UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

ANALISI, CONFIGURAZIONE E TEST DI APPLICAZIONI OPC-UA IN AMBITO INDUSTRY 4.0

Analysis, configuration and testing of OPC-UA applications for industry 4.0

Relatore:
Prof. **Bonci Andrea**

Tesi di laurea di:
Elisei Nicola

Correlatore:
Ing. **Gagliardini Lorenzo**

ANNO ACCADEMICO 2020/2021

“Da umili origini verso grandi imprese.”

Sir. Francis Drake

INDICE

INDICE 1

INDICE DELLE FIGURE 4

INDICE DELLE TABELLE 7

INTRODUZIONE 8

CAPITOLO 1 L'AZIENDA 10

CAPITOLO 2 LO STANDARD OPC-UA 15

2.1. La nascita dello standard OPC-UA 15

2.2. Concetti fondamentali dello standard OPC-UA 15

2.3. Address Space Model (OPC-UA Part 3) 17

2.3.1. Variable NodeClass 19

2.3.2. Object NodeClass 20

2.3.3. Method NodeClass 20

2.3.4. View NodeClass 20

2.3.5. ObjectType e VariableType NodeClass 21

2.3.6. DataType NodeClass 22

2.3.7. ReferenceType NodeClass 24

2.4. Information Model (OPC-UA Part 5) 25

2.5. Services (OPC-UA Part 4) 26

2.6. Security (OPC-UA Part 2) 28

2.6.1. Creazione del canale 29

2.6.2. Creazione della sessione 30

2.7. OPC-UA nell'industria 4.0 31

2.8. OPC-UA nella lavorazione di materiale plastico 35

2.8.1. IMMToRobotType 37

CAPITOLO 3 CASO APPLICATIVO: MACCHINA AVVOLGITRICE 41
3.1. Strumenti utilizzati 41
3.1.1. Premium HMI 42
3.1.2. RDE (Robox Development Environment).....42
3.1.3. RDT (Robox Development Tool) 43
3.1.4. UAExpert 43
3.1.5. Robox RP2 44
3.1.6. Robox HMI (RHMI).....44
3.1.7. ASEM RK10.....45
3.2. Analisi dei requisiti e progettazione.....45
3.3. Sviluppo 48
3.3.1. Software RDE 49
3.3.2. Interfaccia grafica del terminale 53
3.3.3. Interfaccia grafica ed implementazione del Server OPC-UA.....56
3.4. Testing 60
CAPITOLO 4 CASO APPLICATIVO: ROBOT STANDARD 64
4.1. Strumenti utilizzati 64
4.1.1. UAModeler.....65
4.2. Sviluppo 66
4.2.1. Implementazione dell'Information Model per il robot standard.....66
4.2.2. Integrazione con il software RDE 68
CAPITOLO 5 CASO APPLICATIVO: SIMULATORE MACCHINA IMM 70
5.1. Strumenti utilizzati 71
5.1.1. OPC-UA Java SDK.....72
5.2. Sviluppo 73
5.2.1. Configurazione del Server OPC-UA.....74

5.2.2. Implementazione dell'interfaccia grafica 80

5.2.3. Implementazione della simulazione 83

5.3. Testing 89

CONCLUSIONI 97

BIBLIOGRAFIA 98

APPENDICE A CORRISPONDENZE EUROMAP 67/EUROMAP 79 100

APPENDICE B CODICE SIMULATORE PRESSA 101

RINGRAZIAMENTI 110

INDICE DELLE FIGURE

Figura 1: robot cartesiano Campetella della famiglia CX 11
Figura 2: robot cartesiano Campetella della famiglia ME 11
Figura 3: robot Campetella side-entry 12
Figura 4: robot cartesiano Campetella per applicazioni stack-molding 13
Figura 5: robot SCARA Campetella 13
Figura 6: sistemi di controllo Camptella, a sinistra la serie PRIME e a destra la serie EVO 14
Figura 7: specifiche dello standard OPC-UA.....16
Figura 8: esempio generico di nodi e riferimenti di un Address Space 17
Figura 9: definizione dell'oggetto Motor1 a partire dal tipo di oggetto MotorType 22
Figura 10: riferimenti tra le classi di nodi DataType, VariableType e Variable 23
Figura 11: gerarchia dei tipi di dato built-in per lo standard OPC-UA 24
Figura 12: gerarchia dei principali tipi di base che definiscono l'Informatio Model dello standard OPC-UA 25
Figura 13: architettura del modello di sicurezza adottato dallo standard OPC-UA 28
Figura 14: scambio di messaggio per la creazione del canale tra Client e Server 29
Figura 15 scambio di messaggio per la creazione della sessione tra Client e Server 30
Figura 16: architettura RAMI.....33
Figura 17: possibili configurazione per lo scambio di messaggi tra macchina IMM e robot di asservimento.....36
Figura 18: strutta del tipo di oggetto IMMToRobotType.....37
Figura 19: definizione delle variabili "IntermediatePosition" 39
Figura 20: controllore Robox RP2 44
Figura 21: terminale grafico RHMI 45
Figura 22: router ASEM RK10 45
Figura 23: infrastruttra di rete che permette lo scambio di dati tra macchina e gestionale del cliente 48
Figura 24: definizione dei registri del Robox RP2 riservati alla comunicazione OPC-UA50
Figura 25: ciclo di controllo dei dati OPC-UA 51

Figura 26: struttra della schermata visibile tramite il terminale RHMI 53
Figura 27: definzione delle variabili necessarie alla comunicazione OPC-UA per il terminale RHMI.....54
Figura 28: interfaccia grafica per l'accesso al Server OPC-UA.....57
Figura 29: variabili definite per l'interfaccia grafica di accesso al Server OPC-UA.....58
Figura 30: VB Script implementato per associare le variabili esposte dal Server OPC-UA con le variabili associate ai registri fisici del controllore Robox RP2 59
Figura 31: interfaccia grafica di accesso al Server OPC-UA a seguito dei test effettuati 61
Figura 32 interfaccia grafica visibile dal terminale RHMI a seguito dei test effettuati ..61
Figura 33: lampeggio giallo del pulsante "AVVIA NUOVA PRODUZIONE" 62
Figura 34: lampeggio rosso del pulsante "AVVIA NUOVA PRODUZIONE" 62
Figura 35: schermata di accesso ai dati del Server OPC-UA dal Client OPC-UA UAExpert 63
Figura 36: funzionalità del software UAModeler 65
Figura 37: definizione del tipo di oggetto MachineStateType tramite il software UAModeler 67
Figura 38: struttura del tipo di oggetto StandardRobotType.....68
Figura 39: associazione dei dati definiti tramite il Software UAModeler con i registri del controllore Robox RP2 69
Figura 40: simulatore del funzionamento di una macchina IMM basato su specifiche EUROMAP 67 70
Figura 41: interfaccia grafica configurata per il nuovo simulatore.....81
Figura 42: schermata di messaggio di inizializzazione del Server OPC-UA 82
Figura 43: test effettuato per verificare la corretta simulazione della chiusura dello stampo con posizioni intermedie disabilitate.....90
Figura 44: test effettuato per verificare la corretta simulazione della chiusura dello stampo con posizioni intermedie abilitate 91
Figura 45: test effettuato per verificare la corretta simulazione dell'apertura dello stampo con posizioni intermedie disabilitate.....91
Figura 46: test effettuato per verificare la corretta simulazione dell'apertura dello stampo con posizioni intermedie abilitate 92

Figura 47: test effettuato per verificare la corretta simulazione della ritrazione degli estrattori.....92
Figura 48: test effettuato per verificare la corretta simulazione dell'avanzamento degli estrattori.....93
Figura 49: test effettuato per verificare la corretta simulazione della ritrazione dei martinetti.....93
Figura 50: test effettuato per verificare la corretta simulazione dell'avanzamento dei martinetti.....94
Figura 51: test effettuato per verificare la corretta simulazione di un "GOOD CYCLE" .94
Figura 52: test effettuato per verificare la corretta simulazione di un "BAD CYCLE".....95
Figura 53: test effettuato per verificare la corretta simulazione del funzionamento in automatico disabilitato 95
Figura 54: test effettuato per verificare la corretta simulazione del funzionamento in automatico abilitato.....96
Figura 55: elenco delle corrispondenze da EUROMAP 67 a EUROMAP 79.....100

INDICE DELLE TABELLE

Tabella 1: struttra del NodeId 18
Tabella 2: attributi e property dei nodi di tipo Methods 20
Tabella 3: attributi della classe di nodi ObjectType 22
Tabella 4: attributi della classe di nodi VariableType 22
Tabella 5: esempio di enumerazione 23
Tabella 6: parametri caratterizzanti un messaggio di richiesta 26
Tabella 7: parametri caratterizzanti un messaggio di risposta 26
Tabella 8: requisiti di Industria 4.0 soddisfatti dallo standard OPC-UA 35
Tabella 9: definizione delle posizioni finali per gli assi di una macchina IMM 38
Tabella 10: enumerazione del tipo di variabile “MovementEnum” 39
Tabella 11: enumerazione del tipo di variabile “CycleQualityEnumeration” 40
Tabella 12: organizzazione logica dei dati relativi alla nuova produzione 46
Tabella 13: organizzazione logica dei dati relativi alla produzione attuale 46
Tabella 14: organizzazione logica dei dati relativi alla produzione precedente 47
Tabella 15: organizzazione logica dei dati relativi allo stato della macchina 47
Tabella 16: elenco degli stati macchina 55
Tabella 17: elenco dei messaggi macchina 56
Tabella 18: elenco delle stringhe definite nella realizzazione dell’interfaccia grafica di accesso al Server OPC-UA 60
Tabella 19: elenco degli Information Model necessari al caricamento dello specifiche EUROMAP 79 76
Tabella 20: elenco degli stati implementati per la simulazione automatica della macchina IMM 88

INTRODUZIONE

Negli ultimi anni è in atto una corsa all'automazione dei processi industriali, che ha visto l'introduzione nelle fabbriche di un numero sempre crescente di macchine, computer, robot e sistemi di controllo. Con l'avvento di Industria 4.0 il numero di oggetti interconnessi e comunicanti negli impianti è aumentato molto ed è destinato ad aumentare ulteriormente, così come la quantità di dati raccolti e le tipologie di soluzioni volte alla memorizzazione, elaborazione e presentazione di tali dati. Allo stesso tempo la normazione delle specifiche per l'interoperabilità in rete dei sistemi e dei dispositivi industriali, ha iniziato ad evolvere nella stessa direzione, con particolare riferimento alla pianificazione di future architetture aperte, dinamiche e flessibili in tali sistemi. Occorre inoltre considerare la necessità di coniugare l'operatività dei componenti già presenti nell'industrie con le nuove soluzioni. In questo scenario, lo standard OPC-UA (Open Platform Communications – Unified Architecture) si pone come principale candidato per soddisfare rigorosi vincoli di interoperabilità e condivisione dell'informazione, rispettando la gestione real-time della risposta. Per questi motivi, l'oggetto del presente elaborato è lo sviluppo di applicazioni in ambito Industria 4.0, basate sullo scambio di dati e di informazioni tramite l'utilizzo dello standard OPC-UA. Più nello specifico, il presente documento è volto a descrivere il lavoro svolto presso la Campetella Robotic Center srl, nel quale sono state svolte attività di configurazione e testing della comunicazione OPC-UA in una serie di macchine industriali, di tipo avvolgitrice, al fine di scambiare una lista di informazioni con i sistemi informativi gestionali dell'attuale parco clienti dell'azienda. Tale procedura è stata replicata e applicata nell'ambito di prodotti industriali di serie, quali robot cartesiani e SCARA, di fabbricazione Campetella. Per poter raggiungere tali obiettivi, in via preliminare sono stati analizzati i principi fondanti dello standard OPC-UA, contestualizzandone l'utilizzo nell'ambito di Industria 4.0 e della lavorazione di materiale platico, core business dell'azienda. Il documento redatto è diviso in cinque capitoli. Nel primo capitolo è presentata la Campetella Robotic Center srl, realtà marchigiana nella progettazione e produzione di macchine automatiche industriali, presso cui è stato svolto il lavoro oggetto di questo elaborato. È riportata inoltre un'analisi e descrizione della struttura e del funzionamento delle principali serie di robot "standard" e delle macchine avvolgitrici di fabbricazione Campetella. Nel secondo capitolo è riportata l'analisi effettuata dello standard di comunicazione OPC-UA, ponendo l'attenzione sulle funzionalità principali e di utilità per il successivo sviluppo delle applicazioni che ne

fanno uso e sul contributo che apporterà nell'ambito della quarta rivoluzione industriale e nell'ambito della comunicazione tra presse ad iniezione di materiale plastico e loro sistemi di asservimento. Nel terzo capitolo sono descritte le attività svolte per l'analisi, lo sviluppo ed il testing della configurazione della comunicazione OPC-UA per la macchina avvolgitrice. In questo primo caso applicativo, il Server OC-UA è configurato realizzando un'interfaccia grafica, tramite l'utilizzo del software di sviluppo Premium HMI. Sono infine riportati i test effettuati per verificare la corretta configurazione della comunicazione, sia testando la stessa utilizzando il Client OPC-UA UAExpert, sia esaminando la comunicazione di tutta l'infrastruttura eseguendo test a bordo macchina. Il quarto capitolo, invece, descrive una possibile alternativa per la configurazione del Server OPC-UA andando ad agire direttamente sul controllore della macchina, richiedendo pertanto l'uso del software di configurazione proprietario del controllore stesso. Per questo caso applicativo la configurazione del Server OPC-UA è effettuata per essere applicabile ai robot standard, come robot cartesiani e SCARA. Infine, nel quinto ed ultimo capitolo è descritto lo sviluppo di un'applicazione in linguaggio Java, comprensivo di interfaccia grafica, che permetta di simulare il funzionamento di una macchina per lo stampaggio ad iniezione di materiale plastico, macchina IMM (Injection Moulding Machine). Tale applicazione è volta ad aggiornare il dispositivo precedentemente sviluppato dall'azienda per simulare il funzionamento della macchina IMM, replicandone le medesime funzioni, ma basando il tutto sui concetti espressi dallo standard OPC-UA. Per verificare la corretta configurazione del simulatore realizzato in Java sono, infine, riportati i test effettuati utilizzando il Client OPC-UA UAExpert.

CAPITOLO 1

L'AZIENDA

L'azienda presso cui si è svolto il lavoro di configurazione e test di applicazioni OPC-UA è la Campetella Robotic Center srl. Fondata nel 1897, la Campetella Robotic Center è un'azienda italiana specializzata nella progettazione, produzione, installazione negli impianti, vendita e post-vendita del prodotto robot per l'industria. Si tratta di un'azienda a conduzione familiare, ma che può essere considerata una media impresa a tutti gli effetti. A fondamento della produzione e progettazione di macchine automatiche per l'industria vi è una ricerca continua, volta allo sviluppo di soluzioni che favoriscano l'utilizzo delle migliori tecnologie e garantiscano la semplicità d'uso del prodotto. Inoltre la Campetella Robotic Center dedica particolare attenzione al post-vendita, organizzando appositi corsi ed affiancamenti per garantire un conoscenza approfondita del prodotto ed un'assistenza e supporto al cliente di alta qualità e rapidità. L'esperienza maturata nel mondo dell'automazione industriale ha portato i robot Campetella ad essere scelti dai maggiori gruppi industriali nazionali ed internazionali e pertanto a rappresentare un punto di riferimento per l'Industria 4.0. La Campetella Robotic Center offre un'ampia varietà di robot per rispondere alle particolari esigenze dei diversi campi applicativi delle materie plastiche, aiutando i clienti ad aggiungere velocità, flessibilità ed affidabilità al processo produttivo. Il core business dell'azienda è rappresentato da soluzioni robotizzate per il settore dello stampaggio delle materie plastiche. Le principali tipologie di robot, sviluppate e realizzate internamente alla Campetella Robotic Center, sono: robot cartesiani (top-entry), robot laterali (side-entry), robot stack-mold e SCARA. L'azienda vanta una vasta gamma di robot cartesiani che si differenziano tra loro sulla base di particolari caratteristiche che derivano dall'incrocio tra allestimento e famiglia meccanica. Sono disponibili tre differenti tipi di allestimento:

- Prime Series: dedicato ai robot di piccola taglia, semplici ma allo stesso tempo funzionali.
- E-Series: robot di piccola e media taglia, con funzionalità avanzate e massima accuratezza di posizionamento.
- X-Series: dedicato ai robot di media e grade taglia, dotati delle massime prestazioni in termini di funzionalità, velocità e controllo.

La meccanica dei robot cartesiani si divide invece fra due differenti configurazioni. Nella prima, quella delle famiglie CL, CO e CX, di cui un esempio è riportato in Figura 1, l'asse di estrazione (asse X) è fissato a sbalzo rispetto all'asse trasversale (asse Z); l'estrazione è quindi possibile grazie al movimento del solo asse verticale (asse Y), libero di scorrere lungo le guide dell'asse di estrazione. Il principale vantaggio di questa configurazione è dato dall'assenza di problemi di ingombro dal lato opposto al lato di estrazione.

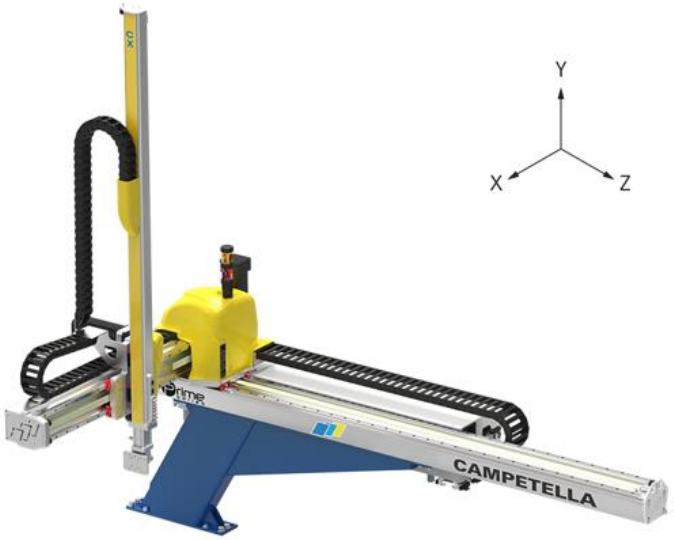


Figura 1: robot cartesiano Campetella della famiglia CX

Nella seconda configurazione della meccanica, quella delle famiglie ME, MC e GS, di cui è visibile un esempio in Figura 2, l'asse verticale (asse Y) è vincolato frontalmente all'asse di estrazione (asse X); l'estrazione è quindi possibile grazie allo scorrimento dell'asse di estrazione lungo il carrello dell' asse trasversale (asse Z). Questa configurazione è caratterizzata dall'essere esente da problemi di collisione.

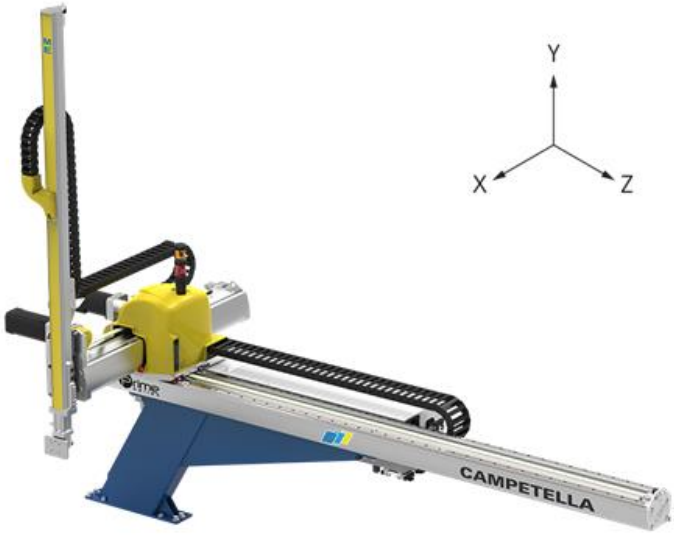


Figura 2: robot cartesiano Campetella della famiglia ME

Entrambe le configurazioni appena descritte sono accomunate dalla medesima architettura top-entry, ovvero con entrata ed uscita in verticale rispetto al punto di estrazione interno allo stampo.

Mantenendo le medesime caratteristiche di linearità e semplicità di programmazione del robot cartesiano, i robot laterali, di cui un esempio è visibile in Figura 3, grazie alla loro particolare architettura side-entry (entrata ed uscita laterale allo stampo) permettono di minimizzare il percorso degli assi dall'area di estrazione verso l'area di deposito. Sono soluzioni dotate di una struttura portante a terra che mitiga completamente l'effetto delle vibrazioni derivanti dalla pressa, garantendo la massima precisione anche per velocità elevate.

Figura 3: robot Campetella side-entry

L'azienda offre inoltre un'innovativa e raffinata gamma di robot, sia top-entry che side-entry, progettata per esprimere il massimo delle prestazioni nelle applicazioni stack-molding. La tecnologia stack-molding consiste in una particolare architettura delle macchine per lo stampaggio ad iniezione, che consente di distribuire il materiale plastico su due o più piani, raddoppiando di fatto la produttività della macchina pur lasciando invariati i tempi necessari allo stampaggio. Questa tecnologia offre indiscutibili vantaggi in quei settori ad alta produttività, ottimizzando i costi aziendali e abbattendo i consumi energetici e le emissioni. Uno dei diversi modelli di robot progettati per questo tipo di applicazioni è mostrato in Figura 4.

Figura 4: robot cartesiano Campetella per applicazioni stack-molding

Un'ulteriore tipologia di robot sviluppata dall'azienda è quella degli SCARA (Selective Compliance Assembly Robot Arm). Si tratta di un robot industriale con una cinematica simile a quella del braccio umano e che pertanto fornisce un'elevata destrezza al robot. Come visibile in Figura 5, lo SCARA è caratterizzato da tre assi rotazionali e da un asse lineare per gli spostamenti in verticale. Il punto di forza degli SCARA di fabbricazione Campetella è sicuramente rappresentato dalla particolare progettazione dell'asse verticale, il quale consente lo scorrimento dell'intero braccio lungo la colonna portante del robot. Tale caratteristica porta lo SCARA Campetella ad essere considerato come valida alternativa dei robot antropomorfi, in particolare in termini di ingombro.

Figura 5: robot SCARA Campetella

Inoltre, l'azienda si occupa anche della progettazione e realizzazione delle unità di controllo con cui equipaggiare i robot appena citati e descritti. Due diverse categorie di sistemi di controllo (PRIME e EVO), visibili in Figura 6, consentono di programmare e gestire in modo semplice e sicuro i robot e gli impianti di produzione tramite un'interfaccia intuitiva, comandi intelligenti e un design ergonomico.

Figura 6: sistemi di controllo Camptella, a sinistra la serie PRIME e a destra la serie EVO

Parallelamente alla produzione in linea dei robot appena descritti, che possono essere definiti come prodotti standard "da catalogo", la Campetella Robotic Center si occupa anche della progettazione e realizzazione di macchine automatiche industriali per applicazioni customizzate e personalizzate, al fine di soddisfare le esigenze dei suoi clienti. Un esempio di queste applicazioni è rappresentato dalle macchine avvolgitrici per la realizzazione e pallettizzazione di bobine di tubi di plastica. In particolare, questa tipologia di macchine viene progettata nei minimi dettagli in quanto si deve integrare con la linea di produzione specifica del cliente. Solitamente sono caratterizzate da due avvolgitori che possono lavorare sia uno in serie all'altro sia in parallelo ed, eventualmente, anche su lotti di produzioni differenti l'uno dall'altro. Sono poi dotate di un braccio meccanico che si occupa dell'estrazione della bobina dall'avvolgitore e del suo trasporto su un nastro automatico per lo scarico. Il tutto è racchiuso all'interno di una struttura ben salda a terra al fine di garantire il funzionamento della macchina in piena sicurezza.

Durante il lavoro svolto per l'azienda, le applicazioni sviluppate in ambito Industria 4.0 per lo scambio di dati ed informazioni, che fossero di utilità per i clienti finali dei robot Campetella, tramite l'uso dello standard di comunicazione OPC-UA sono state studiate e realizzate per essere integrate sia su macchine custom (Capitolo 3), sia su macchine standard "da catalogo" (Capitolo 4).

CAPITOLO 2

LO STANDARD OPC-UA

2.1. La nascita dello standard OPC-UA

Il primo standard di grande successo per la comunicazione di dati per il controllo di processo e nelle applicazioni per il settore dell'automazione industriale è stato OPC Data Access COM/DCOM, noto ad oggi come Classic OPC. Questo standard è stato introdotto intorno all'anno 1996 dalla OPC Foundation, una task-force no-profit di produttori del settore dell'automazione, nata nel 1994 con lo scopo di definire un nuovo standard che fornisse un'interfaccia comune per leggere e scrivere dati sui driver dei dispositivi industriali. Una delle maggiori ragioni che ha portato al successo di questo standard è stata l'adozione della tecnologia Microsoft COM/DCOM. Questo fattore d'altro canto ha rappresentato per lo standard anche una delle sue maggiori limitazioni, in quanto lo teneva legato all'utilizzo di piattaforme Windows. Il primo tentativo di creare una versione dello standard indipendente dalla piattaforma è stato nel 2003, quando la OPC Foundation ha rilasciato la standard OPC XML-DA che rimpiazzava la tecnologia COM/DCOM con tecnologie quali HTTP/SOAP e Web Services. Questo nuovo standard però presentava notevoli limitazioni dal punto di vista delle performance, pertanto la OPC Foundation si è rimessa subito alla ricerca di uno standard in grado di rimpiazzare definitivamente tutte le tipologie precedenti, mantenendo le ottime performance di Classic OPC ma superandone le limitazioni. Nel 2006, la OPC Foundation rilascia la prima versione dello standard OPC-UA. Ad oggi, la OPC Foundation sta continuando ad aggiornare regolarmente OPC-UA, arrivato alla sua versione 1.04 rilasciata per la prima volta nel 2017.

2.2. Concetti fondamentali dello standard OPC-UA

OPC-UA (*Open Platform Communications – Unified Architecture*) è uno standard di comunicazione che nasce dalla volontà di standardizzare lo scambio di dati ed informazioni tra controllori logici programmabili (PLC), interfacce uomo-macchina (HMI), server, client e altri macchinari indipendentemente dal sistema operativo, dal linguaggio di programmazione e dal produttore. Esso è basato, da un lato, sull'esperienza del primo standard Classic OPC, dall'altro, tenendo in considerazione desideri e osservazioni dei suoi utenti.

Le principali ragioni che hanno portato allo sviluppo di OPC-UA sono dunque riassumibili nei seguenti dieci punti:

1. Interruzione dello sviluppo della tecnologia COM/DCOM
2. Limiti della tecnologia COM/DCOM
3. Necessità di stabilire comunicazioni attraverso Firewall
4. Utilizzo in ambito industriale di piattaforme non-Windows
5. Alte prestazione nelle comunicazioni tramite Web Services
6. Modello dei dati unificato
7. Supporto di strutture dati complesse
8. Affidabilità nella comunicazione dei dati
9. Protezione contro l'accesso ai dati non autorizzato
10. Supporto alle chiamate di metodi

Lo standard OPC-UA è descritto da un insieme di specifiche (OPC-UA Specifications) strutturato in più parti, come visibile in Figura 7, che possono essere raggruppate in 3 differenti categorie: *Core Specification Part, Access Type Specification Part, Utility Specification Part*. Nel Core Specification Part sono definiti i concetti principali: la struttura dello spazio degli indirizzi (Address Space) ed i servizi (Services) che permettono di operare su di esso. Nell'Access Type Specification Part i concetti descritti nella prima categoria sono applicati a specifiche tipologie di accesso, descritte nelle specifiche del Classic OPC, mentre nell'Utility Specification Part sono esposte le funzionalità addizionali dell'OPC-UA.

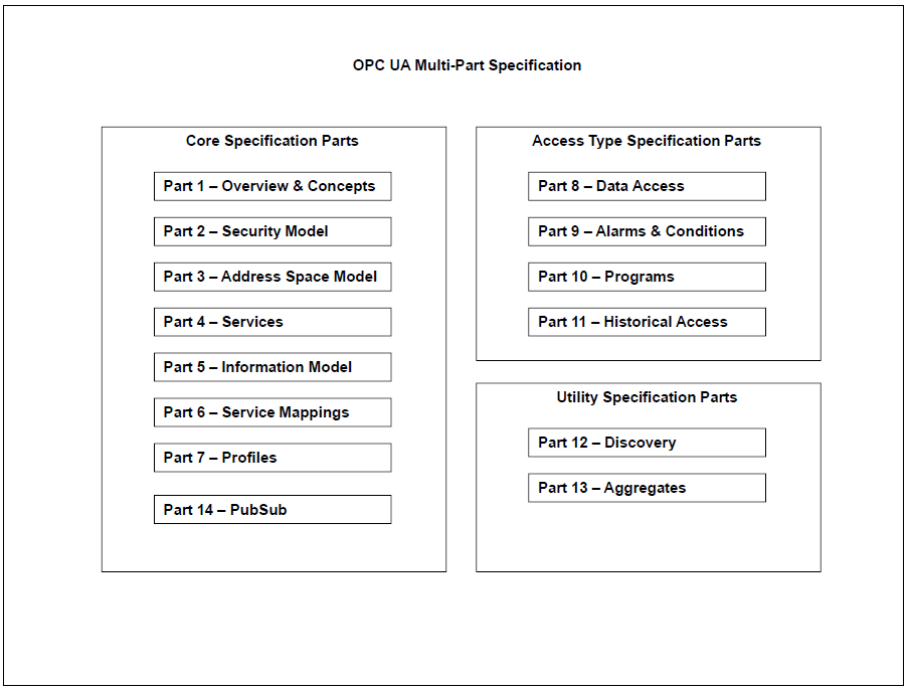


Figura 7: specifiche dello standard OPC-UA

Nel seguito saranno descritte solo le parti principali e le parti di interesse ed utilità ai fini dello svolgimento del lavoro svolto presso la Campetella Robotic Center Srl., descritto nei capitoli successivi. Per quanto siano complete e dettagliate, le OPC-UA Specifications sono descritte secondo una struttura particolarmente complessa, pertanto non rappresentano il miglior modo per approcciarsi ad OPC-UA. Questo elaborato tratta quindi i concetti introdotti dallo standard seguendo il libro "OPC – From Data Access to Unified Architecture" degli autori Jürgen Lange, Frank Iwanitz e Thomas Burke, integrando dove ritenuto opportuno con le OPC-UA Specifications. Un altro testo di riferimento, indicato dalla OPC Foundation per approcciare allo studio dello standard OPC-UA, è il libro "OPC Unified Architecture" redatto da Wolfgang Mahnke, Stefan-Helmut Leitner e Matthias Damm.

2.3.Address Space Model (OPC-UA Part 3)

Le fondamenta dell'Address Space nello standard OPC-UA sono i Nodes (nodi) e le References (collegamenti). I Nodes rappresentano le informazioni disponibili sul Server OPC-UA e per qualunque Client OPC-UA sottoforma di variabili (Variables), oggetti (Objects) e tipi di dato (Data Types). Le References rappresentano le relazioni che intercorrono tra due Nodes e possono essere di differenti tipi. Il nodo referenziato dalla Reference è detto nodo target, mentre il nodo da cui ha origine la Reference è detto nodo sorgente. Un esempio di Nodes e References è visibile in Figura 8, dalla quale si nota che ogni Nodes è descritto da un insieme di Attributes (attributi). Ogni attributo è contraddistinto da: ID, nome, descrizione, tipo di dato e informazioni circa lo stato (obbligatorio/opzionale). L'insieme dei Nodes e delle References caratterizza l'Address Space dell'OPC-UA.

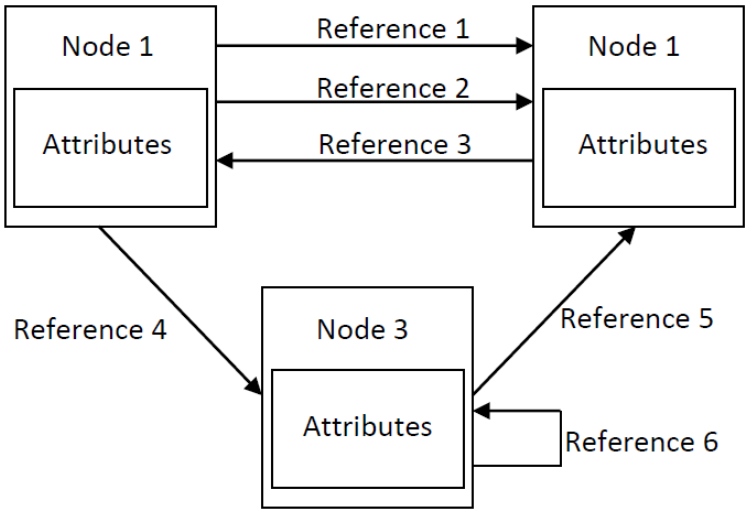


Figura 8: esempio generico di nodi e riferimenti di un Address Space

Per definire la natura dei Nodes esistono varie classi di nodi (Node Classes), ma tutti i Nodes appartengono in primo luogo alla stessa *BaseNodeClass*. La *BaseNodeClass* è descritta dai seguenti attributi, che dunque saranno comuni a tutti i Nodes dell'Address Space di OPC-UA.

- NodeId: ha una struttura ben definita, visibile in Tabella 1, e permette di individuare in modo univoco un nodo all'interno dell'Address Space.

NodeId	Descrizione
namespaceIndex	È l'indice del namespace
identifierType	Indica il formato ed il tipo del campo identifier
identifier	È l'identificatore del nodo all'interno dell'Address Space

Tabella 1: struttra del NodeId

- NodeClass: identifica la classe a cui appartiene il nodo.
- BrowseName: viene utilizzato per identificare il nodo all'interno dell'Address Space, rendendolo più leggibile all'utente, quando si utilizza il servizio Browse; è composto dal namespaceIndex e da un campo di tipo stringa che riporta il nome del nodo.
- DisplayName: è il nome del nodo visibile ai Client.
- Description: contiene una descrizione testuale del nodo.
- WriteMask: maschera che specifica quali attributi del nodo possono essere modificati dal Client; non tiene in considerazione la modalità di accesso dell'utente.
- UserWriteMask: maschera che specifica quali attributi del nodo possono essere modificati dal Client; tiene in considerazione la modalità di accesso dell'utente.
- RolePermission: specifica i permessi da applicare al nodo per ogni ruolo che ha accesso a quel nodo, dove per ruolo (Role) si intende la funzione che assume un Client quando accede ad un Server. In questo modo si riesce a separare tra autenticazione, per determinare l'identità del Client, e autorizzazione, per determinare cosa è permesso fare ad un Client.
- UserRolePermission: specifica i permessi da applicare ad un nodo per ogni ruolo concesso per la sessione in corso.
- AccessRestrictions: specifica le restrizioni da applicare al nodo.

