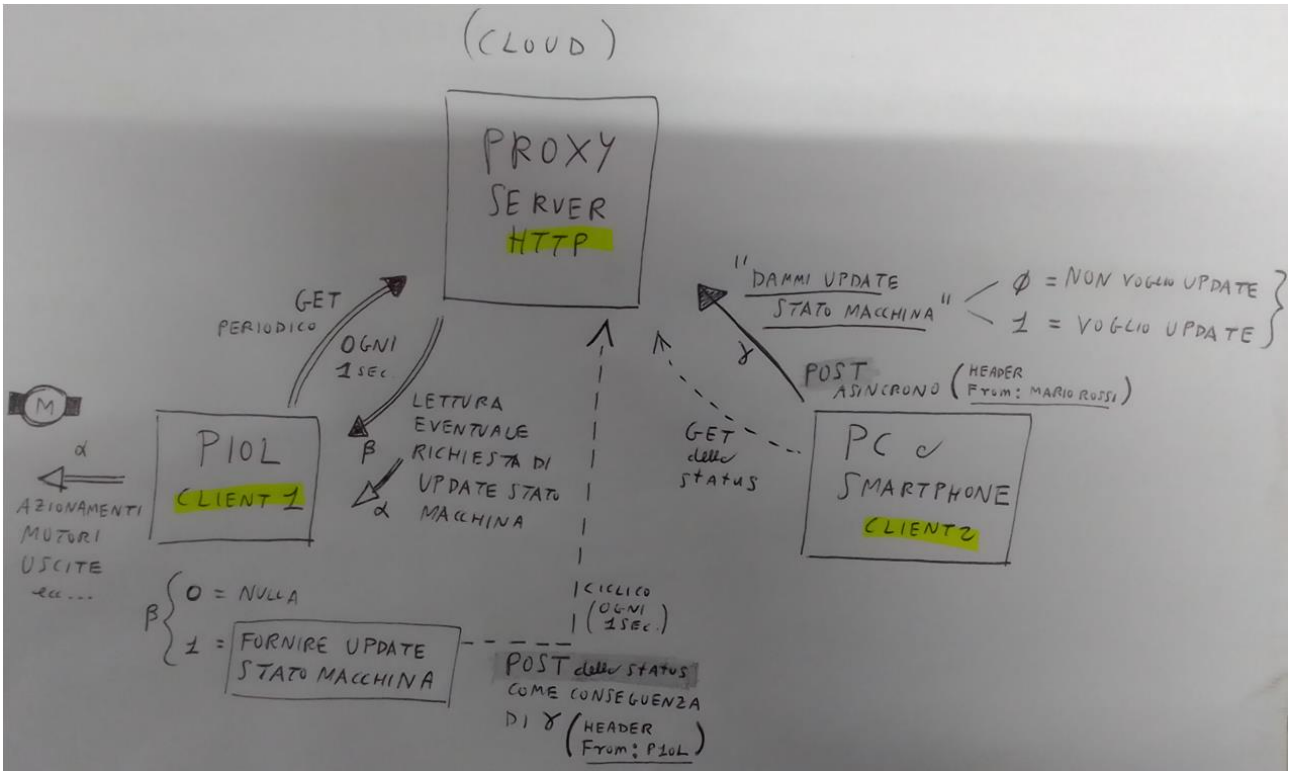
Breve nota generale sulla teleassistenza nell'ecosistema Syel + Codesys

