la routine eseguita al click su un bottone metodo: il codice VBA dell'HMI apre una connessione con la tabella PT_PLCs_MethodRequests e scrive nella tabella (già analizzata durante la trattazione sul client) il nome della macchina corrente, il NodeId del metodo cliccato e il NodeId del motore sul quale è stato invocato. Attende poi di leggere il campo *Result* di tale tupla inserita per ottenere una qualche informazione ritornata dal server (tramite il client). Una volta letta tale informazione e mostrata in una message box, la tupla viene cancellata dalla tabella.

All'invocazione dei metodi alcuni valori delle variabili mostrate nel gruppo macchina cambieranno (in maniera statica o dinamica); è possibile inoltre impostare i valori direttamente dall'HMI cliccando sui DataLink e inserendo il valore desiderato.

Run-time

Analizzati tutti gli elementi presenti nel progetto finale, vediamo ora l'esecuzione a run-time di tutto il sistema.

I server che simulano i PLC delle macchine sono nove e sono caricati su una macchina virtuale Windows collocata in un server aziendale su cui è installato IGS.

Su tale macchina virtuale sono installati anche iFIX e Sequel Server Manager Studio, utilizzato per la generazione e gestione dell'HMI

Infine, il client è stato lasciato su una macchina fisica all'interno dell'ufficio collegata in rete con il server su cui è installata la macchina virtuale.

Avviando i vari server e IGS (espandiamo solo pochi nodi) si può vedere:

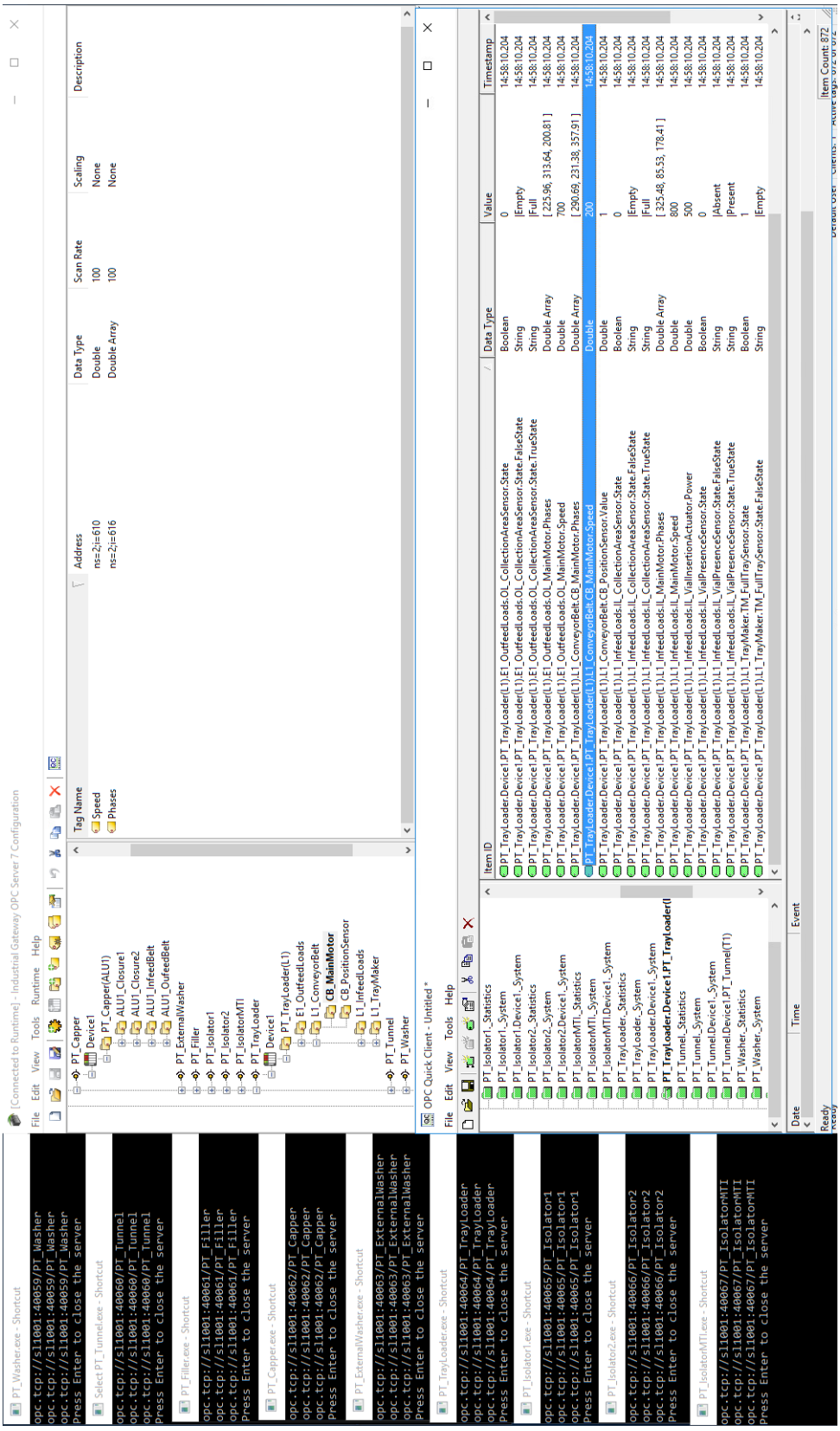

Immagine 21: runtime server

Il client avrà poi questo aspetto:

Immagine 22: runtime client

Sul database la tabella PT_PLCs avrà questo aspetto:

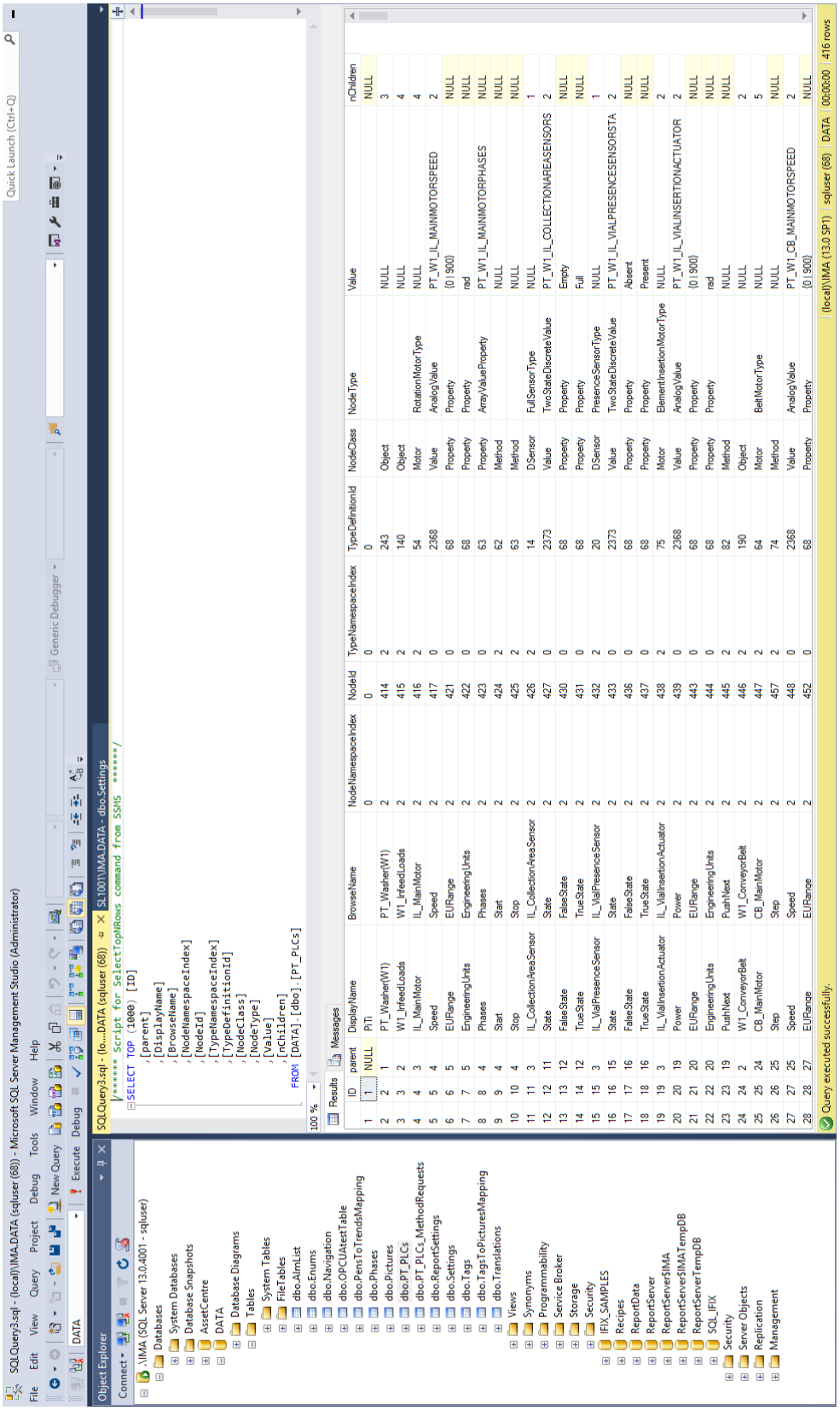

Immagine 23: runtime database

Mostriamo ora come appare l'HMI all'avvio (BaseLine):

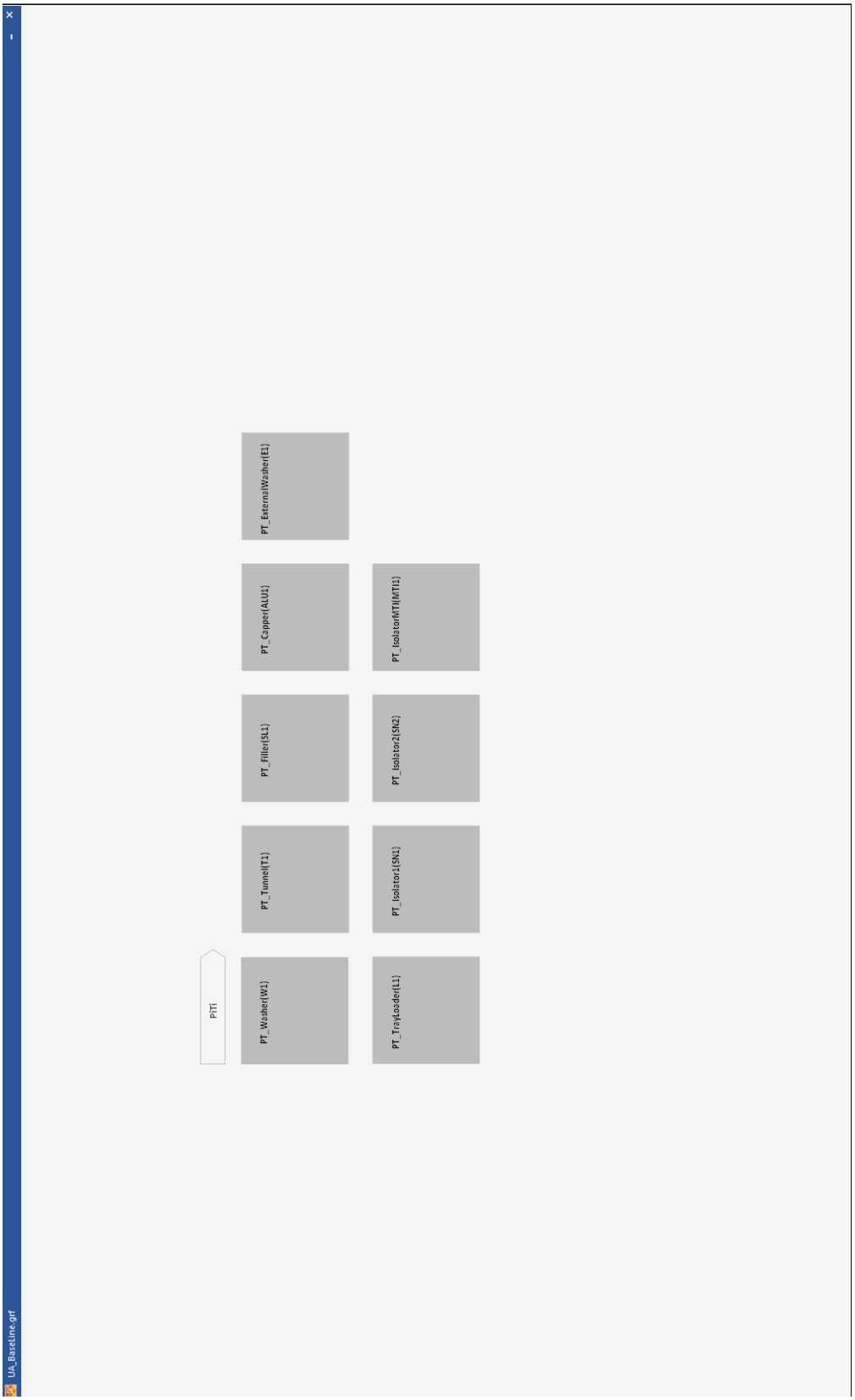

Immagine 24: runtime HMI - BaseLine

Cliccando su una macchina si può ottenere:

