ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

Fondamenti di informatica T2

WINDOWS 10 IoT SU RASPBERRY PI 2: MULTIPARADIGM PROGRAMMING TRA JAVA E C#

CANDIDATO

Luca Marzaduri

RELATORE:
Chiar.mo Prof. Enrico Denti

CORRELATORE/CORRELATORI
Dott. Ing. Roberta Calegari

Anno Accademico 2014/15

sessione III

Sommario

Introduzione

In un mondo in continua evoluzione, la domotica[1] si è fatta largo nel panorama tecnologico nella seconda metà del Novecento allo scopo di rendere intelligenti apparecchiature, impianti e sistemi. Con il termine domotica, derivato dall'unione delle parole *domus* (che in latino significa "casa") e robotica, ci si riferisce alla scienza interdisciplinare che si occupa dello studio delle tecnologie atte a migliorare la qualità della vita nella casa e più in generale negli ambienti antropizzati.

Il settore della home automation è strettamente legato ad un'evoluzione dell'uso della rete, definito per la prima volta da Kevin Ashton come Internet of Things[2] (solitamente sostituito con l'acronimo IoT). In tale sviluppo gli oggetti si rendono riconoscibili e acquisiscono intelligenza, creando un sistema pervasivo ed interconnesso, in maniera che il mondo elettronico tracci una mappa di quello reale, dando un'identità elettronica alle cose e ai luoghi dell'ambiente fisico. Tutti i dispositivi connessi ad internet possono così scambiare informazioni e cooperare allo scopo di raggiungere un fine comune.

Numerose aziende in tutto il mondo sono interessate al concetto dell'IoT, in particolare Microsoft Corporation[3] ha reso ufficialmente disponibile il 10 agosto 2015 Windows 10 IoT Core[4], una versione del sistema operativo progettato per personal computer Windows 10, ottimizzata per dispositivi low-cost di piccola taglia in scenari IoT, come Raspberry Pi 2[5] e Minnowboard Max.

Home Manager[6] è un prototipo di applicazione sviluppato presso il DISI implementato sulla base dell'infrastruttura di coordinazione TuCSoN[7], al fine di controllare e amministrare una casa intelligente. Il sistema realizza un'architettura comprensiva dei dispositivi domestici, permettendo di gestirli in maniera coordinata e altamente configurabile.

L'obiettivo di questa tesi è duplice:

- in primo luogo, sperimentare il sistema Windows 10 IoT Core su tecnologia Raspberry Pi 2, valutandone la compatibilità con i dispositivi disponibili;

- in secondo luogo, realizzare un agente atto alla gestione dell'accensione e dello spegnimento di luci LED, al fine di una futura integrazione con l'applicazione Home Manager.

La tesi avrà quindi la seguente struttura:

- Nel primo capitolo si analizzerà l'ultima versione del prototipo di Home Manager e dell'infrastruttura su cui si basa.
- Il secondo capitolo sarà dedicato alla presentazione della tecnologia Raspberry.
- Nel terzo capitolo sarà preso in esame Windows 10 e le novità portate, concentrandosi particolarmente sulla versione dedicata a scenari IoT per Raspberry Pi 2.
- Il quarto capitolo riporterà gli esperimenti effettuati con i dispositivi disponibili, descrivendone la finalità e la realizzazione.
- Nel quinto capitolo verranno analizzati il problema e i requisiti necessari, seguiti dall'implementazione e dal collaudo del sistema realizzato.
- Infine il sesto capitolo completerà la tesi, presentando le conclusioni ed alcuni sviluppi futuri.

1. Home Intelligence

La domotica ha come obiettivo finale la gestione delle aree di automazione all'interno di una casa, possibile solo nel caso in cui diversi sistemi semplici siano connessi e controllati in modo intelligente. Ci si riferisce quindi ad un ambiente domestico, opportunamente progettato e tecnologicamente attrezzato al fine di eseguire funzioni parzialmente o completamente autonome, con il nome di "casa intelligente". Una Smart Home può essere controllata dall'utilizzatore tramite opportune interfacce utente, inviando comandi e ricevendo informazioni dal sistema intelligente di controllo, basato su un'unità computerizzata centrale oppure su un sistema a intelligenza distribuita.

In questo contesto si colloca il progetto Home Manager, applicazione agent-based per il controllo di una intelligent home, che verrà presentato nel seguente capitolo al fine di offrirne una visione generale.

1.1 Home Manager

Home Manager[6] è un sistema multi-agente, realizzato seguendo la metodologia SODA, per il controllo di una casa intelligente. Il sistema prende in considerazione dispositivi indipendenti nell'ambiente domestico simulato, come una televisione, uno stereo o una lavatrice, ognuno fornito di un particolare agente. Gli agenti non sono semplici componenti software da realizzare separatamente e combinare solo successivamente: essi sono entità singole con uno scopo preciso, le quali devono inserirsi e partecipare in una "società degli agenti"[8].

Il programma permette di dividere l'utenza in semplici visitatori e abitanti della casa. Quest'ultimi possono avere privilegi di amministratore, ottenendo quindi il pieno controllo, oppure essere utenti ordinari, ai quali non è permesso agire sugli altri utenti o sul sistema, ma solo impartire comandi ed esprimere le proprie preferenze. Ai visitatori infine è garantita solo un'assistenza di base.

Grazie ad una recente reinterpretazione, il prototipo attuale è progettato per soddisfare le richieste dell'utente, permettendo l'invio di comandi anche da remoto o tramite

un’applicazione Android. Utilizzando la posizione dell'utente, ottenuta tramite sistemi di geo-localizzazione (es. GPS) integrate nei moderni smartphone, Home Manager è in grado di prendere alcune decisioni in maniera autonoma, anticipando se possibile i bisogni dell'individuo, tenendo sempre in considerazione le sue preferenze. Un ulteriore obiettivo consiste nella gestione e la riduzione del consumo energetico complessivo dell'abitazione.

Home Manager è realizzato in Java ed è basato sulla tecnologia di coordinazione TuCSoN.

1.2 TuCSoN

TuCSoN (Tuple Centres Spread over the Network)[7] è un modello per la coordinazione di processi distribuiti, così come agenti mobili, autonomi ed intelligenti. Esso viene implementato come Java-based middleware distribuito, disponibile open-source sotto licenza GNU LGPL. Alla base vi sono centri di tuple, i quali offrono uno spazio condiviso per la comunicazione basata sullo scambio di tuple, con l'aggiunta di specifiche comportamentali programmabili, rendendo possibile l'individuazione di eventi di interazione e la conseguente associazione a precise reazioni. Tutto ciò è possibile grazie al linguaggio di coordinazione ReSpecT[9].

1.2.1 Modello di base

L'infrastruttura TuCSoN è caratterizzata da tre entità principali:

- Agenti TuCSoN, i quali sono le entità base da coordinare all'interno del sistema distribuito.
- Centri di Tuple ReSpecT, che sono il mezzo sul quale avviene la coordinazione.
- Nodi TuCSoN, che rappresentano l'astrazione topologica di base, sul quale sono ospitati i centri di tuple.

Agenti, centri di tuple e nodi hanno identità uniche all'interno del sistema. Contrariamente ai centri di tuple che hanno un comportamento reattivo, gli agenti hanno

un ruolo pro-attivo. Essi interagiscono tramite scambi di tuple attraverso i centri di tuple utilizzando le primitive di coordinazione TuCSoN.
Ogni nodo, il quale ospita uno o più centri di tuple, fornisce agli agenti uno spazio di coordinazione completo, in modo che le primitive siano invocabili su qualsiasi centro di tuple appartenente al nodo considerato.
I centri di tuple in TuCSoN possono essere raggruppati logicamente, dando vita al concetto di ACC. ACC è l'acronimo di Agent Coordination Context e consiste in un'interfaccia assegnata ad un agente che gli consente di effettuare operazioni su determinati centri di tuple facenti parte di una specifica organizzazione.

