ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Il protocollo OPC UA

Elaborato in

Reti di Telecomunicazione

Relatore
Prof. Franco Callegati

Presentato da
Gianni Tumedei

ANNO ACCADEMICO 2018-2019

Indice

Glossario delle abbreviazioni

Di seguito è riportato il significato tutte le abbreviazioni utilizzate in questo documento. Per acronimi che non riguardano strettamente gli argomenti trattati è inoltre fornita una breve definizione. L'elenco è ordinato alfabeticamente.

CA	Certification Authority
COM	Component Object Model
Interfaccia per componenti software introdotta da Microsoft. Permette la comunicazione tra processi e la creazione dinamica di oggetti con qualsiasi linguaggio di programmazione che la supporta.	
DCOM	Distributed Component Object Model
DCS	Distributed Control System
Sistema di controllo utilizzato per gestire cicli di automazione di processi di grandi dimensioni, che hanno la caratteristica di essere distribuiti su vari controllori, anche geograficamente dislocati.	
ERP	Enterprise Resource Planning
Software di gestione che integra tutti i processi di business rilevanti di un'azienda.	
GUID	Global Unique Identifier
HMI	Human-Machine Interface
Funzione o componente di un dispositivo o software che permette all'essere umano di interagire con le macchine.	
JNI	Java Native Interface
Framework che consente a codice Java di eseguire ed essere eseguito da applicativi scritti in altri linguaggi, come C e C++.	
MES	Manufacturing Execution System
Sistema informatizzato che ha la principale funzione di gestire e controllare la funzione produttiva di un'azienda.	
OLE	Object Linking and Embedding
OPC	• OLE for Process Control (OPC classic) • Open Platform Communications (significato attuale)
OPC A&E	OPC Alarms & Events

OPC DA	OPC Data Access
OPC HDA	OPC Historical Data Access
OPC XML/DA	OPC Data Access via XML
PKI	Public Key Infrastructure
SCADA	System Control and Data Acquisition
Tipologia molto usata di sistemi machine to machine, caratterizzata dalla presenza di un'entità di supervisione che esegue controllo da remoto e acquisizione dati su sistemi, equipaggiamenti e impianti attraverso opportuni canali di comunicazione.	
SOAP	Simplified Object Access Protocol
Protocollo per eseguire lo scambio di messaggi tra componenti software attraverso il paradigma della programmazione ad oggetti.	
UA	Unified Architecture
URI	Uniform Resource Identifier
VA	Validation Authority

1 Introduzione

Il mondo dell'industria è in continua evoluzione. Già da anni è in atto una corsa all'automazione dei processi, che ha visto l'introduzione nelle fabbriche di un numero sempre crescente di macchine, computer, robot e sistemi di controllo. Questo ha permesso di aumentare la produttività e la flessibilità dei processi, realizzare prodotti di qualità superiore, incrementare il livello di sicurezza e ridurre la necessità di interventi da parte degli operatori. Con l'avvento di Industria 4.0 [1] il numero di oggetti interconnessi e comunicanti negli impianti 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.

In questo ambiente di sistemi e dispositivi eterogenei, un argomento di importanza critica è come mettere in comunicazione queste entità in maniera standard. È necessario un sistema platform-independent e altamente scalabile, che sia in grado di fornire un ristretto set di feature orientate alla massima efficienza ad un dispositivo embedded, ma anche un'ampia gamma di soluzioni ad un software gestionale, per permettere la comunicazione con tutti i sistemi sottostanti e con impianti geograficamente distanti. Perché i dati inviati siano comprensibili al ricevente, va definita un'infrastruttura efficace per la modellazione delle informazioni.

La sicurezza dev'essere un altro obiettivo di questa soluzione. Con un numero sempre più elevato di dispositivi connessi alla rete, è necessario garantire che i dati siano autentici e che vi abbia accesso solo chi dispone delle autorizzazioni necessarie.

Infine, è indispensabile che questo sistema sia facile da adottare ed estendere, in modo che occorra il minimo sforzo per renderlo compatibile con nuove tecnologie e soluzioni.

Questo trattato analizza nel dettaglio OPC Unified Architecture, un protocollo per lo scambio di informazioni in ambito industriale che si pone tutti gli obiettivi sopra citati.

1.1 OPC Foundation

La OPC Foundation [2] [3] è un'organizzazione no-profit nata nel 1994, con lo scopo di definire uno standard Plug&Play per i driver di dispositivi industriali, fornendo così

un'interfaccia comune per leggere e scrivere dati su tali device. Il risultato del progetto fu il protocollo OPC Data Access [4], rilasciato nel 1996.
In origine la OPC Foundation era composta da cinque venditori di soluzioni per l'automazione industriale (Fisher-Rosemount, Rockwell Software, Opto 22, Intellution e Intuitive Technology). Oggi, dopo 25 anni, comprende più di 600 membri in Cina, Europa, Giappone e America.

1.2 Certificazione

La OPC Foundation definisce un Compliance Program per la certificazione dei prodotti che adottano lo standard OPC UA.
La certificazione si compone di due livelli. Il primo, detto *self certification*, sfrutta appositi strumenti forniti dalla fondazione ai membri. Questi programmi analizzano un prodotto e rilasciano un certificato di conformità qualora i test eseguiti siano andati a buon fine. I prodotti conformi possono esibire il logo Self-Tested, che indica un livello base di conformità allo standard OPC UA.
Il secondo livello consiste in un insieme di test più approfonditi, eseguiti da appositi laboratori di certificazione indipendenti. Un prodotto che passa la certificazione può esibire il logo OPC Certified, che indica un alto livello di conformità allo standard.
La OPC Foundation incoraggia l'acquisto di prodotti OPC Certified, per ridurre i problemi di interoperabilità e assicurare affidabilità e prestazioni elevate.

1.3 Storia

1.3.1 OLE for Process Control

Già nei primi anni '90, l'automazione era un argomento di cruciale importanza per le industrie manifatturiere e sempre più aziende decidevano di effettuare investimenti in quella direzione. Le soluzioni software per la raccolta dati da dispositivi di controllo dell'automazione non mancavano certo, tuttavia erano proprietarie, usavano driver

personalizzati per la comunicazione e trasmettevano i dati nei formati più disparati. Questo aveva un impatto non indifferente sulle aziende, che si vedevano sostanzialmente legate ad un fornitore non appena adottavano uno dei suoi software, ed erano costrette a contattarlo ogni volta che l'applicazione doveva essere integrata in un nuovo ambiente, ovviamente con le relative spese per l'implementazione di una soluzione ad hoc.
In questo scenario, una task force di produttori decise di riunirsi per definire uno standard di comunicazione con i dispositivi per l'automazione industriale. Dal loro lavoro nacque nel 1996 il protocollo OPC Data Access e la OPC Foundation fu creata per mantenerlo [4].

OPC DA ebbe un successo immediato grazie all'utilizzo della tecnologia Microsoft COM/DCOM, evoluzione di OLE, che ne semplificò notevolmente il processo di adozione e rese semplice la creazione di applicativi basati su standard OPC. Negli anni successivi, la OPC Foundation continuò il su lavoro con il rilascio di OPC Alarms & Events (1999), OPC Historical Data Access (2001) e numerose altre soluzioni.

1.3.2 Da OPC a OPC UA

Una delle maggiori ragioni del successo di OPC, ossia l'adozione di COM/DCOM, è anche la sua limitazione principale, in quanto lega lo standard alla piattaforma Windows. Il primo tentativo nella creazione di una versione platform-independent si ebbe nel 2003, quando la OPC Foundation rilasciò OPC XML-DA, che rimpiazza COM/DCOM con HTTP/SOAP e Web Service.
A causa di notevoli limitazioni dal punto di vista delle performance, fu subito chiaro che XML-DA non avrebbe sostituito OPC DA. La OPC Foundation si mise quindi al lavoro per la creazione di uno standard in grado di rimpiazzare definitivamente tutte le soluzioni precedenti, mantenendo le ottime performance di OPC classico ma superando le sue limitazioni, ovvero:

- La dipendenza dalla piattaforma Windows a causa dell'adozione della tecnologia COM/DCOM
- I problemi di DCOM per quanto riguarda gli accessi da remoto, oltre alla sua rigidità e complessità di configurazione
- La difficoltà da parte delle compagnie che adottavano OPC nell'esposizione di dati e sistemi complessi

Nel 2006, la OPC Foundation rilasciò il protocollo OPC Unified Architecture.

1.3.3 OPC UA oggi

Dopo la pubblicazione iniziale, la OPC Foundation ha continuato ad aggiornare regolarmente il protocollo negli anni. Attualmente (novembre 2019), la versione più recente di OPC UA è la 1.04, rilasciata nel 2017. Alcune delle novità introdotte sono il pattern architetturale publish-subscribe e l'introduzione dei SectionLess Service.

1.4 Specifiche

Le specifiche di OPC UA [5] sono composte da 15 parti. Per quanto siano complete, dettagliate e indispensabili nel caso si cercasse la risposta ad un quesito molto specifico, sono anche particolarmente complesse, pertanto non costituiscono il miglior modo per approcciarsi a OPC UA da parte di uno sviluppatore. Una lettura più adatta a un primo contatto con il protocollo è il libro "OPC Unified Architecture" degli autori Wolfgang Mahnke, Stefan-Helmut Leitner e Matthias Damm [6], che hanno collaborato alla creazione di OPC UA. Pertanto, questo trattato analizza i concetti introdotti dallo standard in un ordine simile a quello adottato dal libro. Per ogni argomento esposto è comunque indicata la sezione delle specifiche corrispondente.
Di seguito sono riportati i capitoli delle specifiche di OPC UA, accompagnati da una breve spiegazione.

1. Overview and Concepts: introduzione
2. Security Model: requisiti e modello di sicurezza
3. Address Space Model: blocchi per la costruzione e l'esposizione di dati e la creazione di uno spazio degli indirizzi
4. Services: definizione delle modalità di interazione tra applicazioni client e server
5. Information Model: framework per tutti gli Information Model che usano OPC UA; definisce:
    - Gli entry points allo spazio degli indirizzi

- I tipi di dato di base attraverso cui costruire gerarchie
- I tipi di dato estendibili
- I Server Object che forniscono capacità e informazioni diagnostiche

6. Mappings: mapping dei servizi su messaggi e modalità di trasferimento di tali messaggi
7. Profiles: subset di funzionalità strutturati in:
    - Conformance Unit: definiscono un piccolo set di funzioni che è buona prassi usare in congiunzione
    - Profiles: liste di Conformance Unit
8. Data Access: Information Model per rappresentare e utilizzare dati in tempo reale (corrisponde a OPC DA)
9. Alarms and Conditions: Information Model per processare allarmi, monitorare le condizioni delle macchine e rilevare l'accadere di eventi (corrisponde a OPC A&E)
10. Programs: Information model per la definizione del concetto di programma e del suo utilizzo in ambito OPC UA
11. Historical Access: Information Model per l'accesso, la modifica e la presentazione di dati storici (corrisponde a OPC HDA)
12. Discovery and Global Services: definizione delle modalità di ricerca dei server OPC nella rete da parte dei client e dei metodi per l'ottenimento delle informazioni necessarie alla connessione
13. Aggregates: generazione e computazione di aggregati a partire da dati storici o attuali
14. PubSub: definizione di un modello di comunicazione publish/subscribe in alternativa a quello client/server standard
100. Device Information Model: definizione di Information Model per dispositivi generici

2 Information modelling

Le specifiche di OPC UA forniscono un'infrastruttura per modellare informazioni, che può essere sfruttata per creare degli Information Model. La OPC Foundation mette già a disposizione dei modelli di base, che possono essere estesi dai fornitori in modo da aggiungere informazioni relative ai loro dispositivi. In questo modo, un client può accedere in maniera standard alle informazioni memorizzate in server OPC UA di venditori diversi.

Questo concetto non è esclusivo agli Information Model di dispositivi per l'automazione, ma può essere esteso ad altri scenari, come ad esempio il caricamento dati verso sistemi MES ed ERP.

I principi base dell'information modelling in OPC UA sono i seguenti:

- Utilizzo di tecniche object-oriented, come ereditarietà e gerarchie di tipi
- Accesso alle informazioni relative ai tipi di dato nello stesso modo in cui si accede alle istanze di tali tipi
- Creazione di una rete a maglia di nodi, in cui le informazioni sono collegate tra loro in più modi, permettendo di fornire diverse rappresentazioni a seconda della situazione
- Estendibilità delle gerarchie di tipi e delle tipologie di associazioni tra nodi
- Nessuna limitazione sulle possibilità di modellazione dell'informazione
- Modellazione eseguita sempre lato server

Gli argomenti relativi all'information modelling sono trattati nelle sezioni 3 e 5 delle specifiche di OPC UA [5].

2.1 Rappresentazione dei dati

2.1.1 Nodi e riferimenti

OPC UA introduce due concetti base per la modellazione delle informazioni: Node e Reference.

Un Node ha una NodeClass che ne definisce lo scopo (ad esempio istanza, tipo, ecc.). Per descrivere un nodo si utilizzano gli Attribute. Il set di attributi di un nodo è interamente determinato dalla sua NodeClass e non può essere esteso. Per modellare informazioni aggiuntive relative ad un nodo è necessario usare le Property. Un elenco di attributi comuni a tutte le NodeClass è riportato nella tabella 2.1.

Tabella 2.1: attributi comuni

Attributo	**DataType**	**Descrizione**
NodeId	NodeId	Identificatore univoco del nodo all'interno del server OPC UA
NodeClass	NodeClass	Identifica la NodeClass a cui il nodo appartiene, ad esempio Object o Method.
BrowseName	QualifiedName	Permette ai client di identificare il nodo in fase di navigazione del server OPC UA.
DisplayName	LocalizedText	Contiene il nome da utilizzare quando si mostra il nodo in un'interfaccia, pertanto ne è gestita la traduzione.
Description	LocalizedText	Attributo opzionale per la descrizione del nodo.
WriteMask	UInt32	Attributo opzionale che specifica quali attributi del nodo possono essere modificati dai client.
UserWriteMask	UInt32	Attributo opzionale che specifica quali attributi del nodo possono essere modificati dall'utente attualmente connesso al server. Il set dev'essere un sottoinsieme di quello indicato in WriteMask.

Una Reference è una connessione tra due nodi. Nella pratica, non è altro che un puntatore a un nodo. Non è possibile accedere in maniera diretta ad una Reference, occorre farlo in maniera indiretta accedendo al nodo che la contiene.
Una Reference non ha attributi o proprietà che la contraddistinguono; invece, per stabilire la semantica di una Reference (ossia *in che modo* due nodi sono collegati) un server OPC può definire dei ReferenceType, che vengono esposti nello spazio degli indirizzi come nodi veri e propri, in modo da permettere ai client di accedere alle relative informazioni.

I ReferenceType personalizzati supportano l'ereditarietà singola e devono avere esattamente un ReferenceType padre.
Oltre agli attributi comuni a tutte le tipologie di nodo, un ReferenceType dispone anche di quelli riportati nella tabella 2.2.
Nel caso in cui si vogliano definire caratteristiche aggiuntive per un'associazione tra due nodi, si sostituisce la Reference con un Proxy, ossia un oggetto che referenzia entrambi i nodi e contiene le proprietà necessarie.

Tabella 2.2: attributi specifici per un nodo ReferenceType

Attributo	**DataType**	**Descrizione**
IsAbstract	Boolean	Stabilisce se il ReferenceType può essere usato per creare riferimenti o se esiste al solo scopo di organizzare la gerarchia dei ReferenceType.
Symmetric	Boolean	Indica se il riferimento è simmetrico o meno. Un esempio di riferimento simmetrico è *sibling-of*, mentre uno asimmetrico è *parent-of*.
InverseName	LocalizedText	Specifica la semantica del riferimento nella direzione inversa. Ad esempio, l'InverseName del riferimento *parent-of* potrebbe essere *son-of*. Può essere valorizzato solo se il riferimento è asimmetrico e deve essere valorizzato se non è astratto.

2.1.2 Oggetti, variabili e metodi

Le NodeClass più importanti di OPC UA sono Object, Variable e Method.

Variabili

Un nodo di classe Variable rappresenta una variabile ti un certo tipo. Un client può leggere e scrivere il campo Value di una variabile, e può inoltre richiedere di essere notificato in seguito ad un suo cambiamento di valore.
La tabella 2.3 riporta i principali attributi dei nodi di tipo Variable.

Tabella 2.3: attributi specifici per un nodo Attribute

Attributo	**DataType**	**Descrizione**
Value	-	Valore della variabile. Il DataType è calcolato dalla combinazione degli attributi DataType, ValueRank e ArrayDimensions.
DataType	NodeId	Essendo i DataType rappresentati come nodi, questo campo contiene il NodeId del DataType dell'attributo.
ValueRank	Int32	Stabilisce se il valore è un array e, in tal caso, permette di specificarne la dimensione (monodimensionale, bidimensionale, ecc.)
ArrayDimensions	UInt32[]	Attributo opzionale che può essere valorizzato soltanto qualora il valore sia un array. Indica, per ogni dimensione dell'array, il numero di elementi contenuti.