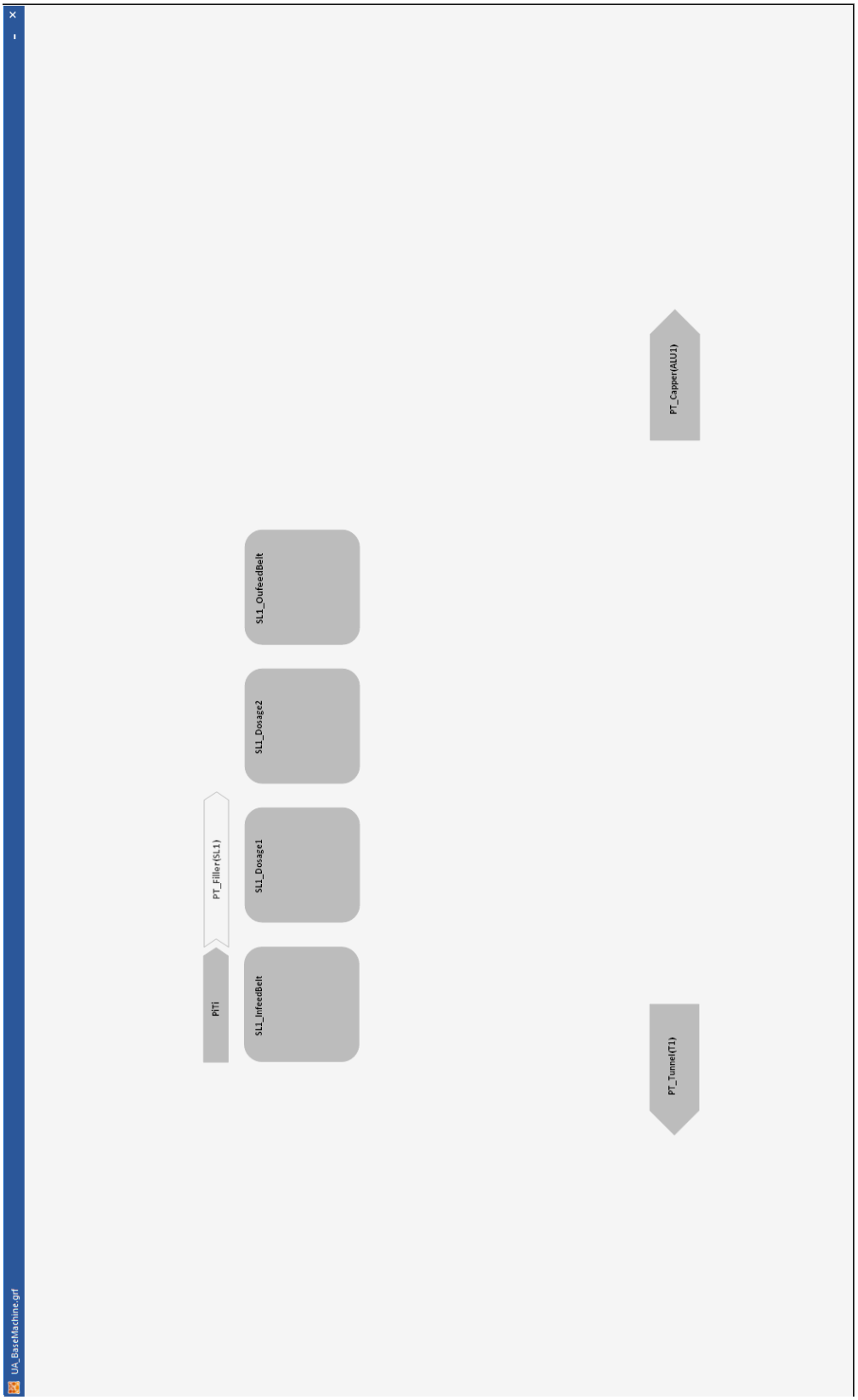

Immagine 25: runtime HMI – BaseMachine

Cliccando su un gruppo macchina si può ottenere:

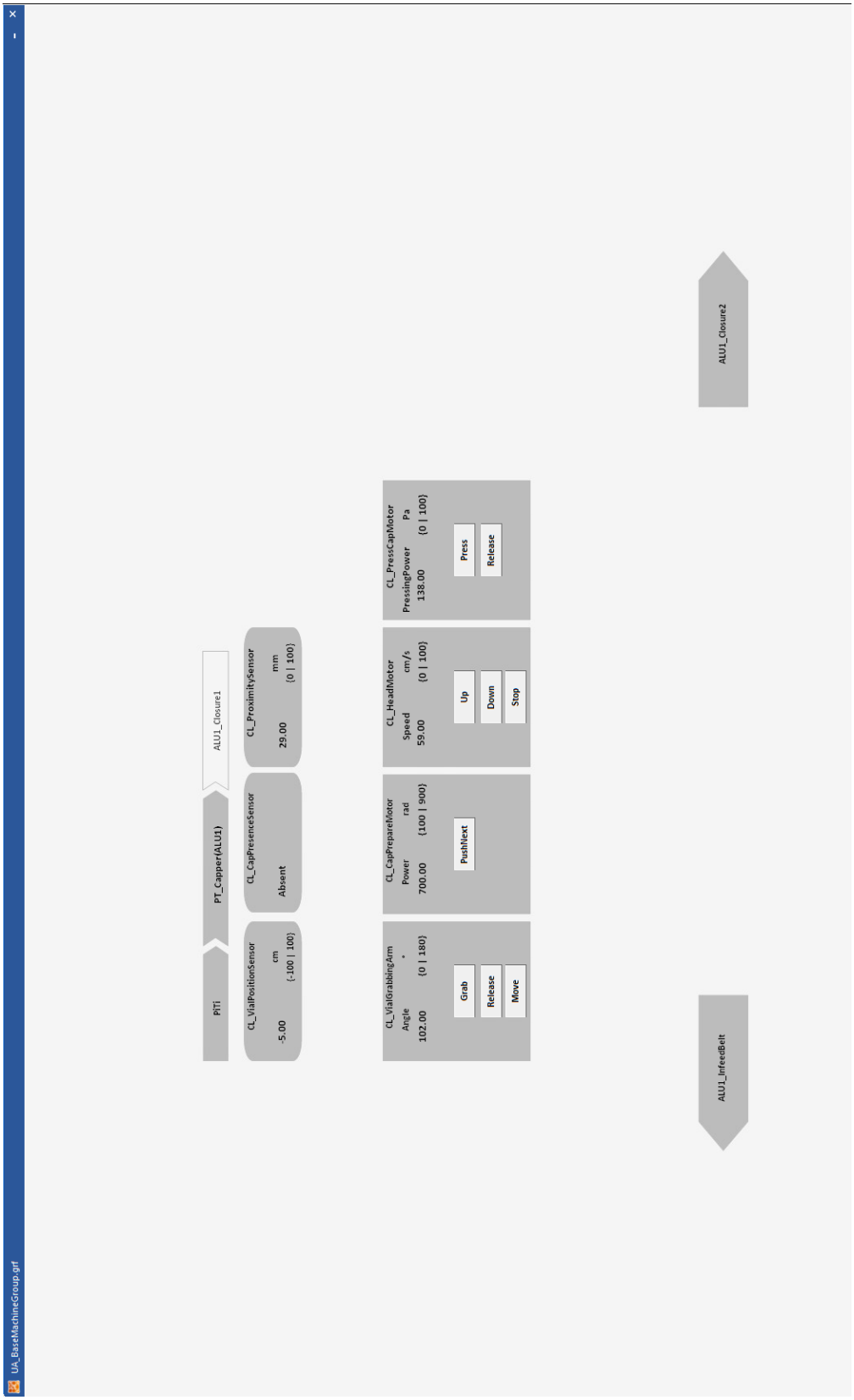

Immagine 26: runtime HMI - BaseMachineGroup

Cliccando su un metodo si può ottenere:

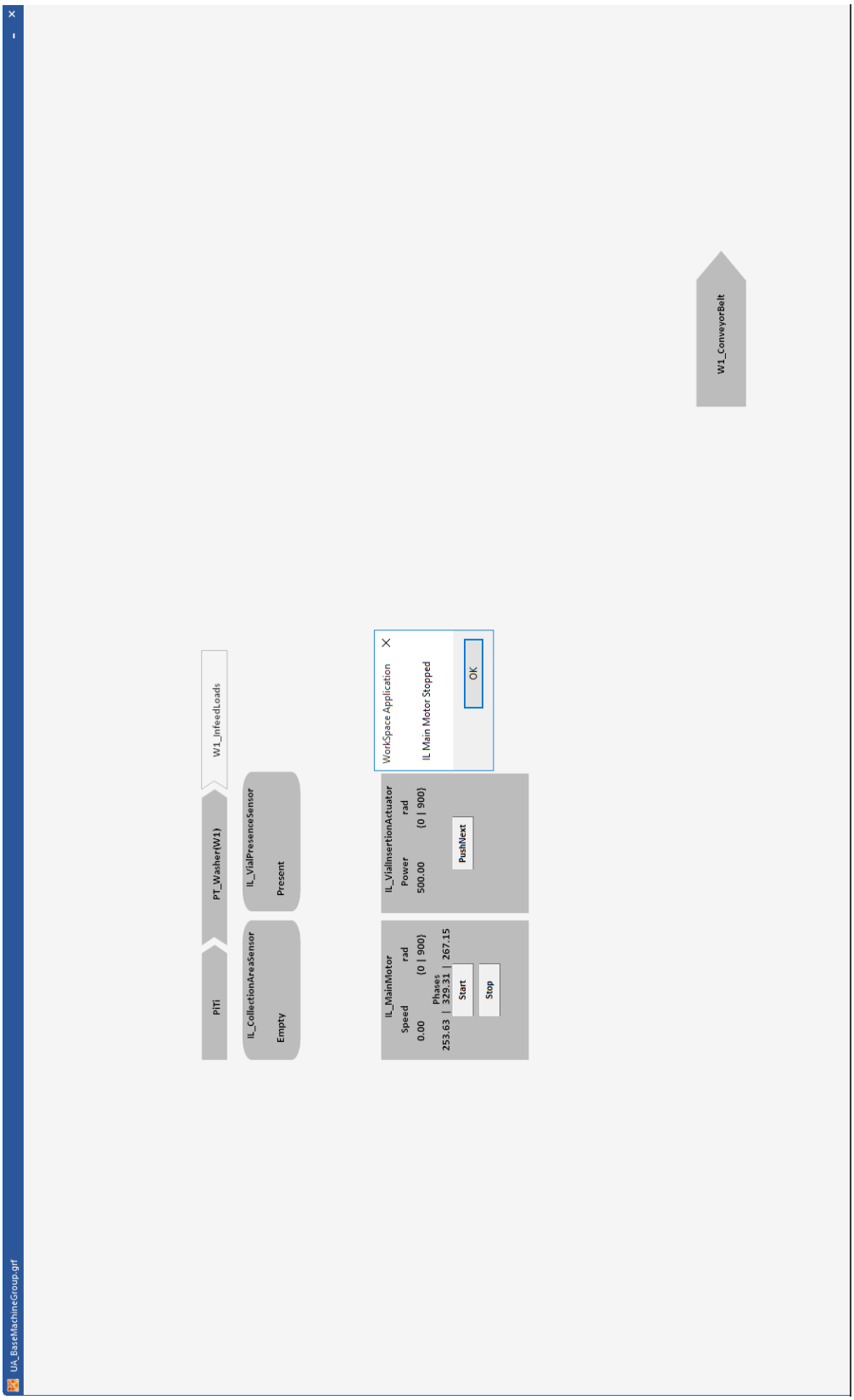

Immagine 27: runtime HMI - method invocation and result

Appendice A

Codice

In questa appendice viene mostrato parte del codice sviluppato durante la redazione del progetto di tesi descritto nei capitoli precedenti. Questo è qui riportato per non riempire la trattazione con snippet di codice che potrebbero distogliere l'attenzione dai concetti cardine oppure potrebbero non interessare il lettore se la lettura non fosse improntata all'implementazione.

Server

Il primo elemento mostrato è il file PT_WasherDesignModel.xml

PT_WasherDesignModel.xml

```xml
<?xml version="1.0" encoding="utf-8" ?>
<opc:ModelDesign
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:opc="http://opcfoundation.org/UA/ModelDesign.xsd"
xmlns:ua="http://opcfoundation.org/UA/"
xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
xmlns="http://opcfoundation.org/PT_Washer"
TargetNamespace="http://opcfoundation.org/PT_Washer">

    <opc:Namespaces>
        <opc:Namespace Name="OpcUa" Prefix="Opc.Ua" XmlNamespace="http://opcfoundation.org/UA/2008/02/Types.xsd">http://opcfoundation.org/UA/</opc:Namespace>
        <opc:Namespace Name="PT_Washer" Prefix="PT_Washer">http://opcfoundation.org/PT_Washer</opc:Namespace>
    </opc:Namespaces>


    <!-- sensors -->

    <opc:ObjectType SymbolicName ="ValueSensorType" BaseType="ua:BaseObjectType" >
        <opc:Description>A generic analog sensor.</opc:Description>
        <opc:Children>
            <opc:Variable SymbolicName="Value" DataType="ua:Double" ValueRank="Scalar" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
                <opc:Children>
                    <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
                    <opc:Property SymbolicName="EURange" DataType="ua:Range" ModellingRule="Mandatory" />
                </opc:Children>
            </opc:Variable>
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="StateSensorType" BaseType="ua:BaseObjectType">
        <opc:Description>A generic binary sensor.</opc:Description>
        <opc:Children>
            <opc:Variable SymbolicName="State" DataType="ua:Boolean" ValueRank="Scalar" TypeDefinition="ua:TwoStateDiscreteType" AccessLevel="ReadWrite" />
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="FullSensorType" BaseType="StateSensorType">
        <opc:Description>A sensor telling whether an object is full</opc:Description>
    </opc:ObjectType>