Tra gli attributi appena elencati, è necessario porre una maggiore attenzione sul NodeId al fine di specificare al meglio alcune delle sue caratteristiche. Come mostrato nella Tabella 1, il NodeId è formato da tre campi, tra i quali quello di maggior interesse è il namespaceIndex. Quest'ultimo individua l'indice del namespace, associato ad un URI (Uniform Resource Identifier) che identifica l'autorità responsabile dell'assegnazione degli

identificatori per i vari NodeId, ed introduce implicitamente una semplificazione nell'accesso alle informazioni disponibili sul Server. Ogni Server infatti espone una variabile chiamata NamespaceArray, definita come un array contenente per ogni indice un namespace URI e che svolge il compito di una look-up table. In questo modo, per accedere ad un determinato nodo di un namespace, non è necessario essere a conoscenza del namespace URI, ma è sufficiente conoscere l'indice intero (namespaceIndex) che identifica quel namespace URI all'interno della variabile NamespaceArray. Tutti i nodi che compongono l'Address-Space di un Server OPC-UA possono poi essere distinti in 8 classi differenti: Variable, Object, Method, View, ObjectType, VariableType, DataType e ReferenceType. Per ognuna di queste classi le OPC-UA Specifications definiscono l'insiemi di attributi che saranno esposti dal nodo e i riferimenti che il nodo può avere con altri nodi.

2.3.1. Variable NodeClass

La Variable NodeClass viene utilizzata per rappresentare valori, semplici o complessi. Una Variable è sempre definita come DataVariable o Property di un altro nodo appartenente all'Address Space: si utilizza DataVariable per rappresentare i dati di un nodo, mentre si utilizza Property per rappresentare caratteristiche aggiuntive del nodo. Si supponga ad esempio di voler rappresentare la misura ottenuta da un sensore: per rappresentare il valore letto dal sensore si utilizza il tipo DataVariable, mentre per indicare l'unità di misura, il fondo scala o la sensitività del sensore si utilizza il tipo Property. Gli attributi principali che definiscono la Variable NodeClass sono i seguenti:

- Value: rappresenta il valore più recente della Variable.
- DataType: questo attributo contiene il NodeId che rappresenta il tipo di dato della Variable.
- ValueRank: specifica se il campo Value è un array e ne indica il numero di dimensioni.
- AccessLevel: indica la tipologia di accesso (lettura e/o scrittura) del campo Value; non tiene in considerazione la modalità di accesso dell'utente.
- UserAccessLevel: indica la tipologia di accesso (lettura e/o scrittura) del campo Value; tiene in considerazione la modalità di accesso dell'utente.
- Historizing: indica quando il Server è abilitato a collezionare i dati per lo storico della Variable.

2.3.2. Object NodeClass

La Object NodeClass viene utilizzata per definire un nodo Object (oggetto). Gli Objects sono fondamentali nella definizioni dell'Address-Space di un Server OPC-UA, in quanto sono utilizzati per rappresentare sistemi, componenti di sistemi, hardware e software; pertanto a tale scopo, gli Object possono contenere Variables, Methods e/o altri Objects. La Object NodeClass, oltre agli attributi che eredita dalla BaseNodeClass, possiede solo l'attributo EventNotifier. Quest'ultimo specifica se l'oggetto in questione può essere usato per sottoscriversi agli eventi (Events), in particolare l'attributo EventNotifier è di tipo Byte quindi è solamente il primo bit, denominato SubscribeToEvents, ad indicare o meno la sottoscrizione agli eventi. Un evento rappresenta il verificarsi di una particolare situazione all'interno di un sistema, come ad esempio un cambio di configurazione o un errore di sistema, e in OPC-UA questi eventi non sono direttamente accessibili esplorando l'Address-Space. Solo i nodi Objects e Views (descritti in seguito) permetto la sottoscrizione ad un evento tramite il settaggio dell'attributo EventNotifier.

2.3.3. Method NodeClass

I Methods (metodi) rappresentano funzioni richiamabili tramite il servizio Call dello standard OPC-UA. Gli argomenti di input ed output di un Method non sono attributi della Method NodeClass, ma sono rappresentati da variabili di tipo Property. La Tabella 2, visibile di seguito, riporta e descrive gli attributi e le Property che caratterizzano la Method NodeClass.

Attribute	DataType	Descrizione
Executable	Boolean	Indica se il metodo è eseguibile; non tiene in considerazione il metodo di accesso dell'utente.
UserExecutable	Boolean	Indica se il metodo è eseguibile; tiene in considerazione il metodo di accesso dell'utente.
Property		
InputArguments	Argument[]	Definisce l'array di argomenti di input del metodo.
OutputArguments	Argument[]	Definisce l'array di argomenti di output del metodo.

Tabella 2: attributi e property dei nodi di tipo Methods

2.3.4. View NodeClass

L'Address-Space di un server OPC-UA è spesso formato da un insieme vasto di Nodes e, generalmente, un Client è interessato solo ad un particolare sottoinsieme di Nodes che

compongono l'Address-Space. Per risolvere tale problematica, lo standard OPC-UA mette a disposizione il concetto di View (vista). Una View è un sottoinsieme di nodi che compongono l'intero Address-Space. Di fatto è possibile vedere una View come un vero e proprio Address-Space composto dai soli nodi di interesse per un dato Client, in cui ogni nodo sarà presente con le sole Reference che lo collegano agli altri nodi presenti nella View. In questo modo ogni nodo che caratterizza la vista sarà accessibile a partire dal nodo radice, detto ViewNode, per quella stessa View. Le Views sono definite a partire dalla View NodeClass, i cui attributi principali sono riportati di seguito:

- ContainsNoLoop: indica se i nodi contenuti nella vista formano una struttura gerarchica senza alcun loop, ciò implica che una volta visitato un nodo appartenente alla vista e proseguendo nella gerarchia quel nodo non sarà visitabile una seconda volta.
- EventNotifier: come per gli Object, indica se il nodo può essere usato o meno per sottoscriversi agli eventi.

2.3.5. ObjectType e VariableType NodeClass

Un'importante funzionalità dello standard OPC-UA è data dalla possibilità di definire tipi di oggetto (ObjectType) e tipi di variabile (VariableType). Per descrivere le caratteristiche e i vantaggi di questa funzionalità si usa il termine TypeDefinition. Una TypeDefinition permette di definire sia la semantica di un oggetto o di una variabile, sia la struttura dei nodi che compongono il tipo di oggetto o il tipo di variabile. Il vantaggio principale delle TypeDefinition è rappresentato dal poter definire il tipo di oggetto o il tipo di variabile una singola volta, per poi andarlo a referenziare, tramite l'apposita Reference "HasTypeDefinition", in tutte le occasioni che si intende definire un oggetto o una variabile di quel tipo. Ad esempio, si supponga di volere definire un oggetto che rappresenti un motore (Motor). Inizialmente si deve definire il tipo di oggetto MotorType indicandone la struttura che lo caratterizza, ovvero il numero di nodi che lo compongono e per ogni nodo indicarne il tipo (un ObjectType può contenere nodi di tipo Object, Variable e Method, mentre una VariableType può contenere solo nodi di tipo Variable). A seguito di ciò, è possibile quindi creare l'oggetto Motor e istanziarlo al tipo di oggetto MotorType, al fine di fargli ereditare la medesima struttura. Un esempio di quanto appena descritto è visibile in Figura 9, riportata di seguito.

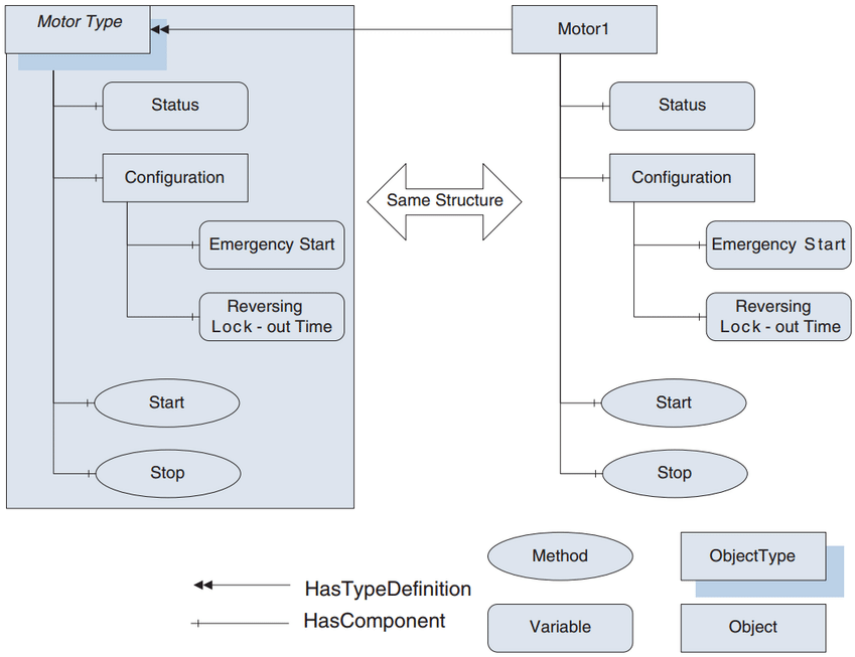


Figura 9: definizione dell'oggetto Motor1 a partire dal tipo di oggetto MotorType

Gli attributi principali che caratterizzano la ObjectType NodeClass e la VariableType NodeClass sono riportati rispettivamente nella Tabella 3 e nella Tabella 4.

Attribute	DataType	Descrizione
IsAbstract	Boolean	Specifica se il tipo di oggetto è astratto o meno.

Tabella 3: attributi della classe di nodi ObjectType

Attribute	DataType	Descrizione
DataType	NodeId	Contiene il NodeId che rappresenta il tipo di dato per le istanze di questo tipo di variabile.
ValueRank	Int32	Specifica se il campo Value del tipo di variabile è un array e ne indica il numero di dimensioni; per le VariableType l'attributo Value è opzionale.
IsAbstract	Boolean	Specifica se il tipo di variabile è astratto o meno.

Tabella 4: attributi della classe di nodi VariableType

2.3.6. DataType NodeClass

Oltre a tipi di oggetto e tipi di variabile, OPC-UA offre la possibilità di definire tipi di dato (DataType) che vengono utilizzati per descrivere l'attributo Value delle variabili e dei tipi di variabile. Infatti, come riportato in Figura 10, il tipo di variabile (VariableType) e la variabile (Variable), che è un'istanza di quel tipo di variabile, puntano allo stesso tipo di dato (DataType), appositamente scelto in fase di definizione per rappresentare e descrivere il tipo di variabile stesso.

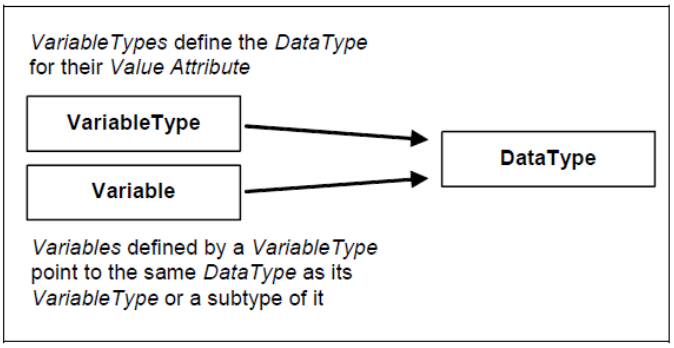


Figura 10: riferimenti tra le classi di nodi DataType, VariableType e Variable

Lo standard OPC-UA distingue i tipi di dato in 4 differenti categorie:

- Built-in: si tratta di un set di tipi di dato pre-definito e non estendibile; comprende i tipi di base come Int32, Double e Boolean, ma anche tipi di dato strutturati e specifici di OPC-UA come NodeId.
- Simple: sono sottotipi dei tipi built-in; un esempio è il tipo di dato UtcTime sottotipo di DateTime, utilizzato per rappresentare data e ora secondo il fuso orario globale.
- Structured: sono tipi di dato complessi e strutturati definiti dallo standard OPC-UA o dall'utente; un esempio è il tipo Argument usato per rappresentare i parametri dei metodi.
- Enumeration: questa categoria raggruppa quei tipi di dato che vengono usati per mappare un set di stringhe (nomi o messaggi) su dei valori interi di tipo Int32; un esempio di questo tipo di dato è il NodeClass, visibile nella Tabella 5.

Value (Int32)	EnumStrings
1	Object
2	Variable
3	Method
4	ObjectType
5	VariableType
6	ReferenceType
7	DataType
8	View

Tabella 5: esempio di enumerazione

La Figura 11 mostra la gerarchia dei tipi di dato built-in e mette in evidenza una parte dei tipi di dato "simple".

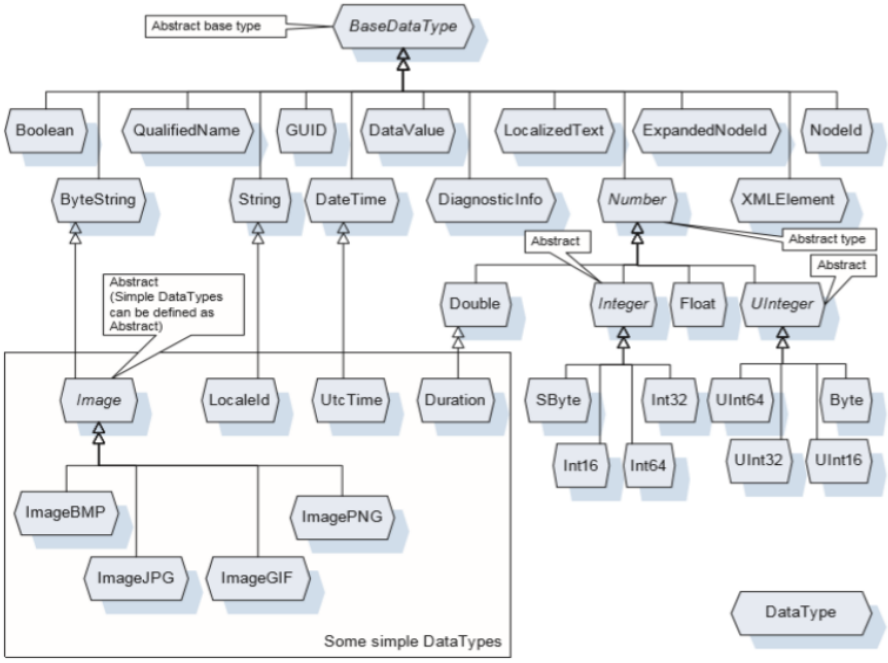


Figura 11: gerarchia dei tipi di dato built-in per lo standard OPC-UA

I tipi di dato sono definiti a partire dalla DataType NodeClass, da cui ne ereditano l'attributo IsAbstract: un booleano che specifica se il tipo di dato è astratto o meno. Sia i tipi di dato astratti che i tipi di dato non astratti possono essere usati nella definizione di una variabile o di un tipo di variabile.

2.3.7. ReferenceType NodeClass

Come descritto in precedenza, una Reference (riferimento) rappresenta la relazione che intercorre tra due nodi appartenenti all'Address Space. A differenza degli oggetti, delle variabili e dei metodi, un riferimento non ha attributi o proprietà che la contraddistinguono; infatti, per definire la semantica di un riferimento si utilizzano direttamente i ReferenceType. Gli attributi principali che caratterizzano la ReferenceType NodeClass sono:

- IsAbstract: questo attributo specifica se la ReferenceType è astratta o meno; in questo caso, a differenza di quanto accade nei tipi di dato, se la ReferenceType è astratta può essere istanziata, ma può essere utilizzata solo per ragioni organizzative.
- Symmetric: indica se la ReferenceType ha lo stesso significato in entrambe le direzioni (dal nodo sorgente al nodo target e viceversa).
- InverseName: nel caso la ReferenceType sia asimmetrica, specifica la semantica del riferimento nella direzione inversa (dal nodo target al nodo sorgente).

Inoltre, di particolare importanza nella descrizione della semantica della ReferenceType NodeClass è l'attributo BrowseName, che come descritto nel paragrafo 2.3, eredita dalla BaseNodeClass. In tal caso infatti, il BrowseName è utilizzato anche per specificare il significato della ReferenceType partendo dal nodo sorgente. Ad esempio, il valore del BrowseName "Contains" viene utilizzato nella ReferenceType che indica che il nodo sorgente contiene il nodo target. Per tale motivo il BrowseName di una ReferenceType deve essere unico all'interno dell'Address Space.

2.4.Information Model (OPC-UA Part 5)

Questa sezione delle specifiche dello standard OPC-UA definisce e descrive i nodi standard di base che compongono l'Address Space di un Server OPC-UA. In altri termini, si può dire che l'Information Model definisce lo scheletro dell'Address Space di un Server OPC-UA. In Figura 12 è riportata la gerarchia dei principali tipi di base dell'Information Model. L'Information Model dello standard OPC-UA è stato progettato per garantire l'estendibilità dell'Address Space, in particolare permette di definire tipi di oggetto (ObjectType), tipi di variabile (VariableType) e tipi di dato (DataType) ereditando rispettivamente dai tipi di base BaseObjectType, BaseVariableType e BasaDataVariableType.

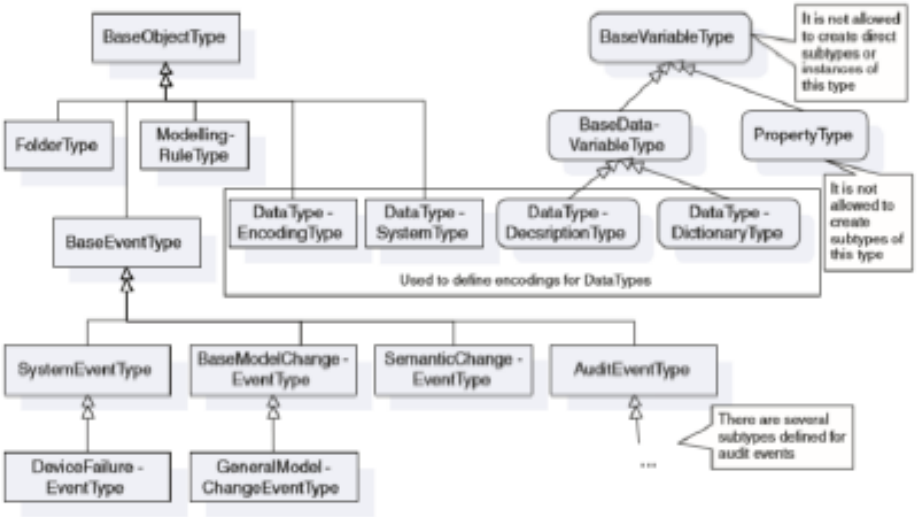


Figura 12: gerarchia dei principali tipi di base che definiscono l'Informatio Model dello standard OPC-UA

La OPC Foundation mette a disposizione alcuni Information Model di base e inoltre fornisce ai propri utenti gli strumenti necessari alla creazione di nuovi Information Model, con lo scopo di integrare informazioni relative ai loro dispositivi all'interno dell'Address Space. Uno di questi strumenti è il software "UAModeler", rilasciato dalla Unified Automation GmbH, una delle aziende membro dell'OPC Foundation, utilizzato nello sviluppo dei casi applicativi descritti nei Capitoli 4 e 5 e presentato di seguito nel paragrafo 4.1.1.

2.5.Services (OPC-UA Part 4)

I Services (servizi) sono il metodo usato da un Client per comunicare e accedere ai dati dell'Address Space di un Server. Un servizio stabilisce quindi la struttura dell'interfaccia di comunicazione tra due applicazioni OPC-UA. Il funzionamento di questi servizi prevede un modello di comunicazione derivante da quello dei Web Services, in cui il servizio si compone di messaggi di richiesta e risposta (Request e Response). Per invocare un servizio un Client deve quindi inviare una richiesta al Server, che la processa ed invia una risposta al Client. Questo scambio di messaggi avviene in modo asincrono, pertanto tutte le chiamate di servizi sono asincrone per definizione. Ciò implica che una volta inviata una richiesta un Client può tranquillamente continuare la sua esecuzione in attesa della risposta del Server. I principali parametri che vengono scambiati nei messaggi di richiesta e di risposta sono riportati rispettivamente in Tabella 6 e in Tabella 7.

Parametro	Descrizione
AuthenticationToken	Identificatore segreto usato per assegnare la chiamata del servizio alla sessione precedentemente stabilita.
Timestamp	Istante di tempo in cui il client ha inviato la richiesta.
RequestHandle	Identificativo assegnato alla chiamata del servizio, definito dal client.
ReturnDiagnostic	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice *status code.*
AuditEntryId	Identifica il client o l'user che ha iniziato l'azione; usato in caso di auditing.
TimeoutHint	Time-out settato per la chiamata del servizio lato client nell' UA Stack; quando termina, il server annulla la chiamata ricevuta.

Tabella 6: parametri caratterizzanti un messaggio di richiesta

Parametro	Descrizione
Timestamp	Istante di tempo in cui il server ha inviato la risposta.
RequestHandle	Identificativo assegnato alla chiamata di servizio definito dal client.
ServiceResult	Classe definita dallo standard per descrivere il risultato della chiamata del servizio. Contiene un uint a 32 bit in cui i 16 più significativi rappresentano il valore numerico dell'errore ed i rimanenti contengono informazioni aggiuntive.
ServiceDiagnostic	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice status code.

Tabella 7: parametri caratterizzanti un messaggio di risposta

OPC-UA può essere applicato in contesti molto diversi tra loro; lo standard mette quindi a disposizione dei Client la possibilità di stabilire time-out individuali e personalizzabili per ogni chiamata di servizio, in modo da potersi adattare alle diverse tempistiche fisiche richieste dalla comunicazione. La sezione 4 delle specifiche di OPC-UA fornisce una

definizione dei servizi suddividendoli sulla base delle loro funzionalità in differenti categorie, note come ServiceSet.

- Discovery ServiceSet: fanno parte di questa categoria i servizi FindServers, GetEndpoints e RegisterServer. È un insieme di servizi che permette ad un OPC-UA Client di individuare un OPC-UA Server e leggere le informazioni di configurazione del canale sicuro (SecureChannel), reperibili presso gli endopints messi a disposizione da ogni Server OPC-UA.
- SecureChannel ServiceSet: è un insieme di servizi che permetto ad un OPC-UA Client di stabilire un canale di comunicazione che assicuri confidenzialità e integrità dei messaggi scambiati con l'OPC-UA Server. Fanno parte di questa categoria i servizi OpenSecureChannel e CloseSecureChannel.
- Session ServiceSet: in questa categoria sono racchiusi i servizi CreateSession, ActivateSession, CloseSession e Cancel. Questo insieme di servizi permettono ad un utente lato Client di creare ed operare in una sessione a seguito della sua autenticazione.
- Node Management ServiceSet: sono servizi che consentono all'OPC-UA Client di manipolare nodi e riferimenti che compongono l'Address Space. Questa categoria racchiude i servizi AddNode, DeleteNode, AddReference e DeleteReference.
- View ServiceSet: i servizi che fanno parte di questa categoria sono Browse, BrowseNext, RegisterNode, UnregisterNode e TranslateNodePathsToNodeIds, che consentono ad un OPC-UA Client di esplorare l'Address Space o le View.
- Query ServiceSet: sono servizi che permettono ad un OPC-UA Client di generare dei filtri con lo scopo di restringere la lista di nodi e riferimenti restituita dall'OPC-UA Server. I servizi di questa categoria sono QueryFirst e QueryNext.
- Attribute ServiceSet: in questa categoria si trovano i servizi Read, HistoryRead, Write e HistoryUpdate, che permettono ad un OPC-UA Client di leggere e scrivere gli attributi di un nodo, inclusi i valori dello storico.
- Method ServiceSet: questa categoria è formata da un solo servizio, chiamato Call, il quale consente le chiamate dei metodi esposti dall'OPC-UA Server.
- MonitoredItem ServiceSet: è un insieme di servizi che consente ad un OPC-UA Client di creare, modificare ed eliminare MonitoredItem, i quali vengono impiegati per controllare il verificarsi di eventi negli oggetti e i cambi di valore degli attributi dei nodi. Fanno parte di questa categoria i servizi

CreateMonitoredItems, ModifyMonitoredItems, SetMonitoringMode, SetTriggering e DeleteMonitoredItems.

- Subscription ServiceSet: sono servizi che permettono ad un OPC-UA Client di creare, modificare ed eliminare Subscriptions, che hanno il compito di inviare le notifiche generate dai MonitoredItem. Costituiscono questa categoria i servizi CreatSubscription, ModifySubscription, SetPublishingMode, Publish, Republish, TransferSubscription e DeleteScubscription.

2.6.Security (OPC-UA Part 2)

Lo standard OPC-UA può essere impiegato a livello industriale su vari livelli, partendo dai gestionali aziendali di alto livello fino ad arrivare al controllo di processo di basso livello direttamente sul dispositivo. L'utilizzo di OPC-UA porta quindi necessariamente ad uno scambio di dati e di informazioni tra i fornitori ed i clienti che rappresentano un punto di accesso per lo spionaggio aziendale e possono quindi essere esposti a minacce esterne. OPC-UA inoltre supporta diversi protocolli e tecnologie di comunicazione che richiedono differenti livelli e infrastrutture di sicurezza. Lo scenario appena descritto permette di comprendere come OPC-UA può essere utilizzato a vari livelli e con differenti obiettivi all'interno dello stesso sistema; di conseguenza i requisiti di sicurezza variano. Per tali motivazioni lo standard definisce un modello di sicurezza flessibile che si possa adattare ad ogni ambiente di esecuzione. L'architettura sulla quale si basa il modello di sicurezza è suddivisa in livelli ed è presentata in Figura 13.

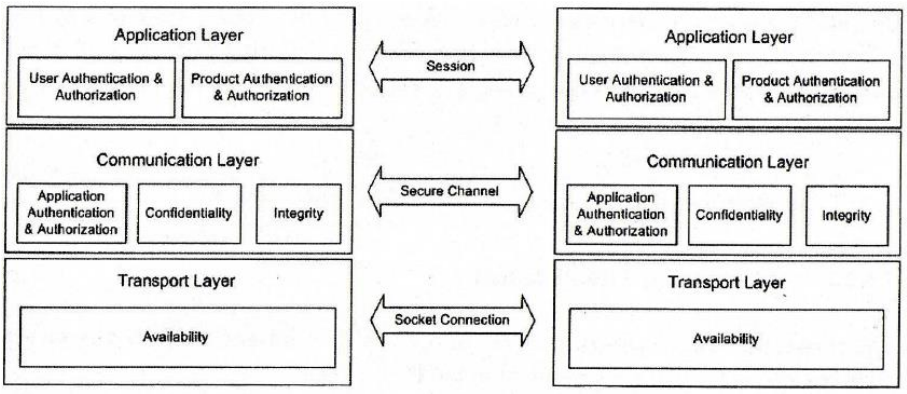


Figura 13: architettura del modello di sicurezza adottato dallo standard OPC-UA

Il Transport Layer gestisce la trasmissione, la ricezione ed il trasporto dei dati, stabilendo una connessione socket (Indirizzo IP: porta) e fornendo meccanismi per la gestione degli errori e perdita dei dati. Il Communication Layer permette di stabilire un canale sicuro (Secure Channel) che ha il compito di assicurare l'integrità dei dati mediante lo scambio di firme

digitali, la confidenzialità dei dati attraverso la cifratura delle informazioni sensibili, l'autenticazione e l'autorizzazione delle applicazioni mediante l'utilizzo di speciali certificati X.509. Il vantaggio principale nell'adozione della tecnologia X.509 per la creazione dei certificati è rappresentato dal supporto all'aggiunta di campi personalizzati all'interno del certificato. L'Application Layer è il livello che si occupa di creare una sessione (Session) tra Client e Server, con lo scopo di gestire l'autenticazione e l'autorizzazione del Client.

2.6.1. Creazione del canale

Per stabilire un canale sicuro (Secure Channel) il Client segue una serie di passaggi visibili in Figura 14, utilizzando i servizi del SecureChannel ServiceSet riportati nel paragrafo 2.5.

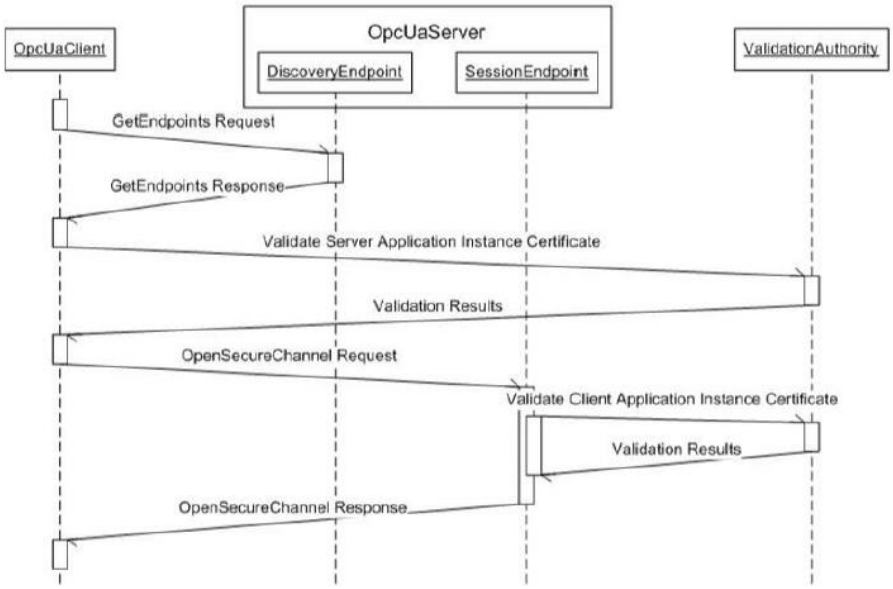


Figura 14: scambio di messaggio per la creazione del canale tra Client e Server

In primo luogo il Client inoltra una GetEndpoints Request al DiscoveryEndpoint per chiedere informazioni relative agli Endpoint disponibili per il Server al quale desidera collegarsi. Il DiscoveryEndpoint risponderà direttamente al Client con una GetEndpoints Response, inoltrando al Client i descrittori degli Endpoints contenenti le configurazioni di sicurezza. Ricevute le informazioni desiderate, il Client seleziona un Endpoint avente una configurazione di sicurezza supportata e ne controlla la validità dell'Application Instance Certificate presso la ValidationAuthority associata. Se il certificato è considerato valido, il Client procede alla creazione del canale sicuro, inoltrando una OpenSecureChannel Request al SessionEndpoint del Server. In questa fase Client e Server si accordano sulla Security Mode (modalità di sicurezza) da adottare, scegliendo tra:

- Sign, i messaggi inviati saranno firmati con la chiave privata associata al certificato del Client.
- SignAndEncrypt: i messaggi inviati saranno firmati con la chiave privata associata al certificato del Client e cifrati con la chiave pubblica associata al certificato del Server.
- None, i messaggi inviati non saranno ne firmati ne cifrati e pertanto non sicuri.

Infine il SessionEndpoint valida il certificato del Client con un'apposita richiesta alla relativa ValidationAuthority e, se il certificato è considerato valido, inoltrerà al Client una OpenSecureChannel Response per confermare l'effettiva creazione del canale sicuro. Va sottolineato che il canale sicuro ha una validità temporale limitata e al suo scadere va rinnovato eseguendo nuovamente la OpenSecureChannel Response e Request.

2.6.2. Creazione della sessione

Per instaurare una sessione (Session) tra Client e Server, il Client segue una serie di passaggi, visibili in Figura 15 , sfruttando i servizi del Session ServiceSet riportati nel paragrafo 2.5.

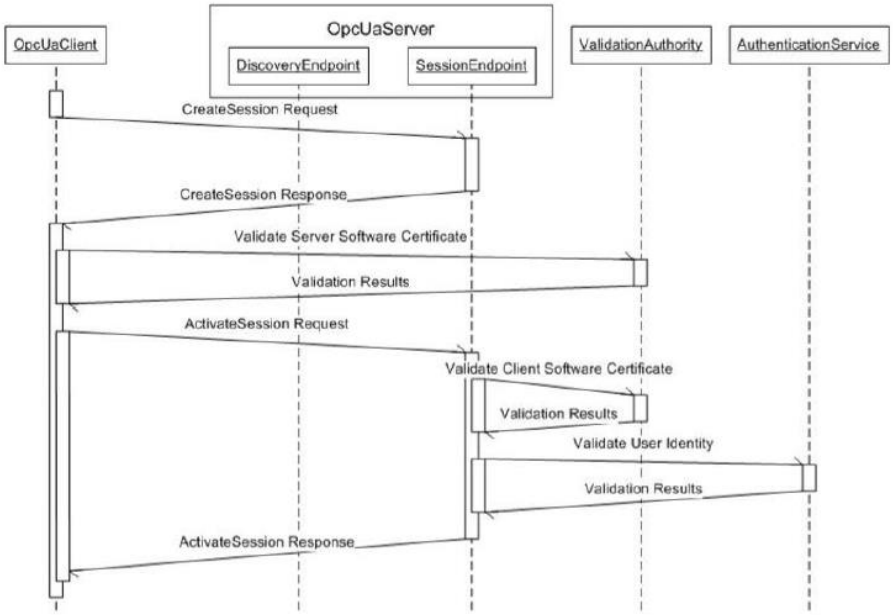


Figura 15 scambio di messaggio per la creazione della sessione tra Client e Server

In prima istanza il Client invia al Server una CreateSession Request per richiedere la creazione della sessione, a cui il Server risponde con una CreateSession Response comunicando al Client il proprio certificato (Server Software Certificate). Il Client valida quindi certificato del Server presso la ValidationAuthority preposta; se il certificato è valido la

creazione della sessione può essere considerata conclusa. La creazione della sessione è immediatamente seguita dalla sua attivazione: il Client inoltra al Server il suo certificato (Client Software Certificate) e le credenziali dell'utente corrente tramite un ActivateSession Request. Il Server si occupa quindi di validare il certificato del Client presso la ValidationAuthority, di verificare le credenziali dell'utente e di rispondere con una ActivateSession Response. Seguendo questi passaggi la sessione può considerarsi attiva e il Client può iniziare ad accedere ai dati esposti dal Server. Attivare la sessione separatamente rispetto alla sue creazione comporta vari vantaggi, tra cui:

- Garanzia che le credenziali dell'utente siano inviate al Server usato per la creazione del canale, in quanto la creazione della sessione include la verifica dei suoi certificati.
- Supporto log-over, ossia la possibilità di cedere l'uso della sessione ad un altro utente senza dover distruggere e ricreare la connessione.