Come è possibile dedurre dalla tabella, OPC UA supporta nativamente gli array multidimensionali. Un client può leggere, scrivere o sottoscrivere alle modifiche di un array o di una sua parte.

Metodi

Un nodo di classe Method rappresenta un metodo, ossia un'operazione che dev'essere eseguita in maniera relativamente rapida. Per procedimenti di elevata durata, avviati e controllati dal client, OPC UA usa invece il concetto di *programma*, analizzato al capitolo **2.2.3 - Information Model di OPC UA**. Dichiarare un metodo in OPC UA significa specificarne il nome e le proprietà InputArguments e OutputArguments, ossia definirne la signature, ma non l'implementazione.
La tabella 2.4 riporta le proprietà specifiche ai nodi di tipo Method.

Tabella 2.4: proprietà specifiche per un nodo di tipo Method

Proprietà	**DataType**	**Descrizione**

InputArguments	Argument[]	Proprietà opzionale che definisce i parametri del metodo. Se non è valorizzata, il metodo non ha parametri di input.
OutputArguments	Argument[]	Come InputArguments ma per l'output di un metodo.

Ogni nodo di tipo Argument contiene un nome, una descrizione e un DataType (che comprende anche gli attributi necessari a definire parametri di tipo array).

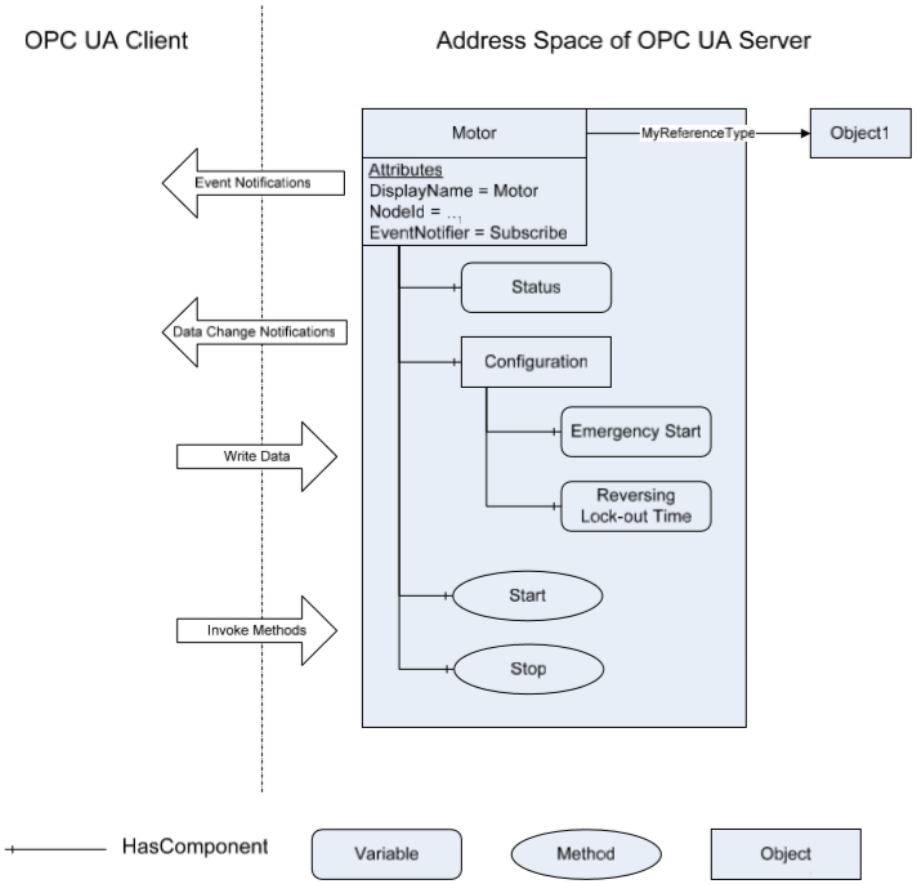
Oggetti

I nodi di classe Object sono utilizzati per strutturare lo spazio degli indirizzi. Un oggetto non contiene attributi oltre a quelli comuni a tutti i nodi, ma i dati che racchiude sono presentati tramite l'utilizzo delle variabili. Un oggetto può inoltre essere un EventNotifier, a cui un client può iscriversi per ricevere notifiche relative all'accadere di eventi.

Gli oggetti sono utilizzati per raggruppare variabili, metodi e altri oggetti. Metodi e variabili fanno sempre parte di almeno un oggetto e possono essere richiamati solo nel contesto dell'oggetto di appartenenza.

La figura 2.1 riporta un esempio di oggetto che contiene oggetti, variabili e metodi e genera eventi.

Figura 2.1: esempio di oggetto in OPC UA [6]



2.1.3 Tipi per oggetti e variabili

Un'importante caratteristica di OPC UA è il fatto che fornisce non soltanto informazioni sui tipi di dato (per esempio permettendo di sapere se un valore è intero o stringa), ma anche sui tipi di oggetto (permettendo quindi di sapere se un determinato tipo di oggetto dispone o meno di una certa proprietà).

Lo spazio degli indirizzi di OPC UA include le NodeClass ObjectType per definire tipi di oggetto e VariableType per definire i tipi di variabile. Non c'è una NodeClass per definire i tipi di metodo, in quanto un metodo è già classificabile in base a nome, parametri di input e output.

Per comodità, quando ci si riferisce sia a ObjectType che a VariableType per esprimere concetti comuni ad entrambi, si parla di TypeDefinition.

TypeDefinition

Una TypeDefinition può essere semplice o complessa. Un tipo semplice definisce solo la semantica dell'oggetto o della variabile, mentre uno complesso può stabilire anche la struttura dei nodi interni.

Una TypeDefinition può inoltre essere concreta o astratta (cioè non utilizzabile per istanziare oggetti o variabili).

Le TypeDefinition supportano l'ereditarietà multipla, anche se è consigliato assegnare a ciascuna esattamente un super-tipo, ad eccezione di BaseObjectType e BaseVariableType, che formano rispettivamente la radice della gerarchia degli oggetti e quella delle variabili.

Sottotipi di una VariableType possono restringere l'elenco dei tipi di dato ammessi da una variabile, ma non ampliarlo.

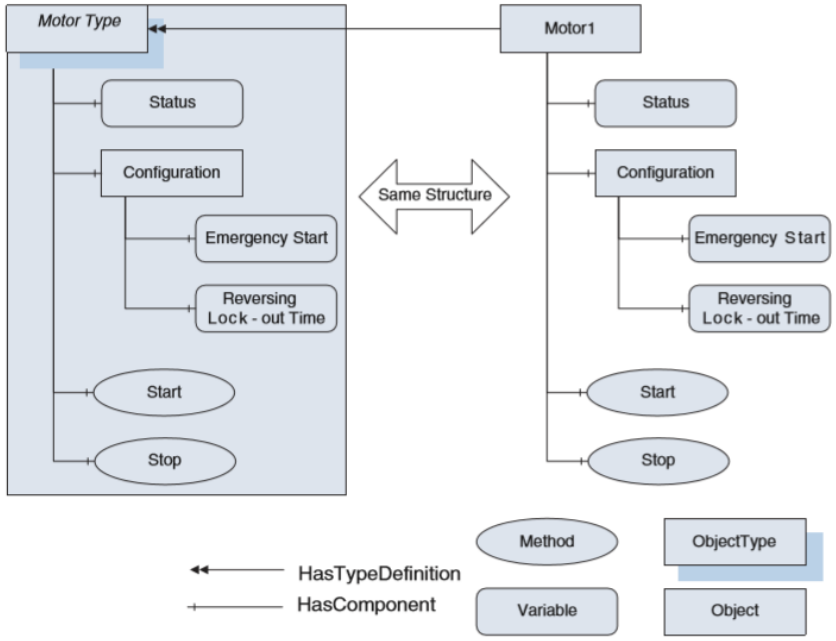
TypeDefinition complesse

Gli ObjectType complessi consentono al server di esporre informazioni relative alla struttura degli oggetti, permettendo ai client di organizzare le proprie applicazioni sulla base di tale disposizione.

Un ObjectType può essere definito in un progetto, nella libreria di un fornitore o anche in un Information Model standard, ed è poi possibile riutilizzarlo ed accedervi in più punti.

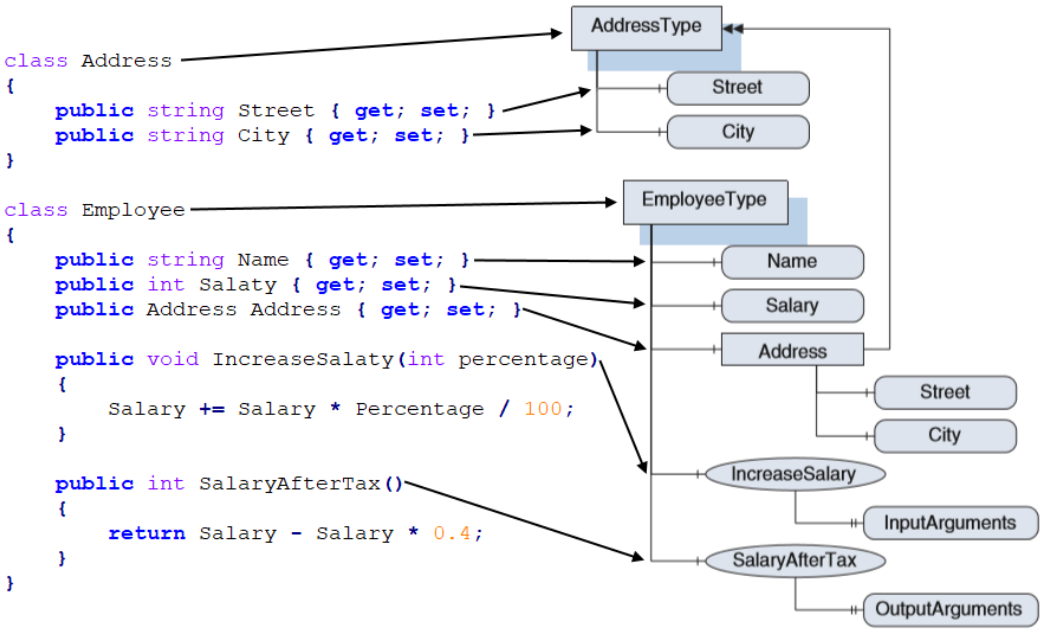
La figura 2.2 riporta un esempio di ObjectType complesso e di una sua istanza.

Figura 2.2: ObjectType complesso e sua istanza [6]



Così come nei linguaggi di programmazione a oggetti una classe definisce variabili, metodi e oggetti contenuti dalle sue istanze, in OPC UA un ObjectType ha delle InstanceDeclaration che ne definiscono il contenuto. La figura 2.3 riporta un esempio di mapping da classe C# a ObjectType OPC UA.

Figura 2.3: mapping di una classe C# su un ObjectType OPC UA



OPC UA permette inoltre di estendere specifiche istanze di variabili e oggetti senza necessariamente creare un sottotipo. Ad esempio, una certa istanza di Employee potrebbe contenere un oggetto chiamato SecondaryAddress, cosa tipicamente non consentita nella programmazione ad oggetti senza la creazione di una classe figlio.
I VariableType complessi funzionano in maniera analoga agli ObjectType complessi, con la differenza che possono solo contenere le InstanceDeclarations di variabili, e non di oggetti o metodi.

InstanceDeclaration e ModellingRule

In fase di creazione di un oggetto o di una variabile, una InstanceDeclaration può fare riferimento a nodi già esistenti. In particolare, può riferirsi anche al nodo della InstanceDeclaration della TypeDefinition, creando così una variabile statica. Variabili e metodi non sono infatti proprietà di un oggetto, ma possono essere condivisi. All'eliminazione di un oggetto, variabili e metodi al suo interno vengono cancellati solo se nessun oggetto si riferisce più ad essi.
Ogni InstanceDeclaration ha una ModellingRule, che può essere di tre tipi principali:

- *Mandatory*: la valorizzazione dell'InstanceDeclaration è obbligatoria in fase di dichiarazione di un oggetto
- *Optional*: la valorizzazione è opzionale
- *Constraint*: la valorizzazione va effettuata tenendo conto di certi requisiti (un esempio è la restrizione della cardinalità di una InstanceDeclaration)

È possibile creare nuove ModellingRule a partire dalle tre sopra riportate.
Quando si esegue *subtyping* di tipi complessi, le InstanceDeclaration non vengono duplicate a meno che non debbano essere ridefinite dal sotto-tipo. Inoltre, tutte le ModellingRules del super-tipo devono essere rispettate e possono al più essere ristrette.

2.1.4 Data Variable e Property

OPC UA definisce due tipologie di variabile: Data Variable e Property.
Le Data Variable sono usate per rappresentare dati misurabili relativi ad un oggetto. Possono essere semplici o complesse e sono dotate di un DataType. Dovendo appartenere ad un oggetto, devono essere referenziate da un'associazione HasComponent proveniente da un tipo Object, ObjectType, Variable o VariableType.

Un esempio di Data Variable semplice è la temperatura misurata da un dispositivo, mentre una Data Variable complessa potrebbe contenere tutti i valori rilevati da un sensore, con sotto-variabili per ogni dato.
Le Property sono usate per descrivere le caratteristiche di un nodo (non necessariamente un oggetto) che non sono comprese nei suoi attributi. Sono nodi di tipo PropertyType e non possono essere complesse, ma solo semplici. Un nodo è collegato alle sue proprietà tramite la Reference HasProperty.
Esempi di proprietà sono InputArguments e OutputArguments di un metodo e l'unità di misura di un sensore.

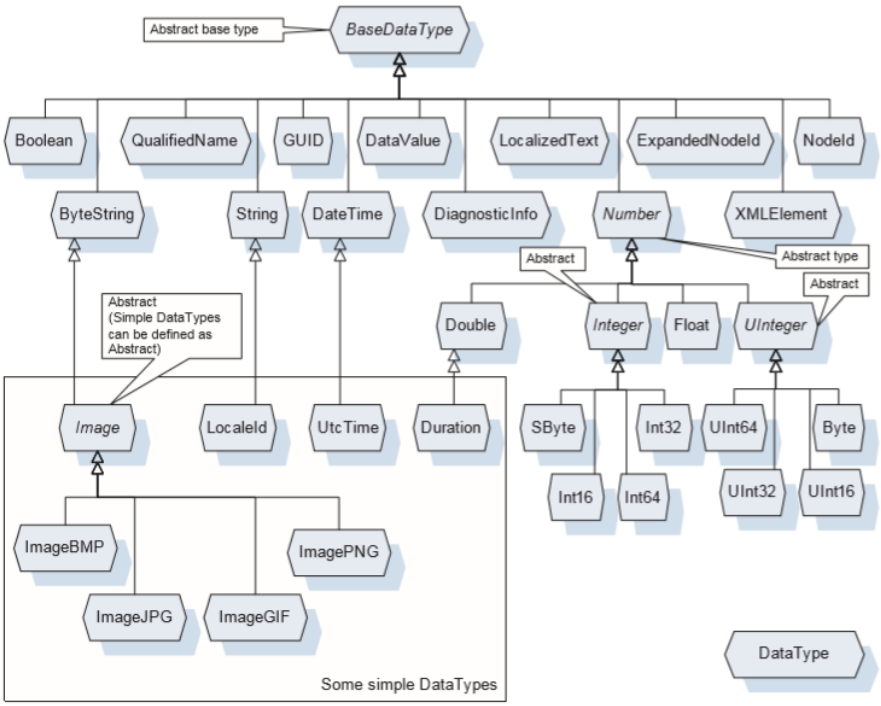
2.1.5 DataTypes

Tutti gli attributi, eccetto Value di Variable e VariableType, hanno un tipo di dato fisso.
I DataType sono rappresentati come nodi della NodeClass DataType e supportano l'ereditarietà singola. Tutti i DataType devono avere esattamente un super-tipo, con l'eccezione di BaseDataType, che costituisce la radice della gerarchia.
L'organizzazione dei DataType è esposta nello spazio degli indirizzi, in modo da permettere ai client di accedere alle informazioni relative alla gerarchia dei DataType dei server con cui devono interagire.
OPC UA distingue i DataType in quattro categorie:

1. *Built-in*: un set di DataType interamente definito da OPC UA e non estendibile; comprende i tipi di base come Int32 e Boolean, ma anche tipi di dato specifici di OPC UA come NodeId e LocalizedText
2. *Simple*: sono sotto-tipi dei DataType Built-in; un esempio è la durata di un intervallo in millisecondi come sotto-tipo di Double
3. *Enumeration*: rappresentano un set di nomi mappati su valori Int32; un esempio è l'attributo NodeClass
4. *Structured*: rappresentano dati complessi e strutturati, definiti dallo standard (come il tipo Argument per i parametri di un metodo) o dall'utente

La figura 2.4 riporta la struttura della gerarchia dei DataTypes built-in e simple.