```

```
    <opc:ObjectType SymbolicName="PresenceSensorType" BaseType="StateSensorType">
        <opc:Description>A sensor telling whether an object is present</opc:Description>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="PositionSensorType" BaseType="ValueSensorType">
        <opc:Description>A sensor telling where an object is</opc:Description>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="ProximitySensorType" BaseType="ValueSensorType">
        <opc:Description>A sensor telling the distance from an object</opc:Description>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="TemperatureSensorType" BaseType="ValueSensorType">
        <opc:Description>A sensor telling the temperature of something</opc:Description>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="WetSensorType" BaseType="ValueSensorType">
        <opc:Description>A sensor telling how much a vial is wet</opc:Description>
    </opc:ObjectType>


    <!-- motors -->

    <opc:ObjectType SymbolicName="RotationMotorType" BaseType="ua:BaseObjectType">
        <opc:Description>A motor with an rotational motion</opc:Description>
        <opc:Children>
            <opc:Variable SymbolicName="Speed" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
                <opc:Children>
                    <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
                </opc:Children>
            </opc:Variable>
            <opc:Variable SymbolicName="Phases" DataType="ua:Double" ValueRank="Array" />
            <opc:Method SymbolicName="Start" ModellingRule="Mandatory" />
            <opc:Method SymbolicName="Stop" ModellingRule="Mandatory" />
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="BeltMotorType" BaseType="RotationMotorType">
        <opc:Description>A rotational motor with Step method</opc:Description>
        <opc:Children>
            <opc:Method SymbolicName="Step" ModellingRule="Mandatory" />
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="ElementInsertionMotorType" BaseType="ua:BaseObjectType">
        <opc:Description>A motor that push an element somewhere</opc:Description>
        <opc:Children>
            <opc:Variable SymbolicName="Power" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
                <opc:Children>
                    <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
                </opc:Children>
            </opc:Variable>
            <opc:Method SymbolicName="PushNext" ModellingRule="Mandatory" />
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="HeadMotorType" BaseType="ua:BaseObjectType">
        <opc:Children>
            <opc:Variable SymbolicName="Speed" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
                <opc:Children>
                    <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
                </opc:Children>
            </opc:Variable>
            <opc:Method SymbolicName="Up" ModellingRule="Mandatory" />
            <opc:Method SymbolicName="Down" ModellingRule="Mandatory" />
            <opc:Method SymbolicName="Stop" ModellingRule="Mandatory" />
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="PumpMotorType" BaseType="ua:BaseObjectType">
        <opc:Children>
            <opc:Variable SymbolicName="FlowRate" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
                <opc:Children>
                    <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
                </opc:Children>
            </opc:Variable>
            <opc:Method SymbolicName="Flow" ModellingRule="Mandatory" />
            <opc:Method SymbolicName="Stop" ModellingRule="Mandatory" />
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="ArmMotorType" BaseType="ua:BaseObjectType">
        <opc:Children>
            <opc:Variable SymbolicName="Angle" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
                <opc:Children>
                    <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
                </opc:Children>
            </opc:Variable>
            <opc:Method SymbolicName="Grab" ModellingRule="Mandatory" />
            <opc:Method SymbolicName="Release" ModellingRule="Mandatory" />
            <opc:Method SymbolicName="Move" ModellingRule="Mandatory" />
        </opc:Children>
    </opc:ObjectType>

    <opc:ObjectType SymbolicName="PressingMotorType" BaseType="ua:BaseObjectType">
        <opc:Children>
            <opc:Variable SymbolicName="PressingPower" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
                <opc:Children>
                    <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
                </opc:Children>
            </opc:Variable>
            <opc:Method SymbolicName="Press" ModellingRule="Mandatory" />
            <opc:Method SymbolicName="Release" ModellingRule="Mandatory" />
        </opc:Children>
    </opc:ObjectType>
```

```
147
148     <opc:ObjectType SymbolicName="InOutMotorType" BaseType="ua:BaseObjectType">
149         <opc:Children>
150             <opc:Variable SymbolicName="Speed" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
151                 <opc:Children>
152                     <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
153                 </opc:Children>
154             </opc:Variable>
155             <opc:Method SymbolicName="Insert" ModellingRule="Mandatory" />
156             <opc:Method SymbolicName="Extract" ModellingRule="Mandatory" />
157         </opc:Children>
158     </opc:ObjectType>
159
160     <opc:ObjectType SymbolicName="ShifterMotorType" BaseType="ua:BaseObjectType">
161         <opc:Children>
162             <opc:Variable SymbolicName="Speed" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite" >
163                 <opc:Children>
164                     <opc:Property SymbolicName="EngineeringUnits" DataType="ua:EUInformation" ModellingRule="Mandatory" />
165                 </opc:Children>
166             </opc:Variable>
167             <opc:Method SymbolicName="Forward" ModellingRule="Mandatory" />
168             <opc:Method SymbolicName="Back" ModellingRule="Mandatory" />
169             <opc:Method SymbolicName="Stop" ModellingRule="Mandatory" />
170         </opc:Children>
171     </opc:ObjectType>
172
173
174     <!-- machine groups -->
175
176     <opc:ObjectType SymbolicName="InfeedLoadsType" BaseType="ua:BaseObjectType">
177         <opc:Children>
178             <opc:Object SymbolicName="IL_MainMotor" TypeDefinition="RotationMotorType" />
179             <opc:Object SymbolicName="IL_CollectionAreaSensor" TypeDefinition="FullSensorType" />
180             <opc:Object SymbolicName="IL_VialPresenceSensor" TypeDefinition="PresenceSensorType" />
181             <opc:Object SymbolicName="IL_VialInsertionActuator" TypeDefinition="ElementInsertionMotorType" />
182         </opc:Children>
183     </opc:ObjectType>
184
185     <opc:ObjectType SymbolicName="ConveyorBeltType" BaseType ="ua:BaseObjectType">
186         <opc:Children>
187             <opc:Object SymbolicName="CB_MainMotor" TypeDefinition="BeltMotorType" />
188             <opc:Object SymbolicName="CB_PositionSensor" TypeDefinition="PositionSensorType" />
189         </opc:Children>
190     </opc:ObjectType>
191
192     <opc:ObjectType SymbolicName="WashingUnitType" BaseType="ua:BaseObjectType">
193         <opc:Children>
194             <opc:Object SymbolicName="WU_HeadMotor" TypeDefinition="HeadMotorType" />
195             <opc:Object SymbolicName="WU_ProximitySensor" TypeDefinition="ProximitySensorType" />
196             <opc:Object SymbolicName="WU_WaterPump" TypeDefinition="PumpMotorType" />
197             <opc:Object SymbolicName="WU_TemperatureSensor" TypeDefinition="TemperatureSensorType" />
198         </opc:Children>
199     </opc:ObjectType>
200
201
202     <!-- machine -->
203
204     <opc:ObjectType SymbolicName="WasherType" BaseType="ua:BaseObjectType" SupportsEvents="true">
205         <opc:Children>
206             <opc:Object SymbolicName="W1_InfeedLoads" TypeDefinition="InfeedLoadsType" />
207             <opc:Object SymbolicName="W1_ConveyorBelt" TypeDefinition="ConveyorBeltType" />
208             <opc:Object SymbolicName="W1_WashingUnit" TypeDefinition="WashingUnitType" />
209         </opc:Children>
210     </opc:ObjectType>
211
212     <opc:Object SymbolicName="PT_WasherMachine" TypeDefinition="WasherType" SupportsEvents="true">
213         <opc:BrowseName>PT_Washer(W1)</opc:BrowseName>
214         <opc:References>
215             <opc:Reference IsInverse="true">
216                 <opc:ReferenceType>ua:Organizes</opc:ReferenceType>
217                 <opc:TargetId>ua:ObjectsFolder</opc:TargetId>
218             </opc:Reference>
219         </opc:References>
220     </opc:Object>
221
222
223     <!-- updated versions -->
224
225     <opc:Property SymbolicName="lastVersion" DataType="ua:Int32">
226         <opc:DefaultValue>
227             <uax:Int32>1</uax:Int32>
228         </opc:DefaultValue>
229     </opc:Property>
230
231 </opc:ModelDesign>
```

Il secondo elemento mostrato è il file PT_Washer.Config.xml

PT_Washer.Config.xml

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <ApplicationConfiguration
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xmlns:ua="http://opcfoundation.org/UA/2008/02/Types.xsd"
5    xmlns="http://opcfoundation.org/UA/SDK/Configuration.xsd"
6  >
7      <ApplicationName>OPCUAServer/PT_Washer</ApplicationName>
8      <ApplicationUri>urn:localhost:PT_Washer</ApplicationUri>
9      <ProductUri>http://opcfoundation.org/PT_Washer</ProductUri>
10     <ApplicationType>Server_0</ApplicationType>
11
```

```
    <SecurityConfiguration>

        <!-- Where the application instance certificate is stored (MachineDefault) -->
        <ApplicationCertificate>
            <StoreType>Directory</StoreType>
            <StorePath>%CommonApplicationData%/OPC Foundation/CertificateStores/MachineDefault</StorePath>
            <SubjectName>PT_Washer</SubjectName>
        </ApplicationCertificate>

        <!-- Where the issuer certificate are stored (certificate authorities) -->
        <TrustedIssuerCertificates>
            <StoreType>Directory</StoreType>
            <StorePath>%CommonApplicationData%/OPC Foundation/CertificateStores/UA Certificate Authorities</StorePath>
        </TrustedIssuerCertificates>

        <!-- Where the trust list is stored (UA Applications) -->
        <TrustedPeerCertificates>
            <StoreType>Directory</StoreType>
            <StorePath>%CommonApplicationData%/OPC Foundation/CertificateStores/UA Applications</StorePath>
        </TrustedPeerCertificates>

        <!-- The directory used to store invalid certficates for later review by the administrator. -->
        <RejectedCertificateStore>
            <StoreType>Directory</StoreType>
            <StorePath>%CommonApplicationData%/OPC Foundation/CertificateStores/RejectedCertificates</StorePath>
        </RejectedCertificateStore>

        <!-- WARNING: The following setting (to automatically accept untrusted certificates) should be used
    for easy debugging purposes ONLY and turned off for production deployments! -->
        <AutoAcceptUntrustedCertificates>true</AutoAcceptUntrustedCertificates>

    </SecurityConfiguration>

    <TransportConfigurations></TransportConfigurations>
    <TransportQuotas>
        <OperationTimeout>600000</OperationTimeout>
        <MaxStringLength>1048576</MaxStringLength>
        <MaxByteStringLength>1048576</MaxByteStringLength>
        <MaxArrayLength>65535</MaxArrayLength>
        <MaxMessageSize>4194304</MaxMessageSize>
        <MaxBufferSize>65535</MaxBufferSize>
        <ChannelLifetime>300000</ChannelLifetime>
        <SecurityTokenLifetime>3600000</SecurityTokenLifetime>
    </TransportQuotas>
    <ServerConfiguration>
        <BaseAddresses>
            <ua:String>opc.tcp://localhost:40059/PT_Washer</ua:String>
        </BaseAddresses>
        <SecurityPolicies>
            <ServerSecurityPolicy>
                <SecurityMode>None_1</SecurityMode>
                <SecurityPolicyUri>http://opcfoundation.org/UA/SecurityPolicy#None</SecurityPolicyUri>
                <SecurityLevel>0</SecurityLevel>
            </ServerSecurityPolicy>
            <ServerSecurityPolicy>
                <SecurityMode>SignAndEncrypt_3</SecurityMode>
                <SecurityPolicyUri>http://opcfoundation.org/UA/SecurityPolicy#Basic256</SecurityPolicyUri>
                <SecurityLevel>5</SecurityLevel>
            </ServerSecurityPolicy>
            <ServerSecurityPolicy>
                <SecurityMode>SignAndEncrypt_3</SecurityMode>
                <SecurityPolicyUri>http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256</SecurityPolicyUri>
                <SecurityLevel>6</SecurityLevel>
            </ServerSecurityPolicy>
        </SecurityPolicies>

        <MinRequestThreadCount>5</MinRequestThreadCount>
        <MaxRequestThreadCount>100</MaxRequestThreadCount>
        <MaxQueuedRequestCount>200</MaxQueuedRequestCount>

        <!-- The SDK expects the server to support the same set of user tokens for every endpoint. -->
        <UserTokenPolicies>
            <!-- Allows anonymous users -->
            <ua:UserTokenPolicy>
                <ua:TokenType>Anonymous_0</ua:TokenType>
                <ua:SecurityPolicyUri>http://opcfoundation.org/UA/SecurityPolicy#None</ua:SecurityPolicyUri>
            </ua:UserTokenPolicy>

            <!-- Allows username/password -->
            <ua:UserTokenPolicy>
                <ua:TokenType>UserName_1</ua:TokenType>
                <!-- passwords must be encrypted - this specifies what algorithm to use -->
                <ua:SecurityPolicyUri>http://opcfoundation.org/UA/SecurityPolicy#Basic256</ua:SecurityPolicyUri>
            </ua:UserTokenPolicy>

            <!-- Allows user certificates -->
            <ua:UserTokenPolicy>
                <ua:TokenType>Certificate_2</ua:TokenType>
                <!-- certificate possession must be proven with a digital signature - this specifies what algorithm to use -->
                <ua:SecurityPolicyUri>http://opcfoundation.org/UA/SecurityPolicy#Basic256</ua:SecurityPolicyUri>
            </ua:UserTokenPolicy>
        </UserTokenPolicies>
        <DiagnosticsEnabled>true</DiagnosticsEnabled>
        <MaxSessionCount>100</MaxSessionCount>
        <MinSessionTimeout>10000</MinSessionTimeout>
        <MaxSessionTimeout>3600000</MaxSessionTimeout>
        <MaxBrowseContinuationPoints>10</MaxBrowseContinuationPoints>
        <MaxQueryContinuationPoints>10</MaxQueryContinuationPoints>
        <MaxHistoryContinuationPoints>100</MaxHistoryContinuationPoints>
        <MaxRequestAge>600000</MaxRequestAge>
        <MinPublishingInterval>100</MinPublishingInterval>
        <MaxPublishingInterval>3600000</MaxPublishingInterval>
        <PublishingResolution>50</PublishingResolution>
        <MaxSubscriptionLifetime>3600000</MaxSubscriptionLifetime>
        <MaxMessageQueueSize>100</MaxMessageQueueSize>
        <MaxNotificationQueueSize>100</MaxNotificationQueueSize>
```

```
        <MaxNotificationsPerPublish>1000</MaxNotificationsPerPublish>
        <MinMetadataSamplingInterval>1000</MinMetadataSamplingInterval>
        <AvailableSamplingRates>
            <SamplingRateGroup>
                <Start>5</Start>
                <Increment>5</Increment>
                <Count>20</Count>
            </SamplingRateGroup>
            <SamplingRateGroup>
                <Start>100</Start>
                <Increment>100</Increment>
                <Count>4</Count>
            </SamplingRateGroup>
            <SamplingRateGroup>
                <Start>500</Start>
                <Increment>250</Increment>
                <Count>2</Count>
            </SamplingRateGroup>
            <SamplingRateGroup>
                <Start>1000</Start>
                <Increment>500</Increment>
                <Count>20</Count>
            </SamplingRateGroup>
        </AvailableSamplingRates>
        <MaxRegistrationInterval>0</MaxRegistrationInterval>
        <NodeManagerSaveFile>PT_Washer.nodes.xml</NodeManagerSaveFile>
    </ServerConfiguration>