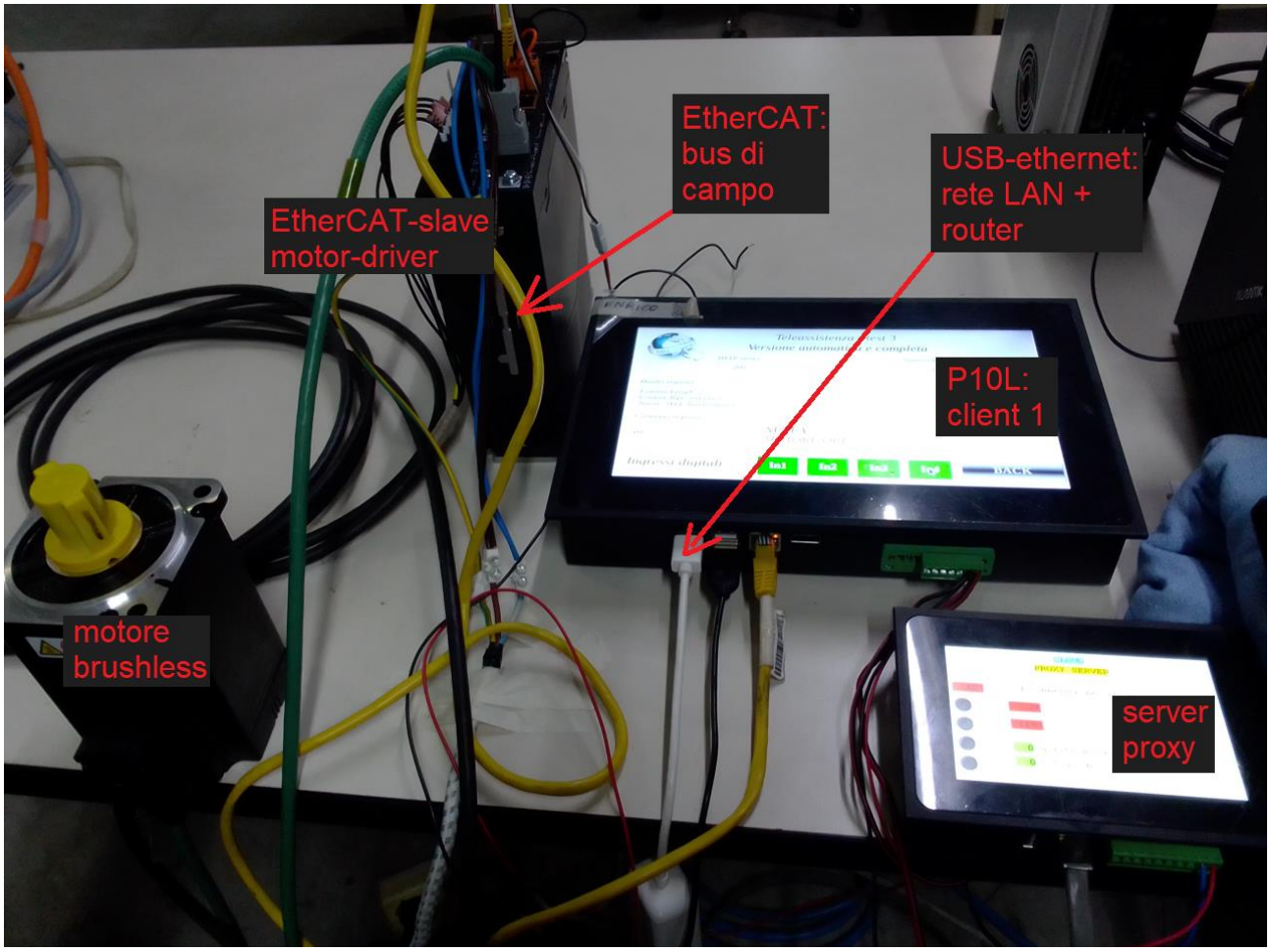
16-12-2022

Draft, version 1.0

Abbiamo la possibilità di implementare una forma di teleassistenza che si sviluppi via protocollo HTTP1.1, che prevede il seguente schema a blocchi di massima:



Supponiamo di avere un nostro HMI P10L che controlla un bus di campo, ad esempio un CANbus, oppure, come nel nostro esempio particolare (vedasi fotto sottostante), un bus di campo EtherCAT (nell'esempio sviluppato da noi, P10L-master è collegato ad un EtherCAT-slave, nella fattispecie un driver per motore brushless):