Figura 2.4: gerarchia dei DataType built-in e simple [6]



Sono ora analizzati alcuni dei DataTypes più importanti all'interno della logica di OPC UA.

DataType strutturati

I DataType strutturati sono derivati del DataType astratto Structure. Una caratteristica importante di questi tipi di dato è che sono codificati dal mittente in fase di invio e decodificati dal destinatario in ricezione. Anche i tipi di dato built-in seguono tale procedura, ma per questi ultimi la modalità di codifica è ben definita nelle specifiche di OPC UA e non dev'essere quindi trasmessa in rete. Questo non si applica per i dati strutturati: il server deve definire ed esporre nel suo spazio degli indirizzi almeno un tipo di codifica per ogni DataType structured, in modo da permettere ai client di ottenere informazioni a riguardo e selezionare il tipo di codifica che intendono applicare. Ogni nodo rappresentante un DataType strutturato punta quindi ad almeno un nodo di tipo DataTypeEncoding.

Quando si valuta se utilizzare un DataType strutturato o meno, può essere importante tenere conto dell'overhead dovuto alla trasmissione in rete delle informazioni di codifica. La differenza di prestazioni è spesso trascurabile in quanto i dati per la codifica vengono memorizzati in cache dal client e non devono quindi essere trasmessi di frequente.

DataType NodeId

Il DataType NodeId è fondamentale per OPC UA in quanto identifica univocamente un nodo. Si compone di tre parti: NamespaceIndex, DataType e Identifier.
Il campo NamespaceIndex è un puntatore al NamespaceArray di un server OPC UA e permette di associare ad un indice numerico un Namespace URI come "http:// opcfoundation.org/UA". Utilizzando il NamespaceIndex invece dell'URI si rimuove un overhead non indifferente nella comunicazione.
DataType indica il tipo di dato del terzo campo, che può essere un valore numerico, un GUID o una stringa.

DataType QualifiedName

Il DataType QualifiedName è usato per il campo BrowseName dei nodi. È formato dal NamespaceIndex seguito da una stringa che identifica il nodo nello spazio degli indirizzi del server.

2.1.6 View

I nodi di NodeClass View sono utilizzati per raggruppare i nodi all'interno di uno spazio degli indirizzi di dimensioni elevate. Permettono al server di fornire viste apposite in base al caso d'uso. Per esempio, un server potrebbe avere una View dedicata alle operazioni di manutenzione.
Una View è presentata come un nodo nello spazio degli indirizzi, che funge da entry point per tutti i contenuti collegati direttamente o indirettamente ad esso.
Come gli oggetti, una View può fungere da EventNotifier.

2.1.7 Eventi

Gli eventi vengono inviati ai client tramite notifica, dopo che questi ultimi si sono iscritti ad un EventNotifier. Gli eventi non sono generalmente visibili nello spazio degli indirizzi (eccezioni a questa regola sono Alarms e Conditions), ma fanno riferimento ad un EventType, che fa parte di una gerarchia di tipi accessibile ai client. L'EventType

definisce le proprietà di un evento e permette ai client di creare filtri, stabilendo quali eventi vogliono ricevere in base al tipo e a quali proprietà di un evento sono interessati.
Gli EventNotifier si occupano di notificare i client dell'accadere di determinati eventi, ma non sono necessariamente la fonte di tali eventi. Di conseguenza, all'interno di un oggetto Event sono presenti campi per identificarne la sorgente.

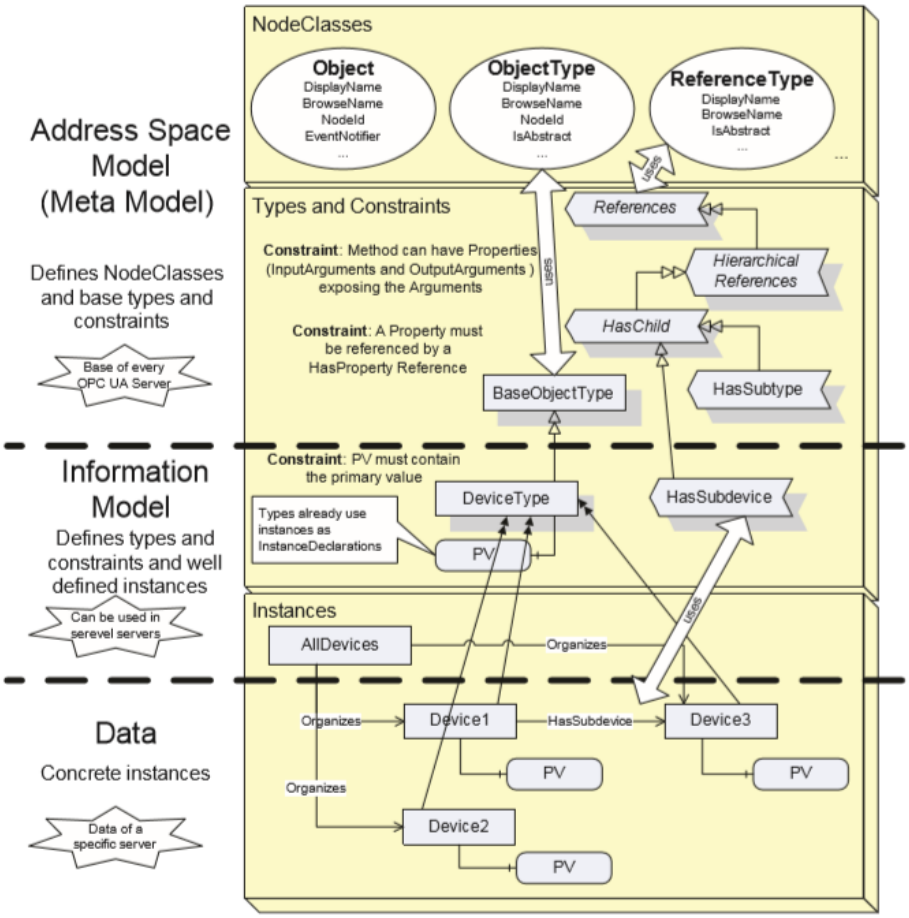
2.1.8 Accesso allo storico

Vi sono tre categorie per quanto riguarda lo storico in OPC UA: storico dei dati, degli eventi e della struttura dell'Address Space.
OPC UA consente, attraverso appositi servizi (**3.2.9 - Accesso allo storico dati di un server**), la visualizzazione e la modifica dello storico dei dati relativi all'attributo Value di una variabile. Ogni variabile ha tre attributi che riguardano il suo storico: AccessLevel e UserAccessLevel indicano se lo storico è accessibile, mentre Historized indica se i valori storici vengono effettivamente raccolti.
Lo storico degli eventi può essere reperito tramite gli EventNotifier. A differenza delle variabili, non vi è un metodo standard per esporlo nell'Address Space di un server.
Per quanto riguarda lo spazio degli indirizzi, OPC UA permette opzionalmente ai server memorizzare la proprietà NodeVersion dei nodi e generare eventi di tipo ModelChangeEvent. Qualora il server non implementasse tali funzioni, un client può sempre tenere traccia dello storico dell'Address Space eseguendo query periodiche sulla sua struttura attraverso appositi servizi (**3.2.3 - Ricerca di informazioni sull'Address Space**).

2.1.9 Address Space Model e Information Model

L'insieme delle NodeClass, dei loro attributi e dei nodi standard come TypeDefinition e Input/OutputArguments definiti in OPC UA formano il meta modello del protocollo, detto Address Space Model.
Un Information Model estende gli elementi dell'Address Space Model per definire tipi, vincoli e istanze relativi al dominio di appartenenza. I dati concreti di un server sono poi creati basandosi sull'Information Model, come mostrato in figura 2.4.

Figura 2.4: Address Space Model, Information Model e dati [6]



2.2 Information Model standard

2.2.1 Cosa è definito in un Information Model

I concetti che possono essere definiti in un Information Model sono riportati nella tabella 2.5, insieme ad un esempio per ogni elemento.

Tabella 2.5: quali elementi si possono specificare in un Information Model

Concetto	**Esempio**
ObjectType	ObjectType che rappresenta una tipologia di dispositivo
ModellingRule	Restrizione di cardinalità

EventType	Tipo di evento per gli allarmi
VariableType	Tipo di variabile con proprietà come l'unità di misura
Semantica di una proprietà	Proprietà per memorizzare l'unità di misura di una DataVariable
Semantica di un metodo	Definizione della signature di un metodo
ReferenceType	Una Reference che indica che due dispositivi comunicano tra di loro
DataType	Lo stato di un dispositivo
DataTypeEncoding	Encoding per i dati rilevati da un dispositivo
Object	Oggetto come entry point per le informazioni di diagnostica del server
View	Una View contenente tutti i dispositivi istanziati sul server
Variable	Variabile contenente lo stato del server
Method	Un metodo per arrestare il server
Constraint	Tutti gli eventi generati dai dispositivi devono essere presentati come nodi dal server

2.2.2 Come viene definito un Information Model

Al momento del rilascio, OPC UA non includeva uno standard per la definizione di un Information Model. Ad esempio, DataAccess e Program erano definiti attraverso documenti di testo, in cui i nodi erano inseriti in tabelle e le Constraint in campi di testo. Questa procedura aveva lo svantaggio principale di rendere impossibile l'automatizzazione del processo di popolazione dell'Address Space di un server.
Nel 2015, la OPC Foundation ha rilasciato il Model Compiler, uno strumento in grado di convertire un file XML contenente la definizione di un Information Model in codice ANSI C o C# per servizi, DataType, ErrorCode, ecc. e file CSV con nodi, attributi, ecc.
Nella prossima sezione vengono introdotti brevemente gli Information Model definiti nelle specifiche di OPC UA.

2.2.3 Information Model di OPC UA

Information model di base

L'Information Model di base è definito nelle sezioni 3 e 5 delle specifiche di OPC UA [5]. Le figure 2.5 e 2.6 rappresentano rispettivamente la gerarchia delle TypeDefinition di base e la struttura di oggetti e variabili all'interno dell'Information Model.

Questo modello è progettato per l'estendibilità. In particolare, nuovi oggetti devono ereditare da BaseObjectType, nuove variabili da BaseVariableType e nuovi tipi di dato da BaseDataVariableType

Figura 2.5: gerarchia delle TypeDefinition nell'Information Model base di OPC UA [6]

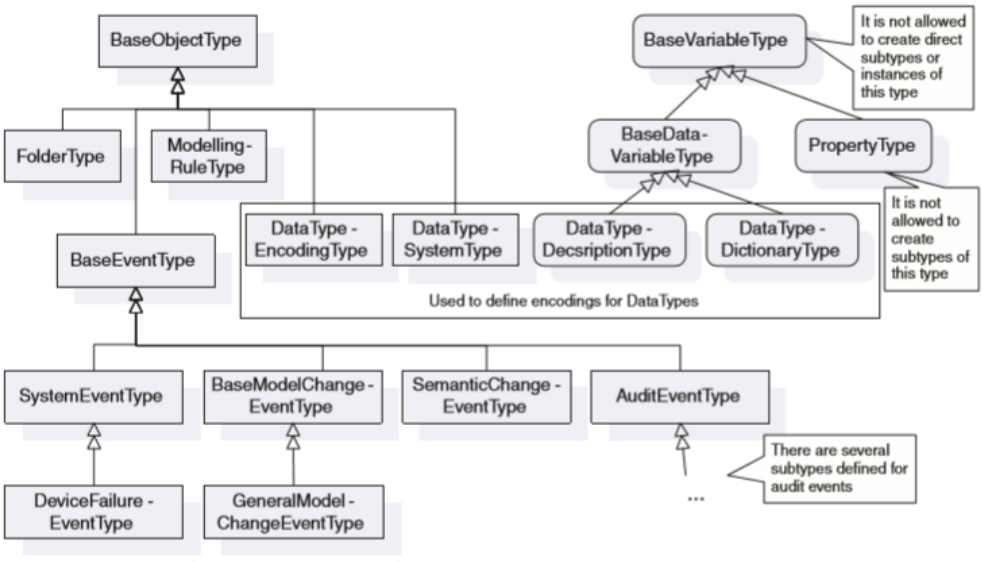
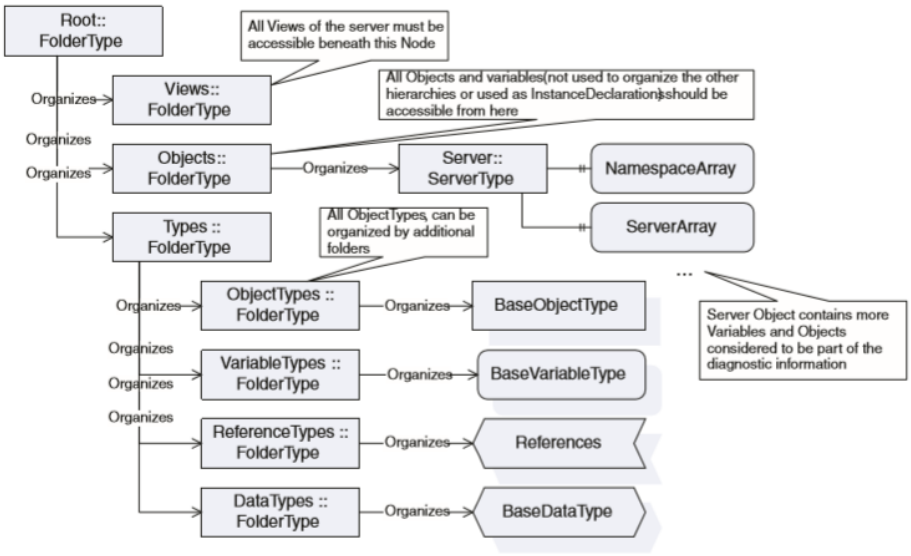


Figura 2.6: oggetti e variabili nell'Information Model di base [6]



Capabilities and Diagnostics

L'Information Model Capabilities and Diagnostics contiene tutte le informazioni riguardo a stato del server, risorse disponibili, client connessi e chiamate a servizi. Fornisce inoltre degli entry point a cui i venditori possono aggiungere informazioni specifiche.

Data Access

L'Information Model per il Data Access è definito nella sezione 8 delle specifiche di OPC UA [5] e stabilisce quali sono i tipi di variabile standard e le loro proprietà.

DataItemType è usato per rappresentare dati arbitrari per l'automazione. Per questo tipo sono definite due proprietà aggiuntive: Definition, che contiene una stringa con la spiegazione di come il valore viene calcolato, e ValuePrecision, ossia la precisione della misurazione.

Per quantità fisiche che cambiano continuamente nel tempo si usa AnalogItemType.

Historical Access and Aggregates

Historical Access and Aggregates descrive le modalità di rappresentazione e di accesso alle informazioni relative allo storico di dati ed eventi. Estende il modello Capabilities and Diagnostics definendo dove trovare informazioni per la configurazione dei dati storici e come EventNotifier e variabili devono presentare tale configurazione. Stabilisce inoltre le modalità per il raccoglimento dei dati storici.

State Machine

Questo Information Model definisce come presentare macchine a stati nello spazio degli indirizzi di OPC UA. In particolare, il tipo StateMachineType espone solo informazioni sullo stato attuale della macchina, mentre FiniteStateMachineType mette a disposizione tutte le informazioni relative a stati, transizioni, condizioni per il verificarsi di una transizione e azioni da intraprendere al suo accadere.

Programmi

Un metodo è invocato da un client, eseguito dal server e il suo risultato è restituito al client in tempi relativamente rapidi. In contrasto, un programma è usato per operazioni più lunghe e complesse, può essere suddiviso in stati e restituire risultati intermedi. I programmi sono rappresentati in OPC UA attraverso un sotto-tipo di FiniteStateMachineType, chiamato ProgramType.

Alarms and Conditions

Questa specifica definisce un Information Model estendibile per condizioni, conferme ed allarmi.

Device Information Model

Questo Information Model è contenuto nella sezione 100 delle specifiche di OPC UA [5] e definisce delle entità per la rappresentazione di dispositivi generici, prevedendo funzioni per rilevamento dati, attuazione, comunicazione e controllo.

3 Servizi

3.1 Introduzione ai servizi

I servizi sono il metodo usato da un client OPC UA per accedere ai dati dell'Information Model di un server OPC UA. Un servizio stabilisce quindi la struttura dell'interfaccia di comunicazione tra due applicazioni OPC UA.
La sezione 4 delle specifiche di OPC UA [5] definisce i servizi disponibili in maniera astratta e indipendente dal protocollo per il trasporto dei dati e dall'ambiente di programmazione scelto per un'applicazione.
La definizione di un servizio si compone di due parti: messaggio di richiesta e messaggio di risposta. Il client invia una richiesta al server, che la processa e restituisce una risposta. Tutte le service call sono asincrone di default, per cui è possibile per un'applicazione client processare altre informazioni nell'attesa di ricevere la risposta del server alla chiamata di un servizio. La maggior parte delle implementazioni di OPC UA mette a disposizione, per convenienza, anche una versione sincrona delle service call.

