UNIVERSITÀ DEGLI STUDI DI CATANIA

OPC Unified Architecture (OPC UA)
Concetti e documentazione dei servizi

Realizzazione a cura di:

- Docente del Corso: Prof. Ing. Salvatore Cavalieri
- Studenti:
  - Alberto Bruccini, A63/000273
  - Giovanni Torrisi, A63/000229

-

Indice

1 Introduzione (Richiesto per l'Esame)

Il primo standard OPC di grande successo – OPC Data Access COM/DCOM – fu concepito per definire interfacce client/server per l'accesso in lettura e scrittura ai dati di processo, permettendo una semplice e completa integrazione di tutti i dispositivi hardware/software per l'automazione, di differenti produttori.

In ambito industriale, però, non sempre si è riusciti ad usufruire dei vantaggi offerti dalla piattaforma OPC-DA a causa della dipendenza da COM e delle limitazioni nei meccanismi per l'accesso remoto forniti da DCOM.

La prima risposta di OPC Foundation al problema è stata OPC XML-DA, che manteneva le caratteristiche fondamentali di OPC utilizzando però un'infrastruttura di comunicazione non legata né a un particolare produttore né a una particolare piattaforma software. Tuttavia, la sola conversione delle specifiche OPC-DA in versioni basate sui Web Service non è stata sufficiente a soddisfare la necessità di uno standard di nuova generazione, specialmente a causa delle limitate performance di XML.

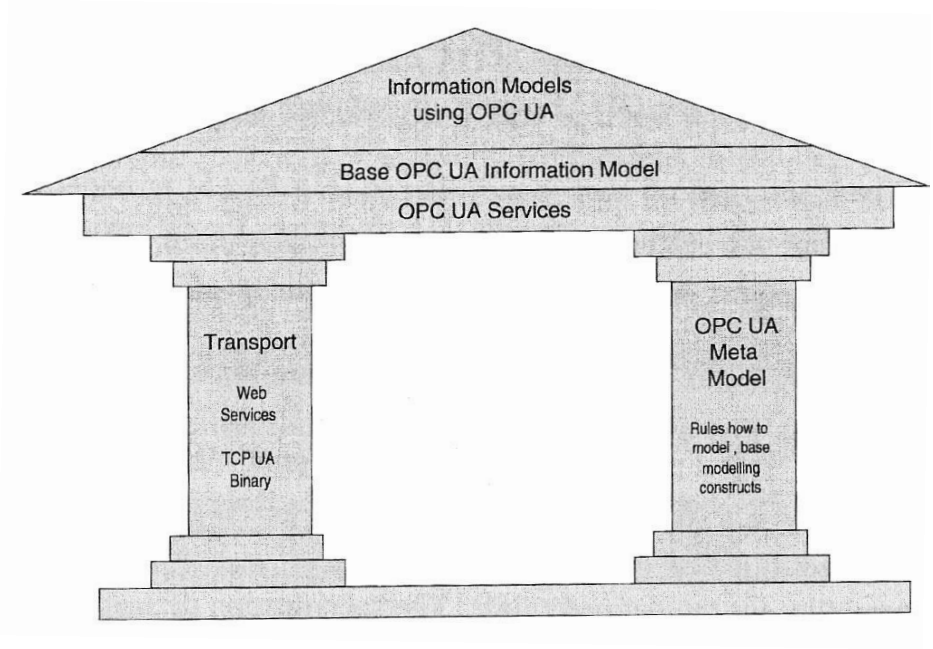
OPC Unified Architecture (**OPC-UA**) nasce quindi dalla volontà di creare un vero sostituto di tutte le esistenti versioni COM-based senza la perdita di nessuna funzionalità e senza alcun problema di efficienza; soddisfa quindi la necessità di interfacce platform-independent e permette la creazione di ricchi ed estensibili modelli dati per descrivere sistemi complessi.

La seguente tabella riassume i requisiti soddisfatti con OPC-UA.

Comunicazione tra sistemi distribuiti	Modello Dati
Robustezza e fault-tolerance	Comune per tutti gli OPC data
Indipendenza dalla piattaforma	Orientato agli oggetti
Scalabilità	Tipi estensibili
Alte performance	Metadati
Internet e firewall	Dati complessi e metodi
Sicurezza e controllo degli accessi	Scalabilità da modelli semplici a complessi
Interoperabilità	Modello di base astratto
Ridondanza dei server	Base per altri modelli dati standard

Per raggiungere tali requisiti OPC-UA è strutturata in diversi strati, mostrai dalla figura seguente. Le componenti principali sono i protocolli di trasporto e il modello dei dati. Nel livello di trasporto sono definiti due diversi meccanismi ottimizzati per differenti casi d'uso: un protocollo TCP binario per la comunicazione intranet ad alta performance e un protocollo basato sui Web Services per la comunicazione internet firewall-friendly. Entrambi utilizzano lo stesso modello di sicurezza message-based utilizzato nei Web Services.

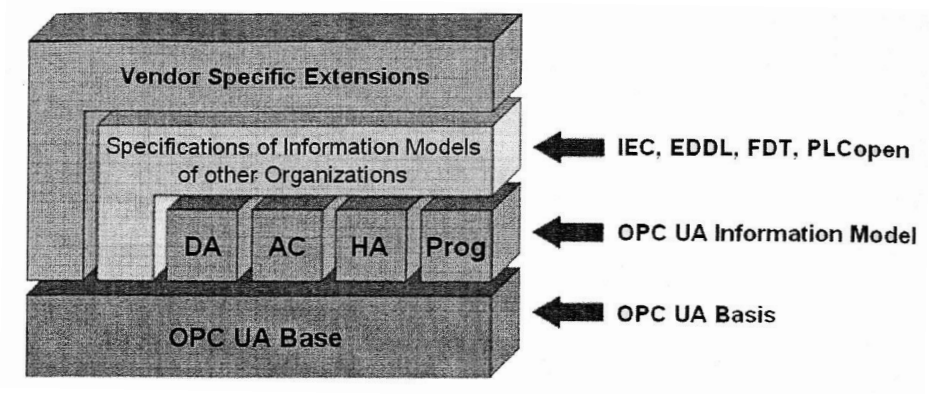
Il modello dei dati definisce le regole e gli elementi base necessari per fornire un valido modello informativo e comprende inoltre elementi avanzati come quelli per descrivere le macchine a stati. Gli elementi base possono essere estesi da altri modelli informativi ad un più alto livello.



I servizi UA (OPC UA Services in figura) costituiscono interfacce tra server e client, i primi intesi come fornitori di modelli informativi e gli altri come "consumatori" di tali modelli; essi utilizzano i meccanismi di trasporto per lo scambio dei dati tra client e server.

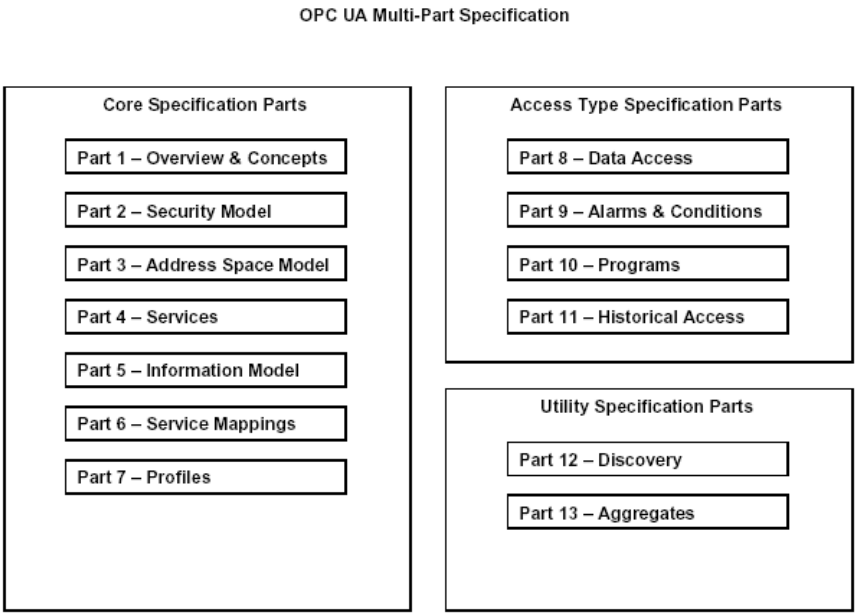
Uno dei concetti alla base di OPC-UA è che un client può accedere alla più piccola porzione di dati di un sistema complesso senza essere a conoscenza dell'intero modello informativo.

La figura seguente mostra i differenti strati del modello informativo definito da OPC:

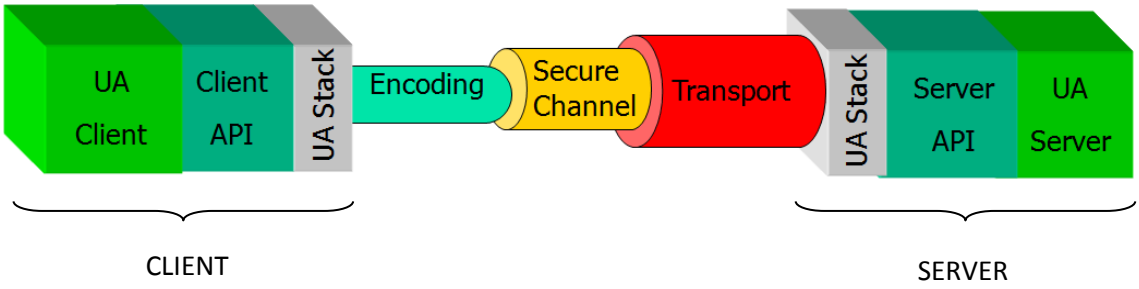


DA definisce estensioni automation-data-specific come i modelli per la descrizione di dati analogici e digitali e quelli per la qualità del servizio; tutte le altre caratteristiche di DA sono già ricoperte dall'architettura di base. AC (*Alarm & Conditions*) consiste in un modello avanzato per la gestione degli allarmi ed il monitoraggio dello stato dei processi. HA (*Historical Access*) definisce i meccanismi per accedere alla cronologia dei dati e degli eventi. Prog (*Programs*) specifica i metodi per avviare, manipolare e monitorare l'esecuzione dei programmi.

Nella figura seguente sono rappresentate le varie componenti dello standard OPC-UA, approvate dalla IEC (*International Electrotechnical Commission*):



In OPC-UA, pur essendo utilizzata un'architettura client-server, è tipico che un'applicazione rivesta entrambi i ruoli, ciò perché spesso nei dispositivi fisici è integrato anche il lato server (comunicazione device to device). Una tipica applicazione OPC-UA è composta da tre strati software descritti nella figura sottostante:



L'intero stack software può essere implementato con C, C++,.NET o java; OPC-UA non si limita a tali linguaggi e piattaforme ma questi sono attualmente gli unici sviluppati.

Un'applicazione OPC-UA è un sistema che desidera esporre o "consumare" dati e contiene sia le sue funzionalità specifiche che il loro mapping a OPC-UA attraverso OPC-UA Stack (che implementa soltanto i meccanismi di comunicazione) e OPC-UA SDK (Software Development Kit). L'utilizzo di una SDK riduce lo sforzo in fase di sviluppo e facilita una veloce interoperabilità.

L'OPC-UA Stack implementa i diversi meccanismi di trasporto ed è suddiviso in tre strati:

- **Message Serialization**; definisce i metodi per serializzare i dati scambiati in modalità binaria o XML.
- **Message Security**; specifica come i messaggi devono essere protetti utilizzando gli standard di sicurezza dei Web Services o una versione binaria di essi.
- **Message Transport**; definisce il protocollo di rete utilizzato, che può essere UA TCP o HTTP e SOAP per i Web Service.

2 Information Modeling (Richiesto per l'Esame)

Nei precedenti standard OPC era possibile gestire soltanto dati "semplici"; per esempio, nel caso della temperatura misurata da un sensore, le sole informazioni disponibili per comprendere la semantica del dato erano il nome della variabile e pochi altri elementi come l'unità di misura. OPC-UA invece, a tale fine, fornisce funzionalità maggiori; ad esempio, nel caso precedente, la misura della temperatura può essere accompagnata dal tipo specifico del sensore che ha effettuato la misurazione.

Le specifiche di base di OPC-UA forniscono soltanto l'infrastruttura della model information; l'informazione può essere poi modellata direttamente dai produttori (vendors), il che ovviamente porta a differenti soluzioni per dati di tipi simili. Questo rende complicato l'accesso ai dati da parte dei client; per evitare tutto ciò OPC-UA offre la possibilità di definire delle estensioni a partire da un information model di base fornito da OPC Foundation.

Ogni produttore può creare le proprie estensioni con le informazioni specifiche dei suoi dispositivi. I client sono in grado di accedere a tali informazioni allo stesso modo indipendentemente dal produttore, perché saranno rese disponibili utilizzando lo stesso modello di base.

I principi di base dell'information modeling in OPC-UA sono i seguenti:

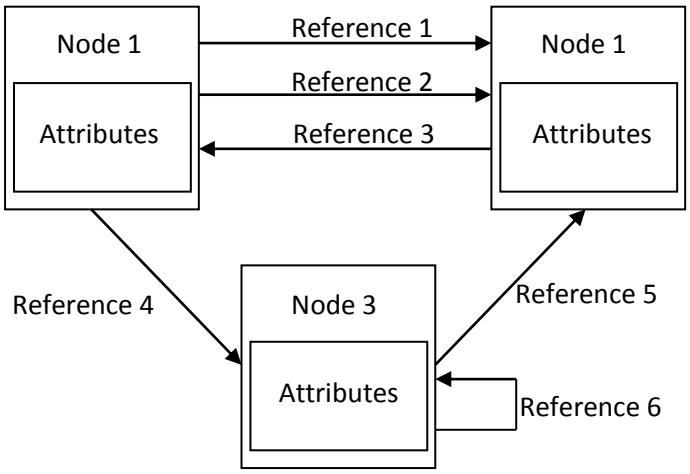
- ***Utilizzo di tecniche object-oriented, incluse gerarchie di tipi e ereditarietà***. I client possono gestire istanze dello stesso tipo nella stessa maniera. Gerarchie di tipi consentono di agire sui tipi di base senza considerare quelli specializzati.
- ***Modalità di accesso ai tipi identica a quella delle istanze.***
- ***Possibilità di esporre le informazioni in vari modi sfruttando reti completamente connesse di nodi.*** OPC-UA consente di esporre differenti semantiche e relazioni di una stessa rete di nodi, in base al caso d'uso.
- ***Estensibilità delle gerarchie dei tipi e dei tipi di collegamento tra i nodi.***
- ***Nessuna limitazione su come modellare l'informazione.*** Questo permette di costruire sempre un appropriato modello dei dati in base alle necessità.
- ***Information Modeling collocato sempre lato server.***

I concetti di base dell'information modeling in OPC-UA sono i *Nodes* (nodi) e le *References* (collegamenti) tra nodi. Nodi e References vengono raggruppate nell'Address Space che costituisce il set di informazioni esposte dal Server OPC UA e accessibile da qualunque Client OPC UA.

2.1 Nodes (Richiesto per l'Esame)

I nodi possono essere di differenti *NodeClasses* in base al loro obiettivo specifico. Ci sono nodi che rappresentano istanze, altri che rappresentano tipi, etc.

La seguente figura mostra un esempio di Nodes e References; come si vede i Nodes (i Node1, Node2 e Node3 in figura) contengono Attributes e sono connessi attraverso References.



Gli *Attributes* si usano per descrivere i Nodes ed in base alla NodeClass un nodo può avere un differente set di attributi. In particolare, gli *Attributes* di un *Node* dipendono dal proprio NodeClass di appartenenza; alcuni di essi comunque sono presenti in tutti i nodi.

Un Client può accedere al valore degli Attrributes utilizzando i servizi OPC UA Read, Write, Query e Subscription/MonitoredItem; tali servizi verranno descritti nel seguito.

Ogni Attributo è contraddistinto da: Attribute id, il nome, la descrizione, il tipo di dato di appartenenza, e l'indicazione circa il fatto che l'attributo sia mandatory/optional (obbigatorio/opzionale).

I valori che ogni singolo attributo deve assumere, vengono assegnati quando il Node è istanziato nell'AddressSpace. l'indicatore mandatory/opzional associato ad un singolo Attributo indica se tale Attributo debba essere istanziato o meno.

2.2 References (Richiesto per l'Esame)

Concettualmente, una Reference descrive la relazione tra due nodi; è identificata univocamente dal nodo sorgente, dal nodo target, dalla ReferenceType (la semantica della reference) e dalla sua direzione.

Considerare le References come puntatori ad altri nodi aiuta a capire meglio il loro ruolo; per ogni nodo, i server OPC-UA possono esporre una sola Reference in una direzione e la Reference può "puntare" ad un nodo, che può esistere nello stesso Server, non più esistente nel server locale, oppure può appartenere ad un altro server. Dunque una Reference contiene il NodeId del nodo a cui punta, il relativo OPC UA Server dove tale nodo è mantenuto, la semantica della Reference (il ReferenceType) e la sua direzione.

Il Nodo che contiene la Reference è detto SourceNode e il Node a cui punta viene detto TargetNode. Il TargetNode di una Reference può appartenere allo stesso AddressSpace o ad un AddressSpace di un altro OPC UA Server.

Una Reference viene definita come istanza di un particolare NodeClasse detto ReferenceType Node.

2.3 Standard NodeClasses (Richiesto per l'Esame)

Come è stato detto in OPC UA esistono i NodeClasses utilizzati per la definizione dei Nodes e per l'istanza dei References. Tutti i NodeClasses sono derivati da un Base NodeClass.

I NodeClasses definiti nello standard sono: Variable, VariableType, Object, ObjectType, ReferenceType, DataType, Method, View.

2.4 Base NodeClasse (Richiesto per l'Esame)

In OPC UA esiste una particolare NodeClass da cui tutte le altre NodeClass vengono derivate. Come detto, essa si chiama Base NodeClass.

Il Base NodeClass presenta un determinato numero di attributi e non presenta References. Tutti gli attributi del Base NodeClass vengono ovviamente ereditati da tutti i NodeClasses, e dunque sono comuni a tutti i Nodes di OPC UA.

La seguente tabella riassume tali *Attributes*, che risultano comuni a tutti i NodeClasses e dunque a tutti i Nodes del NameSpace di OPC UA.

Attribute	DataType	Descrizione
NodeId	NodeId	Identifica univocamente un nodo in un server OPC-UA, per poterlo correttamente specificare nell'ambito dei servizi OPC-UA.
NodeClass	NodeClass	Enumerazione che identifica la NodeClass di appartenenza di un nodo, come ad esempio Object o Method.
BrowseName	QualifiedName	Identifica il nodo quando il server viene visitato col servizio Browse. Non è localizzato, ossia il suo valore non dipende dal valore di LocaleId, ossia dai settaggi relativi ad una determinata lingua di appartenenza.
DisplayName	LocalizedText	Contiene il nome del nodo che dovrebbe essere utilizzato nella user interface. Per questo motivo, è localizzato, cioè dipende dal valore di LocaleId, ossia dal setting linguistico di un determinato paese di appartenenza. Un Server potrebbe fornire diversi valori per il DisplayName, a seconda della lingua e del settaggio LocaleId.
Description	LocalizedText	Attributo opzionale che contiene una descrizione testuale localizzata del nodo.
WriteMask	UInt32	E' opzionale e specifica quali attributi del nodo sono modificabili.
UserWriteMask	UInt32	E' opzionale e specifica quali attributi del nodo sono modificabili dall'utente attualmente collegato al server.

Il NodeId identifica in modo univoco un Node nell'Address Space di un Server. Il Server restituisce il NodeId quando il Client effettua un browsing delle informazioni oppure il Client indirizza un particolare NodeId nella chiamata ai servizi OPC UA. Il NodeId ha la seguente struttura.

Struttura del NodeId		
Name	**Type**	**Description**
namespaceIndex	UInt16	Indice del URI del namespace
identifierType	Enum (Numeric, String, GUID, Opaque)	Formato e Tipo di Dato dell'Indentifier
Identifier	Tipo dipendente dall'identifierType	Identificatore di un Nodo

Il namespaceIndex del NodeId è definito al fine di introdurre una semplificazione nella gestione delle informazioni mantenute dal Server. Ciascun Server espone una variabile chiamata NamespaceArray, definito come un array di URI dello spazio dei nomi. Gli indici nella tabella NamespaceArray sono indicati come namespaceIndexes, che sono utilizzati nel NodeId specificato nei servizi OPC UA. L'indice 0 di questo array è riservato per lo spazio dei nomi OPC UA , e l'indice 1 è riservato per il server locale. Il Namespace URI OPC UA (indice 0) è: http://opcfoundation.org/UA. I client possono leggere l'intera tabella NamespaceArray o possono leggere le singole voci della tabella NamespaceArray. Il server non potrà modificare o eliminare le voci della tabella mentre ogni cliente ha una sessione aperta al server, perché i client possono memorizzare nella cache la tabella NamespaceArray che deve essere consistente sempre con quella del Server. Un Server può aggiungere voci alla tabella, anche se i clienti sono collegati al server. Si raccomanda che i server non cambino gli indici della tabella NamespaceArray, ma si limitino solo ad aggiungere nuove entries. Tuttavia, potrebbe non essere sempre possibile per i server evitare la modifica di indici nella tabella NamespaceArray. Dunque i client dovrebbero sempre verificare che i namespaceIndexes mantenuti in cache non siano cambiati.