1.2.2 Linguaggio e primitive di coordinazione

TuCSoN offre delle API (Application Programming Interface) allo scopo di consentire un facile accesso alle primitive di coordinazione in linguaggio Java. Vi è inoltre la libreria alice.tucson.api.TucsonAgent che permette le creazione e la gestione di agenti in Java.

Realizzato con il modello client-server, dove i client sono gli agenti TuCSoN, mentre i server sono i nodi TuCSoN, tale infrastruttura non è solo un middleware Java-based, ma anche Prolog-based: il sistema si basa sulla tecnologia tuProlog Java-based per la definizione delle tuple e dei loro template, per le operazioni di parsing di primitive e identificatori, o per la definizione del linguaggio ReSpecT[6].

Per partecipare in un sistema TuCSoN, qualsiasi entità deve essere distinguibile:

- un nodo TuCSoN è identificato univocamente dalla coppia *"netid : portno"*. *netid* rappresenta il numero IP o la entry DNS del dispositivo sul quale è in esecuzione il servizio e *portno* il numero di porta dove il servizio di coordinazione TuCSoN è in ascolto di richieste per l'esecuzione di operazioni di coordinazione.
- un centro di tuple è identificato univocamente dall'espressione *"tname @ netid : portno"*, dove *tname* indica il nome dell'entità considerata.

- un agente è identificato univocamente dalla coppia *"aname : uuid"*. *aname* rappresenta il nome dell'agente, mentre *uuid* è un universally unique identifier assegnato all'agente quando entra in un sistema TuCSoN.

Il linguaggio di coordinazione TuCSoN permette agli agenti di interagire con i centri di tuple eseguendo operazioni di coordinazione, le quali sono composte da due fasi:

- invocation, ovvero la richiesta inoltrata dall'agente verso il centro di tuple, la quale trasporta tutte le informazioni riguardanti l'operazione stessa.
- completion, ovvero la risposta proveniente dal centro di tuple di destinazione, indirizzata all'agente richiedente, contenente tutte le informazioni riguardo l'esecuzione dell'operazione dal centro di tuple.

Viene inoltre definita una sintassi astratta generale per indicare qualsiasi operazione di coordinazione TuCSoN:

tname @ netid : portno ? op

Le primitive offerte per la comunicazione sono:

- *out*, scrive una tupla specificata nel centro di tuple.
- *rd*, legge una tupla specificata nel centro di tuple.
- *in*, legge ed elimina una tupla specifica nel centro di tuple.
- *rdp*, legge una tupla specificata nel centro di tuple, se non è presente fallisce l'esecuzione.
- inp, legge ed elimina una tupla specificata nel centro di tuple, se non è presente fallisce l'esecuzione.
- *no*, cerca una tupla specifica nel centro di tuple e, se non la trova, termina correttamente, in caso contrario rimane in attesa che non ci sia più alcuna tupla facente match.
- *nop*, ha lo stesso effetto della precedente primitiva, però non comporta sospensione ma fallimento in caso di match positivo.
- *get*, restituisce una lista con tutte le tuple appartenenti al centro di tuple.
- *set*, sovrascrive le tuple nel centro di tuple con le nuove tuple passate come argomento alla primitiva.

Nella versione attuale di TuCSoN sono state aggiunte inoltre alcune "bulk primitives": out_all, rd_all, in_all che operano su tutte le tuple che fanno match con il template specificato.

2. Raspberry Pi 2

Il Raspberry Pi 2[5] è un single-board computer, ovvero un calcolatore implementato su una sola scheda elettronica. Sviluppato nel Regno Unito dalla Raspberry Pi Foundation, esso è l'evoluzione del Raspberry Pi 1. Viene distribuito in un unico modello chiamato Model B, contrariamente al predecessore, il quale è attualmente in commercio in tre configurazioni: il più economico Model A+, il Model B di fascia intermedia e il Model B+. Raspberry Pi 2 Model B condivide con quest'ultimo diverse caratteristiche :

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

Si differenza invece per quanto riguarda la CPU e la RAM, essendo equipaggiato con:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

Il dispositivo riduce inoltre la corrente (potenza) assorbita, passando da 700mA (3,5 W) a 600 mA (3 W).

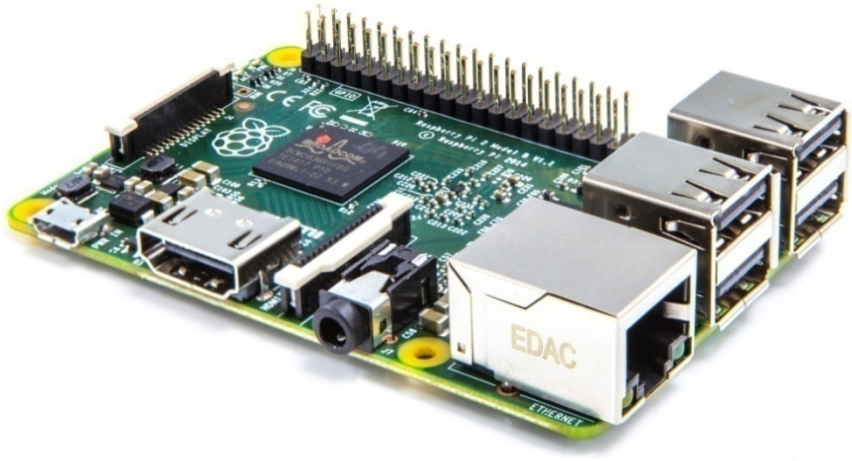


Figura 1: Raspberry Pi 2 Model B

Basato su un System-on-a-chip (SoC) Broadcom BCM2836 ARM (Cortex A7), il progetto non prevede né hard disk né una unità a stato solido, affidandosi invece a una scheda SD per il boot e per la memoria non volatile. Sulla memoria flash possono essere caricate ed eseguite numerose distribuzioni ARM/GNU Linux, tra cui Snappy Ubuntu Core e Raspbian, e solo recentemente una versione dedicata a scenari IoT di Microsoft Windows 10.

3. Microsoft Windows 10

In questo capitolo verrà presentato il sistema operativo Windows 10, al fine di mostrarne le caratteristiche principali.

3.1 Introduzione a Windows 10

Windows 10[10] è un sistema operativo per personal computer rilasciato da Microsoft, facente parte della famiglia di sistemi operativi Windows NT. Presentato il 30 settembre 2014, è stato ufficialmente reso disponibile dal 29 luglio 2015 in sette edizioni denominate Home, Pro, Enterprise, Education, Mobile, Mobile Enterprise e IoT Core.

I requisiti per l'installazione sono gli stessi del predecessore Windows 8.1:

- Processore 1 gigahertz (GHz) o superiore oppure SoC
- RAM 1 gigabyte (GB) per sistemi a 32 bit o 2 GB per sistemi a 64 bit
- Spazio su disco rigido 16 GB per sistemi a 32 bit, 20 GB per sistemi a 64 bit
- Scheda video DirectX 9 o versione successiva con driver WDDM 1.0
- Display 800x600

Alcune delle novità principali sono il ritorno allo storico menù start, la cui assenza nell'edizione precedente aveva suscitato parecchie critiche, l'integrazione di DirectX 12, un nuovo browser web chiamato Microsoft Edge, un nuovo centro notifiche, la possibilità di creare Desktop virtuali, un rinnovato pannello impostazioni e il software di assistenza e riconoscimento vocale Cortana, attivabile sia vocalmente che tramite digitazione dall'apposito comando presente nella barra delle applicazioni.

3.2 Universal App

Windows 8 ha introdotto Windows Runtime (WinRT), un'evoluzione del modello di applicazione di Windows, con l'intento di creare un'architettura applicativa comune.

Windows 10 introduce la piattaforma UWP (Universal Windows Platform)[11], che evolve ulteriormente il modello di Windows Runtime e lo inserisce nella memoria centrale unificata di Windows 10. Essendo parte della memoria centrale, UWP offre ora una piattaforma comune, disponibile su tutti i dispositivi che eseguono Windows 10.