3.1.1 Gestione dei timeout

Il protocollo OPC UA è pensato per essere applicabile su sistemi distribuiti, pertanto lo scambio dati tra applicazioni OPC UA può avvenire in contesti molto diversi. Può capitare che due applicazioni comunichino sulla stessa macchina scambiandosi messaggi tra processi, oppure su sistemi diversi nella stessa rete locale, o ancora su nodi geograficamente molto distanti. OPC UA mette quindi a disposizione dei client la possibilità di stabilire timeout individuali per ogni service call, in modo da potersi adattare alle diverse tempistiche di comunicazione.

3.1.2 Request Header e Response Header

La tabella 3.1 riporta i parametri più importanti contenuti nel Request Header di tutti i servizi OPC UA, mentre la tabella 3.2 elenca i parametri del Response Header.

Tabella 3.1: parametri principali del Request Header

Parametro	**Descrizione**
AuthenticationToken	Identificatore segreto sella sessione stabilita tra client e server
Timestamp	Timestamp di invio della richiesta
TimeoutHint	Timeout che il client ha deciso di adottare. Utile per permettere al server di cancellare le chiamate in esecuzione qualora non siano state completate allo scadere del timeout
ReturnDiagnostic	Indica se il client ha richiesto o meno informazioni di diagnostica dettagliate

Tabella 3.2: parametri del Response Header

Parametro	**Descrizione**
ServiceResult	Codice di risultato della chiamata
Timestamp	Timestamp di invio della risposta
ServiceDiagnostics	Informazioni di diagnostica dettagliate fornite su richiesta del client

3.1.3 Gestione degli errori

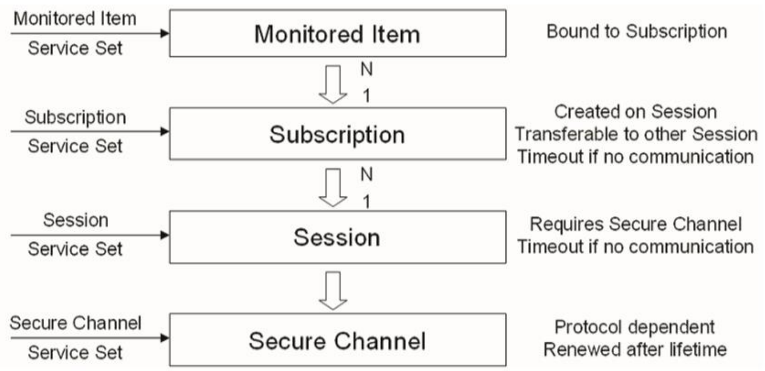
Durante l'uso dei servizi in OPC UA, gli errori che possono verificarsi sono generalmente classificabili su due livelli. Al primo si hanno gli errori di comunicazione che accadono quando la connessione tra client e server viene interrotta in fase di trasmissione dati. Al secondo vi sono invece gli errori nell'esecuzione delle operazioni, che accadono ad esempio quanto un client fornisce parametri errati, come il NodeId di un nodo non disponibile.

La gestone degli errori da parte dei servizi è realizzata attraverso l'uso di due campi. *StatusCode* è un tipo di dato UInt32 in cui i 16 bit più significativi rappresentano il codice numerico dell'errore, mentre gli altri 16 sono flag aggiuntivi che forniscono maggiori dettagli.
DiagnosticInformation è una struttura contenente informazioni aggiuntive sull'errore, come la descrizione ed eventuali ErrorCode specifici del venditore.
Le informazioni relative agli errori sono distribuite su due livelli. Il primo livello riporta il risultato della Service call, mentre il secondo è la lista dei risultati delle singole operazioni contenute nella chiamata.

3.1.4 Contesto per la comunicazione

I servizi OPC UA sono stateful e perché un client possa eseguire una service call deve aver prima creato un contesto di comunicazione con il server su più livelli, riportati nella figura 3.1.

Figura 3.1: contesto di comunicazione [6]



Secure Channel è il livello più basso, che si occupa di garantire la sicurezza del mezzo di comunicazione. Il canale ha una validità temporale, di durata stabilita in fase di creazione, che dev'essere periodicamente rinnovata in modo da non compromettere la sicurezza della trasmissione.
Session è il collegamento tra due applicazioni all'interno di un Secure Channel. Una sessione non è legata al Secure Channel sottostante e può essere trasferita su un altro canale in caso di necessità. Ogni sessione ha un timeout, che permette al server di liberare

le risorse allocate dopo un periodo di tempo specificato. Il timeout viene resettato ad ogni service call effettuata nella sessione.
Una Subscription è il contesto necessario per lo scambio dati tra client e server. Più Subscription possono essere create all'interno di una sessione, ma sono indipendenti da essa e si possono trasferire da una sessione all'altra. Hanno un timeout che funziona in maniera simile a quello delle sessioni.
Un Monitored Item indica un attributo o un evento da monitorare ed è legato a una Subscription.

3.2 Servizi principali di OPC UA

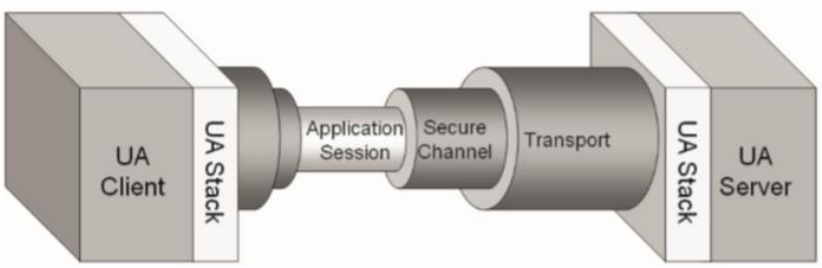
3.2.1 Ricerca dei server

Il servizio Discovery è usato dai client OPC per trovare i server disponibili e ottenere informazioni riguardo ai loro Endpoint.
Il servizio RegisterServer è usato da un server per registrarsi su un Discovery Server fornendo i suoi Endpoint. Questo servizio richiede la presenza di una connessione sicura per poter essere utilizzato e va eseguito periodicamente dal server per comunicare la sua disponibilità. Viene inoltre richiamato in fase di arresto per indicare che il server sta per andare offline.
Un Endpoint definisce i protocolli di rete e le misure di sicurezza che un client deve adottare per collegarsi al server e instaurare un Secure Channel e una sessione. Tramite il servizio GetEndpoints un server deve restituire l'elenco completo dei suoi Endpoint.
Il servizio FindServers è implementato dai Discovery Server per restituire l'elenco dei server registrati e da ogni server per restituire se stesso.

3.2.2 Gestione delle connessioni

Per assicurare una comunicazione sicura, affidabile e flessibile, OPC UA richiede lo stabilimento di vari canali di trasferimento dati, riportati nella figura 3.2.

Figura 3.2: i differenti canali di comunicazione di OPC UA [6]



Il servizio SecureChannel è usato per la creazione dei canali Transport e SecureChannel mostrati nella figura 3.2.
CreateSession è il servizio per la creazione dell'Application Session. Quando invocato, il server restituisce al client un identificatore di sessione. La sessione dev'essere poi convalidata con il servizio ActivateSession, che permette anche di cambiarne i parametri (compreso trasferirla su un altro canale), e può essere eliminata con il servizio CloseSession.

3.2.3 Ricerca di informazioni sull'Address Space

Il servizio Browse è invocato dal client per navigare nell'Address Space. Il server riceve una lista di nodi e restituisce, per ognuno, l'elenco dei nodi ad esso connessi, che comprende tutti gli attributi necessari per rappresentare il nodo nell'Address Space, come NodeClass e TypeDefinition.
Il servizio TranslateBrowsePathsToNodeIds permette, partendo da un ObjectType, di ottenere un elenco degli oggetti di quel tipo presenti su un server.
Grazie al servizio RegisterNodes è possibile ottimizzare operazioni cicliche di accesso al valore di una variabile o di chiamata a un metodo. La registrazione è consiste in una chiamata del client che invia un elenco di NodeId e in una risposta del server contenente degli handle numerici da usare nelle comunicazioni al posto dei NodeId. In questo modo, per eseguire le chiamate successive sarà necessario inviare una quantità inferiore di dati. Inoltre, essendo il server a conoscenza di quali nodi saranno utilizzati di frequente, può predisporre quanto necessario per ottimizzare l'accesso a tali nodi.
Con il servizio UnregisterNodes il client comunica al server che non ha più la necessità di accedere ai nodi tramite handle.

3.2.4 Ricerca di informazioni su Address Space complessi

Il servizio Browse è perfetto per navigare attraverso aree conosciute di un Address Space di dimensioni contenute. Tuttavia, gli Address Space di server con rich Information Model possono contenere milioni di nodi e quelli di sistemi MES e ERP sono molto dinamici in quanto contengono, ad esempio, informazioni relative agli ordini di lavoro attualmente in fase di produzione.
Per la ricerca di informazioni all'interno di Address Space con un alto livello di complessità, OPC UA mette a disposizione una feature detta Query. Mentre il servizio Browse prende come parametro un elenco di nodi da utilizzare come entry point, Query utilizza dei filtri per la ricerca delle informazioni.
Tramite il servizio QueryFirst viene avviata una Query. Se il numero di risultati è particolarmente elevato, ne viene restituita al client solo una parte. Per l'accesso ai risultati rimanenti si deve usufruire del servizio QueryNext.

3.2.5 Modifica dell'Address Space

I servizi della categoria NodeManagement abilitano i client OPC UA alla creazione e cancellazione di nodi e riferimenti nell'Address Space di un server OPC UA.
In questo set sono compresi i seguenti servizi:

- AddNodes per la creazione di nodi
- AddReferences per la creazione di riferimenti tra nodi
- DeleteNodes per la cancellazione di nodi
- DeleteReferences per la cancellazione di riferimenti

3.2.6 Lettura e scrittura di dati

I servizi Read e Write non permettono solo di leggere e scrivere i campi Value di una variabile, ma anche di accedere ai metadati di attributi e nodi nello spazio degli indirizzi. L'alternativa principale per l'accesso ai dati è la ricezione di notifiche al cambiamento dei valori, da utilizzare in caso di accessi ciclici.

Il servizio Read è richiamato da un client fornendo un elenco di NodeIds e di AttributeIds. Il server restituisce per ogni identificatore i valori corrispondenti.
Il servizio Write permette di scrivere su attributi, nodi e subset di questi elementi (ad esempio su parte di un array).

3.2.7 Sottoscrizione a cambi di dato ed eventi

Un client può sottoscrivere a tre tipi diversi di informazione da un server OPC UA. Una Subscription è usata per raggruppare sorgenti di informazioni. Una sorgente è gestita da un Monitored Item, mentre l'unità minima di informazione è la notifica. Una Subscription può contenere Monitored Items, notifiche o altre Subscription e continua a notificare il client fino a quando essa stessa o gli elementi al suo interno non vengono eliminati.
Perché il meccanismo di notifica sia sicuro ed affidabile vanno rispettati i seguenti requisiti:

- Invio delle notifiche da parte del server con il servizio Publish
- Invio di ping periodici da client a server e viceversa, per verificare lo stato della connessione quanto non ci sono notifiche da inviare (sempre tramite il servizio Publish)
- Identificazione delle notifiche tramite numeri di sequenza e invio di tali numeri insieme alle notifiche
- Invio di ACK da client a server per i numeri di sequenza ricevuti
- Re-invio delle notifiche non recapitate tramite il servizio Republish

Creazione e gestione di Monitored Items

Per la gestione delle Subscriptions, OPC UA mette a disposizione i seguenti servizi:

- CreateSubscription: gestisce la creazione di una Subscription e definisce le impostazioni iniziali
- ModifySubscription: permette di modificare le impostazioni di una Subscription
- TransferSubscription: permette di trasferire una o più Subscription su una nuova sessione
- DeleteSubscription: cancella una Subscription

Per la creazione, la modifica e la cancellazione dei MonitoredItems si usano rispettivamente i servizi CreateMonitoredItem, ModifyMonitoredItem e DeleteMonitoredItem.

Monitorare i cambi di valore

OPC UA fornisce due metodi per monitorare cambiamenti sui metadati. Il primo e il più utilizzato è pensato appositamente per il cambio di valore dell'attributo Value delle proprietà, che solitamente avviene di frequente. Il secondo è l'evento SemanticsChange, che scaturisce dai cambi nella semantica dell'Address Space di un server.

Monitorare eventi

L'unico modo di ricevere notifiche relative a eventi di un server è la creazione di un Event Monitored Item in una Subscription. La differenza principale da un normale Monitored Item è nel modo in cui il client specifica il subset di informazioni da ricevere. Mentre per i Monitored Item relativi a dati è possibile scegliere esattamente una variabile, per quelli relativi a eventi vanno specificati dei filtri. Questi filtri sono espressi attraverso due parametri:

- SelectClauses permette di specificare i campi di interesse degli eventi
- WhereClause limita le notifiche agli eventi che corrispondono ai criteri indicati (ad esempio un certo EventType o uno specifico SourceNode)

Monitorare dati aggregati

Attraverso gli Aggregate Monitored Items un client può chiedere al server di essere notificato a intervalli regolari con un aggregato relativo ai valori di più variabili.

3.2.8 Chiamata di metodi definiti dal server

OPC UA permette ai server di esporre metodi personalizzati nell'Address Space, che possono essere invocati dai client tramite il servizio Call. Questo servizio è ottimizzato per l'esecuzione di più metodi con una sola richiesta, in modo da ottimizzare il numero di messaggi trasmessi, ma gli SDK OPC UA offrono generalmente anche una versione semplificata da utilizzare per l'invocazione di un singolo metodo.

3.2.9 Accesso allo storico dati di un server

La differenza principale tra l'accesso ai dati in tempo reale, realizzato attraverso i servizi Read, Write e Subscription, e quello ai dati storici, è la possibilità di specificare intervalli temporali di interesse su cui effettuare la ricerca.
I due servizi utilizzati per l'accesso allo storico dati sono HistoryRead, per la lettura, e HistoryUpdate per l'inserimento, la modifica e la cancellazione di informazioni storiche. Entrambi utilizzano un parametro che permette di indicare il tipo di operazione, i nodi coinvolti e il dominio temporale di interesse.
Per HistoryRead, questo parametro prende il nome di HistoryReadDetails e può assumere i seguenti DataType:

- ReadRawModified per la lettura degli attributi Value delle variabili specificate nel range temporale indicato
- ReadProcessed per ottenere dati aggregati da un insieme di valori in uno specifico intervallo di tempo
- ReadAtTime per ottenere un elenco di valori corrispondenti ad una serie di timestamp
- ReadEvent per ottenere un elenco di eventi a partire da un range temporale

Il servizio HistoryUpdate utilizza invece il parametro HistoryUpdateDetails, il cui DataType può essere:

- UpdateData per inserire, rimpiazzare e aggiornare dati storici
- UpdateEvent per inserire, rimpiazzare e aggiornare eventi storici
- DeleteRawModified per la cancellazione di determinati valori nel range temporale indicato
- DeleteAtTime per cancellare una serie di valori ai timestamp specificati
- DeleteEvent per la cancellazione di eventi

4 Tecnologia

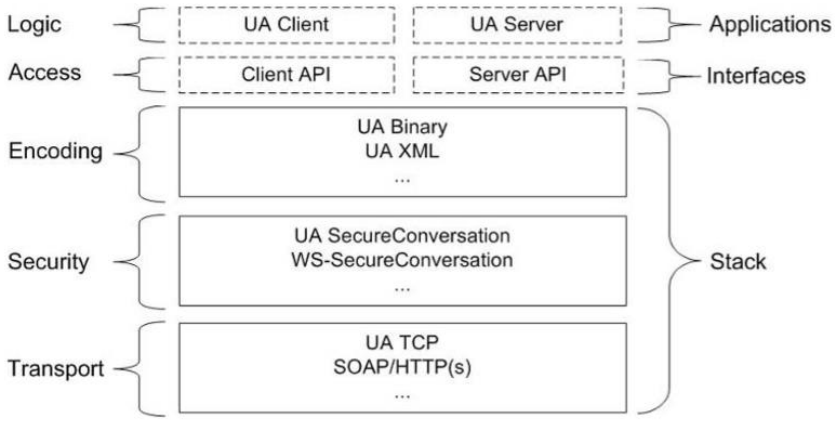
Nello sviluppo di uno standard, in fase di scelta della tecnologia da utilizzare bisogna tenere conto delle esigenze di tutte le possibili categorie di utilizzatori. Per il trasferimento dati, a volte può essere preferibile sacrificare parte della sicurezza in favore di prestazioni ed affidabilità, mentre in altri casi non è così. Dev'essere preso in considerazione anche lo scenario in cui una tecnologia adottata viene dismessa.