2.7.OPC-UA nell'industria 4.0

Il termine Industria 4.0, o quarta rivoluzione industriale, definisce un nuovo stadio di sviluppo dell'odierna industria manifatturiera, volto ad inserire alcune nuove tecnologie produttive per migliorare le condizioni di lavoro, creare nuovi modelli di business, aumentare la produttività degli impianti e migliorare la qualità dei prodotti. L'industria 4.0 passa per il concetto di *Smart Factory*, che si compone di tre parti:

- *Smart production*: nuove tecnologie produttive che creano collaborazione tra tutti gli elementi presenti nella produzione ovvero collaborazione tra operatore, macchine e strumenti.
- *Smart service*: tutte le "infrastrutture informatiche" e tecniche che permettono di integrare i sistemi; ma anche tutte le strutture che permettono, in modo collaborativo, di integrare le aziende tra loro.
- *Smart energy*: tutto questo sempre con un occhio attento ai consumi energetici, creando sistemi più performanti e riducendo gli sprechi di energia secondo i paradigmi tipici dell'energia sostenibile.

La chiave di volta dell'industria 4.0 sono i sistemi cyberfisici (CPS – Cyber Physical System) ovvero sistemi fisici che sono strettamente connessi con i sistemi informatici e che possono interagire, collaborare e controllare altri sistemi. Da uno studio di Boston Consulting

emerge che la quarta rivoluzione industriale si basa sull'adozione di alcune tecnologie definite abilitanti, quali le seguenti:

- *Advanced Manufacturing Solution*: in queste tecnologie rientrano i sistemi di movimentazione dei materiali automatici e la robotica avanzata, che oggi entra sul mercato con i robot collaborativi o Cobot.
- *Additive manufacturing*: sistemi di produzione additiva che aumentano l'efficienza dell'uso dei materiali.
- *Realtà aumentata*: sistemi di visione con realtà aumentata per guidare meglio gli operatori nello svolgimento dei processi produttivi.
- *Simulazioni*: simulazione tra macchine interconnesse per ottimizzare i processi.
- *Integrazione orizzontale e verticale*: integrazione e scambio di informazioni in orizzontale e in verticale, tra tutti gli attori del processo produttivo.
- *Industrial internet*: comunicazione multidirezionale tra elementi della produzione, non solo all'interno dell'azienda, ma anche all'esterno, grazie all'utilizzo di internet.
- *Cloud*: implementazione di tutte le tecnologie cloud, come l'archiviazione online delle informazioni, l'uso del cloud computing, di servizi esterni di analisi dati, ecc.
- *Cybersecurity*: l'aumento delle interconnessioni interne ed esterne aprono la porta a tutta la tematica della sicurezza delle informazioni e dei sistemi che non devono essere alterati dall'esterno.
- *Big Data Analytics*: tecniche di elaborazione, gestione ed analisi di grandissime quantità di dati attraverso sistemi aperti che forniscono previsioni e decisioni al fine di ottimizzare la qualità della produzione ed il risparmio energetico.

Alcuni dei vantaggi più evidenti nell'adozione di tecnologie digitali tipiche della quarta rivoluzione industriale possono quindi essere sintetizzati nei seguenti punti:

- Maggiore flessibilità attraverso la produzione di piccoli lotti ai costi della grande scala.
- Maggiore velocità dalla fase di prototipizzazione alla produzione in serie.
- Maggiore produttività attraverso una riduzione di costi e sprechi, un aumento dell'affidabilità dei sistemi produttivi e della qualità resa.
- Maggiore sicurezza in termini di errori, infortuni e dell'ergonomia sul luogo di lavoro attraverso una migliore interazione uomo-macchina.

- Maggiore sostenibilità attraverso una riduzione dei consumi energetici, delle emissioni e dell'uso di materie prime.

Queste nuove tecnologie sono già ampiamente disponibili ed utilizzate nelle aziende, ma attualmente la loro applicazione è ancora limitata e sporadica, essendo principalmente focalizzata sul controllo di processo industriale destinato alla produzione in serie di componenti, integrata con la robotica solo in determinati contesti e in applicazioni che si limitano a ripetere le stesse azioni. Di fatto, l'innovazione 4.0 non consiste nell'introdurre macchine all'avanguardia dal punto di vista tecnologico, ma nel saper combinare diverse tecnologie in modo da integrare il sistema di fabbrica e le linee di produzione allo scopo di realizzare un sistema integrato e connesso, in cui macchine, persone e sistemi informativi collaborano per creare prodotti, servizi e ambienti di lavoro più intelligenti. È chiaro quindi che sono necessari sforzi considerevoli per attuare con successo la visione di Industria 4.0, poiché deve essere soddisfatta un'ampia gamma di requisiti per far sì che l'intera infrastruttura possa funzionare. Per gestire la complessità intrinseca di questa impresa sono dunque richiesti una modularizzazione completa ed una standardizzazione ad ampio raggio. In quest'ottica, all'interno del progetto "Plattform Industrie 4.0", formato dalle società tedesche "Bitkom e.V.", "VDMA e.V." e "ZVEI e.V.", sono stati raggiunti importanti traguardi nella standardizzazione di Industria 4.0 con lo sviluppo di una prima versione di un'architettura di riferimento, nota come RAMI 4.0 (Reference Architecture Model Industrie 4.0) e visibile in Figura 16.

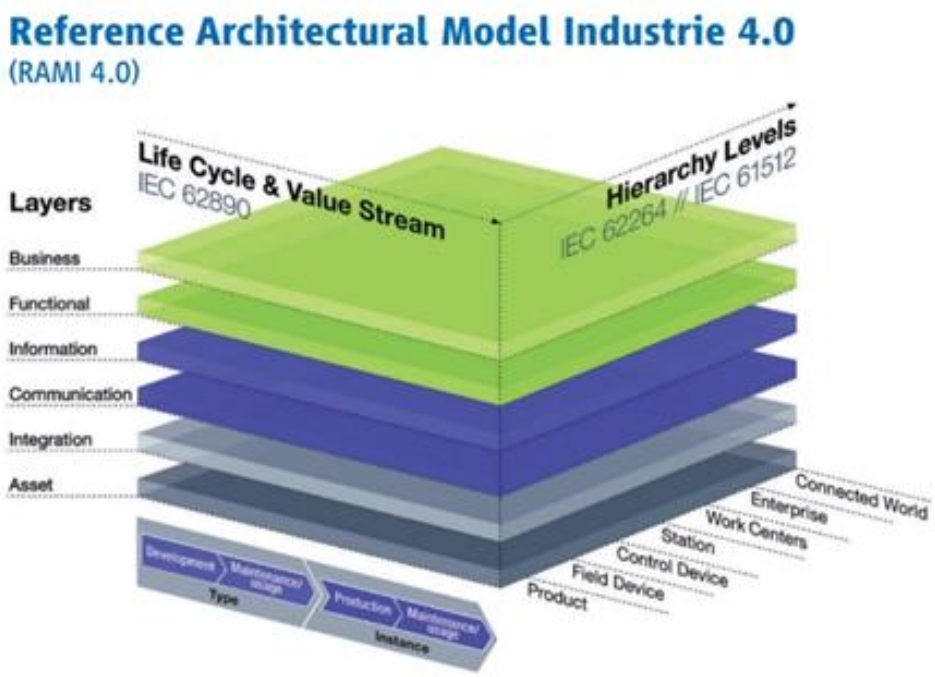


Figura 16: architettura RAMI

Come parte dell'architettura complessiva di RAMI 4.0, OPC-UA è raccomandato come unico standard per la realizzazione del livello di comunicazione e questo ha chiarito che non ci sarà Industria 4.0 senza OPC-UA. Lo standard OPC-UA, oggetto di questo elaborato, è un protocollo di comunicazione standard che rende facile ed accessibile a tutti la raccolta dati, fornendo un linguaggio comune indipendente dal sistema specifico. Nella Tabella 8 sono riportate alcune delle caratteristiche di OPC-UA che soddisfano i requisiti richiesti dalla quarta rivoluzione industriale.

Requisiti industria 4.0	OPC-UA
Indipendenza dal produttore, dal settore, dal sistema operativo e dal linguaggio di programmazione.	OPC-UA è principalmente utilizzato in ambito di automazione, ma è indipendente dal settore. OPC-UA funziona su tutti i sistemi operativi e può essere implementato in tutti i linguaggi: attualmente sono disponibili stack di comunicazione in Ansi C/C++, .NET e Java.
Scalabilità per il networking integrato, inclusi i più piccoli sensori, dispositivi embedded, controllori PLC/CNN, PC, smartphone e applicazioni cloud. Comunicazione orizzontale e verticale su tutti i livelli.	OPC-UA è scalabile su hardware single-core e multi-core e su un'ampia gamma di architetture CPU (Intel, ARM, ecc.). OPC-UA può essere utilizzato in dispositivi di campo embedded come lettori RFID, convertitori di protocollo e praticamente in tutti i controller, prodotti SCADA/HMI e sistemi MES/ERP. I progetti sono già stati realizzati con successo in vari ambienti cloud, tra cui: Amazon, Google e Microsoft Azure Cloud.
Trasferimento e autenticazione sicuri a livello di utente e applicazione.	OPC-UA fornisce meccanismi per l'autenticazione e l'autorizzazione delle applicazioni e degli utenti. Include inoltre meccanismi di trasferimento firmati e crittografati per l'integrità e la riservatezza dei dati.
Trasporto tramite standard consolidati per lo scambio di dati, comandi ed eventi live e storici.	OPC-UA è stato concepito sulla base di un'architettura SOA. Sono disponibili diversi protocolli per diversi casi d'uso (ad esempio applicazioni ad alte prestazioni, accesso al browser Web, ecc.). Inoltre, OPC-UA offre un modello di comunicazione Publish/Subscribe (PubSub). Gli stack di comunicazione garantiscono il trasporto coerente di tutti i dati OPC-UA. Oltre ai dati in tempo reale, anche i dati storici e i loro aggregati matematici sono standardizzati in OPC-UA.
Contenuti informativi con qualsiasi grado di complessità per la rappresentazione di prodotti reali e delle loro fasi di produzione.	OPC-UA fornisce uno spazio degli indirizzi completamente collegato in rete e orientato agli oggetti, che include metadati e descrizioni degli oggetti. Le strutture di oggetti possono essere generate tramite il riferimento tra le istanze di oggetti e le relative definizioni di sottotipi, che possono essere estese tramite ereditarietà. Poiché i server OPC-UA gestiscono sia le istanze oggetto che i tipi di oggetto associati, i client OPC-UA possono navigare nello spazio degli indirizzi per ottenere tutte le informazioni sull'istanza e sul tipo di cui hanno bisogno.
Partecipazione auto-organizzata dei componenti "intelligenti" alla rete.	OPC-UA definisce diversi meccanismi di identificazione e notifica dei dispositivi compatibili con OPC-UA e delle loro funzioni all'interno di una rete. I partecipanti possono essere

	collocati sullo stesso host, in una subnet o distribuiti a livello globale. L'aggregazione tra subnet e procedure intelligenti viene utilizzata per identificare e indirizzare i partecipanti alla rete.
Integrazione ingegneristica ed estensione semantica.	La OPC Foundation ha collaborato con successo con altre organizzazioni (PLCopen, MDIS, FDI, AIM, VDMA, MTConnect, AutomationML, ecc.) e continua ad espandere le sue attività di collaborazione con gruppi di una gamma sempre più ampia di settori.
Verifica della conformità alla norma definita.	OPC-UA è uno standard IEC (IEC 62541) per il quale sono disponibili strumenti e laboratori di prova per testarne e certificarne la conformità. Sono necessari test estesi per apportare estensioni e/o modifiche. Inoltre, integrazioni della sicurezza dei dati e della sicurezza funzionale vengono eseguite e certificate da organismi di test esterni.

Tabella 8: requisiti di Industria 4.0 soddisfatti dallo standard OPC-UA

Indipendentemente dal sistema specifico o dal dispositivo che si può avere nel proprio stabilimento, lo standard OPC-UA fornisce un linguaggio comune che semplifica la raccolta e lo scambio di dati e informazioni: un traguardo importante e una risorsa fondamentale per lo sviluppo del concetto di Industria 4.0.

2.8.OPC-UA nella lavorazione di materiale plastico

La OPC Foundation lavora a stretto contatto con organizzazioni e associazioni di vari settori, al fine di definire nuove interfacce standard in grado di esporre le informazioni specifiche scelte da una determinata organizzazione, attraverso le funzionalità offerte dallo standard OPC-UA. Una di queste è EUROMAP, l'organizzazione ombrello dell'industria europea dei macchinari per la lavorazione della plastica e della gomma, che rappresenta circa 1000 aziende del settore. EUROMAP fornisce raccomandazioni tecniche e standard di utilizzo delle più moderne tecnologie del settore e informazioni di mercato sui migliori produttori di attrezzature innovative e affidabili. Dalla collaborazione fra OPC Foundation ed EUROMAP, nata nel 2019, è nato quindi il progetto congiunto "OPC-UA Plastics and Rubber Machinery", il quale si compone ad oggi dei seguenti documenti tecnici:

- EUROMAP 83 (OPC 40083): OPC-UA interfaces for plastics and rubber machinery – General Types definitions
- EUROMAP 77 (OPC 40077): OPC-UA interfaces for plastics and rubber machinery – Data exchange between injection moulding machines and MES
- EUROMAP 82 (OPC 40082): OPC-UA interfaces for plastics and rubber machinery – Peripheral Devices

- EUROMAP 84 (OPC 40084): OPC-UA interfaces for plastics and rubber machinery – Extrusion
- EUROMAP 79 (OPC 40079): OPC-UA interfaces for plastics and rubber machinery – Data exchange between injection moulding machines and robots

Di seguito sono descritti i contenuti principali del documento EUROMAP 79 necessari allo sviluppo del simulatore di una macchina per lo stampaggio ad iniezione (pressa), riportato nel capitolo 5. Si segnala che tale documento al momento non è ancora stato rilasciato nella sua versione definitiva, pertanto si fa riferimento alla sua versione più recente: "Release Candidate 1.00.00" pubblicata in data 8 novembre 2021; inoltre, tale documento coincide con le specifiche "OPC 40079" rilasciate dalla OPC Foundation. L'obiettivo delle specifiche definite all'interno del documento EUROMAP 79 (OPC 40079) è quello di fornire un'interfaccia unica per lo scambio di dati ed informazioni tra macchine IMM (Injection Moulding Machine) e loro robot di asservimento, di produttori differenti, al fine di garantirne la compatibilità. In particolare, all'interno del documento sono definite la struttura dati esposta dalla macchina IMM per il robot e la struttura dati del robot esposta per la macchina IMM. In questo elaborato è descritta solamente la prima (IMMToRobotType), nel paragrafo seguente, in quanto nello sviluppo del simulatore è l'unica utilizzata, dovendo realizzare un'architettura di comunicazione in cui la macchina IMM fornisce il Server OPC-UA ed il robot è il Client OPC-UA. In Figura 17 sono visibili tutte le possibili configurazioni per lo scambio di dati tra macchina IMM e suo robot di asservimento:

- la configurazione di tipo Client-Server permette alle due macchine di scambiarsi dati relativi al loro stato generale, alla progressione del loro ciclo di lavoro e alla qualità del pezzo lavorato.
- la configurazione di tipo Publisher-Subscriber permette lo scambio di segnali in real-time al fine di evitare collisioni tra le due macchine.

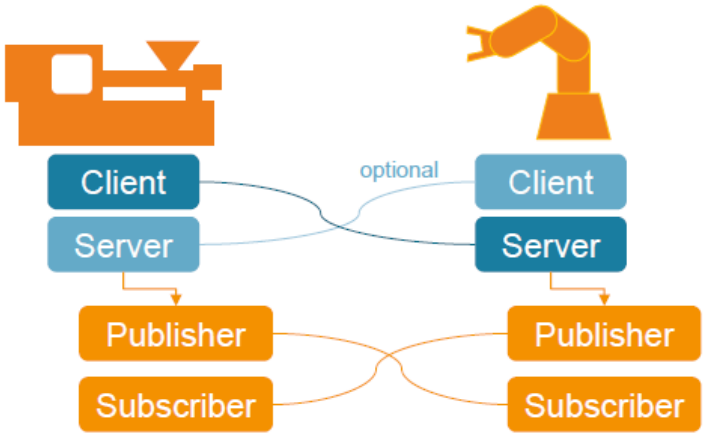


Figura 17: possibili configurazione per lo scambio di messaggi tra macchina IMM e robot di asservimento

2.8.1. IMMToRobotType

La Figura 18 mostra come è organizzato l'Address Space nel caso in cui sia la macchina IMM ad esporre i propri dati fornendo un OPC-UA Server.

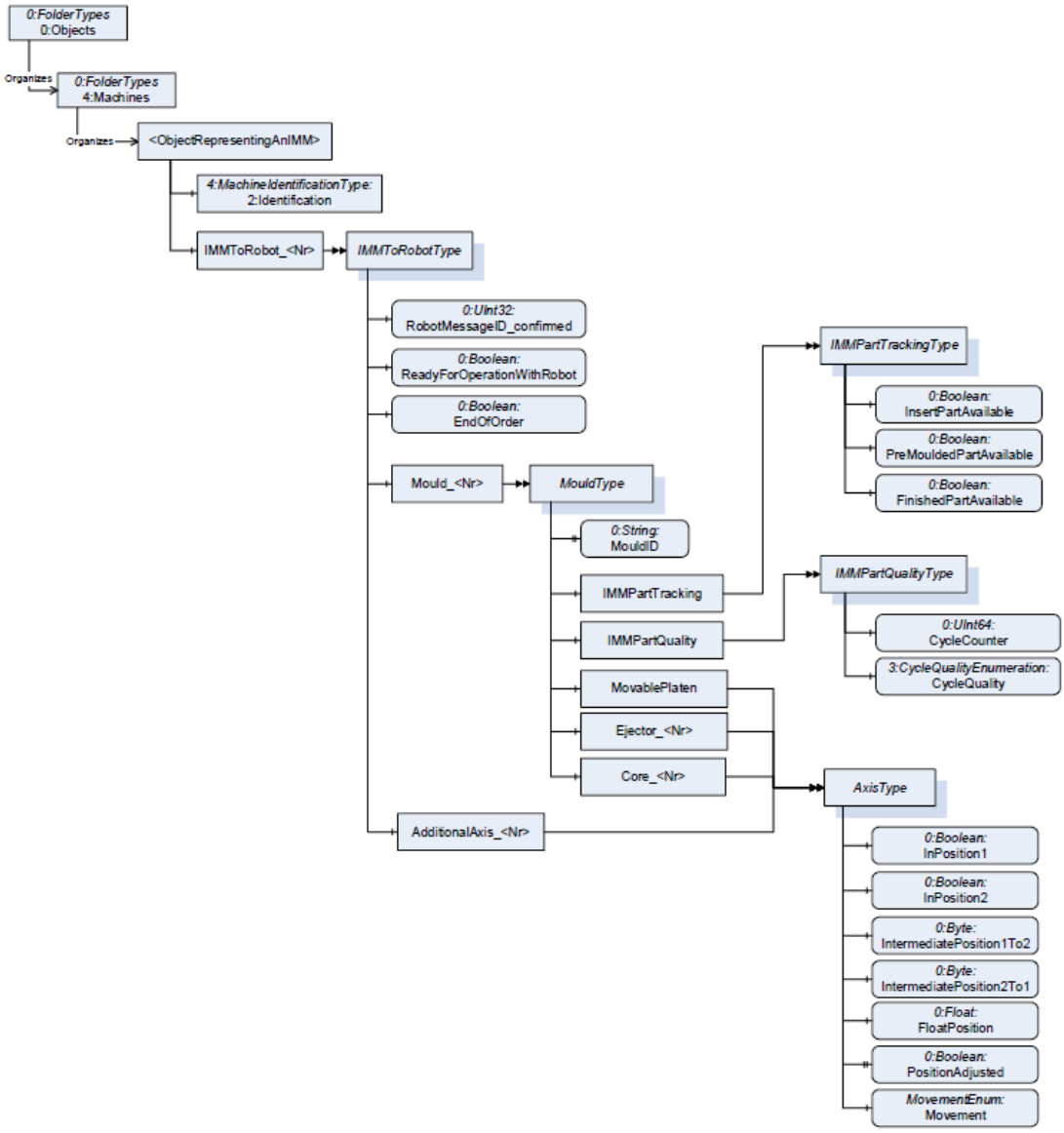


Figura 18: strutta del tipo di oggetto IMMToRobotType

Per rappresentare l'interfaccia della macchina IMM si utilizza il tipo di oggetto IMMToRobotType, caratterizzato dalle seguenti componenti:

- RobotMessageID_confirmed: variabile di tipo "Uint32" che la macchina IMM setterà per confermare la ricezione del comando proveniente dal robot e di aver operato sui suoi assi come indicato dal robot stesso.
- ReadyForOperationWithRobot: variabile di tipo "Boolean" che indica se la macchina IMM è abilitata a lavorare in accordo con il robot.

- EndofOrder: variabile di tipo “Boolean” che indica al robot il completamento di un comando.
- Mould_<Nr>: oggetto generico di tipo MouldType.
- AdditionalAxis_<Nr>: oggetto generico di tipo AxisType.

Come indicato all’interno delle specifiche, per applicazioni base è sufficiente istanziare una solo stampo (Mould_1) e un solo asse addizionale (AddiotionalAxis_1). Il tipo di oggetto MouldType è utilizzato per rappresentare informazioni relative allo stampo, come il suo stato, la posizione dei suoi assi e informazioni relative allo stato di avanzamento della lavorazione del manufatto e alla sue caratteristiche qualitative. La sua struttura e le sue componenti, definite all’interno delle specifiche OPC 40079, sono le seguenti:

- MouldID: variabile di tipo “String” utilizzata per identificare lo stampo.
- MovablePlaten: oggetto di tipo AxisType che rappresenta la piastra mobile dello stampo.
- Ejector_<Nr>: oggetto generico di tipo AxisType che rappresenta gli estrattori dello stampo.
- Core_<Nr>: oggetto generico di tipo AxisType che rappresenta i martinetti dello stampo.
- IMMPartTracking: oggetto di tipo IMMPartTrackingType.
- IMMPartQuality: oggetto di tipo IMMPartQualityType.

Sempre facendo riferimento ad applicazioni base, le specifiche indicano di istanziare due estrattori (Ejector_1, Ejector_2) e 10 martinetti (Core_1, …, Core_10). Il tipo di oggetto AxisType è utilizzato per descrivere i movimenti e la posizione degli assi dello stampo ed è caratterizzato dalle componenti : InPosition1, InPosition2, IntermediatePosition1To2, IntermediatePosition2To1, FloatPosition, PositionAdjusted e Movement.

InPosition1 e InPosition2 sono entrambe variabili di tipo “Boolean” e indicano quando l’asse raggiunge, in entrambe le direzioni del movimento, una delle due posizioni limite predefinite. Queste posizioni limite variano in base al produttore/programmatore della macchina, tuttavia all’interno delle specifiche sono indicate le direttive visibili in Tabella 9.

Axis	InPosition1	InPosition2
Movable Platen	Pressa chiusa	Pressa aperta
Ejector	Estrattore indietro	Estrattore avanti
Core	Martinetto indietro	Martinetto avanti
Additional Axis	Non utilizzato (sempre False)	In posizione di lavoro

Tabella 9: definizione delle posizioni finali per gli assi di una macchina IMM

IntermediatePosition1To2 e IntermediatePosition2To1 sono variabili di tipo “Byte” che possono essere utilizzate per definire delle soglie intermedie ($1\ Byte = 8\ Bit = 256\ soglie$) per ogni direzione del movimento dell’asse. Come mostrato in Figura 19, la variabile IntermediatePosition1To2 è utilizzata per definire soglie intermedie nella direzione di movimento positiva dell’asse, cioè per movimenti che partono con “InPosition1 = TRUE” e che terminano con “InPosition2 = TRUE”. Viceversa, la variabile IntermediatePosition2To1 è impiegata per definire soglie intermedie nella direzione di movimento negativa dell’asse, ovvero per le transizioni da “InPosition2 = TRUE” a “InPosition1 = TRUE”.

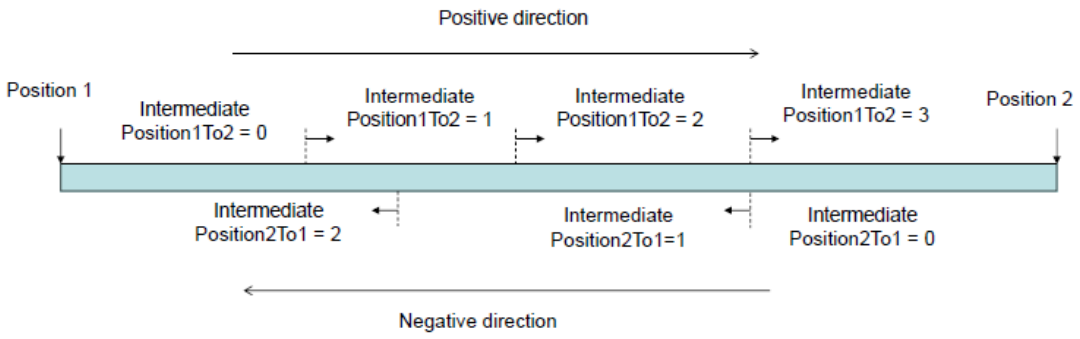


Figura 19: definizione delle variabili “IntermediatePosition”

FloatPosition è una variabile di tipo “Float” utilizzata per indicare la posizione esatta dell’asse; è espressa in [mm] per gli assi lineari (piastra mobile, estrattori e martinetti) e in [°] per gli assi rotazionali (assi addizionali). PositionAdjusted è una variabile di tipo “Boolean” che indica al robot se è stata raggiunta la posizione da lui richiesta. Movement è una variabile di tipo “MovementEnum”, visibile in Tabella 10, che viene utilizzata per specificare se l’asse è in movimento, e in tal caso in che direzione si sta muovendo, oppure se è fermo.

Nome	Valore	Descrizione
NOT_MOVING	0	L’asse è fermo.
MOVING_IN_POSITIVE_DIRECTION	1	L’asse si sta muovendo dalla posizione 1 alla posizione 2 (direzione positiva).
MOVING_IN_NEGATIVE_DIRECTION	2	L’asse si sta muovendo dalla posizione 2 alla posizione 1 (direzione negativa).

Tabella 10: enumerazione del tipo di variabile “MovementEnum”

Al fine di raggiungere uniformità nella rappresentazione, all’interno delle specifiche sono riportate precise indicazioni su quali di queste variabili istanziare in base al tipo di asse:

- Per gli oggetti che rappresentano la piastra mobile, gli estrattori e l’asse addizionale dello stampo devono essere istanziate tutte le variabili.
- Per gli oggetti che identificano i martinetti dello stampo vanno utilizzate tutte le variabili tranne FloatPosition e PositionAdjusted.

Infine, i tipi di oggetto IMMPartTrackingType e IMMPartQualityType restituiscono rispettivamente informazioni relative allo stato di avanzamento della lavorazione e informazioni relative alla qualità della parte prodotta. Pertanto, il tipo di oggetto IMMPartTrackingType è caratterizzato dai seguenti componenti:

- InsertPartAvailable: è una variabile di tipo "Boolean" che segnala al robot la presenza di un pezzo pronto per la lavorazione.
- PreMouldedPartAvailable: è una variabile di tipo "Boolean" che segnala al robot la presenza di un pezzo in fase di lavorazione.
- FinishedPartAvailable: : è una variabile di tipo "Boolean" che segnala al robot la presenza di un pezzo che ha subito l'intero processo di lavorazione e pronto per essere estratto.

Mentre, il tipo di oggetto IMMPartQualityType ha due componenti:

- CycleCounter: è una variabile di tipo "UInt64" che fornisce il numero di cicli completati per la produzione attuale.
- CycleQuality: è una variabile di tipo "CycleQualityEnumeration", visibile in Tabella 11, che indica informazioni relative alla qualità di esecuzione del ciclo di lavorazione della macchina IMM.

Nome	Valore	Descrizione
GOOD_CYCLE	0	La macchina non ha riscontrato problematiche durante il ciclo e il pezzo lavorato è considerato buono.
BAD_CYCLE	1	Il pezzo lavorato è considerato difettoso.
TEST_SAMPLE_CYCLE	2	Il ciclo della macchina viene etichettato come ciclo di test.
FAILED_CYCLE	3	La macchina ha riscontrato problematiche durante il ciclo, pertanto il pezzo lavorato va segnalato come difettoso.

Tabella 11: enumerazione del tipo di variabile "CycleQualityEnumeration"

La definizione del tipo di variabile "CycleQualityEnumeration" è descritta nel documento delle specifiche EUROMAP 83 (OPC 40083).

CAPITOLO 3

CASO APPLICATIVO: MACCHINA AVVOLGITRICE

In questo capitolo dell'elaborato sono presentati gli aspetti progettuali e implementativi riguardanti la realizzazione di un Server OPC-UA, tramite l'uso del software "Premium HMI". Il contesto all'interno del quale si pone l'applicativo è quello dello sviluppo e ottimizzazione delle macchine "custom", in particolare delle macchine avvolgitrici, in ottica Industria 4.0. Il Server ha il compito di rendere disponibili dati relativi allo stato macchina, al lotto di produzione attuale e precedente reperibili dal controllore della macchina avvolgitrice e di garantire la possibilità di comunicare da remoto con il gestionale del cliente per inviare informazioni riguardanti nuovi lotti di produzione. Inoltre, è riportato lo sviluppo delle applicazioni realizzate tramite i software RDE (Robox Development Environment) e RDT (Robox Development Tool), rispettivamente descritti nel paragrafo 3.3.1 e nel paragrafo 3.3.2, utilizzate per integrare la comunicazione OPC-UA con il controllore e il terminale della macchina avvolgitrice. Infine, sono riportati i risultati ottenuti dai test effettuati per dimostrare la corretta configurazione dello scambio di dati tra i registri del controllore con le due interfacce grafiche realizzate e della comunicazione OPC-UA, collegandosi al Server tramite il software UAExpert.

3.1. Strumenti utilizzati

La realizzazione di questa prima applicazione ha richiesto l'utilizzo di diversi pacchetti software e di hardware specifici, resi disponibili dalla Campetella Robotic Center e descritti brevemente nei paragrafi successivi. Alcuni di questi strumenti sono poi utilizzati anche per lo sviluppo e il test delle altre applicazioni descritte all'interno di questo elaborato nei Capitoli 4 e 5. Di seguito sono quindi introdotte le principali funzionalità di interesse ai fini del contesto OPC-UA e Industria 4.0 dei software e degli hardware utilizzati.

- Software: Premium HMI, RDE (Robox Development Environment), RDT (Robox Development Tool) e UAExpert.
- Hardware: Robox RP2, RHMI (Robox HMI), e ASEM RK10.

3.1.1. Premium HMI

Premium HMI è una piattaforma software, rilasciata da ASEM, che permette di realizzare interfacce grafiche basate su hardware ARM e x86 e compatibili con i sistemi operativi Windows e WindowsCE. Premium HMI rappresenta una delle soluzioni per la realizzazione di applicazioni compatibili con gli standard ed i requisiti di Industria 4.0; infatti, è possibile configurare l'applicativo in modo da acquisire i dati dal campo attraverso i protocolli di comunicazione dei PLC e condividerli quindi, attraverso un Server OPC-UA, con altri sistemi o macchinari. Le principali funzionalità offerte da Premium HMI sono le seguenti:

- Ampia libreria di driver di comunicazione per i PLC più diffusi sul mercato (tra cui i driver Robox).
- Compatibilità con lo standard VBA (Visual Basic for Application) e ampio set di API per le diverse funzionalità di progetto.
- Una delle migliori tecnologie Web Client per l'accesso da remoto.

Il software Premium HMI permette di configurare un Server OPC-UA senza passare per i concetti fondamentali dello standard (tipi di oggetto, tipi di variabili, etc.), ma è sufficiente definire quali variabili devono essere trasmesse tramite OPC-UA, abilitandone la relativa opzione nella finestra delle proprietà. In questo modo viene comunque generato un Address Space, all'interno del quale le variabili vengono identificate da un NodeId.

3.1.2. RDE (Robox Development Environment)

RDE (Robox Development Environment) è un ambiente di sviluppo concepito per implementare software applicativi utilizzando i linguaggi di programmazione Robox R3, RPE, Ladder e Object Block. Consente inoltre di compilare e debuggare il codice al fine di studiare il comportamento delle macchine controllate e di scegliere le migliori soluzioni per ottimizzarne le dinamiche e le tempistiche. RDE permette inoltre di colloquiare con il controllore Robox utilizzando l'apposita finestra di Shell, la quale permette di:

- Esaminare lo stato di qualunque risorsa hardware del controllore collegato.
- Leggere ed eventualmente modificare il valore di qualunque registro del controllore collegato.
- Leggere, trasferire ed eventualmente cancellare la directory dei file presenti nelle memorie flash del controllore collegato.

I registri dei controllori Robox possono essere divisi in differenti categorie in base al tipo di valore che possono rappresentare e in base alla volatilità del registro. È possibile quindi distinguere tra:

- Registri di tipo “r”: sono registri non ritentivi (volatili) utilizzati per rappresentare valori interi che non necessitano di essere conservati in memoria.
- Registri di tipo “nvr”: sono registri ritentivi (non volatili) utilizzati per rappresentare valori interi che devono essere conservati in memoria.
- Registri di tipo “rr”: sono registri non ritentivi (volatili) utilizzati per rappresentare valori reali che non necessitano di essere conservati in memoria
- Registri di tipo “nvrr”: sono registri ritentivi (non volatili) utilizzati per rappresentare valori reali che devono essere conservati in memoria.
- Registri di tipo “sr”: sono registri non ritentivi (volatili) utilizzati per rappresentare stringhe che non necessitano di essere conservati in memoria
- Registri di tipo “nvsr”: sono registri ritentivi (non volatili) utilizzati per rappresentare stringhe che devono essere conservate in memoria.

L’ambiente di sviluppo RDE permette di definire un “alias” da associare a questi registri in modo da poter successivamente referenziare uno specifico indirizzo con il suo alias, semplificando il lavoro del programmatore.

3.1.3. RDT (Robox Development Tool)

RDT (Robox Development Tool) è un ambiente di sviluppo che permette all’utente di realizzare in maniera semplice ed intuitiva interfacce grafiche che possono essere integrate con tutte le famiglie di controllori Robox. Tali interfacce possono essere eseguite sia in locale sul pc (in questo caso è richiesto anche il software RTM) sia sui terminali operatore offerti dalla Robox. Il software mette a disposizione del programmatore un set di controlli che permettono di sviluppare facilmente l’interfaccia ritenuta più idonea al contesto inserendo azioni, macro, finestre di help e gestendo i livelli di accesso e lo storico allarmi.

3.1.4. UAExpert

UAExpert è un software sviluppato e rilasciato dalla Unified Automation, una delle aziende membro della OPC Foundation. Esso fornisce tutte le funzionalità di base di un OPC-UA Client, come l’accesso ai dati, allo storico dei dati, la sottoscrizione agli eventi e le

chiamate dei metodi e pertanto viene utilizzato come software di test e verifica per la comunicazione tramite standard OPC-UA.

