Client OPC UA in C

Luca Prezzavento

Lo scopo del progetto è realizzare un client OPC UA che dimostri l'utilizzo dei servizi di base forniti dallo standard utilizzando *open62541*, che è un'implementazione parziale dello stack protocollare OPC UA e fornisce un SDK per lo sviluppo di client e server. Allo stato attuale, solo la codifica binaria e il protocollo TCP sono supportati.

1 Installazione di open62541

1.1 Windows

Il sito open62541.org fornisce dei pacchetti precompilati in cui sono presenti le cartelle *include* e *lib* da aggiungere eventualmente alla lista dei percorsi in cui l'IDE o il compilatore deve cercare gli header e le librerie. Tuttavia, questi pacchetti sono stati compilati disattivando alcune funzionalità sperimentali, tra cui la crittografia.
In alternativa, si può compilare lo stack manualmente e attivare le funzionalità desiderate. Per fare ciò è necessario prima di tutto installare Python (almeno 2.7), CMake e Visual Studio 2015. Per attivare la crittografia è necessaria la libreria *mbedtls*, che va compilata manualmente. Bisogna dunque scaricare i sorgenti della libreria dal sito ufficiale, estrarre e configurare con CMake:

```
cmake . -DUSE_SHARED_MBEDTLS_LIBRARY=ON -D CMAKE_BUILD_TYPE=Release
        -G "Visual Studio 14 2015 Win64"
```

Questo creerà un progetto .sln da aprire con Visual Studio e compilare. Nella cartella *library/Release/lib* si troveranno le librerie compilate, mentre su *include* gli header.
In maniera analoga bisogna configurare open62541 con il seguente comando:

```
cmake . -DUA_ENABLE_ENCRYPTION=ON -DMBEDTLS_LIBRARY="<PATH>\mbedtls.lib"
        -DMBEDX509_LIBRARY="<PATH>\mbedx509.lib"
        -DMBEDCRYPTO_LIBRARY="<PATH>\mbedcrypto.lib"
        -DBUILD_SHARED_LIBS=ON -DMBEDTLS_INCLUDE_DIRS=<PATH>\include
        -DUA_ENABLE_AMALGAMATION=off  -DCMAKE_INSTALL_PREFIX=.
        -DCMAKE_BUILD_TYPE=RelWithDebInfo
        -G "Visual Studio 14 2015 Win64"
```

Dove *<PATH>* è il percorso in cui si trovano le librerie e gli header di mbedtls. Anche in questo caso verrà generato un progetto Visual Studio da compilare, scegliendo il target INSTALL. Fatto questo si avranno a disposizione le cartelle *include* e *lib* come nel caso del pacchetto precompilato.

1.2 Linux

Come nel caso precedente, sono disponibili dei pacchetti precompilati con le funzionalità sperimentali disattivate. Per poter abilitare la crittografia è necessario compilare lo stack dal sorgente utilizzando CMake e il Makefile generato da quest'ultimo. Si può usare il comando *ccmake* per visualizzare graficamente tutti i flag disponibili e attivare UA_ENABLE_ENCRYPTION.

2 Client tutorial

2.1 Compilazione

L'applicazione può essere compilata sia con supporto alla crittografia (usando il flag -DUA_ENABLE_ENCRYPTION) che senza. In quest'ultimo caso non sarà possibile caricare un certificato per il client e la scelta degli endpoint sarà limitata.
Su Windows, l'applicazione può essere compilata dal progetto Visual Studio impostando le dipendenze (open62541 e opzionalmente mbedtls), oppure utilizzando MinGW e gcc:

```
gcc main.c utils.c -I <PATH>\include -L <PATH>\lib -l open62541 -l ws2_32
    -o client_tutorial.exe
```

Dove *<PATH>* è il percorso della directory di *open62541*. Per Linux è presente un Makefile per compilare il progetto usando GCC.