L'attributo IdentifierType identifica il tipo utilizzato per l'identifier del *NodeId*. I possibili valori sono: numerico, string, Global Unique Identifier (GUID), Opaque (namespace specific format).

L'attributo identifier permette l'identificazione univoca di un NodeId nell'ambito dello spazio del NamespaceArray individuato dall'indice namespaceIndex; per ciascun elemento del NamespaceArray, l'identifier identifica un particolare nodo. Il tipo di dato e il formato dell'identifier è individuato dall'IdentifierType.

2.5 ReferenceType NodeClass (Richiesto per l'Esame)

In precedenza è stata descritta la struttura di un Node ed è stato detto che ogni Reference è definita come istanza di un ReferenceType Node.

I ReferenceType Node sono visibili nell'AddressSpace, sono in genere organizzati in maniera gerarchica e sono definiti utilizzando il ReferenceType NodeClass, descritto nel seguito.

Come tutti i Nodes, il ReferenceType NodeClass è definito in termini di Attributes e References.

Nella tabella seguente sono descritti gli Attributes del ReferenceType NodeClass.

Attribute	DataType	Descrizione
Base NodeClass Attributes		Sono tutti gli attributi ereditati dal Base NodeClass, descritti in precedenza.
IsAbstract	Boolean	Se TRUE, specifica che il ReferecenceType è utilizzato solo per scopi organizzativi per una migliore definizione della gerarchia dei ReferenceType.
Symmetric	Boolean	Indica se la Reference è simmetrica, cioè se è valida in entrambe le direzioni.
InverseName	LocalizedText	Attributo opzionale che specifica la semantica della Reference nella direzione inversa (per la direzione diretta si usano BrowseName e DisplayName). Può essere applicato solo a References asimmetriche e deve essere obbligatoriamente presente in caso di ReferenceType non astratto.

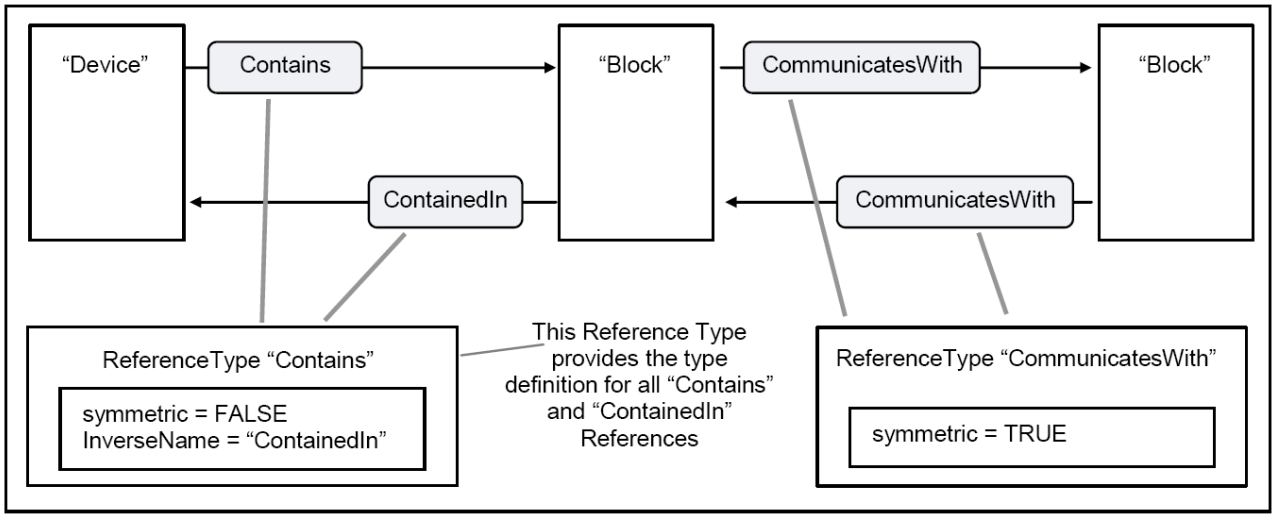
Come si vede dalla precedente tabella, il ReferenceType NodeClass eredita gli Attributes del Base NodeClass. In particolare l'attributo BrowseName è utilizzato per specificare il meaning (semantica) del ReferenceType dal punto di vista del SourceNode, ossia che significa la particolare reference per il nodo sorgente. Ad esempio il valore del BrowseName "Contains" è usato nelle References che specificano che il SourceNode contiene il TargetNode. L'attributo DisplayName contiene una traduzione del BrowseName, visto che il DiplayName è localizzato ossia dipende dai settaggi linguistici.

Se l'attributo IsAbstract indica se il ReferenceType è astratto. I ReferenceTypes astratti non possono essere istanziati e sono usati solo per ragioni organizzative, ad esempio per specificare semantiche o vincoli.

L'attributo Symmetric è usato per indicare se il significato del ReferenceType è lo stesso sia per il SourceNode sia per il TargetNode. Se il ReferenceType è simmetrico, allora l'attributo InverseName deve essere omesso. Esempi di ReferenceType simmetrici sono "Connects To" oppure "Communicates with", visto che entrambi implicano la stessa semantica dal SourceNode e dal TargetNode. Un altro esempio è "is-sibling-of" (è fratello di).

Se il ReferenceType è non-symmetric e non è astratto, allora l'attributo InverseName deve essere settato, visto che l'attributo InverseName specifica il significato del ReferenceType visto dal TargetNode. Esempi di ReferenceType non-symmetric sono "Contains", "Contained In", "Receives From" e "Sends To". Altri esempi sono "has-parent" (ha un genitore), in una direzione, o "is-child-of" (è figlio di), nell'altra direzione.

La seguente figura mostra un esempio di References symmetric e non-symmetric e mostra l'utilizzo dell'attributo BrowseName e InverseName.



Nella tabella seguente sono descritti le References del ReferenceType NodeClass.

Attribute	Descrizione
HasProperty	E' usato per specificare le Proprietà di un ReferenceType e può unicamente riferirsi (puntare a) Nodes appartenenti al NodeClass Variable
HasSubtype	E' usato per identificare i sottotipi di un ReferenceType.

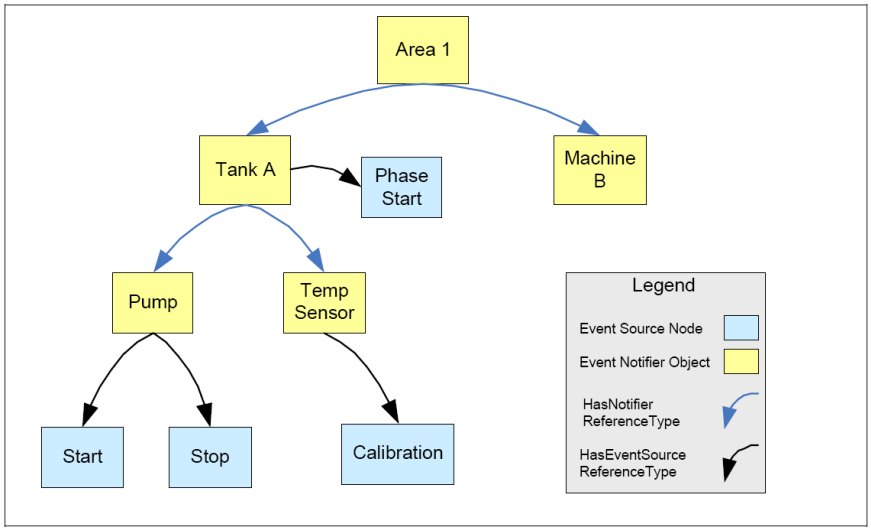
2.6 Object NodeClass (Richiesto per l'Esame)

Gli Objects sono fondamentali in quanto utilizzati in OPC UA per rappresentare: sistemi, componenti di sistemi, oggetti reali e oggetti software. Gli Object sono definiti usando l'Object NodeClass.

In OPC UA un Object contiene sempre Variables, Methods e altri Objects. Inoltre, un Object può essere un *EventNotifier* a cui i client possono registrarsi per essere avvisati sugli *Events*.

Gli *Events* rappresentano specifiche situazioni che possono accadere improvvisamente in un sistema, quali ad esempio cambi di configurazione e errori di sistema. Sono ricevuti dai client tramite opportune notifiche chiamate Event Notifications. Gli Events non sono direttamente visibili nell'Address Space, ma sono accessibili solo tramite Object e View (descritte nel seguito). Dunque Object e View sono gli unici Nodes in OPC UA che si possono utilizzare per sottoscriversi ad un Evento; la sottoscrizione ad un evento avviene tramite il settaggio di un particolare Attribute dell'Object o View, chiamato EventNotifier Attribute (in particolare viene settato un bit al suo interno chiamato SubscribeToEvent). OPC-UA definisce una gerarchia di base di EventTypes che può essere estesa; ogni server espone la propria gerarchia in modo che i client possano accedervi: con questa informazione i client possono creare filtri che indicano a quali eventi sono interessati, e per ognuno di essi gli attributi desiderati. Il Client utilizzerà i meccanismi basati sui servizi di Subscriptions e di MonitoredItem (si vedano le sezioni successive sui Servizi) per ricevere notifiche su eventi e per definire tali filtri. Come detto, la sorgente di un Evento è tipicamente un Object o una View con l'attributo EventNotifier (o meglio uno dei suoi bit, il bit SubscribeToEvent) settato; ma la sorgente di un evento può anche essere un generico Node, chiamato Source Node. In tal caso viene usata una particolare

reference (HasEventSource) per consentire all'Object o View con attributo EventNotifier di specificare il Source Node a lui correlato. Il verificarsi di un evento nell'Object con EventNotifier abilitato produce la sottoscrizione dell'evento da parte del Source Node ad esso connesso. Nella figura successiva, si vede come i Nodi "Start", "Stop", "Calibration" e "Phase Start" sono connessi agli Event Notifier Object "Pump", "Tank A" e "Temp Sens", tramite il Reference HasEventSource (sono object con attributo EventNotifier abilitato). Ciò significa che un Evento rilevato ad esempio dall'Object Pump verrà diffuso dai Source Node "Start" e "Stop". Esiste un altro Reference usato per l'Event Notifier Object (ossia Object con l'attributo EventNotifier abilitato): HasNotifier. Esso permette ad un Object di distribuire l'occorrenza di un evento a più Object con attributo EventNotifier abilitato. Nella figura mostrata in seguito, un Evento rilevato dall'Object "Area 1" diffonderà tale Evento agli Object "Machine B", "Tank A" e a tutti i nodi e oggetti sottostanti all'Object "Tank A", mostrati in figura.



L'Object NodeClass possiede sia Attributi sia References. Per quanto riguarda gli Attributi, esso eredita tutti i Base NodeClass Attributes e in più possiede l'attributo EventNotifier, già descritto.

Attribute	DataType	Descrizione
Base NodeClass Attributes		
EventNotifier	Byte	Rappresenta una maschera di bit che specifica se un Object può essere usato per sottoscriversi ad eventi e se, in caso affermativo, la cronologia di questi eventi può essere accessibile o modificabile. Tra i Bit ve ne è uno da ricordare: SubscribeToEvents. Questo bit, se settato, indica che l'Object può essere usato per sottoscriversi ad eventi.

I References di un Object NodeClass sono specificati nel seguito:

References	Descrizione
HasComponent	Identifica i DataVariable (descritti nel seguito), Metodi (descritti nel seguito) e altri Objects contenuti nell'Object
HasProperty	Identifica le Properties (descritti nel seguito) dell'Object

HasModellingRule	
HasTypeDefinition	Identifica (punta a) all'ObjectType (descritto nel seguito) che viene usato per la definizione del tipo di appartenenza dell'Object.
HasEventSource	Esiste solo se il bit SubscribeToEvents dell'EventNotifier è settato e in tal caso permette di indicare (punta al) il Source Node. Si veda l'esempio precedente per capire meglio questo reference.
HasNotifier	Esiste solo se il bit SubscribeToEvents dell'EventNotifier è settato e in tal caso permette di puntare ad un altro Object Event Notifier, ossia anch'esso con bit SubscribeToEvents dell'EventNotifier settato. Si veda l'esempio precedente per capire meglio questo reference.
Organizes	
HasDescription	

2.7 ObjectType NodeClass (Richiesto per l'Esame)

Un'importante caratteristica di OPC-UA è quella di fornire informazioni sui tipi non solo a livello di tipo di dato (per esempio se un Value è un Int32 o una String) ma anche a livello di oggetto. Ciò consente di recuperare ad esempio l'informazione su un tipo specifico di sensore che misura una temperatura. Nei precedenti standard OPC per fare questo non c'era altro modo che utilizzare convenzioni specifiche del produttore; in OPC-UA si può rendere disponibile questa informazione definendo un tipo per i sensori di temperatura e creando oggetti di quel tipo. Si potranno poi ottenere tipi vendor-specific ereditando dai tipi di base creati.

Gli *ObjectTypes* vengono utilizzati per fornire la descrizione completa di Objects. Gli ObjectTypes sono definiti utilizzando l'ObjectType NodeClass.

2.8 Variable NodeClass (Richiesto per l'Esame)

Le Variables sono utilizzate per rappresentare valori semplici e complessi; le variabili appartengono a determinati Variable Types. Una Variable può essere usata, ad esempio, per rappresentare la temperatura misurata da un sensore.

Le Variables possono essere definite come Properties o come DataVariables di Nodes contenuti nell'AddressSpace.

Le Variables sono definite utilizzando il NodeClass Variable. La seguente tabella mostra gli Attributi di tale NodeClass.

Attribute	DataType	Descrizione
Base NodeClass Attributes		Attributi ereditati dal Base NodeClass.
Value	*Il relativo tipo di dato è definito dall'attributo DataType specificato di*	Rappresenta il valore più recente della Variable che il Server possiede per essa. Il suo tipo di dato è specificato dal dall'attributo DataType.

	seguito	
DataType	NodeId	I DataTypes sono rappresentati nell'Address Space come Nodes. Questo attributo contiene l'id del Node che rappresenta il tipo della Variable.
ValueRank	Int32	Specifica se il Value è un array e ne specifica il numero di dimensioni (non il numero di elementi per ogni dimensione).
ArrayDimensions	UInt32[]	Attributo opzionale che, in caso di array, specifica il numero di elementi per ciascuna dimensione.
AccessLevel	Byte	Maschera di bit che indica se l'attributo Value è leggibile/scrivibile e se il contenuto rappresenta il valore corrente o un dato storico.
UserAccessLevel	Byte	Identico a AccessLevel, ma riferito ai permessi dell'user. Indica se l'attributo Value è leggibile/scrivibile tenendo conto dei permessi dell'utente.
MinimumSamplingInterval	Duration	Attributo opzionale che indica quanto frequentemente il server può rilevare i cambiamenti dell'attributo Value. Per dati non direttamente accessibili dal server, ad es. la temperatura rilevata da un sensore, il server deve periodicamente effettuare un polling al dispositivo fisico e quindi questo sampling interval dovrà essere per forza di una certa consistenza. Un valore 0 di questo attributo significa che il Server monitora il cambiamento di valore continuamente.
Historizing	Boolean	Specifica se attualmente il server memorizza la cronologia dei cambiamenti della variabile.

L'attributo DataType si usa per definire il tipo di dato utilizzato per l'attributo Value e contiene il NodeId di un certo nodo di tipo DataType. Ogni server può definire i propri DataTypes, che non sono altro che nodi del suo Address Space, accessibili dai client. OPC-UA comprende quattro tipi di DataTypes:

- **Built in**: sono definiti in OPC-UA e non possono essere estesi; comprendono tipi di base come Int32, Boolean, Double, ed anche tipi specifici dello standard come NodeId, LocalizedText e QualifiedName.
- **Simple**: sono sottotipi dei DataTypes built in, come ad es. Duration, che è un sottotipo di Double che descrive un intervallo di tempo in millisecondi. Ogni Information Model può definire i propri.
- **Enumeration**: consistono in un insieme discreto di valori nominali, sono gestiti alla stessa maniera dei DataTypes built in. Un esempio è il DataType NodeClass. Ogni Information Model può definire i propri.
- **Structured**: rappresentano dati strutturati, e sono il tipo di dati ridefinibile più potente. Un esempio è il DataType Argument che definisce un argomento di un metodo. Ogni Information Model può definire i propri.

I References di un Variable NodeClass sono specificati nel seguito:

References	Descrizione
HasModellingRule	

HasProperty	Identifica le Properties (descritti nel seguito) di una DataVariable
HasComponent	Questo Reference è utilizzato solo da DataVariable complesse per identificare le DataVariables componenti. Le Properties non possono utilizzare questo Reference.
HasTypeDefinition	Identifica (punta a) al VariableTye della Variable. Per ogni variable è ammesso solo un Reference HasTypeDefinition che punta ad un VariableType.

OPC-UA definisce due tipi di Variables: DataVariables e Properties. Entrambe utilizzando il NodeClass Variable appena descritto, ma con alcune limitazioni e vincoli.

Le *DataVariables* sono utilizzate per rappresentare i dati di un Object, come la temperatura misurata da un sensore; esse possono essere complesse, cioè avere sottovariabili contenenti parte dei dati e proprietà che le descrivono.

Le *Properties* si utilizzano per descrivere le caratteristiche di un nodo, per esempio l'unità di misura della temperatura misurata. In generale le Properties sono usate quando devono essere descritte caratteristiche non presenti tra gli Attributes del nodo. Un altro esempio di Properties sono gli InputArguments e gli OutputArguments dei Methods. Le Properties sono semplici, non possono esporre sottovariabili e rappresentano sempre le "foglie" di ogni gerarchia.

2.9 VariableType NodeClass (Richiesto per l'Esame)

Analogamente a quanto detto per gli ObjectType, i VariableType definiscono la semantica di una Variable oppure permettono di restringere il dominio dell'attributo Value; ad esempio in OPC-UA è definito un tipo di base per le variabili chiamato BaseDataVariableType che non prevede alcuna limitazione [UA parte 5]. Con un suo sottotipo si potrebbe definire una variabile "contatore" la cui semantica imponga, ad es., che il tipo di dato di Value sia uno scalare.

I VariableType vengono definiti utilizzando il VariableType NodeClass.

2.10 Views

Un Server OPC UA può offrire un vasto insieme di Nodes, di cui un particolare client è interessato ad un limitato sottoinsieme. OPC UA permette la definizioni di sottoinsiemi del proprio Address Space, differenti per diversi Client. Ogni client accede ad una *View* (vista) che restringe l'intero insieme di Nodes ad un suo sottoinsieme definito in base alle esigenze specifiche del client.

Ogni Node contenuto in una View può contenere solo un sottoinsieme delle sue References; il numero e il tipo di References visibili in ogni View per ogni Node viene deciso in fase di definizione della View.

Esiste il View Node che rappresenta la root dei Nodes per quella View. Una View è rappresentata nell'Address Space come un nodo che fornisce un punto d'ingresso ai dati che si vogliono mostrare. Tutti i nodi che fanno parte di una View sono accessibili a partire dal nodo di partenza.

Come per gli Object, anche per le View è possibile settare il bit SubscribeToEvent dell'attributo EventNotifier, in modo che la View può essere usata per sottoscrivere a Events.

Le Views sono definite utilizzando la View NodeClass. Le due tabelle seguenti mostrano gli Attributes e le References.

Per quanto riguarda gli Attributi, la View NodeClass eredita tutti i Base NodeClass Attributes e in più possiede gli attributi ContainsNoLoop e EventNotifier, già descritto.

Attribute	DataType	Descrizione
Base NodeClass Attributes		Attributi ereditati dal Base NodeClass
ContainsNoLoop	Boolean	Indica se i nodi contenuti nella vista sono organizzati in una struttura gerarchica senza alcun loop. Se settato a TRUE, questo attributo indica che i References non contengono loop nel contesto della View corrente; ciò significa che partendo dal Nodo A e seguendo la forward Reference, il Nodo A non sarà visitato un'altra volta.
EventNotifier	Byte	Rappresenta una maschera di bit che specifica se un Object può essere usato per sottoscriversi ad eventi e se, in caso affermativo, la cronologia di questi eventi può essere accessibile o modificabile. Tra i Bit ve ne è uno da ricordare: SubscribeToEvents. Questo bit, se settato, indica che l'Object può essere usato per sottoscriversi ad eventi.

