Sistema di telegestione di un generico impianto domotico via web.

Modello di un generico impianto domotico

Ispirandosi a uno dei maggiori standard in ambito domotico, Konnex, è possibile determinare le caratteristiche fondamentali che deve avere un impianto.

Come anche in altri standard, un impiato è basato su un **bus** al quale sono collegati tutti i dispositivi che si differenziano tra **attuatori** e **sensori**.

In generale possiamo dire che l'impianto non funziona secondo un'architettura master/slave, bensì peer to peer, ovvero la logica computazionale dell'impianto è distribuita su tutti i dispositivi, ognuno dei quali decide quando scrivere o leggere sul bus, il protocollo prevederà quindi la gestione delle collisioni dei pacchetti.

L'architettura sarà fortemente Event-Based, quando un sensore, per esempio, rileverà un cambiamento della grandezza che deve misurare, immetterà nel bus un pacchetto contenente il nuovo valore della grandezza.

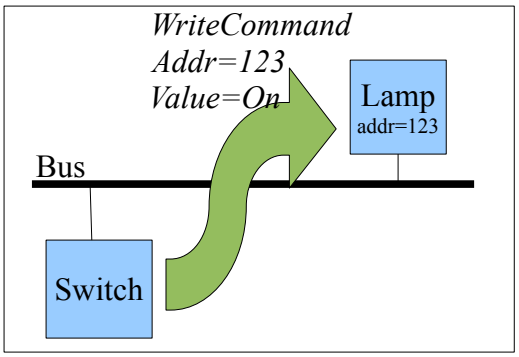
Allo stesso modo un dispositivo può dire ad un attuatore di cambiare stato mandandogli un pacchetto.

Semplificando e generalizzando al massimo, sono stati individuati 3 tipi fondamentali di pacchetti:

- **Write Command**: viene inviato ad un dispositivo quando si vuole cambiarne lo stato
- **Read Request**: viene inviato ad un dispositivo per interrogarlo sul valore che memorizza
- **Read Responce**: il dispositivo in questione risponde al comando precedente inviando il valore da leggere su questo pacchetto.

Da notare che un sensore reagirà solo ai pacchetti Read Request, essendo in sola lettura, mentre invece un attuatore reagisce sia ai Write Command che ai Read Responce essendo disponibile in lettura e scrittura.

La figura a lato mostra un esempio pratico, se si volesse accendere la luce che ha come indirizzo "123" l'interruttore invierebbe sul bus un comando Write contenente l'indirizzo della lampada e lo stato che dovrà assumere, ovvero "Acceso".



Si può immaginare che l'interruttore piuttosto di inviare il comando acceso/spento invii il comando "Toggle" che invertirà lo stato della lampada a prescindere da quale esso fosse l'istante prima.

In genere lo standard domotico definisce anche dei tipi di dati, in modo che chiunque legga i valori all'interno dei dispositivi sappia come interpretarli a prescindere da chi l'ha costruito o dai loro criteri di funzionamento.

I Datatypes spesso non si limitano solo a definire il formato binario dei dati (integer, float...) ma anche il significato fisico ovvero Temperatura, Potenza, Stato di una Lampada.

Quando viene fatta una Read Request è opportuno che il dispositivo risponda comunicando tutte queste informazioni.

Interfacciamento con l'esterno

Dato che si vuole poter gestire l'impianto dal Web è necessario definire come avviene questo interfacciamento.

Si è già detto che per inviare comandi a qualsiasi dispositivo è necessario solamente immettere nel bus dei pacchetti ed essere in grado di leggerne.

Un dispositivo farà da ponte tra il Bus dell'impianto e Internet, questo dispositivo verrà chiamato "Bridge Device".

Il Bridge Device essendo connesso al bus, è in grado di intercettare tutti i pacchetti che vengono spediti attraverso di esso, se un'interruttore comandasse a una lampada di accendersi, ne verrebbe a conoscenza.

Ipotizzando che il Bridge sappia già a priori il numero di dispositivi presenti nell'impianto, ipotesi realistica dato che in fase di configurazione dell'impianto questo numero è noto e quindi può essere programmato all'interno del bridge, potrebbe succedere che esso non conosca lo stato di alcuni dei dispositivi. In fase di inizializzazione quindi eseguirà delle Read Request su tutti i dispositivi e dopo essersi costruito in memoria un'immagine dell'impianto, resterà in ascolto dei soli Write Command.

