UNIVERSITÀ DEGLI STUDI DI CATANIA

OPC-UA Aggregation Server

Industrial Informatics a.a 2019/2020

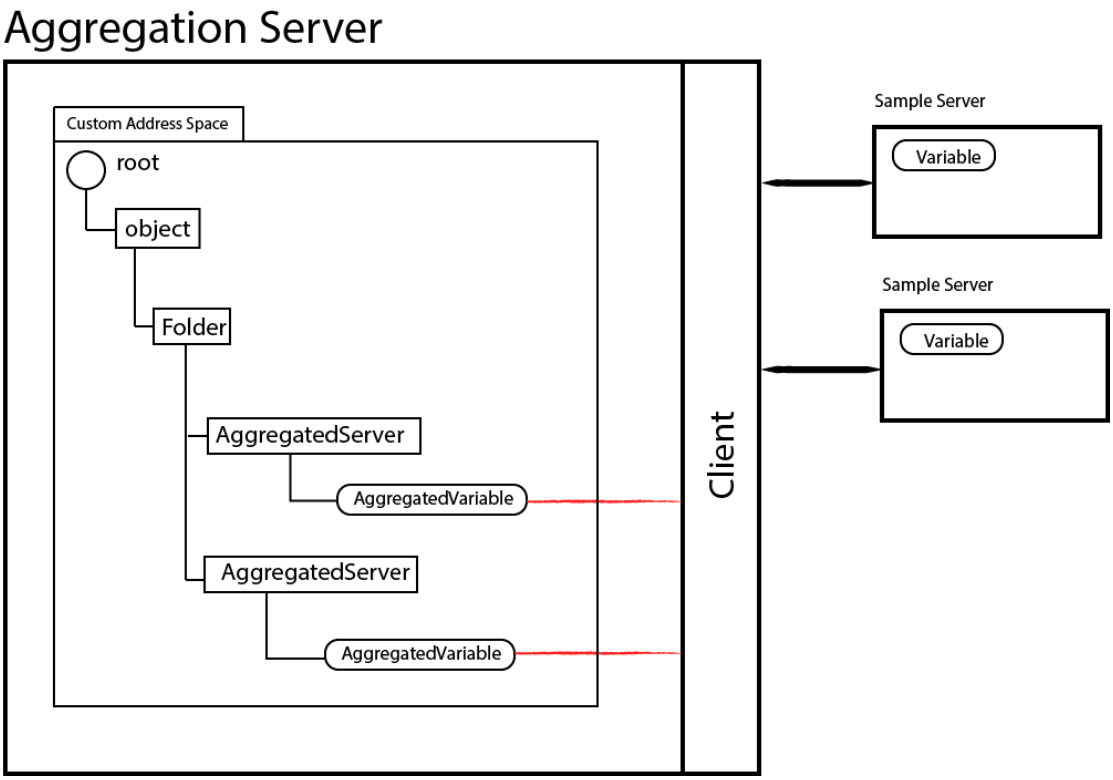
Raiti Mario O55000434
Nardo Gabriele Salvatore O55000430

Lo scopo di questa tesina di fine corso è la realizzazione di un Aggregation Server utilizzando la versione in python dello stack OPC-UA , disponibile gratuitamente su github al seguente link (https://github.com/FreeOpcUa/python-opcua).

Il codice sorgente dell'elaborato è disponibile su github al corrispondente indirizzo (aggiungere link).

Sommario

Aggregation Server – Architettura



La figura in alto mostra l'architettura di base dell'elaborato. L'elemento Aggregation server sarà un Server OPC-UA. L'address space è stato customizzato creando un nuovo namespace specifico per l'applicazione e ai suoi componenti di base è stato aggiunto un Node di tipo folder che avrà lo scopo di raccogliere e organizzare gli oggetti AggregatedServer. Tali oggetti modellano i sample server da aggregare. A tal proposito è stato creato un nuovo Object Type custom chiamato AggratedServer al quale è stato aggiunto un set di variabili che modellano i valori di cui si vuole tener traccia.