I References di una View NodeClass sono specificati nel seguito:

References	Descrizione
HierarchicalReferences	Servono per contraddistinguere i top level Nodes in una View
HasProperty	Identifica le Properties della View

2.11 Methods NodeClass

I *Method* rappresentano funzioni richiamabili (con parametri di input e output), che vengono invocate con il Servizio Call di OPC UA. Possono essere metodi, ad esempio, quelli per aprire una valvola, avviare un motore, o anche calcolare i risultati di una simulazione in base ai parametri d'ingresso forniti. Un Method è esposto all'esterno soltanto mediante la sua signature ed OPC-UA non impone alcun vincolo su come deve essere effettivamente implementato.

Gli argomenti di input ed output di un Method non sono specificati da attributi ma dalle particolari Variables dette *Properties*.

I Methods sono definiti usando i Method NodeClass.

La seguente tabella riporta tutti gli attributi e una parte delle property specifici del Method NodeClass.

Attribute	DataType	Descrizione
Base NodeClass Attributes		
Executable	Boolean	Un flag che indica se il Method è attualmente eseguibile.
UserExecutable	Boolean	Un flag che indica se il Method può essere attualmente

		eseguito da un user, tenendo in conto i suoi diritti di accesso.
Property		
InputArguments	Argument[]	Property opzionale che definisce un array di argomenti di input del metodo; l'ordine dell'array definisce l'ordine degli argomenti.
OutputArguments	Argument[]	Analogo a InputArguments ma relativo ai parametri di output.

Struttura del DataType *Argument*:

Name	DataType	Descrizione
Name	String	Nome dell'argomento.
DataType	NodeId	NodeId del DataType Node relativo all'argomento.
ValueRank	Int32	Indica se l'argomento è formato da uno scalare, da un array o da una matrice
ArrayDimensions	UInt32 []	Definisce la grandezza dell'array o della matrice.
Description	LocalizedText	Descrizione dell'argomento.

2.12 Tabella Riassuntiva degli attributi presenti nei NodeClasses

Attribute	Variable	Variable Type	Object	Object Type	Reference Type	DataType	Method	View
AccessLevel	M							
ArrayDimensions	O	O						
BrowseName	M	M	M	M	M	M	M	M
ContainsNoLoops								M
DataType	M	M						
Description	O	O	O	O	O	O	O	O
DisplayName	M	M	M	M	M	M	M	M
EventNotifier			M					M
Executable							M	
Historizing	M							
InverseName					O			
IsAbstract		M		M	M	M		
MinimumSamplingInterval	O							
NodeClass	M	M	M	M	M	M	M	M
NodeId	M	M	M	M	M	M	M	M
Symmetric					M			
UserAccessLevel	M							
UserExecutable							M	
UserWriteMask	O	O	O	O	O	O	O	O
Value	M	O						
ValueRank	M	M						
WriteMask	O	O	O	O	O	O	O	O

3 I Servizi (Richiesto per l'Esame)

Il funzionamento di un servizio OPC UA prevede pattern di comunicazione derivati da quelli dei Web Service, in cui il servizio si compone di messaggi di richiesta e di risposta (Request, Response).

Per invocare un Service un client invia un messaggio di *Request* al server; dopo aver processato la richiesta, il server invia un messaggio di *Response* al client. Poiché questo scambio di messaggi è asincrono, tutte le invocazioni di servizi sono asincrone per definizione. Dopo aver mandato il messaggio di Request, l'applicazione client può continuare la propria esecuzione in attesa del messaggio di Response.

3.1 Timeout (Richiesto per l'Esame)

In OPC UA la comunicazione dei dati è concepita per poter funzionare tra diversi sistemi, connessi tipicamente in una rete; la comunicazione è quindi soggetta in qualsiasi momento a possibili interruzioni che devono essere rilevate e gestite correttamente. Per questo motivo ogni chiamata di un servizio ha come parametri dei *Timeout* definiti dal client, che servono proprio a rilevare l'assenza della comunicazione.

3.2 Request e Response Headers (Richiesto per l'Esame)

I messaggi di Request e Response relativi a qualunque servizio contengono il medesimo tipo di Header (RequestHeader - ResponseHeader).

namespace: Opc.Ua.RequestHeader

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

namespace: Opc.Ua.ResponseHeader

Proprietà pubbliche	Descrizione
DateTime Timestamp	Istante di tempo in cui il server ha inviato la risposta.
uint RequestHandle	Identificativo assegnato alla chiamata di servizio definito dal client.
StatusCode ServiceResult	Classe definita dallo standard per descrivere il risultato della chiamata del servizio. Contiene un uint a 32 bit in cui i 16 più significativi rappresentano il valore numerico dell'errore ed i

	rimanenti contengono informazioni aggiuntive. In particolare, i 2 bit più significativi si riferiscono all'attendibilità del risultato (00=Good, 01=Uncertain, 10=Bad, 11=not used).
DiagnosticInfo ServiceDiagnostics	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice status code.

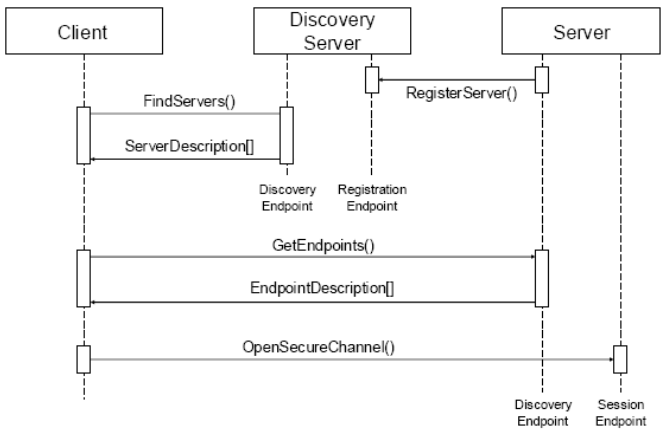
4 Discovery Services Set (Richiesto per l'Esame)

Il Discovery Service Set è l'insieme dei servizi utilizzati dai client per individuare gli endpoints implementati da un Server OPC UA e per leggere le configurazioni di sicurezza contenute in tali endopoint. Le informazioni contenute in ciascun endpoint verranno utilizzate dal Client per creare un SecureChannel con il Server, se il client supporta una o più configurazioni di sicurezza esposte dagli endpoint.

Ogni Server OPC UA deve esporre almeno un Discovery Endpoint, attraverso cui il Client accederà per leggere le configurazioni di sicurezza contenute negli endpoint; il servizio utilizzato a tal fine è il GetEndpoints. L'accesso da parte del Client al Discovery Endpoint avviene senza la necessità di stabilire una Sessione; inoltre la lettura delle relative configurazione avviene totalmente in chiaro, senza alcun meccanismo di sicurezza.

La sequenza logica di funzionamento è la seguente:

1. Il server attivo si registra presso un Discovery Server (RegisterServer). Questa operazione è opzionale se il Client conosce già il Server
2. Il client richiede al Discovery Server la lista dei server attivi (FindServers), selezionandone uno. Questa operazione è opzionale se il Client conosce già il Server
3. Il client richiede al Server l'elenco e il contenuto degli endpoints, tramite il servizio GetEndpoints invocato attraverso il Discovery Endpoint del Server. A tal fine, il client specificherà l'URL del Discovery Endpoint quando invocherà il servizio GetEndpoints
4. Sulla base delle configurazioni di sicurezza contenute negli endpoints ricevuti, il client può creare un canale sicuro con il server (OpenSecureChannel). Ovviamente può succedere che le configurazioni di sicurezza sono tutte incompatibili con le caratteristiche del client, che dovrà rinunciare alla creazione di una canale sicuro.



Ovviamente tutta questa procedura può essere semplificata nel caso in cui il client conosca già il server a cui connettersi (indirizzo preconfigurato, ad esempio); in tal caso le azioni che vengono intraprese sono la terza e la quarta.

4.1 FindServers (Richiesto per l'Esame)

Il servizio è implementato sia dai Discovery Server, sia dai generici server OPC UA; nel primo caso viene restituita la lista dei server registrati, nel secondo caso invece il server restituisce se stesso. Questa caratteristica assicura il corretto funzionamento anche nel caso di un unico nodo server in una rete dove non è presente alcun Discovery Server.

La lista dei server ritornati dalla funzione FindServers contiene, per ogni server, l'elenco degli URL che il client può utilizzare per invocare il servizio GetEndpoint.

FindServer può essere chiamato senza che vengano attivati meccanismi di sicurezza e senza una sessione attiva, perché la creazione di quest'ultima è subordinata alla conoscenza delle informazioni restituite da questo servizio.

namespace: Opc.Ua.DiscoveryServerBase

```
ResponseHeader FindServers(
            RequestHeader                   requestHeader,
            string                          endpointUrl,
            StringCollection                localeIds,
            StringCollection                serverUris,
            out ResponseHeader              responseHeader)
            out ApplicationDescription      servers)
```

Request	
Parametri	**Descrizione**
requestHeader	
endpointUrl	Network address (URL) utilizzato dal client per accedere al Discovery Endpoint che supporta questo servizio
localeIds[]	Lista di configurazioni linguistiche (locale) richieste dal client. Il Server dovrebbe ritornare il ServerName usando uno delle configurazioni specificate. Se nessuna richiesta viene supportata, allora il Server sceglierà una propria configurazione linguistica.
serverUris[]	Lista di server usati per restringere la ricerca; se vuota, restituisce tutti i server registrati. Viene specificato l'URI per ogni server.
Response	
responseHeader	
servers[]	Lista delle descrizioni dei server recuperati, che soddisfano le richieste del client. Ogni elemento della lista è di tipo ApplicationDescription ed è composto dai seguenti campi.
ApplicationUri	Identificatore globale univoco per l'istanza del server .
ApplicationName	Nome del server.

ProductUri	Identificatore globale univoco per i risultati del server.
ApplicationType	Tipo di applicazione, che può essere: server, client, client e server o discovery server.
DiscoveryUrls[]	Lista degli URLs disponibili che consentono la chiamata a *GetEndpoints* senza richiedere una connessione sicura.

4.2 GetEndpoints (Richiesto per l'Esame)

Il servizio restituisce gli endpoint disponibili nel server. Viene invocato specificato l'URL del Discovery Endpoint del Server, che supporta il servizio GetEndpoints. Le informazioni restituite sono necessarie per stabilire un SecureChannel tra client e server.

Non è richiesta né una connessione sicura né una sessione per la sua esecuzione. Per tale motivo è vulnerabile ad attacchi di tipo Denial Of Service (DOS).

Ciascun endpoint contiene delle specifiche configurazioni, rappresentate dalla seguente struttura dati:

Parametri	Descrizione
endpointUrl	Network address (URL) dell'endpoint
Server	La descrizione del Server a cui l'endpoint appartiene. Tra gli attributi utilizzati per la descrizione vi sono: ServerUri (global unique identifier dell'instanza del Server) e i discoveryUrls (gli URLs dei Discovery Endpoint)
serverCertificate	Certificato dell'istanza del server utilizzato dall'endpoint; è la chiave pubblica del server utilizzata dal client per rendere sicuro lo scambio di messaggi col server.
securityMode	Tipologia di meccanismo di sicurezza utilizzato per proteggere lo scambio di messaggi tra client e server, che possono essere firmati in modo da rilevare variazioni del contenuto ed evitare violazioni della privacy. I possibili valori sono: NONE_1 No security is applied. SIGN_2 All messages are signed but not encrypted. SIGNANDENCRYPT_3 All messages are signed and encrypted.
securityPolicyUri	La Politica di Sicurezza disponibile; viene specificato un Uri in cui è definito il set di algoritmi utilizzati dai meccanismi di sicurezza e la lunghezza della chiave usata per il Secure Channel.
userIdentityTokens[]	Lista di Identity Tokens utilizzati dal server per autenticare un utente durante la creazione di una sessione. Possibili tokens possono essere una combinazione di username e password, un certificato o un'autenticazione anonima.
transportProfileUri	Profilo di protocollo di trasporto supportato dall'endpoint, che specifica il formato di codifica dei messaggi e la versione di protocollo. Viene specificato in termini di Uri, in cui è definito tale profilo.
securityLevel	Livello di sicurezza; più alto è più sicuro è l'endpoint. Viene utilizzato dal client per confrontare tra loro gli endpoint offerti da un server in termini di sicurezza.

Come si evince dalla tabella precedente, la Security Policy viene identificata in OPC UA da un certo URI. Una Security Policy contiene tutti gli algoritmi di sicurezza che verranno sfruttati per cifrature, firme digitali, etc. Ad esempio, l'URI: http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15 identifica una policy che comprende l'algoritmo AES con chiave di 128 bit per la cifratura e l'RSA 1.5 per tutte le operazioni di crittografia asimmetrica.

Quando un client chiede l'elenco degli endpoints disponibili in un server, il *Client* può ridurre l'elenco degli endpoint disponibili nel server specificando dei criteri di filtraggio basati su: *LocaleIds* e *Transport Profile* URIs. Il primo specifica il settaggio linguistico desiderato dal client, mentre il secondo indica le caratteristiche dei protocolli di trasporto desiderati dal client. In tal caso, il *Server* ritorna un elenco vuoto se non riscontra endpoints che soddisfano i requirements del client.

namespace: Opc.Ua.DiscoveryServerBase

```
ResponseHeader GetEndpoints(
            RequestHeader                   requestHeader,
            string                          endpointUrl,
            StringCollection                localeIds[],
            StringCollection                profileUris[],
            out ResponseHeader              responseHeader)
            out EndpointDescription         Endpoints[])
```

Request	
Parametri	**Descrizione**
requestHeader	
endpointUrl	Indirizzo di rete (URL) del Discovery Endpoint del Server che supporta il servizio GetEndpoint
localeIds[]	Lista dei settaggi linguistici da utilizzare quando il server deve restituire stringhe contenenti valori leggibili dall'uomo
profileUris[]	Lista dei Transport Profile richiesti, ossia delle richieste di specifici protocolli di trasporto, usati per restringere la ricerca sugli endpoint disponibili. Se tale lista è vuota allora vengono ritornati tutti gli endpoint disponibili.
Response	
Parametri	**Descrizione**
responseHeader	
Endpoints[]	Lista di endpoints che soddisfano i criteri di ricerca imposti dal client; la lista è vuota se non vi sono endpoint che soddisfano le richieste. Ogni endpoint ritornato ha la struttura specificata in precedenza.

4.3 RegisterServer

Questo servizio permette la registrazione di un server presso un Discovery Server.

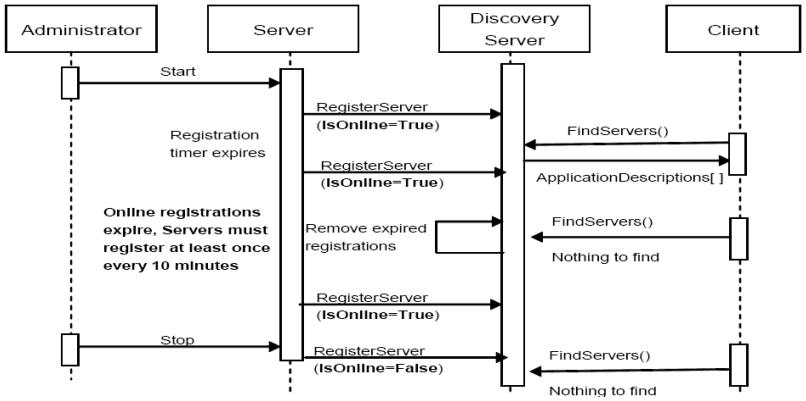
Il servizio può essere chiamato da un Server o da una utility di configurazione separata; le applicazioni client non possono mai chiamarlo.

Un Server deve necessariamente stabilire un SecureChannel con il Discovery Server prima di chiamare questo Servizio; ciò per essere certi che solo server trusted possano essere registrati nei Discovery Servers. A tal fine, l'amministratore del Server (Server Administrator) dovrà fornire al Server un Endpoint Description per il server Discovery come parte del processo di configurazione; in altri termini l'Endpoint Description del Discovery Server deve essere già noto. I Server devono essere in grado di registrarsi con un Discovery Server in esecuzione sulla stessa macchina.

Il processo di registrazione da parte del Server consiste nel fatto che esso passa al Discovery Server essenzialmente il proprio ServerUri (global unique identifier dell'instanza del Server) e i discoveryUrls (gli URLs dei Discovery Endpoint), che verranno successivamente chiesti dai client tramite il servizio GetEndpoints. Un'altra informazione che il server deve fornire in fase di registrazione è il nome di se stesso (servernames[], vettore di stringhe) offerto in modo "localizzato" (localised descriptive names), ossia in tutte le lingue supportate.

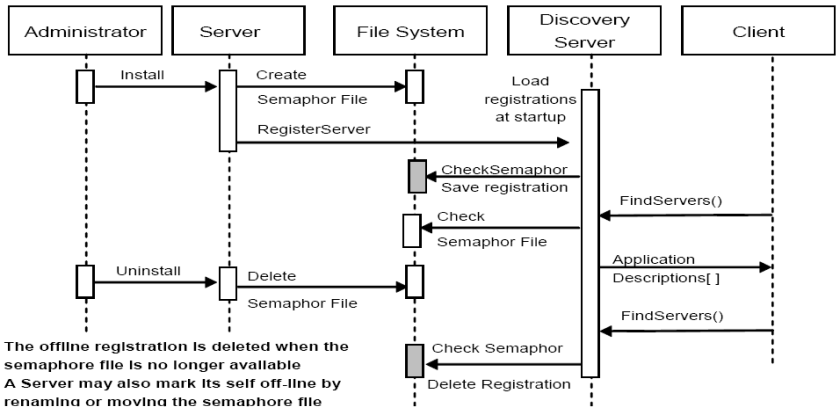
La registrazione può essere realizzata in due diverse modalità in base alla tipologia di applicazioni Server: quelle che vengono lanciati manualmente (ad esempio da un amministratore di sistema, compreso il caso in cui l'avvio avviene alla partenza del sistema operativo) e quelli che vengono avviati automaticamente quando un client tenta di connettersi.

La seguente figura si riferisce alla prima categoria di Server.



I server lanciati manualmente devono registrarsi periodicamente con il Discovery Server. La frequenza di registrazione dovrebbe essere configurabile entro un tempo massimo di 10 minuti.

La seguente figura si riferisce al processo di registrazione nel caso di applicazioni server eseguite automaticamente alla richiesta da parte di un client.



Quando un server viene lanciato automaticamente, il server viene registrato nel Discovery Server e viene fornito un percorso di un Semaphore File, creato nel file system locale al Server, che il Discovery Server può utilizzare per determinare se il server è stato disinstallato dalla macchina. Il Discovery Server deve avere accesso in lettura al file system che contiene il Semaphore File. Il parametro semaphoreFilePath viene passato al Discovery Server per indicare il percorso del semaphore File

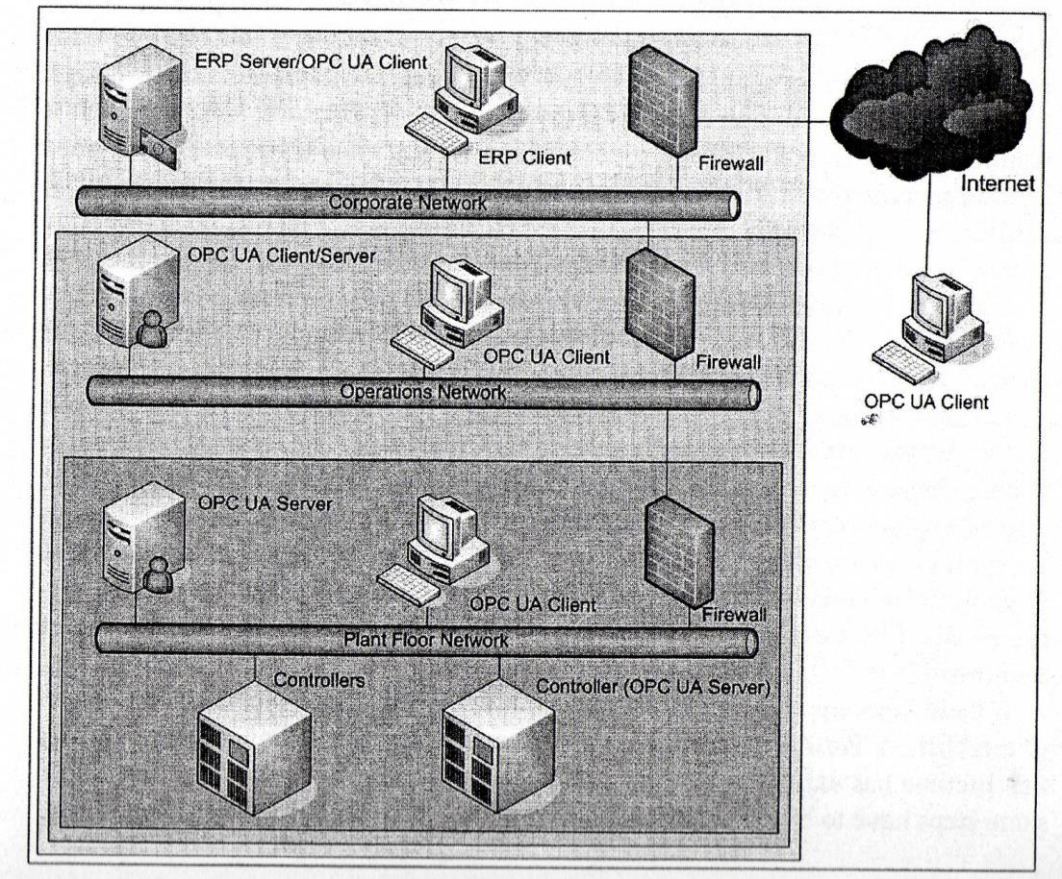
namespace: Opc.Ua.DiscoveryServerBase

```
ResponseHeader RegisterServer(
            RequestHeader    requestHeader,
            RegisteredServer server)
```

Parametri	Descrizione
Server	Contiene informazioni dettagliate sul server che si vuole registrare, tra cui i parametri serverUri, discoveryUrls[], serverNames[], semaphoreFilePath descritti in precedenza.