    <TraceConfiguration>
        <OutputFilePath>PT_Washer.log.txt</OutputFilePath>
        <DeleteOnLoad>true</DeleteOnLoad>
        <!-- Show Only Errors -->
        <!-- <TraceMasks>1</TraceMasks> -->
        <!-- Show Only Security and Errors -->
        <!-- <TraceMasks>513</TraceMasks> -->
        <!-- Show Only Security, Errors and Trace -->
        <!-- <TraceMasks>515</TraceMasks> -->
        <!-- Show Only Security, COM Calls, Errors and Trace -->
        <!-- <TraceMasks>771</TraceMasks> -->
        <!-- Show Only Security, Service Calls, Errors and Trace -->
        <!-- <TraceMasks>523</TraceMasks> -->
        <!-- Show Only Security, ServiceResultExceptions, Errors and Trace -->
        <!-- <TraceMasks>519</TraceMasks> -->
    </TraceConfiguration>
</ApplicationConfiguration>
```

Il terzo elemento mostrato è il file Program.cs (relativo al server PT_Washer)

Program.cs | PT_Washer | PT_Washer.Program | Main(string[] args)

```
using System;
using Opc.Ua;
using Opc.Ua.Configuration;

namespace PT_Washer
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            ApplicationInstance app = new ApplicationInstance();
            app.ApplicationType = ApplicationType.Server;
            app.ConfigSectionName = "PT_Washer";

            try
            {
                if (app.ProcessCommandLine())
                {
                    return;
                }

                if (!Environment.UserInteractive)
                {
                    app.StartAsService(new Server());
                    return;
                }

                app.LoadApplicationConfiguration(@"Data/PT_Washer.Config.xml", false).Wait();
                app.CheckApplicationInstanceCertificate(false, 0).Wait();

                app.Start(new Server()).Wait();


                foreach (EndpointDescription endpoint in app.Server.GetEndpoints())
                {
                    Console.WriteLine(endpoint.EndpointUrl);
                }

                Console.WriteLine("Press Enter to close the server");
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }
    }
}
```

Il quarto elemento mostrato è il file Server.cs (relativo al server PT_Washer)

Server.cs

PT_Washer | PT_Washer.Server | LoadServerProperties()

```
using Opc.Ua;
using Opc.Ua.Server;
using System.Collections.Generic;

namespace PT_Washer
{
    2 references
    class Server : StandardServer
    {
        0 references
        protected override MasterNodeManager CreateMasterNodeManager(IServerInternal server, ApplicationConfiguration configuration)
        {
            List<INodeManager> nodeManagers = new List<INodeManager>();
            nodeManagers.Add(new NodeManager(server, configuration));
            return new MasterNodeManager(server, configuration, null, nodeManagers.ToArray());
        }

        0 references
        protected override ServerProperties LoadServerProperties()
        {
            ServerProperties prop = new ServerProperties();

            prop.ManufacturerName = "FerraresiMatteo";
            prop.ProductName = "OPCUA server simulation for PT_Washer machine";
            prop.ProductUri = "http://opcfoundation.org/OPCServer/InformationModelServer/v1.0";
            prop.SoftwareVersion = Utils.GetAssemblySoftwareVersion();
            prop.BuildNumber = Utils.GetAssemblyBuildNumber();
            prop.BuildDate = Utils.GetAssemblyTimestamp();

            return prop;
        }
    }
}
```

Il quinto elemento mostrato è il file NodeManager.cs (relativo a PT_Washer)

NodeManager.cs

PT_Washer | PT_Washer.NodeManager | CreateAddressSpace(IDictionary<NodeId, IList<IReference>> exter

```
using Opc.Ua.Server;
using Opc.Ua;
using System;
using System.Collections.Generic;
using System.Reflection;

namespace PT_Washer
{
    3 references
    class NodeManager : CustomNodeManager2
    {
        private static WasherState ptWasher;

        1 reference
        public NodeManager(IServerInternal server, ApplicationConfiguration configuration) : base(server, configuration)
        {
            SystemContext.NodeIdFactory = this;

            string[] namespaceUrls = new string[2];
            namespaceUrls[0] = Namespaces.PT_Washer;
            namespaceUrls[1] = Namespaces.PT_Washer + "/Instance";
            SetNamespaces(namespaceUrls);
        }

        0 references
        protected override NodeStateCollection LoadPredefinedNodes(ISystemContext context)
        {
            NodeStateCollection predefinedNodes = new NodeStateCollection();
            predefinedNodes.LoadFromBinaryResource(context,
                @"Data/PT_Washer.PredefinedNodes.uanodes",
                typeof(NodeManager).GetTypeInfo().Assembly,
                true);

            return predefinedNodes;
        }

        0 references
        public override void CreateAddressSpace(IDictionary<NodeId, IList<IReference>> externalReferences)
        {
            lock (Lock)
            {
                LoadPredefinedNodes(SystemContext, externalReferences);

                // find the untyped Batch Plant 1 node that was created when the model was loaded.
                BaseObjectState passiveNode = (BaseObjectState)FindPredefinedNode(new NodeId(PT_Washer.Objects.PT_WasherMachine,
                    NamespaceIndexes[0]), typeof(BaseObjectState));

                // convert the untyped node to a typed node that can be manipulated within the server.
                ptWasher = new WasherState(null);
                ptWasher.Create(SystemContext, passiveNode);

                // replaces the untyped predefined nodes with their strongly typed versions.
                AddPredefinedNode(SystemContext, ptWasher);
```

```
            // EURange & EngineeringUnits defition
            EUInformation motorEU = new EUInformation("rad", "radiants", "http://www.opcfoundation.org/UA/units/un/cefact");
            Opc.Ua.Range motorEURange = new Opc.Ua.Range() { High = 900, Low = 0 };
            EUInformation headMotorEU = new EUInformation("mm/s", "centimetre per second", "http://www.opcfoundation.org/UA/units/un/cefact");
            Opc.Ua.Range headMotorEURange = new Opc.Ua.Range() { High = 10, Low = 0 };
            EUInformation flowEU = new EUInformation("1/s", "litre per second", "http://www.opcfoundation.org/UA/units/un/cefact");
            Opc.Ua.Range flowEURange = new Opc.Ua.Range() { High = 1.0, Low = 0.00 };
            EUInformation positionEU = new EUInformation("cm", "centimetre", "http://www.opcfoundation.org/UA/units/un/cefact");
            Opc.Ua.Range positionEURange = new Opc.Ua.Range() { High = 100, Low = -100 };
            EUInformation proximityEU = new EUInformation("mm", "millimetre", "http://www.opcfoundation.org/UA/units/un/cefact");
            Opc.Ua.Range proximityEURange = new Opc.Ua.Range() { High = 100, Low = 0 };
            EUInformation temperatureEU = new EUInformation("°C", "degree Celsius", "http://www.opcfoundation.org/UA/units/un/cefact");
            Opc.Ua.Range temperatureEURange = new Opc.Ua.Range() { High = 300, Low = 0 };

            /************** machine values ***************/
            Random rnd = new Random();
            IList<BaseInstanceState> machinegroups = new List<BaseInstanceState>();
            ptWasher.GetChildren(SystemContext, machinegroups);
            foreach (var machinegroup in machinegroups)
            {
                IList<BaseInstanceState> elements = new List<BaseInstanceState>();
                machinegroup.GetChildren(SystemContext, elements);
                foreach (var element in elements)
                {
                    switch (element.GetType().Name)
                    {
                        /* motors */
                        case "RotationMotorState":
                            /* variables */
                            ((RotationMotorState)element).Speed.Value = rnd.Next(1, 9) * 100;
                            ((RotationMotorState)element).Speed.EngineeringUnits.Value = motorEU;
                            ((RotationMotorState)element).Speed.EURange.Value = motorEURange;
                            double[] phases = new double[3];
                            for (int i = 0; i < phases.Length; i++)
                                phases[i] = (double)rnd.Next(0, 36000) / 100;
                            ((RotationMotorState)element).Phases.Value = phases;
                            /* methods - replaced by specific ones */
                            ((RotationMotorState)element).Start.OnCallMethod = new GenericMethodCalledEventHandler(OnStart);
                            ((RotationMotorState)element).Stop.OnCallMethod = new GenericMethodCalledEventHandler(OnStop);
                            break;
                        case "BeltMotorState":
                            /* variables */
                            ((BeltMotorState)element).Speed.Value = rnd.Next(1, 9) * 100;
                            ((BeltMotorState)element).Speed.EngineeringUnits.Value = motorEU;
                            ((BeltMotorState)element).Speed.EURange.Value = motorEURange;
                            phases = new double[3];
                            for (int i = 0; i < phases.Length; i++)
                                phases[i] = (double)rnd.Next(0, 36000) / 100;
                            ((RotationMotorState)element).Phases.Value = phases;
                            /* methods - replaced by specific ones */
                            ((BeltMotorState)element).Start.OnCallMethod = new GenericMethodCalledEventHandler(OnStart);
                            ((BeltMotorState)element).Stop.OnCallMethod = new GenericMethodCalledEventHandler(OnStop);
                            ((BeltMotorState)element).Step.OnCallMethod = new GenericMethodCalledEventHandler(OnStep);
                            break;
                        case "ElementInsertionMotorState":
                            /* variables */
                            ((ElementInsertionMotorState)element).Power.Value = rnd.Next(1, 9) * 100;
                            ((ElementInsertionMotorState)element).Power.EngineeringUnits.Value = motorEU;
                            ((ElementInsertionMotorState)element).Power.EURange.Value = motorEURange;
                            /* methods - replaced by specific ones */
                            ((ElementInsertionMotorState)element).PushNext.OnCallMethod = new GenericMethodCalledEventHandler(OnPush);
                            break;
                        case "HeadMotorState":
                            /* variables */
                            ((HeadMotorState)element).Speed.Value = Convert.ToDouble(rnd.Next(1, 100)) / 10;
                            ((HeadMotorState)element).Speed.EngineeringUnits.Value = headMotorEU;
                            ((HeadMotorState)element).Speed.EURange.Value = headMotorEURange;
                            /* methods - replaced by specific ones */
                            ((HeadMotorState)element).Up.OnCallMethod = new GenericMethodCalledEventHandler(OnUp);
                            ((HeadMotorState)element).Down.OnCallMethod = new GenericMethodCalledEventHandler(OnDown);
                            ((HeadMotorState)element).Stop.OnCallMethod = new GenericMethodCalledEventHandler(OnStop);
                            break;
                        case "PumpMotorState":
                            /* variables */
                            ((PumpMotorState)element).FlowRate.Value = Convert.ToDouble(rnd.Next(1, 100)) / 100;
                            ((PumpMotorState)element).FlowRate.EngineeringUnits.Value = flowEU;
                            ((PumpMotorState)element).FlowRate.EURange.Value = flowEURange;
                            /* methods - replaced by specific ones */
                            ((PumpMotorState)element).Flow.OnCallMethod = new GenericMethodCalledEventHandler(OnFlowStart);
                            ((PumpMotorState)element).Stop.OnCallMethod = new GenericMethodCalledEventHandler(OnFlowStop);
                            break;

                        /* sensors */
                        case "FullSensorState":
                            ((FullSensorState)element).State.Value = rnd.Next(2) == 0;
                            ((FullSensorState)element).State.TrueState.Value = "Full";
                            ((FullSensorState)element).State.FalseState.Value = "Empty";
                            break;
                        case "PresenceSensorState":
                            ((PresenceSensorState)element).State.Value = rnd.Next(2) == 0;
                            ((PresenceSensorState)element).State.TrueState.Value = "Present";
                            ((PresenceSensorState)element).State.FalseState.Value = "Absent";
                            break;
                        case "PositionSensorState":
                            ((PositionSensorState)element).Value.Value = rnd.Next(0, 20000) / 100 - 100;
                            ((PositionSensorState)element).Value.EngineeringUnits.Value = positionEU;
                            ((PositionSensorState)element).Value.EURange.Value = positionEURange;
                            break;
                        case "ProximitySensorState":
                            ((ProximitySensorState)element).Value.Value = rnd.Next(0, 10000) / 100;
                            ((ProximitySensorState)element).Value.EngineeringUnits.Value = proximityEU;
                            ((ProximitySensorState)element).Value.EURange.Value = proximityEURange;
                            break;
                        case "TemperatureSensorState":
                            ((TemperatureSensorState)element).Value.Value = rnd.Next(0, 3000) / 10;
                            ((TemperatureSensorState)element).Value.EngineeringUnits.Value = temperatureEU;
                            ((TemperatureSensorState)element).Value.EURange.Value = temperatureEURange;
                            break;
```

```
                default:
                    break;
            }
        }
    }

    System.Timers.Timer timer_IL_MM = new System.Timers.Timer();
    ptWasher.W1_InfeedLoads.IL_MainMotor.Start.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
    {
        string result = "";
        if (ptWasher.W1_InfeedLoads.IL_MainMotor.Speed.Value == 0)
        {
            result = "IL Main Motor Started";
            timer_IL_MM.Interval = 3 * 1000;
            timer_IL_MM.Elapsed += (source, e) => { ptWasher.W1_InfeedLoads.IL_MainMotor.Speed.Value = new Random().Next(1, 9) * 100; };
            timer_IL_MM.Start();
        }
        else result = "IL Main Motor already working";
        outputArgs.Add(result);
        Console.WriteLine(result);
        return ServiceResult.Good;
    });
    ptWasher.W1_InfeedLoads.IL_MainMotor.Stop.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
    {
        string result = "";
        if (ptWasher.W1_InfeedLoads.IL_MainMotor.Speed.Value > 0)
        {
            result = "IL Main Motor Stopped";
            ptWasher.W1_InfeedLoads.IL_MainMotor.Speed.Value = 0;
            timer_IL_MM.Stop();
        }
        else result = "IL Main Motor already stoppoed";
        outputArgs.Add(result);
        Console.WriteLine(result);
        return ServiceResult.Good;
    });