All'interno dell'aggregation server è previsto un modulo client che avrà il compito di stabilire le connessioni con i sample server al fine di leggere e scrivere le variabili aggregate. Le informazioni relative ai sample server da aggregare, e che quindi il modulo client deve raggiungere, sono contenuti all'interno di un file di configurazione in formato *json* (tale file verrà discusso in dettaglio in seguito) in cui sono anche indicati i nodeId delle informazioni da recuperare e le modalità di recupero cioè tramite subscription o polling (read/write).

I valori prelevati dal modulo client devono essere sincronizzati con le copie locali dell'aggregation server, cioè le variabili degli AggregatedServer. Tali variabili devono quindi mantenere come source timestamp quello del sample server.

File di Configurazione

In questa sezione verranno descritti i file di configurazione json utilizzati per il passaggio delle informazioni di configurazione e per la creazione dei certificati x509v3.

Config.json

Questo file contiene un elemento sample server per ogni server che si vuole aggregare. Per ogni elemento sono previsti sei campi da configurare opportunamente per settare le informazioni relative al server e ai dati da tracciare. Di seguito vengono descritti tali campi e per ognuno di essi sarà presentato in basso un esempio di valore ammissibile:

- **Endpoint**: deve contenere l'url del server che si vuole aggregare;
- **security_policy**: deve contenere una stringa che rappresenti l'algoritmo utilizzato per le operazioni di sicurezza ove previste, in accordo al campo security mode;
- **security_mode**: deve contenere una stringa contente la modalità di sicurezza richiesta; i valori ammissibili sono: None, Sign e SignAndEncrypt;
- **node_id**: deve contenere il node id della variabile di cui si vogliono ottenere i valori sotto forma di stringa formattata nel seguente modo: 'ns=valore;i=valore';
- **variable_type**: deve contenere il tipo della variabile da leggere;
- **service_req**: definisce il tipo di servizio per otternere i dati; i valori ammissibili sono tre: read, write, subscribe;
- **new_value**: definisce il nuovo valore che si vuole scrivere (da settare solamente se si sceglie service_req: *write*);
- **publish_interval**: definisce il publish interval in millisecondi come intero senza segno (da settare solamente se si sceglie service_req: *subscribe)*.

In basso viene riportato un esempio di come riempire i campi del suddetto file.

```
{

   "sample_server1": {

        "endpoint":"opc.tcp://pc-mario:51210/UA/SampleServer",

        "security_policy":"None",

        "security_mode":"None",

        "node_id":"ns=2;i=10852",

        "variable_type":"DataValue",

        "service_req":"read",

        "new_value": "",

        "publish_interval": ""

    },

}
```

Openssl_conf.json

In questo file sono presenti due campi da settare opportunamente per la corretta esecuzione del programma, utilizzati per la creazione dei certificati. Anche questo file è in formato *json* ed è composto da due campi:

- **ssl_installation_path:** in questo campo va inserito il proprio path di installazione di openssl.
- **ssl_confing_file_name:** va inserito il nome del file di configurazione di openssl.

In basso viene riportato un esempio di come riempire i campi del suddetto file.

```
{

    "ssl_installation_path":"C:\Program Files\OpenSSL-Win64\bin\openssl.cfg",

    "ssl_confing_file_name":"openssl.cfg"

}
```

Implementazione

In questa sezione saranno mostrati i dettagli implementativi, discusse le scelte progettuali e le funzionalità sviluppate nei file *aggregationServer.py*, *Client.py*, *Thread_client.py*.

L'elaborato è stato sviluppato in ambiente Windows (Windows 10 Professional), per tale motivo le scelte implementative sono mirate all'esecuzione su tale piattaforma (gestione dei path). Da ciò ne consegue che potrebbero incorrere errori durante l'esecuzione su altre piattaforme diverse da quella presa in considerazione.