Figura 2: Universal Windows Platform

Contrariamente allo sviluppo in Windows 8.1, con le applicazioni Windows 10 non si è più obbligati a scegliere un sistema operativo, bensì solo una o più famiglie di dispositivi come destinazione delle applicazioni. Una famiglia di dispositivi identifica le API, le caratteristiche del sistema, i comportamenti attesi da tali dispositivi ed il set di dispositivi su cui può essere installata un'applicazione.

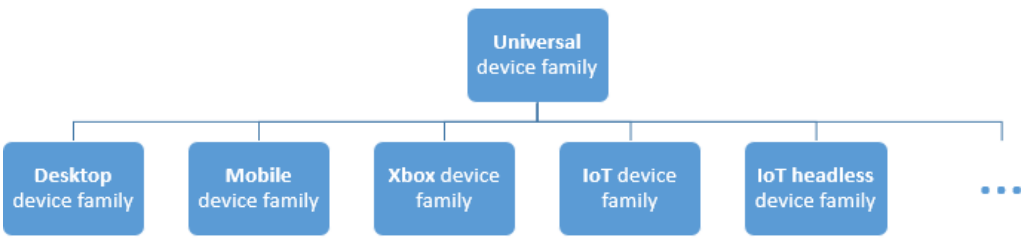


Figura 3: Famiglie di dispositivi

Grazie a questa evoluzione, le applicazioni destinate alla piattaforma UWP, denominate Universal App, possono chiamare non solo le API WinRT comuni per tutti i dispositivi, ma anche API specifiche della famiglia di dispositivi su cui viene eseguita l'app, incluse le API Win32 e .NET. Ciò permette di sviluppare un'unica applicazione, la quale potrà essere installata su un'ampia gamma di dispositivi. Tale approccio è facilitato dal Windows Store, ovvero un canale di distribuzione unificato da cui è possibile scaricare e installare le app.

Considerata la larga scala di dispositivi e le loro diverse caratteristiche, la piattaforma UWP rende automaticamente disponibili le seguenti funzionalità:

- un algoritmo per normalizzare la visualizzazione sullo schermo dei controlli, dei tipi di carattere e di altri elementi dell'interfaccia utente, tenendo conto della distanza di visualizzazione e della densità dello schermo (pixel per pollice) per ottimizzare la dimensione percepita
- un sistema di input basato su interazioni "intelligenti": questo significa possono essere progettate azioni correlate ad un'interazione con clic, senza dover sapere se l'origine è un vero e proprio clic del mouse oppure il tocco di un dito.
- un set di controlli universali completo. Esso include controlli comuni come pulsanti di opzione e caselle di testo, ma anche controlli elaborati come la visualizzazione griglia e la visualizzazione elenco, in grado di generare elenchi di elementi da un flusso di dati e un modello.
- un set di stili e modelli universali, i quali garantiscono animazioni predefinite per le interazioni, supporto automatico di altre lingue, temi personalizzabili e una gamma di tipi di carattere basata su Segoe.

I linguaggi di programmazione con i quali possono essere create applicazioni UWP sono molteplici, tra cui Javascript, C++ e C#. Vi è la possibilità di scrivere alcuni componenti in un linguaggio e successivamente usarli in un'app scritta in un altro linguaggio.

3.3 Windows 10 IoT Core

Windows 10 IoT Core[4] è una versione di Windows 10 progettata specificatamente per dispositivi low-cost di piccola taglia in scenari IoT, trattandosi in particolare di una rivisitazione di immagine della precedente famiglia di sistemi operativi Windows Embedded. Tale distribuzione non presenta le caratteristiche complete di un tradizionale sistema operativo, contrariamente ai sistemi operativi basati su Linux/Unix come Raspbian. Windows 10 IoT Core può essere eseguito in due modalità: headed, dedicata ai dispositivi con un display, i quali usano il sottosistema video di Windows, e headless, per i dispositivi privi di periferica video. I requisiti hardware per l'installazione dipendono in parte dalla modalità di esecuzione:

- **Memory** **Headless**

  256 MB RAM (128 MB free to OS) / 2 GB Storage

  Headed

  512 MB RAM (256 MB free to OS) / 2 GB Storage

- **Processor**

  400 MHz or faster (x86 requires PAE, NX and SS support)

Windows 10 IoT Core è al momento supportato dai sistemi embedded Raspberry Pi 2, Arrow DragonBoard 410c e MinnowBoard MAX.

Al fine di poter installare Windows 10 IoT Core su Raspberry Pi 2 è necessario soddisfare i seguenti requisiti[12]:

- avere un pc con sistema operativo Windows 10 (versione 10.0.10240 o successiva).
- avere installato e aggiornato Visual Studio Community 2015 (oppure le edizioni Visual Studio Professional 2015 / Visual Studio Enterprise 2015) avendo premura di installare Universal Windows App Development Tools e Windows SDK, selezionando le relative checkbox all'atto dell'installazione.

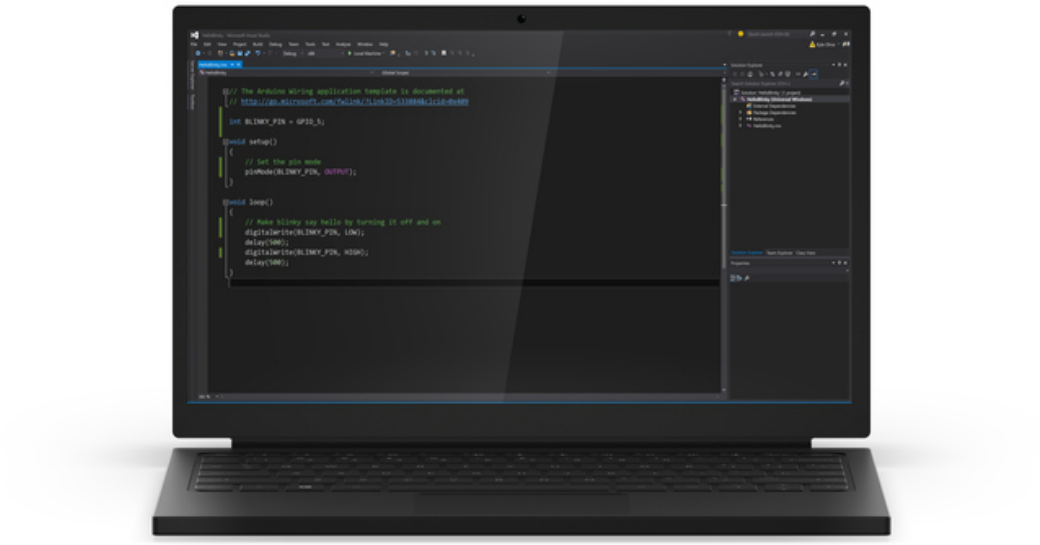


Figura 4: Setup computer con Visual Studio 2015

- avere scaricato Windows IoT Core Project Templates al fine di abilitare i modelli standard tra cui scegliere alla creazione di un progetto in Visual Studio nei linguaggi C#, C++, Visual Basic e Javascript.
- avere abilitato la modalità sviluppatore sul pc con sistema operativo Windows 10.