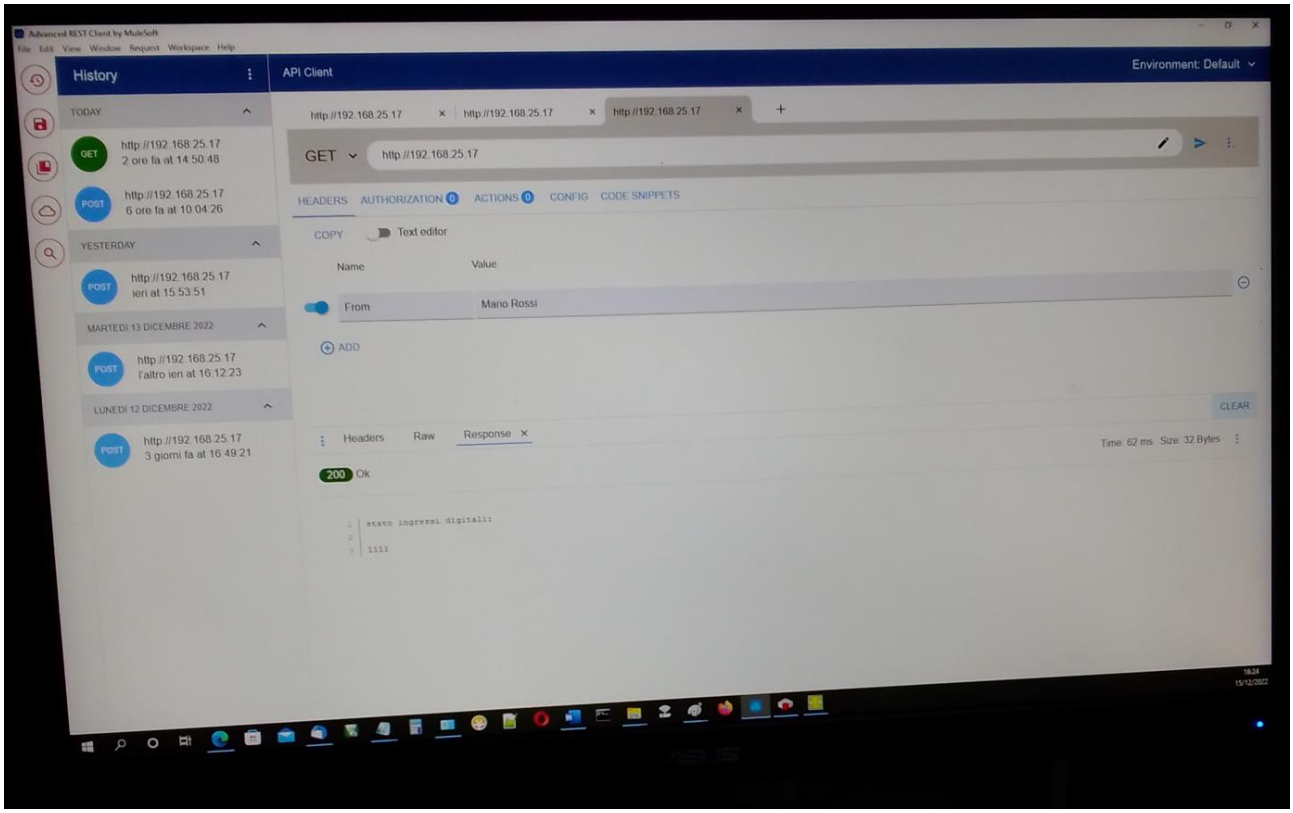


P10L è collegato tramite cavo ethernet giallo allo slave EtherCAT, quindi il cavetto giallo costituisce il bus di campo, mentre tramite adattatore USB-ethernet il P10L è collegato alla LAN, nella quale sono stati inseriti altri due noti.

Su P10L gira un SW che implementa sia un EtherCAT-master, per il controllo del bus di campo, quindi per il pilotaggio del motore, e lo stesso SW realizza anche un HTTP-client (client 1).

Il primo nodo aggiuntivo è il server-proxy, ovvero un server HTTP a tutti gli effetti.

Il secondo nodo aggiuntivo è un PC-Windows, sul quale gira appunto un HTTP-client (client 2, foto sottostante).



Il concetto alla base dell'esperimento/esempio condotto da noi è il seguente:

PC-Windows (HTTP-client 2) e P10L (HTTP-client 1) non colloquiano direttamente fra di loro, bensì ciascun client intrattiene comunicazione, tramite i metodi GET e POST, con un server centrale, inserito per comodità nella stessa rete LAN dei 2 clients, che nel nostro esempio è un HMI by Syel, un pannello piccolo a 5 pollici, che realizza appunto un HTTP-server, ma tale server può essere implementato anche su un compute remoto, quindi esterno alla LAN, ad esempio un web-server/cloud (concettualmente nulla cambia).