> ***Il prototipo...***
>
> *Per quanto riguarda il prototipo, l'impianto è interamente simulato da un software scritto in C#, e simula in modo statico 3 lampade, 1 lampada per la quale è possibile regolare la luminosità, 1 tapparella di una finestra per la quale è possibile regolare il grado di apertura, e 1 sensore di temperatura.*
> *Tramite l'interfaccia grafica è possibile modificare lo stato di ogni dispositivo. Il bus è implementato attraverso una porta USB, quindi il software su evento invierà un opportuno comando Write che verrà intercettato dal bridge, inoltre risponderà anche alle Read Request.*
> *Il Bridge Device è realizzato con un sistema a microprocessore basato sulla piattaforma Open Source Arduino, dotato nella scheda di base di uscite digitali (per accendere dei led che simulano le lampade), porta USB connessa alla periferica UART, mentre la scheda di espansione è dotata di interfaccia Ethernet, utilizzabile grazie a librerie Open Source.*
> *Il microprocessore (Atmel Atmega328, 32KB Flash Memory, 2KB SRAM, 16MHz), è programmato in modo da inviare sulla connessione seriale le read request, una per ogni dispositivo che sa essere attivo nell'impianto, terminata la fase di inizializzazione resta in ascolto unicamente dei Write Command. In memoria tiene traccia dello stato corrente dell'impianto con un array di Stringhe (String DevList[6]) Nel quale per ogni dispositivo è associato una stringa nella quale compaiono tutti i dati utili separati dal carattere ';' i campi memorizzati sono il Valore corrente, il Nome arbitrario del dispositivo (esempio LampadaCucina), il DataType associato al valore, il carattere A se è un attuatore, S se è un sensore. L'indirizzo del dispositivo, d'ora in poi chiamato Codice o Code, coincide con l'indice nell'array.*
> *L'informazione "Attuatore/Sensore" è ridondante in quanto desumibile dal DataType che in qualche modo definisce anche questo , tuttavia è comoda per alcune computazioni relative alla parte Web.*
> *Alcuni LED presenti su una board di espansione vengono accesi e spenti in base allo stato delle lampade presenti nell'impianto demo in modo da mostrare l'istante esatto in cui viene aggiornato DevList[].*
> *Da notare che l'immagine in memoria dell'impianto è assolutamente inutile ai fini del funzionamento di sistema, il bridge deve solo inoltrare da una parte all'altra i pacchetti, è stato realizzato in questo modo solo e unicamente per poter creare il file status.xml e per l'accensione dei led a solo scopo di debug.*
> *I pacchetti sono implementati sottoforma di stringhe la cui sintassi è:*

- *Write Command: "W;[Codice];[Valore]"*
- *Read Request: "R;[Codice]"*
- *Read Reply: "RR;[Codice];[Valore];[Nome];[DataType];[A|S]"*

Altre funzionalità aggiuntive (parte Ethernet) verranno descritte in seguito.
Per quanto riguarda la tabella dei DataType si veda l'appendice.

Funzionamento Generale

Il bridge in sostanza dovrà fornire al sito le informazioni necessarie sull'impianto, e dovrà inoltrare al bus i comandi provenienti dall'esterno.

Sapendo questo, sono stati pensati 2 possibili comportamenti del sistema.

La prima idea è stata di fare in modo che il sito quando necessario richiedesse al bridge una risorsa (status.xml) contenente lo stato attuale dell'impianto. Questo file viene generato al momento della richiesta a partire da un'immagine dell'impianto (DevList[]) che il bridge costruisce in memoria man mano che intercetta i pacchetti sul bus. Il sito dovrà quindi richiedere periodicamente questo file per poter visualizzare dinamicamente l'impianto.

I comandi dall'esterno invece verrebbero inviati sottoforma di request da parte del sito di una pagina (dataIn) che estrapolando i parametri passati aggiorna l'immagine interna e inoltra il pacchetto sul bus.

La seconda idea prevede un funzionamento del bridge il più minimale possibile, ovvero l'inoltro semplice dei comandi da una parte all'altra. I comandi dall'interno vengono inoltrati mediante la richiesta di una pagina sul sito, che andrà ad aggiornare l'immagine dell'impianto questa volta sottoforma di tabella in un database interno al sito stesso. I comandi dall'esterno verranno trattati nello stesso modo.

Tralasciando il caso dei comandi dall'esterno dato che è costante, si può studiare come variano la complessità computazionale e il traffico nelle due ipotesi.