5 Sicurezza (Richiesto per l'Esame)

Le applicazioni OPC-UA possono essere eseguite in vari tipi di ambienti con diversi requisiti di sicurezza. La figura sottostante mostra un esempio di come esse possono essere organizzate:

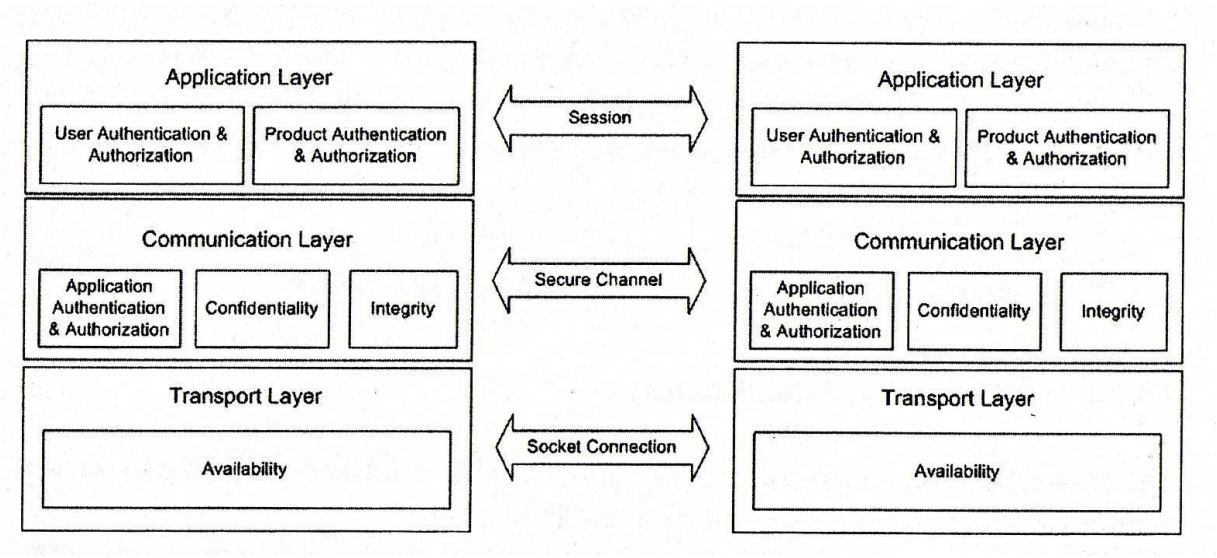


In questo esempio, OPC-UA è applicato a differenti livelli della automation pyramid: al plant floor level un server OPC-UA può essere in esecuzione all'interno dei controllori che forniscono ai client i dati estratti dai dispositivi di campo. Nell' operation level un'applicazione OPC-UA può fungere da client e da server allo stesso tempo: da client quando recupera dati dai server in esecuzione a livello più basso e genera allarmi, crea la history, etc; da server, quando fornisce ad altri OPC-UA client le elaborazioni effettuate. Al livello più alto un client OPC-UA integrato in un sistema ERP è in grado di ottenere informazioni sulle ore di lavoro dei dispositivi utilizzati nel plant floor e, se necessario, creare automaticamente richieste di manutenzione; inoltre in questi livello possono essere presenti server OPC-UA che possono essere acceduti da remoto via internet per eseguire particolari servizi o task di manutenzione.

Lo scenario descritto fa comprendere come OPC-UA può essere utilizzato a vari livelli con differenti obiettivi all'interno dello stesso sistema e quindi i requisiti di sicurezza possono variare in molti modi. Ciò influisce ad esempio sul compromesso tra sicurezza e performance: ai livelli più alti la sicurezza ha maggiore importanza della performance per via della connessione ad internet; ai livelli più bassi è prioritaria la performance perché i dati devono essere acquisiti in maniera veloce ed efficiente ai fini del controllo dei processi produttivi. Per questi motivi OPC-UA deve fornire un livello di sicurezza flessibile che permetta alle applicazioni di soddisfare i requisiti dei vari livelli.

5.1 Architettura (Richiesto per l'Esame)

L'architettura di sicurezza in OPC-UA, divisa in livelli, è descritta nella figura sottostante:



L'Application Layer viene usato per la trasmissione di informazioni sugli impianti, settaggi, istruzioni e dati real time provenienti dai dispositivi, tra client e server che hanno stabilito una Session. Una OPC-UA Session viene stabilita su un Secure Channel (che si trova al Communication Layer), che rende sicuro lo scambio dei dati di una sessione in diversi modi: assicura integrità mediane firme digitali, confidenzialità con la cifratura delle informazioni sensibili, autenticazione e autorizzazione delle applicazioni mediante l'utilizzo di speciali certificati X.509. Il Transport Layer è il livello responsabile della trasmissione e ricezione dei dati, attraverso una connessione socket, cui vengono applicati meccanismi di gestione degli errori per assicurare la protezione del sistema da attacchi come il Denial of Service.

Le tecnologie usate nei diversi layer dipendono dall'ambiente reale in cui le applicazioni sono dislocate.

5.2 Secure Channel (Richiesto per l'Esame)

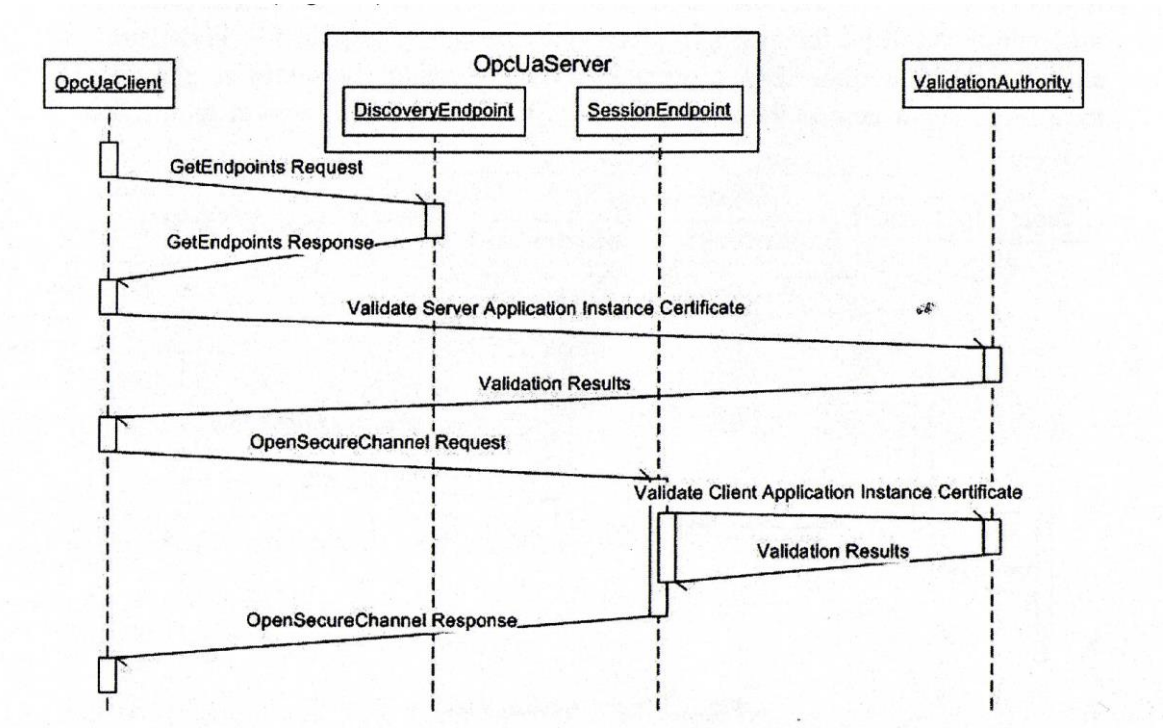
Per stabilire un SecureChannel tra un OPC-UA client e un OPC-UA server si devono seguire i seguenti passi.

Il primo passo è stato visto nella sezione precedente e consiste nel settaggio delle opzioni di configurazione della connessione; se l'applicazione è preconfigurata e già conosce come connettersi al server questa operazione può essere tralasciata, in caso contrario il client deve inviare una GetEndpoints request al Discovery Endpoint del server a cui vuole collegarsi per ricevere le descrizioni degli Endpoints esistenti, inclusa la configurazione di sicurezza, che contiene per esempio le Security Policies, i Security Modes, le User Token Policies e gli Application Instance Certificate del server.

Ricevute le informazioni desiderate, il client seleziona un Endpoint con una configurazione di sicurezza supportata e valida l'Application Instance Certificate del server. Questo avviene controllandone lo stato di validità presso la Validation Autority associata (VA).

Se il certificato è attendibile si procede col secondo passo e viene quindi inviata una richiesta di OpenSecureChannel in accordo alla Security Police e al Security Mode del Session Endpoint selezionato. Il Security Mode può avere tre stati: "None","Sign" e "SignAndEncrypt"; con il primo i messaggi inviati nel canale non saranno sicuri, col secondo saranno firmati con la chiave privata associata al certificato del client

OPC-UA, col terzo oltre la firma il messaggio è cifrato con la chiave pubblica presente nel certificato del server. La Security Police definisce invece qual algoritmi utilizzare per la firma e la cifratura dei messaggi. Il server, ricevuto un messaggio, valida il certificato del client con un'apposita richiesta alla relativa VA; se attendibile, il messaggio viene decifrato con la chiave privata del server e la firma verificata con la chiave pubblica nel certificato del client. Il server invia quindi al client una response cui sono applicati gli stessi meccanismi di sicurezza utilizzati per il messaggio proveniente dal client. La creazione del Secure Channel viene usata principalmente per lo scambio di informazioni segrete tra client e server, che vengono poi utilizzate per derivare chiavi simmetriche con cui verranno cifrati e firmati tutti i successivi messaggi; questo per evitare l'uso della crittografia a chiave pubblica, meno efficiente in termini di velocità computazionale. Derivate tali chiavi, il Secure Channel è stabilito e possiede un certo lifetime finito, terminato il quale dovranno essere ripercorsi i primi 2 step per derivare delle nuove chiavi simmetriche.

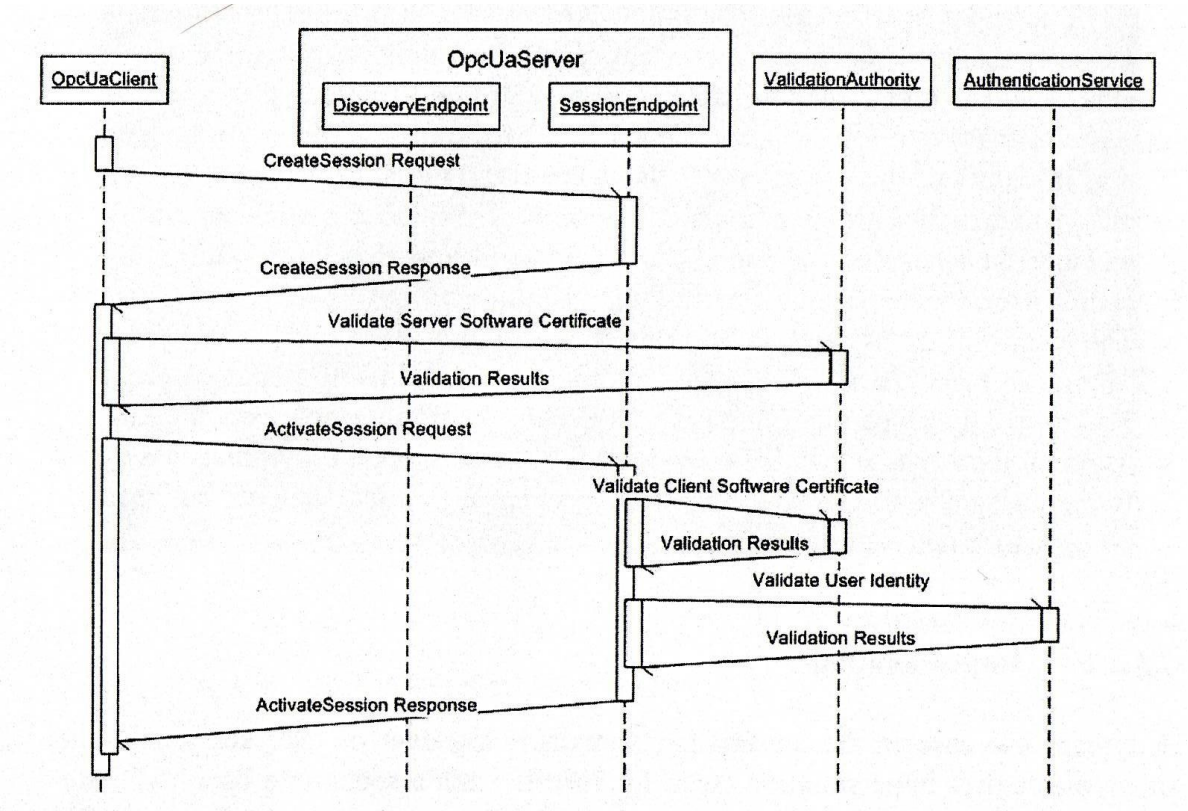


Creazione di un Secure Channel

5.3 Session (Richiesto per l'Esame)

Una volta creato il Secure Channel, il client può istaurare una Session con il Server; viene quindi inviata una richiesta CreateSession al server, che risponde fornendo i propri Software Certificates per comunicare le proprie funzionalità e per dimostrare il possesso del certificato utilizzato nella creazione del sottostante Secure Channel (tramite challenge-response).

La creazione della sessione deve essere seguita dalla sua attivazione; ciò viene realizzato inviando al server una richiesta di tipo ActivateSession che include le credenziali dell'user corrente e i Software Certificates del client. Le credenziali possono essere rappresentate da un certificato X.509 (validato da un VA) oppure da una coppia username-password. Una volta validate le user credentials e i Software Certificates da parte del server, la Session è instaurata e attiva, ed il client può quindi iniziare ad accedere ai dati di processo del server.



Creazione di una Session

5.4 Autenticazione ed autorizzazione (Richiesto per l'Esame)

Benché i due termini siano spesso considerati sinonimi, indicano in realtà due concetti diversi.

L'autenticazione è il processo di verifica dell'identità di un certo soggetto (una persona, un computer, un certificato, etc.); ad esempio, se un utente vuole essere identificato come "A", il sistema lo autentica verificando che username e password forniti durante il login siano corretti. Oppure, nel caso, invece, di un certificato inviato ad "A" che attesta che la chiave pubblica al suo interno appartiene a "B", l'autenticazione può essere effettuata facendo in modo che "A" e "B" inizino un test challenge-response in cui "B" invia ad "A" dei dati firmati con la propria chiave privata e si attende che "A" possa re-inviarglieli decifrati attraverso la chiave pubblica presente all'interno del certificato di "B".

L'autorizzazione è invece il processo che verifica se una certa entità *già autenticata* ha la possibilità di eseguire una certa operazione.

6 SecureChannel Service Set (Richiesto per l'Esame)

Il SecureChannel Service Set è l'insieme dei servizi disponibili in OPC UA che servono per gestire un SecureChannel, assicurando confidenzialità ed integrità dei messaggi scambiati con il Server.

Essenzialmente esistono due soli servizi: OpenSecureChannel e CloseSecureChannel.

7 Session Service Set (Richiesto per l'Esame)

Il Session Service Set è l'insieme dei servizi disponibili in OPC UA che servono per stabilire una connessione a livello application layer tra client e server, denominata Session; nell'ambito di ciascuna Session, il client potrà realizzare il classico flusso dati di lettura/scrittura e (se vuole) potrà creare una o più Subscription (Sottoscrizioni) nell'ambito delle quali innescare un meccanismo di aggiornamento automatico dei dati descritto nel seguito di questo documento.

I servizi appartenenti al Session Service Set sono:

- due servizi per istanziare la sessione tra due applicazioni (*CreateSession* e *ActivateSession*)
- un servizio per chiudere la sessione (*CloseSession*)
- un servizio per annullare le chiamate effettuate (*Cancel*)

La creazione di una sessione da parte del client prevede uno scambio di informazioni realizzato tramite l'invocazione del servizio CreateSession; tra le informazioni restituite dal Server vi è un sessionId (che identifica in modo univoco la sessione appena creata) e un authenticationToken (che dovrà essere usato dal client per tutte le future richieste avanzate al Server).

La creazione di una Session potrà essere realizzata solo dopo aver creato un SecureChannel e dunque assicurato l'integrità dei messaggi scambiati. L'autenticationToken restituito dal Server dopo la creazione della sessione è associato all'identificativo del SecureChannel sul quale la Session è stata creata.

Una Session non è utilizzabile se prima non viene effettuata una chiamata al servizio ActivateSession e termina ovviamente su richiesta del Client tramite il servizio di CloseSession. La Sessione può anche terminare quando il timeout (stabilito nello scambio di informazioni tra client e server al momento della creazione della sessione) scade senza che il server abbia più ricevuto chiamate dal client. Ciò assicura che il server liberi le risorse utilizzate solo se il client è effettivamente non più disponibile. Questo è un grande vantaggio rispetto ad OPC COM, in cui la gestione degli errori era basata su timeout fissati dallo standard.

Un flag apposito nel servizio CloseSession indica se devono essere cancellate o meno tutte le Subscription associate alla sessione; nel caso in cui il client non desideri eliminarle, esse potranno essere trasferite ad un'altra sessione attiva.