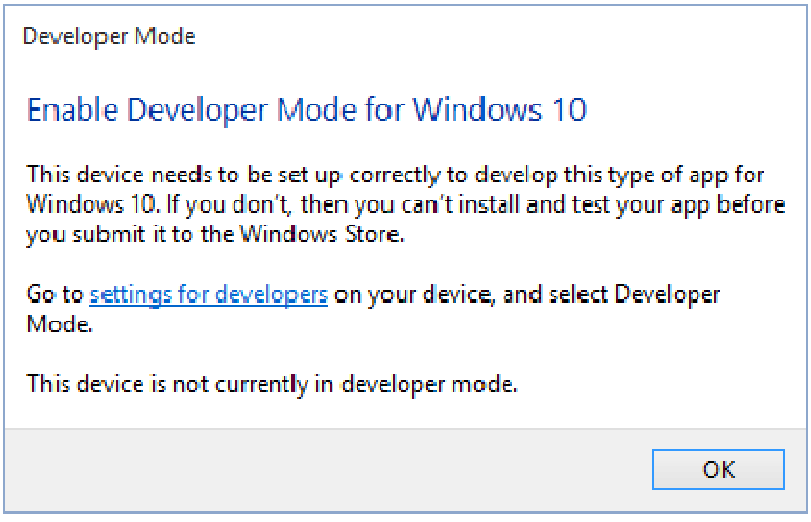


Figura 5: Finestra di dialogo per abilitare la modalità sviluppatore

Sono inoltre necessari i seguenti strumenti:

- Raspberry Pi 2
- Alimentatore micro USB da 5v e almeno 1.0A
- Micro SD card da 8Gb di classe 10 o superiore
- Cavo HDMI e monitor
- Cavo ethernet

Per la vera e propria installazione si devono seguire i seguenti passi:

1. Effettuare il download dell'immagine di disco ISO di Windows 10 IoT Core e montarlo come unità virtuale cliccando due volte sopra il file appena scaricato

Figura 6: Immagine di disco Windows 10 IoT Core

2. Installare *Windows_10_IoT_Core_RPi2.msi* al fine di ottenere il file *flash.ffu* nella destinazione C:\Program Files (x86)\Microsoft IoT\FFU\RaspberryPi2

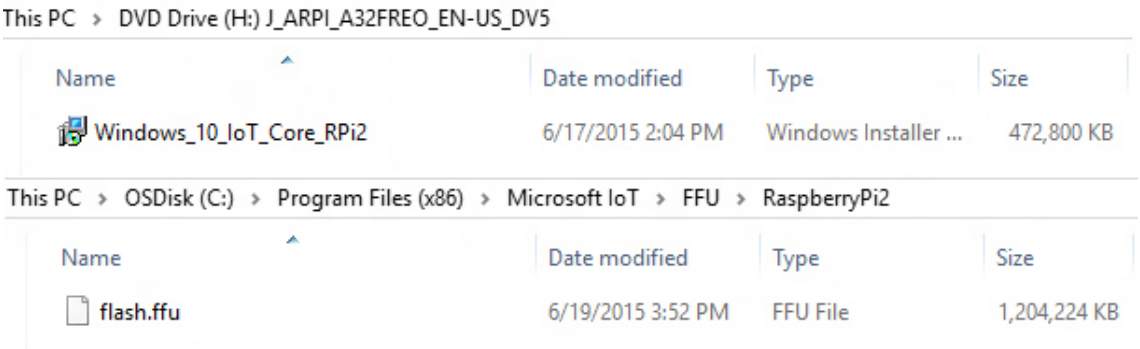


Figura 7: File Windows_10_IoT_Core_RPi2.msi e flash.ffu

3. Utilizzare IoTCoreImageHelper.exe per scrivere sulla Micro SD card

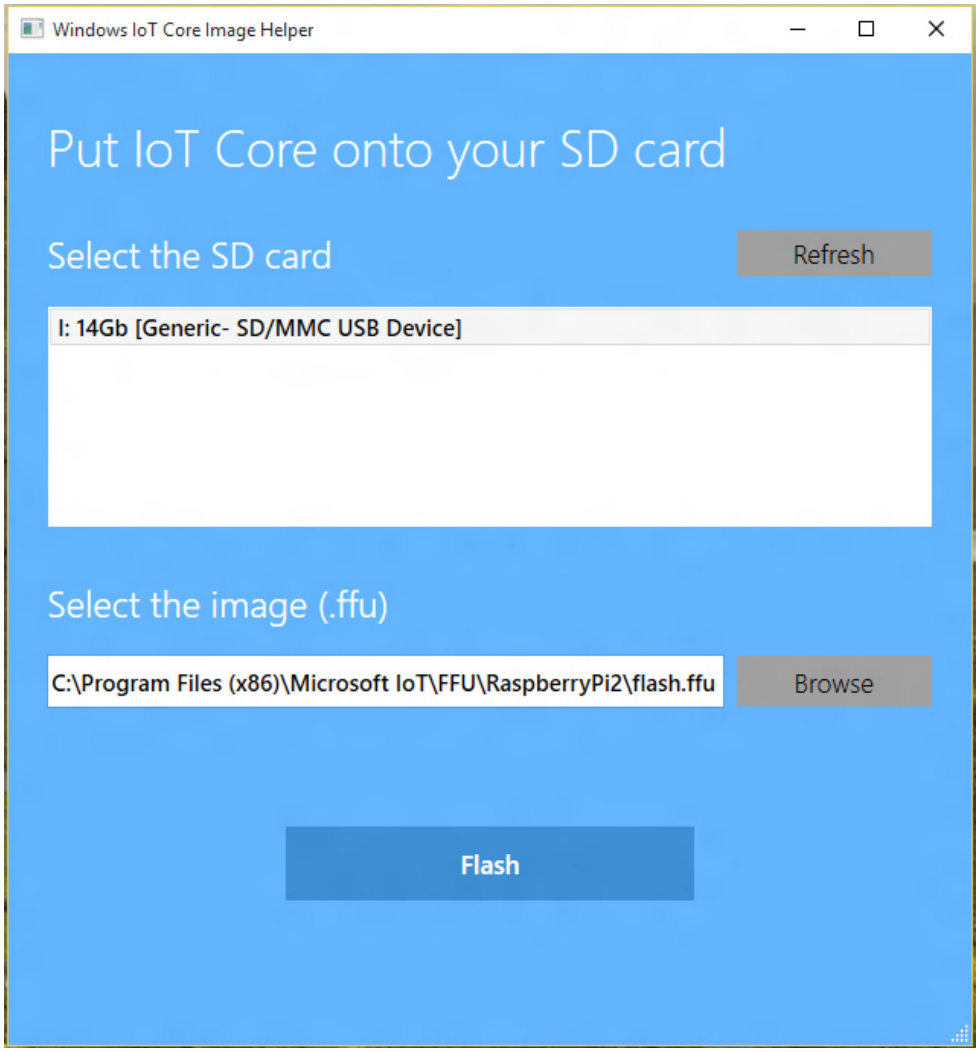


Figura 8: IoTCoreImageHelper

Inserendo ora la Micro SD card nell'apposito slot del Rapsberry Pi 2, avendo precedentemente collegato l'alimentatore, il cavo HDMI e il cavo ethernet, verrà lanciata la DefaultApp.