Per i motivi sopra citati, OPC UA definisce servizi (**3 - Servizi**) e concetti in maniera astratta, quindi procede a mapparli su tecnologie specifiche.

Come mostrato in figura 4.1, OPC UA è diviso in stack organizzati per funzionalità. I tre stack per cui è necessario eseguire mapping su una tecnologia sono: Encoding, Security e Transport. Per ogni stack, OPC UA, la sezione 6 delle specifiche di OPC UA [5] definisce alcune implementazioni, analizzate in questo capitolo.

Per sviluppare un prodotto conforme alle specifiche OPC è necessario implementare almeno una di queste tecnologie per ciascun livello.

Figura 4.1: struttura a stack di OPC UA e mapping su specifiche tecnologie [6]



4.1 Codifica dei dati

La codifica dei dati consiste nella serializzazione del messaggio di un servizio, comprendente anche parametri di input e di output, in un formato ottimizzato per la trasmissione in rete.

Per questo compito, OPC UA adotta due tecnologie: OPC UA Binary e XML. Con entrambe viene stabilito un metodo di codifica per tutti i tipi di dato Built-in, che permette anche di codificare tipi di dato più complessi composti da aggregati di dati Built-in.
Un altro aspetto in comune è l'utilizzo di ExtensionObject, un contenitore per dati complessi che definisce particolari modalità di codifica. Questa funzionalità torna utile in due ambiti:

- Invio di informazioni relative al tipo di codifica
- Trasferimento di dati proprietari codificati in maniera personalizzata

4.1.1 OPC UA Binary

OPC UA Binary è la tecnologia per il trasferimento dati da prendere in considerazione quando è importante mantenere performance elevate e introdurre un overhead minimo nella comunicazione, ad esempio per applicazioni in esecuzione su dispositivi embedded. Utilizza una serializzazione binaria per la codifica dei tipi di dato Built-in, che è il metodo più efficiente di trasferimento dati tra sistemi.
Tipi di dato complessi sono codificati combinando il set di primitive usato per i DataType Built-in.
La figura 4.2 riporta un esempio di codifica OPC UA Binary.

Figura 4.2: esempio di codifica OPC UA Binary di una stringa [6]

5				O	P	C	U	A
05	00	00	00	4F	50	43	55	41

4.1.2 XML

Quando è preferibile trasferire informazioni in un formato facilmente interpretabile da diversi sistemi, applicazioni e anche dagli umani, OPC UA si affida alla codifica XML. La maggior parte delle applicazioni gestionali, come i sistemi MES e ERP, includono un parser XML e sono quindi in grado di decodificare nativamente informazioni XML provenienti da un dispositivo OPC UA.
Due esempi di codifica XML sono riportati nelle figure 4.3 e 4.4.

Figura 4.3: esempio di codifica XML di una primitiva [6]

```
<xs:element name="String" type="xs:string" minOccurs="0" />
```

```
<String>OPCUA</String>
```

Figura 4.4: esempio di codifica XML della struttura LocalizedText

```
<xs:complexType name="LocalizedText">
  <xs:sequence>
    <xs:element name="Locale" type="xs:string" minOccurs="0" />
    <xs:element name="Text" type="xs:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
```

```
<LocalizedText>
    <Locale>DE</Locale>
    <Text>OPCUA</Text>
</LocalizedText>
```

4.1.3 JSON

La codifica dei dati in formato JSON non era presente al rilascio di OPC UA, in quanto tale notazione era ancora poco conosciuta. Tuttavia, in seguito al successo di questo formato, il mapping su JSON è stato aggiunto alle specifiche. Questo ha permesso agli applicativi OPC UA di interoperare nativamente con i sistemi che utilizzano questa tecnologia, in particolare servizi web e gestionali che eseguono importazione ed esportazione di dati in JSON. Il funzionamento della codifica è in gran parte simile a XML.

4.2 Protocolli di sicurezza

Entrambe le soluzioni implementate da OPC UA in questo ambito sono basate sullo stabilimento di connessioni attraverso l'uso dei certificati.

4.2.1 WS-SecureConversation

WS-SecureConversation è un'estensione di WS-Security, che definisce concetti e tecnologie per lo scambio dati sicuro attraverso Web Service. Questi standard sono sviluppati dall'organizzazione no-profit OASIS.
In un'applicazione OPC UA, WS-SecureConversation si usa:

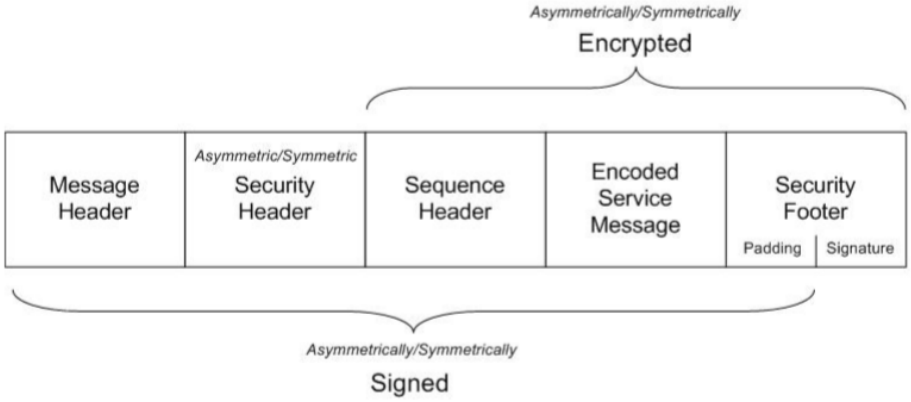
- Per negoziare parametri segreti per la creazione di Secure Channel con un altro applicativo
- Come base per le SecurityPolicy e i SecurityProfile

Questa tecnologia è ottimizzata per mettere in sicurezza dati in formato XML, il che la rende ideale per le applicazioni OPC UA che interagiscono con sistemi MES ed ERP.
Lo stabilimento della connessione richiede la presenza di un SecureChannel e di una sessione tra le due entità comunicanti. La richiesta astratta OpenSecureChannel è mappata su RequestSecurityToken e RequestSecurityTokenResponse di WS-SecureConversation. Questi messaggi permettono di condividere una chiave usata per la crittografia simmetrica di tutti i messaggi successivi. La codifica a chiave simmetrica è preferita a quella asimmetrica per motivi di prestazioni.

4.2.2 UA-SecureConversation

UA-SecureConversation non è un protocollo, ma una combinazione di tecniche e meccanismi di TLS e WS-SecureConversation.
La struttura di un messaggio generato con UA-SecureConversation è mostrata nella figura 4.5.

Figura 4.5: messaggio generato con UA-SecureConversation [6]



Il Message Header contiene informazioni per l'identificazione del tipo di messaggio (ad esempio se si tratta dell'apertura di una canale o della creazione di una sessione).
Questo header è seguito dal Security Header, che può essere di due tipi:

- Asymmetric Security Header: usato per OpenSecureChannel in quanto richiede una crittografia asimmetrica basata su chiave pubblica e privata; in questo caso l'header contiene le policy di sicurezza applicate, come l'algoritmo di codifica e il certificato del mittente
- Symmetric Security Header: utilizzato in tutti gli altri messaggi, contiene il TokenId che identifica il set di chiavi simmetriche usato per la codifica

Il Sequence Header contiene un numero di sequenza per permettere di ricomporre un messaggio dopo che è stato suddiviso in chunk.
Dopo il messaggio vero e proprio si trova il Security Footer, che contiene la Signature del messaggio, permettendo di verificare integrità e autenticità del pacchetto.

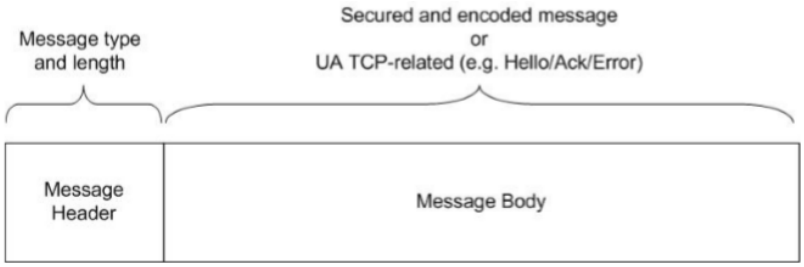
4.3 Protocolli per il trasferimento dati

I protocolli in questa sezione hanno lo scopo di stabilire una connessione di rete tra client e server, su cui costruire Secure Channel e sessioni OPC UA.

4.3.1 UA TCP

UA TCP definisce un semplice protocollo che opera in congiunzione con TCP per aggiungere alcune funzioni necessarie per OPC UA. Queste funzioni sono: stabilimento a livello applicativo delle dimensioni del buffer di comunicazione, possibilità di condividere indirizzo IP e numero di porta tra più Endpoint OPC UA che risiedono sullo stesso device e aggiunta di meccanismi di error recovery.
Nella figura 4.6 viene mostrata la struttura di un messaggio UA TCP.

Figura 4.6: struttura di un messaggio UA TCP [6]



Il Message Header contiene informazioni sul tipo e le dimensioni del messaggio. Quando si combina UA TCP con UA-SecureConversation questo campo è condiviso, in modo da ridurre l'overhead nella comunicazione.
Il Message Body può contenere un messaggio relativo a una service call oppure uno necessario all'uso del protocollo.
UA TCP definisce tre diverse tipologie di messaggi di gestione:

- Hello è inviato dal client OPC UA al server per creare un socket verso uno specifico Endpoint del server
- Acknowledge conferma la creazione del socket e l'accettazione dei parametri relativi alle dimensioni di buffer, messaggi e chunk
- Error è usato per l'invio di informazioni di diagnostica nel caso di rilevamento di errori

Il meccanismo di error recovery di UA TCP entra in funzione quando un client perde la connessione ad un socket e consiste nella creazione immediata di un nuovo socket, a cui viene assegnato il SecureChannel esistente. Non appena la comunicazione viene ristabilita, il server ri-autentica il SecureChannel e il trasferimento dati può riprendere.

4.3.2 SOAP/HTTP

SOAP over HTTP è uno schema di comunicazione molto comune in ambienti Web perché è semplice e firewall-friendly. Adottando questa tecnologia, le applicazioni OPC UA possono comunicare tra di loro tramite internet allo steso modo in cui un browser comunica con un Web server.

Il messaggio relativo ad una service call è inserito nell'elemento Body del messaggio SOAP, mentre la trasmissione viene eseguita tramite una operazioni POST HTTP.

In scenari dove non è richiesto soddisfare tutti i requisiti di OPC UA è possibile utilizzare il protocollo HTTPS per il trasferimento dei dati ed evitare l'implementazione dello stack di sicurezza di OPC UA.

5 Sicurezza

Uno dei metodi più efficaci per mantenere un sistema sicuro è l'esecuzione di valutazioni periodiche per identificare il livello di sicurezza necessario e provvedere quindi a sviluppare soluzioni adeguate a mantenerlo.
Una valutazione della sicurezza si suddivide generalmente in tre fasi:

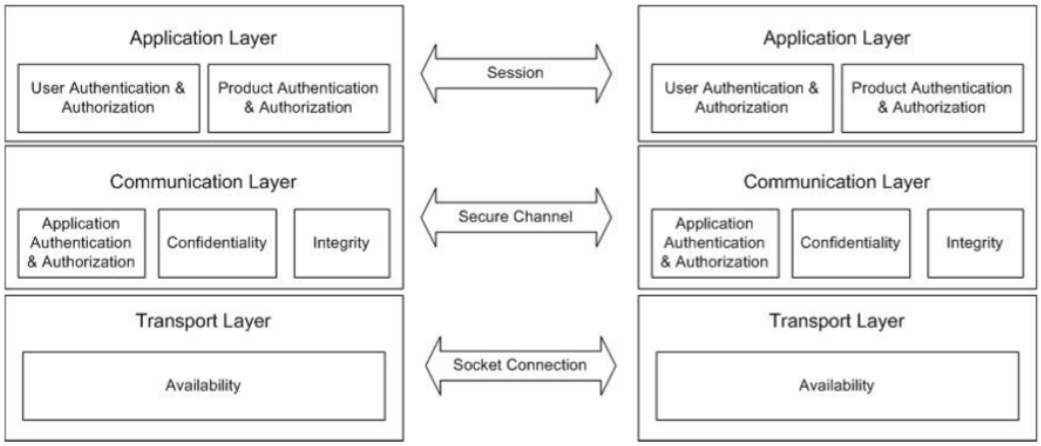
- Definizione degli obiettivi di sicurezza: consiste nello stabilire quali parti del sistema devono essere messe in sicurezza e che tipo di protezione devono avere (ad esempio confidenzialità, disponibilità, integrità, non ripudiabilità)
- Identificazione delle minacce: per ogni obiettivo definito al punto precedente vanno ricercate le minacce che potrebbero danneggiarne la sicurezza
- Determinazione delle contromisure: è la definizione di contromisure per scongiurare le minacce rilevate al punto precedente

Al termine della valutazione di sicurezza si ottiene così un insieme di contromisure da implementare, che costituiscono il livello di sicurezza appropriato per l'ambiente analizzato.
La sezione 2 delle specifiche di OPC UA [5] contiene una parte informativa in cui vengono analizzate le migliori modalità per attuare una valutazione della sicurezza su un ambiente in cui vengono eseguite applicazioni OPC UA.

5.1 Modello di sicurezza di OPC UA

Le applicazioni OPC UA sono presenti a tutti i livelli della piramide dell'automazione, dalla comunicazione tra dispositivi embedded al trasferimento dati tra sistemi ERP e MES. È quindi importante definire un modello di sicurezza flessibile che si possa adattare ad ogni ambiente di esecuzione.

Figura 5.1: architettura della sicurezza di OPC UA [6]



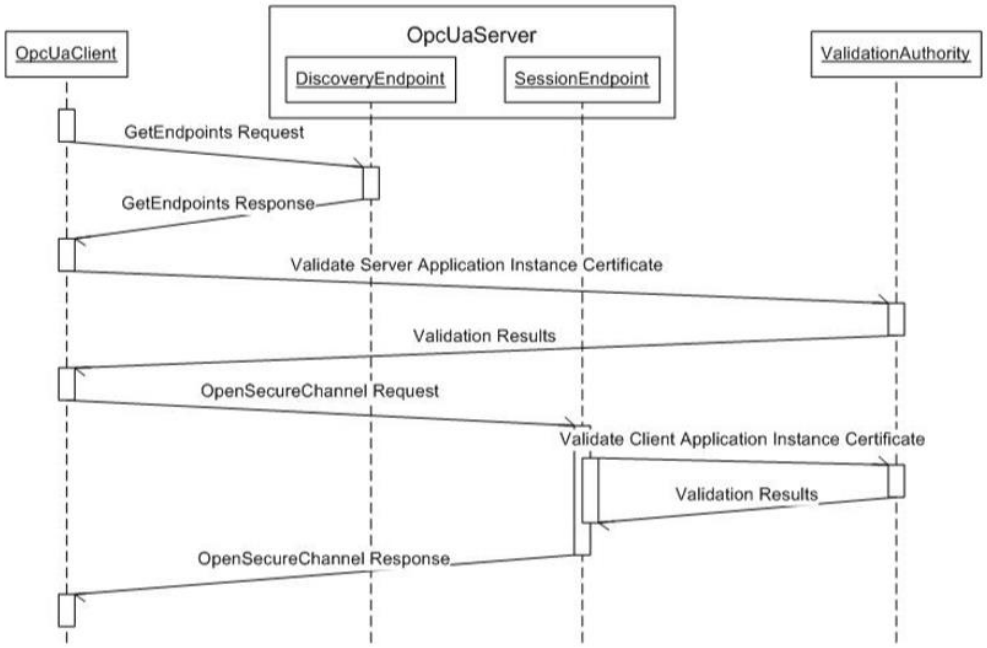
La figura 5.1 mostra come l'architettura della sicurezza di OPC UA è suddivisa in livelli. L'Application Layer è usato per la trasmissione dei dati attraverso una sessione, che si occupa di autenticare e autorizzare utenti e prodotti.
Il Communication Layer è gestito attraverso il Secure Channel, che garantisce integrità dei dati e confidenzialità dove necessario.
Al livello più basso, il Transport Layer è responsabile della trasmissione e ricezione dei dati attraverso un socket e adotta appositi meccanismi per la gestione degli errori, come quelli descritti al capitolo **4.3.1 - UA TCP**.