Nel primo caso viene eseguita un'unica grossa richiesta con una frequenza pari a quella di refresh della pagina di telegestione (nel prototipo 1 ogni 10 secondi), nel secondo caso ci sono più richieste indipendenti con frequenza che dipende dai dispositivi presenti nell'impianto, stimabile grossomodo in una richiesta ogni 5 secondi ovvero pari al traffico presente sul bus dell'impianto. Il traffico è più consistente nel secondo caso, per il bridge è facilmente gestibile in quanto non riceve ulteriori richieste, il server invece dovrà gestire un numero di richieste pari a quelle del bridge moltiplicato per il numero di bridge presenti nel sistema.

Per quanto riguarda la scalabilità l'ipotesi 2 è peggiore in quanto il numero massimo di bridge collegabili ad un unico server è minore, tuttavia il numero di richieste da gestire è assolutamente accettabile per un qualsiasi server web, in quanto non è molto diverso da quelle generate dalla navigazione normale degli utenti mediante browser, e quindi non è necessario prendere misure particolari per garantire una sufficiente scalabilità.

Per quanto riguarda la complessità computazionale, nel primo caso il bridge dovrà costruire l'immagine dell'impianto e creare il file status.xml, nel caso si vogliano implementare anche la gestione di allarmi sarà sempre compito del bridge verificare se è il caso di inviarne al sito, dato che gli allarmi probabilmente verranno creati ad hoc dall'utente sempre tramite il sito, la gestione di tutto questo diventa molto ardua, e anche se è possibile realizzarla su dispositivi embedded complessi, è praticamente impossibile farlo usando dispositivi economici e minimali, come la scheda arduino usata nel prototipo che dispone solamente di 2KB di RAM (quasi già del tutto utilizzata senza questa feature). Il server invece ha un'attività molto semplice dato che si limita a richiedere a polling il file status.xml e visualizzarne il contenuto in una pagina di gestione.

Nel secondo caso il bridge ha un comportamento minimale, in quanto si limita ad inoltrare i pacchetti senza nessuna ulteriore verifica. Sarà compito del server capire quando è il caso di generare allarmi, cosa che è assolutamente in grado di fare con estrema facilità in quanto dispone di risorse hardware molto più consistenti.

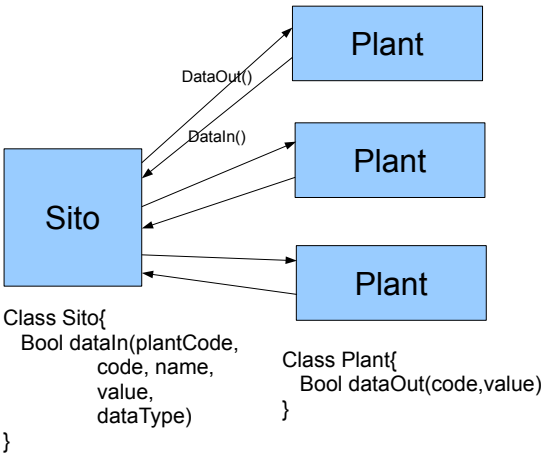
È stata adottata la seconda ipotesi in quanto più praticabile per la realizzazione del prototipo e in quanto permette l'utilizzo di schede economiche che abbattono significativamente i costi di upgrade dell'impianto.

Il prototipo...

Per la parte Ethernet è stata usata un'espansione (Shield) della scheda Arduino Uno: la Ethernet Shield dotata del chip Wiznet W5100 che permette di gestire fino a 4 connessioni contemporanee. Inoltre è stata usata la libreria OpenSource Ethernet che implementa già al suo interno gli oggetti socket Server e Client.
Il firmware inizializza un oggetto Socket Server in ascolto sulla porta 80, quindi una volta rilevato un dato in ingresso viene analizzato per verificare se si tratta di una request Http, in questo caso provvede ad isolare la riga contenente il comando GET che contiene il nome della pagina richiesta e i parametri passati col metodo GET. Una volta estrapolati tutti questi dati vengono chiamate le "pagine" che consistono in funzioni alle quali si passa in ingresso un array contenente tutti i parametri di tipo GET rilevati, queste funzioni dopo aver eseguito la computazione che gli spetta (modificare DevList[] e scrivere sulla porta seriale) rispondono al client inviando la pagina HTML che ne risulterà che consisterà semplicemente in una stringa "OK" oppure "ERROR" per verificare la riuscita dell'operazione.
Allo stesso modo quando si vuole contattare il sito, verrà istanziato un oggetto client che effettuerà una richiesta http al server web. I parametri passati tramite metodo GET verranno poi usati dal sito per aggiornare il database interno descritto meglio in seguito.
In fase di debugging si è notato che il motivo principale dei malfunzionamenti era il riempimento della RAM del microprocessore dovuto all'istanziamento di tutte le stringhe usate in vari punti del firmware che poi non venivano deallocate data la mancanza di un garbage collector. È stato deciso mediante direttive al compilatore di istanziare tutte le stringhe nella memoria Flash di ben 32KB.
Si è voluto lasciare comunque la possibilità di contattare il bridge direttamente tramite browser per visualizzare il file status.xml, accompagnato da una pagina css per la rappresetnazione grafica.