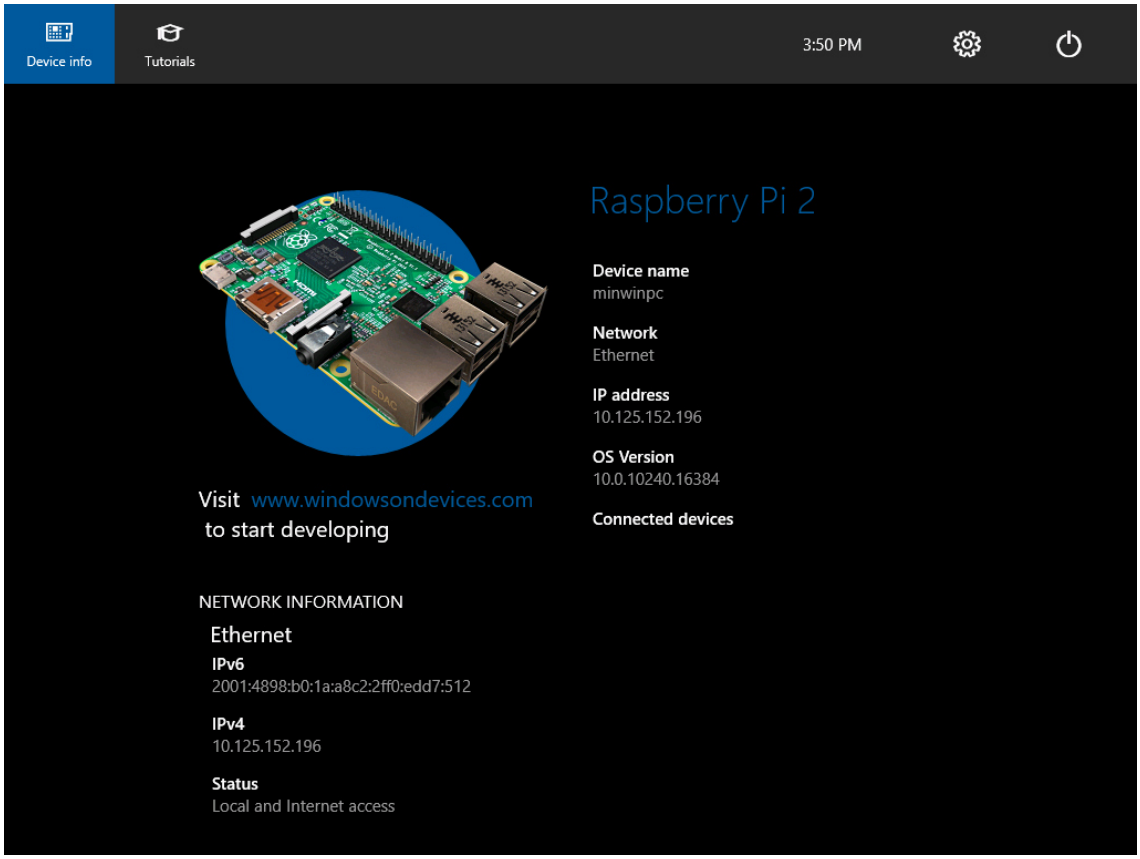


Figura 9: DefaultApp all'avvio di Raspberry Pi 2

3.4 Deployment su Raspberry Pi 2

Dopo aver creato un nuovo progetto di Visual Studio per Windows 10 e la piattaforma UWP (Universal Windows Platform), è possibile eseguire l'applicazione su Raspberry Pi 2 grazie a Visual Studio stesso. E' necessario indicare nella barra degli strumenti l'architettura per cui compilare, ovvero ARM nel caso di Raspberry Pi 2, e selezionare l'opzione "*Computer remoto*", inserendo l'indirizzo IP o il nome del dispositivo. Infine è essenziale specificare l'intenzione di effettuare debug dell'applicazione direttamente sul dispositivo o di effettuarne il vero e proprio deployment.

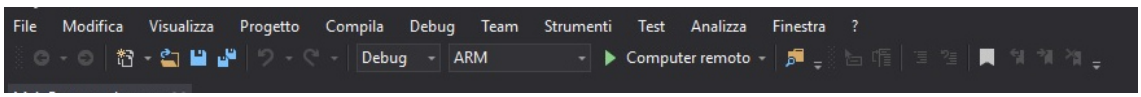


Figura 10: Barra degli strumenti Visual Studio 2015

In tale maniera Visual Studio può identificare il dispositivo dalla connessione ethernet diretta con il computer oppure individuarlo nella rete alla quale è connesso, effettuando così la compilazione e distribuzione adeguata, alla pressione del tasto F5.

4. Esperimenti con dispositivi elettronici disponibili

In questo capitolo saranno illustrati gli esperimenti effettuati con i dispositivi elettronici a disposizione, evidenziandone la finalità e la realizzazione.

4.1 LED

In elettronica il LED[13] (Light Emitting Diode), anche chiamato diodo a emissione luminosa, è un dispositivo optoelettronico che sfrutta le proprietà ottiche di alcuni materiali semiconduttori di produrre fotoni attraverso un fenomeno di emissione spontanea. Esso è uno tra i componenti più semplici che esistano. Vi sono due pin collegati ai capi di un diodo, chiamati anodo e catodo, i quali vanno collegati rispettivamente al voltaggio positivo e a quello negativo. Per facilitarne il riconoscimento, l'anodo corrisponde solitamente al pin più lungo. Al fine di evitare la rottura del componente è necessario inserire nel circuito una resistenza adatta, grazie alla legge di Ohm sostituendo i dati forniti dal relativo datasheet per il corretto funzionamento. Si regola in tal maniera il flusso di corrente che attraversa il LED, permettendo in aggiunta di regolarne la luminosità, diminuendola al crescere del valore del resistore.

Con l'intento di accendere o spegnere un LED via software, è fondamentale scegliere la logica di funzionamento:

- logica negativa, collegando l'anodo ad un pin del Raspberry che fornisca 3.3V e il catodo ad un GPIO (General Purpose Input Output). Così facendo, fornendo via software un 1, astrazione di alto voltaggio, il LED rimarrà spento in quanto non sarà presente differenza di potenziale ai capi del diodo. Al contrario, al presentarsi di uno 0, astrazione di basso voltaggio, il LED si accenderà grazie alla differenza di potenziale ai capi del diodo e quindi allo scorrimento di corrente.
- logica positiva, collegando l'anodo ad un pin GND (ground) del Raspberry, ovvero una messa a terra, e il catodo ad un GPIO. Così facendo, fornendo via software uno 0, il LED rimarrà spento in quanto non sarà presente differenza di

potenziale ai capi del diodo. Al contrario, al presentarsi di un 1, il LED si accenderà grazie alla differenza di potenziale ai capi del diodo e quindi allo scorrimento di corrente.

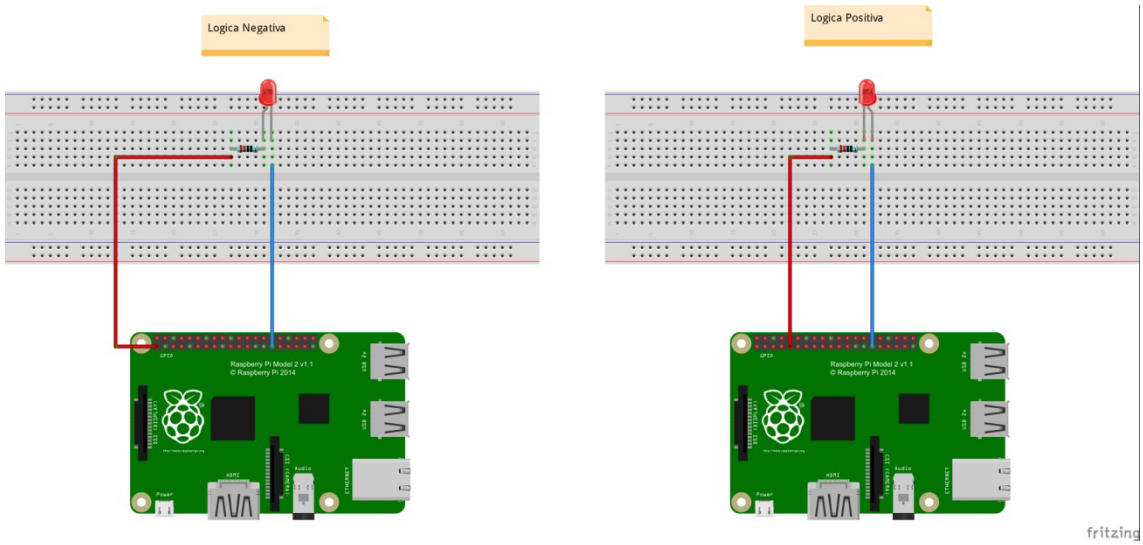


Figura 11: Logica negativa e logica positiva a confronto

In questa tesi si farà sempre riferimento per semplicità alla logica positiva.