5.1.1 Creazione e terminazione della connessione

Lo stabilimento della connessione tra un client e un server OPC UA consiste nella creazione del Secure Channel e nello stabilimento di una Sessione al suo interno. Questo procedimento si divide in quattro step.
Nel primo passaggio il client recupera, se necessario, le informazioni relative agli Endpoint del server al quale vuole collegarsi. Sceglie poi un Endpoint con una Security Policy adeguata al contesto e ne verifica la validità tramite una Validation Authority (**5.2.2 - Public Key Infrastructure per OPC UA**). Due applicazioni OPC UA possono comunicare tra di loro soltanto se hanno almeno una Security Policy in comune.
Se il certificato dell'Endpoint è considerato affidabile, il secondo passaggio consiste nella creazione del Secure Channel. In questa fase, le parti più importanti sono la definizione della Security Mode (ossia se i dati devono essere firmati e crittografati o meno) e la

trasmissione sicura di una chiave simmetrica che sarà usata per la codifica di tutti i messaggi successivi in quanto offre prestazioni superiori alla crittografia asimmetrica. È importante notare che il Secure Channel ha una validità temporale limitata e al suo scadere va rinnovato eseguendo nuovamente il passaggio due.
I primi due step per la creazione della connessione sono illustrati in figura 5.2.

Figura 5.2: ricerca degli Endpoint e creazione del Secure Channel [6]



Il terzo passaggio consiste nella creazione sul Secure Channel stabilito, mentre il quarto e ultimo step è l'attivazione di tale sessione.
Attivare la sessione in maniera separata rispetto alla sua creazione ha vari vantaggi, tra cui:

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

La figura 5.3 illustra i passaggi necessari per creare e attivare una sessione OPC UA, mentre la figura 5.4 mostra la terminazione di una connessione OPC UA.

Figura 5.3: stabilimento di una sessione OPC UA [6]

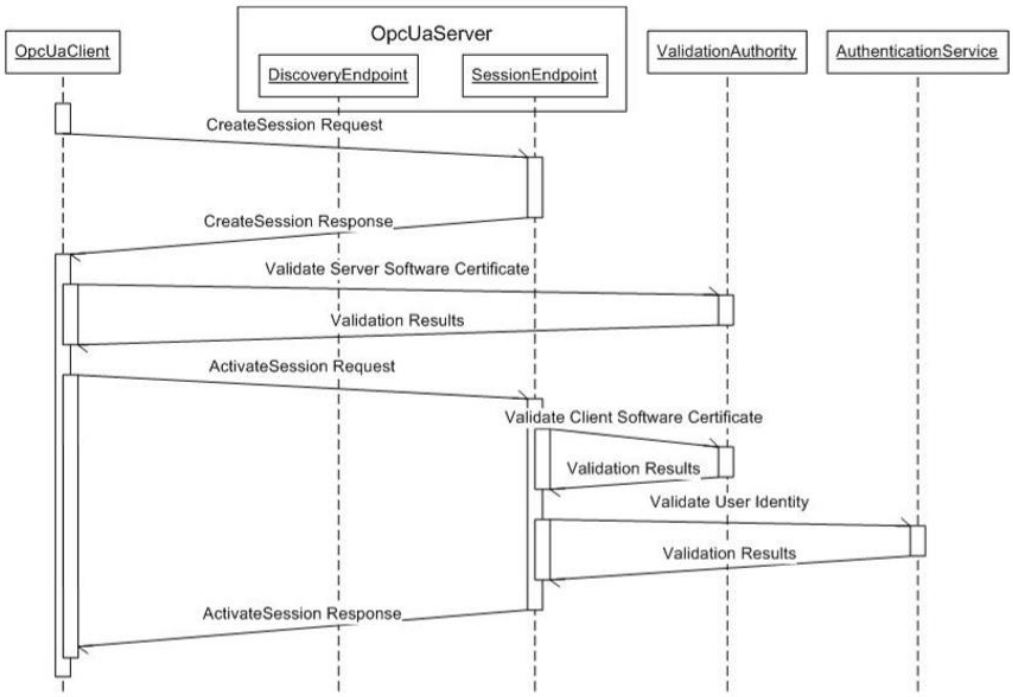
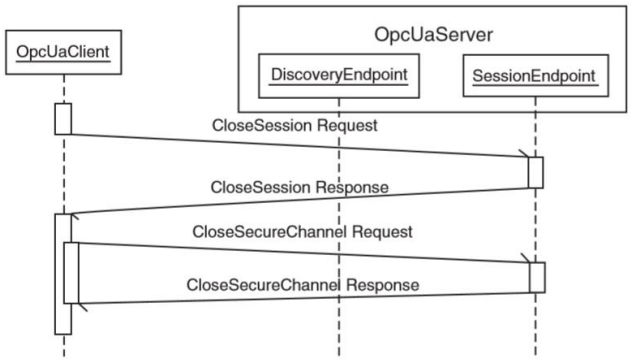


Figura 5.4: smantellamento di una connessione OPC UA [6]



Negli SDK OPC UA tutto il procedimento di creazione e smantellamento di una connessione è generalmente raggruppato in metodi *connect* e *disconnect*.

5.2 Tecnologie

5.2.1 Certificati

OPC UA adotta certificati di tipo X.509v3 per lo stabilimento delle connessioni. Il vantaggio principale di questa tecnologia è che supporta l'aggiunta di campi personalizzati ad un certificato. OPC UA distingue quindi tre tipi di certificati X.509.

Application Instance Certificate

Ogni installazione di un prodotto OPC UA richiede un certificato denominato Application Instance Certificate, che identifica l'istanza di un'applicazione OPC UA in esecuzione su un determinato host ed è ottenuto dalla Certification Authority responsabile. Tramite questo certificato è possibile eseguire autenticazione e autorizzazione di un'applicazione, permettendole di partecipare alla creazione di un Secure Channel.

Software Certificate

Un certificato software è rilasciato da un laboratorio di certificazione autorizzato OPC UA e attesta che un certo prodotto ha superato determinati test e supporta quindi un elenco di profili e servizi. Con questo certificato è possibile autenticare e autorizzare un software, verificando che supporti i servizi necessari alla creazione di una sessione.

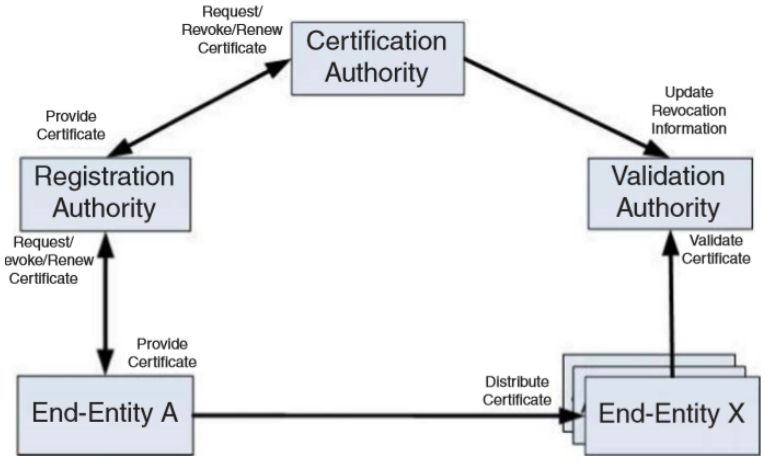
User Certificate

Gli User Certificate sono uno dei modi per eseguire autenticazione e autorizzazione di un utente nell'ambiente OPC UA, ossia per verificare che un utente sia veramente chi afferma di essere e che abbia i permessi necessari per le operazioni che intende eseguire. L'adozione degli User Certificate in un ambiente OPC UA non è obbligatoria, in quanto sono supportati altri metodi di autenticazione, come la coppia username e password o i token WS-Security (ad esempio di ticket di Kerberos).

5.2.2 Public Key Infrastructure per OPC UA

OPC UA adotta una Public Key Infrastructure (PKI) per la gestione del ciclo di vita dei certificati. Questo include richiesta, creazione, installazione, distribuzione, validazione, revoca e rinnovo di tutte le tipologie di certificato elencate nel capitolo precedente.

Figura 5.5: entità di una PKI [6]



Le entità che compongono una PKI sono mostrate nella figura 5.5.
Una End-Entity (EE) è l'utilizzatore di un certificato. Si tratta quindi di un prodotto OPC UA, di una sua istanza o di un utente.
La Registration Authority (RA) è l'entità che le EE contattano in caso di quesiti riguardanti i certificati. Funge da tramite tra le EE e la Certification Authority occupandosi di identificare il richiedente e inoltrare le informazioni alla CA perché siano processate.
La Certification Authority (CA) si occupa di rilasciare, rinnovare e revocare certificati, informando anche le altre entità quando esegue una di queste operazioni.
La Validation Authority (VA) è responsabile della validazione dei certificati forniti dalle EE.

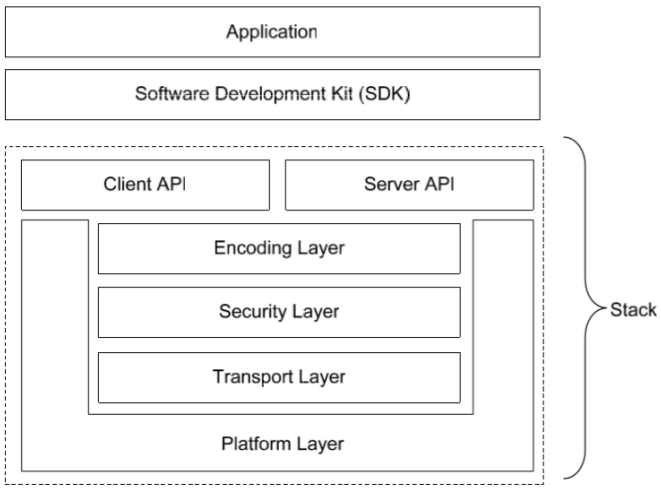
6 Architettura

6.1 Architettura applicativa

Un'applicazione OPC UA è strutturata su tre livelli: Application, SDK e Stack. Mentre il livello applicativo è fortemente dipendente dal tipo di programma (client o server) e dai casi d'uso, le parti SDK e Stack sono generalmente comuni a client e server e gestiscono rispettivamente le funzioni di alto livello, come l'elaborazione dei servizi, e quelle di basso livello, come la codifica dei messaggi.

6.1.1 Stack

Figura 6.1: livello Stack di un applicativo OPC UA [6]



Interfacce

Client e server possono usare lo stesso stack in quanto molte delle funzionalità necessarie sono in comune, come quelle per la codifica dei messaggi. Tuttavia ci sono operazioni specifiche ai client, come inviare richieste e processare risposte, e ai server, come l'esecuzione delle richieste e l'invio delle risposte. Queste funzioni sono raggruppate al livello delle interfacce in API per i client e API per i server.

Encoding Layer

Le strutture che rappresentano i messaggi dei servizi vengono passate dal livello delle API a quello della codifica, che si occupa della serializzazione dei dati e dell'inoltro al livello sottostante.

Security Layer

Il livello della sicurezza riceve messaggi codificati dall'Encoding Layer e li mette in sicurezza secondo la policy in uso, che può prevedere firma dei dati, firma e crittografia o nessuna procedura di sicurezza. In base alle operazioni eseguite, vengono aggiunti al messaggio opportuni header e footer con informazioni su come verificare la firma e decrittare il messaggio.
La struttura così ottenuta è poi inviata al Transport Layer.

Transport Layer

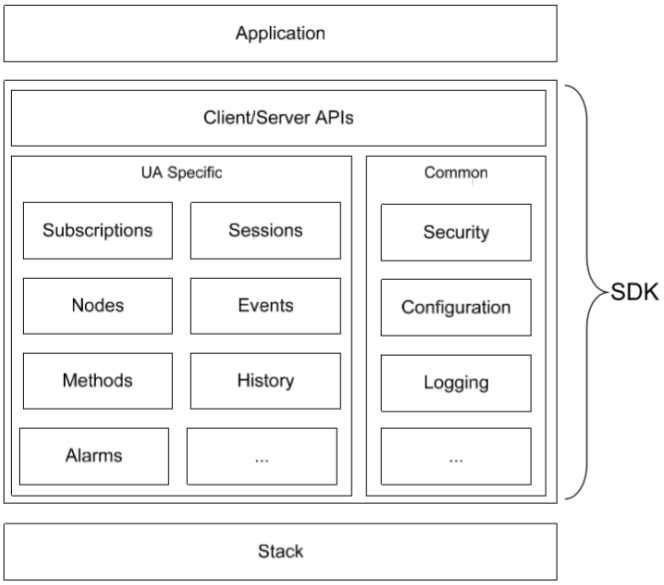
Il Transport Layer si occupa di inviare e ricevere i messaggi, oltre a gestire errori di rete. In fase di trasmissione, aggiunge al messaggio degli header con informazioni come il tipo e la dimensione del messaggio, mentre in ricezione verifica tali dati e inoltra il messaggio al livello soprastante.

Platform Layer

Il Platform Layer è introdotto per rendere tutti i livelli soprastanti platform-independent, in quanto contiene tutto il codice specifico alla piattaforma in uso. Ciò significa che per effettuare il porting dello stack su un'altra piattaforma è necessario sostituire solo questo livello.

6.1.2 SDK

Figura 6.2: livello SDK di un applicativo OPC UA [6]



Funzionalità specifiche di OPC UA

Questa parte dell'SDK comprende l'implementazione di concetti e servizi introdotti nelle specifiche OPC UA.

Funzionalità comuni

Questo livello contiene l'implementazione tutte le funzionalità comuni ad ogni applicativo, ma che non sono strettamente legate a OPC UA.

In base all'ambiente OPC UA su cui l'applicativo dovrà andare in esecuzione, è possibile decidere di inserire alcune funzioni a livello Stack, SDK o Application. Ad esempio, se non tutti i programmi utilizzano lo stesso metodo di validazione dei certificati, è necessario che l'implementazione sia effettuata a livello Application, mentre in caso contrario queste funzioni possono essere inglobate a livello SDK.

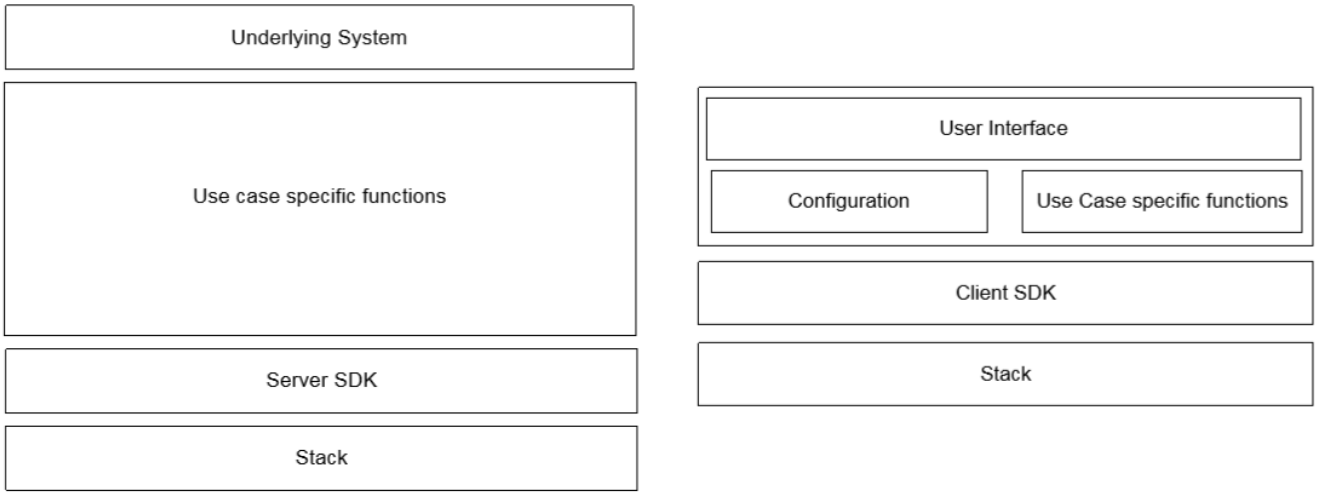
Interfacce

Contiene i metodi per l'interfacciamento con il livello Application. Le interfacce client servono a inviare richieste al server e a riceverne le risposte, mentre quelle server devono

fornire metodi per eseguire operazioni con i data provider sottostanti e per inoltrarne i risultati al client.

6.1.3 Application

Figura 6.3: architettura di un server e di un client OPC UA, con dettaglio sul livello Application [6]



Il livello Application è distinto per client e server e dipende fortemente dal tipo di applicativo che si sta realizzando. In generale, il programma client è dotato di un'interfaccia utente e di una parte di configurazione, mentre quello server dispone di funzioni per la gestione dell'Address Space, che può essere effettuata in maniera diretta o tramite interfacciamento con sistemi sottostanti.

6.2 Architettura di sistema

6.2.1 Pattern architetturali

OPC UA adotta cinque pattern architetturali principali, illustrati di seguito.

Client-server

Questo è il pattern architetturale di base, in cui un client OPC UA invia una richiesta con un formato ben definito ad un server OPC UA per l'esecuzione di determinati task. Il server processa la richiesta ed invia una risposta appropriata.

Chained server

Il pattern chained server coinvolge un client A, un server B con client integrato e un normale server C. In questo caso il server B funge da intermediario tra A e C. Uno scenario di esempio è la comunicazione del client A, che supporta solo HTTP, col server C che supporta invece TCP. B, supportando entrambi i protocolli, può fare da gateway per la comunicazione.