Architettura di sistema

Si è pensato di strutturare il progetto secondo uno stile architetturale **Object-Based**.

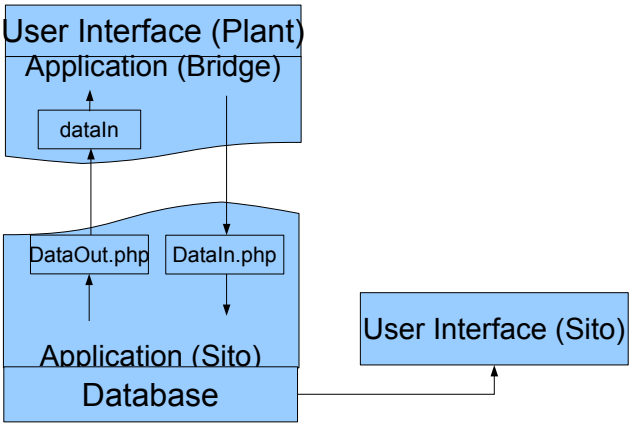


L'oggetto per così dire "master" è il sito stesso, dopodichè i vari impianti possono essere visti come una collezione di oggetti "Impianto".

L'invio di comandi nelle due direzioni avviene secondo un meccanismo di **Remote Procedure Calls**, ovvero l'interfaccia del generico oggetto impianto offrirà il metodo "dataOut(codice,valore)" che prende in ingresso i parametri necessari per fare sì che il bridge possa generare il pacchetto di tipo Write Command corrispondente.
L'oggetto "Sito" invece offrirà il metodo "dataIn(plantCode,codice,nome,valore,dataType)", dove i parametri permettono di unificare i pacchetti Write Command con quelli Read Responce, che entrambi devono essere inoltrati al sito,

plantCode serve per far capire al sito chi è il mittente di questa procedura. È stato volutamente omesso il campo A\S dato che è in realtà un'informazione ridondante e deducibile dal dataType come già detto.

L'architettura software è **two-tiered** e prevede quindi che i meccanismi di RPC vengano forniti in parte dal sito e in parte dal bridge, in particolare se il bridge vuole inoltrare un pacchetto al sito richiederà una specifica pagina, mentre il sito per inoltrare un pacchetto al bridge richiederà una pagina sul sito stesso che a sua volta effettuerà una request direttamente al bridge, il meccanismo sarà descritto meglio in seguito.



Si potrebbe pensare ad un altro stile architetturale che rispecchia ugualmente l'implementazione effettuata, ovvero lo stile **Event-Based**.

Il componente "Sito" e i componenti "Impianto" comunicano tra di loro attraverso un Event-Bus, realizzato in modo analogo a quanto già descritto dalle pagine apposite presenti sul sito e la pagina sul bridge che viene richiesta dal sito.

I componenti reagiranno ugualmente ai pacchetti che vengono inoltrati nelle due direzioni, con l'unica differenza che ora questi pacchetti assumono il significato di eventi, come per esempio "accensione di una luce" o "cambiamento di temperatura". Gli eventi così trasmessi incorporeranno per ragioni ovvie anche il dato corrispondente.

Il prototipo...

Il sito è stato sviluppato interamente in php + AJAX su un server apache utilizzando per i database mysql.

All'interno del sito sono presenti 3 tabelle: "utenti" contiene i dati relativi agli utenti, ovvero password (cifrate con md5) email, e l'indirizzo IP del proprio Bridge.

Una pagina provvede ad eseguire la registrazione al sito verificando lato client la validità di tutti i campi del form prima dell'invio.

Nella barra laterale è possibile fare il login, che se effettuato imposta una variabile SESSION che conterrà l'username dell'utente. Tutte le altre pagine verificheranno se questa variabile è settata prima di visualizzare qualsiasi informazione riservata.