3.1.5. Robox RP2

Il Robox RP2 è un dispositivo hardware, prodotto dalla Robox, atto al controllo assi di una macchina automatiche industriale. È equipaggiato con un microprocessore ARM Cortex A9 con frequenza di clock massima pari a 800 MHz, 1 Gbyte di memoria RAM e 512 Kbyte di memoria ram ritentiva (non volatile) per il salvataggio di parametri e allarmi ed è in grado di gestire fino a 32 assi interpolati, pilotati tramite bus di campo EtherCAT. Le altre caratteristiche del Robox RP2 sono rappresentate dai canali Ethernet per le funzioni di comunicazione in LAN (Local Area Network) e in WAN (Wide Area Network), canali seriali per la comunicazione con il pc e con altri dispositivi Robox, come ad esempio il terminale RHMI. Il dispositivo Robox RP2 è visibile in Figura 20.

Figura 20: controllore Robox RP2

3.1.6. Robox HMI (RHMI)

Il terminale RHMI è un dispositivo hardware, prodotto dalla Robox, progettato per interfacciarsi con il controllore Robox e mostrare a schermo i valori dei registri fisici del controllore stesso. Il terminale RHMI è caratterizzato da un display a colori con software di supervisione integrato ed è disponibile in diversi formati. È equipaggiato con un microprocessore ARM Cortex A8, 512 Mbyte di memoria RAM, 256 Mbyte di memoria Flash per ospitare il sistema operativo WindowsCE e 2 Gbyte di memoria eMMC per ospitare il software di sistema RTM che consente l'esecuzione dei progetti sviluppati con il software RDT. Il dispositivo RHMI è visibile in Figura 21.

Figura 21: terminale grafico RHMI

3.1.7. ASEM RK10

Il router RK10 è un dispositivo hardware, prodotto dalla ASEM, dedicato alla teleassistenza in ambito industriale. È un dispositivo basato su un processore ARM Cortex A8 con frequenza di clock pari a 1 GHz, 1 Gbyte di memoria RAM, 256 Mbyte di memoria Flash per ospitare il sistema operativo WindowsCE e 4 Gbyte di memoria eMMC. Inoltre, è dotato di porte Ethernet per la connessione WAN e LAN. Il router ASEM RK10 permette anche di eseguire e rendere disponibile l'accesso alle interfacce grafiche sviluppate con il software Premium HMI. Il dispositivo è visibile in Figura 22.

Figura 22: router ASEM RK10

3.2. Analisi dei requisiti e progettazione

Prima di procedere con la descrizione dello sviluppo del Server OPC-UA e della gestione dello scambio di dati tra Server, interfaccia RHMI integrata a bordo macchina e gestionale del cliente, è importante definire i requisiti da rispettare affinchè la macchina avvolgitrice sia funzionale alle necessità del cliente e allo stesso tempo conforme agli standard di Industria 4.0. Oltre al requisito fondamentale, rappresentato dall'utilizzo dello standard di comunicazione OPC-UA, è necessario che la macchina avvolgitrice disponga della possibilità di ricevere richieste di nuovi lotti di produzione da remoto. In particolare, alla ricezione della

nuova richiesta, l'operatore deve caricare manualmente la ricetta richiesta agendo sul terminale della macchina avvolgitrice, prima di poter avviare il nuovo lotto di produzione. L'arrivo di una richiesta di nuova produzione va segnalato all'operatore, implementando un lampeggio di uno degli elementi che compongono l'interfaccia grafica mostrata dal terminale. Al caricamento della nuova ricetta deve essere gestita automaticamente la terminazione della produzione attuale e l'avvio della nuova produzione. Nell'ambito dell'automazione industriale il concetto di "ricetta" viene utilizzato per identificare la sequenza di operazioni automatiche che la macchina deve eseguire. Sulla base di tali requisiti tecnici si è scelto di organizzare il Server OPC-UA in strutture dati che rappresentano il lotto di produzione richiesto (Requested Batch), il lotto di produzione attuale (Active Production), il lotto di produzione precedente (Previous Production) e lo stato attuale della macchina (Machine State). Le variabili esposte dal Server per ognuna di queste strutture sono presentate e descritte brevemente di seguito in Tabella 12, Tabella 13, Tabella 14 e Tabella 15.

Nome	Nome Server OPC-UA	Tipo	Descrizione
ID_lotto	RequestedProduction: Batch_ID	Stringa	Codice del lotto di produzione richiesto
Ricetta	RequestedProduction: Recipe_Name	Stringa	Codice o nome ricetta richiesta
Pezzi_da_produrre	RequestedProduction: Parts_To_Be_Produced	Numerico	Numero dei pezzi da produrre per il lotto di produzione richiesto

Tabella 12: organizzazione logica dei dati relativi alla nuova produzione

Nome	Nome Server OPC-UA	Tipo	Descrizione
ID_lotto_attuale	ActiveProduction: Batch_ID	Stringa	Codice del lotto di produzione attuale
Ricetta_attuale	ActiveProduction: Recipe_Name	Stringa	Codice o nome ricetta attuale
Pezzi_da_produrre _attuale	ActiveProduction: Parts_To_Be_Produced	Numerico	Numero dei pezzi da produrre per il lotto di produzione attuale
Pezzi_processati _attuale	ActiveProduction: Produced_Parts	Numerico	Numero di pezzi processati per il lotto di produzione attuale
Pezzi_buoni _attuale	ActiveProduction: OK_Parts	Numerico	Numero di pezzi buoni per il lotto di produzione attuale
Pezzi_scarto _attuale	ActiveProduction: NOK_Parts	Numerico	Numero di pezzi scartati per il lotto di produzione attuale
Data_inizio _attuale	ActiveProduction: Start_Date	Stringa (Data-ora)	Data e ora di inizio lotto di produzione attuale
Data_fine_attuale	ActiveProduction: End_Date	Stringa (Data-ora)	Data e ora di fine lotto di produzione attuale (istanziato solo alla conclusione del lotto)

Tabella 13: organizzazione logica dei dati relativi alla produzione attuale

Nome	Nome Server OPC-UA	Tipo	Descrizione
ID_lotto _precedente	PreviousProduction: Batch_ID	Stringa	Codice del lotto di produzione precedente
Ricetta _precedente	PreviousProduction: Recipe_Name	Stringa	Codice o nome ricetta precedente
Pezzi_da_produrre _precedente	PreviousProduction: Parts_To_Be_Produced	Numerico	Numero dei pezzi da produrre per il lotto di produzione precedente
Pezzi_processati _precedente	PreviousProduction: Produced_Parts	Numerico	Numero di pezzi processati per il lotto di produzione precedente
Pezzi_buoni _ precedente	PreviousProduction: OK_Parts	Numerico	Numero di pezzi buoni per il lotto di produzione precedente
Pezzi_scarto _ precedente	PreviousProduction: NOK_Parts	Numerico	Numero di pezzi scartati per il lotto di produzione precedente
Data_inizio _ precedente	PreviousProduction: Start_Date	Stringa (Data-ora)	Data e ora di inizio lotto di produzione precedente
Data_fine _precdente	PreviousProduction: End_Date	Stringa (Data-ora)	Data e ora di fine lotto di produzione precedente

Tabella 14: organizzazione logica dei dati relativi alla produzione precedente

Nome	Nome Server OPC-UA	Tipo	Descrizione
Stato	MachineState: state_str	Testo	Stato della macchina (Automatico, Manuale, PowerOff, PowerOn, etc.)
Messaggio	MachineState: message_str	Testo	Messaggio di informazione per allarmi, o situazioni particolari (porte aperte, funghi, etc.)
Stato_ID	MachineState: state_ID	Numerico	Identificatore dello stato della macchina (Automatico, Manuale, PowerOff, PowerOn, etc.)
Messaggio_ID	MachineState: message_ID	Numerico	Identificatore del messaggio di informazione per allarmi, o situazioni particolari (porte aperte, funghi, etc.)
Ricetta_caricata	MachineState: recipe_loaded	Testo	Nome o codice della ricetta caricata

Tabella 15: organizzazione logica dei dati relativi allo stato della macchina

Un'altra caratteristica da evidenziare di queste variabili è rappresentata dalla diversificazione tra variabili di ingresso e variabili di uscita per il controllore Robox presente a bordo macchina. Le variabili che formano la struttura relativa alla richiesta del nuovo lotto di produzione (Requested Batch) sono variabili di ingresso al controllore, in quanto dovranno essere istanziate da remoto dal cliente per comunicare le indicazioni in merito al nuovo lotto di produzione; le variabili rimanenti invece hanno la funzione opposta, ovvero quella di rendere disponibili le informazioni relative al lotto di produzione attuale, al lotto di produzione precedente e allo stato della macchina per il cliente che accede da remoto.

L'infrastruttura di rete che consente lo scambio di questi dati tra la macchina avvolgitrice ed il gestionale del cliente è organizzata come presentato in Figura 23.

Figura 23: infrastruttra di rete che permette lo scambio di dati tra macchina e gestionale del cliente

Il controllore della macchina avvolgitrice, il Robox RP2, comunica da un lato con il terminale RHMI che è utilizzato per comunicare all'operatore tutti i dati relativi alle variabili appena presentate tramite apposita interfaccia grafica, mentre dall'altro lato comunica con il router ASEM RK10, sul quale gira nell'effettivo il Server OPC-UA. Lo scambio di dati tra il Robox RP2 e il router ASEM RK10 avviene tramite protocollo Ethernet/IP e grazie alle librerie di comunicazione Robox, disponibili nativamente sul software Premium HMI, che consentono di mappare le variabili definite in Premium HMI su specifici registri fisici del Robox RP2, come descritto più nel dettaglio nel paragrafo 3.3.3. Inoltre, sul router ASEM RK10 sono state configurate sia la porta LAN che la porta WAN per consentire l'accesso al Server OPC-UA rispettivamente dalla rete locale (LAN), collegandosi quindi direttamente al dispositivo ASEM, e da remoto tramite la rete globale (WAN). In quest'ultimo caso entrano in gioco anche tematiche legate alla sicurezza del dato e alla sicurezza degli accessi tramite WAN che però non sono affrontate all'interno di questo lavoro e pertanto lasciate al cliente finale della macchina avvolgitrice. In entrambi i casi, invece, i dati sono esposti dal Server OPC-UA tramite l'interfaccia grafica realizzata con il software Premium HMI, descritta nel paragrafo 3.3.3.

3.3. Sviluppo

L'implementazione e lo sviluppo di quanto appena descritto passa per i seguenti punti:

- Definizione dei registri fisici del Robox RP2 da associare alle variabili esposte dal Server OPC-UA e implementazione della funzione che permette di aggiornare tali variabili.
- Realizzazione dell'interfaccia grafica per esporre tali variabili all'operatore tramite il terminale RHMI presente a bordo macchina.

- Realizzazione dell'interfaccia grafica tramite software Premium HMI per consentire l'accesso, sia in locale che da remoto, al Server OPC-UA.
- Definizione delle strutture dati che formano il Server OPC-UA e conseguente mappatura sugli indirizzi fisici del Robox RP2.

3.3.1. Software RDE

Per permettere al Robox RP2 di scambiare dati ed informazioni da un lato con il terminale e dall'altro con il Server OPC-UA è necessario riservare dei registri fisici che possano contenere le istanze di questi dati. Come descritto precedentemente nel paragrafo 3.1.2, i registri fisici del Robox RP2 si dividono in 6 categorie: registri interi non ritentivi (identificati con "r"), registri interi ritentivi (identificati con "nvr"), registri reali non ritentivi (identificati con "rr"), registri reali ritentivi (identificati con "nvrr"), registri stringa non ritentivi (identificati con "sr"), registri stringa ritentivi (identificati con "nvsr"). Sulla base di questi fatti, per rappresentare dunque le variabili necessarie alla comunicazione con il Server OPC-UA, in Figura 24 sono visibili i registri fisici riservati nel controllore Robox RP2. Per le variabili esposte dal Server OPC-UA sono utilizzati:

- I registri "nvr" da 2401 a 2408, compresi, per le variabili che rappresentano i pezzi processati, i pezzi buoni, i pezzi scartati e i pezzi da produrre per il lotto di produzione attuale e per quello precedente.
- I registri "r" da 2401 a 2403, compresi, per le variabili che rappresentano pezzi da produrre per il lotto di produzione richiesto da remoto e gli identificatori dello stato macchina e del messaggio.
- I registri "nvsr" da 1 a 8, compresi, per le variabili che rappresentano l'identificatore del lotto, il nome della ricetta (programma in Figura 24), la data di inizio e la data di fine per il lotto di produzione attuale e per quello precedente.
- I registri "sr" da 1 a 5, compresi, per le variabili che rappresentano l'identificatore del lotto e il nome della ricetta per il lotto di produzione richiesto da remoto, lo stato macchina, il messaggio e la ricetta caricata.

In Figura 24 sono inoltre visibili gli alias associati ai registri appena elencati.

```
1200    ;////////////////////////////////////////////////////////////////////////////
1201    ; RITENTIVE STRING REGISTER -NVSR
1202    ;////////////////////////////////////////////////////////////////////////////
1203
1204    ;<<<<<<<<<<< RITENTIVE INTEGER REGISTER -NVR- >>>>>>>>>>>
1205    lit OPCUA_pezzi_processati_attuali      nvr(2401)
1206    lit OPCUA_pezzi_buoni_attuali           nvr(2402)
1207    lit OPCUA_pezzi_scarto_attuali          nvr(2403)
1208    lit OPCUA_pezzi_da_produrre_attuali     nvr(2404)
1209    lit OPCUA_pezzi_processati_precedenti   nvr(2405)
1210    lit OPCUA_pezzi_buoni_precedenti        nvr(2406)
1211    lit OPCUA_pezzi_scarto_precedenti       nvr(2407)
1212    lit OPCUA_pezzi_da_produrre_precedenti  nvr(2408)
1213
1214    ;<<<<<<<<<<< NO RITENTIVE INTEGER REGISTER -R- >>>>>>>>>>>
1215    lit OPCUA_pezzi_da_produrre         r(2401)
1216    lit OPCUA_stato                     r(2402)
1217    lit OPCUA_messaggio                 r(2403)
1218
1219    lit HMI_ricetta_comp                r(2410) ;HMI_variable
1220    lit cmd_HMI_START_LOTTO             r(2411) ;HMI_command
1221    lit cmd_HMI_FINE_LOTTO              r(2412) ;HMI_command
1222
1223    ;<<<<<<<<<<< RITENTIVE STRING REGISTER -NVSR- >>>>>>>>>>>
1224    lit OPCUA_ID_lotto_attuale          nvsr(1)
1225    lit OPCUA_programma_attuale         nvsr(2)
1226    lit OPCUA_data_inizio_attuale       nvsr(3)
1227    lit OPCUA_data_fine_attuale         nvsr(4)
1228    lit OPCUA_ID_lotto_precedente       nvsr(5)
1229    lit OPCUA_programma_precedente      nvsr(6)
1230    lit OPCUA_data_inizio_precedente    nvsr(7)
1231    lit OPCUA_data_fine_precedente      nvsr(8)
1232
1233    ;<<<<<<<<<<< NO RITENTIVE STRING REGISTER -SR- >>>>>>>>>>>
1234    lit OPCUA_ID_lotto              sr(1)
1235    lit OPCUA_programma             sr(2)
1236    lit OPCUA_stato_str             sr(3)
1237    lit OPCUA_messaggio_str         sr(4)
1238    lit OPCUA_programma_loaded      sr(5)
```

Figura 24: definizione dei registri del Robox RP2 riservati alla comunicazione OPC-UA

Oltre agli indirizzi appena elencati, necessari per lo scambio di informazioni con il Server OPC-UA, sono stati riservati i registri “r” da 2410 a 2412, compresi, per definire degli alias da mappare con variabili e comandi definiti per l’interfaccia grafica che sarà visibile sul terminale. In particolare, questi tre alias rappresentano rispettivamente, lato Robox RP2, la variabile che gestisce il lampeggio del pulsante “AVVIA NUOVA PRODUZIONE”, per segnalare all’operatore la presenza di una nuova richiesta, e i comandi di avvio di un nuovo lotto di produzione e di conclusione del lotto di produzione attuale, inviati dai rispettivi pulsanti. Definiti quindi i registri del Robox RP2 riservati per le variabili OPC-UA, si può passare ad implementare la funzione che ha il compito di aggiornare tali registri, al fine di rendere disponibile sempre il valore più recente in entrambe le direzioni della comunicazione. La funzione riservata a questo compito è visibile in Figura 25. Questa funzione è inserita all’interno di tutto il codice di controllo della macchina avvolgitrice e viene eseguita ad ogni

ciclo di controllo; pertanto si è cercato di integrare il più possibile la sua definizione con quanto era già presente nel codice, allo scopo rendere minimo l'impatto della sua esecuzione sull'intero ciclo di controllo della macchina, in termini computazionali.

```
2400    ;<<<<  OPC-UA >>>>>>>
2401    ;-----------------------------------------------------
2402    function  OPCUA_01
2403    ;-----------------------------------------------------
2404
2405        ricetta_comp = (NOT (str_equal(OPCUA_programma,OPCUA_programma_attuale))) AND (NOT (str_equal(OPCUA_programma,"")))
2406        ricetta_comp = ricetta_comp OR (NOT (str_equal(OPCUA_ID_lotto,OPCUA_ID_lotto_attuale))) AND (NOT (str_equal(OPCUA_ID_lotto,"")))
2407
2408        if ((rise(cmd_HMI_START_LOTTO)) OR rise(NOT (str_equal(OPCUA_programma_temp,OPCUA_programma_loaded))))
2409            OPCUA_programma_temp = OPCUA_programma_loaded
2410            if (str_equal(OPCUA_data_fine_attuale,""))
2411                str_format(OPCUA_data,"%02d-%02d-%04d %02d:%02d:%02d", day, month, year, hour, minute, second)
2412                OPCUA_data_fine_attuale = OPCUA_data
2413            endif
2414            OPCUA_ID_lotto_precedente = OPCUA_ID_lotto_attuale
2415            OPCUA_programma_precedente = OPCUA_programma_attuale
2416            OPCUA_data_inizio_precedente = OPCUA_data_inizio_attuale
2417            OPCUA_data_fine_precedente = OPCUA_data_fine_attuale
2418            OPCUA_pezzi_processati_precedenti = OPCUA_pezzi_processati_attuali
2419            OPCUA_pezzi_buoni_precedenti = OPCUA_pezzi_buoni_attuali
2420            OPCUA_pezzi_scarto_precedenti = OPCUA_pezzi_scarto_attuali
2421            OPCUA_pezzi_da_produrre_precedenti = OPCUA_pezzi_da_produrre_attuali
2422            str_format(OPCUA_data,"%02d-%02d-%04d %02d:%02d:%02d", day, month, year, hour, minute, second)
2423            OPCUA_data_inizio_attuale = OPCUA_data
2424            OPCUA_data_fine_attuale = ""
2425            OPCUA_programma_attuale = OPCUA_programma_loaded
2426            OPCUA_ID_lotto_attuale = OPCUA_ID_lotto
2427            OPCUA_pezzi_da_produrre_attuali = OPCUA_pezzi_da_produrre
2428            OPCUA_pezzi_processati_attuali = 0
2429            OPCUA_pezzi_buoni_attuali = 0
2430            OPCUA_pezzi_scarto_attuali = 0
2431        endif
2432
2433        if (rise(cmd_HMI_FINE_LOTTO))
2434            str_format(OPCUA_data,"%02d-%02d-%04d %02d:%02d:%02d", day, month, year, hour, minute, second)
2435            OPCUA_data_fine_attuale = OPCUA_data
2436        endif
2437
2438        OPCUA_stato = rOpeMode
2439        OPCUA_messaggio = MessaggioAttuale
2440
2441    end_fun
```

Figura 25: ciclo di controllo dei dati OPC-UA

Il primo valore che viene aggiornato ad ogni ciclo è quello relativo al registro fisico del Robox RP2 referenziato dall'alias "ricetta_comp", il quale controlla il lampeggio del pulsante "AVVIA NUOVA PRODUZIONE" per segnalare l'arrivo di una richiesta di nuovo lotto all'operatore. Come descritto più nel dettaglio nel paragrafo 3.3.2, dedicato all'interfaccia grafica da presentare sul terminale RHMI, quando questo registro assume valore "TRUE" (corrispondente al valore logico "1" per i registri di tipo "r" del Robox RP2) si abilita l'oggetto che gestisce il lampeggio del pulsante "AVVIA NUOVA PRODUZIONE", viceversa quando il registro assume valore "FALSE" (corrispondente al valore logico "0") l'oggetto viene disabilitato e di conseguenza non è più visibile sulla schermata. Affinchè questo registro assuma valore "TRUE" si devono verificare le seguenti condizioni:

- La ricetta per il nuovo lotto di produzione richiesto deve essere diversa da quella attuale, oppure l'identificatore del nuovo lotto di produzione richiesto deve essere diverso dall'identificatore del lotto attuale.
- La ricetta e l'identificatore del nuovo lotto di produzione richiesto non devono essere registri vuoti.

La seconda condizione è stata inserita per evitare che il lampeggio del pulsante di avvio di una nuova produzione fosse abilitato anche all'avvio della macchina, quando le variabili di richiesta del lotto di produzione risultano essere vuote, essendo associate a registri non ritentivi. Il corpo centrale della funzione, invece, è caratterizzato da due blocchi "if" che si occupano di aggiornare i valori dei registri legati alle variabili che contengono le informazioni relativi al lotto di produzione attuale e a quello precedente. In particolare, il primo viene eseguito nei casi in cui l'operatore prema il pulsante "AVVIA NUOVA PRODUZIONE" agendo sul terminale RHMI oppure carichi una nuova ricetta a seguito di una richiesta di un nuovo lotto di produzione; il secondo blocco viene invece eseguito solo nel caso in cui l'operatore prema il pulsante "CONCLUDI PRODUZIONE ATTUALE". Il primo blocco "if" si occupa di trasferire i valori del lotto di produzione attuale nei registri associati alle variabili del lotto di produzione precedente e successivamente di aggiornarli con i valori del nuovo lotto di produzione richiesto; inoltre, è stato implementato in modo tale che nel caso in cui il lotto di produzione attuale non fosse ancora concluso, esso gestisca anche l'aggiornamento della data di fine produzione del lotto (questa condizione è facilmente verificabile in quanto un lotto non concluso non può avere una data di fine). Per gestire l'avvio di una nuova produzione a seguito del caricamento di una nuova ricetta, effettuato manualmente dall'operatore, si utilizza la variabile di appoggio "OPCUA_programma_temp" che ad ogni ciclo di lavoro viene in primo luogo confrontata con la ricetta caricata (OPCUA_programma_loaded) e immediatamente dopo viene aggiornata con tale valore. Il secondo blocco "if" si occupa solamente settare la data di fine del lotto di produzione attuale. Infine, ad ogni ciclo, la funzione si occupa di aggiornare gli identificatori dello stato e del messaggio della macchina, utilizzando gli alias dei registri che erano già stati definiti precedentemente alla definizione della comunicazione OPC-UA per rappresentare lo stato ed il messaggio della macchina. Lo stato e il messaggio in formato testuale non vengono gestiti all'interno di questo codice, ma sono le due interfacce grafiche, quella mostrata dal terminale e quella di accesso al Server OPC-UA, ad associare i corrispettivi formati testuali agli identificatori. L'idea iniziale era quella di gestire l'aggiornamento delle stringhe dello stato e del messaggio della macchina all'interno del ciclo di controllo, ma si è presentata una problematica legata al passaggio di stringhe tra il Robox RP2 e il terminale RHMI: il passaggio di stringhe è infatti limitato ad una dimensione massima e, superando tale limite, il passaggio viene troncato. Ciò comporta sia errori nella visualizzazione delle stringhe nell'interfaccia esposta dal terminale RHMI sia un passaggio errato dei valori testuali al Server OPC-UA. Le uniche stringhe che non dovevano essere aggiornate agendo obbligatoriamente sui registri del Robox RP2 erano le due stringhe dello stato e del messaggio della macchina; pertanto, si

è deciso di proseguire come descritto precedentemente, riuscendo a risolvere tale problematica.

3.3.2. *Interfaccia grafica del terminale*

L'interfaccia grafica, realizzata per mostrare i dati esposti dal Server OPC-UA sul terminale RHMI e tramite la quale l'operatore può inviare comandi di inizio di una nuova produzione e conclusione della produzione attuale, si presenta come visibile in Figura 26.

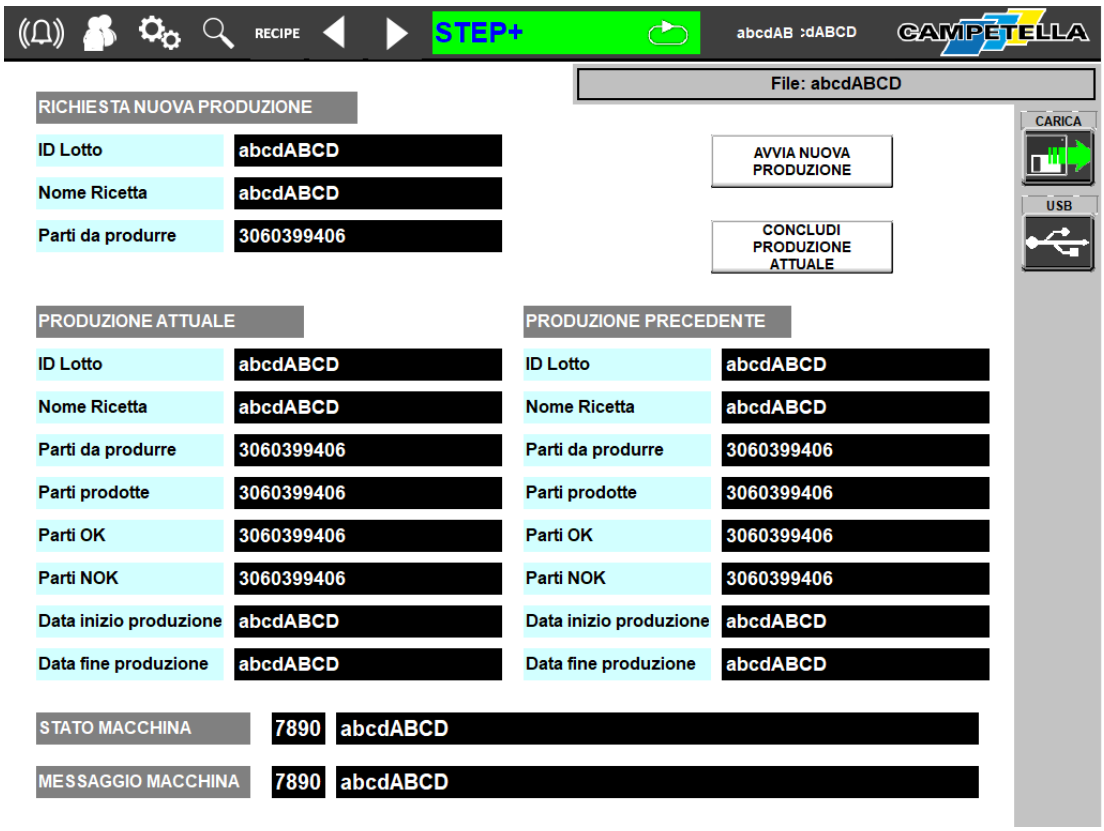


Figura 26: struttra della schermata visibile tramite il terminale RHMI

Si è scelto di realizzare tale schermata seguendo la struttura, i colori ed il font del testo delle altre schermate che erano già state implementate per il progetto macchina avvolgitrice sul quale è stata aggiunta la comunicazione OPC-UA, allo scopo di rendere l'esperienza d'uso del terminale RHMI il più lineare possibile. La struttura di base dalla quale si è partiti per realizzare tale interfaccia consisteva nella barra di navigazione posta in alto, nella barra dei comandi posta nel margine destro della schermata e nell'oggetto posto tra le due che ha il compito di riportare sulla schermata la ricetta caricata pronta ad essere eseguita. La parte rimanente della schermata è stata realizzata utilizzando alcuni oggetti messi a disposizione nativamente dal software RDT. In particolare, va evidenziato l'utilizzo di 2 pulsanti per permettere all'operatore di avviare una nuova produzione e di concludere la produzione attuale e 1 immagine dinamica, non visibile in Figura 26, per realizzare il lampeggio del

pulsante “AVVIA NUOVA PRODUZIONE”. Il resto degli oggetti visibili nella schermata sono oggetti di tipo “Testo”, che consentono di mostrare un testo fisso sulla schermata, e di tipo “Edit”, che invece consentono di mostrare testi e valori numerici che possono variare durante l’esecuzione. Questi ultimi, i campi a sfondo nero in Figura 26, sono utilizzati di fatto per mostrare sulla schermata il contenuto dei registri riservati alla comunicazione OPC-UA sul Robox RP2. Per effettuare tale operazione è necessario definire un numero di variabili pari al numero di valori che si vogliono mostrare a schermo e mappare tali variabili sui corrispettivi registri. Le variabili definite per implementare quanto appena descritto sono visibili in Figura 27, in cui per ogni variabile sono mostrati anche: il nome della variabile, il tipo di connessione con il Robox RP2, il tipo e il numero di registro.

OPCUA_stato_str	P_SERIALE	SR	3
OPCUA_stato	P_SERIALE	R32	2402
OPCUA_ricetta_comp	P_SERIALE	R32	2410
OPCUA_programma_precedente	P_SERIALE	NVSR	6
OPCUA_programma_loaded	P_SERIALE	SR	5
OPCUA_programma_attuale	P_SERIALE	NVSR	2
OPCUA_programma	P_SERIALE	SR	2
OPCUA_pezzi_scarto_precedenti	P_SERIALE	NVR32	2407
OPCUA_pezzi_scarto_attuali	P_SERIALE	NVR32	2403
OPCUA_pezzi_processati_precedenti	P_SERIALE	NVR32	2405
OPCUA_pezzi_processati_attuali	P_SERIALE	NVR32	2401
OPCUA_pezzi_da_produrre_precedenti	P_SERIALE	NVR32	2408
OPCUA_pezzi_da_produrre_attuali	P_SERIALE	NVR32	2404
OPCUA_pezzi_da_produrre	P_SERIALE	R32	2401
OPCUA_pezzi_buoni_precedenti	P_SERIALE	NVR32	2406
OPCUA_pezzi_buoni_attuali	P_SERIALE	NVR32	2402
OPCUA_messaggio_str	P_SERIALE	SR	4
OPCUA_messaggio	P_SERIALE	R32	2403
OPCUA_ID_lotto_precedente	P_SERIALE	NVSR	5
OPCUA_ID_lotto_attuale	P_SERIALE	NVSR	1
OPCUA_ID_lotto	P_SERIALE	SR	1
OPCUA_data_inizio_precedente	P_SERIALE	NVSR	7
OPCUA_data_inizio_attuale	P_SERIALE	NVSR	3
OPCUA_data_fine_precedente	P_SERIALE	NVSR	8
OPCUA_data_fine_attuale	P_SERIALE	NVSR	4

Figura 27: definzione delle variabili necessarie alla comunicazione OPC-UA per il terminale RHMI

In Figura 27 è visibile anche la variabile “OPCUA_ricetta_comp”, la quale ha il compito di gestire il lampeggio del pulsante di avvio di una nuova produzione andando a rendere visibile l’immagine dinamica, posta sotto al pulsante, e che mostra a schermo l’immagine di un led rosso e quella di un led giallo passando da un’immagine all’altra con un timer pari a 5 millisecondi. Oltre a queste variabili sono definite altre due variabili che vengono settate al valore logico “1” rispettivamente quando i due pulsanti sono premuti e pertanto sono associate ai registri del Robox RP2 riservati alla gestione dei comandi inviati dai pulsanti. Più nello specifico, i pulsanti visibili sulla schermata eseguono rispettivamente i comandi “CMD_OPCUA_START” e “CMD_OPCUA_START”, i quali a loro volta hanno il compito di settare le suddette variabili. Come già accennato precedentemente nel paragrafo 3.3.1,

l'associazione tra gli identificatori dello stato e del messaggio della macchina con le rispettive stringhe viene effettuata dalle due interfacce grafiche. Per l'interfaccia grafica che viene mostrata sul terminale RHMI, oggetto di questo paragrafo, il software RDT consente di definire dei vettori di stringhe. Facendo corrispondere gli identificatori dello stato e del messaggio macchina con l'indice di accesso al vettore, è di conseguenza possibile associare ad ogni identificatore la rispettiva stringa. Per semplicità, l'associazione identificatore-stringa per lo stato della macchina e per il messaggio della macchina è riportata rispettivamente in Tabella 16 e Tabella 17.

ID Stato	Stringa
0	NULL
1	MISSING POWER
2	POWER ON
3	RESET
4	STEP
5	MANUAL
6	AUTOMATIC
7	ALARM

Tabella 16: elenco degli stati macchina

ID Messaggio	Stringa
0	NULL
1	RIPETERE ENERGIZZAZIONE MACCHINA
2	PRESSIONE ARIA BASSA
3	FUNGHI EMERGENZA NON OK
4	ATTESA RESET
5	RESET ASSI
6	RESET FINTO
7	ALLARME GENERALE CONTROLLO
8	VERIFICARE CONDIZIONI INIZIALI DI PARTENZA
9	PROTEZIONI NON OK
10	REGGIA NON OK LEGATORE 1
11	BRACCIO REGGIA BASSO LEGATORE 1
12	SVOLGIREGGIA LEGATORE 1
13	RESET REGGIA NON OK LEGATORE
14	NASTRO USCITA NON LIBERO
15	ATTESA TASTO MARCIA
16	INSERIRE LEGATORE
17	PARATIA INTERNA NON APERTA
18	PARATIA NON CHIUSA
19	MANCATO MOVIMENTO CANALE TUBO
20	MANCATO MOVIMNETO COLTELLO
21	MANCATA APERTURA SPATOLE ASPO 1
22	ATTESA ESPULSORI INDIETRO ASPO 1
23	ATTESA ROTAZIONE ESPULSORI ASPO 1

24	ATTESA ROTAZIONE NASTRO SCARICO
25	ATTESA PULSANTE RESET CONTAMETRI
26	ATTESA PULSANTE AVVOLGIMENTO ON
27	ATTESA PULSANTE START ASPO 1
28	ATTESA FC CANALE AGGANCIO ALTO
29	ENTRAMBE ASPI PIENI
30	ATTESA TASTO STEP
31	ATTESA FINECORSA LEGATORE INDIETRO
32	ATTESA AVANZAMENTO ESPULSORI
33	DISABILITARE IL LEGATORE
34	MANCATA SALITA CANALE
35	MANCATA DISCESA CANALE
36	ROBOT IN RIPOSIZIONAMENTO
37	MANCATA DISCESA LAMA SALDANTE
38	MANCATA VARIAZIONE FORZA MOTORE
39	ALLARME LEGATURA NON ESEGUITA
40	ASPO NON IN POSIZIONE DI ESPULSIONE

Tabella 17: elenco dei messaggi macchina

3.3.3. Interfaccia grafica ed implementazione del Server OPC-UA

In questo paragrafo è descritta la procedura seguita per realizzare la comunicazione OPC-UA tra il gestionale del cliente e il controllore Robox RP2, attraverso il software Premium HMI. In primo luogo, è descritta l'interfaccia grafica che permette di accedere al Server OPC-UA e pertanto di leggere o scrivere le variabili esposte dal server stesso; successivamente sono riportati gli step necessari a mappare le variabili esposte dall'interfaccia sui registri del Robox RP2 e all'abilitazione della comunicazione OPC-UA per l'accesso da remoto. Per realizzare l'interfaccia grafica sono stati utilizzati due diversi tipi di oggetto messi a disposizione nativamente dal software Premium HMI:

- Oggetti di tipo "Testo" che permettono di visualizzare sulla schermata un testo statico.
- Oggetti di tipo "Display" che consentono di visualizzare sulla schermata i valori delle variabili cui vengono associati ed eventualmente di poterli modificare.