Una volta ottenuto l'eseguibile, si può lanciare normalmente senza argomenti oppure specificare l'URL del server. Su Windows è necessario che il file *open62541.dll* sia nello stesso percorso dell'eseguibile.

Connessione all'endpoint

Con lo SDK utilizzato, l'apertura del SecureChannel e della Session viene effettuata tramite la funzione *UA_Client_connect*, che può essere utilizzata anche nel caso in cui non sia stato specificato un Session Endpoint. In tale situazione la chiamata richiederà automaticamente la lista degli endpoint al server e aprirà una sessione con un endpoint scelto sulla base della configurazione del client. A quest'ultima si accede usando la funzione *UA_Client_getConfig*, che ritorna un puntatore con il quale è possibile sia leggere che modificare la configurazione. In particolare, le funzioni *UA_ClientConfig_setDefault* e *UA_ClientConfig_setDefaultEncryption* permettono di inizializzare tale configurazione con dei valori predefiniti, eventualmente specificando il certificato e la chiave privata se si utilizza la seconda funzione.
Nell'applicazione sviluppata è stata inserita la possibilità di fare manualmente il discovery degli endpoint e specificare quello verso cui aprire la sessione. Ciò viene fatto utilizzando *UA_Client_getEndpoints* e copiando la struttura *UA_EndpointDescription* relativa all'endpoint scelto nella configurazione del client. Facendo ciò verrà disattivata anche la scelta automatica della UserTokenPolicy, che andrà dunque selezionata manualmente e copiata nella configurazione.

Servizi

I servizi utilizzati nell'applicazione sono:

Browse È gestito dalla chiamata *UA_Client_Service_browse*, che accetta una struttura *UA_BrowseRequest* con la quale è possibile scegliere il nodo radice e specificare un filtro per la Node Class.

Read Gestito da *UA_Client_Service_read*. In questo caso la richiesta è rappresentata dal tipo *UA_ReadRequest* nel quale si specificano, tra le altre cose, i nodi da leggere. Nell'applicazione questo servizio viene usato per leggere i valori di va-

riabili a scelta e il namespace array. Ciò viene fatto specificando come attributo UA_ATTRIBUTEID_VALUE.

CreateSubscription Gestito da *UA_Client_Subscriptions_create*. In questo caso si specificano i parametri della sottoscrizione (Publishing Interval, Lifetime Count, ecc.).

CreateMonitoredItem Si utilizza una chiamata diversa in base al tipo di Item che si vuole monitorare. In questo caso è stata utilizzata *UA_Client_MonitoredItems_createDataChang* che permette di specificare un filtro e dei callback per gestire le notification e la deallocazione delle risorse. Nell'applicazione sviluppata il primo callback viene utilizzato per mostrare il valore della variabile ogni volta che si riceve una notification, mentre il secondo viene utilizzato per deallocare la struttura *UA_DataChangeFilter* eventualmente allocata al momento della creazione del Monitored Item nel caso in cui sia stato specificato un filtro.

Publish Request e Response

L'invio delle Publish Request e la ricezione delle Publish Response e dei Notification Message è gestito dalla funzione *UA_Client_run_iterate*, che deve essere chiamata ripetutamente per mantenere attive le sottoscrizioni. Se l'applicazione viene compilata con il flag PTHREAD, viene utilizzata la libreria *pthread* per permettere all'utente di continuare a utilizzare i comandi mentre le sottoscrizioni vengono gestite in background. Le funzioni fornite da open62541 non sono normalmente thread-safe, per cui è stato aggiunto un mutex per limitare l'accesso alla struttura UA_Client. Il mutex non è più necessario se si utilizza uno stack compilato manualmente con il flag UA_MULTITHREADING. In ogni caso, il client tutorial può essere compilato senza multithreading, nel qual caso viene aggiunto un comando che inizia un ciclo in cui viene chiamata *UA_Client_run_iterate* per il resto dell'esecuzione del programma.