Nel caso sia stato effettuato il login nella barra laterale comparirà la voce del menù per accedere al pannello di controllo dell'impianto.

Le altre due tabelle servono per la gestione dei dati degli impianti, "AddressCodeAssociation" contiene le associazioni tra l'indirizzo IP del bridge e il codice identificativo dell'impianto, permette dato un utente di risalire alle entry dell'ultima tabella relative al proprio impianto, inoltre funge ancha da meccanismo per verificare se il bridge è mai stato connesso in rete e inizializzato.

L'ultima tabella "plantData" contiene un'entry per ogni dispositivo presente in qualunque impianto, ogni entry contiene anche il codice dell'impianto di appartenenza in modo che visualizzando il pannello di controllo di un utente venga fatta una query solo su quelle entry.

La pagina di telegestione esegue semplicemente una query sulla tabella plantData, e per ogni entry controlla il dataType corrispondente, e in base a quello genere un'interfaccia grafica specifica, per una luce visualizzerà un'icona di una lampada e un bottone per inviare il comando di accensione o spegnimento, per un sensore di temperatura visualizzerà

semplicemente il valore eccetera.

La pagina viene aggiornata ogni 10 secondi mediante i meccanismi asincroni di AJAX e quindi controllerà lo stato dell'impianto a polling.

Per quanto riguarda l'inserimento dei dati sull'impianto si rimanda al capitolo successivo sulla comunicazione.

Comunicazione

La comunicazione tra i componenti di sistema, come già detto, avviene mediante **RPC**, implementata tramite 3 pagine dinamiche che si passano i parametri del caso.

Le RPC impongono un primo vincolo sull'accoppiamento temporale, ovvero perchè la chiamata funzioni entrambi i componenti (chiamante e chiamato) siano attivi in quel momento. Questo è un vincolo accettabile, in quanto non avrebbe senso inviare comandi dall'esterno al bridge se questo è scollegato, anzi si vorrebbe poter visualizzare un messaggio di errore nel caso succeda!

È sconsigliabile quindi implementare un meccanismo Message-Oriented con comunicazione persistente, in quanto se il bridge fosse scollegato, un eventuale comando rimarrebbe sospeso in memoria finchè non venisse ricollegato, e a questo punto verrebbe eseguito anche se magari è trascorso un lasso di tempo significativo dal momento in cui è stato lanciato.

L'implementazione delle RPC dovrà essere di tipo **Deferred Synchronous**, in quanto dato che l'esecuzione della chiamata impiega un po' di tempo, l'utente potrebbe nel frattempo inviare altri comandi, inoltre ogni chiamata deve ritornare se è andata a buon fine.

In realtà dato che le RPC sono implementate tramite pagine web dinamiche, questo meccanismo è già naturalmente funzionante, in quanto dal browser posso inviare più richieste http contemporaneamente e ricevere i risultati di ognuna di esse in modo indipendente.

Infine una breve analisi del traffico, mostra che il sistema deve poter supportare un numero molto basso di pacchetti per unità di tempo, infatti in normale utilizzo il traffico maggiore derivano dai sensori che modificano il loro valore letto a una velocità relativamente bassa, nell'ordine dei 30 secondi, quindi complessivamente il traffico standard consiste in una decina di pacchetti al minuto. Tuttavia possono capitare brevi burst con qualche pacchetto nel giro di pochi secondi, nel caso qualcuno accenda e spenga ripetutamente una luce per esempio, in questo caso ciò che è vincolante sono la capacità del server e del bridge, quest'ultimo è il più critico ma comunque il traffico è circoscritto solo a quello generato dall'utente e quindi è sicuramente sopportabile nel caso di dispositivi embedded adibiti all'utilizzo in questo ambito.

Il prototipo...

Come è già stato detto, il meccanismo di RPC è implementato mediante alcune pagine php.

Il bridge per inserire un nuovo dato nella tabella plantData del sito, dovrà richiedere la pagina dataIn.php passandogli tramite il metodo GET codice dell'impianto, il codice del dispositivo, nome del dispositivo, datatype, e valore.

La pagina non fa altro che fare l'update nella tabella plantData dell'entry per la quale il codice del dispositivo corrisponde col codice trasmesso (in caso non esista farà un'insert), appena la pagina di telegestione si aggiorna sarà disponibile il nuovo valore.

Nel caso in cui l'utente voglia inviare un comando al bridge dovrà richiamare la pagina "dataIn" presente sul bridge passandogli col metodo GET il codice del dispositivo e il valore nuovo.