7.1 CreateSession (Richiesto per l'esame)

È usato da un client OPC UA per creare una Session e ricevere dal server un identificatore univoco che verrà utilizzato nelle successive chiamate effettuate nella sessione corrente.

namespace: Opc.Ua.SessionClient

```
ResponseHeader CreateSession(
            RequestHeader                   requestHeader,
            ApplicationDescription          clientDescription,
            String                          serverUri,
            String                          endpointUrl,
            String                          sessionName,
            ByteString                      clientNonce,
```

```
ApplicationInstanceCertificate          clientCertificate,
Duration                                RequestedSessionTimeout,
uint32                                  maxResponseMessageSize,
out ResponseHeader                      responseHeader,
out NodeId                              sessionId,
out SessionAuthenticationToken          authenticationToken,
out Duration                            revisedSessionTimeout,
out ByteString                          serverNonce,
out ApplicationInstanceCertificate      serverCertificate,
out EndpointDescription                 serverEndpoints[],
out SignedSoftwareCertificate           serverSoftwareCertificates[],
out SignatureData                       serverSignature,
out uint32                              maxRequestMessageSize)
```

Parametri	Descrizione
Request	
requestHeader	
ClientDescription	Informazioni che descrivono il Client. Il tipo di definizione è ApplicationDescription. Tra i campi che definiscono il client vi è: applicationUri (global unique identifier dell'istanza del client) e applicationName (il nome localizzato dell'applicazione client).
ServerUri	
endpointUrl	L'indirizzo di rete (URL) che il client deve utilizzare per accedere al Session Endpoint da cui può invocare il servizio CreateSession
sessionName	Una stringa human readable che specifica la Sessione. Il client dovrà fornire un nome che deve essere unico per l'istanza stessa del client. Non è obbligatorio e se manca è il server ad assegnare un nome alla sessione.
ClientNonce	Numero casuale fornito dal client e che non dovrebbe essere utilizzato in nessun'altra richiesta.
clientCertificate	Certificato fornito dal client, solo nel caso in cui la securityPolicy sia differente da NONE.
RequestedSessionTimeout	Numero massimo di millisecondi richiesto dal client, che rappresenta il massimo intervallo di tempo per una Session può rimanere aperta anche se non si registra alcuna attività.
maxResponseMessageSize	La massima dimensione, in bytes, del corpo di un messaggio di risposta, richiesta dal client.
Response	
responseHeader	
sessionId	E' un valore univoco di tipo NodeId assegnato dal Server alla sessione appena aperta.
authenticationToken	E' un identificativo univoco assegnato dal Server alla Sessione. Tale identificativo dovrà essere passato dal client in ogni successiva richiesta nel campo RequestedHeader.
revisedSessionTimeout	Valore massimo di millisecondi assegnato dal Server, che rappresenta il massimo intervallo di tempo per una Session può rimanere aperta anche se non si registra alcuna attività.
serverNonce	Numero casuale fornito dal server
serverCertificate	Certificato del Server
serverEndpoints[]	Lista di enpoints supportati dal server.
maxRequestMessageSize	La massima lunghezza (in byte) del corpo di un messaggio di richiesta fissato dal server.

7.2 ActivateSession

E' usato solo dopo che la Session è stata creata; si usa anche per cambiare l'utente della Session, modificare i settaggi per la lingua ed assegnare un nuovo canale sicuro alla Session.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader ActivateSession(
            RequestHeader                           requestHeader,
            SignatureData                           clientSignature,
            SignedSoftwareCertificate               clientSoftwareCertificates[],
            String                                  localeIds[],
            Extensible Parameter UserIdentityToken  userIdentityToken,
            SignatureData                           userTokenSignature,
            out ResponseHeader                      responseHeader,
            out ByteString                          serverNonce,
            out StatusCode                          results[],
            out DiagnosticInfo                      diagnosticInfos[])
```

Parametri	Descrizione
Request	
requestHeader	
clientSignature	Signature generata con la chiave privata associata con il clientCertificate
clientSoftwareCertificates[]	Lista dei certificati dell'applicazione client che identificano il prodotto, i profili supportati e i livelli di conformità per ogni profilo.
localeIds[]	Lista di "localizzazioni" che devono essere usate dal server per le successive comunicazioni. Il server utilizza la prima supportata della lista.
userIdentityToken	Token di identità utente per validare e far loggare uno specifico user alla Session; esistono diversi tipi di token, come username e password.
userTokenSignature	
Response	
responseHeader	
serverNonce	Numero casuale che non dovrebbe essere usato in altre richieste.
results[]	Risultati della richiesta di validazione del client per i suoi certificati.
diagnosticInfos[]	Lista di informazioni diagnostiche associate agli eventuali errori di validazione

7.3 CloseSession

Se il client non ha più bisogno della connessione verso il server deve usare il servizio CloseSession per iniziare la disconnessione e liberare le risorse di sessione nel server. Successivamente la disconnessione deve terminare con la chiusura del SecureChannel; queste due fasi sono unificate in un unico metodo fornito ai client OPC UA.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader CloseSession(
            RequestHeader    requestHeader,
            bool             deleteSubscriptions)
```

Parametri	Descrizione
deleteSubscription	Un flag che indica al server se deve cancellare tutte le Subscriptions associate alla sessione; le sottoscrizioni possono esistere indipendentemente dalle sessioni e possono essere trasferite da una sessione all'altra.

7.4 Cancel

Permette al client di cancellare le richieste in sospeso; ciò può essere molto utile nel caso di servizi che potenzialmente possono impiegare molto tempo per la loro esecuzione (ad esempio il servizio Query).

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader Cancel(
            RequestHeader requestHeader,
            uint          requestHandle,
            out uint      cancelCount)
```

Parametri	Descrizione
requestHandle	Handle assegnato dal client ad una o più richieste precedenti che si vogliono annullare.
cancelCount	Numero di richieste cancellate.

8 View Service Set (Richiesto per l'Esame)

Uno dei principali obiettivi di OPC UA è quello di combinare le modalità di organizzazione dati delle precedenti versioni in un unico Address Space, garantendone comunque l'accessibilità.

In particolare, OPC UA permette di descrivere ed inviare informazioni. I due principali servizi che implementano tali funzionalità sono *Browse*, per navigare attraverso i nodi dell'Address Space, e *Read* per accedere ai metadati dei nodi. Browse fa parte dei ViewService Set e verrà descritto nel seguito.

8.1 Browse (Richiesto per l'Esame)

È usato dai client per visitare l'Address Space, dati uno o più nodi d'inizio e dei *browse filters*. Il browsing si basa totalmente sui References dei Nodes. Dato un Node di partenza, il servizio di browsing permette di scoprire tutti i References di quel Node. Dunque, a seguito dell'invocazione del servizio di browse, il server ritorna, per ogni nodo iniziale, la lista dei nodi ad esso connessi.

Il Servizio di Browse si può applicare anche alle View e in tal caso i risultati stessi vengono limitati al sottoinsieme di References presenti in una View.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader Browse(
            RequestHeader                requestHeader,
            ViewDescription              view,
            Counter                      requestedMaxReferencesPerNode,
            BrowseDescription            nodesToBrowse[],
            out ResponseHeader           responseHeader,
            out BrowseResult             results[],
            out DiagnosticInfo           diagnosticInfos[])
```

Parametri	Descrizione
Request	
requestHeader	
view	Serve a limitare la visita alla View specificata; per visitare l'intero Address Space tale parametro non viene settato .
requestedMaxReferencesPerNode	Limita il numero di nodi restituiti per evitare un infinito numero di risultati. Per superare questo limite si usa il servizio BrowseNext.
nodesToBrowse[]	Lista dei nodi d'inizio e filtri di browsing. Ogni elemento della lista è composto dai seguenti campi.
nodeId	Id del nodo di partenza.
browseDirection	Indica se il server deve seguire i collegamenti in avanti, all'indietro o in entrambe le direzioni.
referenceTypeId	Id del ReferenceType desiderato. Questo parametro è tipicamente combinato con IncludeSubtypes per filtrare un intero set di ReferenceType.

includeSubtypes	Indica se devono essere restituiti dal server anche i sottotipi del ReferenceType specificato. I client dovrebbero sempre settare tale valore a true.
nodeClassMask	Filtra le NodeClass dei nodi restituiti, per esempio soltanto Objects e Variables.
resultMask	Filtra i risultati restituiti; l'unica informazione sempre restituita è il NodeId dei nodi visitati, tutti gli altri valori possono essere esclusi con questa maschera. Ciò consente al client di far elaborare al server richieste meno onerose, facendogli scartare informazioni a cui il client non è interessato, e contribuisce a ridurre il traffico.

Response	
responseHeader	
results[]	Lista dei risultati in base a nodi e filtri forniti dal client. Ogni elemento della lista è composto dai seguenti campi.

statusCode	Codice relativo a nodi e filtri forniti dal client; è significativo solo in caso di filtri non validi o starting node non conosciuti. Se results è una lista vuota non viene restituito un StatusCode di errore.
continuationPoint	Viene restituito quando il server non è in grado di ritornare tutti i risultati all'interno della response, a seguito di una limitazione che può essere stata settata dal client nell'ambito della richiesta o dal server durante l'elaborazione della Browse. Viene passato dal client al servizio *BrowseNext* per ottenete i rimanenti risultati.
References[]	Lista di reference riguardanti i nodi che hanno superato i criteri di filtraggio indicati nella richiesta. Ogli elemento della lista a sua volta è fatto dai seguenti campi.

referenceTypeId	Id del ReferenceType tra lo starting node ed ogni nodo restituito.
IsForward	Indica se la modalità di attraversamento dal nodo d'inizio al nodo restituito è in avanti.
nodeId	Id del nodo (che ha superato i criteri di filtraggio).
browseName	Il nome del nodo, utilizzato per il browsing.
displayName	Il nome del nodo che verrà mostrato in accordo alla localizzazione specificata in fase di attivazione della sessione .
nodeClass	NodeClass del nodo.
typeDefinition	Id del tipo oggetto o variabile del nodo; questo parametro è settato solo se il nodo rappresenta una variabile o un oggetto.

diagnosticInfos[]	Lista di informazioni diagnostiche in relazione ai risultati (results) ottenuti nella Response. La dimensione e l'ordine degli elementi della lista coincide con la dimensione e l'ordine degli elementi della lista delle results. Ad ogni elemento della lista dei results, corrisponde un elemento della lista delle informazioni diagnostiche. Ogni informazione diagnostica è composta dai

seguenti campi.	
namespaceUri	
symbolicId	
locale	
localizedText	
additionalInfo	
innerDiagnosticInfo	

8.2 BrowseNext

BrowseNext è usato successivamente ad una chiamata al servizio Browse (o allo stesso BrowseNext) quando quest'ultima non è riuscita a restituire tutti i risultati in un unico messaggio di response.

Il numero di nodi da restituire può essere limitato dal client nella richiesta di Browse o dal server durante l'elaborazione della chiamata al servizio.

namespace: Opc.Ua.SessionClient

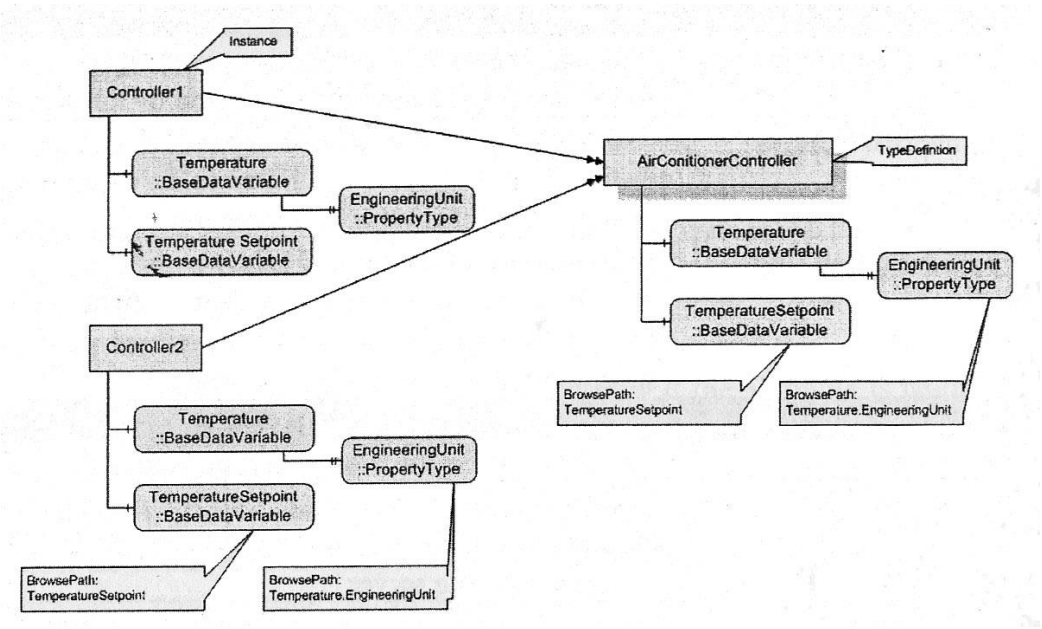
```
public virtual ResponseHeader BrowseNext(
            RequestHeader                requestHeader,
            bool                         releaseContinuationPoints,
            ByteStringCollection         continuationPoints,
            out BrowseResultCollection   results,
            out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
releaseContinuationPoints	Flag che indica se il servizio è chiamato soltanto per far rilasciare al server la memoria associata al continuation point o se effettivamente il server deve restituire il prossimo set di risultati. I client dovrebbero sempre chiamare questo servizio anche quando non vogliono continuare il browsing (in questo caso il parametro deve essere settato a true).
continuationPoints	Lista di continuation point restituita da una precedente chiamata a Browse o BrowseNext.
I parametri di out sono esattamente uguali a quelli del servizio Browse.	

8.3 TranslateBrowsePathsToNodeIds

In OPC UA è possibile accedere ai componenti di un object basandosi sulla conoscenza dell'object type. Ciò in quanto il BrowseName dei componenti dell'istanza è uguale al BrowseName dei componenti del relativo tipo (struttura e nomi dell'istanza coincidono con struttura e nomi del tipo – vedi figura).

Questo servizio consente di ricavare il NodeId di un componente (ad esempio una Variable o una Property) di un'istanza, fornendo il NodeId dell'istanza e il BrowsePath del componente desiderato.



namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader TranslateBrowsePathsToNodeIds(
            RequestHeader                requestHeader,
            BrowsePathCollection         browsePaths,
            out BrowsePathResultCollection results,
            out DiagnosticInfoCollection   diagnosticInfos)
```

Parametri	Descrizione
browsePaths	Lista dei browse path i cui NodeId sono richiesti.
StartingNode	NodeId del nodo da cui il server deve iniziare a seguire il path fornito.
relativePath	Browse path che il server deve seguire, composto da una lista di elementi.
ReferenceTypeId	Id del ReferenceType desiderato. Questo parametro è tipicamente combinato con IncludeSubTypes per filtrare un intero set di ReferenceType.
IncludeSubTypes	Indica se devono essere restituiti dal server anche i sottotipi del ReferenceType specificato.
IsInverse	Indica se il server deve seguire il ReferenceType in direzione inversa. Di default questo flag è false (direzione in avanti).
TargetName	BrowseName del nodo target. Può essere vuoto per l'ultimo elemento del browse path; in questo caso tutti i nodi collegati mediante il ReferenceType specificato sono restituiti.
results	Lista dei risultati in base a starting node e path forniti dal client.
StatusCode	Result code per starting node e path forniti. Se il path non corrisponde a nessun nodo target questo parametro è settato a "BadNoMatch" dal

	server.
Targets	Lista dei target Nodes per ogni combinazione starting Node – browse path.
TargetId	Id del target Node.
RemainingPathIndex	I server possono avere collegamenti a nodi situati in altri server. In questo caso il browse path completo non può essere processato da un solo server e TargetId indica lo starting Node da usare nell'altro server assieme al RemainingPath.

8.4 RegisterNodes

Questo servizio permette ai client di ottimizzare l'accesso ciclico ai nodi, per esempio per scrivere valori nelle variabili o per chiamare metodi; ci sono due livelli di ottimizzazione:

- Riduzione della quantità di dati scambiata per l'indirizzamento dell'informazione: poiché i NodeId possono essere molto lunghi OPC UA fornisce un metodo di indirizzamento ottimizzato che consiste nell'uso di NodeId numerici che possono essere usati in tutte le funzioni che accedono alle informazioni nel server. L'invio di tali codici numerici è molto efficiente nel protocollo di trasporto binario di OPC UA.
- Meccanismi di ottimizzazione interni al server che permettono al client un rapido accesso ai nodi più frequentemente utilizzati.

I client devono cancellare la registrazione ad un nodo se non è più necessaria, per liberare le risorse utilizzate nel server per l'ottimizzazione. Inoltre, la registrazione non deve essere usata per ottimizzare letture cicliche dei dati perché OPC UA fornisce un altro set di servizi interamente dedicati a tale scopo (Subscription Service Set).

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader RegisterNodes(
            RequestHeader        requestHeader,
            NodeIdCollection     nodesToRegister,
            out NodeIdCollection registeredNodeIds)
```

Parametri	Descrizione
nodesToRegister	Lista dei NodeId dei nodi a cui registrarsi.
registeredNodeIds	Lista dei NodeId dei nodi a cui ci si è registrati. Questi Id sono tipicamente codici numerici ottimizzati da utilizzare come handle ai nodi e sono validi soltanto nell'ambito della Session in cui sono stati creati. Se il server non conosce un NodeId in ingresso o se non è in grado di ottimizzare l'accesso a tale nodo, restituisce semplicemente lo stesso NodeId.

8.5 UnregisterNodes

Questo servizio è usato dai client per annullare le registrazioni effettuate precedentemente e liberare le risorse nel server.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader UnregisterNodes(
            RequestHeader    requestHeader,
            NodeIdCollection nodesToUnregister)
```

Parametri	Descrizione
nodesToUnregister	Lista dei NodeId dei nodi di cui annullare le registrazioni fatte precedentemente.

9 Read and Write Services (Richiesto per l'Esame)

Una delle più importanti caratteristiche di OPC è la lettura e scrittura dei dati mediante meccanismi di scambio dati standard.

I servizi di *Read* e *Write* non permettono la lettura e la scrittura degli attributi dei nodi (includendo dunque i valori delle Variables).

9.1 Read (Richiesto per l'Esame)

Questo servizio si utilizza per leggere uno o più Attributes di uno o più Nodes, e permette inoltre la lettura di sottoinsiemi o singoli valori di array.

Il servizio Read consente al client di definire un tempo massimo di validità dei valori da restituire; tale parametro viene detto maxAge. Esso obbliga il Server ad accedere alla sorgente dei dati (ad esempio un sensore), se la copia mantenuta dal Server è più vecchia rispetto al parametro maxAge specificato dal client.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader Read(
            RequestHeader                requestHeader,
            Duration                     maxAge,
            TimestampsToReturn           timestampsToReturn,
            ReadValueId                  nodesToRead[],
            out ResponseHeader           responseHeader,
            out DataValue                results[],
            out DiagnosticInfo           diagnosticInfos[])
```

Parametri	Descrizione
Request	
requestHeader	
maxAge	Tempo massimo di validità del valore da leggere, in millisecondi. Questo parametro permette al server di ridurre la quantità di dati scambiati con la sorgente del dato, consentendogli di restituire un valore in cache che abbia "età" minore di maxAge. Se impostato a zero, obbliga il server a restituire sempre il valore attuale.
timestampsToReturn	In OPC UA sono definiti due timestamps, quello della sorgente e quello del server. Questo parametro permette al client di stabilire quale dei due il server deve restituire insieme al valore. Possono essere ritornati entrambi o nessuno dei due. I valori che può assumere l'attributo sono: SOURCE_0, SERVER_1, BOTH_2 e NEITHER_3.
nodesToRead	Lista di nodi e relativi attributi da leggere.
nodeId	Id del nodo da leggere.
attributeId	Id dell'attributo del nodo da leggere.
indexRange	Questo parametro identifica un singolo elemento o un range di elementi di un array.
dataEncoding	Necessario soltanto nel caso della lettura dell'Attribute Value di

		una Variable. In questo caso i valori possono essere inviati con differenti codifiche; in OPC UA quelle di default sono XML o UA binary format. Questo parametro permette al client di definire il tipo di codifica.
Response		
responseHeader		
Results[]	Lista dei valori degli Attributi richiesti	
	value	Contiene il valore letto.
	StatusCode	Codice che indica l'esito dell'operazione di lettura. Il valore letto è utilizzabile solo se StatusCode è "good".
	SourceTimestamp	Timestamp relativo alla sorgente da cui proviene il valore. È disponibile solo per attributi di tipo Value. Indica l'istante dell'ultima variazione avvenuta nella sorgente del dato.
	ServerTimestamp	Timestamp relativo al server da cui proviene il valore. Indica l'istante in cui il server ha ricevuto il valore dalla sorgente oppure quello dell'ultimo aggiornamento del server (se il Value non è cambiato).
diagnosticInfos[]	Lista di informazioni diagnostiche.	

9.2 Write

Con questo servizio è possibile scrivere uno o più Attributes di uno o più nodi, e permette inoltre la scrittura di sottoinsiemi o singoli valori di array. Come la maggior parte degli altri servizi, Write è pensato per operazioni generiche e non relative al tipo di dato considerato.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader Write(
            RequestHeader                requestHeader,
            WriteValueCollection         nodesToWrite,
            out StatusCodeCollection     results,
            out DiagnosticInfoCollection diagnosticInfos)
```

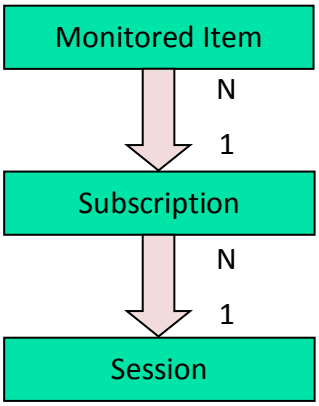
Parametri	Descrizione
nodesToWrite	Lista di Nodes, Attributes e valori da scrivere.
NodeId	Id del nodo da scrivere.
AttributeId	Id dell'attributo del nodo da scrivere, che può essere sia di tipo Value che di qualsiasi tipo scrivibile.
IndexRange	Questo parametro identifica un singolo elemento o un range di elementi di un array.
Value	Contiene il valore da scrivere.
results	Lista dei codici di stato per ogni operazione di scrittura.