Un LED ha la possibilità di assumere un solo colore, contrariamente ad un LED RGB, il quale può assumere 8 diversi colori, grazie alle combinazioni di rosso, verde e blu. Diversamente dal semplice LED, esso presenta 4 pin. Può inoltre presentarsi in due configurazioni: *common cathode* e *common anode*.

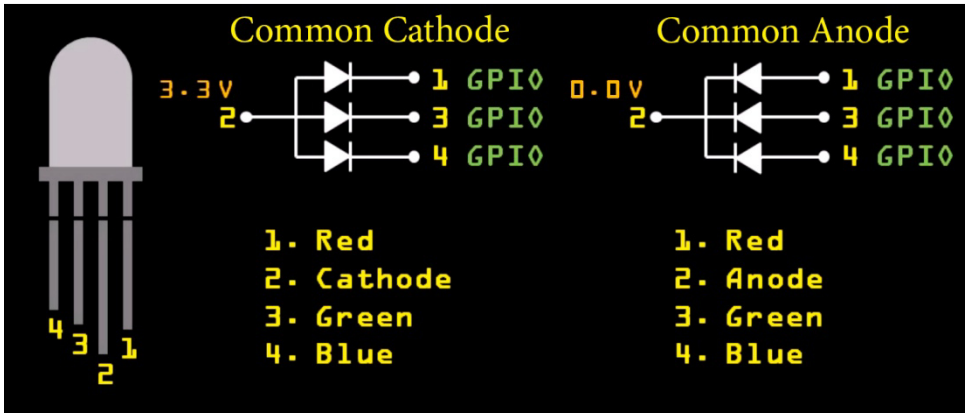


Figura 12: Common cathode e common anode a confronto

In questa tesi si farà sempre riferimento per semplicità alla configurazione *common anode*.

4.1.1 Obiettivi esperimento

Con questo esperimento si vuole verificare la possibilità di accendere da software un semplice LED. Successivamente si riprenderà tale obiettivo, mostrando tutte le 8 possibili combinazioni di colori utilizzando un LED RGB.

4.1.2 Realizzazione

Per permettere l'accensione di un LED, collegato precedentemente in logica positiva al pin GPIO 26 e ad un pin 3.3V, è necessario individuare il GpioController e successivamente aprire il pin GPIO in questione, settando il giusto valore per l'accensione, tramite il seguente codice C#:

```
const int LED_PIN = 26;
GpioPin pin;
GpioPinValue pinValue;
var gpio = GpioController.GetDefault();

if (gpio == null)
{
    pin = null;
    return;
}

pin = gpio.OpenPin(LED_PIN);
pinValue = GpioPinValue.High;
pin.Write(pinValue);
pin.SetDriveMode(GpioPinDriveMode.Output);
```

Figura 13: Codice C# per l'accensione di un LED

Similmente al semplice LED, il LED RBG necessita l'individuazione del GpioController, l'apertura dei pin GPIO collegati come da figura 14 ed il settaggio dei valori di quest'ultimi per ottenere la combinazione desiderata di rosso, verde e blu.

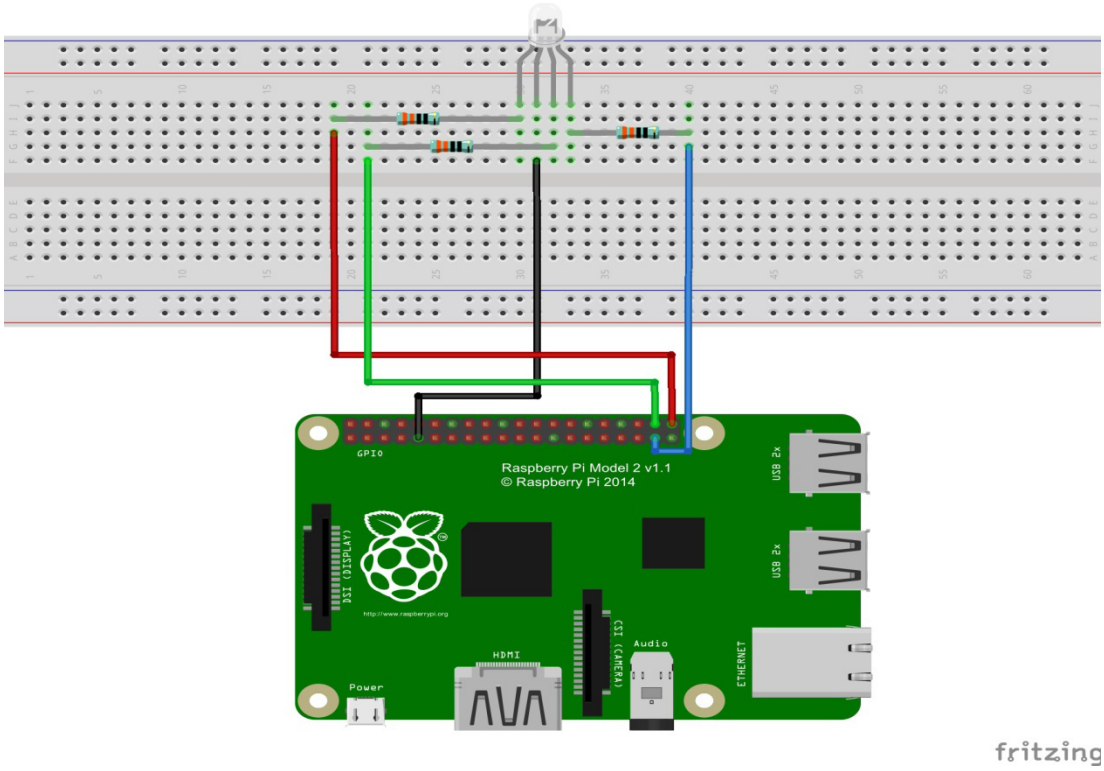


Figura 14: Schema dei collegamenti tra LED RGB e Raspberry

```
const int RLED_PIN = 21;
const int GLED_PIN = 20;
const int BLED_PIN = 26;
var gpio = GpioController.GetDefault();

if (gpio == null)
{
    return;
}

rledPin = gpio.OpenPin(RLED_PIN);
gledPin = gpio.OpenPin(GLED_PIN);
bledPin = gpio.OpenPin(BLED_PIN);

rledPin.Write(GpioPinValue.Low);
gledPin.Write(GpioPinValue.Low);
bledPin.Write(GpioPinValue.Low);

rledPin.SetDriveMode(GpioPinDriveMode.Output);
gledPin.SetDriveMode(GpioPinDriveMode.Output);
bledPin.SetDriveMode(GpioPinDriveMode.Output);

//RED
rledPin.Write(GpioPinValue.High);
gledPin.Write(GpioPinValue.Low);
bledPin.Write(GpioPinValue.Low);

//GREEN
rledPin.Write(GpioPinValue.Low);
gledPin.Write(GpioPinValue.High);
bledPin.Write(GpioPinValue.Low);

//BLUE
rledPin.Write(GpioPinValue.Low);
gledPin.Write(GpioPinValue.Low);
bledPin.Write(GpioPinValue.High);

//MAGENTA
rledPin.Write(GpioPinValue.High);
gledPin.Write(GpioPinValue.Low);
bledPin.Write(GpioPinValue.High);

//YELLOW
rledPin.Write(GpioPinValue.High);
gledPin.Write(GpioPinValue.High);
bledPin.Write(GpioPinValue.Low);

//CYAN
rledPin.Write(GpioPinValue.Low);
gledPin.Write(GpioPinValue.High);
bledPin.Write(GpioPinValue.High);

//WHITE
rledPin.Write(GpioPinValue.High);
gledPin.Write(GpioPinValue.High);
bledPin.Write(GpioPinValue.High);

//OFF
rledPin.Write(GpioPinValue.Low);
gledPin.Write(GpioPinValue.Low);
bledPin.Write(GpioPinValue.Low);
```

Figura 15: Codice C# per la gestione di un LED RGB

4.2 Tactile Switch

Un tactile switch è un componente elettrico il quale può interrompere un circuito elettrico, bloccando il flusso di corrente oppure deviandolo da un conduttore ad un altro. Solitamente esso han quattro pin, come mostrato in figura 16, nonostante le connessioni effettive siano solo due, in quanto B e C sono collegati, così come A e D.