    System.Timers.Timer timer_IL_VIA = new System.Timers.Timer();
    ptWasher.W1_InfeedLoads.IL_VialInsertionActuator.PushNext.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
    {
        string result = "";
        if (! ptWasher.W1_InfeedLoads.IL_VialPresenceSensor.State.Value)
        {
            result = "IL Vial Insertion Actuator: Vial pushed";
            ptWasher.W1_InfeedLoads.IL_VialPresenceSensor.State.Value = true;
            timer_IL_VIA.Interval = 10 * 1000;
            timer_IL_VIA.Elapsed += (sender, e) => {
                ptWasher.W1_InfeedLoads.IL_VialPresenceSensor.State.Value = false;
                timer_IL_VIA.Stop(); };
            timer_IL_VIA.Start();
        }
        else result = "IL Vial Insertion Actuator: Vial already in place";
        outputArgs.Add(result);
        Console.WriteLine(result);
        return ServiceResult.Good;
    });

    ptWasher.W1_ConveyorBelt.CB_MainMotor.Start.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
    {
        string result = "";
        if (ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value == 0)
        {
            result = "CB Main Motor started";
            ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value = new Random().Next(1, 9) * 100;
        }
        else result = "CB Main Motor already working";
        outputArgs.Add(result);
        return ServiceResult.Good;
    });
    ptWasher.W1_ConveyorBelt.CB_MainMotor.Stop.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
    {
        string result = "";
        if (ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value > 0)
        {
            result = "CB Main Motor stopped";
            ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value = 0;
        }
        else result = "CB Main Motor already stopped";
        outputArgs.Add(result);
        return ServiceResult.Good;
    });
    ptWasher.W1_ConveyorBelt.CB_MainMotor.Step.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
    {
        string result = "";
        double step = (ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.EURange.Value.High - ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.EURange.Value.Low) / 20;
        if (ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value < ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.EURange.Value.High)
        {
            result = "CB_MainMotor stepped";
            if (ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value + step < ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.EURange.Value.High)
                ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value += step;
            else
                ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.Value = ptWasher.W1_ConveyorBelt.CB_MainMotor.Speed.EURange.Value.High;
        }
        else result = "CB_MainMotor already max";
        outputArgs.Add(result);
        return ServiceResult.Good;
    });

    System.Timers.Timer timer_WU_HM = new System.Timers.Timer();
    ptWasher.W1_WashingUnit.WU_HeadMotor.Up.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
    {
        string result = "";
        if (ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value < ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.EURange.Value.High)
        {
            if (timer_WU_HM.Enabled)
                timer_WU_HM.Stop();
            result = "WU Head Motor started going up";
            timer_WU_HM.Interval = 1 * 1000;
```

```
                    timer_WU_HM.Elapsed += (sender, e) => {
                        ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value += ptWasher.W1_WashingUnit.WU_HeadMotor.Speed.Value;
                        if (ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value >= ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.EURange.Value.High) {
                            ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value = ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.EURange.Value.High;
                            timer_WU_HM.Stop();
                        }
                    };
                    timer_WU_HM.Start();
                }
                else result = "WU Head Motor already in top position";
                outputArgs.Add(result);
                Console.WriteLine(result);
                return ServiceResult.Good;
            });
            ptWasher.W1_WashingUnit.WU_HeadMotor.Down.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
            {
                string result = "";
                if (ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value > ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.EURange.Value.Low)
                {
                    if (timer_WU_HM.Enabled)
                        timer_WU_HM.Stop();
                    result = "WU Head Motor started going down";
                    timer_WU_HM.Interval = 1 * 1000;
                    timer_WU_HM.Elapsed += (sender, e) => {
                        ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value -= ptWasher.W1_WashingUnit.WU_HeadMotor.Speed.Value;
                        if (ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value <= ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.EURange.Value.Low)
                        {
                            ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.Value = ptWasher.W1_WashingUnit.WU_ProximitySensor.Value.EURange.Value.Low;
                            timer_WU_HM.Stop();
                        }
                    };
                    timer_WU_HM.Start();
                }
                else result = "WU Head Motor already in bottom position";
                outputArgs.Add(result);
                Console.WriteLine(result);
                return ServiceResult.Good;
            });
            ptWasher.W1_WashingUnit.WU_HeadMotor.Stop.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
            {
                string result = "";
                if (!timer_WU_HM.Enabled)
                {
                    result = "WU Head Motor stopped";
                    timer_WU_HM.Stop();
                }
                else result = "WU Head Motor already stopped";
                outputArgs.Add(result);
                Console.WriteLine(result);
                return ServiceResult.Good;
            });

            ptWasher.W1_WashingUnit.WU_WaterPump.Flow.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
            {
                string result = "";
                if (ptWasher.W1_WashingUnit.WU_WaterPump.FlowRate.Value == 0)
                {
                    result = "WU Water Pump started";
                    ptWasher.W1_WashingUnit.WU_WaterPump.FlowRate.Value = Convert.ToDouble(rnd.Next(1, 100)) / 100;
                }
                else result = "WU Water Pump already working";
                outputArgs.Add(result);
                return ServiceResult.Good;
            });
            ptWasher.W1_WashingUnit.WU_WaterPump.Stop.OnCallMethod = new GenericMethodCalledEventHandler((context, method, inputArgs, outputArgs) =>
            {
                string result = "";
                if (ptWasher.W1_WashingUnit.WU_WaterPump.FlowRate.Value > 0)
                {
                    result = "WU Water Pump stopped";
                    ptWasher.W1_WashingUnit.WU_WaterPump.FlowRate.Value = 0;
                }
                else result = "WU Water Pump stopped";
                outputArgs.Add(result);
                return ServiceResult.Good;
            });
        }
    }


    // method handlers

    2 references
    private ServiceResult OnStart(ISystemContext context, MethodState method, IList<object> inputArgs, IList<object> outputArgs)
    {
        string result = "Motor started";
        Console.WriteLine(result);
        outputArgs.Add(result);
        return ServiceResult.Good;
    }

    3 references
    private ServiceResult OnStop(ISystemContext context, MethodState method, IList<object> inputArgs, IList<object> outputArgs)
    {
        string result = "Motor stopped";
        Console.WriteLine(result);
        outputArgs.Add(result);
        return ServiceResult.Good;
    }

    1 reference
    private ServiceResult OnStep(ISystemContext context, MethodState method, IList<object> inputArgs, IList<object> outputArgs)
    {
        string result = "Motor stepped";
        Console.WriteLine(result);
        outputArgs.Add(result);
        return ServiceResult.Good;
    }

```

```
1 reference
private ServiceResult OnPush(ISystemContext context, MethodState method, IList<object> inputArguments, IList<object> outputArguments)
{
    string result = "Element pushed";
    Console.WriteLine(result);
    outputArguments.Add(result);
    return ServiceResult.Good;
}

1 reference
private ServiceResult OnUp(ISystemContext context, MethodState method, IList<object> inputArguments, IList<object> outputArguments)
{
    string result = "Head up";
    Console.WriteLine(result);
    outputArguments.Add(result);
    return ServiceResult.Good;
}

1 reference
private ServiceResult OnDown(ISystemContext context, MethodState method, IList<object> inputArguments, IList<object> outputArguments)
{
    string result = "Head down";
    Console.WriteLine(result);
    outputArguments.Add(result);
    return ServiceResult.Good;
}

1 reference
private ServiceResult OnFlowStart(ISystemContext context, MethodState method, IList<object> inputArguments, IList<object> outputArguments)
{
    string result = "Flow started";
    Console.WriteLine(result);
    outputArguments.Add(result);
    return ServiceResult.Good;
}

1 reference
private ServiceResult OnFlowStop(ISystemContext context, MethodState method, IList<object> inputArguments, IList<object> outputArguments)
{
    string result = "Flow stopped";
    Console.WriteLine(result);
    outputArguments.Add(result);
    return ServiceResult.Good;
}

}
}
```

Quelli mostrati fino a qui erano i file significativi relativi al server PT_Washer.

Il codice relativo agli altri server non verrà mostrato.

Client

Il primo elemento mostrato è il file ClientMultiPLC.Config.xml

ClientMultiPLC.Config.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<ApplicationConfiguration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ua="http://opcfoundation.org/UA/2008/02/Types.xsd"
  xmlns="http://opcfoundation.org/UA/SDK/Configuration.xsd"
>
  <ApplicationName>Multi PLC Client</ApplicationName>
  <ApplicationUri>urn:localhost:Multi_PLCClient</ApplicationUri>
  <ProductUri>uri:opcfoundation.org:Multi_PLCClient</ProductUri>
  <ApplicationType>Client_1</ApplicationType>

  <SecurityConfiguration>

    <!-- Where the application instance certificate is stored (MachineDefault) -->
    <ApplicationCertificate>
      <StoreType>Directory</StoreType>
      <StorePath>%CommonApplicationData%\OPC Foundation\pki\own</StorePath>
      <SubjectName>CN=Multi PLC Client, C=US, S=Arizona, O=OPC Foundation, DC=localhost</SubjectName>
    </ApplicationCertificate>

   <!-- Where the issuer certificate are stored (certificate authorities) -->
    <TrustedIssuerCertificates>
      <StoreType>Directory</StoreType>
      <StorePath>%CommonApplicationData%\OPC Foundation\pki\issuer</StorePath>
    </TrustedIssuerCertificates>

    <!-- Where the trust list is stored -->
    <TrustedPeerCertificates>
      <StoreType>Directory</StoreType>
      <StorePath>%CommonApplicationData%\OPC Foundation\pki\trusted</StorePath>
    </TrustedPeerCertificates>

    <!-- The directory used to store invalid certficates for later review by the administrator. -->
    <RejectedCertificateStore>
      <StoreType>Directory</StoreType>
      <StorePath>%CommonApplicationData%\OPC Foundation\pki\rejected</StorePath>
    </RejectedCertificateStore>

    <!-- WARNING: The following setting (to automatically accept untrusted certificates) should be used
    for easy debugging purposes ONLY and turned off for production deployments! -->
    <AutoAcceptUntrustedCertificates>false</AutoAcceptUntrustedCertificates>

  </SecurityConfiguration>

  <TransportConfigurations></TransportConfigurations>

  <TransportQuotas>
    <OperationTimeout>600000</OperationTimeout>
    <MaxStringLength>1048576</MaxStringLength>
    <MaxByteStringLength>1048576</MaxByteStringLength>
    <MaxArrayLength>65535</MaxArrayLength>
    <MaxMessageSize>4194304</MaxMessageSize>
    <MaxBufferSize>65535</MaxBufferSize>
    <ChannelLifetime>300000</ChannelLifetime>
    <SecurityTokenLifetime>3600000</SecurityTokenLifetime>
  </TransportQuotas>

  <ClientConfiguration>
    <DefaultSessionTimeout>60000</DefaultSessionTimeout>
    <WellKnownDiscoveryUrls>
      <ua:String>opc.tcp://{0}:4840</ua:String>
      <ua:String>http://{0}:52601/UADiscovery</ua:String>
      <ua:String>http://{0}/UADiscovery/Default.svc</ua:String>
    </WellKnownDiscoveryUrls>
    <DiscoveryServers></DiscoveryServers>
    <MinSubscriptionLifetime>10000</MinSubscriptionLifetime>
  </ClientConfiguration>

  <Extensions>
  </Extensions>

  <TraceConfiguration>
    <OutputFilePath>%CommonApplicationData%\OPC Foundation\Logs\MultiPLCClient.log.txt</OutputFilePath>
    <DeleteOnLoad>true</DeleteOnLoad>
    <!-- Show Only Errors -->
    <!-- <TraceMasks>1</TraceMasks> -->
    <!-- Show Only Security and Errors -->
    <!-- <TraceMasks>513</TraceMasks> -->
    <!-- Show Only Security, Errors and Trace -->
    <!-- <TraceMasks>515</TraceMasks> -->
    <!-- Show Only Security, COM Calls, Errors and Trace -->
    <!-- <TraceMasks>771</TraceMasks> -->
    <!-- Show Only Security, Service Calls, Errors and Trace -->
    <!-- <TraceMasks>523</TraceMasks> -->
    <!-- Show Only Security, ServiceResultExceptions, Errors and Trace -->
    <!-- <TraceMasks>519</TraceMasks> -->
  </TraceConfiguration>

</ApplicationConfiguration>
```

Il prossimo elemento è il file DBElement.cs

DBElement.cs

```
using System;

namespace ClientMultiPLC
{
    6 references
    public class DBElement
    {
        3 references
        public int ID { get; private set; } // aggiunto ID diretto per poter avere un riferimento veloce all'ID del padre
        4 references
        public int Parent { get; private set; }
        3 references
        public string DisplayName { get; private set; }
        3 references
        public string BrowseName { get; private set; }
        3 references
        public int NodeNamespaceIndex { get; private set; }
        3 references
        public uint NodeId { get; private set; }
        3 references
        public int TypeNamespaceIndex { get; private set; }
        3 references
        public uint TypeDefinitionId { get; private set; }
        3 references
        public string NodeClass { get; set; }
        2 references
        public string NodeType { get; set; }
        5 references
        public string Value { get; private set; }
        4 references
        public int NChildren { get; private set; }

        2 references
        public DBElement(int id, int parent, string displayName, string browseName, int nodeNamespaceIndex, object nodeId,
            int typeNamespaceIndex, object typeDefinitionId, string nodeClass, string nodeType, string value, int nChildren)
        {
            ID = id;
            Parent = parent;
            DisplayName = displayName;
            BrowseName = browseName;
            NodeNamespaceIndex = nodeNamespaceIndex;
            NodeId = (UInt32)nodeId;
            TypeNamespaceIndex = typeNamespaceIndex;
            TypeDefinitionId = (UInt32)typeDefinitionId;
            NodeClass = nodeClass;
            NodeType = nodeType;
            Value = value;
            NChildren = nChildren;
        }

        override
        0 references
        public string ToString()
        {
            return String.Format("{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}",
                ID, Parent, DisplayName, BrowseName, NodeNamespaceIndex, NodeId, TypeNamespaceIndex, TypeDefinitionId, NodeClass, Value, NChildren);
        }
    }
}
```