Si ricorda che per tale applicativo, il cliente che accede da remoto al Server OPC-UA può solamente comunicare una nuova produzione; pertanto, sulla base di tali esigenze si è reso possibile modificare soltanto i valori delle variabili relative alla richiesta di una nuova produzione. Perciò, gli oggetti "Display" associati a tali variabili consentono sia la lettura che la scrittura, mentre gli altri oggetti hanno solamente lo scopo di mostrare sulla schermata i

valori assunti dalle rispettive variabili, non consentendo la scrittura. L'interfaccia grafica così realizzata si presenta dunque come in Figura 28.

RICHIESTA NUOVA PRODUZIONE	
ID LOTTO	x
NOME RICETTA	x
PARTI DA PRODURRE	x

RICETTA CARICATA	x

PRODUZIONE ATTUALE	
ID LOTTO	x
NOME RICETTA	x
PARTI DA PRODURRE	x
PARTI PRODOTTE	x
PARTI OK	x
PARTI OK	x
DATA INIZIO PRODUZIONE	x
DATA FINE PRODUZIONE	x

PRODUZIONE PRECEDENTE	
ID LOTTO	x
NOME RICETTA	x
PARTI DA PRODURRE	x
PARTI PRODOTTE	x
PARTI OK	x
PARTI OK	x
DATA INIZIO PRODUZIONE	x
DATA FINE PRODUZIONE	x

STATO MACCHINA	x	x
MESSAGGI MACCHINA	x	x

Figura 28: interfaccia grafica per l'accesso al Server OPC-UA

Le variabili definite nel software Premium HMI, visibili in Figura 29, possono essere divise in due differenti categorie:

- Variabili semplici
- Variabili struttura

Le variabili semplici sono variabili di tipo "Signed Word" o di tipo "String" che saranno utilizzate per essere mappate sui registri fisici del Robox RP2, mentre le variabili struttura raggruppano queste variabili semplici all'interno di strutture più complesse che sono poi abilitate per la comunicazione OPC-UA da remoto.

Nome	Tipo
Sign DWord (32 Bit con segno)	
pezzi_da_produrre_attuali	Sign DWord (32 Bit con segno)
messaggio	Sign DWord (32 Bit con segno)
pezzi_processati_precedenti	Sign DWord (32 Bit con segno)
pezzi_da_produrre	Sign DWord (32 Bit con segno)
pezzi_buoni_precedenti	Sign DWord (32 Bit con segno)
pezzi_scarto_attuali	Sign DWord (32 Bit con segno)
pezzi_processati_attuali	Sign DWord (32 Bit con segno)
pezzi_buoni_attuali	Sign DWord (32 Bit con segno)
stato	Sign DWord (32 Bit con segno)
pezzi_da_produrre_precedenti	Sign DWord (32 Bit con segno)
pezzi_scarto_precedenti	Sign DWord (32 Bit con segno)
Stringa (Terminazione con 0)	
stato_str	Stringa (Terminazione con 0)
data_inizio_precedente	Stringa (Terminazione con 0)
programma_precedente	Stringa (Terminazione con 0)
ID_lotto_attuale	Stringa (Terminazione con 0)
ID_lotto	Stringa (Terminazione con 0)
messaggio_str	Stringa (Terminazione con 0)
programma	Stringa (Terminazione con 0)
data_fine_precedente	Stringa (Terminazione con 0)
programma_caricato	Stringa (Terminazione con 0)
ID_lotto_precedente	Stringa (Terminazione con 0)
programma_attuale	Stringa (Terminazione con 0)
data_fine_attuale	Stringa (Terminazione con 0)
data_inizio_attuale	Stringa (Terminazione con 0)
Struttura (ActiveProduction)	
Active_Production	Struttura (ActiveProduction)
Struttura (MachineState)	
Machine_State	Struttura (MachineState)
Struttura (PreviousProduction)	
Previous_Production	Struttura (PreviousProduction)
Struttura (RequestedProduction)	
Requested_Production	Struttura (RequestedProduction)

Figura 29: variabili definite per l'interfaccia grafica di accesso al Server OPC-UA

Per mappare le variabili semplici sui registri fisici del Robox RP2, Premium HMI mette a disposizione una libreria dei driver di comunicazione dei dispositivi di controllo più diffusi nel settore dell'automazione industriale. Aggiungendo quindi il driver di comunicazione della Robox, è possibile raggiungere direttamente il controllore con il quale l'interfaccia deve comunicare. In particolare, per abilitare tale comunicazione è stato indicato l'indirizzo IP con il quale il Robox RP2 espone i propri registri al router ASEM RK10, così come indicato in Figura 23, sul quale gira l'applicativo che si sta descrivendo in questo paragrafo. Mappate correttamente tali variabili con i rispettivi registri del Robox RP2, le variabili abilitate alla scrittura possono sovrascrivere i valori di tali registri e quindi agendo sull'interfaccia sarà possibile comunicare alla macchina i nuovi valori; viceversa, le variabili in sola lettura mostrano semplicemente il valore più recente per quel registro. Le variabili struttura, oltre ad essere abilitate per la comunicazione OPC-UA e a raggruppare le variabili semplici in strutture più articolate, rispettando il senso logico presentato in Tabella 12, Tabella 13, Tabella 14 e Tabella 15, hanno il compito di associare i valori inseriti o visualizzati negli oggetti "Display" con le rispettive variabili semplici. Per realizzare tale associazione si è resa necessaria l'implementazione di un VBA script, riportato in Figura 30.

```
Sub On_SysVar__ToggleBitsChanged(ByRef value As Variant)

    'update requested_production variables
    ID_lotto = Requested_Production.Batch_ID
    programma = Requested_Production.Program_Name
    pezzi_da_produrre = Requested_Production.Parts_To_Be_Produced

    'update machine_state variables
    Machine_State.state_ID = stato
    Machine_State.state_str = stato_str
    Machine_State.message_ID = messaggio
    Machine_State.message_str = messaggio_str
    Machine_State.Loaded_Program = programma_caricato

    'update active_production variables
    Active_Production.Accepted_Parts = pezzi_buoni_attuali
    Active_Production.Batch_ID = ID_lotto_attuale
    Active_Production.End_Date = data_fine_attuale
    Active_Production.Parts_To_Be_Produced = pezzi_da_produrre_attuali
    Active_Production.Produced_Parts = pezzi_processati_attuali
    Active_Production.Progam_Name = programma_attuale
    Active_Production.Rejected_Parts = pezzi_scarto_attuali
    Active_Production.Start_Date = data_inizio_attuale

    'update previous_production variables
    Previous_Production.Accepted_Parts = pezzi_buoni_precedenti
    Previous_Production.Batch_ID = ID_lotto_precedente
    Previous_Production.End_Date = data_fine_precedente
    Previous_Production.Parts_To_Be_Produced = pezzi_da_produrre_precedenti
    Previous_Production.Produced_Parts = pezzi_processati_precedenti
    Previous_Production.Progam_Name = programma_precedente
    Previous_Production.Rejected_Parts = pezzi_scarto_precedenti
    Previous_Production.Start_Date = data_inizio_precedente

    'update str variables
    stato_str =  StringFromID ("STATE" + stato)
    message_str = StringFromID ("MESSAGE" + messaggio)

End Sub
```

Figura 30: VB Script implementato per associare le variabili esposte dal Server OPC-UA con le variabili associate ai registri fisici del controllore Robox RP2

Lo script implementato ha il compito di aggiornare le variabili semplici associate ai registri fisici del Robox RP2 con i valori inseriti dall'utente tramite interfaccia, per quelle variabili che rappresentano una richiesta di una nuova produzione (Requested Production) e che dunque vanno comunicate da remoto in direzione del controllore della macchina. Viceversa, per quelle variabili che seguono il percorso inverso, ovvero dal controllore all'interfaccia del Server OPC-UA, lo script deve aggiornare le singole componenti delle variabili struttura (Machine_State, Active_Production e Previous_Production) con le rispettive variabili semplici. Oltre a ciò, lo script si occupa di associare le stringhe di stato e messaggio della macchina con i rispettivi identificatori. A differenza di quanto accade con il software RDT, Premium HMI consente di definire un identificatore interno per ogni stringa che deve essere esposta sulla schermata. Definendo in modo opportuno questi identificatori interni, come mostrato in Tabella 18, in cui per semplicità sono riportate solamente le stringhe relative allo stato macchina, lo script riesce a mappare automaticamente le stringhe corrette con i rispettivi identificatori grazie al comando "StringFromID".

ID Stringa	Testo stringa
STATE0	NULL
STATE 1	MISSING POWER
STATE 2	POWER ON
STATE 3	RESET
STATE 4	STEP
STATE 5	MANUAL
STATE 6	AUTOMATIC
STATE 7	ALARM

Tabella 18: elenco delle stringhe definite nella realizzazione dell'interfaccia grafica di accesso al Server OPC-UA

L'ultimo step è quello di abilitare la comunicazione OPC-UA effettuando due operazioni differenti:

- Selezionare l'opzione "Abilita OPC-UA Server" tra le proprietà di tutte le variabili struttura.
- Definire l'URL (Uniform Resource Locator) dell'endpoint che permette il collegamento al Server OPC-UA.

Per definire un URL di un endpoint OPC-UA va specificato il protocollo da utilizzare, l'indirizzo IP che identifica l'host che rende disponibile il server e la porta che definisce la connessione socket (Indirizzo IP: porta). In questo caso, l'endpoint è identificato dall'URL: $opc.tcp://[NodeName]:62840$, in questo modo la label "[NodeName]" verrà sostituita automaticamente con l'indirizzo IP assegnato all'host presso il quale è disponibile il Server OPC-UA. Si ricorda che per il caso analizzato in questo capitolo, l'applicativo viene simulato dal router ASEM RK10.

3.4. Testing

Per dimostrare la corretta configurazione ed implementazione dell'infrastruttura appena descritta e simulare una situazione tipica di lavoro per il quale l'applicativo è stato realizzato, sono stati effettuati alcuni test nei quali i dati relativi alle richieste di nuove produzioni sono inseriti operando sull'interfaccia grafica che rappresenta l'accesso al Server OPC-UA, mentre i dati relativi alla produzione attuale, alla produzione precedente, allo stato e al messaggio della macchina sono inseriti agendo direttamente sui registri fisici del Robox RP2 tramite la shell messa a disposizione dal software RDE. Il tutto è stato simulato in locale su un unico pc sul quale erano in esecuzione le due interfacce grafiche e collegandolo unicamente al controllore Robox RP2. In Figura 31 e Figura 32 sono visibili le due interfacce grafiche a seguito di tali test.

RICHIESTA NUOVA PRODUZIONE	
ID LOTTO	Lotto Prod. 3
NOME RICETTA	Recipe_1.r01
PARTI DA PRODURRE	560

RICETTA CARICATA	Recipe_1.r01

PRODUZIONE ATTUALE	
ID LOTTO	Lotto Prod. 2
NOME RICETTA	Recipe_2.r01
PARTI DA PRODURRE	1000
PARTI PRODOTTE	10
PARTI OK	9
PARTI NOK	1
DATA INIZIO PRODUZIONE	27-01-2022 08:50:03
DATA FINE PRODUZIONE	

PRODUZIONE PRECEDENTE	
ID LOTTO	Lotto Prod. 1
NOME RICETTA	Recipe_1.r01
PARTI DA PRODURRE	180
PARTI PRODOTTE	200
PARTI OK	198
PARTI NOK	2
DATA INIZIO PRODUZIONE	27-01-2022 08:49:01
DATA FINE PRODUZIONE	27-01-2022 08:49:31

STATO MACCHINA	1	MISSING POWER
MESSAGGI MACCHINA	12	SVOLGIREGGIA LEGATORE 1

Figura 31: interfaccia grafica di accesso al Server OPC-UA a seguito dei test effettuati

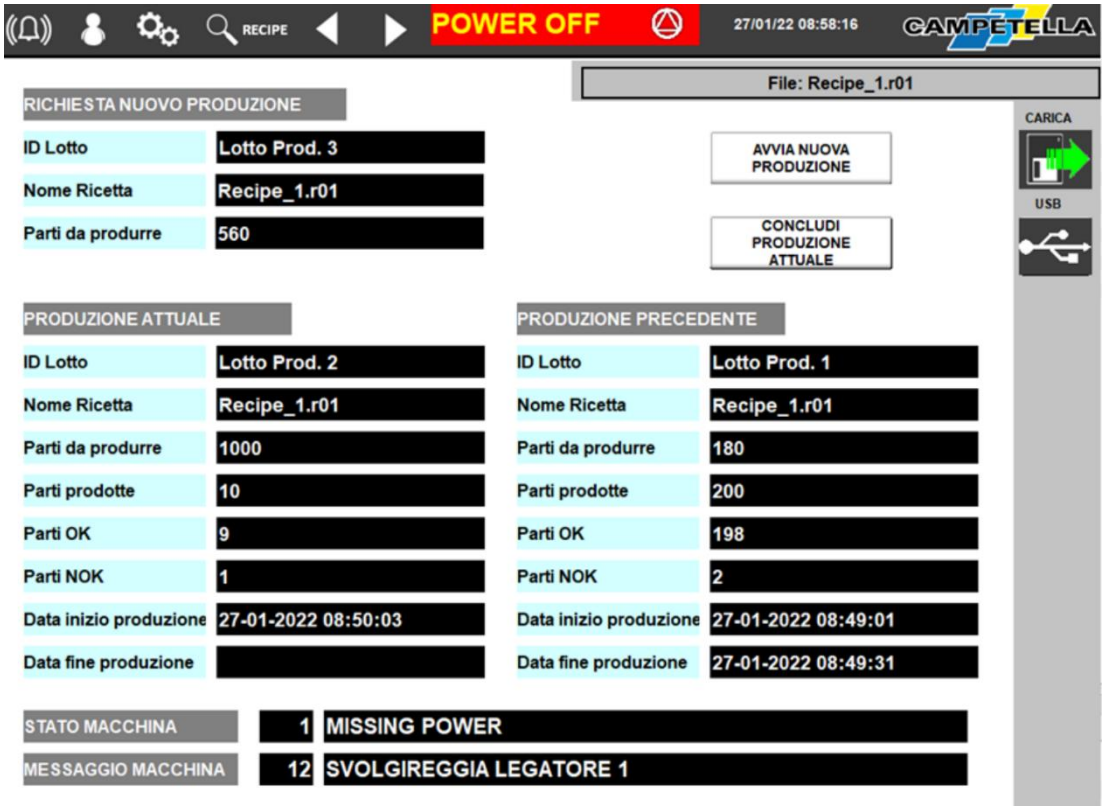


Figura 32 interfaccia grafica visibile dal terminale RHMI a seguito dei test effettuati

Nelle due figure la data di fine produzione era nulla in quanto la produzione attuale risultava ancora in esecuzione, mentre le restanti variabili risultano identiche in entrambe le interfacce grafiche come atteso. In Figura 33 e in Figura 34 è riportato invece un caso in cui la ricetta e l'identificatore del lotto della richiesta di una nuova produzione risultano differenti da quelli della produzione attuale per verificare il lampeggio del pulsante "AVVIA NUOVA PRODUZIONE".

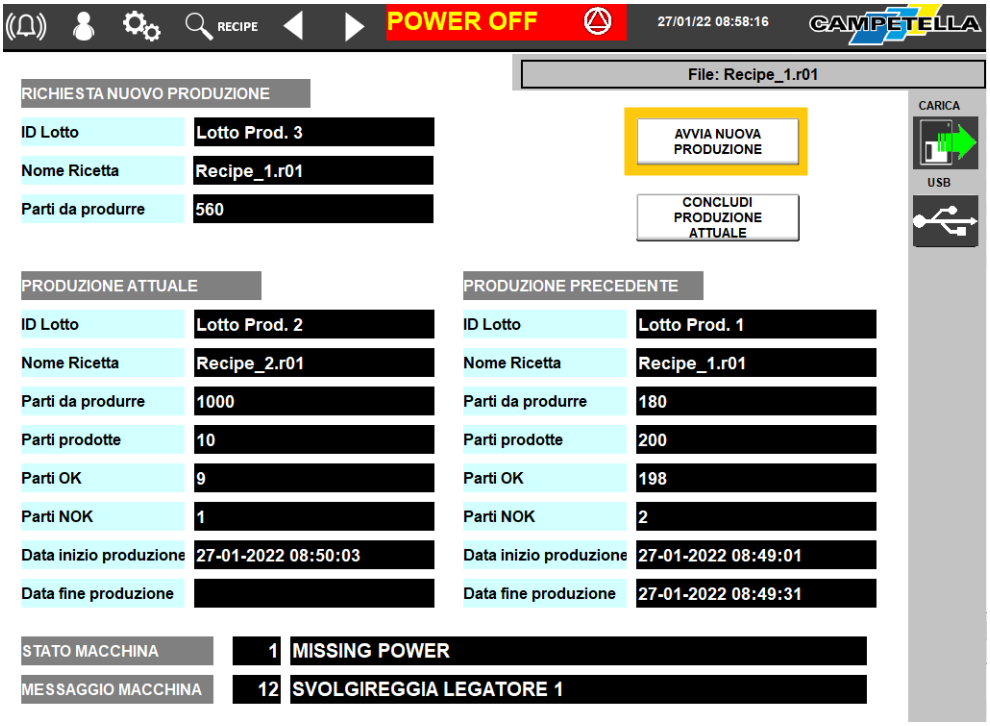


Figura 33: lampeggio giallo del pulsante "AVVIA NUOVA PRODUZIONE"

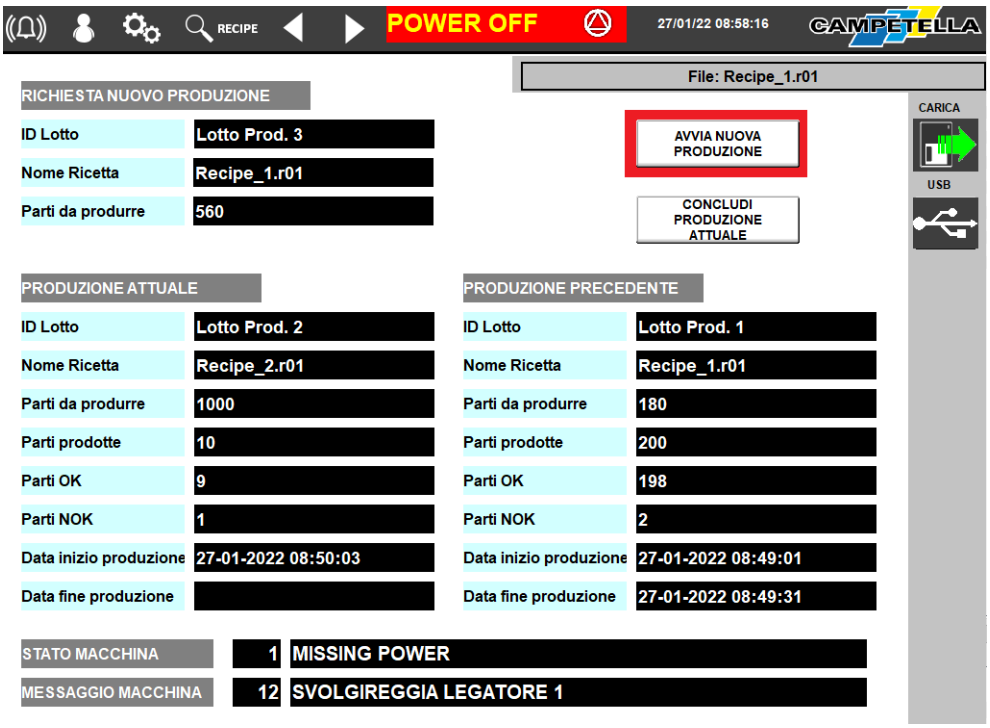


Figura 34: lampeggio rosso del pulsante "AVVIA NUOVA PRODUZIONE"

Per dimostrare infine l'effettiva implementazione della comunicazione OPC-UA è stato utilizzato il software UAExpert, che, come detto in precedenza, rappresenta uno dei Client OPC-UA più impiegati per testare la comunicazione OPC-UA. In Figura 35 è riportata la schermata di accesso ai dati esposti dal Server OPC-UA, nella quale per ogni variabile sono riportate informazioni relative al NodeId, il nome, il valore ed il tipo di dato. Come si nota, tutte le variabili hanno gli stessi valori visibili nelle due interfacce grafiche.

#	Server	Node Id	Display Name	Value	Datatype
1	PremiumHMI	NS2|String|Active_Production\Batch_ID	Batch_ID	Lotto Prod. 2	String
2	PremiumHMI	NS2|String|Active_Production\End_Date	End_Date		String
3	PremiumHMI	NS2|String|Active_Production\NOK_Parts	NOK_Parts	1	Int32
4	PremiumHMI	NS2|String|Active_Production\OK_Parts	OK_Parts	9	Int32
5	PremiumHMI	NS2|String|Active_Production\Parts_To_Be_Produced	Parts_To_Be_Pro...	1000	Int32
6	PremiumHMI	NS2|String|Active_Production\Produced_Parts	Produced_Parts	10	Int32
7	PremiumHMI	NS2|String|Active_Production\Recipe_Name	Recipe_Name	Recipe_2.r01	String
8	PremiumHMI	NS2|String|Active_Production\Start_Date	Start_Date	27-01-2022 08:50:03	String
9	PremiumHMI	NS2|String|Previous_Production\Batch_ID	Batch_ID	Lotto Prod. 1	String
10	PremiumHMI	NS2|String|Previous_Production\End_Date	End_Date	27-01-2022 08:49:31	String
11	PremiumHMI	NS2|String|Previous_Production\NOK_Parts	NOK_Parts	2	Int32
12	PremiumHMI	NS2|String|Previous_Production\OK_Parts	OK_Parts	198	Int32
13	PremiumHMI	NS2|String|Previous_Production\Parts_To_Be_Produced	Parts_To_Be_Pro...	180	Int32
14	PremiumHMI	NS2|String|Previous_Production\Produced_Parts	Produced_Parts	200	Int32
15	PremiumHMI	NS2|String|Previous_Production\Recipe_Name	Recipe_Name	Recipe_1.r01	String
16	PremiumHMI	NS2|String|Previous_Production\Start_Date	Start_Date	27-01-2022 08:49:01	String
17	PremiumHMI	NS2|String|Machine_State\Loaded_Recipe	Loaded_Recipe	Recipe_1.r01	String
18	PremiumHMI	NS2|String|Machine_State\message_ID	message_ID	12	Int32
19	PremiumHMI	NS2|String|Machine_State\message_str	message_str	SVOLGIREGGIA LEGATORE 1	String
20	PremiumHMI	NS2|String|Machine_State\state_ID	state_ID	1	Int32
21	PremiumHMI	NS2|String|Machine_State\state_str	state_str	MISSING POWER	String
22	PremiumHMI	NS2|String|Requested_Production\Batch_ID	Batch_ID	Lotto Prod. 3	String
23	PremiumHMI	NS2|String|Requested_Production\Parts_To_Be_Produced	Parts_To_Be_Pro...	560	Int32
24	PremiumHMI	NS2|String|Requested_Production\Recipe_Name	Recipe_Name	Recipe_1.r01	String

Figura 35: schermata di accesso ai dati del Server OPC-UA dal Client OPC-UA UAExpert

Infine, questi test appena riportati sono stati effettuati a bordo macchina per verificare nuovamente il corretto funzionamento di tutto lo scambio di dati, dal Server OPC-UA al Robox RP2 e viceversa, ottenendo i medesimi risultati.

CAPITOLO 4

CASO APPLICATIVO: ROBOT STANDARD

In questo capitolo dell'elaborato sono esposti gli aspetti implementativi fondamentali riguardanti la configurazione di un Server OPC-UA, agendo direttamente sul controllore della macchina, il Robox RP2, tramite l'utilizzo del software RDE, descritto nel paragrafo 3.1.2. L'ambito all'interno del quale si pone lo sviluppo di questo caso applicativo è quello della realizzazione e ottimizzazione delle macchine industriali di serie, descritte nel capitolo 1, (Robot Standard, quali robot cartesiani, robot laterali e SCARA) di fabbricazione Campetella, in ottica Industria 4.0. A differenza del caso applicativo descritto nel capitolo 3, la configurazione del Server OPC-UA non passa per l'uso del Software Premium HMI, ma viene utilizzato il software UAModeler al fine di realizzare un'Information Model, che possa essere impiegato genericamente in tutte le macchine standard di fabbricazione Campetella, seguendo i concetti fondamentali di OPC-UA espressi nel capitolo 2. Si è scelto di configurare il Server OPC-UA in modo tale da esporre le informazioni relative allo stato macchina, al lotto di produzione attuale e precedente, garantendo allo stesso tempo la possibilità per il cliente di inviare richieste di nuove produzioni da remoto. Pertanto, per semplicità, i dati esposti dal Server OPC-UA sono organizzati seguendo l'organizzazione logica presentata in Tabella 12, Tabella 13, Tabella 14 e Tabella 15. Di conseguenza, anche la definizione dei registri del Robox RP2 da riservare alla comunicazione OPC-UA risulta analoga a quanto presentato per il caso applicativo precedente. Allo stesso modo anche i test effettuati alla fine della configurazione ed i risultati ottenuti da tali test sono identici a quelli mostrati nel paragrafo 3.4. Per tali motivazioni, in questo capitolo non sono discussi nuovamente questi argomenti, ma si pone l'attenzione sull'implementazione del Server OPC-UA tramite il software UAModeler e la sua configurazione sul controllore Robox RP2.

4.1. Strumenti utilizzati

Gli strumenti richiesti per lo sviluppo di questa configurazione sono solamente il controllore Robox RP2 e il software di sviluppo RDE (Robox Development Environment), descritti rispettivamente nel paragrafo 3.1.5 e nel paragrafo 3.1.2. Il software Premium HMI e il router ASEM RK10 non sono impiegati nello sviluppo di questa applicazione in quanto il Server OPC-UA viene eseguito direttamente sul Robox RP2 e la sua configurazione è

effettuata tramite il software RDE; mentre il terminale RHMI ed il software RDT (Robox Development Tool) non sono utilizzati, in quanto nelle macchine standard l'operatore si interfaccia con il robot tramite uno dei due sistemi di controllo (PRIME e EVO) progettati e sviluppati internamente alla Campetella Robotic Center. Di conseguenza, i test effettuati per verificare la corretta configurazione del Server OPC-UA passano univocamente per l'utilizzo del software UAExpert che svolge la funzione di Client OPC-UA. Oltre al controllore Robox RP2, i software RDE e UAExpert, è stato utilizzato il software UAModeler, le cui caratteristiche principali sono presentate nel paragrafo seguente.

4.1.1. UAModeler

UAModeler è un tool software, rilasciato dalla Unified Automation, che consente la definizione di nuovi Information Model passando per la definizione di nuovi tipi di oggetto e tipi di variabile, allo scopo di aggiungere nodi nell'Address Space. Il vantaggio principale nell'utilizzo di questo tool è rappresentato dal fatto che l'utente non deve preoccuparsi di assegnare le Reference corrette ad ogni nuovo nodo, ma è il tool stesso ad occuparsi di ciò. Come visibile in Figura 36, il software presenta le seguenti caratteristiche e funzionalità:

- Un'interfaccia grafica che facilita la creazione di un Information Model personalizzato, aggiungendo nodi e riferimenti all'Address Space in automatico.
- Un generatore di codice C++, ANSI C o .NET da utilizzare nei rispettivi SDK (Software Development Kit), fornite sempre dalla Unified Automation.
- Permette di esportare l'Information Model in formato XML, per poterlo integrare in SDK esterne, come ad esempio la Java SDK (utilizzata nell'applicativo descritto nel capitolo 5) fornita dalla Prosys OPC Ltd, anch'essa membro di OPC Foundation.

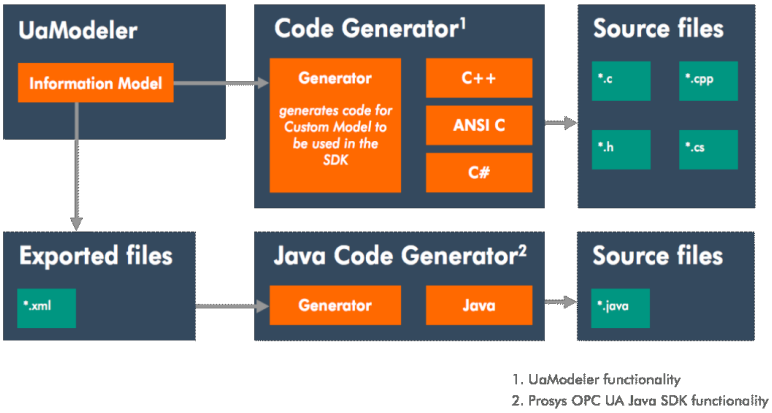


Figura 36: funzionalità del software UAModeler

4.2. Sviluppo

La configurazione del Server OPC-UA, per lo scambio di dati ed informazioni tra le macchine standard di fabbricazione Campetella e che consenta l'accesso da remoto al cliente a tali dati ed informazioni, passa per i seguenti punti:

- Definizione dei tipi di oggetto necessari a realizzare l'Information Model che definisca la struttura logica delle variabili esposte dal Server OPC-UA.
- Integrazione dell'Information Model con il Robox RP2 per associare le variabili esposte dal Server OPC-UA con i rispettivi registri fisici del controllore e configurazione della comunicazione OPC-UA per il controllore Robox RP2.

4.2.1. Implementazione dell'Information Model per il robot standard

La configurazione del Server OPC-UA passa per l'implementazione di un Information Model, tramite l'utilizzo del software UAModeler, che possa essere impiegato genericamente in tutte le macchine standard di fabbricazione Campetella. La struttura del Server OPC-UA permette quindi di esporre dati relativi allo stato e al messaggio del robot, alla produzione attuale, alla produzione precedente e di ricevere indicazioni da remoto relative alla nuova produzione richiesta. Al fine di raggiungere tale obiettivo, la struttura del Server OPC-UA è organizzata seguendo l'organizzazione logica visibile Tabella 12, Tabella 13, Tabella 14 e Tabella 15. Per definire le singole strutture dati, che rappresentano il lotto di produzione richiesto (Requested Batch), il lotto di produzione attuale (Active Production), il lotto di produzione precedente (Previous Production) e lo stato attuale della macchina (Machine State), all'interno dell'Information Model sono stati definiti i seguenti tipi di oggetto:

- RequestedProductionType, per rappresentare la struttura dati che identifica la nuova produzione richiesta (Requested Batch).
- ProductionType, per rappresentare le strutture dati che identificano la produzione attuale e quella precedente (Active Production e Previous Production); essendo le due strutture dati caratterizzate dalle medesime variabili, si è scelto di definire un unico tipo di oggetto per rappresentarle.
- MachineStateType, per rappresentare la struttura dati che identifica lo stato, il messaggio e la ricetta caricata della macchina.

Ad ogni avvio di nuovo progetto, il software UAModeler integra automaticamente l'Information Model di base dello standard OPC-UA, nel quale sono definite tutte le BaseNodeClass necessarie per integrare informazioni relative a nuovi dispositivi da

rappresentare nel Server OPC-UA. Di particolare interesse, è ovviamente la classe BaseObjectType, dalla quale si partirà per aggiungere i nuovi tipi di oggetto. UAModeler mette a disposizione l'apposito comando "Add Subtype" per definire nuovi tipi di oggetto a partire dalla classe BaseObjectType. In Figura 37 è riportato come esempio uno dei nuovi tipi di oggetti definiti.

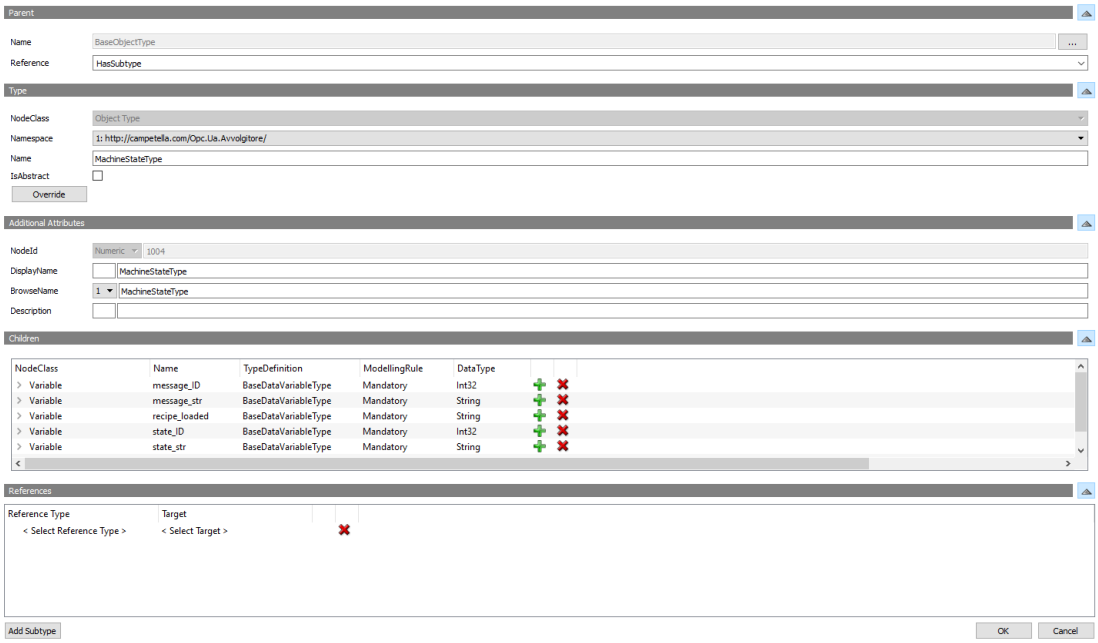


Figura 37: definizione del tipo di oggetto MachineStateType tramite il software UAModeler

In Figura 37 sono visibili informazioni relative al tipo di oggetto appena definito come: il nodo parante e la Reference che lo collega ad esso, il Namespace nel quale il nuovo nodo è definito, il nome, il DisplayName e il BrowseName. Inoltre, sono visibili anche le variabili che caratterizzano il tipo di oggetto e che dunque saranno ereditate da tutti gli oggetti di quel tipo. Le variabili vanno definite manualmente indicandone il nome, la TypeDefinition e il tipo di dato (DataType). Per le variabili che caratterizzano i nuovi tipi di oggetto sono utilizzati tipi di dato definiti nell'Information Model di base di OPC-UA. Oltre a questi tipi di oggetto, è stato definito un'ulteriore tipo di oggetto per rappresentare nell'Information Model la struttura dati complessiva di un robot standard generico. Come mostrato in Figura 38, questo nuovo tipo di oggetto (StandardRobotType):

- È caratterizzato dagli oggetti RequestedProduction, ActiveProduction, PreviousProduction e MachineState, i quali ereditano le variabili dai rispettivi tipi di oggetto.
- Estende la classe BaseObjectType.