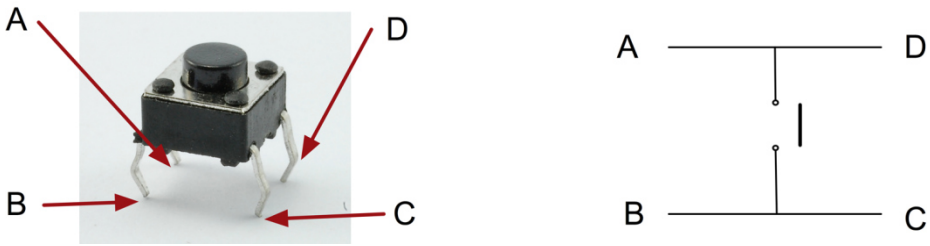


Figura 16: Tactile Switch

4.2.1 Obiettivi esperimento

Si vuole sperimentare la possibilità di rilevare la pressione di un tactile switch per iniziare un ciclo di accensione sequenziale di tre LED.

4.2.2 Realizzazione

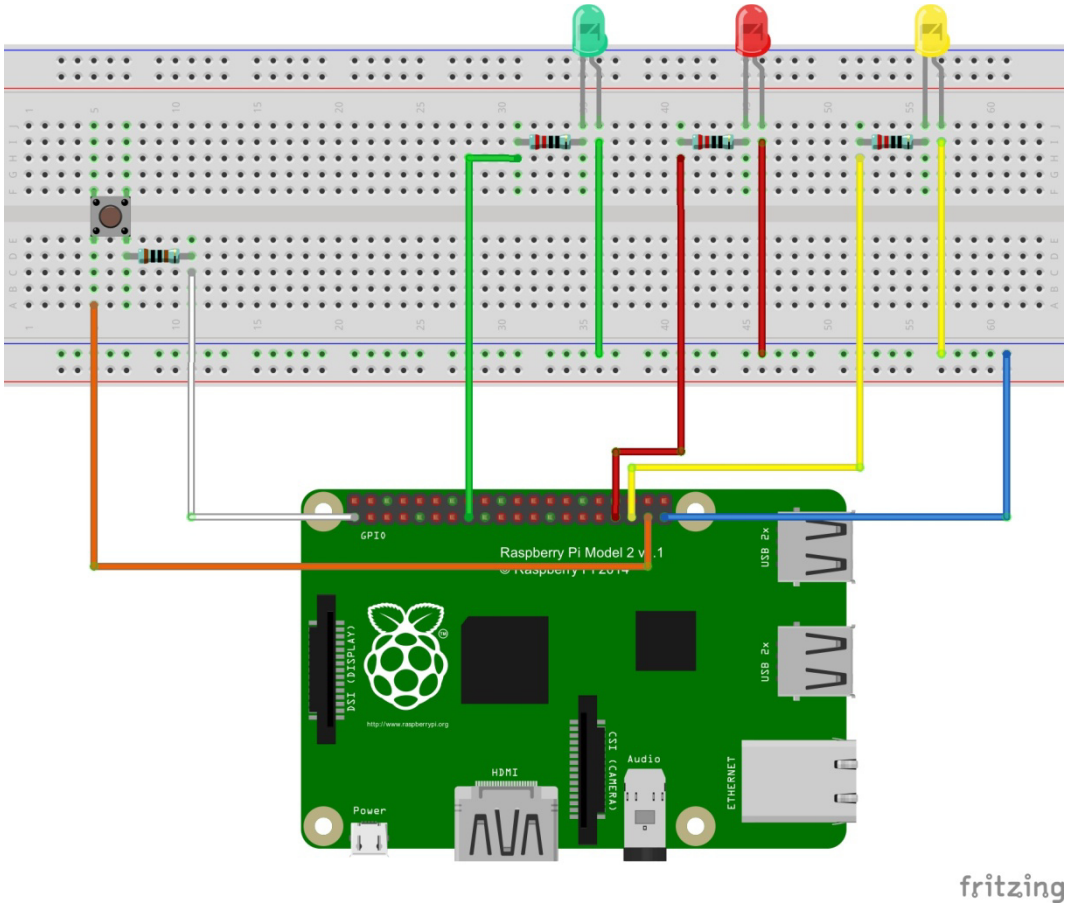


Figura 17: Schema collegamenti Tactile Switch e LED

Dopo aver collegato i componenti come da figura 17 e avendo scelto la logica con cui pilotare il componente, positiva o negativa similmente ai LED, si rende necessario inserire rispettivamente una resistenza di pull down o di pull up, allo scopo di evitare letture errate causate da valori di tensione fluttuanti. Ciò viene facilitato da Raspberry Pi 2 che offre la possibilità di generare tale resistenza internamente, senza bisogno di collegamenti esterni. Utilizzato come componente di input, contrariamente ai LED, il tactile switch è monitorabile grazie all'evento *ValueChanged*, generato alla sua pressione.

```
    public MainPage()
    {
        const int BUTTON_PIN = 26,  GLED_PIN = 22, RLED_PIN = 13, YLED_PIN = 19;
        GpioPin buttonPin, gledPin, rledPin, yledPin;
        var gpio = GpioController.GetDefault();

        if (gpio == null)
        {
            return;
        }

        buttonPin = gpio.OpenPin(BUTTON_PIN);
        gledPin = gpio.OpenPin(GLED_PIN);
        rledPin = gpio.OpenPin(RLED_PIN);
        yledPin = gpio.OpenPin(YLED_PIN);

        gledPin.Write(GpioPinValue.Low);
        rledPin.Write(GpioPinValue.Low);
        yledPin.Write(GpioPinValue.Low);

        gledPin.SetDriveMode(GpioPinDriveMode.Output);
        rledPin.SetDriveMode(GpioPinDriveMode.Output);
        yledPin.SetDriveMode(GpioPinDriveMode.Output);

        if (buttonPin.IsDriveModeSupported(GpioPinDriveMode.InputPullDown))
            buttonPin.SetDriveMode(GpioPinDriveMode.InputPullDown);
        else
            buttonPin.SetDriveMode(GpioPinDriveMode.Input);

        order = new GpioPin[] { gledPin, rledPin, yledPin };

        buttonPin.DebounceTimeout = TimeSpan.FromMilliseconds(50);
        buttonPin.ValueChanged += this.ValueChangedHandler;

        timer = new DispatcherTimer();
        timer.Interval = TimeSpan.FromMilliseconds(500);
        timer.Tick += Roll;
    }

    private void ValueChangedHandler(GpioPin gpiopin, GpioPinValueChangedEventArgs args)
    {
        if (args.Edge == GpioPinEdge.RisingEdge)
        {
            var thread = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () => { timer.Start(); });
        }
    }

    private void Roll(object sender, object e)
    {
        int current = i % 3;
        int prev = (current == 0 ? 2 : current - 1);
        this.order[current].Write(GpioPinValue.High);
        this.order[prev].Write(GpioPinValue.Low);
        i++;
    }
}
```

Figura 18: Codice C# per la gestione di un tactile switch e tre LED

4.3 LCD Display

Un display LCD (Liquid Cristal Display) è una tipologia di display a schermo piatto utilizzata nei più svariati ambiti, con dimensioni dello schermo che variano da poche decine di millimetri a oltre cento pollici. Il dispositivo considerato in questa tesi è in particolare il TC1602A-01T, il quale ha uno schermo da 16 righe e due colonne, comandato tramite 16 pin configurati come illustrato nella figura 19.