Il prossimo elemento è il file DatabaseIO.cs

DatabaseIO.cs

```
using System.Data.SqlClient;
using System;

namespace ClientMultiPLC
{
    3 references
    class DatabaseIO
    {
        // fields
        private string IP;
        private string ConnectionString;

        //constructor
        1 reference
        public DatabaseIO(string ipAddr, string serverName, string dbName , string user = "sqluser", string password = "sql")
        {
            IP = ipAddr;
            ConnectionString = "Server=" + IP + "\\" + serverName + ";Database=" + dbName + ";User Id=" + user + ";Password= " + password + ";";
        }

        //methods
        2 references
        public bool WriteElementOnDB(string table, DBElement tuple)
        {
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                connection.Open();
                string id = tuple.ID.ToString();
                string parent = tuple.Parent > 0 ? tuple.Parent.ToString() : "NULL";
                string displayName = tuple.DisplayName;
```

```
                string browseName = tuple.BrowseName;
                string nodeNamespaceIndex = tuple.NodeNamespaceIndex.ToString();
                string nodeId = tuple.NodeId.ToString();
                string typeNamespaceIndex = tuple.TypeNamespaceIndex.ToString();
                string typeDefinitionId = tuple.TypeDefinitionId.ToString();
                string nodeClass = tuple.NodeClass;
                string nodeType = tuple.NodeType;
                string value = tuple.Value != null && tuple.Value != "(null)" ? tuple.Value : "NULL";
                string nChildren = tuple.NChildren > 0 ? tuple.NChildren.ToString() : "NULL";
                string insert = String.Format("INSERT INTO " + table +
                    " (ID, parent, DisplayName, BrowseName, NodeNamespaceIndex, NodeId, TypeNamespaceIndex, TypeDefinitionId, NodeClass, NodeType, Value, nChildren)" +
                    " VALUES ({0}, {1}, '{2}', '{3}', {4}, {5}, {6}, {7}, '{8}', '{9}', '{10}', {11})",
                    id, parent, displayName.Replace("'", "''"), browseName.Replace("'", "''"), nodeNamespaceIndex, nodeId, typeNamespaceIndex, typeDefinitionId,
                    nodeClass, nodeType, value.Replace("'", "''"), nChildren);
                using (SqlCommand command = new SqlCommand(insert))
                {
                    command.Connection = connection;
                    command.ExecuteNonQuery();
                }
            }
            return false;
        }

        1 reference
        public void EmptyTable(string tableName)
        {
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                connection.Open();
                string delete =
                    "DELETE FROM " + tableName;
                using (SqlCommand command = new SqlCommand(delete))
                {
                    command.Connection = connection;
                    command.ExecuteNonQuery();
                }
            }
        }

        6 references
        public object ReadValueFromDB(string table, string selectColumn, string whereColumn = null, string whereValue = null)
        {
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                connection.Open();
                string read =
                    "SELECT " + selectColumn +
                    " FROM " + table;
                read += whereColumn == null ? "" : " WHERE [" + whereColumn + "]";
                read += whereColumn != null && whereValue == "NULL" ? "IS NULL" : "= '" + whereValue + "'";
                using (SqlCommand command = new SqlCommand(read))
                {
                    command.Connection = connection;
                    using (SqlDataReader reader = command.ExecuteReader())
                    {
                        reader.Read();
                        return reader.GetValue(0);
                    }
                }
            }
        }

        2 references
        public bool AlterValueOnDB(string table, string setColumn, string setValue, string whereColumn, string whereValue)
        {
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                connection.Open();
                string update =
                    "UPDATE " + table +
                    " SET " + setColumn + " = '" + setValue + "'" +
                    " WHERE [" + whereColumn + "] = '" + whereValue + "'";
                //update = "UPDATE Settings WHERE [Key] = 'loadedVersion' SET Value = '6'";
                using (SqlCommand command = new SqlCommand(update))
                {
                    command.Connection = connection;
                    command.ExecuteNonQuery();
                }
            }
            return true;
        }

        1 reference
        public bool WriteValueOnDB(string table, string values)
        {
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                connection.Open();
                string update =
                    "INSERT INTO " + table +
                    " VALUES (" + values + ")";
                //update = "INSERT INTO Settings VALUES ('loadedVesion_PT_Washer', '1')";
                using (SqlCommand command = new SqlCommand(update))
                {
                    command.Connection = connection;
                    command.ExecuteNonQuery();
                }
            }
            return true;
        }

    }
}
```

Il prossimo elemento è il file Program.cs (relativo al client multi PLC)