Server-to-server

La differenza principale tra chained server e server-to-server è che nel secondo caso entrambi i server possono avviare la comunicazione, in quanto sono dotati di client OPC UA incorporati. Uno scenario comune per l'utilizzo di questo pattern è la gestione delle repliche tra server ridondanti.

Aggregating server

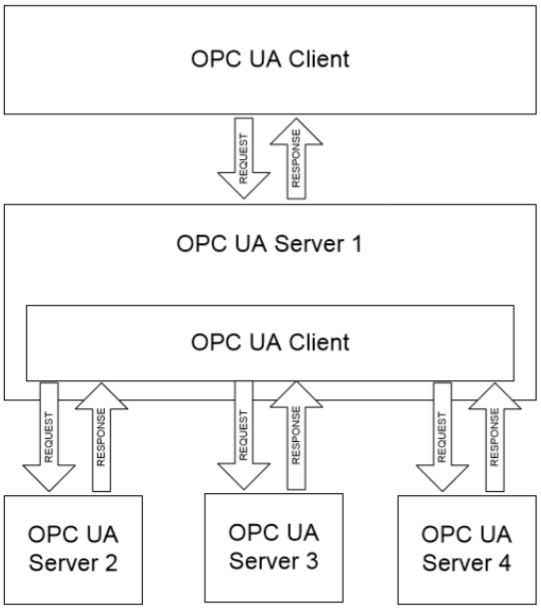
Il pattern aggregating server prevede l'invio di una richiesta da parte di un client OPC UA a un server con client integrato. Questo server è in contatto con più server OPC UA sottostanti, a cui inoltra le informazioni necessarie, per poi processare i dati ricevuti in risposta e mandarli al client.

La differenza principale tra chained server e aggregating server è che nel secondo pattern il server intermedio non funge solo da gateway, ma processa le informazioni per presentarle al client nella maniera più appropriata.

Un esempio di applicazione di aggregating server è la gestione degli ordini da parte di un sistema MES. Un client OPC UA crea e supervisiona gli ordini di produzione comunicando con un unico server. Questo server suddivide le richieste del client in subtask e li fa eseguire ai server sottostanti, ciascuno responsabile di una certa fase del processo di produzione. Resta poi in attesa dei risultati di tali task e quando li riceve, elabora le informazioni e compone una risposta per il client.

Nella figura 6.4 è illustrato il pattern aggregating server

Figura 6.4: pattern aggregating server [6]



Publish-subscribe

Questo pattern coinvolge tre tipi di entità: publisher, middleware e subscriber. Uno o più publisher inviano dati al middleware, senza sapere se ci sono dei subscriber che li ricevono. Questi ultimi si registrano presso il middleware specificando opportuni filtri per gli argomenti di interesse, e il middleware si occupa dell'invio dei messaggi.
Publish-subscribe è stato introdotto nella versione 1.04 di OPC UA. Essendo un pattern altamente scalabile e adatto a innumerevoli casi d'uso, un intero capitolo delle specifiche è stato aggiunto per un adeguato approfondimento (14 - PubSub [5]).

6.2.2 Ridondanza

Client

La ridondanza dei client è supportata da OPC UA con il servizio TransferSubscriptions, usato in combinazione al monitoraggio delle informazioni del client sull'Address Space del server.
Un client che esegue operazioni critiche deve essere ridondato da un client di backup, che monitora lo stato della sessione nell'Address Space del server. Non appena lo stato cambia, indicando una disconnessione, il client di backup usa il servizio

TransferSubscriptions per subentrare nella gestione. Il ciclo di vita delle Subscription è indipendente da quello delle sessioni (**3.1.4 - Contesto per la comunicazione**), per cui il trasferimento su una nuova sessione non crea alcun tipo di problema.

Server

La ridondanza del server può essere gestita in maniera trasparente o non trasparente al client.

Per il primo approccio i server ridondati devono essere in mirroring, ossia avere esattamente gli stessi dati e le stesse informazioni di sessione. Non appena il server attivo smette di essere disponibile, le richieste vengono inoltrate al suo clone senza che il client si accorga di nulla.

La ridondanza non trasparente può essere gestita in tre modalità:

- Cold: il server di backup è in esecuzione, ma viene attivato e contattato dai client solo in caso di malfunzionamento di quello principale
- Warm: il server di backup è attivo, ma le sue subscription vengono attivate solo in caso di malfunzionamento del server principale
- Hot: il server di backup è attivo e può essere utilizzato per accedere ai dati; in caso di malfunzionamento di uno dei due server, il rimanente continuerà a funzionare, prendendo in carico anche le richieste dell'altro

6.2.3 Discovery

I servizi per il processo di discovery sono definiti nel capitolo 4 delle specifiche di OPC UA [5].

Le entità e i processi di discovery sono definiti nel capitolo 12 delle specifiche OPC UA [5] e riportati di seguito.

Entità

Session Endpoint è l'entità di un server OPC UA usata per la creazione di Secure Channel e sessioni e per l'accesso ai dati forniti dal server.

Discovery Endpoint fornisce informazioni riguardo ad altri Endpoint. Può essere utilizzato da un normale server per inviare ai client l'elenco dei suoi Endpoint o da un Local o Global Discovery server per condividere gli Endpoint dei server registrati presso di esso.
Se un Discovery server risiede sulla stessa macchina dei server di cui fornisce informazioni, allora è chiamato Local Discovery Server, mentre in caso contrario si parla di Global Discovery server.

Processi

OPC UA definisce tre diversi procedimenti di Discovery.
Simple Discovery si esegue quando un client ha già l'indirizzo del server OPC UA a cui vuole collegarsi e consiste nell'invio di una richiesta GetEndpoints, nella selezione dell'Endpoint desiderato e nello stabilimento della connessione.
Nel procedimento Normal Discovery, il client usa una richiesta FindServers per ottenere, da un Local Discovery Server, l'elenco dei server disponibili. Se il server di interesse è nella lista si continua come nel procedimento Simple Discovery.
Hierarchical Discovery prevede una richiesta a un Global Discovery Server. Se nell'elenco restituito è presente il server a cui il client vuole collegarsi, allora si procede come in Simple Discovery, mentre in caso contrario viene selezionato un Local Discovery Server appropriato e si continua come in Normal Discovery.

6.2.4 Auditing

In ambito OPC UA, per auditing si intende il tenere traccia delle attività delle applicazioni OPC UA, garantendo la tracciabilità del sistema. Questo può tornare utile in varie situazioni, come per eseguire debugging sugli applicativi in caso di errori o per supportare le attività forensi dopo un incidente di sicurezza.
L'auditing può essere eseguito direttamente dagli applicativi, che generano eventi e memorizzano le relative informazioni in file o database di log. In alternativa, gli applicativi OPC UA possono pubblicare gli eventi di auditing per l'uso da parte di appositi client che si occupano di memorizzarli.

7 Profili

OPC UA offre un range molto ampio di funzionalità ed è normale che non tutti gli applicativi le supportino nella loro interezza. Per classificare applicazioni con feature differenti, la sezione 7 delle specifiche di OPC UA [5] introduce il concetto di profili.
I profili definiscono le funzionalità di un'applicazione OPC UA. Possono essere usati dal venditore per pubblicizzare le feature di un software, dai clienti per richiedere un determinato set di funzioni o dagli applicativi stessi per rifiutare le connessioni da parte di entità OPC che non supportano i profili necessari.
Un'applicazione OPC UA può implementare più profili ed ogni profilo può contenerne altri. Un profilo è composto da varie Conformance Unit (un esempio è la chiamata ad un servizio) che sono dotate di Test Case predefiniti per verificarne il corretto funzionamento.
Vi sono quattro categorie di profili: server-related, client-related, transport-related e security-related. I profili server-related sono utili per permettere ad un server di presentare l'elenco delle feature che supporta. Quelli client-related sono un buon metodo per stabilire quali operazioni può eseguire un applicativo client. Infine, non è possibile stabilire una connessione sicura tra due applicazioni OPC UA se non hanno un profilo transport-related ed uno security-related in comune.

8 API

Al momento del rilascio di OPC UA, nel 2006, la OPC Foundation mise a disposizione un SDK contenente due implementazioni dello stack OPC UA, una in ANSI C e una in C#.

Oggi, SDK commerciali sono disponibili nel linguaggi C, C++, C# e Java, mentre esistono soluzioni open source scritte in C, C++, C#, Java, JavaScript (Node.js), Python e Rust [7].

Di seguito sono analizzate alcune delle soluzioni di rilievo.

C++

Lo stack open62541 [8] supporta Windows, Linux, Android e vari sistemi embedded, è ottimizzato attraverso l'uso di plugin specifici in base all'architettura (come POSIX su Linux) e certificato Self-Tested. Al momento della stesura di questo trattato, open62541 è una delle soluzioni open source che con il maggior numero di feature e più attivamente supportate, in quanto l'unica (salvo gli stack della OPC Foundation) a mettere già a disposizione il pattern architetturale publish-subscribe introdotto con la versione 1.04 di OPC UA. È concesso in licenza con Mozilla Public Licence 2.0.

Altre soluzioni sono FreeOpcUa [9], ASNeG [10] e UAF [11] (C++/Python).

C#

Lo stack OPC UA C# più popolare è quello offerto dalla OPC Foundation [12]. Questo SDK rispetta lo Standard .NET [13], pertanto supporta nativamente le piattaforme Windows, Linux, iOS e Android (tramite Xamarin). Inoltre, essendo sviluppato dall'organizzazione fondatrice di OPC UA, è stato testato e convalidato OPC Certified.

Una caratteristica particolare di questa implementazione è che il Transport Layer, che si occupa della trasmissione dei dati, è implementato in ANSI C, mentre i livelli soprastanti usano C#. Questo permette di mantenere l'efficienza di ANSI C per la parte di rete, ma di eseguire le operazioni di de-serializzazione in C#, ottenendo direttamente oggetti utilizzabili all'interno dell'applicativo.

Java

Esistono varie implementazioni dello stack OPC UA in linguaggio Java, ma possono essere classificate in tre categorie.

- SDK che incapsulano lo stack ANSI C nella sua completezza usando JNI. Questo consente di usufruire delle buone prestazioni dell'ANSI C, ma rende la portabilità complicata in quanto il codice C non è platform independent
- Stack che codifica in Java solo i livelli sopra al Transport Layer (come nell'implementazione C#)
- Stack scritti interamente in Java, come l'SDK open source Eclipse Milo [14]

JavaScript

L'implementazione node-opcua [15], realizzata in JavaScript per Node.js, mette a disposizione uno stack completo con una particolare attenzione alla programmazione asincrona.

Python

Il progetto FreeOpcUa [16] fornisce l'implementazione di un discreto set di funzionalità dello stack OPC UA in linguaggio Python. In particolare, esiste un fork di questo progetto, denominato opcua-asyncio [17], che è basato sulla libreria asyncio [18] ed è orientato alla programmazione asincrona e concorrente.

Tra le feature lato client supportate da opcua-asyncio sono presenti: connessione ai server OPC UA con creazione di Secure Channel e sessione, browsing dell'Address Space, sottoscrizione a cambi di dato ed eventi, chiamata di metodi, crittografia, aggiunta e rimozione di nodi. Feature lato server di rilievo sono invece: supporto alla connessione di client con Secure Channel e sessione, crittografia, gestione dei certificati.

Alcune importanti funzionalità non al momento presenti in questo stack sono: supporto al formato XML, supporto a LocalizedText, viste, allarmi.

Il **capitolo 9** illustra una semplice demo di comunicazione client-server sfruttando opcua-asyncio.

Rust

Esiste un'implementazione open source sotto licenza MPL-2.0, che include i profili embedded, micro e nano e supporta la codifica binaria su UA TCP [19].

9 Demo

Per mostrare quanto sia semplice realizzare un applicativo utilizzando una delle implementazioni open source di OPC UA, è stata realizzata una breve demo Python con lo stack opcua-asyncio [17].

La demo è composta da un server e un client OPC UA, che sono indipendenti tra di loro e vengono eseguiti sulla stessa macchina. Se risiedessero su due dispositivi diversi connessi alla stessa rete, basterebbe cambiare l'Endpoint che il client utilizza sostituendo *localhost* con l'IP del server. Per la comunicazione è utilizzato UA TCP, mentre la sicurezza è disabilitata. Il traffico dati generato è stato catturato tramite WireShark.

All'avvio, il server popola il suo Address Space creando un oggetto e inserendovi una variabile e un metodo. Successivamente entra in un ciclo infinito in cui, a intervalli di un secondo, incrementa il valore della variabile creata.

Il client esegue le seguenti operazioni:

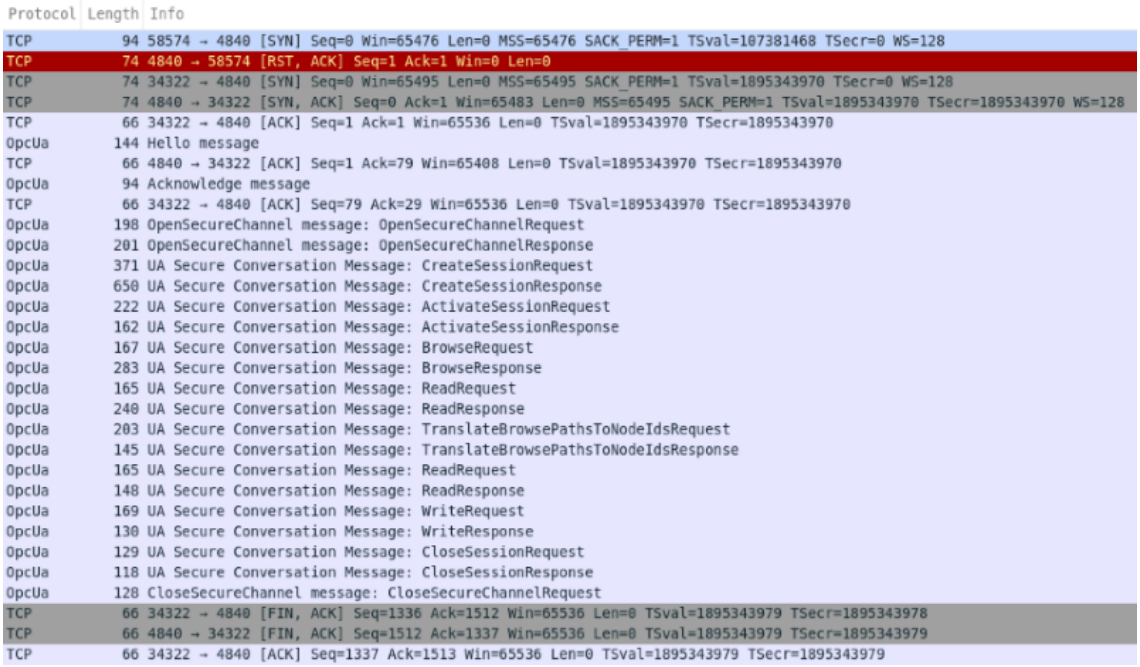
- Si collega al server
- Richiede il nodo radice della gerarchia di oggetti del server
- Richiede il NamespaceIndex relativo all'Endpoint che ha utilizzato per il collegamento
- Usa il NamespaceIndex per leggere il valore della variabile
- Sovrascrive il valore della variabile

Nella figura 9.1 si può vedere chiaramente la struttura della comunicazione, con l'apertura del SecureChannel, la creazione e l'attivazione della sessione, l'esecuzione dei servizi e la terminazione. Le figure 9.2 mostra la struttura del messaggio OPC di risposta a una richiesta di Read, mentre nella figura 9.3 è illustrata la struttura della richiesta di Write della variabile.

Le figure 9.4 e 9.5 mostrano l'esecuzione di server e client da riga di comando. Da notare il cambio di valore della variabile da 0.6 a 1000.1, accaduto quando la richiesta di Write del client è stata eseguita dal server.

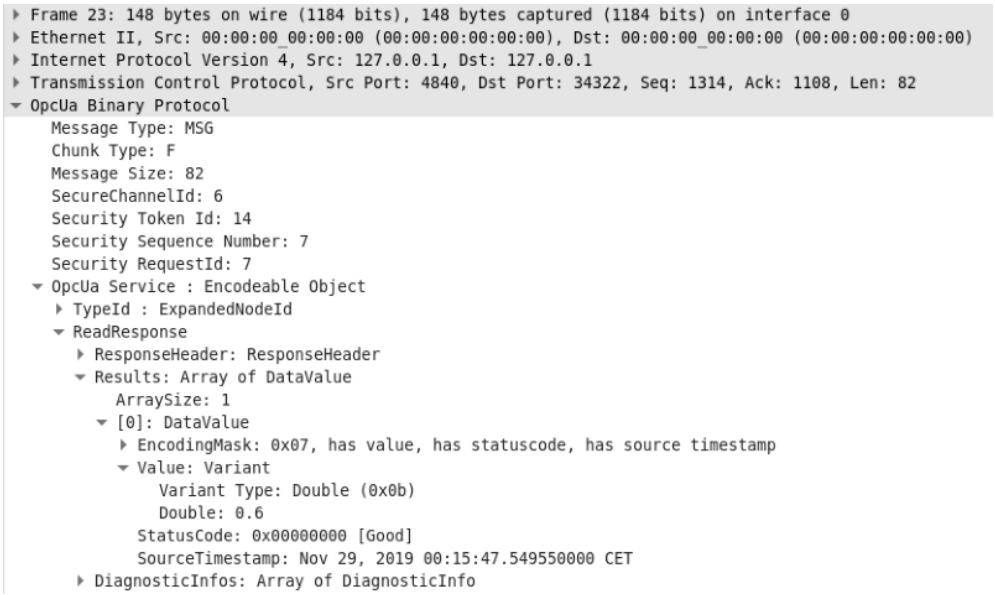
Il codice relativo alla demo è riportato in appendice a questo trattato (**12 - Appendice**).