10 Subscription Service Set (Richiesto per l'Esame)

Le sezioni precedenti hanno evidenziato come nell'ambito di una Session sia possibile invocare servizi di View e servizi di lettura/scrittura. Uno strumento ancora più potente che un client può utilizzare nell'abito di una Session è l'attivazione di un meccanismo che richiama l'aggiornamento automatico dei dati esistente in OPC COM/DCOM.

Essenzialmente per ciascuna Session, un client può creare una o più Subscription; nell'ambito di ciascuna Subscription il client può creare dei Monitored Items, agganciati ad item reali. Ciascun Monitored Item è in grado di creare le così dette Notification, legate a cambiamenti di valori di attributi o di variabili scelte dal client o legate all'avvenimento di particolari eventi. Ciascun Monitored Item produce continuamente Notifications fin quando la sottoscrizione non viene annullata, o i Monitored Items cancellati o, infine, tale produzione non viene disabilitata dal client.

La figura sottostante rappresenta le relazioni tra Sessioni, Sottoscrizioni e Monitored Items:



Come si vede per ogni sessione è possibile creare differenti sottoscrizioni, e nell'ambito di ciascuna di esse è possibile creare differenti Monitored Item.

In questa sezione verrà approfondito il meccanismo di funzionamento della Subscription. Essenzialmente un Subscription permette di realizzare un meccanismo automatico di creazione di particolari messaggi che verranno inviati al client; tali messaggi sono detti NotificationMessage. Essi contengono i valori prodotti dai Monitored Item creati per quella Subscription. Come detto, ciascun valore prodotto da un Monitored Item è chiamato Notification. Essenzialmente la Subscription ha il compito di collezionare e raggruppare tutti i Notification prodotti fino a quel momento nell'ambito della stessa Subscription, con cadenza periodica. Tutti i Notification così raggruppati vengono inseriti in un NotificationMessage per poter essere inviato al Client. Il meccanismo di invio dei NotificationMessage al Client verrà approfondito nelle sezioni successive.

I Monitored Item non possono esistere se non viene creata, dal client, una Subscription nell'ambito di una Session. Per tale motivo, è necessario che il client crei una Subscription prima di poter definire i MonitoredItem.

Una *Subscription* ha diversi parametri illustrati nel seguito; vi sono sicuramente due settings principali: il Publishing Interval, che definisce l'intervallo di tempo con il quale il server crea un NotificationMessage per

la determinata Subscription, ed il Publishing Enabled (un valore booleano), che di fatto abilita o meno l'inoltro del NotificationMessage al Client.

Ogni Subscription mantiene un keep-alive counter che conta il numero di Publishing Interval che sono trascorsi senza che la Subscription abbia avuto Notifications da inoltrare al Client. L'assenza di Notifications prodotte per quella Subscription può rappresentare un problema, perché il client non riceverà alcuna informazioni per quella Subscription e dunque potrebbe pensare che la Subscription non sia più attiva. Il keep-alive counter è importantissimo perché al suo scadere, il Server dovrà provvedere ad inviare al client un particolare messaggio al client (keep-alive Message) per informarlo che la Subscription è attiva. Ciò verrà spiegato nelle seguenti sezioni.

Inoltre ogni Subscription mantiene un lifetime counter che conta il numero consecutivo di Publishing Interval entro i quali non vi è stata attività da parte del client; come si vedrà nelle sezioni successive, l'attività viene monitorata attraverso il controllo dell'invio da parte del client di particolari messaggi di Publish Request. Il contatore viene resettato nel momento in cui viene monitorata attività da parte del client. Quando questo contatore raggiunge il massimo per quella determinata Subscription, allora essa viene chiusa. La chiusura comporta l'eliminazione di tutti i Monitored Items in essa contenuti.

Nel seguito vengono specificati i servizi disponibili in OPC UA per la gestione delle Subscription.

10.1 CreateSubscription (Richiesto per l'Esame)

Questo servizio si usa per creare una Subscription e definire i suoi settaggi iniziali. La sottoscrizione può essere cancellata con il servizio DeleteSubscription, o settando il flag DeleteSubscription all'atto della chiusura della Session.

Generalmente, i settaggi più importanti per una sottoscrizione sono publishingEnabled e requestedPublishingInterval, mentre requestedMaxKeepAliveCount, requestedLifetimeCount, maxNotificationsPerPublish e priority sono parametri addizionali:

- **publishingEnabled** se è TRUE abilita la pubblicazione di NotificationMessage da parte della Susbscription. Se è FALSE la pubblicazione è disabilitata.
- **requestedPublishingInterval** definisce l'intervallo di tempo che il client desidera impostare per il Publishing Interval nell'ambito della Subscription che sta creando. Tale intervallo temporale è espresso in ms e corrisponde all'intervallo di tempo dopo il quale la Subscription prepara un NotificationMessage contenente tutti i Notifications non ancora inoltrati al client, prodotti dai MonitoredItem della Subscription. Se il valore è 0, allora il Server dovrà settare il Publishing Interval al valore più basso (ciclo più veloce).
- **requestedMaxKeepAliveCount** definisce quante volte il publishing interval deve trascorrere senza che sia disponibile alcuna Notifications da inviare al Client, perché il server mandi un keep-alive messagge al client in grado di comunicargli che quella particolare Subscription è ancora attiva.
- **requestedLifetimeCount** indica quante volte il publishing interval può trascorrere senza che sia monitorata alcuna attività da parte del client. Passato questo lasso di tempo, il server cancella la Subscription e libera le risorse. Questo parametro dev'essere grande almeno 3 volte il keep alive count, ed entrambi i parametri vengono negoziati da client e server.
- **maxNotificationsPerPublish** serve a limitare il numero di Notifications inviate dal server al client all'interno di un NotificationMessage. È deciso dal client, ma il server può inviarne un numero massimo minore in base al proprio limite stabilito. Se non tutte le notifiche disponibili possono

essere inviate in un unico messaggio, ne viene inviato uno ulteriore. Un valore 0 indica che il client non desidera impostare alcun limite.

- **priority** definisce la priorità della sottoscrizione rispetto alle altre sottoscrizioni già create dal client. Tale priorità viene utilizzata nella trasmissione dei NotificationMessage, come verrà spiegato nelle sezioni successive.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader CreateSubscription(
            RequestHeader          requestHeader,
            double                 requestedPublishingInterval,
            uint                   requestedLifetimeCount,
            uint                   requestedMaxKeepAliveCount,
            uint                   maxNotificationsPerPublish,
            bool                   publishingEnabled,
            byte                   priority,
            out ResponseHeader     requestHeader,
            out uint               subscriptionId,
            out double             revisedPublishingInterval,
            out uint               revisedLifetimeCount,
            out uint               revisedMaxKeepAliveCount)
```

Request	
Parametri	**Descrizione**
requestHeader	Campo contenente diversi parametri che caratterizzano le richieste sottomesse su una Session; ad esempio timestamp, ossia il tempo al quale il client ha inviato la richiesta.
requestedPublishingInterval	Intervallo di tempo, proposto dal client, per la pubblicazione periodica dei NotificationMessage
publishingEnabled	Flag che indica se la pubblicazione dei NotificationMessage è abilitata o meno.
requestedMaxKeepAliveCount	*Vedi descrizione precedente*
requestedLifetimeCount	*Vedi descrizione precedente*
maxNotificationsPerPublish	*Vedi descrizione precedente*
Priority	*Vedi descrizione precedente*
Request	
Parametri	**Descrizione**
responseHeader	Campo contenente diversi parametri che caratterizzano le risposte inviate; ad esempio timestamp, ossia il tempo al quale il Server ha inviato la risposta.
subscriptionId	Id della sottoscrizione settato dal server. Il client deve usare questo Id in tutte le successive chiamate di servizio relative a questa Subscription.
revisedPublishingInterval	Intervallo di pubblicazione deciso dal server. Può non essere uguale a requestedPublishingInterval.
revisedMaxKeepAliveCount	MaxKeepAliveCount deciso dal server. Può non essere uguale a requestedMaxKeepAliveCount.
revisedLifetimeCount	LifetimeCount deciso dal server. Può non essere uguale a

requestedLifetimeCount.

10.2 DeleteSubscription

Questo servizio viene usato per cancellare una sottoscrizione precedentemente creata con CreateSubscription.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader DeleteSubscriptions(
            RequestHeader                requestHeader,
            UInt32Collection             subscriptionIds,
            out StatusCodeCollection     results,
            out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
subscriptionsIds	Lista delle sottoscrizioni da cancellare identificate dai rispettivi Id, che erano stati assegnati dal server durante la chiamata a CreateSubscription.
Results	Lista degli status code per gli Id passati in ingresso, che indicano se le cancellazioni sono avvenute con successo.

10.3 ModifySubscription

Si usa per modificare i settaggi di una Subscription.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader ModifySubscription(
      RequestHeader requestHeader,
      uint          subscriptionId,
      double        requestedPublishingInterval,
      uint          requestedLifetimeCount,
      uint          requestedMaxKeepAliveCount,
      uint          maxNotificationsPerPublish,
      byte          priority,
      out double    revisedPublishingInterval,
      out uint      revisedLifetimeCount,
      out uint      revisedMaxKeepAliveCount)
```

Parametri	Descrizione
subscriptionsId	Id della sottoscrizione da modificare, che dev'essere quello restituito dal server durante la chiamata a CreateSubscription.
Per le descrizioni del resto dei parametri vedi CreateSubscription.	

10.4 SetPublishingMode

Serve per modificare il parametro publishingEnabled di una lista di sottoscrizioni. Può essere utilizzato dai client per disattivare l'invio dei messaggi di notifica da parte del server, senza cancellare la gestione delle risorse monitorate.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader SetPublishingMode(
      RequestHeader                requestHeader,
      bool                         publishingEnabled,
      UInt32Collection             subscriptionIds,
      out StatusCodeCollection     results,
      out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
publishingEnabled	Flag che indica se la pubblicazione dei messaggi di notifica dev'essere abilitata o disabilitata per la lista di sottoscrizioni passate in ingresso.
subscriptionIds	Lista delle sottoscrizioni di cui modificare il valore di publishingEnabled.
Results	Lista degli status code per gli Id passati in ingresso, che indicano se le modifiche dei publishingEnabled sono avvenute con successo.

10.5 TransferSubscriptions

Il servizio è usato per trasferire una lista di Subscriptions alla Session usata per la chiamata. Questa procedura viene effettuata dai client ridondanti per spostare una o più sottoscrizioni dal client principale ad un client di backup, se il client principale non è più disponibile. In alternativa, viene usata per spostare una o più sottoscrizioni ancora attive da una Session già esistente ma non più valida ad una nuova Session.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader TransferSubscriptions(
      RequestHeader                requestHeader,
      UInt32Collection             subscriptionIds,
      bool                         sendInitialValues,
      out TransferResultCollection results,
      out DiagnosticInfoCollection diagnosticInfos)
```

Parametri		Descrizione
subscriptionIds		Lista degli Id relativi alle sottoscrizioni da trasferire.
Results		Esiti dei tentativi di trasferimento delle Subscriptions.
	StatusCode	Codice del risultato dell'operazione di trasferimento.
	AvailableSequenceNumbers	Lista di numeri di sequenza disponibili nella

			sottoscrizione per la ritrasmissione, di cui non è stato inviato l'ack da parte del client.

11 Monitored Item Service Set (Richiesto per l'esame)

La sezione precedente ha spiegato che una Subscription permette di realizzare un meccanismo automatico di creazione di particolari messaggi che verranno inviati al client; tali messaggi sono detti NotificationMessage. Essi contengono i valori prodotti dai Monitored Item creati per quella Subscription. Ciascun valore prodotto da un Monitored Item è chiamato Notification. Essenzialmente la Subscription ha il compito di collezionare e raggruppare tutti i Notification prodotti fino a quel momento nell'ambito della stessa Subscription, con cadenza periodica, entro il Publishing Interval. Tutti i Notification così raggruppati vengono inseriti in un NotificationMessage per poter essere inviato al Client.

I Monitored Item non possono esistere se non viene creata, dal client, una Subscription nell'ambito di una Session.

Un Monitored Item viene definito nell'ambito di una Subscription per monitorare un particolare item del Server che può essere:

- Attributi; un qualunque attributo di un qualunque Nodo può essere associato ad un Monitored Item. Gli attributi vengono monitorati unicamente per indicare quando il loro valore cambia. Il cambiamento del valore produce una Notification.

- Variabili; l'attributo Value dei nodi appartenenti al NodeClass Variabile (Variable) può essere associato ad un MonitoredItem. L'attributo Value di una Variable viene monitorato per indicare un cambiamento nel valore o un cambiamento nello stato (StatusCode) associato al value. Vengono specificati delle condizioni su tali cambiamenti: se tali condizioni sono verificate, allora il cambiamento del valore o dello stato associato al value di una Variabile causa la produzione di un Notification da parte del Monitored Item.

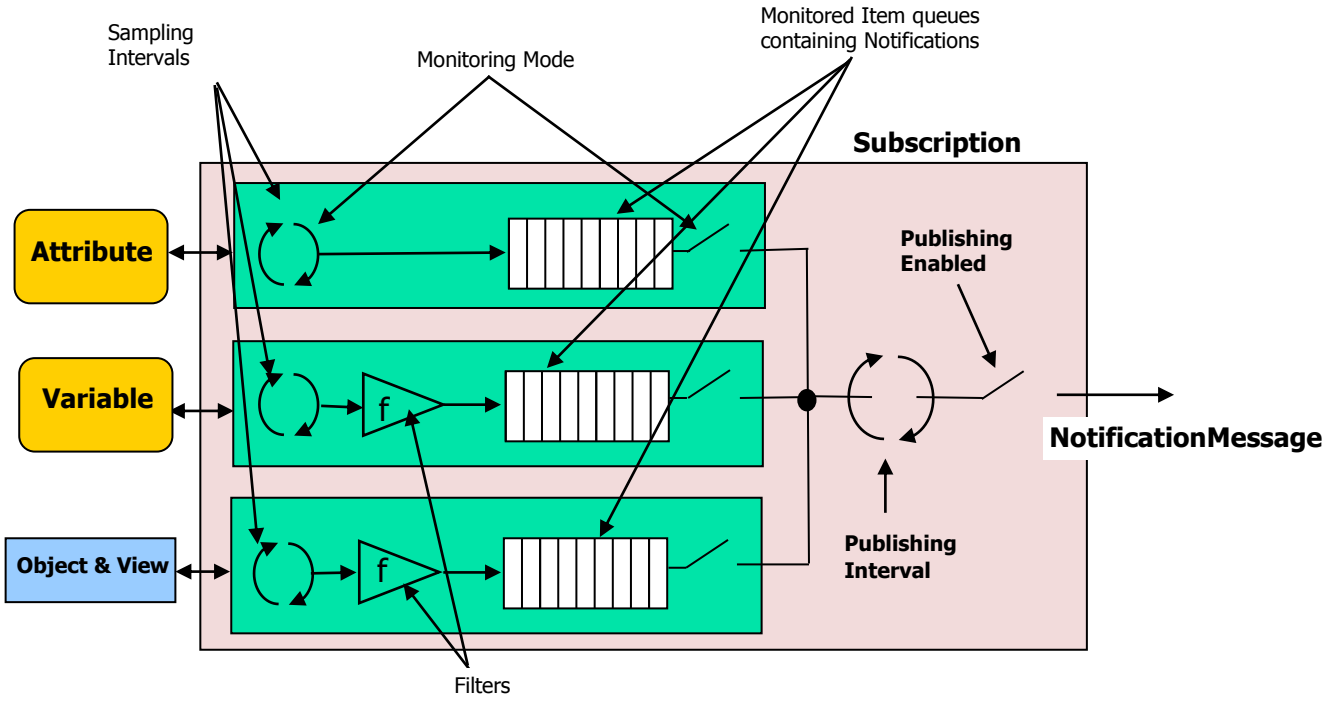
- Oggetti e View; i nodi appartenenti al NodeClass Object e View possono essere associati ad un Monitored Item, al fine di essere monitorati circa l'accadimento di un particolare Evento. A tal fine, però è necessario che il relativo Nodo abbia settato un particolare bit denominato SubscribeToEvents dell'attributo EventNotifier. Il verificarsi dell'evento produce la generazione di una Notification da parte del Monitored Item.

Come detto dunque, per ogni cambiamento di attributo, valore, stato o per ogni evento monitorato dal Monitored Item viene prodotta una Notification. Diverse Notifications vengono impacchettate dalla Subscription entro un NotificationMessage ad intervalli regolari scanditi dal Publishing Interval.

Tutte le Monitored Items hanno alcuni settaggi in comune; ve ne sono 4: sampling interval, monitoring mode, filter settings e queue parameters. La figura sottostante mostra i differenti settings per i Monitored Items; la figura evidenzia inoltre i due parametri della Subscription precedentemente citati, ossia il Publishing Interval e il Publishing Enable.

La figura evidenzia che i Monitored Item possono essere associati unicamente a: Attributi di Nodi (qualunque NodeClass), Attributo Value di Nodi appartenenti a NodeClass Variable, e a Nodi appartenenti a NodeClass Object e View (per i quali esiste l'attributo EventNotifier, sotto l'ipotesi che il Nodo abbia settato un particolare bit denominato SubscribeToEvents dell'attributo EventNotifier).

Si noti infine che per gli Attributi, non è presente l'oggetto triangolare con dentro la lettera f (filtro) come verrà illustrato nel seguito.



Nel seguito verranno illustrati tutti gli elementi presenti nella figura.

11.1 Sampling Interval (Richiesto per l'Esame)

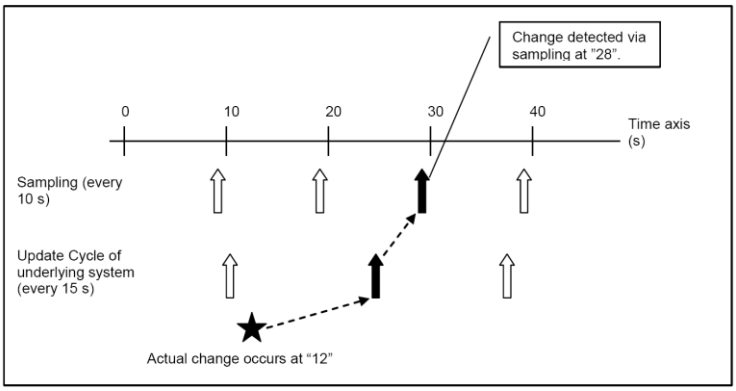
Il Sampling interval definisce la frequenza con cui il server campiona l'item reale al quale si riferisce il Monitored Item (attributo, valore di un Nodo oppure evento).

Il valore di default del sampling interval è il Publishing Interval della relativa Subscription. Un numero negativo indica che il client richiede l'assegnazione del valore di default (Publishing Interval).

Un client deve definire un intervallo di campionamento pari a 0 se la sottoscrizione è relativa ad eventi.

Se il client specifica un particolare Sampling Interval e questo non è supportato dal server, il server assegna al Monitored Item l'intervallo più appropriato, che viene restituito al client.

Il campionamento operato dal server OPC UA e il ciclo di aggiornamento del sistema sottostante di solito non sono sincronizzati. Ciò può causare ulteriori ritardi nel rilevamento cambiamento, come illustrato dalla seguente figura.



11.2 Monitoring Mode (Richiesto per l'Esame)

Il parametro Monitoring Mode è usato per abilitare e disabilitare sia il campionamento che la produzione delle Notifications. Questo parametro permette di configurare un Monitored Item in una delle tre possibili configurazioni:

- DISABLED_0; l'item associato al Monitored Item non è monitorato, e le Notifications non vengono prodotte e dunque non vengono accodate
- SAMPLING_1; l'item associato al Monitored Item è monitorato, e le Notifications vengono prodotte e vengono accodate. Ma le Notifications non vengono inoltrate al Client.
- REPORTING_2; l'item associato al Monitored Item è monitorato, e le Notifications vengono prodotte e vengono accodate. Il reporting è abilitato, ossia le Notifications vengono regolarmente inoltrate al Client allo scadere del Publishing Interval.

11.3 Filter (Richiesto per l'Esame)

Il parametro filtro definisce i criteri che il server utilizza per determinare se una Notification deve essere generata per un determinato Monitored Item. Il tipo di filtro dipende dal tipo di elemento che viene monitorato:

- Attributi. Il Filtro non viene definito nel caso in cui l'item reale associato al Monitored Item è un Attributo, in quanto la Notification viene automaticamente generata per ogni modifica del valore di attributo.
- Variabili; vengono attualmente definiti dalla standard due sole tipologie di filtro: DataChangeFilter e AggregateFilter che vengono utilizzati per il monitoraggio dei valori di variabili
- Oggetti e View; esiste un unico filtro definito dalla standard, denominato EventFilter.

Nel seguito viene riportata la struttura del DataChangeFilter, applicato alle Variabili. Dalla struttura si evincono tutti i parametri che devono essere specificati per tale filtro.

Name	Type	Description
DataChangeFilter	structure	
trigger	Enum DataChangeTrigger	Specifies the conditions under which a data change notification should be reported. It has the following values: STATUS_0 Report a notification ONLY if the *StatusCode* associated with the value changes. See Table 166 for *StatusCodes* defined in this standard. Part 8 specifies additional *StatusCodes* that are valid in particular for device data. STATUS_VALUE_1 Report a notification if either the *StatusCode* or the value change. The *Deadband* filter can be used in addition for filtering value changes. **This is the default setting if no filter is set.** STATUS_VALUE_TIMESTAMP_2 Report a notification if either StatusCode, value or the *SourceTimestamp* change. If a *Deadband* filter is specified, this trigger has the same behaviour as STATUS_VALUE_1. If the DataChangeFilter is not applied to the monitored item, STATUS_VALUE_1 is the default reporting behaviour.
deadbandType	UInt32	A value that defines the *Deadband* type and behaviour. Value deadbandType None_0 No *Deadband* calculation should be applied. Absolute_1 AbsoluteDeadband (see below) Percent_2 PercentDeadband (This type is specified in Part 8).
deadbandValue	Double	The *Deadband* is applied only if * the *trigger* includes value changes and * the *deadbandType* is set appropriately. Deadband is ignored if the status of the data item changes. ***DeadbandType = AbsoluteDeadband***: For this type the *deadbandValue* contains the absolute change in a data value that shall cause a *Notification* to be generated. This parameter applies only to *Variable*s with any *Number* data type. An exception that causes a *DataChange Notification* based on an AbsoluteDeadband is determined as follows: **Generate a *Notification* if (absolute value of (last cached value - current value) > *AbsoluteDeadband*)** The last cached value is defined as the last value pushed to the queue. If the item is an array of values, the entire array is returned if any array element exceeds the AbsoluteDeadband, or the size or dimension of the array changes. ***DeadbandType = PercentDeadband:*** This type is specified in Part 8

Nel seguito viene descritto l'algoritmo utilizzato nel caso di PercentDeadband.

DeadbandType = PercentDeadband
For this type of deadband the *deadbandValue* is defined as the percentage of the *EURange*. That is, it applies only to *AnalogItems* with an *EURange Property* that defines the typical value range for the item. This range shall be multiplied with the *deadbandValue* and then compared to the actual value change to determine the need for a data change notification. The following pseudo code shows how the deadband is calculated:

```
DataChange if (absolute value of (last cached value - current value) > (deadbandValue/100.0) *
((high-low) of EURange)))
```

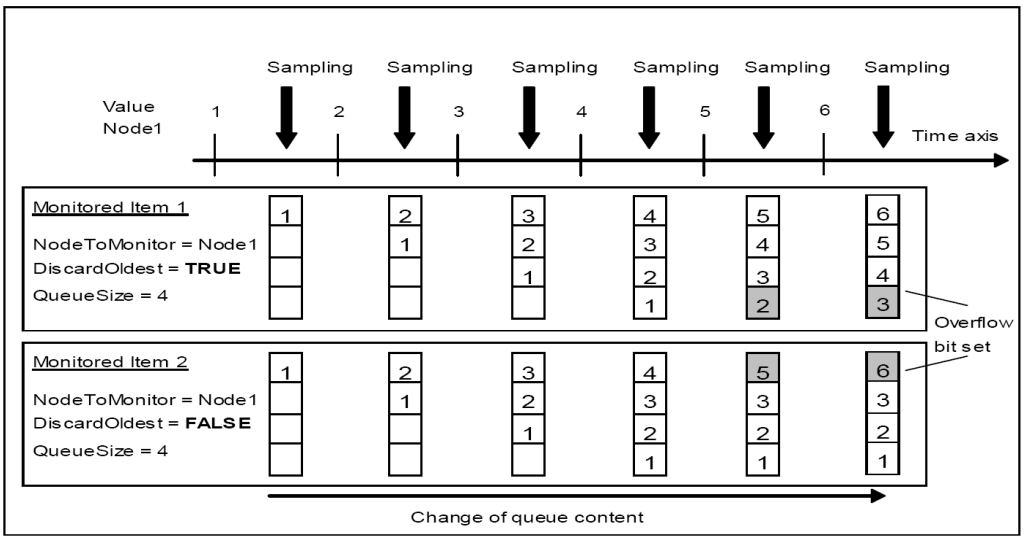
The range of the *deadbandValue* is from 0.0 to 100.0 Percent.

11.4 Queue Parameters (Richiesto per l'Esame)

I queue attributes definiscono le caratteristiche della coda utilizzata per mantenere le Notifications prodotte dai Monitored Items. Tra gli attributi vi è la lunghezza della coda e la politica di gestione della coda.

La dimensione della coda è definita quando viene creata la MonitoredItem; la dimensione può avere anche il valore 1 e in tal caso la coda collassa in un buffer.

Quando la coda è piena e una nuova Notification viene ricevuta, il server può applicare una delle due seguenti politiche: scarta la Notification più vecchia e accoda quella nuova al suo posto, o sostituisce l'ultimo valore aggiunto alla coda con la Notification appena prodotte. La scelta di una di queste due politiche viene fatta alla creazione del Monitored Item (tramite il settaggio di un parametro booleano chiamato DiscartOldest). La seguente figura illustra meglio le due poliche.



Se la dimensione della coda è uno, la coda diventa un buffer che contiene sempre l'ultima Notification prodotta. In questo caso, se l'intervallo di campionamento del Monitored Item è più veloce rispetto al Publishing Interval della sottoscrizione, il Monitored Item sovrascriverà Notification prodotte in precedenza e non trasmesse al client, e quest'ultimo riceverà sempre il valore più aggiornato.

D'altro canto, il Client può decidere di ricevere l'intera sequenza di Notification prodotte senza perderne neanche una; in questo caso, il MonitoredItem verrebbe creato con una dimensione della coda abbastanza grande da contenere tutte le Notifications generate tra due Publishing Interval consecutivi. Ad ogni ciclo di pubblicazione, la sottoscrizione invia così tutte le Notification in coda per il Monitored Item.

Il server deve restituire Notifications nello stesso ordine in cui sono in coda.

11.5 CreateMonitoredItems (Richiesto per l'Esame)

Viene usato per creare Monitored Items nell'ambito di una Subscription e definirne i settaggi iniziali. Le items create possono essere successivamente eliminate mediante il servizio DeleteMonitoredItems, oppure cancellando l'intera sottoscrizione.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader CreateMonitoredItems(
      RequestHeader                          requestHeader,
      uint                                   subscriptionId,
      Enum TimestampsToReturn                timestampsToReturn,
```

```
MonitoredItemCreateRequest          itemsToCreate[],
out ResponseHeader                  responseHeader,
out MonitoredItemCreateResult       results[],
out DiagnosticInfo                  diagnosticInfos[])
```

Request	
Parametri	**Descrizione**
requestHeader	Campo contenente diversi parametri che caratterizzano le richieste sottomesse su una Session; ad esempio timestamp, ossia il tempo al quale il client ha inviato la richiesta.
subscriptionId	Identificativo della Subscription, assegnato dal Server alla creazione della Subscription.
timestampsToReturn	In OPC UA sono definiti due tipi di timestamp, della sorgente dati e del server. Questo parametro permette al client di stabilire quale dei due il server dovrà restituire, assieme al valore dell'item monitorato. Si veda la Tabella riportata nel seguito.
itemsToCreate[]	Lista di MonitoredItems da creare. Ogni elemento della lista è composto dai seguenti tre elementi
itemToMonitor	Identifica l'item da monitorare. E' di tipo ReadValueId, descritto nel seguito.
monitoringMode	Specifica il Monitoring Mode, descritto in precedenza, con cui deve avvenire il monitoring dell'item associato al Monitored Item che si intende creare. Viene dettagliato anche nel seguito.
requestedParameters	Vengono specificati i parametri che regolano il monitoraggio dell'item. Vengono specificati nel seguito.
Response	
Parametri	**Descrizione**
responseHeader	Campo contenente diversi parametri che caratterizzano le risposte inviate; ad esempio timestamp, ossia il tempo al quale il Server ha inviato la risposta.
results[]	Lista dei risultati restituiti dal Server in risposta alla richiesta del client relativa alla creazione dei Monitored Items. Ogni elemento della lista è composto dai seguenti parametri.
StatusCode	Codice che indica se la creazione del Monitored Item è avvenuta con successo.
MonitoredItemId	Id assegnato dal server al Monitored Item creato, che dovrà essere fornito dal client nel caso di modifica o cancellazione dell'Item, in ingresso agli appositi servizi.
revisedSamplingInterval	Intervallo di campionamento accettato dal server, che può non essere uguale a quello richiesto dal client, se troppo basso.
revisedQueueSize	Grandezza della coda delle Notifications, creata dal server; può essere inferiore a quella indicata dal client.
filterResult	Contiene eventuali parametri rivisti dal Server e relativi al filtro richiesto dal client

Nel seguito vengono riportate le principali specifiche sui parametri del precedente servizio.

Table 167 – TimestampsToReturn Values

Value	Description
SOURCE_0	Return the source timestamp.
SERVER_1	Return the *Server* timestamp.
BOTH_2	Return both the source and *Server* timestamps.
NEITHER_3	Return neither timestamp. This is the default value for *MonitoredItems* if a *Variable* value is not being accessed. For *HistoryRead* this is not a valid setting.

Table 154 – ReadValueId

Name	Type	Description
ReadValueId	structure	Identifier for an item to read or to monitor.
nodeId	NodeId	*NodeId* of a *Node*.
attributeId	IntegerId	Id of the *Attribute*. This shall be a valid *Attribute* id. The *IntegerId* is defined in 7.13. The IntegerIds for the Attributes are defined in Part 6.
indexRange	NumericRange	This parameter is used to identify a single element of an array, or a single range of indexes for arrays. If a range of elements is specified, the values are returned as a composite. The first element is identified by index 0 (zero). The *NumericRange* type is defined in 7.21. This parameter is null if the specified *Attribute* is not an array. However, if the specified *Attribute* is an array, and this parameter is null, then all elements are to be included in the range.
dataEncoding	QualifiedName	This parameter specifies the *BrowseName* of the *DataTypeEncoding* that the *Server* should use when returning the Value *Attribute* of a *Variable*. It is an error to specify this parameter for other *Attributes*. A *Client* can discover what *DataTypeEncoding*s are available by following the *HasEncoding Reference* from the *DataType Node* for a *Variable*. OPC UA defines *BrowseNames* which *Servers* shall recognize even if the *DataType Nodes* are not visible in the *Server* address space. These *BrowseNames* are: DefaultBinary The default or native binary (or non-XML) encoding. DefaultXML The default XML encoding. Each *DataType* shall support at least one of these encodings. *DataTypes* that do not have a true binary encoding (e.g. they only have a non-XML text encoding) should use the *DefaultBinary* name to identify the encoding that is considered to be the default non-XML encoding. *DataTypes* that support at least one XML-based encoding shall identify one of the encodings as the DefaultXML encoding. Other standards bodies may define other well-known data encodings that could be supported. If this parameter is not specified then the *Server* shall choose either the DefaultBinary or *DefaultXML* encoding according to what *Message* encoding (see Part 6) is used for the *Session*. If the *Server* does not support the encoding that matches the *Message* encoding then the *Server* shall choose the default encoding that it does support. If this parameter is specified for a *MonitoredItem*, the *Server* shall set the *StructureChanged* bit in the *StatusCode* (see 7.33) if the *DataTypeEncoding* changes. The *DataTypeEncoding* changes if the *DataTypeVersion* of the *DataTypeDescription* or the *DataTypeDictionary* associated with the *DataTypeEncoding* changes.

Table 136 – MonitoringMode Values

Value	Description
DISABLED_0	The item being monitored is not sampled or evaluated, and *Notifications* are not generated or queued. *Notification* reporting is disabled.
SAMPLING_1	The item being monitored is sampled and evaluated, and *Notifications* are generated and queued. *Notification* reporting is disabled.
REPORTING_2	The item being monitored is sampled and evaluated, and *Notifications* are generated and queued. *Notification* reporting is enabled.

Table 128 – MonitoringParameters

Name	Type	Description
MonitoringParameters	structure	Parameters that define the monitoring characteristics of a *MonitoredItem*.
clientHandle	IntegerId	*Client*-supplied id of the *MonitoredItem*. This id is used in *Notifications* generated for the list *Node*. The *IntegerId* type is defined in 7.13.
samplingInterval	Duration	The interval that defines the fastest rate at which the *MonitoredItem*(s) should be accessed and evaluated. This interval is defined in milliseconds. The value 0 indicates that the *Server* should use the fastest practical rate. The value -1 indicates that the default sampling interval defined by the publishing interval of the *Subscription* is requested. A different sampling interval is used if the publishing interval is not a supported sampling interval. Any negative number is interpreted as -1. The sampling interval is not changed if the publishing interval is changed by a subsequent call to the *ModifySubscription Service*. The *Server* uses this parameter to assign the *MonitoredItems* to a sampling interval that it supports. The assigned interval is provided in the *revisedSamplingInterval* parameter. The Server shall always return a *revisedSamplingInterval* that is equal or higher than the requested *samplingInterval*. If the requested *samplingInterval* is higher than the maximum sampling interval supported by the *Server*, the maximum sampling interval is returned.
filter	Extensible Parameter MonitoringFilter	A filter used by the *Server* to determine if the *MonitoredItem* should generate a *Notification*. If not used, this parameter is null. The *MonitoringFilter* parameter type is an extensible parameter type specified in 7.16. It specifies the types of filters that can be used.
queueSize	Counter	The requested size of the *MonitoredItem* queue. The following values have special meaning for data monitored items: Value Meaning 0 or 1 the server returns the default queue size which shall be 1 as *revisedQueueSize* for data monitored items. The queue has a single entry, effectively disabling queuing. For values larger than one a first-in-first-out queue is to be used. The *Server* may limit the size in *revisedQueueSize*. In the case of a queue overflow, the *Overflow* bit (flag) in the *InfoBits* portion of the *DataValue statusCode* is set in the new value. The following values have special meaning for event monitored items: Value Meaning 0 the *Server* returns the default queue size for *Event Notifications* as *revisedQueueSize* for event monitored items. 1 the *Server* returns the minimum queue size the Server requires for *Event Notifications* as *revisedQueueSize*. MaxUInt32 the *Server* returns the maximum queue size that the *Server* can support for *Event Notifications* as *revisedQueueSize*. If a *Client* chooses a value between the minimum and maximum settings of the *Server* the value shall be returned in the *revisedQueueSize*. If the requested *queueSize* is outside the minimum or maximum, the *Server* shall return the corresponding bounding value. In the case of a queue overflow, an *Event* of the type *EventQueueOverflowEventType* is generated.
discardOldest	Boolean	A boolean parameter that specifies the discard policy when the queue is full and a new *Notification* is to be queued. It has the following values: TRUE the oldest (first) *Notification* in the queue is discarded. The new *Notification* is added to the end of the queue. FALSE the last *Notification* added to the queue gets replaced with the new *Notification*.

11.6 DeleteMonitoredItems

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader DeleteMonitoredItems(
      RequestHeader                requestHeader,
      uint                         subscriptionId,
      UInt32Collection             monitoredItemIds,
      out StatusCodeCollection     results,
      out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
subscriptionId	Id della Subscription da cui rimuovere l'item. Dev'essere lo stesso che era stato restituito dal server nella chiamata a CreateSubscription.
monitoredItemIds	Lista degli item da cancellare. Gli id devono essere gli stessi che erano stati restituiti dal server nella chiamata a CreateMonitoredItems.
Results	Lista di codici di stato che indicano se, per ogni item in ingresso, la cancellazione ha avuto successo.

11.7 ModifyMonitoredItems

Il servizio serve a modificare le Monitored Items in una sottoscrizione.

namespace: Opc.Ua.SessionClient

```
virtual ResponseHeader ModifyMonitoredItems(
      RequestHeader                          requestHeader,
      uint                                   subscriptionId,
      TimestampsToReturn                     timestampsToReturn,
      MonitoredItemModifyRequestCollection   itemsToModify,
      out MonitoredItemModifyResultCollection results,
      out DiagnosticInfoCollection           diagnosticInfos)
```

Parametri	Descrizione
I parametri di ingresso e uscita hanno descrizione analoga a quelli di CreateMonitoredItems	

11.8 SetMonitoringMode

Si usa per impostare la modalità di monitoraggio per ogni item.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader SetMonitoringMode(

      RequestHeader                requestHeader,

      uint                         subscriptionId,
      MonitoringMode               monitoringMode,
      UInt32Collection             monitoredItemIds,
      out StatusCodeCollection     results,
      out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
subscriptionId	Id della Subscription contenente l'item di cui impostare la modalità di monitoraggio.
monitoringMode	Modalità di monitoraggio dell'item. Può essere "disabled", "sampling" o "reporting".
monitoredItemIds	Lista degli id degli item di cui impostare il monitoring mode. Devono essere gli stessi che sono stati stabiliti dal server nella chiamata a CreateMonitoredItems.
Results	Codici di stato che indicano, per ogni item in ingresso, se l'impostazione del monitoring mode ha avuto successo o meno.

11.9 SetTriggering

Nell'ambito di una sottoscrizione, è possibile fare in modo che i campionamenti effettuati su un Monitored Item (il triggered item) vengano notificati al client solo quando sia disponibile e pronta per l'invio la notifica di un altro specifico item (il triggering item).Questo viene realizzato con questo servizio, con il quale vengono create delle speciali relazioni tra triggering e triggered items.

Il Monitoring Mode del triggered item deve essere settato a "sampling" (vedi 8.9 – Creazione e gestione di Monitored Items).