Program.cs

ClientMultiPLC | ClientMultiPLC.Program | MethodsCheckRoutine(object sender, ElapsedEventArgs e)

```
using Opc.Ua;
using Opc.Ua.Client;
using Opc.Ua.Configuration;
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using System.Timers;

namespace ClientMultiPLC
{
    23 references
    enum GenType
    {
        Line = 0,
        Machine = 1,
        MachineGroup = 2,
        ASensor = 3,
        DSensor = 4,
        Motor = 5,
        Value = 6,
        Property = 7,
        Method = 8
    }

    25 references
    struct TypeStruct
    {
        public int nsIndex;
        public int nodeTypeId;
        public GenType genericType;
        public string specificType;

        21 references
        public TypeStruct(int nsIndex, int nodeTypeId, GenType genericType, string specificType)
        {
            this.nsIndex = nsIndex;
            this.nodeTypeId = nodeTypeId;
            this.genericType = genericType;
            this.specificType = specificType;
        }
    }

    10 references
    class ServerInfo
    {
        public string name;
        public string address;
        public uint port;
        public uint versionAddress;
        public int loadedVersion;
        public int lastVersion;
        public string channelIGS;
        public string deviceIGS;
        public Session session;

        1 reference
        public ServerInfo(string name, string address, uint port, uint versionAddress, string channelIGS, string deviceIGS)
        {
            this.name = name;
            this.address = address;
            this.port = port;
            this.versionAddress = versionAddress;
            this.loadedVersion = 0;
            this.lastVersion = 0;
            this.channelIGS = channelIGS;
            this.deviceIGS = deviceIGS;
            this.session = null;
        }
    }

    0 references
    class Program
    {
        static int idDB = 0;
        static bool updateDB = false;
        static DatabaseIO db;
        static string ipDB;
        static string serverNameDB;
        static string nameDB;
        static string userDB;
        static string passwordDB;
        static string tableName;
        static List<TypeStruct> lineTypeList;
        static List<string> analogRegisterList;
        static List<string> digitalRegisterList;
        static List<string> tagNameList;
        static List<ServerInfo> serverList;
        static List<string> serverDownList;

        static ApplicationConfiguration m_configuration;

        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("UPC-UA Client - MultiPLC");

            PopulateTypes();

            // database connection info
            ipDB = "10.89.252.15";
            serverNameDB = "IMA";
```

```
        nameDB = "DATA";
        userDB = "sqluser";
        passwordDB = "sql";
        tableName = "PT_PLCs";

        //create client application
        ApplicationInstance application = new ApplicationInstance();
        application.ApplicationType = ApplicationType.Client;

        // load configuration file
        application.LoadApplicationConfiguration(@"..\..\..\ClientMultiPLC.Config.xml", false).Wait();
        m_configuration = application.ApplicationConfiguration;

        // server info
        initLists();
        //string serverInfoFile = "infoServer.csv";
        string serverInfoFile = "infoServer_Remote.csv";
        readServerInfo(serverInfoFile);

        // servers loading
        foreach (ServerInfo server in serverList)
        {
            if (ConnectionTo(server, m_configuration))
            {
                if (server.lastVersion > server.loadedVersion)
                {
                    Console.WriteLine("\t\t" + server.name + " PLC version too old\n");
                    if (!updateDB)
                        updateDB = true;
                }
            }
            else // server down
            {
                serverDownList.Add(server.name);
            }
        }

        if (updateDB)
        {
            UpdateDatabase();
            writeCSV("PT");
        }

        #region online/offline server check
        Thread tServersCheck = new Thread(new ThreadStart(ServersCheckThread));
        tServersCheck.Start();
        #endregion

        #region method call from HMI
        Thread tMethodsCheck = new Thread(new ThreadStart(MethodsCheckThread));
        tMethodsCheck.Start();
        #endregion

        Console.ReadLine();
    }

    1 reference
    private static void MethodsCheckThread()
    {
        System.Timers.Timer timerMethods = new System.Timers.Timer();
        timerMethods.Interval = 0.5 * 1000;
        timerMethods.Elapsed += MethodsCheckRoutine;
        timerMethods.Start();
    }

    1 reference
    private static void MethodsCheckRoutine(object sender, ElapsedEventArgs e)
    {
        System.Collections.Generic.IList<object> outputArgumentsCustom;
        string machine, motor, method;
        if ((int)db.ReadValueFromDB("PT_PLCs_MethodRequests", "COUNT(*)", "Result", "NULL") == 0) { return; }
        machine = db.ReadValueFromDB("PT_PLCs_MethodRequests", "Machine", "Result", "NULL").ToString();
        motor = db.ReadValueFromDB("PT_PLCs_MethodRequests", "MotorNodeId", "Result", "NULL").ToString();
        method = db.ReadValueFromDB("PT_PLCs_MethodRequests", "MethodNodeId", "Result", "NULL").ToString();
        string id = db.ReadValueFromDB("PT_PLCs_MethodRequests", "ID", "Result", "NULL").ToString();

        string serverName = machine.Split('(')[0];

        ServerInfo server = serverList.Find(x => x.name == serverName);
        try
        {
            outputArgumentsCustom = server.session.Call(new NodeId(motor), new NodeId(method));
            foreach (object o in outputArgumentsCustom)
            {
                Console.WriteLine(o);
                db.AlterValueOnDB("PT_PLCs_MethodRequests", "Result", o.ToString(), "ID", id);
            }
            Console.WriteLine("Done");
        }
        catch (Exception ex)

        {
            if (ex is ServiceResultException)
                Console.WriteLine("Error in the call:\n\t" + ex.Message); //viene presa questa
            else if (ex is InvalidOperationException)
                Console.WriteLine("Error in the call:\n\t" + ex.Message);
            else
                Console.WriteLine("Error in the call");
        }
    }

    1 reference
    private static void ServersCheckThread()
    {
        System.Timers.Timer timerServers = new System.Timers.Timer();
        timerServers.Interval = 5 * 1000;
        timerServers.Elapsed += ServersCheckRoutine;
        timerServers.Start();
    }
```

```
1 reference
private static void ServersCheckRoutine(object sender, ElapsedEventArgs e)
{
    Console.WriteLine("\n\n\n\t\tPeriodical servers check\n\n");
    foreach (ServerInfo server in serverList)
    {
        Console.WriteLine(server.name + " analysis");
        if (serverDownList.Contains(server.name))
        {
            Console.WriteLine("-> offline, try reconnection...");
            if (ConnectionTo(server, m_configuration))
            {
                Console.WriteLine("\t\t\tNow is ONLINE!");
                serverDownList.Remove(server.name);
                if (server.lastVersion > server.loadedVersion)
                {
                    Console.WriteLine("A new PLC is on line! The DB is changed. Please import the tags and restart the HMI.");
                    UpdateDatabase();
                }
            }
            else
                Console.WriteLine("-> still offline.");
        }
        else // not in serversDownList
        {
            if (!server.session.KeepAliveStopped)
            {
                Console.WriteLine("Already up");
            }
            else
            {
                Console.WriteLine("\t\t\t!DISCONNECTED!");
                server.session = null;
                serverDownList.Add(server.name);
            }
        }
        Console.WriteLine();
        Thread.Sleep(1000);
    }
}

2 references
private static void UpdateDatabase()
{
    Console.WriteLine("\n\n\t\t\t" + "Update database\n\n");
    Console.ReadLine();
    db.EmptyTable(tableName);

    idDB++;
    DBElement element = new DBElement(idDB, 0, "PiTi", "", 0, (uint)0, 0, (uint)0, "", "", "", 0);
    //Console.WriteLine(element.ToString());
    db.WriteElementOnDB(tableName, element);

    foreach (ServerInfo server in serverList)
    {
        if (server.session.Connected)
        {
            if (server.lastVersion > server.loadedVersion)
                db.AlterValueOnDB("Settings", "Value", server.lastVersion.ToString(), "Key", "loadedVersion_" + server.name);

            // browsing throgh the server
            Console.WriteLine("Browse the server namespace.");
            ReferenceDescriptionCollection refs;
            Byte[] cp;
            server.session.Browse(null, null, ObjectIds.ObjectsFolder, 0u, BrowseDirection.Forward, ReferenceTypeIds.HierarchicalReferences, true,
                (uint)NodeClass.Variable | (uint)NodeClass.Object | (uint)NodeClass.Method, out cp, out refs);

            Console.WriteLine("DisplayName: BrowseName, NodeClass"); // only for space address printing

            foreach (var rd in refs)
            {
                if (rd.DisplayName != "Server" && rd.DisplayName != "Aliases") // don't show Server and Aliases branches
                {
                    string prefix = ""; // only for space address printing
                    string tagName = "";
                    int parent = 1;
                    string startingAddress = server.channelIGS + "." + server.deviceIGS;
                    recoursiveBrowse(server.session, rd, prefix, parent, tagName, startingAddress);
                }
            }
        }
    }
}

2 references
private static bool ConnectionTo(ServerInfo server, ApplicationConfiguration m_configuration)
{
    Console.WriteLine("Connecting " + server.name);
    string myserver = "opc.tcp://" + server.address + ":" + server.port + "/" + server.name;

    // version identification
    string versionName = "loadedVersion_" + server.name;
    int loadedVersion = 0;
    db = new DatabaseIO(ipDB, serverNameDB, nameDB, userDB, passwordDB);
    try
    {
        loadedVersion = Int32.Parse((string)db.ReadValueFromDB("Settings", "Value", "Key", versionName));
    }
    catch (ArgumentNullException)
    {
        string values = "'" + "loadedVersion_" + server.name + "', '" + loadedVersion + "', ''";
        db.WriteValueOnDB("Settings", values);
    }
    server.loadedVersion = loadedVersion;

    // get the endpoint by connecting to server's discovery endpoint
    // try to find the first endpoint without security
    EndpointDescription endpointDescription = null;
```

```
            try
            {
                endpointDescription = CoreClientUtils.SelectEndpoint(myserver, false);
            }
            catch (Exception e)
            {
                Console.WriteLine("\t" + server.name + " unreachable");
                return false;
            }

            EndpointConfiguration endpointConfiguration = EndpointConfiguration.Create(m_configuration);
            ConfiguredEndpoint endpoint = new ConfiguredEndpoint(null, endpointDescription, endpointConfiguration);

            // create the session
            Session session = Session.Create(
                m_configuration,
                endpoint,
                false,
                false,
                m_configuration.ApplicationName,
                (uint)m_configuration.ClientConfiguration.DefaultSessionTimeout,
                new UserIdentity(),
                null).Result;

            server.session = session;

            // find last version
            server.lastVersion =
                (int)server.session.ReadValue(ExpandedNodeId.ToNodeId(new NodeId("ns=2;i=" + server.versionAddress), server.session.NamespaceUris)).Value;
            return true;
        }

        0 references
        private static void Session_SessionClosing(object sender, EventArgs e)
        {
            Console.WriteLine(sender + " is closing");
            serverDownList.Add(serverList.Find(x => x.session == sender).name);
        }

        1 reference
        private static void PopulateTypes()
        {
            lineTypeList = new List<TypeStruct>();

            /* base type */
            lineTypeList.Add(new(0, 63, GenType.Property, "ArrayValueProperty"));
            lineTypeList.Add(new(0, 68, GenType.Property, "Property"));
            lineTypeList.Add(new(0, 2368, GenType.Value, "AnalogValue"));
            lineTypeList.Add(new(0, 2373, GenType.Value, "TwoStateDiscreteValue"));

            /* sensor types */
            lineTypeList.Add(new(2, 1, GenType.ASensor, "ValueSensorType"));
            lineTypeList.Add(new(2, 8, GenType.DSensor, "StateSensorType"));
            lineTypeList.Add(new(2, 14, GenType.DSensor, "FullSensorType"));
            lineTypeList.Add(new(2, 20, GenType.DSensor, "PresenceSensorType"));
            lineTypeList.Add(new(2, 26, GenType.ASensor, "PositionSensorType"));
            lineTypeList.Add(new(2, 33, GenType.ASensor, "ProximitySensorType"));
            lineTypeList.Add(new(2, 40, GenType.ASensor, "TemperatureSensorType"));
            lineTypeList.Add(new(2, 47, GenType.ASensor, "WetSensorType"));

            /* motor types */
            lineTypeList.Add(new(2, 54, GenType.Motor, "RotationMotorType"));
            lineTypeList.Add(new(2, 64, GenType.Motor, "BeltMotorType"));
            lineTypeList.Add(new(2, 75, GenType.Motor, "ElementInsertionMotorType"));
            lineTypeList.Add(new(2, 83, GenType.Motor, "HeadMotorType"));
            lineTypeList.Add(new(2, 93, GenType.Motor, "PumpMotorType"));
            lineTypeList.Add(new(2, 102, GenType.Motor, "ArmMotorType"));
            lineTypeList.Add(new(2, 112, GenType.Motor, "PressingMotorType"));
            lineTypeList.Add(new(2, 121, GenType.Motor, "InOutMotorType"));
            lineTypeList.Add(new(2, 130, GenType.Motor, "ShifterMotorType"));
        }

        1 reference
        private static void readServerInfo(string serverInfoFile)
        {
            try
            {
                using (StreamReader sr = new StreamReader(@"..\..\..\" + serverInfoFile))
                {
                    string line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        var values = line.Split(',');
                        ServerInfo server = new ServerInfo(values[0].Trim(), values[1].Trim(), Convert.ToUInt32(values[2]), Convert.ToUInt32(values[3]),
                            values[4].Trim(), values[5].Trim());
                        serverList.Add(server);
                    }
                }
            } catch (Exception) { }
        }

        2 references
        private static void recoursiveBrowse(Session session, ReferenceDescription rd, string prefix, int parent, string tagName, string address)
        {
            ReferenceDescriptionCollection nextRefs;
            Byte[] nextCp;
            session.Browse(null, null, ExpandedNodeId.ToNodeId(rd.NodeId, session.NamespaceUris), 0u, BrowseDirection.Forward,
                ReferenceTypeIds.HierarchicalReferences, true, (uint)NodeClass.Variable | (uint)NodeClass.Object | (uint)NodeClass.Method, out nextCp, out nextRefs);

            Console.Write("{0}{1}: {2}, {3} [{4}]", prefix, rd.DisplayName, rd.BrowseName, rd.NodeClass, rd.NodeId); // only for space address printing
            string value = null;
            if (rd.NodeClass.ToString() == "Variable")
            {
                try
                {
                    DataValue a = session.ReadValue(ExpandedNodeId.ToNodeId(rd.NodeId, session.NamespaceUris));
                    Console.Write(" (value: {0})", a); // only for space address printing
                    value = a.ToString();
                    if (rd.BrowseName.Name == "EngineeringUnits")
                    {
                        string[] tokens = a.ToString().Split("|");
```

```
            value = tokens[2].Trim();
        }

    }
    catch { Exception e; }
}
Console.WriteLine(); // only for space address printing

// abbreviations
if (rd.DisplayName.ToString().Contains("_"))
{
    int lastUnder = tagName.LastIndexOf("_");
    tagName = tagName.Substring(0, lastUnder + 1);
    tagName += rd.DisplayName.ToString();
}
address += "." + rd.BrowseName.Name;

// parameters to write
string displayName = rd.DisplayName.Text;
string browseName = rd.BrowseName.Name;
int nodeNsIndex = rd.NodeId.NamespaceIndex;
uint nodeId = (UInt32)rd.NodeId.Identifier;
TypeStruct type = lineTypeList.Find(t => t.nsIndex == rd.TypeDefinition.NamespaceIndex && t.nodeTypeId == (UInt32)rd.TypeDefinition.Identifier);
int typeNsIndex = type.specificType != null ? type.nsIndex : rd.TypeDefinition.NamespaceIndex;
uint typeId = type.specificType != null ? (UInt32)type.nodeTypeId : (UInt32)rd.TypeDefinition.Identifier;
string genType = type.specificType != null ? type.genericType.ToString() : rd.NodeClass.ToString();
string specType = type.specificType != null ? type.specificType : null;
string valueDB = value;
if (genType == "Value" || specType == "ArrayValueProperty")
{
    tagName = (tagName + displayName).ToUpper();
    valueDB = tagName.Length <= 30 ? tagName : tagName.Substring(0, 30);
}
int nChild = nextRefs.Count;

// check on possible duplicates name of tag
if (valueDB != null && valueDB != value)
{
    int i = 1;
    while (tagNameList.Contains(valueDB))
    {
        if (valueDB.Length == 30)
        {
            int cut = (int)Math.Truncate(Math.Log10(i)) + 1;
            valueDB = valueDB.Substring(0, 30 - cut);
        }
        valueDB = String.Concat(valueDB, ++i);
    }
    tagNameList.Add(valueDB);
}

bool createCSV = true;
if (createCSV)
{
    string ar = "AR";
    string dr = "DR";
    string device = "IGS";
    System.Threading.Thread.CurrentThread.CurrentCulture = System.Globalization.CultureInfo.GetCultureInfo("en-US");
    //string lowLimit = String.Format("{0:0.00}", 0);
    //string highLimit = string.Format("{0:0.00}", 100);
    if (specType == "AnalogValue" || specType == "ArrayValueProperty")
        analogRegisterList.Add(
            '"' + ar + '"' + ',' +
            '"' + valueDB + '"' + ',' +
            '"' + '"' + ',' +
            '"' + device + '"' + ',' +
            '"' + '"' + ',' +
            '"' + "DECIMAL" + '"' + ',' +
            '"' + address + '"' + ',' +
            '"' + "None" + '"' + ',' +
            '"' + '"' + ',' + //'"' + lowLimit + '"' + ',' +
            '"' + '"' + ',' + //'"' + highLimit + '"' + ',' +
            '"' + '"' + ',' +
            '"' + "YES" + '"' + ',' + // -> enable output
            '"' + "DISABLE" + '"' + ',' +
            '"' + "NONE" + '"' + ',' +
            '"' + "NONE" + '"' + ',' +
            '"' + "NONE" + '"' + ',' +
            '"' + "NONE" + '"' + ',' +
            '"' + "ALL" + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + '"' + ',' +
            '"' + "NONE" + '"' + ',' +
            '"' + "YES" + '"' + ',' +
            '"' + "NO" + '"' + ',' +
            '"' + "REJECT" + '"' + ',' +
            '"' + "NO" + '"' + ',' +
            '"' + "1,000" + '"' + ',' +
            '"' + "300,000" + '"' + ',' +
            '"' + "0" + '"' + ',' +
            '"' + "NO" + '"' + ',' +
            '"' + "NO" + '"' + ',' +
            '"' + "0" + '"' + ',' +
            '"' + '"' + ',' +
            '"' + "NO" + '"' + ',' +
            '"' + "5,000.0" + '"' + ',' +
            '"' + "0.0" + '"' + ',' +
```

```
                    '"' + "Milliseconds" + '"' + ',' +
                    '"' + "DISABLE" + '"' + ',' +
                    '"' + "0.0000" + '"' + ',' +
                    '"' + "Absolute" + '"' + ',' +
                    '"' + "0" + '"' + ',' +
                    '"' + "NO" + '"' + ',' +
                    '"' + "NO" + '"' + ',' +
                    '"' + "YES" + '"' + ',' +
                    '"' + "0.00" + '"' + ',' +
                    '"' + "65,535.00" + '"' + ',' +
                    '"' + "0.00" + '"' + ',' +
                    '"' + "100.00" + '"'
                    );
                else if (specType == "TwoStateDiscreteValue")
                    digitalRegisterList.Add(
                    '"' + dr + '"' + ',' +
                    '"' + valueDB + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + device + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + "DECIMAL" + '"' + ',' +
                    '"' + address + '"' + ',' +
                    '"' + "YES" + '"' + ',' + // -> enable output
                    '"' + "YES" + '"' + ',' + // -> invert output
                    '"' + "TRUE" + '"' + ',' +
                    '"' + "FALSE" + '"' + ',' +
                    '"' + "DISABLE" + '"' + ',' +
                    '"' + "NONE" + '"' + ',' +
                    '"' + "NONE" + '"' + ',' +
                    '"' + "NONE" + '"' + ',' +
                    '"' + "NONE" + '"' + ',' +
                    '"' + "ALL" + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + "NONE" + '"' + ',' +
                    '"' + "YES" + '"' + ',' +
                    '"' + "NO" + '"' + ',' +
                    '"' + "REJECT" + '"' + ',' +
                    '"' + "NO" + '"' + ',' +
                    '"' + "1,000" + '"' + ',' +
                    '"' + "300,000" + '"' + ',' +
                    '"' + "0" + '"' + ',' +
                    '"' + "NO" + '"' + ',' +
                    '"' + "NO" + '"' + ',' +
                    '"' + "0" + '"' + ',' +
                    '"' + '"' + ',' +
                    '"' + "NO" + '"' + ',' +
                    '"' + "5,000.0" + '"' + ',' +
                    '"' + "0.0" + '"' + ',' +
                    '"' + "Milliseconds" + '"' + ',' +
                    '"' + "DISABLE" + '"' + ',' +
                    '"' + "0.0000" + '"' + ',' +
                    '"' + "Absolute" + '"' + ',' +
                    '"' + "0" + '"'
                    );
                else if (rd.BrowseName.Name == "EURange")
                { // add the range to the last AR insert
                    string[] range = value.Substring(1, value.Length - 2).Split("|");
                    for (int i = 0; i < 2; i++)
                        range[i] = String.Format("{0:0.00}", Convert.ToDouble(range[i]));
                    string last = analogRegisterList[analogRegisterList.Count - 1];
                    int inBetweenComma = 9;

                    int posL = -1;
                    int posU = -1;
                    int occur = 0;
                    for (int i = 0; i < last.Length; i++)
                    {
                        if (last[i] == ',')
                            occur += 1;
                        else if (occur == inBetweenComma)
                            posL = i - 2;
                        posU = i + 1;
                        if (posL > 0 && posU > 0)
                            break;
                    }
                    last = last.Insert(posU, range[1]);
                    last = last.Insert(posL, range[0]);
                    analogRegisterList[analogRegisterList.Count - 1] = last;
                }
                else if (rd.BrowseName.Name == "EngineeringUnits")
                { // add the EU to the last AR insert
                    string last = analogRegisterList[analogRegisterList.Count - 1];
                    int euComma = 10;

                    int pos = -1;
                    int occur = 0;
                    for (int i = 0; i < last.Length; i++)
                    {
                        if (last[i] == ',')
                            occur += 1;
                        if (occur == euComma)
                        {
                            pos = i + 2;
                            break;
                        }
                    }
```

```
                last = last.Insert(pos, value);
                analogRegisterList[analogRegisterList.Count - 1] = last;
            }
            else if (rd.BrowseName.Name == "TrueState")
            {
                string last = digitalRegisterList[digitalRegisterList.Count - 1];
                digitalRegisterList[digitalRegisterList.Count - 1] = last.Replace("TRUE", value);
            }
            else if (rd.BrowseName.Name == "FalseState")
            {
                string last = digitalRegisterList[digitalRegisterList.Count - 1];
                digitalRegisterList[digitalRegisterList.Count - 1] = last.Replace("FALSE", value);
            }
        }

        if (updateDB)
        {
            idDB++;
            DBElement element = new DBElement(idDB, parent, displayName, browseName, nodeNsIndex, nodeId, typeNsIndex, typeId, genType, specType, valueDB, nChild);
            db.WriteElementOnDB(tableName, element);
        }

        if (nextRefs.Count > 0)
            parent = idDB;
        foreach (var nextRd in nextRefs)
        {
            recoursiveBrowse(session, nextRd, prefix + "\t", parent, tagName, address);
        }
    }

    1 reference
    private static void initLists()
    {
        serverList = new List<ServerInfo>();
        serverDownList = new List<string>();
        tagNameList = new List<string>();

        analogRegisterList = new List<string>();
        analogRegisterList.Add(
            "[BLOCK TYPE,TAG,DESCRIPTION,I/O DEVICE,H/W OPTIONS,I/O ADDRESS TYPE,I/O ADDRESS,SIGNAL CONDITIONING,LOW EGU LIMIT,HIGH EGU LIMIT,EGU TAG," +
            "OUTPUT ENABLE,EVENT MESSAGES,ALARM AREA(S),SECURITY AREA 1,SECURITY AREA 2,SECURITY AREA 3,ALARM AREA 1,ALARM AREA 2,ALARM AREA 3,ALARM AREA 4," +
            "ALARM AREA 5,ALARM AREA 6,ALARM AREA 7,ALARM AREA 8,ALARM AREA 9,ALARM AREA 10,ALARM AREA 11,ALARM AREA 12,ALARM AREA 13,ALARM AREA 14," +
            "ALARM AREA 15,USER FIELD 1,USER FIELD 2,ESIG TYPE,ESIG ALLOW CONT USE,ESIG XMPT ALARM ACK,ESIG UNSIGNED WRITES,ESIG COMMENT REQUIRED," +
            "PDR Update Rate,PDR Access Time,PDR Deadband,PDR Latch,PDR Disable Output,PDR Array Length,Hist Description,Hist Collect,Hist Interval," +
            "Hist Offset,Hist Time Res,Hist Compress,Hist Deadband,Hist Comp Type,Hist Comp Time,Scale Enabled,Scale Clamping,Scale Use EGU,Scale Raw Low," +
            "Scale Raw High,Scale Low,Scale High]\n" +
            "!A_NAME, A_TAG, A_DESC, A_IODV, A_IOHT, A_NUMS, A_IOAD, A_IOSC, A_ELO, A_EHI, A_EGUDESC, A_OUT, A_EVENT, A_ADI, A_SA1, A_SA2, A_SA3, A_AREA1," +
            " A_AREA2, A_AREA3, A_AREA4, A_AREA5, A_AREA6, A_AREA7, A_AREA8, A_AREA9, A_AREA10, A_AREA11, A_AREA12, A_AREA13, A_AREA14, A_AREA15, A_ALMEXT1," +
            " A_ALMEXT2, A_ESIGTYPE, A_ESIGCONT, A_ESIGACK, A_ESIGTRAP, A_ESIGREQ_COMMENT, A_PDR_UPDATERATE, A_PDR_ACCESSTIME, A_PDR_DEADBAND, A_PDR_LATCHDATA," +
            " A_PDR_DISABLEOUT, A_PDR_ARRAYLENGTH, A_HIST_DESC, A_HIST_COLLECT, A_HIST_INTERVAL, A_HIST_OFFSET, A_HIST_TIMERES, A_HIST_COMPRESS, A_HIST_DEADBAND," +
            " A_HIST_COMPTYPE, A_HIST_COMPTIME, A_SCALE_ENABLED, A_SCALE_CLAMP, A_SCALE_USEEGU, A_SCALE_RAWLOW, A_SCALE_RAWHIGH, A_SCALE_LOW, A_SCALE_HIGH!");

        digitalRegisterList = new List<string>();
        digitalRegisterList.Add(
            "[BLOCK TYPE,TAG,DESCRIPTION,I/O DEVICE,H/W OPTIONS,I/O ADDRESS TYPE,I/O ADDRESS,ENABLE OUTPUT,INVERT OUTPUT,OPEN TAG,CLOSE TAG,EVENT MESSAGES," +
            "ALARM AREA(S),SECURITY AREA 1,SECURITY AREA 2,SECURITY AREA 3,ALARM AREA 1,ALARM AREA 2,ALARM AREA 3,ALARM AREA 4,ALARM AREA 5,ALARM AREA 6," +
            "ALARM AREA 7,ALARM AREA 8,ALARM AREA 9,ALARM AREA 10,ALARM AREA 11,ALARM AREA 12,ALARM AREA 13,ALARM AREA 14,ALARM AREA 15,USER FIELD 1," +
            "USER FIELD 2,ESIG TYPE,ESIG ALLOW CONT USE,ESIG XMPT ALARM ACK,ESIG UNSIGNED WRITES,ESIG COMMENT REQUIRED,PDR Update Rate,PDR Access Time," +
            "PDR Deadband,PDR Latch,PDR Disable Output,PDR Array Length,Hist Description,Hist Collect,Hist Interval,Hist Offset,Hist Time Res," +
            "Hist Compress,Hist Deadband,Hist Comp Type,Hist Comp Time]\n" +
            "!A_NAME, A_TAG, A_DESC, A_IODV, A_IOHT, A_NUMS, A_IOAD, A_OUT, A_INV, A_OPENDESC, A_CLOSEDESC, A_EVENT, A_ADI, A_SA1, A_SA2, A_SA3, A_AREA1," +
            " A_AREA2, A_AREA3, A_AREA4, A_AREA5, A_AREA6, A_AREA7, A_AREA8, A_AREA9, A_AREA10, A_AREA11, A_AREA12, A_AREA13, A_AREA14, A_AREA15, A_ALMEXT1," +
            " A_ALMEXT2, A_ESIGTYPE, A_ESIGCONT, A_ESIGACK, A_ESIGTRAP, A_ESIGREQ_COMMENT, A_PDR_UPDATERATE, A_PDR_ACCESSTIME, A_PDR_DEADBAND, A_PDR_LATCHDATA," +
            " A_PDR_DISABLEOUT, A_PDR_ARRAYLENGTH, A_HIST_DESC, A_HIST_COLLECT, A_HIST_INTERVAL, A_HIST_OFFSET, A_HIST_TIMERES, A_HIST_COMPRESS, A_HIST_DEADBAND," +
            " A_HIST_COMPTYPE, A_HIST_COMPTIME!");
    }

    1 reference
    private static void writeCSV(string lineName = "line", int lastversion = 1)
    {
        // csv creation for tag export
        string fileName = lineName + "_v" + lastversion + "_tagImport.csv";
        string path = @"..\..\..\"; // Directory.GetCurrentDirectory();
        // csv writing
        try
        {
            using (StreamWriter outputFile = new StreamWriter(Path.Combine(path, fileName)))
            {
                if (analogRegisterList.Count > 1)
                {
                    foreach (string register in analogRegisterList)
                        outputFile.WriteLine(register);
                    outputFile.WriteLine();
                }
                if (digitalRegisterList.Count > 1)
                {
                    foreach (string register in digitalRegisterList)
                        outputFile.WriteLine(register);
                }
            }
        } catch (Exception ex) { Console.WriteLine(ex.ToString()); }
    }
}
}
```