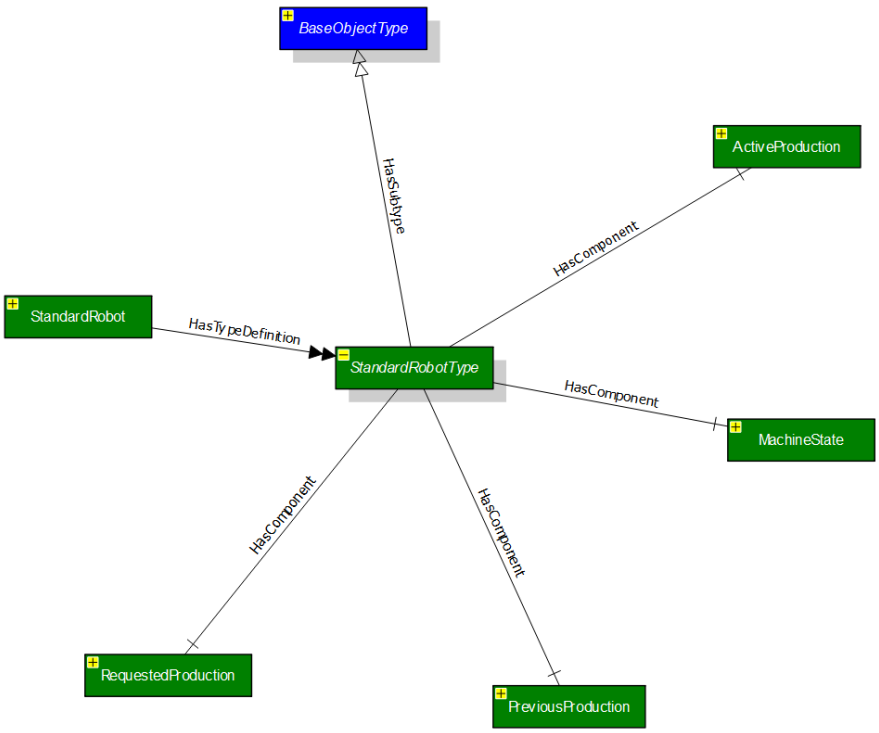


Figura 38: struttura del tipo di oggetto StandardRobotType

A partire da questo tipo di oggetto è istanziato l'oggetto StandardRobot, il quale rappresenta la struttura dati esposta dal Server OPC-UA per una macchina standard. Questo è valido per i casi in cui la macchina riesca a gestire un unico flusso di produzione, ovvero nei casi del robot cartesiano, del robot laterale o dello SCARA; nel caso delle macchine stack-molding, invece, in cui il flusso di produzione è doppio, vanno definiti due oggetti di tipo StandardRobotType per raddoppiare la struttura dati esposta dal Server OPC-UA. L'information Model così realizzato deve essere esportato in formato XML per poterlo integrare nella configurazione del controllore Robox RP2 come descritto nel paragrafo seguente.

4.2.2. Integrazione con il software RDE

Il software di sviluppo RDE mette a disposizione nativamente un servizio che offre la possibilità di integrare il descrittore del Server OPC-UA in formato XML. Per poter integrare tale servizio nel controllore Robox RP2 è necessario abilitare la generazione dei file di configurazione del Server OPC-UA tra le impostazioni del controllore stesso, indicando il nome scelto per il Server, la porta che identifica la socket (Indirizzo IP: porta) che permette la comunicazione con il Server e il percorso dell'Information Model che si desidera integrare nel Server. A seguito di ciò, è possibile mappare i campi esposti dal Server OPC-UA con i

rispettivi registri fisici del Robox RP2, come mostrato in Figura 39, al fine di “popolare” il Server OPC-UA.

Nodo	Tipo	Mappatura
Oggetti		
1:Machines		
1:StandardRobot		
1:ActiveProduction		
1:Produced_Parts	Int32	nvr(2401)
1:OK_Parts	Int32	nvr(2402)
1:NOK_Parts	Int32	nvr(2403)
1:Parts_To_Be_Produ...	Int32	nvr(2404)
1:Batch_ID	String	nvsr(1)
1:Recipe_Name	String	nvsr(2)
1:Start_Date	String	nvsr(3)
1:End_Date	String	nvsr(4)
1:MachineState		
1:state_ID	Int32	r(2402)
1:message_ID	Int32	r(2403)
1:state_str	String	sr(3)
1:message_str	String	sr(4)
1:recipe_loaded	String	sr(5)
1:PreviousProduction		
1:Produced_Parts	Int32	nvr(2405)
1:OK_Parts	Int32	nvr(2406)
1:NOK_Parts	Int32	nvr(2407)
1:Parts_To_Be_Produ...	Int32	nvr(2408)
1:Batch_ID	String	nvsr(5)
1:Recipe_Name	String	nvsr(6)
1:Start_Date	String	nvsr(7)
1:End_Date	String	nvsr(8)
1:RequestedProduction		
1:Parts_To_Be_Produ...	Int32	r(2401)
1:Batch_ID	String	sr(1)
1:Recipe_Name	String	sr(2)

Figura 39: associazione dei dati definiti tramite il Software UAModeler con i registri del controllore Robox RP2

Il Server OPC-UA può essere contatto sia tramite LAN che tramite WAN, che devono quindi essere configurate in modo opportuno. In particolare, sia l’interfaccia LAN che la WAN devono riuscire a raggiungere l’indirizzo IP presso il quale è esposto il Server OPC-UA. È possibile accedere al Server OPC-UA indicando nel Client la socket definita in fase di configurazione. La comunicazione OPC-UA realizzata come appena descritto, risulta verificabile solamente testandola con il software UAExpert, dato che non è stata realizzata un’interfaccia di accesso al Server, differentemente a quanto effettuato nell’applicativo descritto nel capitolo precedente. Si ricorda che, per semplicità, non sono stati riportati i test eseguiti in quanto i risultati ottenuti sono analoghi a quelli riportati per il caso della macchina “custom”.

CAPITOLO 5

CASO APPLICATIVO: SIMULATORE MACCHINA IMM

In questo capitolo dell'elaborato sono descritti gli aspetti relativi alla progettazione ed implementazione di un programma in linguaggio Java, comprensivo di interfaccia grafica, per simulare il funzionamento di una macchina per lo stampaggio ad iniezione di materiale plastico, macchina IMM (Injection Moulding Machine). Come già raccontato precedentemente nel capitolo 1, il core business della Campetella Robotic Center è rappresentato dalla realizzazione di soluzioni robotizzate per il settore dello stampaggio delle materie plastiche. In particolare, i robot cartesiani e i robot laterali di fabbricazione Campetella sono spesso impiegati come robot di asservimento di macchine IMM e richiedono pertanto una fase di collaudo che permetta di simulare il funzionamento dei robot in questo settore applicativo. A tale scopo, la Campetella Robotic Center ha progettato e sviluppato negli anni un proprio dispositivo, visibile in Figura 40, che una volta collegato al robot di interesse, simuli il funzionamento di una macchina IMM. Lo sviluppo di questo dispositivo è basato sullo standard EUROMAP 67, nel quale sono definite le specifiche da seguire per favorire lo scambio di segnali tra la macchina IMM e il suo robot di asservimento.

Figura 40: simulatore del funzionamento di una macchina IMM basato su specifiche EUROMAP 67

L'obiettivo dello sviluppo di questa applicazione è quello di aggiornare il dispositivo sia da un punto di vista fisico sia dal punto di vista tecnologico. Il nuovo simulatore è infatti basato sullo standard EUROMAP 79 il quale, come descritto nel paragrafo 2.8, definisce le specifiche da seguire per implementare lo scambio di segnali e informazioni, utilizzando lo standard di comunicazione OPC-UA tra una macchina IMM e il suo robot di asservimento. La realizzazione del simulatore della macchina IMM passa per lo sviluppo di un'interfaccia grafica che replichi la logica e le funzionalità del dispositivo già disponibile, implementando uno scambio di dati basato sullo standard OPC-UA in un'architettura Client-Server. Al fine di garantire lo scambio di dati tra il simulatore della macchina IMM e il suo robot di asservimento, rispettando allo stesso tempo le specifiche EUROMAP 79, l'architettura di comunicazione è realizzata in modo tale che il simulatore della macchina IMM svolga la funzione di Server OPC-UA e il robot di asservimento svolga la funzione di Client OPC-UA, sebbene tale configurazione risulti opzionale all'interno delle specifiche EUROMAP 79, come visibile in Figura 17. L'integrazione dello standard EUROMAP 79 nell'applicazione sviluppata in Java è effettuata caricando l'Information Model corrispondente allo standard stesso, all'interno del quale sono definiti tutti i tipi di oggetto e tipi di variabile che caratterizzano l'Address Space dell'EUROMAP 79. Infine, nel presente capitolo sono presentati i test effettuati per dimostrare la corretta configurazione della logica di funzionamento del simulatore della macchina IMM e della comunicazione OPC-UA, utilizzando il software UAExpert.

5.1. Strumenti utilizzati

Per lo sviluppo di questa applicazioni è richiesto l'utilizzo del linguaggio di programmazione Java. L'ambiente di sviluppo più diffuso ad oggi che permette di scrivere codice in Java è Eclipse. Eclipse è un ambiente di sviluppo open-source, sviluppato dalla Eclipse Foundation, multi-linguaggio e multipiattaforma, che può essere utilizzato per la produzione di software di vario genere. Si passa infatti da un completo IDE per il linguaggio Java ad un ambiente di sviluppo per il linguaggio C++, fino alla possibilità di progettare graficamente una GUI (Graphical User Interface). Come già descritto, l'applicazione da realizzare richiede la necessità di simulare il funzionamento di una macchina IMM, agendo su un'apposita interfaccia grafica, e di configurare un Server OPC-UA che permetta all'applicativo di esporre dati ed informazioni relative alla simulazione stessa. La libreria ufficiale che permette la realizzazione di interfacce grafiche in linguaggio Java è la libreria Swing. Swing è un framework per Java, appartenente alle Java Foundation Classes (JFC) e

orientato allo sviluppo di interfacce grafiche. Alcune delle classi, disponibili per il framework Swing, sono implementazioni di widget (oggetti grafici) come caselle di testo, pulsanti, pannelli e tabelle. Per la configurazione del Server e della comunicazione OPC-UA, invece, si è scelto di adottare la Java SDK (Software Development Kit) fornita dalla Prosys OPC Ltd, una delle aziende membro dell'associazione OPC Foundation, che rappresenta una delle librerie ufficiali segnalate dalla OPC Foundation per lo sviluppo di applicazioni in Java, basate sullo standard di comunicazione OPC-UA. Le principali caratteristiche e funzionalità di questa SDK sono riportate e descritte nel paragrafo seguente.

5.1.1. OPC-UA Java SDK

La Prosys Java SDK rappresenta una delle scelte più adottate per lo sviluppo di applicazioni Client e Server basate sulla comunicazione OPC-UA. La sua interfaccia di programmazione ad alto livello consente uno sviluppo rapido delle applicazioni, accelerando il processo di implementazione. La Prosys Java SDK fornisce le librerie che espongono e gestiscono autonomamente tutte le funzionalità di base dello standard OPC-UA e la documentazione che descrive gli strumenti disponibili per la SDK stessa. Le librerie fornite possono essere suddivise in due categorie:

- OPC-UA Client SDK, caratterizzata dalle librerie che espongono le funzionalità necessarie allo sviluppo di applicazioni Client per la comunicazione OPC-UA, tra cui la gestione della connessione ed attivazione della sessione tra Client e Server, dell'accesso e dell'esplorazione dell'Address Space e dei certificati di sicurezza.
- OPC-UA Server SDK, formata dalle librerie che espongono le funzionalità necessarie allo sviluppo di applicazioni Server per la comunicazione OPC-UA, tra cui la gestione dei nodi che compongono l'Address Space, l'integrazione degli Information Model nell'Address Space, la gestione dei certificati di sicurezza e dei servizi, come le chiamate dei metodi e l'accesso allo storico dei dati.

Entrambe le categorie appena citate inoltre, rendono disponibili alcuni esempi di applicazioni già implementate al fine di mostrare le funzionalità di base delle rispettive librerie. Di interesse per lo sviluppo del simulatore della macchina IMM è sicuramente la OPC-UA Server SDK e l'esempio applicativo che ne mostra le funzionalità. Come descritto di seguito, questo esempio fornisce il punto di partenza e di riferimento per lo sviluppo del Server OPC-UA realizzato per esporre i dati del simulatore della macchina IMM. Un altro

strumento importante della SDK è il generatore di codice, il quale è in grado di creare automaticamente classi Java, a partire da un Information Model in formato XML. Queste classi generate rendono disponibili i metodi di accesso ai nodi, e ai loro attributi, che sono definiti all'interno dell'Information Model. Sfortunatamente, il generatore di codice presente nella versione della SDK utilizzata, non risulta compatibile con l'Information Model dell'EUROMAP 79. Di seguito, nel paragrafo 5.2.1, dedicato alla descrizione della configurazione del Server OPC-UA a partire dalla Java SDK, è descritto il metodo utilizzato per risolvere tale problematica e definire le classi Java che permettono di accedere alla struttura dell'Information Model dell'EUROMAP 79. Tale strumento, invece, risulta particolarmente efficace ed efficiente in tutti quei casi in cui si sviluppa un nuovo Information Model personalizzato, partendo solamente dall'Information Model di base dello standard OPC-UA, tramite il software UAModeler. Infine, nella SDK è presente tutta la documentazione necessaria a descrivere:

- L'integrazione delle librerie all'interno di un nuovo progetto.
- Tutte le funzionalità esposte dalle librerie riservate rispettivamente allo sviluppo di applicazioni Client OPC-UA e/o Server OPC-UA.
- Il funzionamento del generatore di codice.

5.2. Sviluppo

Il primo passo per lo sviluppo dell'applicazione, oggetto di questo capitolo, è quello di configurare il Server OPC-UA che ha il compito di esporre le informazioni relative alla simulazione stessa al robot standard di fabbricazione Campetella da collaudare che, invece, svolge la funzione di Client OPC-UA. Più nello specifico, la configurazione del Server OPC-UA è realizzata a partire dall'esempio applicativo presente nella Java SDK rilasciata dalla Prosys, cercando di adattare quanto già presente alle necessità della nuova applicazione. Nel paragrafo seguente sono descritte le funzionalità di base già implementate nell'esempio di partenza e che sono state mantenute per lo sviluppo del simulatore e, parallelamente, anche le modifiche apportate per adeguare l'esempio di partenza alle esigenze del nuovo simulatore. Successivamente, è stata implementata l'interfaccia grafica che permette di replicare la logica di funzionamento del dispositivo da aggiornare, utilizzando la libreria Swing. Infine, allo scopo di simulare il funzionamento di una macchina IMM, è stata implementata una classe Java nella quale sono definiti nuovi metodi necessari a popolare la struttura che descrive la macchina IMM nel Server OPC-UA, in base alle azioni performate da un operatore sull'interfaccia grafica.

5.2.1. Configurazione del Server OPC-UA

Come già anticipato, la Java Prosys SDK mette a disposizione degli esempi già implementati per mostrare l'utilizzo delle librerie OPC-UA. In particolare, per lo sviluppo dell'applicativo si è partiti dall'esempio relativo all'implementazione di un Server OPC-UA. Pertanto, di seguito sono descritte le funzionalità principali implementate dalle librerie OPC-UA Server, contenute nella SDK, che sono di utilità per la configurazione di un Server OPC-UA che esponga dati ed informazioni relativi ad una macchina IMM, come definito nelle specifiche EUROMAP 79. Parallelamente a ciò, sono quindi introdotte anche le modifiche apportate a tale esempio di partenza. Le principali funzionalità di interesse implementate dalle librerie OPC-UA Server riguardano:

- La definizione di un Server OPC-UA completo di tutte le sue funzionalità.
- La definizione degli Endpoint che consentono il collegamento al Server OPC-UA.
- La definizione di un gestore dei nodi (NodeManager) che compongono l'Address Space del Server OPC-UA.
- L'integrazione di un Information Model all'interno dell'Address Space del Server OPC-UA.
- L'avvio e l'arresto della comunicazione OPC-UA.

Per definire un Server OPC-UA, le librerie della SDK espongono la classe "UaServer" che permette di accedere a tutte le altre funzionalità implementate nelle librerie stessa. La classe UaServer rappresenta dunque il punto di collegamento tra le librerie OPC-UA e l'applicativo che ne intende far uso. Pertanto, è necessario definire un oggetto della classe UaServer come riportato di seguito.

```
protected UaServer server;

// *** Create the server
        server = new UaServer();
```

In seguito a ciò, sono definiti gli Endpoint che consentono l'accesso al Server OPC-UA. Gli Endpoint di un Server OPC-UA sono i punti di accesso ai quali le applicazioni Client devono collegarsi se desiderano connettersi al Server OPC-UA. In questo caso, gli Endpoint sono identificati da un URL (Uniform Resource Locator) del tipo $Protocol://[HostName]:Port/ServerName$. Si ricorda che la label "HostName" viene sostituita automaticamente con l'indirizzo IP dell'host su cui viene eseguita l'applicazione. Le librerie della SDK permettono di definire gli Endpoint per due diversi protocolli di accesso: il protocollo "OpcTcp" e il

protocollo “OpcHttps” che rappresentano rispettivamente l’utilizzo dello standard OPC-UA sul protocollo TCP (Transfert Control Protocol) e sul protocollo HTTPS (HyperText Transfer Protocol Secure). La classe UaServer espone i metodi “setPort” e “setServerName” che consentono di settare la porta che identifica la socket (Indirizzo IP: porta) dei rispettivi protocolli e di settare il nome dell’applicazione. Il codice che implementa quanto descritto è riportato di seguito.

```
int tcpPort = 52520;
int httpsPort = 52443;
String applicationName = "SampleConsoleServer"

// *** Server Endpoints
// TCP Port number for the UA TCP protocol
server.setPort(Protocol.OpcTcp, TcpPort);
// TCP Port for the HTTPS protocol
server.setPort(Protocol.OpcHttps, httpsPort);

// optional server name (default for all protocols)
server.setServerName("OPCUA/" + applicationName);
```

Questa parte appena presentata non è stata modificata rispetto all’esempio iniziale di partenza. A questo punto, si è passato alla configurazione di quelli che sono gli aspetti principali di un Server OPC-UA, ossia l’Address Space che espone i dati del Server e il gestore dei nodi (NodeManager) che permette di operare su di esso. La classe UaServer espone di default l’Address Space definito dall’Information Model di base dello standard OPC-UA, nel quale, si ricorda, sono contenute le definizioni di tutte le BaseNodeClasses che caratterizzano lo standard. Allo scopo di poter aggiungere e modificare nodi e di esplorare tale Address Space, la Prosys Java SDK mette a disposizione del programmatore la possibilità di definire un gestore dei nodi (NodeManager). In particolare, per definire un gestore dei nodi va istanziato un oggetto della classe “MyNodeManager”, come visibile di seguito.

```
protected MyNodeManager myNodeManager;

// My Node Manager
    myNodeManager = new MyNodeManager(server,
MyNodeManager.NAMESPACE);
    myNodeManager.addListener(myNodeManagerListener);
```

Il gestore dei nodi così definito permette, come anticipato, di aggiungere nodi all’Address Space e di esplorarlo. Per aggiungere nodi ed ampliare l’Address Space possono essere seguite due strade differenti:

- Aggiungere manualmente un nodo all'Address Space, indicando il tipo di nodo, il NodeId e le Reference che lo collegano con altri nodi che compongono l'Address Space.
- Caricando l'Information Model che definisce la struttura dell'Address Space.

Il primo metodo risulta sicuramente più meccanico e più problematico per il programmatore, mentre il secondo metodo, grazie anche alla possibilità di definire nuovi Information Model partendo da zero tramite il software UAModeler, risulta molto meno oneroso per il programmatore. Per integrare un Information Model all'interno dell'Address-Space OPC-UA è necessario essere, innanzitutto, in possesso del descrittore XML dell'Information Model e, successivamente, richiamare l'apposito metodo "loadModel" disponibile per il Server precedentemente definito. La parte di codice che implementa ciò è riportata di seguito.

```
    // Load the type nodes from the SampleTypes.xml file.
    server.getAddressSpace().loadModel(new
File("SampleTypes.xml").toURI());
```

Questa parte di codice appena riportata dall'esempio di partenza, è stata modificata per essere adattata a quelle che sono le esigenze del caso applicativo che si sta sviluppando. In particolare, si desidera importare all'interno dell'Address Space l'Information Model che definisce le specifiche EUROMAP 79, allo scopo di poter accedere al tipo di oggetto IMMToRobotType, presentato nel paragrafo 2.8.1, ed utilizzarlo per esporre i dati relativi alla simulazione del funzionamento della macchina IMM. Inoltre, come specificato all'interno del documento EUROMAP 79 stesso, alcune definizioni interne all'EUROMAP 79 fanno riferimento ad altri Information Model. Un esempio su tutti è rappresentato dalla variabile CycleQuality all'interno della definizione del tipo di oggetto IMMPartQualityType. La definizione del tipo di variabile "CycleQualityEnumeration", infatti, è contenuta nell'Information Model delle specifiche EUROMAP 83. In Tabella 19 sono riportati tutti gli Information Model necessari per poter importare quello dell'EUROMAP 79 all'interno dell'Address Space.

Information Model	Riferimento
http://opcfoundation.org/UA/	0:NodeVersion
http://opcfoundation.org/UA/DI/	2:DeviceSet
http://opcfoundation.org/UA/PlasticsRubber/GeneralTypes/	3:CycleQualityEnumeration
http://opcfoundation.org/UA/Machinery	4:MachineIdentificationType

Tabella 19: elenco degli Information Model necessari al caricamento dello specifiche EUROMAP 79

L'Information Model identificato dall'URI "http://opcfoundation.org/UA/" contiene le definizioni di base dello standard OPC-UA e, pertanto, risulta già caricato nell'Address Space del Server, come già detto precedentemente. Gli altri Information Model, invece, sono tutti implementati e resi disponibili dalla OPC Foundation. Il codice a seguito di tali modifiche si presenta dunque come riportato di seguito.

```
//Load Information Model
    server.getAddressSpace().loadModel(new
File("Opc.Ua.Di.NodeSet2.xml").toURI());
    server.getAddressSpace().loadModel(new
File("Opc.Ua.PlasticsRubber.GeneralTypes.NodeSet2.xml").toURI()
);
    server.getAddressSpace().loadModel(new
File("Opc.Ua.Machinery.NodeSet2.xml").toURI());
    server.getAddressSpace().loadModel(new
File("Opc.Ua.PlasticsRubber.IMM2Robot.xml").toURI());
```

Oltre a caricare le definizioni dei tipi e delle istanze da un Information Model, come appena descritto, è necessario poter accedere e poter modificare quanto importato all'interno dell'Address Space. A tale scopo, vanno definite le classi Java che rendono disponibili i metodi di accesso ai nodi, e ai loro attributi, definiti all'interno dell'Information Model. In particolare, queste classi vanno implementate per ogni tipo di oggetto definito all'interno dell'Information Model. Come accennato precedentemente, la SDK rende disponibile un generatore di codice di queste classi, ma questo strumento non risulta compatibile con l'Information Model dell'EUROMAP 79. La SDK fornisce anche alcuni Information Model compatibili con il generatore di codice; perciò, per risolvere tale problematica si è partiti dalle classi Java generate a partire da uno di questi Information Model. Al fine di illustrare le caratteristiche di queste classi Java si fa riferimento ad un caso generico in cui vengono descritte le classi Java generate per il tipo di oggetto "MyObjectType". In pratica il generatore di codice genera tre differenti classi per ogni tipo di oggetto dell'Information Model:

- Una classe di tipo "interface", nominata MyObjectType, che contiene le dichiarazioni delle costanti e dei metodi di utilità per le altre due.
- Una classe, nominata MyObjectTypeNodeBase, che implementa i metodi per accedere ai nodi che compongono la struttura del tipo di oggetto, di accedere all'attributo Value e di modificare l'attributo "Value".

- Una classe, nominata MyObjectTypeNode, che estende le funzionalità della classe MyObjectTypeNodeBase per integrare le chiamate dei metodi (non implementate in questo esempio).

Per semplicità, di seguito sono riportate solamente le definizioni dei metodi implementati per la classe MyObjectTypeNodeBase. Tuttavia, prima di descrivere la definizione di questi metodi, va evidenziata la differenza tra un nodo che rappresenta un'istanza di un tipo di oggetto e una variabile. Nel primo caso infatti sarà definito solo il metodo che permette di accedere al nodo di quel tipo di oggetto, mentre nel secondo caso sono definiti anche i metodi di accesso e modifica dell'attributo "Value". Inoltre, il metodo di accesso al nodo sarà differente tra i due casi appena citati. Il metodo di accesso ad un nodo che identifica un'istanza di un tipo di oggetto generico contenuto nella struttura del tipo di oggetto MyObjectType è definito come segue.

```
public NewObjectTypeNode getNewObjectNode() {
QualifiedName browseName = getQualifiedName("NamespaceURI",
"NewObject");
      return (NewObjectTypeNode) getComponent(browseName);
}
```

Invece, nel caso in cui è necessario accedere ad un nodo che identifica una variabile generica del tipo di oggetto MyObjectType, si utilizza il metodo seguente.

```
public BaseDataVariableNode getMyVariableNode() {
QualifiedName browseName = getQualifiedName("NamespaceURI",
"MyVariable");
      return (BaseDataVariableNode) getComponent(browseName);
}
```

Per il secondo caso, in cui si accede ad un nodo che identifica una variabile generica, sono definiti altri due metodi che permettono rispettivamente di accedere all'attributo "Value" e di modificare l'attributo "Value". Entrambi i metodi sono riportati di seguito.

```
public Float getMyVariable() {
      UaVariable node = getMyVariableNode();
      if(node == null) {
            return null;
      }
      Float value =
(Float)node.getValue().getValue().getValue();
      return value;
}
```

```
public void setMyVariable(Float value) {
      UaVariable node = getMyVariableNode();
      if (node == null) {
            throw new RuntimeException("Mandatory node Position
does not exist)");
      }
      try {
            node.setValue(value);
      } catch(StatusException e) {
            throw new RuntimeException("Setting MyVariable
failed unexpectedly", e);
      }
}
```

Queste tre classi, ed i relativi metodi, sono state definite a partire da quanto appena descritto per tutti i tipi di oggetto che caratterizzano l'interfaccia OPC-UA, mostrata in Figura 18, che definisce la struttura dell'Address Space nel caso in cui la macchina IMM svolge la funzione di Server OPC-UA. Più nello specifico sono state definite le classi:

- IMMToRobotObjectTypeNodeBase, IMMToRobotObjectTypeNode e IMMToRobotObjectType per rappresentare il tipo di oggetto IMMToRobotType.
- MouldObjectTypeNodeBase, MouldObjectTypeNode e MouldObjectType per rappresentare il tipo di oggetto MouldObjectType.
- AxisObjectTypeNodeBase, AxisObjectTypeNode e AxisObjectType per rappresentare il tipo di oggetto AxisObjectType.
- IMMPartTrackingObjectTypeNodeBase, IMMPartTrackingObjectTypeNode e IMMPartTrackingObjectType per rappresentare il tipo di oggetto IMMPartTrackingObjectType.
- IMMPartQualityObjectTypeNodeBase, IMMPartQualityObjectTypeNode e IMMPartQualityObjectType per rappresentare il tipo di oggetto IMMPartQualityObjectType.

Per poter utilizzare i metodi di queste classi, le librerie della SDK permettono, a partire dal gestore dei nodi, di creare un'istanza di un tipo di oggetto definito all'interno di un Information Model. Tornado al caso generico, il codice che permette di definire un'istanza per il tipo di oggetto MyObjectType è presentato di seguito.

```
//Create an instance of MyObjectType
MyObjectType myVar =
myNodeManager.createInstance(MyObjectTypeNode.class,
"MyVariable");
//Call setMyVariable method
myVar.setMyVariable((float)160.5);
```

Questa parte di codice appena riportata è stata adattata a quelle che sono le esigenze del caso applicativo che si sta sviluppando. Quindi è stata definita un'istanza del tipo di oggetto IMMToRobotType. Per rispettare la struttura definita dalle specifiche EUROMAP 79, l'istanza del tipo di oggetto IMMToRobotType è stata inserita manualmente nell'Address Space tramite l'apposito metodo "addNodeAndReference" definito nella classe MyNodeManager disponibile dalle librerie della SDK. Il codice che implementa quanto appena descritto è riportato di seguito.

```
  protected void createInstances() throws StatusException {
        machinesFolder = (UaObject)
getServer().getNodeManagerRoot().getNode(new NodeId(4, 1001));
        imm1 =
myNodeManager.createInstance(IMMToRobotObjectTypeNode.class,
"IMMToRobot_1");
        myNodeManager.addNodeAndReference(machinesFolder, imm1,
Identifiers.Organizes);
  }
```

Oltre a tutte queste funzionalità descritte fino a questo punto, la SDK rende disponibili i metodi che permettono di avviare e terminare la comunicazione OPC-UA. Durante l'intervallo tra l'avvio e la terminazione della conclusione i Client OPC-UA possono accedere al Server OPC-UA passando per gli Endpoint definiti precedentemente. Infine, le librerie mettono a disposizione metodi per la gestione dei certificati, dell'autenticazione e dell'autorizzazione delle applicazioni Client OPC-UA che intendono accedere al Server. Tuttavia, queste tematiche non sono state affrontate durante lo sviluppo di questo applicativo, in quanto l'obiettivo era quello di configurare le funzionalità di base di un Server OPC-UA.

5.2.2. Implementazione dell'interfaccia grafica

Come già accennato, per implementare l'interfaccia grafica è stata utilizzata la libreria Swing. Swing fornisce gli strumenti necessari alla realizzazione di interfacce grafiche in linguaggio Java. Oltre a rendere disponibili le classi Java per integrare widget, come ad esempio caselle di testo, pulsanti, pannelli, slider e liste, Swing offre uno strumento grafico che permette di progettare in modo molto intuitivo e semplice le interfacce grafiche. Più nello specifico, il programmatore può scegliere tra un insieme di widget da aggiungere all'interfaccia grafica preoccupandosi solamente di progettare il design dell'interfaccia grafica. È poi la libreria Swing che si occupa di generare automaticamente il codice Java che implementa quanto progettato agendo sul tool grafico. L'interfaccia grafica realizzata per replicare il dispositivo che deve essere aggiornato si presenta come visibile in Figura 41.

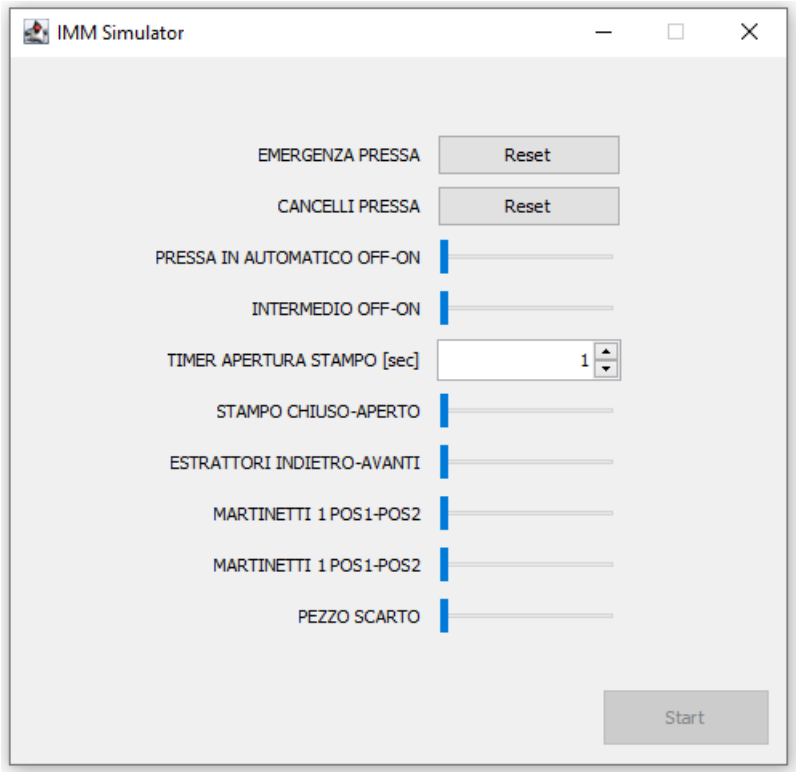


Figura 41: interfaccia grafica configurata per il nuovo simulatore

L'interfaccia grafica è stata progettata in modo da replicare il più possibile l'organizzazione logica del dispositivo precedente. Per replicare gli switch del dispositivo visibile in Figura 40, è stato utilizzato il widget "JSlider" in modo tale da poter passare dallo stato logico "OFF" (valore pari a zero) allo stato logico "ON" (valore pari a uno). Per gli slider che rappresentano gli oggetti stampo, estrattori e martinetti 1/2, il valore logico "OFF" è mappato rispettivamente con "CHIUSO", "INDIETRO" e "POS1"; mentre il valore logico "ON" è mappato rispettivamente con "APERTO", "AVANTI" e "POS2". Gli switch relativi agli allarmi "EMERGENZA PRESSA" e "CANCELLI PRESSA" del vecchio dispositivo, servono solamente a resettare i rispettivi allarmi che sono gestiti elettromeccanicamente; pertanto, nell'interfaccia grafica sono stati inseriti due buttoni che permettano di resettare tali allarmi, sebbene come si vedrà di seguito, le funzioni di reset non sono state gestite. Rispetto alla logica organizzativa del vecchio dispositivo è stato aggiunto un oggetto di tipo "JSpinner" per consentire all'operatore di settare il tempo di apertura e chiusura dello stampo della macchina IMM e per gestire la simulazione delle posizioni intermedie dello stampo. Tutti questi oggetti, inizialmente, sono disabilitati per indicare che non è possibile interagire con loro in quanto il Server OPC-UA non è stato inizializzato e pertanto non è pronto ad esporre dati e informazioni. Questi oggetti sono abilitati solo dopo che il Server OPC-UA è stato inizializzato. A tale scopo, è stato inserito il pulsante "Start" che ha la funzione di inizializzare il Server OPC-UA come descritto precedentemente e di abilitare la comunicazione OPC-UA

stessa. Il pulsante “Start” ha anche il compito di aprire una nuova finestra di dialogo per segnalare all’operatore l’inizializzazione del Server. La schermata della finestra di dialogo è visibile in Figura 42.