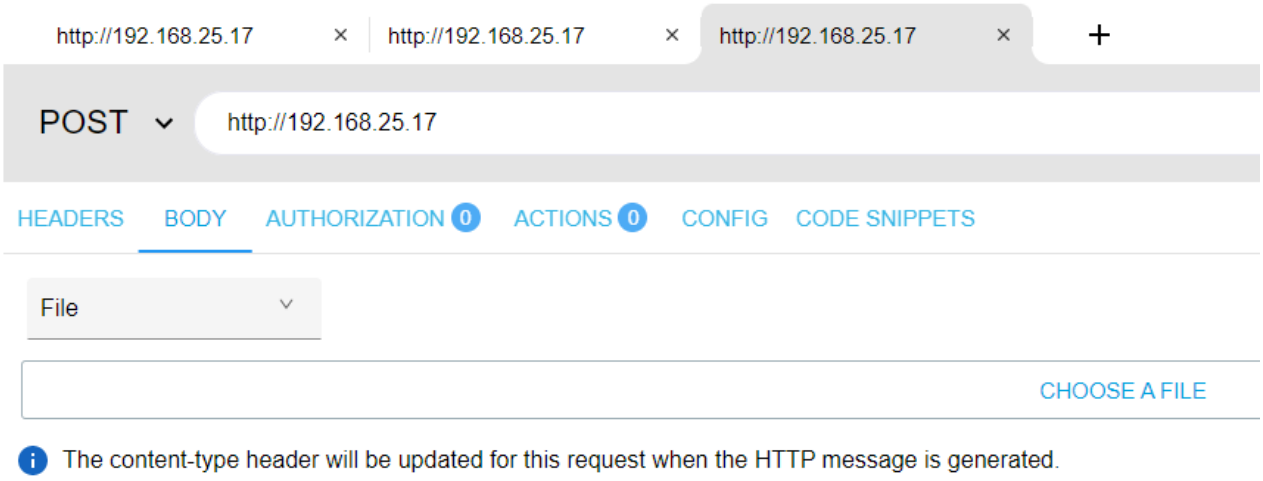
P10L, ogni 1 secondo, invia una GET al server, per ottenere da questo una stringa, la quale contiene informazioni circa due aspetti:

- *Azioni da mettere in atto sul bus di campo, ad esempio far girare oppure stoppare il motore;*
- *Richiesta, da parte di client 2, di ottenere in lettura lo stato della macchina, ovvero lo stato attuale del bus di campo, ad esempio la posizione corrente dell'asse-motore, oppure il valore logico degli ingressi digitali, ecc …*

Quindi P10L – client 1, a polling, ogni 1 secondo, invia la GET al server "dimmi cosa vuole adesso client 2", ed il server risponde.

Client 2, quando vuole, invia una POST al server "fai fare al P10L queste 2 cose: refresh dello stato corrente del bus di campo ON/OFF – motore ON/OFF".

Ecco la POST inviata dal client 2 (un PC nel nostro caso/esempio, ma potrebbe essere lo smartphone del cliente):



- Refresh dello stato corrente del bus di campo ON/OFF:

il primo byte informativo che il server manda al P10L in risposta:

0 = NULLA
1 = INVIARE STATUS MACCHINA

Appena P10L riceve quel primo byte (il primo dei due), P10L si comporta congruentemente, ovvero:

0 = NULLA:

P10L non invia nessuna POST al server contenente lo stato corrente del bus di campo, dato che client 2 non è interessato ad avere in risposta l'aggiornamento informativo sullo stato della macchina.

1 = INVIARE STATUS MACCHINA:

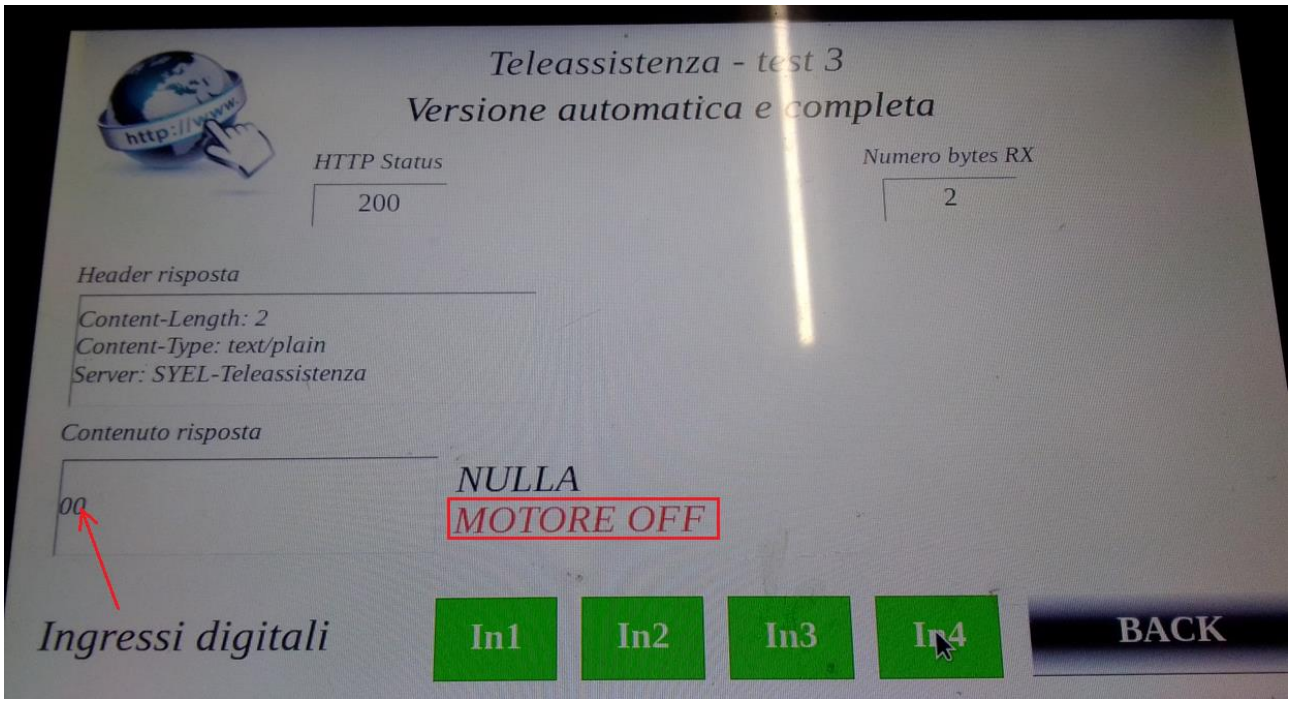
P10L invia ogni 1 secondo una POST al server (appena dopo la GET) contenente lo stato corrente del bus di campo, dato che client 2 è interessato ad avere in risposta l'aggiornamento informativo sullo stato della macchina.

Nel nostro esempio SW, P10L invia al server 4 bytes informativi, uno per ciascun ingresso digitale:

Nel caso della foto, tutti e 4 gli ingressi sono accesi (in una situazione reale, gli ingressi digitali in questione potrebbero rappresentare, ad esempio, degli allarmi legati al motore/al driver sul bus di campo).

- Motore ON/OFF:

il secondo (ed ultimo, nel nostro semplice esempio) byte informativo che il server manda al P10L in risposta:



0 = motore off
1 = motore on

P10L spegne o accende il motore.