Il codice è stato sviluppato utilizzando l'ultima versione di python cioè la 3.8. Come editor è stato utilizzato *VScod*e.

aggregationServer.py

(Descrivere in maniera dettagliata il file aggragtionServer.py)

Client.py

(Descrivere in maniera dettagliata il file Client.py)

Thread_client.py

(Descrivere in maniera dettagliata il file Client.py)

Risultati

(Qui potremmo discutere i risultati delle esecuzioni con screen)

Note sulle funzioni dello stack

Il modulo Client viene istanziato tramite la chiamata al costruttore della classe 'Client' importata dallo stack opc ua per python utilizzato in questo progetto. La chiamata effettuata è Client(server_path), dove *server_path* è l'url del sample server alla quale vogliamo che si colleghi il nostro modulo Client.

Dopo aver istanziato il Client, dobbiamo caricare il certificato, la private key e settare eventuali security policy e security mode: questo viene fatto tramite la funzione *set_security_string* della classe Client dello stack, che prende come parametro una stringa

che ha tutte le informazioni sopra citate. Questo ci permetterà di connetterci all'endpoint consono con le politiche di sicurezza scelte.

Per connetterci all'endpoint, viene usata la funzione *connect* dello stack, che maschera le varie sequenze di azioni da intraprendere per connettere un client ad un endpoint. Infatti, questa funzione dello stack, chiamerà al suo interno altre funzioni innestate per la *creazione del canale sicuro*, *creazione della sessione* e *attivazione* della stessa.

Come controparte della *connect*, la funzione *disconnect* dello stack, maschera le varie sequenze di azioni per disconnettere un client ad un endpoint.

Per leggere un qualsiasi valore dato un node id, ciò che bisogna fare è ottenere innanzi tutto il nodo tramite il node id ed in seguito leggerne il valore. Queste due azioni vengono eseguite tramite la funzione *get_node* della classe Client e la funzione *get_data_value* della classe Node dello stack opc ua.

Per scrivere un valore dato un node id, la sequenza di azioni è identica della read, eccetto che viene utilizzata la funzione *set_value* della classe Node dello stack piuttosto che la get_data_value.

Per effettuare una sottoscrizione e creare i monitored items data una lista di node id, vengono utilizzate due funzioni dello stack: *create_subscription,* chiamata sulla classe Client e *subscribe_data_change,* chiamata sulla sottoscrizione appena creata che appartiene alla classe Subscription dello stack. *create_subscription* ha come parametri di ingresso il publish_interval e un handler che viene istanziato tramite la classe SubHandler creata nel nostro progetto, ma che deve possedere le funzioni *datachange_notifications* ed *event_notifications* specificate dallo stack. Esse verranno chiamate per notificare un cambiamento dei valori delle variabili. *subscribe_data_change* ha come parametro di ingresso una lista di variabili, ottenute tramite la lista dei node id di partenza. Questa funzione creerà per noi un montored item per ogni variabile della lista e ci tornerà una lista contenente i monitored items id dei monitored items appena creati.

Se non siamo più interessati ad essere aggiornati sui cambiamenti di valore di una variabile, possiamo passare il monitored item id associato alla variabile come parametro di ingresso alla funzione dello stack *unsubscribe,* chiamata sulla sottoscrizione di interesse. Possiamo iterare questo comportamento per tutte le variabili non più di interesse: ciò viene fatto tramite la funzione *unsubscribe* del modulo Client creato da noi, che prende in ingresso una lista di monitored item ids e chiama per ognuno di essi la funzione *unsubscribe* dello stack.

Per eliminare una sottoscrizione, basterà chiamare la funzione *delete* dello stack sulla sottoscrizione che si vuole eliminare. Tale funzione eliminerà anche tutti i monitored items presenti. È preferibile però chiamare la funzione di *unsubscribe* appena citata su tutti i monitored items prima di effettuare la cancellazione della sottoscrizione.

Altri Dettagli

I dettagli relativi alla struttura del progetto e all'avvio dell'applicativo sono contenuti all'interno del file ***README*** contenuto nel repository del progetto linkata in alto.