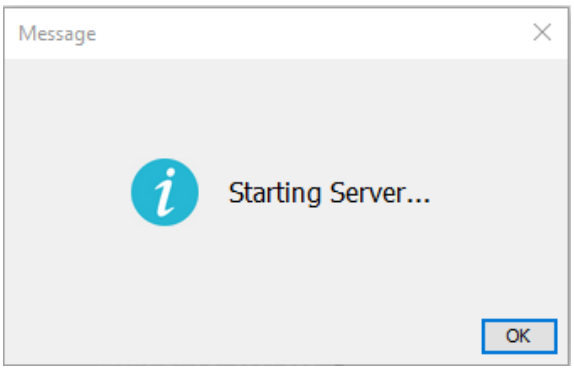


Figura 42: schermata di messaggio di inizializzazione del Server OPC-UA

Quando questa finestra di dialogo è mostrata sullo schermo l’operatore non può interagire con gli oggetti della schermata principale finché non viene premuto il pulsante “OK”. Quest’ultimo viene abilitato solamente quando termina l’inizializzazione del Server OPC-UA; in questo modo, si può tornare alla schermata principale solo quando è terminata l’inizializzazione del Server OPC-UA. Quanto appena descritto è stato implementato per evitare che l’operatore possa interagire nuovamente con il pulsante “START”, andando a generare eccezioni nel codice durante l’inizializzazione del Server OPC-UA. La libreria Swing implementa automaticamente anche i gestori degli eventi che si intendono gestire per gli oggetti presenti nell’interfaccia. Per l’applicativo che si sta descrivendo sono stati implementati i gestori per i seguenti eventi:

- La chiusura della finestra principale, allo scopo di gestire la terminazione del Server OPC-UA.
- La pressione del pulsante “Start”, per gestire l’apertura della finestra di dialogo.
- Il rilascio del pulsante “Start”, per gestire l’avvio del Server OPC-UA e abilitare tutti gli oggetti della schermata principale.
- Il cambio di stato dello slider “PRESSA IN AUTOMATICO”, per gestire l’esecuzione in automatico della simulazione del ciclo della macchina IMM.
- Il cambio di stato dello slider “STAMPO CHIUSO-APERTO”, per gestire l’aggiornamento dello stato dell’oggetto che rappresenta lo stampo all’interno del Server OPC-UA.
- Il cambio di stato dello slider “ESTRATTORI INDIETRO-AVANTI”, per gestire l’aggiornamento dello stato dell’oggetto che rappresenta gli estrattori all’interno del Server OPC-UA.

- Il cambio di stato degli slider “MARTINETTI 1 POS1-POS2” e “MARTINETTI 2 POS1-POS2”, per gestire l’aggiornamento dello stato degli oggetti che rappresentano rispettivamente i due martinetti.
- Il cambio di stato dello slider “PEZZO SCARTO”, per simulare il verificarsi di problematiche che possono insorgere durante il normale ciclo di produzione della macchina IMM e segnalare al robot di asservimento di contrassegnare il pezzo appena prodotto come pezzo difettoso e dunque da scartare.

I gestori di questi eventi hanno quindi lo scopo di aggiornare i dati esposti dal Server OPC-UA per simulare il funzionamento della macchina IMM a seguito delle azioni effettuate dall’operatore sull’interfaccia appena descritta. I gestori vanno quindi a richiamare i metodi disponibili per l’istanza del tipo di oggetto IMMToRobotType, facendo da tramite tra l’interfaccia grafica ed il Server OPC-UA. Come già accennato precedentemente, non sono stati definiti i gestori degli eventi sui pulsanti di reset per gli allarmi “EMERGENZA PRESSA” e “CANCELLI PRESSA”, in quanto gli allarmi sono tutti gestiti elettromeccanicamente dal robot di asservimento e, non avendo gestito l’effettiva comunicazione tra macchina IMM e robot, questi allarmi non sarebbero stati generati. Tuttavia, si è preferito inserire comunque i pulsanti che devono gestire tale funzionalità.

5.2.3. Implementazione della simulazione

Per realizzare la simulazione del funzionamento di una macchina IMM passando per le azioni operate sull’interfaccia grafica da un operatore, è stata implementata una classe Java nella quale sono stati definiti i metodi, descritti di seguito, che permettono di aggiornare i dati esposti dal Server OPC-UA al verificarsi di determinati eventi nell’interfaccia grafica. In questo modo, si è riuscito a mantenere separata l’implementazione della parte grafica da quella logica che simula il funzionamento della macchina IMM. Di seguito sono riportati i metodi “startServer” e “stopServer” che sono rispettivamente eseguiti al rilascio del pulsante “Start” e alla chiusura della finestra principale dell’interfaccia grafica realizzata. Il metodo “startServer” si occupa di inizializzare il Server OPC-UA, creare l’Address Space da esporre e avviare la comunicazione OPC-UA. Il metodo “stopServer” invece si occupa semplicemente di chiudere la comunicazione. Entrambi, richiamano metodi definiti nella classe “SampleConsoleServer” che è stata realizzata come descritto nel paragrafo 5.2.1, a partire dall’esempio contenuto nella Prosys Java SDK, e che per semplicità è riportata per intero in Appendice B.

```
public void startServer() throws SecureIdentityException,
IOException, UaServerException, StatusException,
UaInstantiationException, Service Exception,
NodeBuilderException, SAXException, ModelException {
      // Initializes server
      server = sampleConsoleServer.initialize(52520, 52443,
APP_NAME);
      // Creates address space and starts server
      sampleConsoleServer.createAddressSpace(server);
      sampleConsoleServer.run(server, false);
}
```

```
public void stopServer() throws StatusException,
UaInstantiationException, ServiceException,
SecureIdentityException, IOException, UaServerException,
NodeBuilderException, SAXException, ModelException {
      //Stops server simulation
      sampleConsoleServer.stopServer(server);
}
```

Successivamente è riportato il metodo “platenPosition” che si occupa di aggiornare lo stato dell’oggetto che rappresenta lo stampo all’interno del Server OPC-UA sia nel caso in cui siano abilitate le posizione intermedie dello stampo sia nel caso in cui siano disabilitate. Nel caso in cui siano abilitate le posizione intermedie, va simulata anche l’apertura dello stampo. Come già detto, l’operatore può impostare il tempo di apertura dello stampo agendo tramite l’interfaccia grafica. Per simulare quindi i tempi di apertura della pressa, nel caso in cui sono abilitate le posizioni intermedie, è stato definito un Timer task che verrà eseguito con una determinata frequenza. Il Timer task si occupa di aggiornare i dati relativi alle posizioni finali e alle posizioni intermedie dello stampo. Nello specifico, si è scelto di abilitare solo quattro posizioni intermedie in entrambe le direzioni del movimento dello stato (movimento positivo, in fase di apertura dello stampo, e movimento negativo, in fase di chiusura dello stampo). Pertanto, il rate di esecuzione del Timer task è definito dal rapporto tra il tempo di apertura totale dello stampo, settato dall’operatore agendo sull’interfaccia grafica, e il numero di posizione intermedie abilitate diminuito di un’unità, in quanto il conto delle posizioni intermedie parte da zero. In questo modo, ogni volta che viene eseguito il Timer task vengono aggiornati i valori delle posizioni intermedie nel Server OPC-UA durante le rispettive direzioni di movimento; solo quando si è raggiunta l’ultima posizione intermedia vengono anche aggiornati i valori delle posizioni finali nel Server OPC-UA. Anche in questo caso per semplicità il codice che implementa il Timer task è riportato in Appendice B. Nel caso in cui, invece, non sono abilitate le posizioni intermedie la funzione “platenPosition” si

occupa di aggiornare manualmente solo i valori delle posizioni finali esposte dal Server OPC-UA. Infine, tale funzione si occupa di aggiornare anche il conteggio del numero di cicli eseguiti nell'apposito campo del Server.

```
public void platenPosition(boolean currPos1, boolean
currPos2) {
      // The following if handles the refreshing of window
components and runs the timerTask
      if(handlerWindow.sliderIntermediate.getValue() == 1)
{
            if (handlerWindow.sliderAutomatic.getValue() ==
0)

      handlerWindow.refreshIMMComponents(false);
            t.schedule(timerTask, new Date(),
(long)handlerWindow.spinnerTimer.getValue()*1000/3); // The
period for the timer is in millisecond
      }
      else {
      // The following if increments the Cycle number at
the end of movement of IMM from position 1 to position 2
            if (handlerWindow.sliderMould.getValue() == 1
&& ((currPos1 &&
handlerWindow.sliderAutomatic.getValue()==0)^(currPos2 &&
handlerWindow.sliderAutomatic.getValue()==1))) {
                  UnsignedLong cycleNumber =
sampleConsoleServer.imm1.getMould_1Node().getIMMPartQuality
Node().getCycleCounter();

      sampleConsoleServer.imm1.getMould_1Node().getIMMPartQ
ualityNode().setCycleCounter(cycleNumber.add(1));
            }

      sampleConsoleServer.imm1.getMould_1Node().getMovableP
latenNode().setInPosition1(handlerWindow.sliderMould.getVal
ue() == 0);

      sampleConsoleServer.imm1.getMould_1Node().getMovableP
latenNode().setInPosition2(handlerWindow.sliderMould.getVal
ue() == 1);
            // The following if sets the state variable for
the IMM Cycle
            if (handlerWindow.sliderMould.getValue() == 1
&& handlerWindow.sliderAutomatic.getValue() == 0)
                  state = 3;
            else if (handlerWindow.sliderMould.getValue()
== 0 && handlerWindow.sliderAutomatic.getValue() == 0)
                  state = 2;
            // The following if handles the release of
semaphore to continue the automatic cycle
            if (handlerWindow.sliderAutomatic.getValue() ==
1)
                  sem.release();
      }
}
```

Per aggiornare lo stato degli oggetti che rappresentano rispettivamente gli estrattori e i martinetti della macchina IMM all'interno del Server OPC-UA, sono stati implementati i metodi riportati di seguito.

```
public void ejectorsPosition() {
      sampleConsoleServer.imm1.getMould_1Node().getEjector_1No
de().setInPosition1(handlerWindow.sliderEjectors.getValue() ==
0);
      sampleConsoleServer.imm1.getMould_1Node().getEjector_1No
de().setInPosition2(handlerWindow.sliderEjectors.getValue() ==
1);
      // The following if sets the state variable for the IMM
Cycle
      if (handlerWindow.sliderEjectors.getValue() == 1 &&
handlerWindow.sliderAutomatic.getValue() == 0)
            state = 4;
      else if (handlerWindow.sliderEjectors.getValue() == 0 &&
handlerWindow.sliderAutomatic.getValue() == 0)
            state = 1;
}
```

```
public void corepull1Position() {
      sampleConsoleServer.imm1.getMould_1Node().getCore_1Node(
).setInPosition1(handlerWindow.sliderCorePull1.getValue() ==
0);
      sampleConsoleServer.imm1.getMould_1Node().getCore_1Node(
).setInPosition2(handlerWindow.sliderCorePull1.getValue() ==
1);
      // The following if sets the state variable for the IMM
Cycle
      if (handlerWindow.sliderCorePull1.getValue() == 1 &&
handlerWindow.sliderAutomatic.getValue() == 0)
            state = 5;
      else if (handlerWindow.sliderCorePull1.getValue() == 0
&& handlerWindow.sliderAutomatic.getValue() == 0)
            state = 8;
}
```

```
public void corepull2Position() {
      sampleConsoleServer.imm1.getMould_1Node().getCore_2Node(
).setInPosition1(handlerWindow.sliderCorePull2.getValue() ==
0);
      sampleConsoleServer.imm1.getMould_1Node().getCore_2Node(
).setInPosition2(handlerWindow.sliderCorePull2.getValue() ==
1);
      // The following if sets the state variable for the IMM
Cycle
      if (handlerWindow.sliderCorePull2.getValue() == 1 &&
handlerWindow.sliderAutomatic.getValue() == 0)
            state = 6;
      else if (handlerWindow.sliderCorePull2.getValue() == 0
&& handlerWindow.sliderAutomatic.getValue() == 0)
            state = 7;
}
```

Il metodo "ejectorsPosition" si occupa di aggiornare i valori delle variabili che rappresentano le posizioni finali degli estrattori della macchina IMM, quando si verifica un cambio di stato dello slider "ESTRATTORI INDIETRO-AVANTI" Analogamente, al verificarsi di un cambio di stato degli slider "MARTINETTI 1 POS1-POS2" e "MARTINETTI 2 POS1-POS2", i metodi "corepull1Position" e "corepull2Position" aggiornano i valori delle variabili che rappresentano le posizioni finali dei martinetti della macchina IMM. Si ricorda che le specifiche EUROMAP 79 indicano di istanziare due estrattori (Ejector_1, Ejector_2) e 10 martinetti (Core_1, …, Core_10), ma per rispettare l'organizzazione logica del vecchio dispositivo sono utilizzate solo un'istanza per gli estrattori (Ejector_1) e due istanze per i martinetti (Core_1, Core_2). Di seguito è riportato invece il metodo "automaticIMM" il quale si occupa di gestire la simulazione automatica del processo di produzione della macchina IMM.

```
public void automaticIMM () {
      sampleConsoleServer.imm1.setEnableOperation(handlerWindow.
sliderAutomatic.getValue() == 1);
      th = new AutomaticThread();
      if (handlerWindow.sliderAutomatic.getValue() == 1)
            th.start();
}
```

Questo metodo dopo aver aggiornato il campo "ReadyForOperationWithRobot" all'interno della struttura che rappresenta la macchina IMM nel Server OPC-UA, avvia l'esecuzione del Thread "AutomaticThread", la cui implementazione è riportata in Appendice B. Per simulare il funzionamento automatico del ciclo di produzione della macchina IMM è

stato implementato un ciclo infinito con uno switch-case. Per passare da un caso all'altro è stata definita la variabile "state", di tipo intero, allo scopo di identificare in quale stato si trova la macchina rispetto al ciclo di produzione. In Tabella 20 sono riportati gli stati nei quali la macchina si può trovare e, per ogni stato, è riportata una breve descrizione dello stato e l'operazione da compiere per passare allo stato successivo.

Stato	Descrizione	Azione
0	Caso di default, inserito per gestire la selezione della simulazione automatica all'avvio dell'applicazione	Nessuna azione da applicare all'interfaccia grafica e incremento della variabile state.
1	Stampo aperto, estrattori indietro e martinetti in POS1.	Chiusura dello stampo e incremento della variabile state.
2	Stampo chiuso, estrattori indietro e martinetti in POS1.	Simulazione dell'iniezione del materiale, apertura pressa e incremento della variabile state.
3	Stampo aperto, estrattori indietro e martinetti in POS1.	Avanzamento estrattori e incremento della variabile state.
4	Stampo aperto, estrattori avanti e martinetti in POS1.	Core_1 in POS2 e incremento della variabile state.
5	Stampo aperto, estrattori avanti, Core_1 in POS2 e Core_2 in POS1.	Simulazione del recupero del pezzo prodotto, Core_2 in POS2 e incremento della variabile state.
6	Stampo aperto, estrattori avanti e martinetti in POS2.	Core_2 in POS1 e incremento della variabile state.
7	Stampo aperto, estrattori avanti, Core_1 in POS2 e Core_2 in POS1.	Core_1 in POS1 e incremento della variabile state.
8	Stampo aperto, estrattori avanti, martinetti in POS1.	Ritrazione estrattori e variabile state uguale ad 1.

Tabella 20: elenco degli stati implementati per la simulazione automatica della macchina IMM

Le azioni che riguardano la variazione di stato dello stampo, degli estrattori e dei martinetti vengono eseguite dal Thread richiamando le rispettive funzioni, "platenPosition", "ejectorsPosition", "corepull1Position" e "corepull2Position"; mentre, per semplicità, al fine di simulare sia l'iniezione del materiale plastico a stampo chiuso sia l'estrazione del pezzo prodotto è stata utilizzata l'apposita funzione "sleep", predefinita per la classe "Thread", la quale permette di mettere in pausa l'esecuzione del Thread per un certo periodo di tempo. Ovviamente, anche quando il simulatore della pressa è in automatico, possono comunque essere abilitate le posizioni intermedie, che si ricorda essere implementate con il Timer task. L'esecuzione in parallelo del Thread e del Timer Task porta con se problematiche di sincronizzazione tra i due. In particolare, quando il Thread avvia il Timer Task per simulare l'apertura dello stampo nel caso di posizioni intermedie abilitate, non è possibile sapere con certezza come vengano eseguiti i due. Questo porta a possibili problematiche nella simulazione del ciclo di produzione della macchina IMM. Pertanto, per evitare tale problematica è stato definito un semaforo che permette l'esecuzione di solo uno dei due alla

volta. A tale scopo, il semaforo è stato definito con un numero di token nullo. Più nel dettaglio quindi, il Thread, dopo aver avviato l'esecuzione del Timer task, cerca di acquisire il token del semaforo per poter andare avanti nella sua esecuzione, ma essendo nullo il numero di token iniziale, il Thread rimane bloccato nel cercare di acquisire il token. Il Thread viene liberato dal Timer task, il quale, giunto al termine della sua esecuzione, va ad aggiungere un token al semaforo. In questo modo una volta avviato il Timer task esso riuscirà a terminare la sua normale esecuzione, descritta precedentemente, senza essere interrotto dal Thread. Infine, il Thread è capace di gestire anche l'uscita dal ciclo infinito nel momento in cui l'operatore decide di terminare l'esecuzione automatica del ciclo di produzione agendo sullo slider "PRESSA IN AUTOMATICO". L'ultimo metodo da descrivere è il metodo "partsNOK" che si occupa di aggiornare il campo "CycleQuality" nella struttura che rappresenta la macchina IMM nel Server OPC-UA, per indicare al robot l'esito del ciclo attuale. Il metodo che si occupa di gestire tale aggiornamento è riportato di seguito.

```
public void partsNOK(){
      // Handles only 0 and 1 index for CycleQualityEnumeration
(to add also 2 and 3 index of enumeration change the maximum
property of sliderNOK in class GUI)
      sampleConsoleServer.imm1.getMould_1Node().getIMMPartQualit
yNode().setCycleQuality(handlerWindow.sliderNOK.getValue());
}
```

5.3. Testing

Per dimostrare la corretta configurazione ed implementazione del Server OPC-UA a partire dal caricamento dell'Information Model relativo alle specifiche EUROMAP 79 e il corretto funzionamento della simulazione del ciclo di produzione di una macchina IMM, si riportano i test effettuati. Non avendo implementato la comunicazione effettiva con il robot di asservimento, da collaudare utilizzando il simulatore realizzato, i test sono effettuati utilizzando il software UAExpert che fornisce tutte le funzionalità principali di un Client OPC-UA. Per mostrare i test effettuati nel modo più efficiente ed efficace possibile e per mettere in risalto come variano i dati esposti dal Server OPC-UA, i test sono stati divisi tra i seguenti casi:

- Stampo chiuso sia con posizioni intermedie disabilitate che con posizioni intermedie abilitate.
- Stampo aperto sia con posizioni intermedie disabilitate che con posizioni intermedie abilitate.

- Estrattori avanti.
- Estrattori indietro.
- Martinetti 1 POS1.
- Martinetti 1 POS2.
- Martinetti 2 POS1.
- Martinetti 2 POS2.
- Pezzo scarto OFF.
- Pezzo scarto ON.
- Pressa in automatico OFF.
- Pressa in automatico ON.

In Figura 43 e in Figura 44 sono visibili i test effettuati per mostrare i dati esposti dal Server OPC-UA rispettivamente per i casi in cui lo stampo risulta essere chiuso con posizioni intermedie disabilitate e con posizioni intermedia abilitate. Per entrambi i casi, i risultati mostrati dai test risultano essere in linea con quanto implementato e conformi alle norme EUROMAP 79. In particolare, si può notare che per le variabili che rappresentano le posizioni finali dello stampo (InPosition1 e InPosition2), il valore InPosition1 risulta sempre essere "TRUE", mentre il valore InPosition2 è sempre "FALSE". Tale combinazione di valori, come descritto nel paragrafo 2.8.1, corrisponde alla condizione di stampo chiuso. Per i valori delle posizioni intermedie, invece, si può notare che nel caso in cui sono abilitate il valore IntermediatePosition2To1 è pari a tre, ad indicare che è stata raggiunta la posizione finale durante il movimento di chiusura dello stampo.

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition1	InPosition1	true	Boolean	15:37:18.899	15:37:19.527	Good
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition2	InPosition2	false	Boolean	15:37:18.900	15:37:19.527	Good
3	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition1To2	IntermediatePosition1To2	0	Byte	01:00:00.000	15:36:35.421	Good
4	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition2To1	IntermediatePosition2To1	0	Byte	01:00:00.000	15:36:36.708	Good

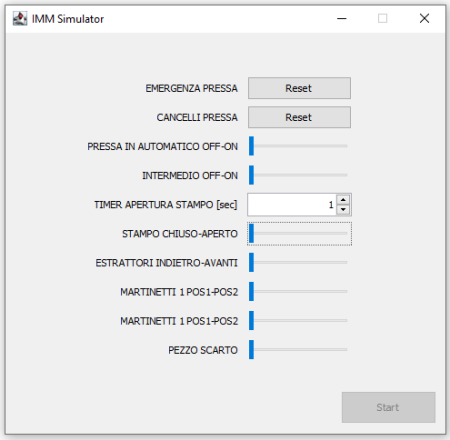


Figura 43: test effettuato per verificare la corretta simulazione della chiusura dello stampo con posizioni intermedie disabilitate

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition1	InPosition1	true	Boolean	15:40:23.162	15:40:23.523	Good
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition2	InPosition2	false	Boolean	15:40:23.162	15:40:23.523	Good
3	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition1To2	IntermediatePosition1To2	0	Byte	15:40:22.162	15:40:22.162	Good
4	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition2To1	IntermediatePosition2To1	3	Byte	15:40:23.162	15:40:23.523	Good

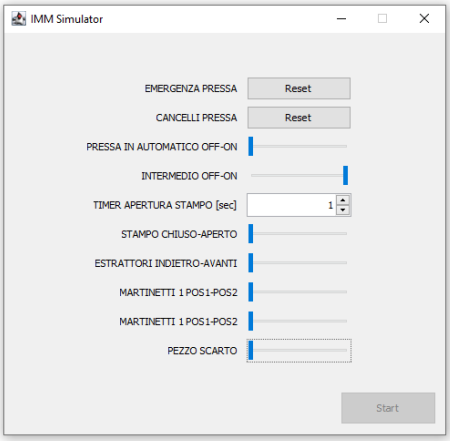


Figura 44: test effettuato per verificare la corretta simulazione della chiusura dello stampo con posizioni intermedie abilitate

Analogamente, la Figura 45 e la Figura 46 mostrano i risultati ottenuti rispettivamente nei casi in cui lo stampo della macchina IMM deve risultare aperto sia con posizioni intermedie disabilitate sia con posizioni intermedie abilitate. Anche in questi due casi i risultati sono in linea con quanto atteso. Al contrario del caso precedente, i valori delle posizioni finali InPosition1 e InPosition2 corrispondono rispettivamente al valore “FALSE” e al valore “TRUE”, mentre nel caso di posizioni intermedie abilitate è il valore IntermediatePosition1To2 ad assumere valore pari a tre.

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition1	InPosition1	false	Boolean	15:55:19.057	15:55:19.058	Good (0x000040...
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition2	InPosition2	true	Boolean	15:55:19.058	15:55:19.058	Good (0x000040...
3	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition1To2	IntermediatePosition1To2	0	Byte	01:00:00.000	15:55:04.978	Good
4	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition2To1	IntermediatePosition2To1	0	Byte	01:00:00.000	15:55:04.978	Good

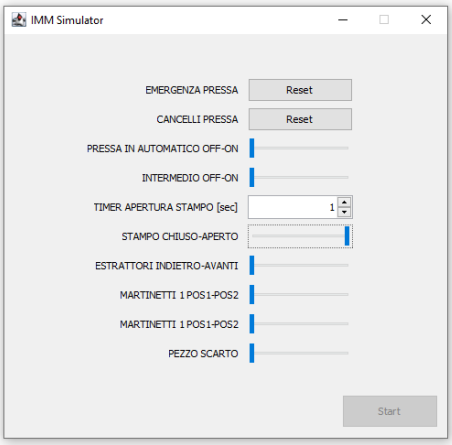
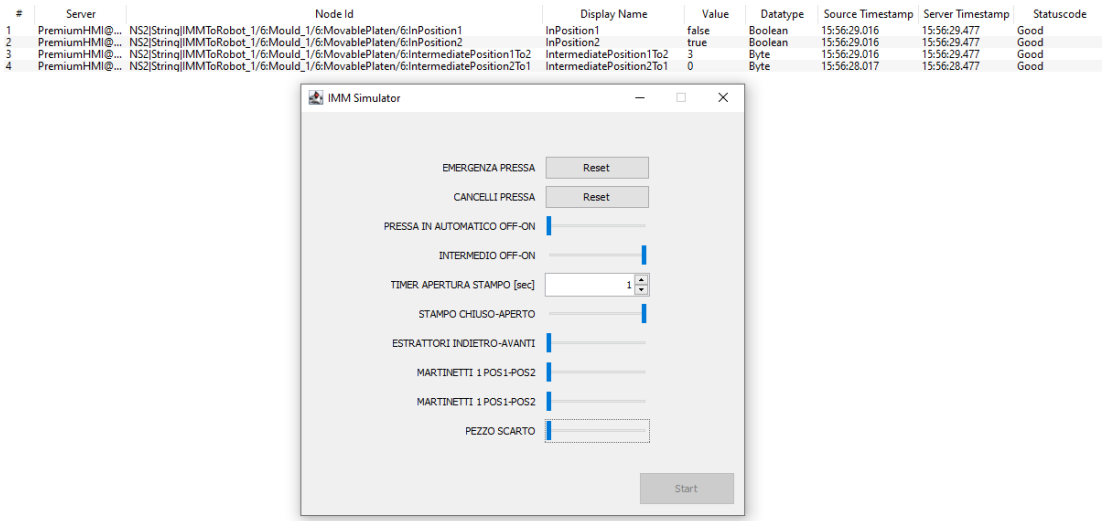


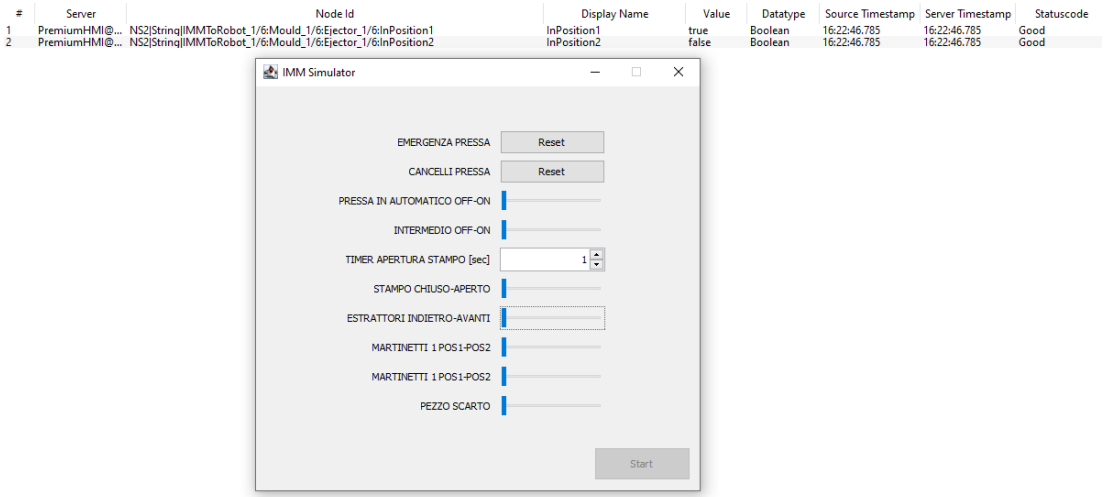
Figura 45: test effettuato per verificare la corretta simulazione dell’apertura dello stampo con posizioni intermedie disabilitate



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition1	InPosition1	false	Boolean	15:56:29.016	15:56:29.477	Good
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:InPosition2	InPosition2	true	Boolean	15:56:29.016	15:56:29.477	Good
3	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition1To2	IntermediatePosition1To2	3	Byte	15:56:29.016	15:56:29.477	Good
4	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:MovablePlaten/6:IntermediatePosition2To1	IntermediatePosition2To1	0	Byte	15:56:28.017	15:56:28.477	Good

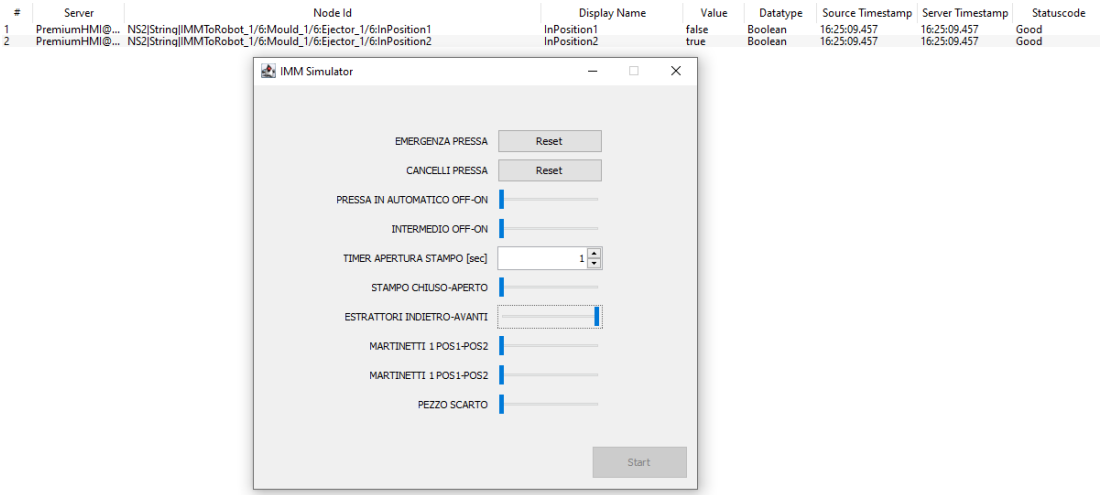
Figura 46: test effettuato per verificare la corretta simulazione dell'apertura dello stampo con posizioni intermedie abilitate

In Figura 47 e in Figura 48 sono visibili i test effettuati per mostrare i dati esposti dal Server OPC-UA rispettivamente per il caso in cui gli estrattori dello stampo risultano essere retratti internamente allo stampo e per il caso in cui gli estrattori sono avanzati esternamente allo stampo. In accordo con quanto descritto nelle specifiche EUROMAP 79, nel primo caso le variabili che rappresentano le posizioni finali degli estrattori, InPosition1 e InPosition2, risultano avere rispettivamente valore "TRUE" e valore "FALSE", mentre risultano opposti nel secondo caso. Va evidenziato che i nomi delle variabili che identificano le posizioni finali sono gli stessi per lo stampo, per gli estrattori e per i martinetti, come vedremo di seguito. Questo deriva dal fatto che è stato utilizzato lo stesso tipo di oggetto in fase di definizione dell'Address-Space, tuttavia esse si riferiscono ad istanze differenti di quel tipo di oggetto e questo è confermato dal fatto che tutte queste variabili hanno NodeId differente.



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Ejector_1/6:InPosition1	InPosition1	true	Boolean	16:22:46.785	16:22:46.785	Good
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Ejector_1/6:InPosition2	InPosition2	false	Boolean	16:22:46.785	16:22:46.785	Good

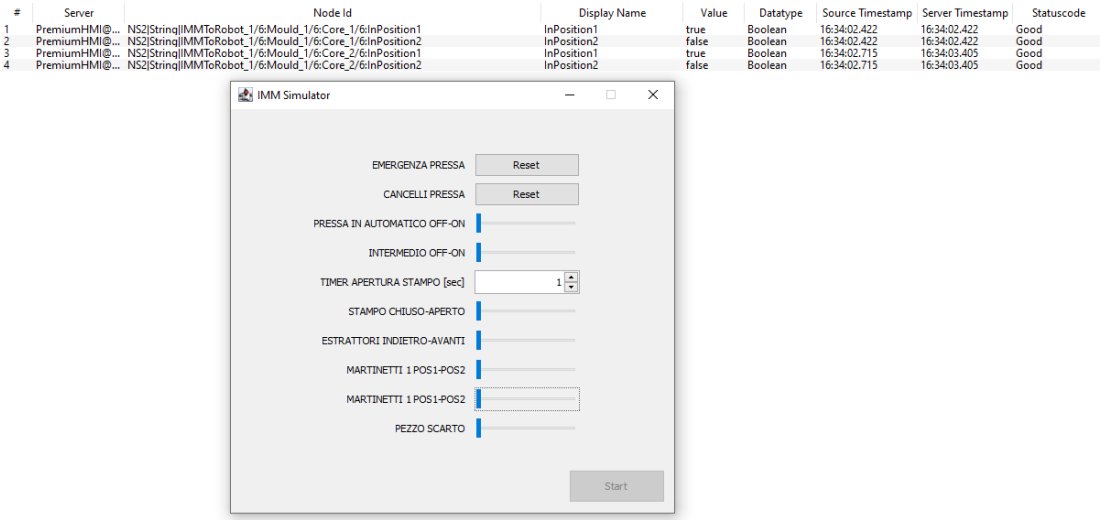
Figura 47: test effettuato per verificare la corretta simulazione della ritrazione degli estrattori



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Ejector_1/6:InPosition1	InPosition1	false	Boolean	16:25:09.457	16:25:09.457	Good
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Ejector_1/6:InPosition2	InPosition2	true	Boolean	16:25:09.457	16:25:09.457	Good

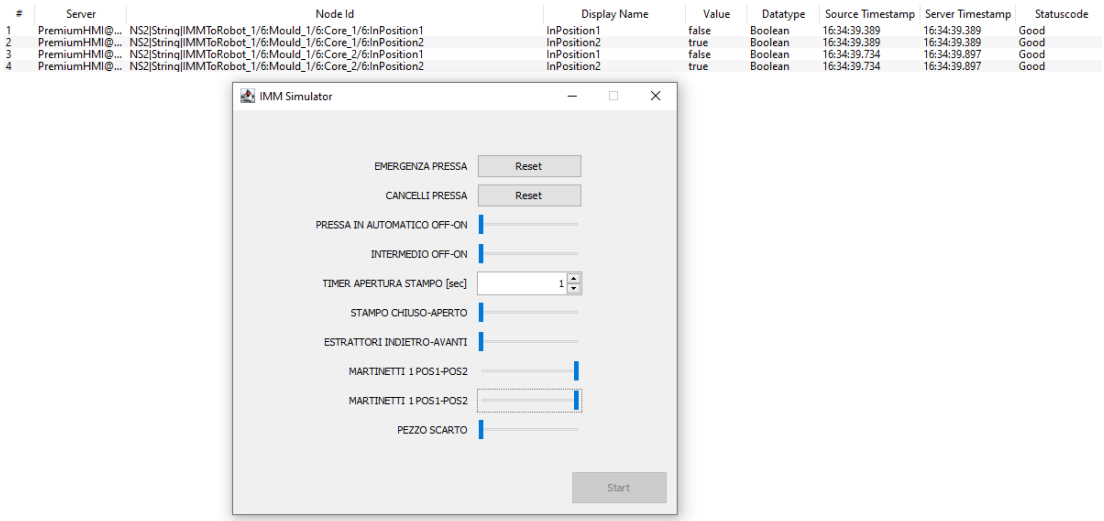
Figura 48: test effettuato per verificare la corretta simulazione dell'avanzamento degli estrattori

I test effettuati per mostrare i dati esposti dal Server OPC-UA rispettivamente per il caso in cui i martinetti dello stampo risultano essere retratti internamente allo stampo (POS1) e per il caso in cui i martinetti sono avanzati esternamente allo stampo (POS2) sono visibili in Figura 49 e in Figura 50. Come per il caso degli estrattori, nel primo caso le variabili che rappresentano le posizioni finali dei martinetti, InPosition1 e InPosition2, risultano avere rispettivamente valore "TRUE" e valore "FALSE", mentre risultano opposti nel secondo caso. Si ricorda che sono state utilizzate due istanze per rappresentare i martinetti della macchina IMM, pertanto, per semplicità nelle due figure sono riportate entrambe le istanze: il NodeId infatti risulta differente tra le due istanze.