Una volta ricevuta la request, il bridge estrapola dall'header i parametri passati col metodo GET, quindi aggiorna la propria immagine interna dell'impianto e invia sul bus il pacchetto di tipo Write Command e risponderà inviando una pagina contenente la stringa "OK",

equivalente all'assenza di errori.

Per aumentare la reattività del sito, la richiesta parte grazie alle parti in AJAX presenti nella pagina di gestione, purtroppo tramite AJAX non è possibile effettuare richieste di pagine remote, quindi la funzione in Javascript richiamerà prima la pagina dataOut.php presente nel sito, passandogli i parametri del caso, questa pagina poi effettuerà la richiesta remota vera e propria al bridge utilizzando l'indirizzo IP memorizzato nella tabella AddressCodeAssociation.

Una volta effettuata la request al bridge e nel caso in cui esso risponda con un "OK", la pagina php provvede a fare l'update dell'entry corrispondente nella tabella plantData, in caso contrario la pagina restituisce un messaggio di errore che viene visualizzato con un alert tramite javascript.

Il valore di ritorno delle procedure remote consisterà quindi nella scritta "OK" nel caso in cui non ci siano stati errori e della stringa "ERROR" in caso di errori, in caso di disconnessioni dei timeout faranno sì che la stringa ritornata sia nulla e comunque diversa da "OK" e rilevata ugualmente come un errore.

Gestione delle Notifiche

Per aggiungere la funzionalità di invio di notifiche via SMS è stato necessario inserire nell'architettura un apposito oggetto che si occupasse di questo compito, l'interfaccia dell'oggetto mostra un metodo "sendSMS" che prende in ingresso numero di telefono del destinatario e testo del messaggio da inviare.

Si vuole dare la possibilità all'utente di impostare notifiche personalizzate su ognuno dei dispositivi presenti nell'impianto.

È necessaria la creazione di un'apposita tabella nel database che contiene tutte le notifiche.

Un'entry di questa tabella avrà tra i campi il codice dell'impianto di appartenenza, il codice del dispositivo a cui quella notifica si riferisce, il tipo di verifica da effettuare secondo una codifica espressa nell'appendice (del tipo uguale, diverso, minore, tra due valori ecc) due campi valore1 e valore2 che sono l'oggetto della condizione da verificare, valore2 verrà trascurato qualora la condizione di verifica abbia solo un termine di confronto, e infine numero e testo del messaggio da inviare.

Inoltre per fare in modo che il messaggio venga inviato solo nel momento in cui la condizione si verifica per la prima volta e non sempre, un ulteriore campo, isActive, è vero quando la condizione è verificata e falso quando invece non è più verificata. L'SMS viene inviato solo nel caso in cui la condizione è verificata ma isActive non è stato ancora settato a vero.

La verifica sulle condizioni presenti nel database avvengono all'interno della pagina dataIn.php e dataOut.php, ovvero ogni volta che vengono chiamate le procedure remote per inviare in entrambi i sensi dei messaggi e quindi per modificare lo stato dell'impianto.

Il prototipo...

Per realizzare l'oggetto che invia gli SMS è stato usato uno smartphone con Android e con attiva un'applicazione chiamata "remoteSMS" che permette di inviare SMS tramite un'interfaccia web sulla porta 8080. Tramite Wireshark è stato analizzato il traffico nell'istante in cui veniva confermato il form presente in questa interfaccia e così si è risalito al formato della richiesta http da inviare al servizio web presente sul telefono. Il fatto che l'interfaccia web sia presente solo sulla porta 8080 è un problema in quanto può essere bloccato il traffico da firewall presenti nella rete locale che non è possibile configurare. Fortunatamente remoteSMS permette di essere utilizzato anche attraverso la porta USB e usando un altro programma sul PC permette di comunicare attraverso la USB utilizzando

comunque un'interfaccia web reindirizzata sul localhost alla porta 8080.

Appendice

Tabella dei DataTypes implementati:

Nome	**Grandezza Rappresentata**	**Formato Dato**
LAMP	Acceso/Spento	1bit (boolean)
DIMMERED_LAMP	Grado di Luminosità	8bit (unsigned char)
TEMPERATURE	Temperatura [°C]	double
POWER	Potenza [W]	Double
DIMMERED_WINDOW	Grado di Apertura di Finestra	8bit (unsigned char)

Tabella dei codici delle condizioni:

Nome	**Codice**
uguale	0
diverso	1
minore	2
maggiore	3
Tra (richiede value2)	4
Non tra (richiede value2)	5