Relazione finale progetto Industrial Informatics

Francesco Capodicasa – Giuseppe Giuffrida

Node OPC UA – File Transfer

Il nostro progetto, disponibile su GitHub all'indirizzo https://github.com/Ciccios96/OPCUA_File_Transfer, consiste nello sviluppo di un'applicazione client-server, che comunicano attraverso il protocollo OPC UA. L'obiettivo dell'applicazione è quello di realizzare un piccolo FileSystem lato server, visualizzabile poi lato client, in cui è possibile effettuare le principali operazioni sui vari file che saranno trasferiti tra client e server (upload, lettura, scrittura, download, rinomina, spostamento, cancellazione).

I tipi di file supportati dall'applicazione sono: *"txt", "pdf", "doc", "docx", "jpg", "png", "pptx".*

Attraverso la comunicazione resa possibile dal protocollo OPC UA, client e server saranno quindi in grado di trasferire i file, tramite dei flussi bidirezionali.

Sia client che server sono stati sviluppati in Node JS, utilizzando lo stack Node OPCUA, scritto in Javascript e TypeScript, disponibile all'indirizzo https://github.com/node-opcua/node-opcua. Nel seguito verrà spiegato da cosa è composta l'applicazione e come sono stati implementati i vari componenti che ne fanno parte.

Per poter utilizzare l'applicazione correttamente, è necessario preventivamente aver installato NodeJS e il relativo pacchetto per l'installazione dei moduli, ovvero NPM (Node Package Manager).

In seguito, all'interno della directory del progetto, bisogna digitare i seguenti comandi da terminale:

- *npm init*
- *npm install node-opcua*
- *npm install node-opcua-file-transfer*
- *npm install inquirer*

• OPC UA Server

Il server, disponibile al file "*server.ts*" è stato scritto in Typescript. Dopo aver creato un'istanza del server OPC UA, impostiamo il suo URL e altre informazioni specifiche come numero di porta e resourcePath. Per quanto riguarda i meccanismi di sicurezza, quelli supportati sono "None", "Sign" con SecurityPolice "Basic128Rsa15", "Basic256" e "Basic256Sha256" e "SignAndEncrypt" con SecurityPolice "Basic128Rsa15", "Basic256" e "Basic256Sha256".

All'interno del nostro namespace, creiamo l'information model 1, con namespaceIndex = 1, aggiungiamo due cartelle "/FileSystem" e "/FileSystem/Documents" dentro la RootFolder, in cui verranno inseriti i nodi FileType da noi creati.
Queste cartelle sono dei nodi di tipo FolderType nell'address space e servono per poter realizzare il comportamento di un piccolo e classico FileSystem locale.
È stato anche aggiunto un nodo di tipo Object, chiamato *ObjectFile*, che si occupa di implementare i metodi utilizzati per la creazione e l'eliminazione dei file all'interno dell'address space.
A questo scopo, sono stati creati i metodi chiamati "*createFileTxtObject*", "*createFileObject*", "*moveFileObject", "renameFileObject"* e "*deleteFileObject*"; questi metodi saranno poi utilizzati dal client attraverso la funzione di "*call method"*.

Il server, quindi, semplicemente espone nel suo address space tutti i nodi di tipo FileType che vengono creati dal client, organizzati secondo la gerarchia di cartelle descritta precedentemente.

Per eseguire il server, basterà posizionarsi nella cartella del progetto e digitare nella console prima "*tsc server.ts*" e successivamente "*node server.js*"

Oltre che con il nostro client, il server è stato testato anche con il client Unified Automation UaExpert.

- # OPC UA Client

Anche il client, così come il server, è stato scritto in Typescript ed è disponibile al file *"client.ts"*. Si è deciso di utilizzare la libreria Inquirer, scritta in Javascript, come interfaccia grafica per il client OPC UA. Questa permette di avere un'interfaccia da linea di comando interattiva, tramite la quale è possibile mostrare all'utente diverse domande da console e in seguito l'utente potrà inserire diversi comandi da input in modo interattivo. La libreria è disponibile all'indirizzo https://github.com/SBoudrias/Inquirer.js/.

Per eseguire il client, basterà posizionarsi nella cartella del progetto e digitare nella console prima "*tsc client.ts*" e successivamente "*node client.js*".

All'avvio del client, viene richiesto di inserire l'hostname della macchina sulla quale girerà il server e l'endpoint URL di quest'ultimo, necessario per poter effettuare la connessione. Una volta inserite queste due informazioni e decise le modalità di SecurityMode e SecurityPolicy che il client vuole utilizzare per la comunicazione, all'utente viene mostrato un menu con le varie operazioni che è possibile compiere. Il menu si presenta in questo modo:

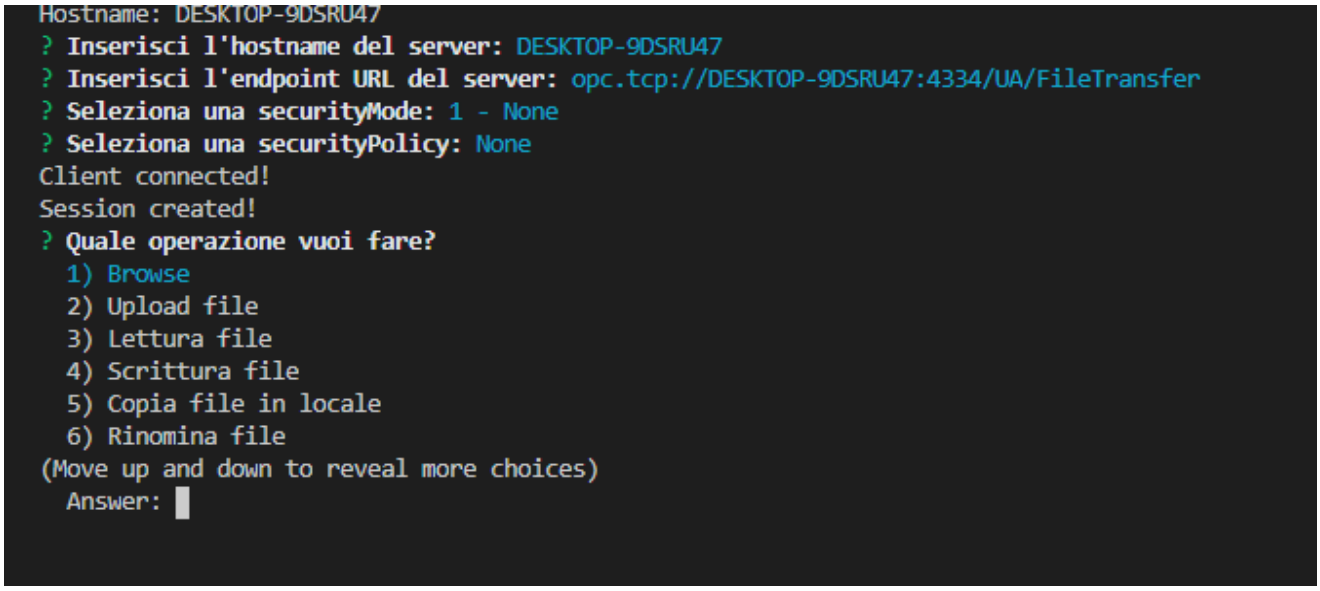

Figura 1: Visualizzazione dei comandi del client

A ogni operazione corrispondono delle particolari funzioni proprie dello standard Node OPC UA per il file transfer. Descriviamo brevemente ciascuna di esse:

- **Browse**

  Questa operazione permette al client di esplorare una parte dell'address space del server a cui è connesso, in particolare quella che simula proprio il FileSystem, ovvero poter andare a vedere il contenuto delle cartelle "/FileSystem" e "/FileSystem/Documents/", tramite le loro references, cioè dei link tra i vari nodi.

- **Upload file**

  Questa operazione permette al client di creare un nuovo nodo file, se non esiste, all'interno dell'address space del server, potendo scegliere il nome del file, se ti tratta di un documento oppure no (e in caso affermativo, il file verrà inserito nella corrispondente cartella Documents); inoltre, nel caso di un file ".txt", è possibile scegliere se andare o no a scrivere sul nuovo file creato.

- **Lettura file**

  Questa operazione permette al client di poter inserire il nome di un file che si desidera leggere. Se il file esiste, verrà mostrato il suo contenuto; altrimenti, verrà fatto comparire un messaggio di errore. Inoltre, dopo aver letto il file desiderato e aver stampato la sua

dimensione in byte, è possibile scegliere se voler effettuare o meno il download del file in locale.

- **Scrittura file**

  Questa operazione permette al client di poter inserire il nome di un file su cui si desidera scrivere. Se il file esiste, all'utente verrà chiesto se vuole sovrascrivere o meno il contenuto del file. Nel primo caso, il file verrà aperto in modalità *WriteEraseExisisting*; nel secondo caso, invece, verrà aperto in modalità *WriteAppend.* La scrittura è possibile solo su file con estensione "*.txt*".

- **Copia file in locale**

  Questa operazione permette al client di poter inserire il nome di un file, presente nell'address space del server, che si desidera scaricare in locale. Se il file esiste, verrà effettuato quindi il download all'interno del proprio filesystem locale; altrimenti, se il file non è stato trovato, verrà fatto comparire un messaggio di errore.

- **Rinomina file**

  Questa operazione permette al client di poter rinominare un file, presente nell'address space del server. Se il file non è stato trovato, verrà fatto comparire un messaggio di errore.

- **Sposta file**

  Questa operazione permette al client di poter spostare un file, presente nell'address space del server, dalla cartella FileSystem alla cartella Documents e viceversa. Se il file non è stato trovato, verrà fatto comparire un messaggio di errore.

- **Cancellazione file**

  Questa operazione permette al client di poter inserire il nome di un file che si desidera eliminare dall'address space del server. Se il file esiste, verrà cancellato nell'address space il nodo relativo a quel file; altrimenti, verrà fatto comparire un messaggio di errore.

- **Esci**

  Questa operazione permette al client di chiudere la sessione instaurata, di disconnettersi dal server e, infine, di terminare l'applicazione.