Figura 9.1: pacchetti rilevati dalla cattura WireShark



Protocol	Length	Info
TCP	94	58574 → 4840 [SYN] Seq=0 Win=65476 Len=0 MSS=65476 SACK_PERM=1 TSval=107381468 TSecr=0 WS=128
TCP	74	4840 → 58574 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
TCP	74	34322 → 4840 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1895343970 TSecr=0 WS=128
TCP	74	4840 → 34322 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1895343970 TSecr=1895343970 WS=128
TCP	66	34322 → 4840 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1895343970 TSecr=1895343970
OpcUa	144	Hello message
TCP	66	4840 → 34322 [ACK] Seq=1 Ack=79 Win=65408 Len=0 TSval=1895343970 TSecr=1895343970
OpcUa	94	Acknowledge message
TCP	66	34322 → 4840 [ACK] Seq=79 Ack=29 Win=65536 Len=0 TSval=1895343970 TSecr=1895343970
OpcUa	198	OpenSecureChannel message: OpenSecureChannelRequest
OpcUa	201	OpenSecureChannel message: OpenSecureChannelResponse
OpcUa	371	UA Secure Conversation Message: CreateSessionRequest
OpcUa	650	UA Secure Conversation Message: CreateSessionResponse
OpcUa	222	UA Secure Conversation Message: ActivateSessionRequest
OpcUa	162	UA Secure Conversation Message: ActivateSessionResponse
OpcUa	167	UA Secure Conversation Message: BrowseRequest
OpcUa	283	UA Secure Conversation Message: BrowseResponse
OpcUa	165	UA Secure Conversation Message: ReadRequest
OpcUa	240	UA Secure Conversation Message: ReadResponse
OpcUa	203	UA Secure Conversation Message: TranslateBrowsePathsToNodeIdsRequest
OpcUa	145	UA Secure Conversation Message: TranslateBrowsePathsToNodeIdsResponse
OpcUa	165	UA Secure Conversation Message: ReadRequest
OpcUa	148	UA Secure Conversation Message: ReadResponse
OpcUa	169	UA Secure Conversation Message: WriteRequest
OpcUa	130	UA Secure Conversation Message: WriteResponse
OpcUa	129	UA Secure Conversation Message: CloseSessionRequest
OpcUa	118	UA Secure Conversation Message: CloseSessionResponse
OpcUa	128	CloseSecureChannel message: CloseSecureChannelRequest
TCP	66	34322 → 4840 [FIN, ACK] Seq=1336 Ack=1512 Win=65536 Len=0 TSval=1895343979 TSecr=1895343978
TCP	66	4840 → 34322 [FIN, ACK] Seq=1512 Ack=1337 Win=65536 Len=0 TSval=1895343979 TSecr=1895343979
TCP	66	34322 → 4840 [ACK] Seq=1337 Ack=1513 Win=65536 Len=0 TSval=1895343979 TSecr=1895343979

Figura 9.2: risposta ad una seconda richiesta di Read



```
▸ Frame 23: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 4840, Dst Port: 34322, Seq: 1314, Ack: 1108, Len: 82
▾ OpcUa Binary Protocol
    Message Type: MSG
    Chunk Type: F
    Message Size: 82
    SecureChannelId: 6
    Security Token Id: 14
    Security Sequence Number: 7
    Security RequestId: 7
  ▾ OpcUa Service : Encodeable Object
    ▸ TypeId : ExpandedNodeId
    ▾ ReadResponse
      ▸ ResponseHeader: ResponseHeader
      ▾ Results: Array of DataValue
          ArraySize: 1
        ▾ [0]: DataValue
          ▸ EncodingMask: 0x07, has value, has statuscode, has source timestamp
          ▾ Value: Variant
              Variant Type: Double (0x0b)
              Double: 0.6
            StatusCode: 0x00000000 [Good]
            SourceTimestamp: Nov 29, 2019 00:15:47.549550000 CET
      ▸ DiagnosticInfos: Array of DiagnosticInfo
```

Figura 9.3: richiesta di Write

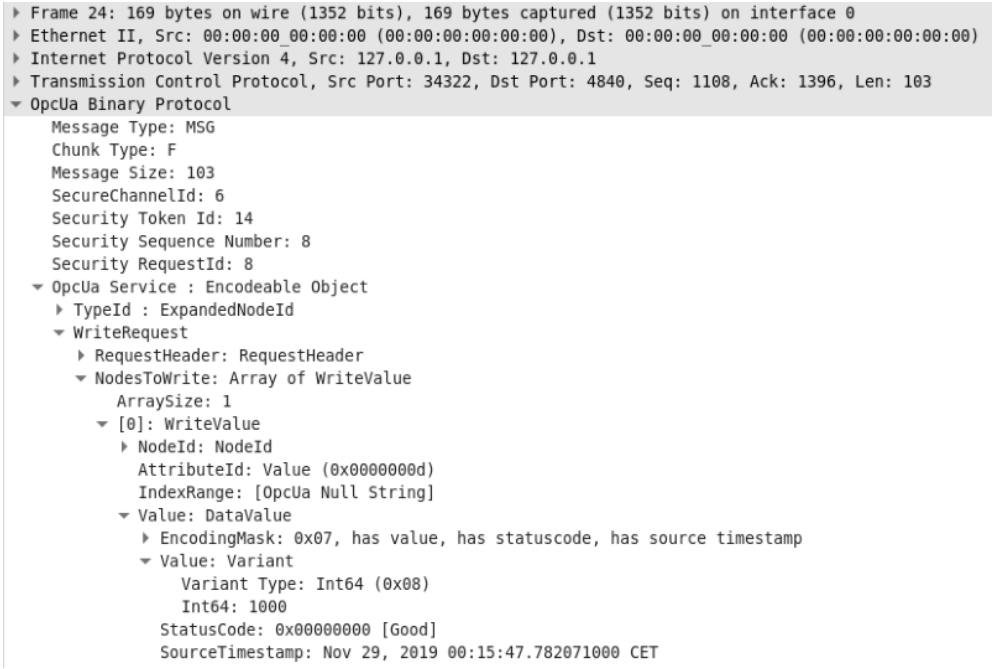


Figura 9.4: server

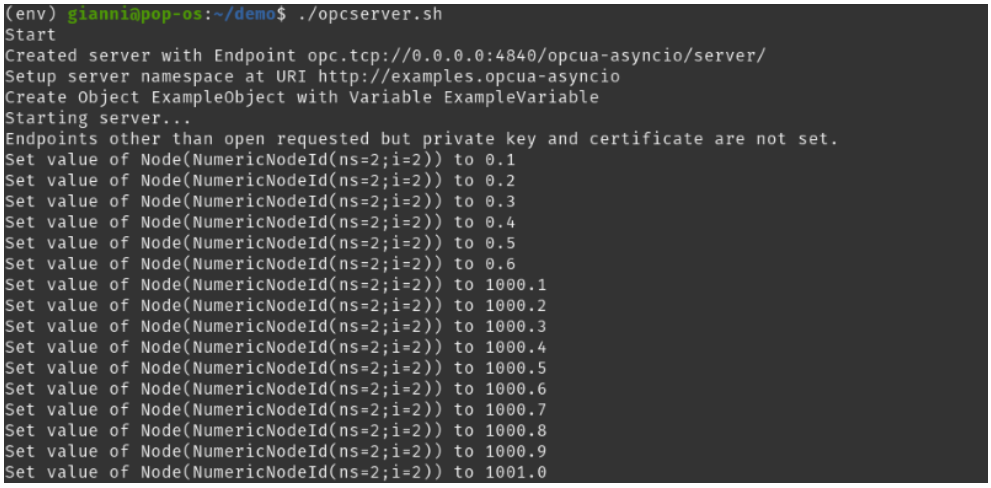
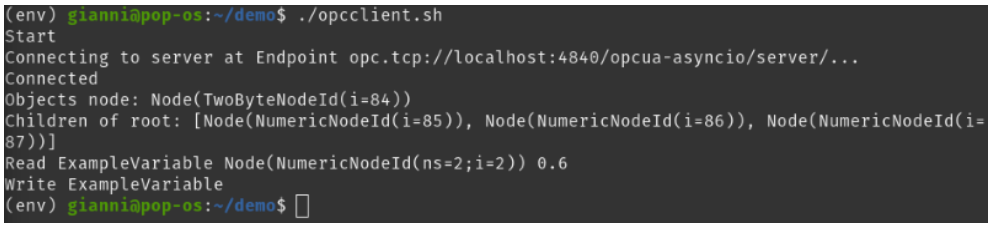


Figura 9.5: client



10 Conclusioni

Perché un protocollo di scambio dati sia rilevante nel mondo delle industrie, deve rispondere a non poche necessità. OPC UA è indipendente dalla piattaforma e dal mezzo di comunicazione, in quanto le sue specifiche sono fornite in maniera astratta ed è possibile aggiungere mapping su nuove tecnologie qualora si rivelasse necessario (ad esempio **4.1.3 - JSON**).

Grazie a un dettagliato standard di Information Modelling, OPC UA permette di gestire virtualmente ogni genere di informazione e dà la possibilità ai venditori di modellare tipi di dato complessi specifici alle loro esigenze.

Con un'architettura di sicurezza incentrata su certificati e PKI, OPC UA è in grado di garantire confidenzialità e non ripudiabilità dei dati. Per una comunicazione orientata esclusivamente alle performance è anche possibile disabilitare la sicurezza.

Infine, grazie al concetto dei profili, OPC UA si presenta come una valida soluzione per una gamma molto ampia di prodotti, da sistemi embedded per il rilevamento dati a software MES ed ERP.

Non a caso, dal suo rilascio nel 2006 OPC UA è stato adottato da un oltre seicento venditori nel campo dell'automazione industriale in tutto il mondo. Grazie alla sua versatilità, si è espanso oltre a quelli che erano i suoi obiettivi iniziali e oggi viene utilizzato in aree come sicurezza, smart home, packaging e petrolchimica. Le svariate soluzioni open source disponibili offrono inoltre un semplice punto di partenza per chiunque voglia utilizzare questa tecnologia per la prima volta.

10.1 Note dell'autore

A causa della carenza di pubblicazioni che offrono dettagli tecnici sugli aspetti di OPC UA, questo trattato fa riferimento soprattutto al libro OPC Unified Architecture, pubblicato nel 2009. Essendo le informazioni riportate in tale documento relative alle prime versioni del protocollo, è stato fatto un controllo incrociato con le specifiche della versione più recente (1.04) per verificare l'attuale validità dei concetti forniti.

Per quanto riguarda le funzionalità introdotte nelle ultime versioni di OPC UA, non è stato possibile reperire articoli o scritti di alcun genere a riguardo, salvo quanto riportato nella documentazione ufficiale. Pertanto, solo JSON (**4.1.3**) e PubSub (**6.2.1**), che sono considerate le aggiunte più rilevanti, sono presenti in questo trattato.

11 Bibliografia

[1] *Wikipedia - Industria 4.0*, [Online] https://it.wikipedia.org/wiki/Industria_4.0, [Consultato: 11 2019]

[2] *OPC Foundation*, [Online] https://opcfoundation.org/, [Consultato: 11 2019]

[3] *Wikipedia - OPC Foundation*, [Online] https://en.wikipedia.org/wiki/OPC_Foundation, [Consultato: 11 2019]

[4] *Wikipedia - OPC Data Access*, [Online] https://en.wikipedia.org/wiki/OPC_Data_Access, [Consultato: 11 2019]

[5] *OPCFoundation - OPC UA Online Reference*, [Online] https://reference.opcfoundation.org/v104/, [Consultato: 11 2019]

[6] Wolfgang Mahnke, Stefan-Helmut Leitner, Matthias Damm, *OPC Unified Architecture*, Springer, 2009

[7] *Wikipedia - OPC Unified Architecture*, [Online] https://en.wikipedia.org/wiki/OPC_Unified_Architecture, [Consultato: 11 2019]

[8] *open62541*, [Online] https://open62541.org/, [Consultato: 11 2019]

[9] *FreeOpcUa*, [Online] http://freeopcua.github.io/, [Consultato: 11 2019]

[10] *ASNeG*, [Online] https://asneg.github.io/, [Consultato: 11 2019]

[11] *GitHub - UAF*, [Online] https://github.com/uaf/uaf, [Consultato: 11 2019]

[12] *OPC UA .NET*, [Online] http://opcfoundation.github.io/UA-.NETStandard/, [Consultato: 11 2019]

[13] *Microsoft Docs - .NET Standard*, [Online] https://docs.microsoft.com/en-us/dotnet/standard/net-standard, [Consultato: 11 2019]

[14] *GitHub - Eclipse Milo*, [Online] https://github.com/eclipse/milo, [Consultato: 11 2019]

[15] *Node-OPCUA*, [Online] http://node-opcua.github.io/, [Consultato: 11 2019]

[16] *GitHub - Python OPC UA*, [Online] https://github.com/FreeOpcUa/python-opcua, [Consultato: 11 2019]

[17] *GitHub - opcua-asyncio*, [Online] https://github.com/FreeOpcUa/opcua-asyncio, [Consultato: 11 2019]

[18] *Python Documentation - asyncio - Asynchronous I/O*, [Online] https://docs.python.org/3/library/asyncio.html, [Consultato: 11 2019]

[19] *GitHub - Rust OPC UA*, [Online] https://github.com/locka99/opcua, [Consultato: 11 2019]

12 Appendice

Questa sezione contiene il codice Python utilizzato per la demo al **capitolo 9**.

opcserver.py

```
import asyncio
import logging
from asyncua import ua, Server
from asyncua.common.methods import uamethod

@uamethod
def func(parent, value):
    return value * 2

async def main():
    print('Start')

    # Setup server
    server = Server()
    await server.init()
    endpointName = 'opc.tcp://0.0.0.0:4840/opcua-asyncio/server/'
    server.set_endpoint(endpointName)
    print(f'Created server with Endpoint {endpointName}');
    # Setup server namespace
    uri = 'http://examples.opcua-asyncio'
    idx = await server.register_namespace(uri)
    print(f'Setup server namespace at URI {uri}');

    # Get Objects node from server
    objects = server.get_objects_node()
    # Populate address space with a new node
    objectName = 'ExampleObject'
    variableName = 'ExampleVariable'
    objectExample = await objects.add_object(idx, objectName)
    variableExample = await objectExample.add_variable(idx, variableName,
0)
    print(f'Create Object {objectName} with Variable {variableName}');
    # Set ExampleVariable to be writable by clients
    await variableExample.set_writable()
    # Add a method to the object
    await objects.add_method(
        ua.NodeId('ServerMethod', 2), ua.QualifiedName('ServerMethod', 2),
        func, [ua.VariantType.Int64], [ua.VariantType.Int64]
    )

    print('Starting server...')
    async with server:
        while True:
            await asyncio.sleep(1)
            value = await variableExample.get_value() + 0.1
            await variableExample.set_value(value)
            print('Set value of %s to %.1f' % (variableExample,
round(value, 1)))
```

```
if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
    loop.close()
```

opcclient.py

```
import asyncio
import sys
import logging
from asyncua import ua, Client, Node

async def main():
    print('Start')

    # Set target OPC UA server URL
    url = 'opc.tcp://localhost:4840/opcua-asyncio/server/'
    print(f'Connecting to server at Endpoint {url}...')

    async with Client(url=url) as client:
        print('Connected')

        # Get Address Space Root node
        root = client.get_root_node()
        print(f'Objects node: {root}')
        # Get children nodes
        print('Children of root:', await root.get_children())
        #Get Namespace Index
        uri = 'http://examples.opcua-asyncio'
        idx = await client.get_namespace_index(uri)

        # Get server Variable by name
        objectName = 'ExampleObject'
        variableName = 'ExampleVariable'
        var = await root.get_child(["0:Objects", f"{idx}:{objectName}",
f"{idx}:{variableName}"])
        print("Read", variableName, var, round(await var.get_value(), 1))
        # Set server Variable
        await var.set_value(1000)
        print("Write", variableName);

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
    loop.close()
```