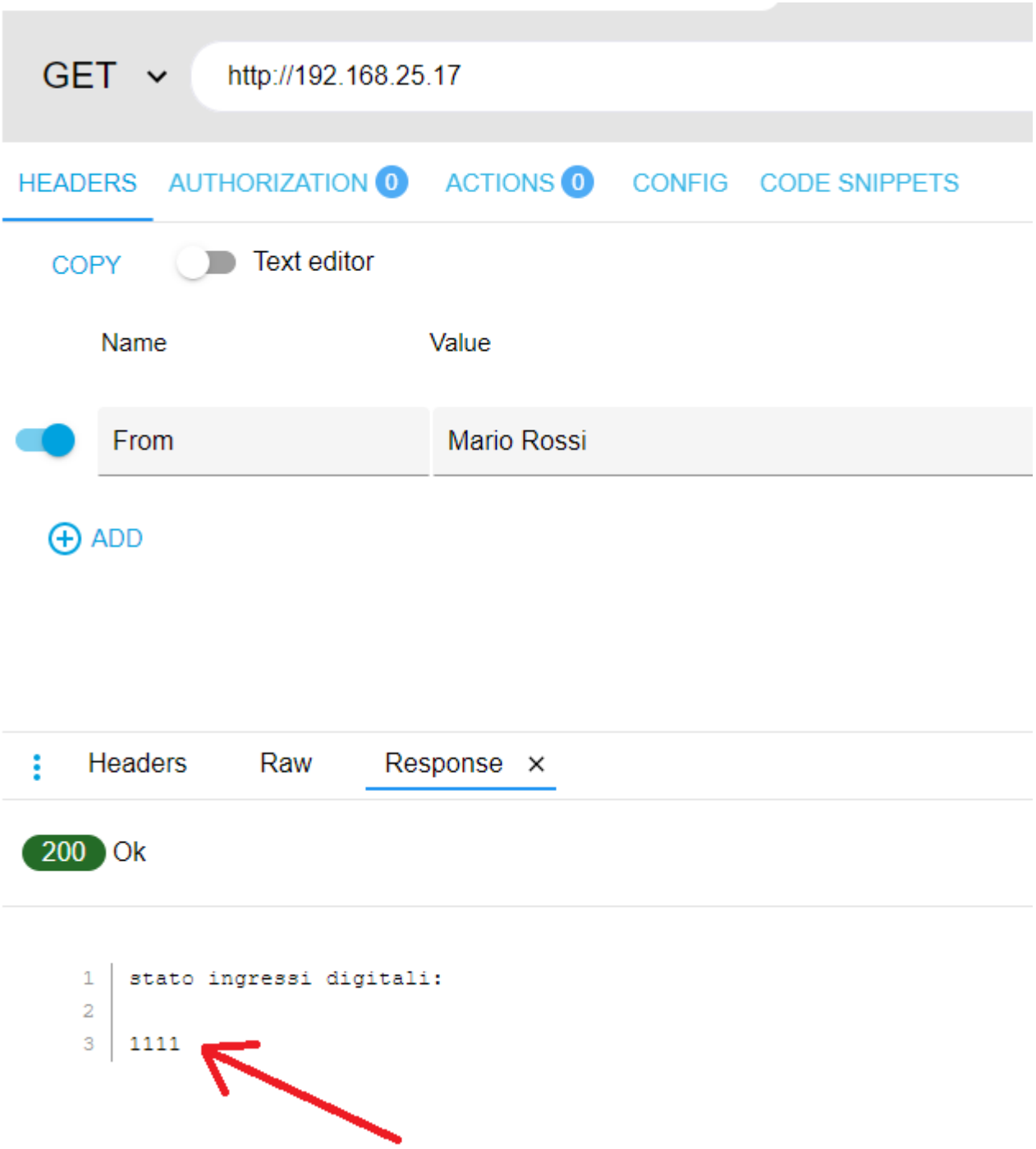
Supponiamo adesso che, dopo una POST inviata da client 2 al server, in cui tale client richiede di ottenere l'aggiornamento dello stato corrente della macchina, client 1, ovvero P10L, ha risposto al server con una POST (in realtà, una successione di POST, una al secondo) contenente lo stato logico corrente degli ingressi digitali (es: allarme-motore-1, 2 .. 4).

Il server, pertanto, ha raccolto questa informazione:

1111

In quanto gli ingressi sono tutti attivati.

A questo punto se il client 2 invia al server una GET “inviami il messaggio inviato dal P10L”, ecco cosa riceve il client 2:

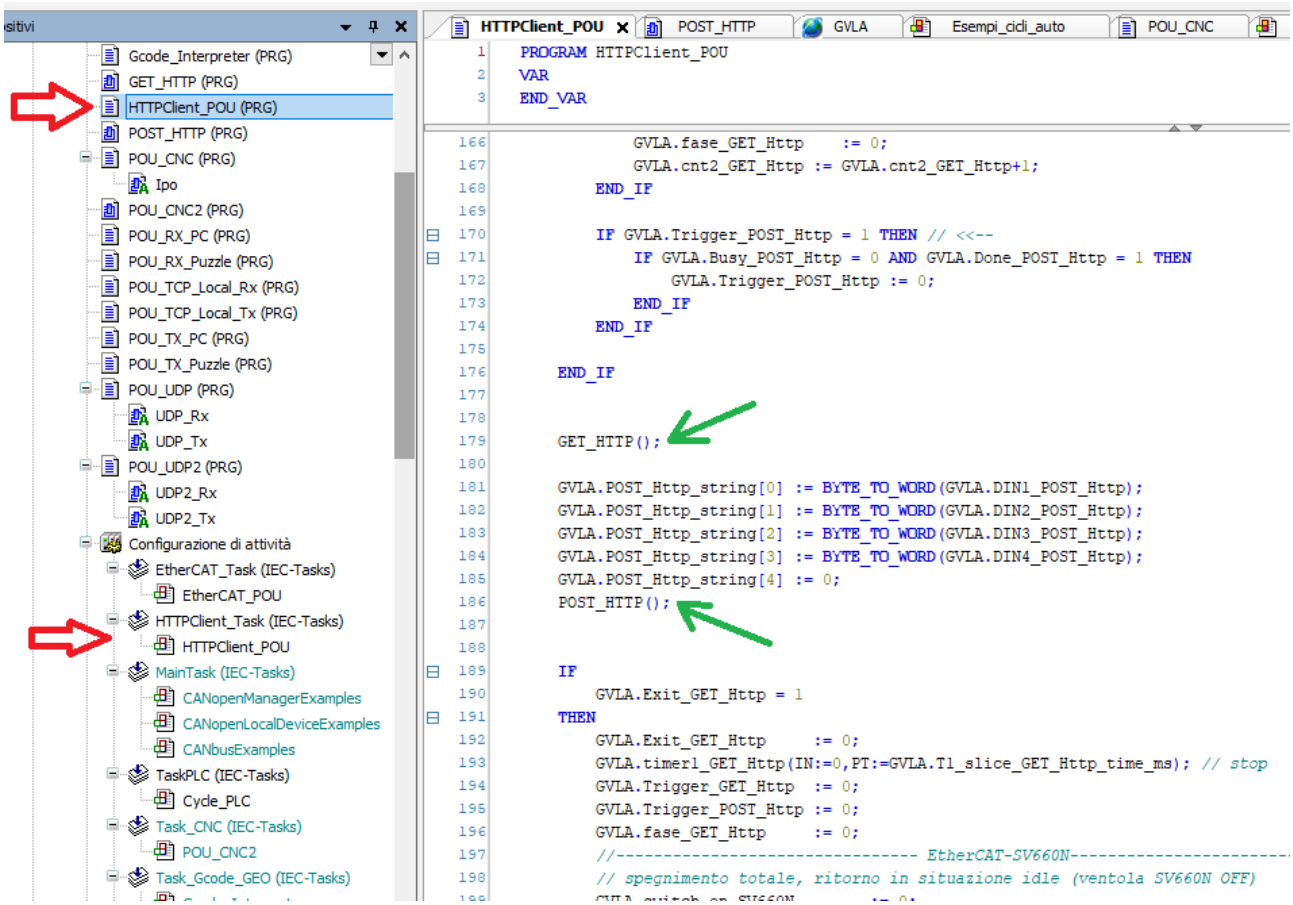


Il server riceve quindi:

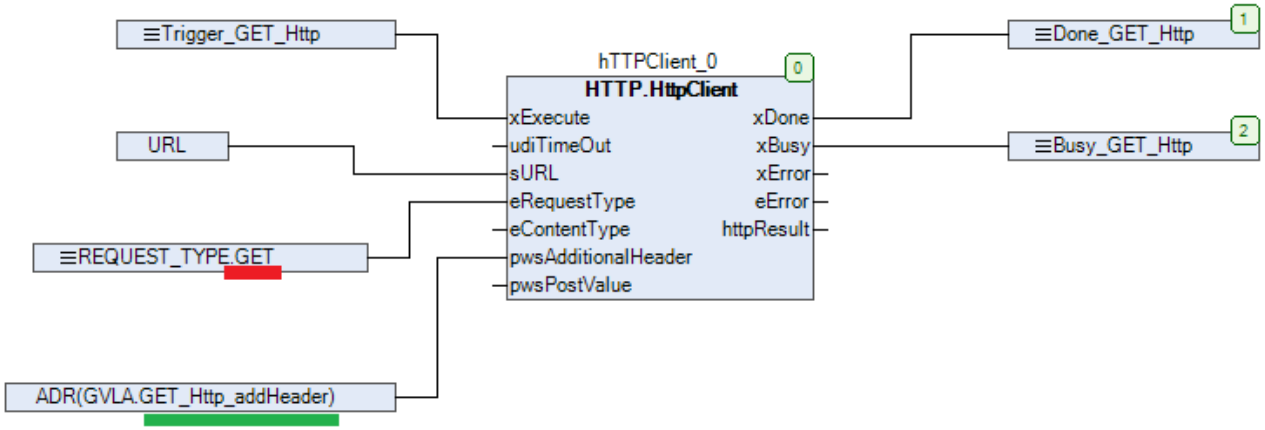
- Una GET dal P10L – client 1
- Una GET dal client 2
- Una POST dal P10L – client 1
- Una POST dal client 2