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader SetTriggering(
      RequestHeader                requestHeader,
      uint                         subscriptionId,
      uint                         triggeringItemId,
      UInt32Collection             linksToAdd,
      UInt32Collection             linksToRemove,
      out StatusCodeCollection     addResults,
      out DiagnosticInfoCollection addDiagnosticInfos,
      out StatusCodeCollection     removeResults,
      out DiagnosticInfoCollection removeDiagnosticInfos)
```

Parametri	Descrizione
subscriptionId	Id della Subscription contenente triggered e triggering items.
triggeringItemId	Id del Monitored Item da impostare come triggering.
linksToAdd	Lista di id degli items da impostare come triggered.
linksToRemove	Lista di id degli items che non devono più essere triggered.
addResults	Lista di codici di stato (uno per ogni item da impostare come triggered) che indicano se l'operazione è andata a buon fine o meno.
removeResults	Lista di codici di stato (uno per ogni item che non deve più essere triggered) che indicano se l'operazione è andata a buon fine o meno.

12 Invio delle Notifications al Client (Richiesto per l'esame)

Il server deve essere in grado di inviare informazioni al client. Come è stato visto nella sezione precedente il client può definire una o più Subscription nell'ambito di una Sessione attiva; per ciascuna Subscription è stato anche spiegato che il Client può creare dei Monitored Item che sono di fatto dei produttori di informazioni (Notifications) accumulate in specifiche code. Ad una frequenza costante (il Publish Interval), per ogni Subscription, i contenuti attuali di tutte le code dei Monitored Items afferenti alla Subscription, vengono inglobati in un NotificationMessage, per essere consegnati al client. In questa sezione viene illustrato il meccanismo di invio dei NotificationMessage prodotti per ogni Subscription di una Sessione attiva.

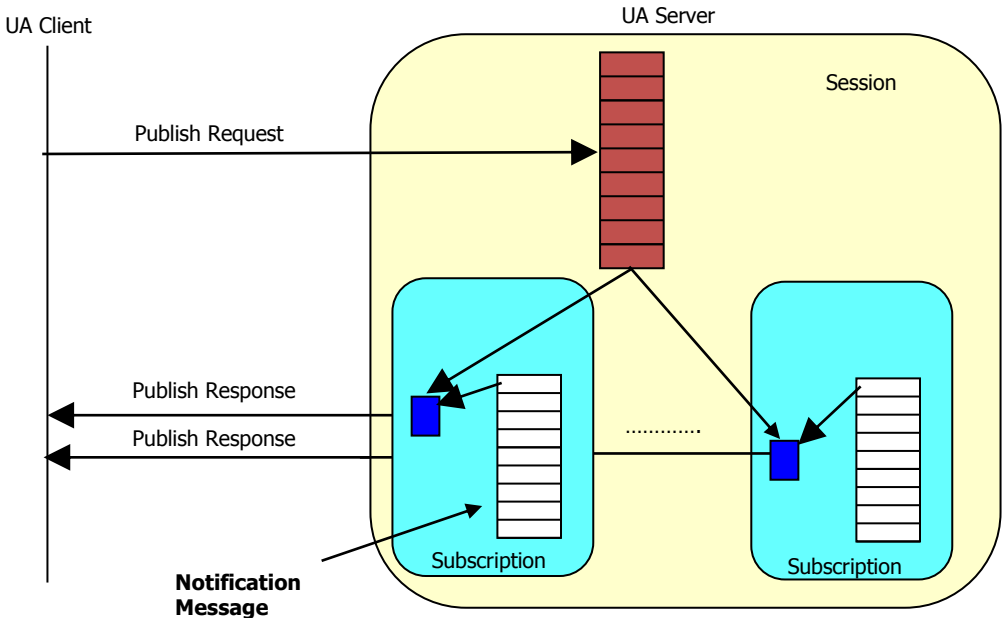
Nelle precedenti versioni di OPC l'invio di informazioni da parte del Server si poteva realizzare definendo, lato Client, apposite interfacce di callback che permettevano al server di richiamare metodi sul client per inviare le informazioni. In OPC UA, il meccanismo di comunicazione si basa su connessioni unidirezionali, rispettando i requisiti di firewall-friendship ed immutabilità al variare del livello di trasporto sottostante, stabiliti in fase di progettazione dello standard, senza definire alcuna interfaccia di callback. Gli elementi principali che caratterizzano l'invio di informazioni da parte del Server sono:

- Invio di NotificationMessage da parte del Server regolato da flussi di richieste inviate dal Client; un NotificationMessage viene inviato al client in risposta a una Publish Request inviata dal client. Il Server mantiene per ogni Sessione attiva una coda di Publish Request ricevute dal client per quella sessione. Per ogni NotificationMessage prodotto da una qualunque delle sottoscrizioni afferenti ad una sessione, un Publish Request viene estratto dalla coda delle Publish Request e il NotificationMessage viene inviato al client tramite una Publish Response inviata in risposta alla Publish Request.
- Ogni NotificationMessage è identificato in modo univoco da un numero di sequenza specificato dal Server nell'invio della Publish Response. L'utilizzo di un numero di sequenza permette al client di scoprire l'eventuale perdita di NotificationMessage inviate dal Server.
- Gestione di Ack inviati dal client al server per informare il server circa la corretta ricezione dei NotificationMessage ricevuti, tramite la notifica al server dei numeri di sequenza ricevuti.
- Re-invio delle NotificationMessage andate perdute, su esplicita richiesta da parte del client.
- Invio periodico di "keep-alive" Message dal server al client quando non sono disponibili NotificationMessage da inviare al client; in questo modo il server notifica al client che la sessione è ancora attiva.

Tramite il Publish Service Request, il client invia al Server una particolare richiesta senza aspettarsi una risposta immediata; a tale richiesta il Client si aspetta di ricevere una Publish Service Response contenente un NotificationMessage. Cosa è una NotificationMessage ? La risposta è semplice se si pensa che durante la creazione di una Subscription (servizio CreateSubscription), il client specifica un parametro denominato maxNotificationsPerPublish. Tale parametro indica il numero di Notifications che il client desidera ricevere in una Publish Service Response; un valore 0 indica che il client non vuole specificare alcun limite particolare. Un NotificationMessage viene prodotto allo scadere del PublishingInterval in ogni sottoscrizione.

Da porre estrema attenzione al seguente concetto; una Publish Service request non si riferisce a nessuna Subscription particolare, ma unicamente alla Sessione entro cui la request viene inoltrata. Dunque per ogni Publish Service request, il Server sarà tenuto ad inoltrare al client un NotificationMessage tra quelli prodotti dalle Subscriptions presenti nella Sessione attiva. Ovviamente se non vi sono NotificationMessage, il Server attenderà che ne verrà prodotto uno, che verrà inoltrato al Client con una Publish Service response. Nel caso in cui più NotificationMessage siano stati prodotti e siano pronti per essere trasmessi, il Server dovrà decidere quale NotificationMessage prelevare ed inviare al Client. La politica che viene definita nello standard tiene conto della priorità della Subscription, definita all'atto della creazione della Subscription (si veda il parametro Priority della CreateSubscription). Nel caso in cui più NotificationMessage siano pronti per diverse Subscription, il Server dovrebbe estrarre la NotificationMessage relativa alla Subscription con più alta priorità. Nel caso le Subscriptions abbiano stessa priorità, il Server dovrebbe servire le diverse Subscriptions secondo una modalità round-robin.

La figura seguente dovrebbe aiutare a comprendere quanto detto; come si vede per ciascuna Publish Request pervenuta, il Server sceglie una Subscription (quella a più priorità o quella scelta in base ad una politica round-robin). Per la subscription scelta, viene prelevato un NotificationMessage che viene poi spedito entro un Publish Response. La stessa figura mostra una coda di Publish Request in cui vengono accodate tutte le richieste avanzate dal client e pervenuta alla sessione. Ogni Publish Response viene estratta da tale coda per ogni NotificationMessage prodotto nella sessione, da una qualunque sottoscrizione.



La precedente figura mostra come un NotificationMessage venga inoltrato al client tramite una Publish Response.

Bisogna prestare attenzione al fatto che l'invio di NotificationMessage da parte del server non è arbitrario, ma la Publish Response è subordinata alla precedente ricezione di una Request; in altre parole, quando è prodotto una NotificationMessage (allo scadere di ogni Publish Interval), esso può essere inoltrato ad un client solo se la coda delle Publish Request non è vuota. Si comprende come la frequenza di invio di Publish

Request da parte del client è un parametro essenziale per le prestazioni dell'intero sistema di scambio dati client/server. Se il client invia Publish request con bassa frequenza, ciò comporterà un accumulo di NotificationMessage lato Server con conseguenti ritardi di trasmissione. Se la frequenza di invio di Publish Request è viceversa molto alta, c'è il rischio di saturare la larghezza di banda del canale di comunicazione, introducendo allo stesso modo ritardi di comunicazione. L'algoritmo di invio di Publish Request viene scelto dal Client e lo standard ovviamente non fornisce alcun suggerimento; in linee generali, comunque la frequenza di invio di Publish Request da parte del Client dovrebbe essere legata ed influenzata da questi parametri:

- Numero di sottoscrizioni create dal Client e dai valori dei relativi Publish Interval;
- Latenza della rete;

L'invio delle Publish Request dovrebbe poi essere modulato (dunque la relativa frequenza variata nel tempo) in base a diversi fattori. Ad esempio, potrebbe essere necessario che il Client incrementi la frequenza di invio di Publish Request se la latenza della connessione di rete è particolarmente alta, il che può essere rilevato basandosi sui valori di timestamp all'interno dei messaggi di richiesta e risposta. In caso di alto numero di sottoscrizioni e bassa latenza di rete, il numero di Publish Request in sospeso può essere ridotto.

Ad ogni NotificationMessage inviato al Client tramite una Publish Response, viene associato un SequenceNumber. Il SequenceNumber è un unsigned integer di 32 bits che è incrementato di uno per ogni NotificationMessage inviato per una determinata subscription. Il primo NotificationMessage inviato per una determinata subscription ha un SequeceNumber pari a 1; per ogni NotificationMessage inviato per una determinata subscription viene incrementato di 1.

Per ogni Publish Response, il Server specifica il NotificationMessage, la SubscriptionId a cui tale NotificationMessage si riferisce, il relativo SequenceNumber. Infine il Server indica un elenco di SequenceNumber relative sempre alla stessa SubscriptionId che ancora non sono state riconosciute dal Client.

Il Client ricevendo la Publish Response e leggendo tale elenco si rende conto dei SequenceNumber non ancora riconosciuti e può valutare se in effetti i relativi NotificationMessages non sono stati mai ricevuti. In tal caso ne richiederà la ritrasmissione tramite una Republish.

Al fine di notificare al Server la corretta ricezione di uno o più NotificationMessage, in ogni Publish Request inviata dal client vengono elencate le coppie <SubscriptionId, SequenceNumber> per ogni NotificationMessage correttamente ricevuto dal Client. In tal modo il Server può liberare risorse deallocando l'area necessaria alla memorizzazione di tali NotificationMessage.

Ogni Session attiva deve mantenere una coda di ritrasmissione contenente tutti i NotificationMessage inviati e non ancora confermati dal client. Per ogni coppia <SubscriptionId, SequenceNumber> ricevuta dal client tramite la Publish Request, il Server rilascerà le risorse necessarie alla memorizzazione del relativo NotificationMessage. Per ogni richiesta di Republish ricevuta dal client, il Server provvederà a ritrasmettere il NotificationMessage accodato nella suddetta coda di ritrasmissione; tale messaggio sarà rilasciato solo alla ricezione della conferma di corretta ricezione da parte del client.

Come è stato detto in precedenza, le Subscriptions matengono un keep-alive counter, al cui scadere una Publish Request deve essere estratta dalla coda delle Publish Request ricevute dal client ed invia una Publish Response includendo in esso non un NotificationMessage (visto che non ce ne sono disponibili al momento dello scadere del keep-alive counter), ma un messaggio di keep-alive. In tal modo il Server informa il client che la Subscription è ancora attiva.

12.1 Publish (Richiesto per l'esame)

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader Publish(
      requestHeader requestHeader,
      SubscriptionAcknowledgementCollection subscriptionAcknowledgements[],
      out ResponseHeader                    responseHeader,
      out uint                              subscriptionId,
      out UInt32Collection                  availableSequenceNumbers[],
      out bool                              moreNotifications,
      out NotificationMessage               notificationMessage,
      out StatusCodeCollection              results[],
      out DiagnosticInfoCollection          diagnosticInfos[])
```

Request	
Parametri	**Descrizione**
requestHeader	Campo contenente diversi parametri che caratterizzano le richieste sottomesse su una Session; ad esempio timestamp, ossia il tempo al quale il client ha inviato la richiesta.
subscriptionAcknowledgments[]	Lista delle coppie <SubscriptionId, SequenceNumber> per consentire al client di notificare la corretta ricezione di NotificationMessage. La lista può contenere diverse notifiche per la stessa sottoscrizione (in tal caso vi saranno diverse coppie con la stessa SubscriptionId). Ogni NotificationMessage notificato può essere liberato dalla coda di ritrasmissione della Sessione.
SubscriptionId	Id della sottoscrizione relativa al NotificationMessage.
SequenceNumber	Numero di sequenza del NotificationMessage.
Response	
responseHeader	Campo contenente diversi parametri che caratterizzano le risposte inviate su una Session; ad esempio timestamp, ossia il tempo al quale il Server ha inviato la risposta.
subscriptionId	Id della sottoscrizione relativa al NotificationMessage restituito.
availableSequenceNumbers[]	Lista di numeri di sequenza per i quali il server non ha ancora ricevuto l'ack da parte del client. I numeri di sequenza sono relativi al subscriptionId precedentemente specificato.
moreNotifications	Flag che indica se il server non è riuscito ad inviare tutte le notifiche disponibili nella Response corrente.
notificationMessage	Struttura contenente il NotificationMessage.
SequenceNumber	Numero di sequenza del NotificationMessage.
PublishTime	Istante di tempo in cui il messaggio corrente è stato inviato al client.

	NotificationData[]	Lista di Notifications, raggruppate per tipologia: DataChange o notifiche di eventi.
results[]	Lista di risultati	
diagnosticInfos[]	Lista di informazioni diagnostiche	

12.2 Republish

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader Republish(
            RequestHeader           requestHeader,
            uint                    subscriptionId,
            uint                    retransmitSequenceNumber,
            out NotificationMessage notificationMessage)
```

Parametri		Descrizione
subscriptionId		Id della sottoscrizione relativa al NotificationMessage da re-inviare.
retransmitSequenceNumber		Numero di sequenza del NotificationMessage da re-inviare
notificationMessage		Struttura contenente il NotificationMessage re-inviato dal Server.
	SequenceNumber	Numero di sequenza del NotificationMessage.
	PublishTime	Istante di tempo in cui il messaggio corrente è stato inviato al client.
	NotificationData[]	Lista di Notifications, raggruppate per tipologia: DataChange o notifiche di eventi.

13 Method Service Set

Lo standard OPC UA permette ai server di esporre nel proprio Address Space dei Methods che possono essere richiamati dai client. I metodi sono componenti degli Objects e possono essere richiamati nel contesto di un Object. Tutte le informazioni necessarie per chiamare un metodo, inclusa la descrizione dettagliata dei parametri di input e di output, sono disponibili nella sua descrizione.

13.1 Call

Questo servizio è usato per richiamare un metodo di un server da parte di un client che abbia già tutte le informazioni necessarie, o perché "built in", oppure perchè ricavate dall'Address Space del server.

Può anche essere utilizzato per richiamare una lista di Methods in modo da ridurre il "roundtrip" tra client e server.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader Call(
      RequestHeader                  requestHeader,
      CallMethodRequestCollection    methodsToCall,
      out CallMethodResultCollection results,
      out DiagnosticInfoCollection   diagnosticInfos)
```

Parametri	Descrizione
methodsToCall	Lista di metodi da richiamare.
ObjectId	NodeId dell'Object o del nodo ObjectType che fornisce il metodo.
MethodId	NodeId del metodo da richiamare nel contesto dell'Object o dell'ObjectType.
InputArguments	Lista dei valori degli argomenti di input del metodo. Il numero di argomenti di input necessari ed i loro tipi di dato opportuni sono definiti nella proprietà InputArgument del metodo nell'Address Space.
results	Lista di risultati per ogni chiamata di metodo effettuata.
StatusCode	Codice del risultato per ogni chiamata.
InputArgumentResults	Lista di codici-risultato per ogni argomento di input della chiamata al metodo.
OutputArguments	Lista dei valori degli argomenti di output del metodo. Il numero di argomenti di output ed i loro tipi di dato sono definiti nella proprietà OutputArgument del metodo nell'Address Space.

14 Query Service Set

Il servizio di Browse si utilizza per navigare attraverso un Address Space e recuperare informazioni, ma è realmente utile solo se lo space non è molto esteso e non è dinamico. Ad es., i server relativi a sistemi con Information Models complessi possono avere Address Spaces composti anche da più di un milione di nodi.

Per ricavare informazioni in questo tipo di spaces, OPC UA fornisce il meccanismo delle Query, con le quali è possibile definire criteri di filtraggio per ottenere particolari sottoinsiemi di nodi (ed informazioni correlate). Il funzionamento delle Query si basa sulle informazioni sui tipi: nella prima parte si specifica il tipo degli Object o Variable a cui il client è interessato; nella parte di SELECT (DataToReturn) si indica quali istanze di quel tipo devono essere ritornate; nella parte di WHERE vengono specificati i criteri di filtraggio.

14.1 QueryFirst

Il servizio QueryFirst si usa per inoltrare una nuova query e riceverne i risultati; nel caso in cui il loro numero sia troppo elevato, si utilizza il servizio QueryNext per recuperare quelli eccedenti.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader QueryFirst(
      RequestHeader                 requestHeader,
      ViewDescription               view,
      NodeTypeDescriptionCollection nodeTypes,
      ContentFilter                 filter,
      uint                          maxDataSetsToReturn,
      uint                          maxReferencesToReturn,
      out QueryDataSetCollection    queryDataSets,
      out byte[]                    continuationPoint,
      out ParsingResultCollection   parsingResults,
      out DiagnosticInfoCollection  diagnosticInfos,
      out ContentFilterResult       filterResult)
```

Parametri	Descrizione
view	vista su cui effettuare la Query.
nodeTypes	Array di strutture contenenti i filtri di tipo per la query.
TypeDefinitionNode	NodeId del TypeDefinitionNode (che può essere VariableType o ObjectType) su cui devono essere restituite le istanze.
IncludeSubTypes	Indica se devono essere incluse anche le istanze dei sottotipi.
DataToReturn	Specifica nodi ed attributi delle istanze che devono essere restituite.
RelativePath	Path dall'istanza del tipo specificato al componente dell'istanza.
AttributeId	Id dell'attributo da restituire per il nodo selezionato.
filter	Filtro utilizzato per ridurre il numero dei Nodi ritornati e dei loro attributi.
maxDataSetsToReturn	Questo parametro permette al client di limitare il numero dei risultati restituiti.
continuationPoint	Viene ritornato quando il server non è in grado di

	restituire tutti i risultati all'interno del Response di QueryFirst, il che può accadere a causa di una limitazione settata dal client nella richiesta o dal server stesso durante il processamento della Query. Può essere passato in ingresso al servizio QueryNext in modo da recuperare i rimanenti risultati.
queryDataSets	Lista dei risultati.
NodeId	NodeId del Nodo istanza recuperato con la Query inoltrata nella richiesta.
TypeDefinitionNode	NodeId del TypeDefinition del Nodo istanza ritornato.
Values	Lista di valori per l'Attribute selezionato.

15 Node Management Service Set

I servizi di NodeManagement consentono ad un client OPC UA di creare e cancellare nodi e riferimenti nell'Address Space di un server OPC UA.

15.1 AddNodes

Con il servizio AddNodes un client può aggiungere uno o più nodi all'Address Space di un server.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader AddNodes(
      RequestHeader                requestHeader,
      AddNodesItemCollection       nodesToAdd,
      out AddNodesResultCollection results,
      out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
nodesToAdd	Lista di nodi da aggiungere all'Address Space del server.
ParentNodeId	Id del nodo che dovrà contenere il riferimento al nuovo nodo.
ReferenceTypeId	Id del tipo di riferimento da creare tra il nuovo nodo ed il nodo padre.
BrowseName	Browse Name per il nuovo nodo.
NodeClass	Node Class per il nuovo nodo.
NodeAttributes	Parametro variabile contenente i valori degli Attributes addizionali, in base al tipo di NodeClass impostata.
TypeDefinition	Id della TypeDefinition utilizzata per descrivere quale tipo di Object o Variable, ognuno con i suoi componenti, dev'essere creato. Questo parametro non deve essere impostato se NodeClass non è un Object o una Variable.
Results	Lista di risultati per ogni operazione di aggiunta richiesta.
StatusCode	Codice del risultato per ogni operazione di aggiunta.
AddedNodeId	Id del nuovo nodo creato.

15.2 AddReferences

È usato per creare una o più Reference tra i nodi nell'Address Space.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader AddReferences(
      RequestHeader                requestHeader,
      AddReferencesItemCollection  referencesToAdd,
      out StatusCodeCollection     results,
```

```
      out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
referencesToAdd	Lista di References da creare.
SourceNodeId	Id del nodo sorgente della nuova Reference.
ReferenceTypeId	Id del tipo di tipo di riferimento da creare tra il nodo sorgente ed il nodo target.
TargetNodeId	Id del nodo target della nuova Reference.
results	Lista di risultati per ogni operazione di aggiunta richiesta.

15.3 DeleteNodes

Il servizio DeleteNodes è usato per rimuovere uno o più nodi dall'Address Space.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader DeleteNodes(
      RequestHeader                requestHeader,
      DeleteNodesItemCollection    nodesToDelete,
      out StatusCodeCollection     results,
      out DiagnosticInfoCollection diagnosticInfos)
```

Parametri	Descrizione
nodesToDelete	Lista di nodi da cancellare dall'Address Space del server.
NodeId	Id del nodo da cancellare.
DeleteTargetReferences	Un flag che indica se il server deve cancellare anche il riferimento al nodo target.
results	Lista di risultati per ogni operazione di cancellazione richiesta.

15.4 DeleteReferences

È usato per rimuovere uno o più riferimenti nell'Address Space.

namespace: Opc.Ua.SessionClient

```
public virtual ResponseHeader DeleteReferences(
      RequestHeader                  requestHeader,
      DeleteReferencesItemCollection referencesToDelete,
      out StatusCodeCollection       results,
      out DiagnosticInfoCollection   diagnosticInfos)
```

Parametri	Descrizione
referencesToDelete	Lista di References da cancellare dall'Address Space del server.

	SourceNodeId	Id del nodo sorgente della Reference da eliminare.
	ReferenceTypeId	Id del tipo di tipo di Reference da cancellare tra il nodo sorgente ed il nodo target.
	TargetNodeId	Id del nodo target della Reference da cancellare.
results		Lista di risultati per ogni operazione di cancellazione richiesta.