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_1/6:InPosition1	InPosition1	true	Boolean	16:34:02.422	16:34:02.422	Good
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_1/6:InPosition2	InPosition2	false	Boolean	16:34:02.422	16:34:02.422	Good
3	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_2/6:InPosition1	InPosition1	true	Boolean	16:34:02.715	16:34:03.405	Good
4	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_2/6:InPosition2	InPosition2	false	Boolean	16:34:02.715	16:34:03.405	Good

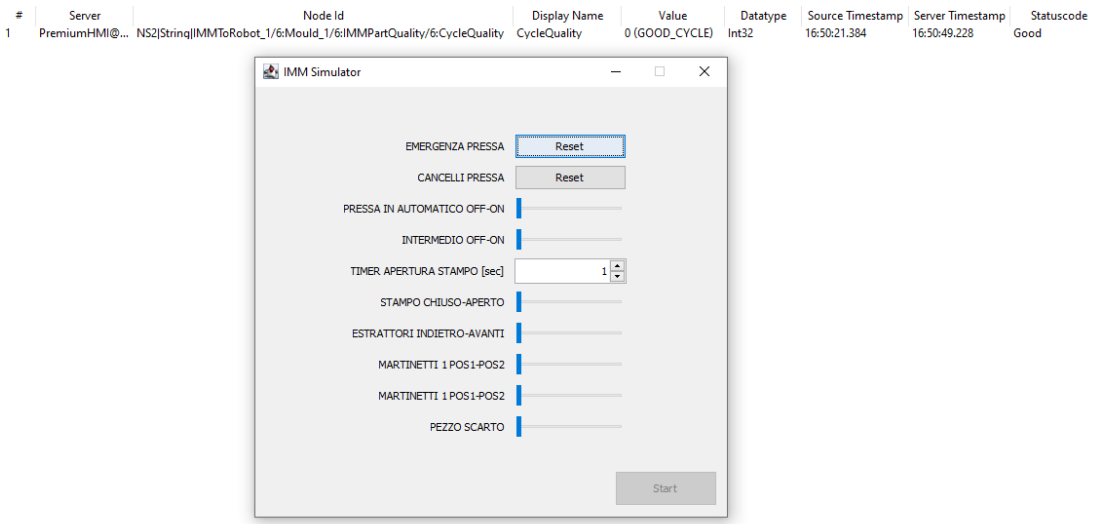
Figura 49: test effettuato per verificare la corretta simulazione della ritrazione dei martinetti



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_1/6:InPosition1	InPosition1	false	Boolean	16:34:39.389	16:34:39.389	Good
2	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_1/6:InPosition2	InPosition2	true	Boolean	16:34:39.389	16:34:39.389	Good
3	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_2/6:InPosition1	InPosition1	false	Boolean	16:34:39.734	16:34:39.897	Good
4	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:Core_2/6:InPosition2	InPosition2	true	Boolean	16:34:39.734	16:34:39.897	Good

Figura 50: test effettuato per verificare la corretta simulazione dell'avanzamento dei martinetti

La Figura 51 e la Figura 52, invece, mostrano il valore della variabile CycleQuality che invece indica al robot di asservimento l'esito del ciclo appena concluso. La variabile CycleQuality è una variabili di tipo "Enumeration" la cui definizione è riportata in Tabella 11, di cui però sono gestiti solamente i valori 0 ed 1. Pertanto, dai test effettuati ci si aspetta che nel caso in cui lo slider "PEZZO SCARTO" sia in posizione di OFF venga esposto la stringa "GOOD CYCLE", viceversa nel caso in cui lo slider "PEZZO SCARTO" sia in posizione di ON venga esposto la stringa "BAD CYCLE".



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	PremiumHMI@...	NS2|String|IMMToRobot_1/6:Mould_1/6:IMMPartQuality/6:CycleQuality	CycleQuality	0 (GOOD_CYCLE)	Int32	16:50:21.384	16:50:49.228	Good

Figura 51: test effettuato per verificare la corretta simulazione di un "GOOD CYCLE"

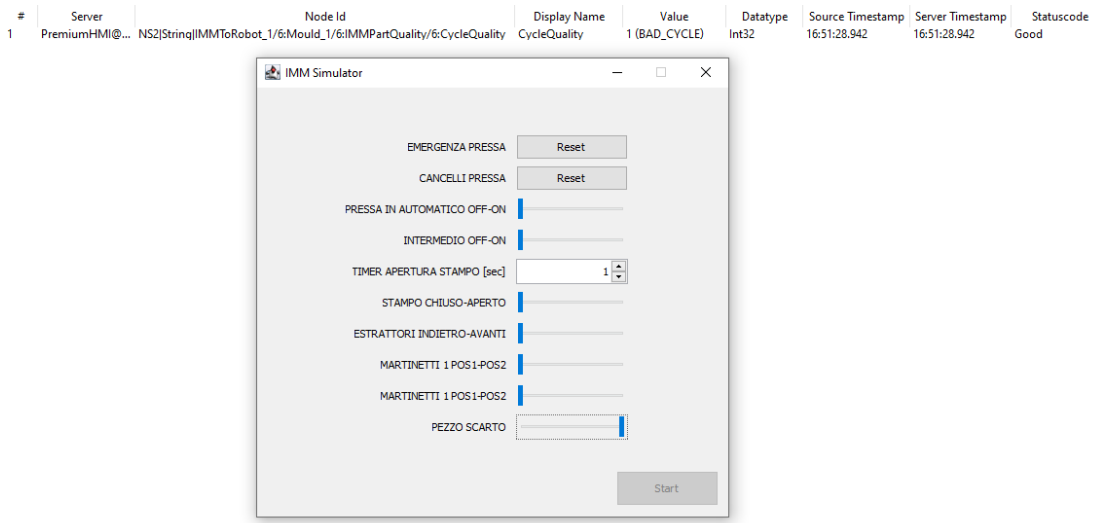


Figura 52: test effettuato per verificare la corretta simulazione di un "BAD CYCLE"

Infine, i test effettuati per mostrare i dati esposti dal Server OPC-UA relativi la caso in cui sia disabilitata o sia abilitata la simulazione automatica del processo produttivo della macchina IMM sono visibili rispettivamente in Figura 53 e in Figura 54. Nel primo caso la variabile ReadyForOperationWithRobot ha valore "FALSE", mentre nel secondo caso ha valore "TRUE" per indicare al robot che il simulatore sta funzionando in automatico.

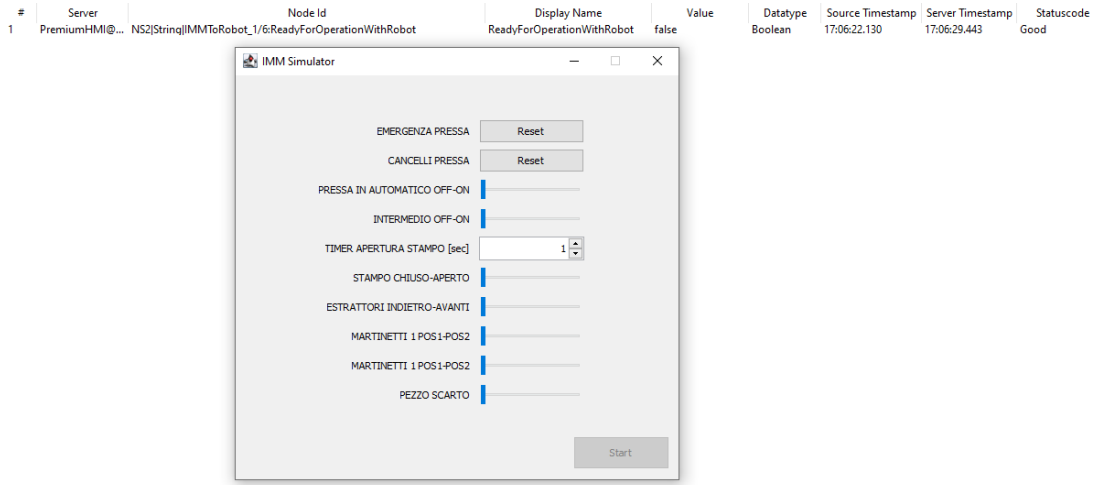


Figura 53: test effettuato per verificare la corretta simulazione del funzionamento in automatico disabilitato

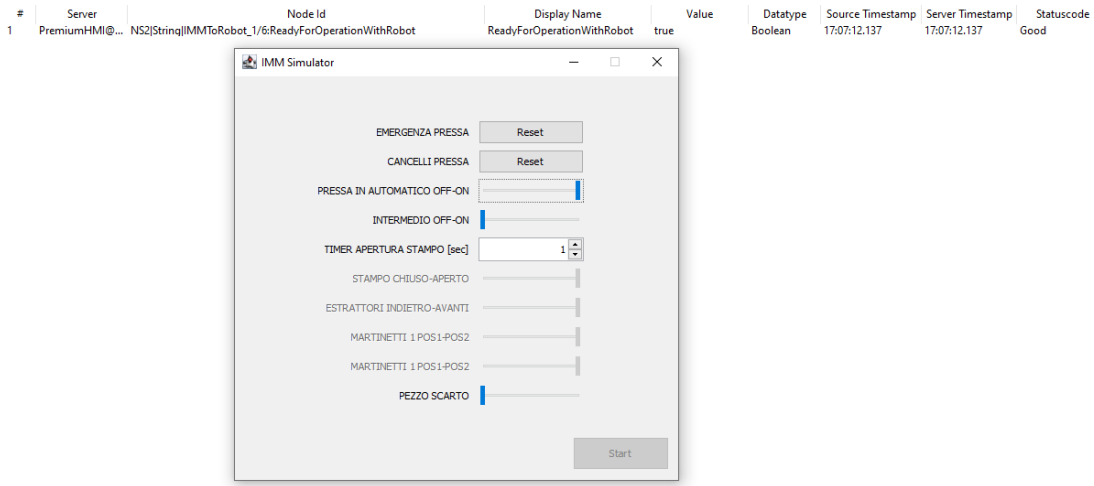


Figura 54: test effettuato per verificare la corretta simulazione del funzionamento in automatico abilitato

Inoltre, in Figura 54 si può notare come, nel caso in cui sia abilitato il funzionamento automatico, gli slider relativi alla gestione dello stampo, degli estrattori e dei martinetti vengono disabilitati allo scopo di evitare che l'operatore possa interagire con gli stessi e di conseguenza interferire con la simulazione automatica del ciclo di produzione della macchina IMM.

CONCLUSIONI

All'interno di questo documento è stato presentato il lavoro svolto presso la Campetella Robotic Center srl, coprendo tutte le fasi e le attività affrontate. La prima parte del documento, volta all'analisi della struttura e del funzionamento di base delle macchine automatiche prodotte dall'azienda e allo studio dei temi cardine dello standard OPC-UA, caratterizzandone l'impiego in ambiti di Industria 4.0 e nella lavorazione di materiale plastico, ha permesso di ottenere una base di conoscenza rispetto all'attuale stato dell'arte di queste tematiche. In questa prima parte dell'elaborato risulta chiaro ed evidente come siano necessari sforzi considerevoli, dal punto di vista della standardizzazione e della collaborazione di tutti i soggetti del mondo dell'industria, per attuare con successo la visione di Industria 4.0. Sotto questo aspetto, è inoltre evidente che lo standard OPC-UA, oggetto di questo documento, rappresenta una risorsa fondamentale per lo sviluppo del concetto di Industria 4.0, grazie alle sue caratteristiche di indipendenza dalla piattaforma e dal mezzo di comunicazione, astrazione e gestione virtuale di ogni genere di informazioni, possibilità di integrare nuovi tipi di dato complessi specifici per le esigenze del singolo utente e alla sua architettura di sicurezza in grado di garantire la confidenzialità dei dati. Nella seconda parte del documento sono invece riportati gli sviluppi di tre casi applicativi nei quali sono messe in evidenza diverse possibilità di configurazione della comunicazione OPC-UA e differenti modalità di utilizzo della stessa, partendo dall'implementazione dello scambio di dati tra macchina e gestionale del cliente nei primi due casi e arrivando allo scambio di dati in applicazioni Machine to Machine, come nel caso del simulatore della macchina IMM. Gli obiettivi prefissati per questi casi applicativi, che prevedevano la configurazione delle funzionalità di base della comunicazione OPC-UA, sono stati pienamente soddisfatti, come visibile dai risultati ottenuti durante la fase di testing delle rispettive applicazioni. Le scelte implementative e le diverse tecnologie utilizzate sono dettate dalla natura industriale del contesto e sono proprie dello sviluppo di applicativi basati sulla tecnologia OPC-UA. Tra i possibili sviluppi futuri degli applicativi presentati, sono da evidenziare l'implementazione di un'applicazione Client OPC-UA che permetta al controllore del robot di asservimento della macchina IMM di consultare i dati e le informazioni esposte dal Server OPC-UA, rappresentato dalla macchina IMM stessa, e la realizzazione di un'architettura di comunicazione Publisher-Subscriber tra il robot e la macchina IMM allo scopo di garantire lo scambio di segnali in ottica real-time per evitare collisioni tra le due macchine.

BIBLIOGRAFIA

[1] EUROMAP, OPC Foundation, *«EUROMAP 79 (OPC 40079), OPC UA interfaces for plastics and rubber - Data exchange between injection moulding machines and robots, Release Candidate 1.00.00»* 2021. [Online]. Consultabile al sito: https://www.euromap.org/euromap79.

[2] EUROMAP, OPC Foundation, *«EUROMAP 83 (OPC 40083), OPC UA for Plastics and Rubber Machinery – General Types Definitions, Release 1.03»* 2021. [Online]. Consultabile al sito: https://www.euromap.org/euromap83.

[3] Boston Consulting, *«Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries»* 2015. [Online]. Consultabile al sito: https://www.bcg.com/publications/2015/engineered_products_project_business_industry_4_future_productivity_growth_manufacturing_industries.

[4] J. Lange, F. Iwanitz e T. J. Burke, *OPC - From Data Access to Unified Architecture*, Vde Verlag GmbH, 2010.

[5] S.-H. Leitner, M. Damm e W. Mahnke, *OPC Unified Architecture*, Springer, 2009.

[6] OPC Foundation, *«OPC Unified Architecture - Interoperability for industry 4.0 and the Internet of Things»* 2020. [Online]. Consultabile al sito: https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf.

[7] OPC Foundation, *«OPC Unified Architecture - Part 1: Overview and Concepts, Release 1.04»* 2017. [Online]. Consultabile al sito: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/.

[8] OPC Foundation, *«OPC Unified Architecture - Part 2: Security Model, Release 1.04»* 2018. [Online]. Consultabile al sito: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/.

[9] OPC Foundation, *«OPC Unified Architecture - Part 3: Address Space Model, Release 1.04»* 2017. [Online]. Consultabile al sito: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-3-address-space-model/.

[10] OPC Foundation, *«OPC Unified Architecture - Part 4: Services, Release 1.04»* 2017. [Online]. Consultabile al sito: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-4-services/.

[11] OPC Foundation, *«OPC Unified Architecture - Part 5: Information Model, Release 1.04»* 2017. [Online]. Consultabile al sito: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model/.

[12] Prosys OPC Ltd, *«OPC-UA Java SDK»*.

[13] EUROMAP, *«Overview OPC UA Specifications»* [Online]. Consultabile al sito: https://www.euromap.org/i40/overview.

[14] OPC Foundation, *«Platform Industrie 4.0 Propose Reference Architecture Model of Industrie 4.0»* 2015. [Online]. Consultabile al sito: https://opcfoundation.org/news/press-releases/platform-industrie-4-0-propose-reference-architecture-model-industrie-4-0/.

[15] Prosys OPC Ltd, *«Prosys OPC-UA SDK for Java, Server Tutorial»*.

[16] *«Sito ufficiale ASEM»* [Online]. Consultabile al sito: https://www.asem.it/.

[17] *«Sito Ufficiale Campetella Robotic Center»* [Online]. Consultabile al sito: https://www.campetella.com/it/.

[18] *«Sito ufficiale Robox»* [Online]. Consultabile al sito: https://www.robox.it/.

[19] ASEM, *«Software Premium HMI»*.

[20] Robox, *«Software RDE»*.

[21] Robox, *«Software RDT»*.

[22] Unified Automation, *«Software UAExpert».*

[23] Unified Automation, *«Software UAModeler».*

APPENDICE A

CORRISPONDENZE EUROMAP 67/EUROMAP 79

La Figura 55 mostra le corrispondenze tra i segnali definiti dalle specifiche EUROMAP 67 e i campi della struttura IMMToRobotType. Queste corrispondenze sono state utilizzate per configurare la logica di simulazione della macchina IMM, seguendo le specifiche EUROMAP.

Signals in EUROMAP 67(.1)		**Matching Variable in OPC 40079**
Contact No.	**Signal designation**	
ZA1 ZC1	Emergency stop of machine, channel 1	*Not included in OPC 40079.* *(Use separate interface, e.g. EUROMAP 81)*
ZA2 ZC2	Emergency stop of machine channel 2	
ZA3 ZC3	Safety devices of machine channel 1	
ZA4 ZC4	Safety devices of machine channel 2	
ZA5 Optional	Reject	*IMMPartQualityType* → *CycleQuality* + *Method IMMToRobotCycleInformation* Note: *CycleQualityEnumeration* and *CavityCycleQualityEnumeration* are defined in OPC 40083
ZA6	Mould closed	*MouldType* → *MovablePlaten* → *InPosition1*
ZA7	Mould open position	*MouldType* → *MovablePlaten* → *InPosition2*
ZA8 Optional	Intermediate mould opening position	*MouldType* → *MovablePlaten* → *IntermediatePosition1To2* / *IntermediatePosition2To1* Note: OPC 40079 allows 255 intermediate positions for each direction
ZB2	Enable operation with handling device / robot (Automatic)	*IMMToRobotType* → *ReadyForOperationWithRobot*
ZB3	Ejector back position	*MouldType* → *Ejector_1* → *InPosition1*
ZB4	Ejector forward position	*MouldType* → *Ejector_1* → *InPosition2*
ZB5 Optional	Core pullers 1 in position 1 (Core pullers 1 free for handling device / robot to approach)	*MouldType* → *Core_1* → *InPosition1*
ZB6 Optional	Core pullers 1 in position 2 (Core pullers 1 in position to remove moulding)	*MouldType* → *Core_1* → *InPosition2*
ZB7 Optional	Core pullers 2 in position 1 (Core pullers 2 free for handling device / robot to approach)	*MouldType* → *Core_2* → *InPosition1*
ZB8 Optional	Core pullers 2 in position 2 (Core pullers 2 in position to remove moulding)	*MouldType* → *Core_2* → *InPosition2*

Figura 55: elenco delle corrispondenze da EUROMAP 67 a EUROMAP 79

APPENDICE B

CODICE SIMULATORE PRESSA

Classe Java “SampleConsoleServer”:

```
/**
 * A sample OPC UA server application.
 */
public class SampleConsoleServer {

  public static Logger logger =
LoggerFactory.getLogger(SampleConsoleServer.class);
  protected static String APP_NAME = "SampleConsoleServer";

  protected static String discoveryServerUrl =
"opc.tcp://localhost:4840";

  private final ScheduledExecutorService simulator =
Executors.newScheduledThreadPool(10);
  protected MyNodeManager myNodeManager;
  protected NodeManagerListener myNodeManagerListener = new
MyNodeManagerListener();
  protected UaServer server;

  private UaObject machinesFolder;
  public IMMToRobotObjectTypeNode imm1;

  /**
   * @param string
   */
  public static void println(String string) {
    System.out.println(string);
  }

  public UaServer getServer() {
          return server;
        }

  private final Runnable simulationTask = new Runnable() {

    @Override
    public void run() {
      if (server.isRunning()) {
        logger.debug("Simulating");
        simulate();
      }
    }
  };

  /**
   * Create a sample address space with a new folder, a device
object, a level variable, and an
   * alarm condition.
   * <p>
```

```
   * The method demonstrates the basic means to create the nodes and
references into the address
   * space.
   * <p>
   * Simulation of the level measurement is defined in {@link
#startSimulation()}
 * @param server
   *
   * @throws StatusException if the referred type nodes are not
found from the address space
   * @throws UaInstantiationException
   * @throws NodeBuilderException
   * @throws URISyntaxException
   * @throws ModelException
   * @throws IOException
   * @throws SAXException
 * @throws ServiceException
   *
   */
  public void createAddressSpace(UaServer server) throws
StatusException, UaInstantiationException, NodeBuilderException,
SAXException, IOException, ModelException, ServiceException {

    // My Node Manager
    myNodeManager = new MyNodeManager(server,
MyNodeManager.NAMESPACE);
    myNodeManager.addListener(myNodeManagerListener);

    // Load the standard information models
    loadInformationModels();

    // Method to create instances
    createInstances();

  }


//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
  // This method: - detects the Machines ObjectFolder for the
NodeManager
  //                    - creates instance of object defined and
loaded from .xml file
  //                    - adds the Node to the Machines ObjectFolder
and set the Reference

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
  protected void createInstances() throws StatusException {
        machinesFolder = (UaObject)
getServer().getNodeManagerRoot().getNode(new NodeId(5, 1001));
        imm1 =
myNodeManager.createInstance(IMMToRobotObjectTypeNode.class,
"IMMToRobot_1");
        myNodeManager.addNodeAndReference(machinesFolder, imm1,
Identifiers.Organizes);
  }


//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
```

```
  // This method initializes Null Node to simulate IMM accordingly
to Euromap79

//////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
  @SuppressWarnings("deprecation")
  protected void initializeNode() {
        imm1.setEnableOperation(false);

imm1.getMould_1Node().getIMMPartQualityNode().setCycleCounter(new
UnsignedLong(0));

imm1.getMould_1Node().getIMMPartQualityNode().setCycleQuality(0);
        imm1.getMould_1Node().getMovablePlatenNode().setMovement(0);
  }

  /**
  *
  */
  public UaServer initialize(int port, int httpsPort, String
applicationName)
      throws SecureIdentityException, IOException, UaServerException
{

    // *** Create the server
    server = new UaServer();

    // *** Server Endpoints
    // TCP Port number for the UA TCP protocol
    server.setPort(Protocol.OpcTcp, port);
    // TCP Port for the HTTPS protocol
    server.setPort(Protocol.OpcHttps, httpsPort);

    // optional server name part of the URI (default for all
protocols)
    server.setServerName("OPCUA/" + applicationName);

    // *** 'init' creates the service handlers and the default
endpoints
    // *** according to the settings defined above
    server.init();

      return server;

  }


//////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
  // This method stop the simulation and closed the server

//////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
  public void stopServer(UaServer server) throws
SecureIdentityException, IOException, UaServerException,
StatusException, UaInstantiationException, ServiceException,
NodeBuilderException, SAXException, ModelException {
          // *** End
          stopSimulation();
          // Notify the clients about a shutdown, with a 5 second
delay
```

```
            logger.info("Shutting down...");
            server.shutdown(0, new LocalizedText("Closed by user",
Locale.ENGLISH));
            println("Closed.");
          }

  /**
   * Load information models into the address space. Also register
classes, to be able to use the
   * respective Java classes with
NodeManagerUaNode.createInstance().
   *
   * See the Codegen Manual on instructions how to use your own
models.
 * @throws ModelException
 * @throws IOException
 * @throws SAXException
 * @throws ServiceException
   */
  protected void loadInformationModels() throws SAXException,
IOException, ModelException, ServiceException {
        //ModNico ----->
        server.registerModel(MyServerInformationModel.MODEL);
        server.getAddressSpace().loadModel(new
File("Opc.Ua.Di.NodeSet2.xml").toURI());
        server.getAddressSpace().loadModel(new
File("Opc.Ua.PlasticsRubber.GeneralTypes.NodeSet2.xml").toURI());
        server.getAddressSpace().loadModel(new
File("Opc.Ua.Machinery.NodeSet2.xml").toURI());
        server.getAddressSpace().loadModel(new
File("Opc.Ua.PlasticsRubber.IMM2Robot.xml").toURI());
  }

  /**
   * Run the server.
 * @param server
   *
   * @param enableSessionDiagnostics
   * @throws UaServerException
   * @throws StatusException
   */
  public void run(UaServer server, boolean enableSessionDiagnostics)
throws UaServerException, StatusException {
    server.start();
    startSimulation();
  }

  /**
   *
   */
  protected void sendEvent() {
    myNodeManager.sendEvent();
  }

  /**
  *
  */
  protected void simulate() {
    myNodeManager.simulate();
  }
```

```
  /**
   * Starts the simulation of the level measurement.
   */
  protected void startSimulation() {
    simulator.scheduleAtFixedRate(simulationTask, 1000, 1000,
TimeUnit.MILLISECONDS);
    logger.info("Simulation started.");
  }

  /**
   * Ends simulation.
   */
  protected void stopSimulation() {
    simulator.shutdown();
    logger.info("Simulation stopped.");
  }
}
```

Timer task implementato per la simulazione del tempo di apertura dello stampo della macchina IMM:

```
// Create a new timer and a new task to handle the
IntermediatePosition fields
Timer t = new Timer();
TimerTask timerTask = new TimerTask() {
int i=0;
@SuppressWarnings("deprecation")
public void run() {
        // Reset the IntermediatePosition fields
sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
IntermediatePosition1To2(new UnsignedByte((byte)0));

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
IntermediatePosition2To1(new UnsignedByte((byte)0));
        // The following if is true when the mould is closed, the
second condition is temporarly? (no initialization of the OPCUA
fields) to handle the first cycle
        if((currPos1 && !currPos2) || (!currPos1 && !currPos2 &&
handlerWindow.sliderMould.getValue() == 1)) {
              // The following if increments the Cycle number at the
end of movement of IMM from position 1 to position 2
              if (i == 3 && currPos1) {
                    UnsignedLong cycleNumber =
sampleConsoleServer.imm1.getMould_1Node().getIMMPartQualityNode().ge
tCycleCounter();

sampleConsoleServer.imm1.getMould_1Node().getIMMPartQualityNode().se
tCycleCounter(cycleNumber.add(1));
              }

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
IntermediatePosition1To2(new UnsignedByte((byte)i));

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
Movement(1);
        }
```

```
        // The following if is true when the mould is opened, the
second condition is temporarly? (no initialization of the OPCUA
fields) to handle the first cycle
        else if ((!currPos1 && currPos2) || (!currPos1 && !currPos2
&& handlerWindow.sliderMould.getValue() == 0)) {

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
IntermediatePosition2To1(new UnsignedByte((byte)i));

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
Movement(2);
        }
        // The following if handles the end of the timer
        if(i == 3) {

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
InPosition1(handlerWindow.sliderMould.getValue() == 0);

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
InPosition2(handlerWindow.sliderMould.getValue() == 1);

sampleConsoleServer.imm1.getMould_1Node().getMovablePlatenNode().set
Movement(0);
              // The following if sets the state variable for the
IMM Cycle
              if (handlerWindow.sliderMould.getValue() == 1 &&
handlerWindow.sliderAutomatic.getValue() == 0)
                        state = 3;
              else if (handlerWindow.sliderMould.getValue() == 0 &&
handlerWindow.sliderAutomatic.getValue() == 0)
                        state = 2;
              // The following if handles the refreshing of window
components and the release of semaphore to continue the automatic
cycle
              if (handlerWindow.sliderAutomatic.getValue() == 0)
                    handlerWindow.refreshIMMComponents(true);
              else
                    sem.release();
              t.cancel();
        }
        i++;
}
    };
```

Definizione della classe “AutomaticThread” che simula il ciclo di produzione della macchina IMM in automatico:

```
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
// This class extends the Thread class in order to create a new
Thread class for simulating the automatic cycle
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
public class AutomaticThread extends Thread{
      public void run() {

            handlerWindow.refreshIMMComponents(false);

            while(true) {

                  switch(state) {

                  // Case 1:  - is verified when mould is opened,
ejectors are in mould position (back), both corepullers are in mould
position (POS1)
                  //                - the action to perform is close
mould (acquire the semaphore is requested to synchronize the
timerTask and the AutomaticThread)
                  case 1:
                        handlerWindow.sliderMould.setValue(0);
                        platenPosition(true, false);
                        try {
                              sem.acquire();
                        } catch (InterruptedException e1) {
                              // TODO Auto-generated catch block
                              e1.printStackTrace();
                        }
                        state++;
                        break;

                  // Case 2:  - is verified when mould is closed,
ejectors are in mould position (back), both corepullers are in mould
position (POS1)
                  //                - the actions to perform is:
                  //
            - sleep the Thread to simulate the injection operation
                  //
            - open mould (acquire the semaphore is requested to
synchronize the timerTask and the AutomaticThread)
                  case 2:
                        try {
                              Thread.sleep(1000);
                        } catch (InterruptedException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        }
                        handlerWindow.sliderMould.setValue(1);
                        platenPosition(false, true);
                        try {
                              sem.acquire();
                        } catch (InterruptedException e1) {
                              // TODO Auto-generated catch block
                              e1.printStackTrace();
                        }
                        state++;
```

```
                        break;

                    // Case 3:  - is verified when mould is opened,
ejectors are in mould position (back), both corepullers are in mould
position (POS1)
                    //                - the action to perform is
ejectors forward
                    case 3:
                        handlerWindow.sliderEjectors.setValue(1);
                        ejectorsPosition();
                        state++;
                        break;

                    // Case 4:  - is verified when mould is opened,
ejectors are out of mould position (forward), both corepullers are
in mould position (POS1)
                    //                - the action to perform is
corePull1 forward
                    case 4:
                        handlerWindow.sliderCorePull1.setValue(1);
                        corepull1Position();
                        state++;
                        break;

                    // Case 5:  - is verified when mould is opened,
ejectors are out of mould position (forward), only corepull1 are in
mould position (POS1)
                    //                - the action to perform is:
                    //
            - corePull2 forward
                    //
            - sleep the Thread to simulate the extraction of the
moulded part
                    case 5:
                        handlerWindow.sliderCorePull2.setValue(1);
                        corepull2Position();
                        try {
                            Thread.sleep(1000);
                        } catch (InterruptedException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                        state++;
                        break;

                    // Case 6:  - is verified when mould is opened,
ejectors are out of mould position (forward), both corepullers are
out of mould position (POS2)
                    //                - the action to perform is
corePull2 backward
                    case 6:
                        handlerWindow.sliderCorePull2.setValue(0);
                        corepull2Position();
                        state++;
                        break;

                    // Case 7:  - is verified when mould is opened,
ejectors are out of mould position (forward), only corepull1 are out
of mould position (POS2)
                    //                - the action to perform is
corePull1 backward
```

```
                    case 7:
                          handlerWindow.sliderCorePull1.setValue(0);
                          corepull1Position();
                          state++;
                          break;

                    // Case 8:  - is verified when mould is opened,
ejectors are out of mould position (forward), both corepullers are
in mould position (POS1)
                    //                - the action to perform is
ejectors backward
                    case 8:
                          handlerWindow.sliderEjectors.setValue(0);
                          ejectorsPosition();
                          state=1;
                          break;

                    // Default case:  - is verified when mould is
opened, ejectors are in mould position (forward), both corepullers
are out of mould position (POS2)
                    //                            - no action to
perform (only increment the state variable of the automatic cycle)
                    default:
                          if (state!=0)
                                break;
                          state++;
                          break;
                    }

                    // The following handles the end of automatic
cycle
                    if (handlerWindow.sliderAutomatic.getValue() == 0)
{
                          handlerWindow.refreshIMMComponents(true);
                          break;
                    }

                    try {
                          Thread.sleep(50);
                    } catch (InterruptedException e) {
                          // TODO Auto-generated catch block
                          e.printStackTrace();
                    }
              }
        }
    }
```

RINGRAZIAMENTI

A chiusura di questo documento, desidero menzionare e ringraziare tutte le persone senza le quali questo lavoro di tesi non esisterebbe nemmeno. Ringrazio, in primis, il Prof. Bonci per avermi guidato nella fase più importante del mio percorso accademico. Un sentito grazie al correlatore di tesi, l'Ing. Lorenzo Gagliardini, per il supporto costante e le dritte indispensabili in ogni fase del mio lavoro. Ringrazio infinitamente la mia famiglia, i miei genitori, mio fratello e i miei nonni, per avermi trasmesso i loro valori senza i quali oggi non sarei ciò che sono e per avermi sempre sostenuto, appoggiando ogni mia decisione. Un grazie speciale alla mia fidanzata Elena, per avermi supportato e sopportato durante tutto il mio percorso e per avermi incoraggiato durante i momenti di sconforto ed ansia. Ringrazio poi i miei amici e i miei compagni di corso, le persone che hanno reso questo percorso più leggero strappandomi un'infinità di risate.

Infine, dedico questa tesi a me stesso, ai miei sacrifici e alla mia tenacia che mi hanno permesso di raggiungere questo traguardo, augurandomi che questo percorso sia di ispirazione per la mia carriera lavorativa e per il mio futuro.