In questo nostro esempio, il server legge l'header "From: " di ogni metodo GET e POST arrivatogli, al fine di capire se la GET corrente, appena arrivata, è stata inviata dal P10L – client 1 oppure dal client 2/PC/smartphone, e lo stesso per la POST.

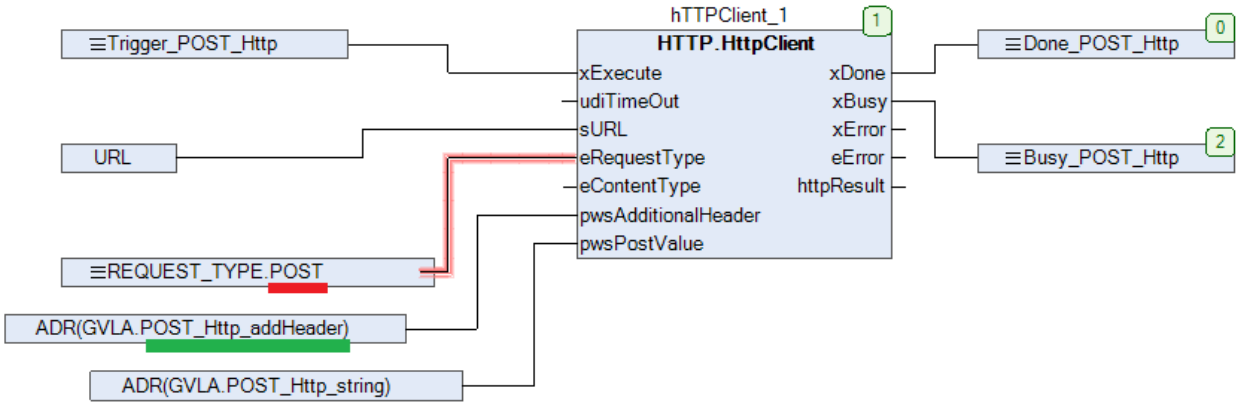
Nel nostro SW, sviluppato in Codesys, abbiamo un task, al quale è associata una POU scritta in linguaggio ST, dentro il cui codice abbiamo la chiamata a due ulteriori POU, disegnate però in linguaggio CFC:



GET_HTTP():



POST_HTTP():



Il blocchetto HTTP.HttpClient è la libreria fornita dalla 3S che implementa l'intelligenza http.

Per entrambi gli schemi CFC, abbiamo valido che:

POST_Http_addHeader : WSTRING(256) := "From: P10L";
GET_Http_addHeader : WSTRING(256) := "From: P10L";

che è l'header grazie al quale il server distingue chi gli ha inviato una GET oppure una POST, e questa distinzione serve al server per poter mandare al nodo giusto il messaggio giusto.

P10L invia l'header "From: P10L" sia nella POST che nella GET HTTP1.1, mentre client 2 invia l'header "Mario Rossi" sia nella POST che nella GET HTTP1.1.

In una prossima versione di esempio di teleassistenza, il server sarà un web-server/cloud delocalizzato nel mondo, non interno alla nostra LAN, ed utilizzeremo l'end-point nell'URL al posto dell'header "From: ", al fine di far distinguere al server il nodo che ha appena inviato il metodo GET o POST, sulla base dell'end-point, non più sulla base dell'header.

Ad esempio:

POST /cmd.html HTTP1.1

l'end-point è in questo caso "/cmd.html", ovvero la risorsa/il file da submittare sul server è "cmd.html".