Termina qui l'esposizione del codice relativo al client.

Il codice relativo all'HMI non sarà mostrato.

Conclusioni

Al termine di questo progetto di tesi possiamo dire di aver raggiunto l'obiettivo preposto, ossia la creazione di un'interfaccia uomo-macchina dinamica in grado di esplorare lo spazio di indirizzamento esposto da un server che implementa il protocollo OPC-UA; questa HMI può adattarsi ai server che compongono qualunque linea produttiva che rispecchi il modello linea – macchine – gruppi macchina – motori e sensori. Il risultato ottenuto è molto buono, utile e apprezzabile.

Inoltre, si è riusciti a sfruttare la potenza di OPC-UA anche per quanto riguarda i metodi: i server che implementano il protocollo possono offrire dei metodi che possono essere chiamati da remoto in maniera efficace e strutturata; questa funzionalità è stata inserita nell'HMI creata, andando a ottenere importanti risultati che possono essere di grande utilità in ottica produttiva.

Sviluppi futuri

Gli sviluppi futuri di questo progetto di tesi possono essere divisi, a seconda del tipo di innovazione o aggiornamento che si intende portare, in tre macro categorie: server, HMI/Client e nuove implementazioni del progetto.

Per quanto riguarda il server sono rimasti alcuni punti aperti (derivanti principalmente dalla carenza di supporto):

- La possibilità di inserire le traduzioni, soprattutto per campi come le descrizioni, tramite le variabili di tipo LocalizedText e il parametro Locale offerti dal protocollo OPC-UA. Il problema riscontrato è che l'implementazione ufficiale dell'SDK e il Model Compiler forniti dalla OPC Foundation ancora non supportano tali proprietà.
- L'esportazione di un modello che contenga solo tutti i tipi base (sensori, motori e in generale i dispositivi di campo) disponibili per le macchine dell'intera azienda. Questo dovrà poi essere importato nella creazione dei design model dei vari server in modo tale da accorciarne notevolmente il codice e ridurre gli errori. Questo porterebbe a una vera ed effettiva unificazione del design.
- Conseguentemente al punto precedente sarebbe possibile andare a simulare realmente le macchine di IMA nella loro completezza, senza essere limitati dalla complessità.

Guardando poi il lato HMI e Client, i passaggi futuri potrebbero essere:

- Il miglioramento dell'aspetto grafico dei componenti dell'HMI rendendo l'interfaccia estremamente più significante e comunicativa al primo impatto visivo; questa miglioria dovrebbe seguire i modelli delle attuali HMI fornite da IMA sulle proprie macchine.
- Il cambiamento della tecnologia alla base della comunicazione tra HMI e Client per l'invocazione dei metodi, rimuovendo i quattro accessi al database e sostituendoli, ad esempio, con una Web API in modo da risultare più veloci e più indipendenti rispetto a strumenti esterni all'applicazione.
- La rimozione della comunicazione con il Process DataBase (PDB) di iFIX: questo passaggio rende il progetto meno dinamico di quello che si vorrebbe in quanto ogni volta che la struttura di un server viene modificata, l'HMI ha necessità di essere interrotta, importare le tag sul PDB e infine essere riavviata. Eliminando questo passaggio l'HMI potrebbe essere sempre attiva e rispecchiare sempre la macchina senza la necessità di interventi esterni.
- Creazione di una nuova interfaccia con un codice che, anziché cercare gli oggetti nella picture andando a valorizzare tutti i vari campi presenti e mostrando/nascondendo oggetti, inizi con una picture praticamente vuota e tramite le funzioni BuildObject generi gli oggetti necessari. Questa soluzione dovrebbe poi essere confrontata con quella attuale in termini di performance per capire quale possa essere più veloce e meno invasiva rispetto al consumo di risorse. Inoltre, questa interfaccia si potrebbe andare ad adattare ancora meglio a qualsiasi tipo di struttura di un ipotetico server, potendo aggiungere livelli di dettaglio differenti e elementi magari non previsti nell'attuale HMI.

Infine, possiamo inserire delle osservazioni per quanto riguarda non tanto degli aggiornamenti al progetto ma più delle nuove implementazioni del progetto, che possiamo definire anche come innovazioni del progetto:

- L'attuale progetto di tesi è stato svolto sullo SCADA GE Digital iFIX. Un altro SCADA utilizzato da IMA su cui sarebbe interessante svolgere un progetto analogo è Rockwell FactoryTalk che potrebbe coprire una fetta di mercato che predilige tale sistema rispetto a iFIX.
- Il progetto di tesi si è svolto basandosi sul modello client/server offerto dal protocollo OPC-UA e questa tipologica di schema segue l'attuale modello di architettura implementato nelle linee di produzione IMA. Il protocollo offre però anche la possibilità di utilizzare un modello di tipo publish/subscribe: lo studio e l'implementazione di questo schema potrebbe portare alla luce nuovi vantaggi.

Bibliografia

[1] L. Bergamaschi, “Manuale di programmazione dei PLC” Ed. Hoepli, 2002

[2] Bimbo e Colaiacovo, “Sistemi SCADA” Ed.Apogeo, 2006

[3] S.A. Boyer, “SCADA Supervisory Control and Data Acquisition”, Isa-International Society of Automation, Englewood, USA, 2009

[4] R. Farana, L. Landryová, “Data Visualization of the SCADA/HMI System on the Internet/Intranet”, International Conference on Automation in Mining Slovak Republic: TU, 1998

[5] A. Daneels., W. Salter, (1999) “What is SCADA?” ICALEPCS'99, Trieste, October 1999

[6] S. Candori, “Supporto alla generazione automatica di interfacce per sistemi di controllo industriale”, 2013

[7] F. Di Giovanni, “Strumenti di progettazione di HMI/SCADA per macchine automatiche tramite l’utilizzo di tecnologie web”, 2020

[8] G. Tumedei, “Il protocollo OPC UA”, 2019

[9] L. Gemma, “Analysis of the OPC UA communication standard applied to control systems architecture”, 2020

[10] OPC Foundation Inc, “OPC 11030 -OPC Unified Architechture – UA Modelling Best Practises”, 2020

[11] Mariusz Postół (CAS), “AddressSpaceInterchangeXML”, ResearchGate, 2016

[12] GE Digital Inc., “iFIX Fundamentals – GFS-154H – Volume 1 of 2”, GE Digital, Charlottesville, VA, 2018

[13] GE Digital Inc., “iFIX Fundamentals – GFS-154H – Volume 2 of 2”, GE Digital, Charlottesville, VA, 2018

Sitografia

[1] https://reference.opcfoundation.org/

[2] https://github.com/OPCFoundation/UA-.NETStandard

[3] https://github.com/OPCFoundation/UA-ModelCompiler

[4] https://www.kepware.com/en-us/products/kepserverex/drivers/opc-ua-client/

[5] https://www.kepware.com/en-us/support/knowledge-base/2015/opc-ua-method-support/

[6] https://www.exorint.com/it/blog/che-cos%C3%A8-lo-standard-opc-ua-e-perch%C3%A9-verr%C3%A0-utilizzato-sempre-di-pi%C3%B9

[7] https://opcua.rocks/custom-information-models/

[8] https://opcconnect.opcfoundation.org/2017/04/using-nodeset-files-to-exchange-information/

[9] https://www.macheronte.com/automazione/protocolli/opc-ua/

[10] https://www.sielcosistemi.com/it/cosa-%C3%A8-scada.html