Pin No.	Symbol	Level	Description
1	VSS	0V	Ground.
2	VDD	+5.0V	Power supply for logic operating.
3	V0	--	Adjusting supply voltage for LCD driving.
4	RS	H/L	A signal for selecting registers: 1: Data Register (for read and write) 0: Instruction Register (for write), Busy flag-Address Counter (for read).
5	R/W	H/L	R/W = "H": Read mode. R/W = "L": Write mode.
6	E	H/L	An enable signal for writing or reading data.
7	DB0	H/L	This is an 8-bit bi-directional data bus.
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	LED+	+5.0V	Power supply for backlight.
16	LED-	0V	The backlight ground.

Figura 19: Configurazione pin TC1602A-01T

4.3.1 Obiettivi esperimento

Con questo esperimento si vuole verificare la possibilità utilizzare il dispositivo come periferica di output, scrivendo a titolo di esempio sulla prima riga "Windows 10 IoT" e sulla seconda l'orario in costante aggiornamento.

4.3.2 Realizzazione

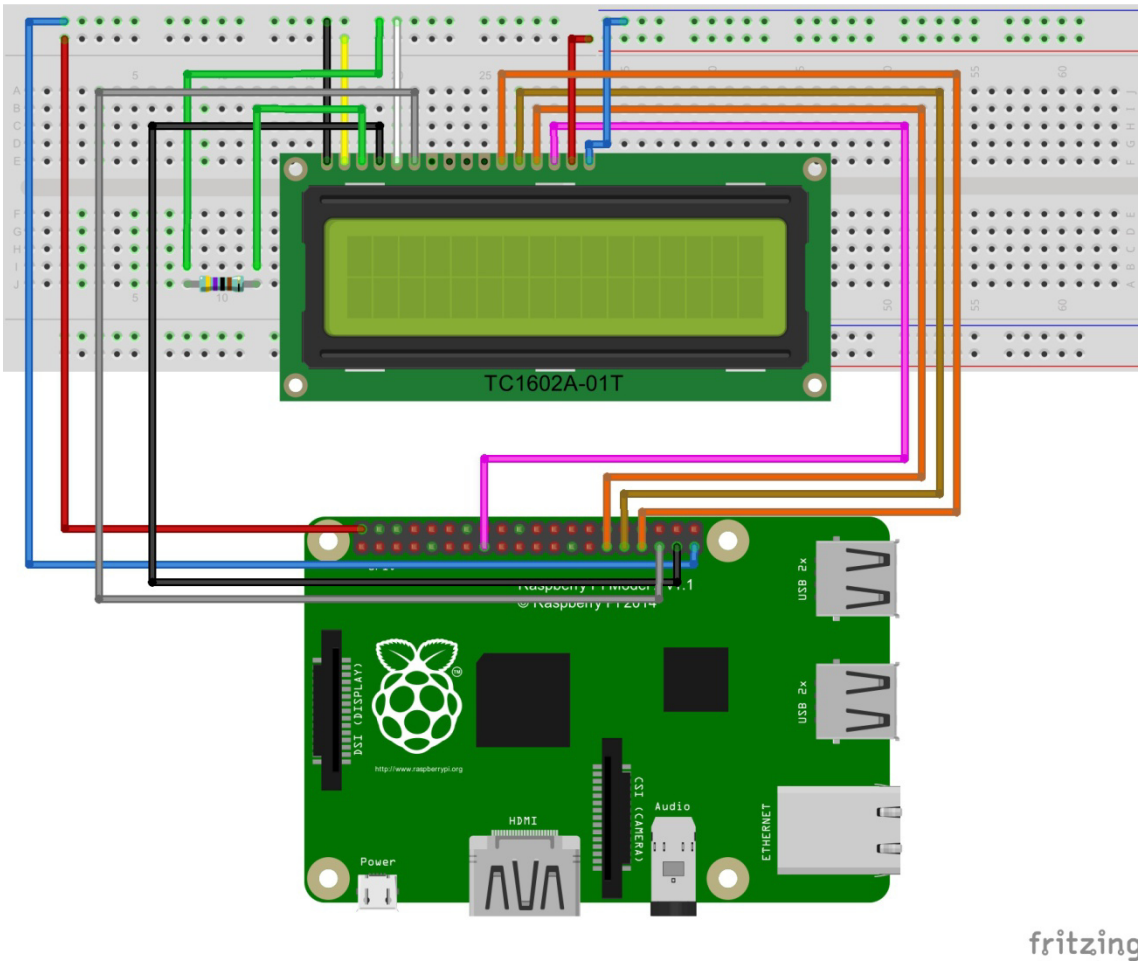


Figura 20: Schema collegamenti LCD Display e Raspberry

Dopo aver collegato il display come illustrato nella figura 20, si rende necessario l'utilizzo di una classe chiamata LCD[14] che si occupi del protocollo di comunicazione con il dispositivo, offrendo così un'interfaccia di più alto livello e API per l'interazione con esso. Tramite il costruttore si definisce la grandezza del display utilizzato, indicando in numero di righe e colonne. Si utilizza poi il metodo *InitAsync* per l'inizializzazione dei pin GPIO, seguito da *clearAsync* per cancellare qualsiasi contenuto momentaneamente visualizzato. Infine con il metodo *write* è possibile scrivere sul dispositivo, prestando attenzione a posizionare correttamente il cursore grazie al metodo *setCursor*.

```
int rs = 26, enable = 19, d4 = 13, d5 = 6, d6 = 5, d7 = 22;
lcd = new LCD(16, 2);

Task<bool> initAsyncRes = lcd.InitAsync(rs, enable, d4, d5, d6, d7);
await initAsyncRes;

Task clearAsyncRes = lcd.clearAsync();
await clearAsyncRes;

String message = " Windows 10 IoT";

lcd.write(message);

while (true)
{
    lcd.setCursor(0, 1);
    lcd.write("Time:" + DateTime.Now.ToString("HH:mm:ss.ffff"));
}
```

Figura 21: Codice C# per la gestione di LCD Display

5. Agente LED integrabile a Home Manager

In questo capitolo saranno riportati i passaggi effettuati per la realizzazione di un agente per la gestione di luci LED, in prospettiva della sua integrazione con il sistema Home Manager.

5.1 Obiettivo

L'obiettivo principale consiste nella progettazione ed implementazione di un agente per la gestione di luci LED su tecnologia Raspberry. Questo agente deve avere la possibilità di comunicare con un centro di tuple TuCSoN, dal quale ottenere informazioni riguardanti l'accensione e lo spegnimento dei LED. In tale maniera si gettano le basi per una futura integrazione con Home Manager, anch'esso basato su architettura TuCSoN.

5.2 Analisi del problema

TuCSoN è un'infrastruttura Java-based, la quale offre API e classi per un facile sviluppo di agenti, configurazione di centri di tuple ed agevole accesso alle primitive di coordinazione. Non essendo però ancora disponibile un'implementazione di JVM (Java Virtual Machine) per Raspberry Pi 2 con distribuzione Windows 10 IoT Core, non è possibile programmare un agente direttamente in linguaggio Java. E' necessario quindi utilizzare un linguaggio supportato come C#. L'agente deve comunicare con un centro di tuple TuCSoN, restando in continua attesa di tuple riguardanti la gestione delle luci.

5.3 Progetto

Essendo TuCSoN progettato ed ottimizzato per Java, esso utilizza la classe *ObjectInputStream* e *ObjectOutputStream* per lo scambio dei dati tra agenti e centri di tuple. Ciò rende però impossibile una comunicazione diretta con altri linguaggi di programmazione, in quanto, prima dell'invio dei dati veri e propri, viene scritto sul canale di trasmissione uno stream header. Quest'ultimo, formato da un numero, definito

nella documentazione ufficiale come numero magico, e dalla versione dello stream, è generato dal metodo *writeStreamHeader* della classe *ObjectOutputStream*. Non essendo però palese l'algoritmo di generazione di tale header, è inevitabile un cambiamento di approccio, interponendo una terza entità che faccia da ponte per la comunicazione.

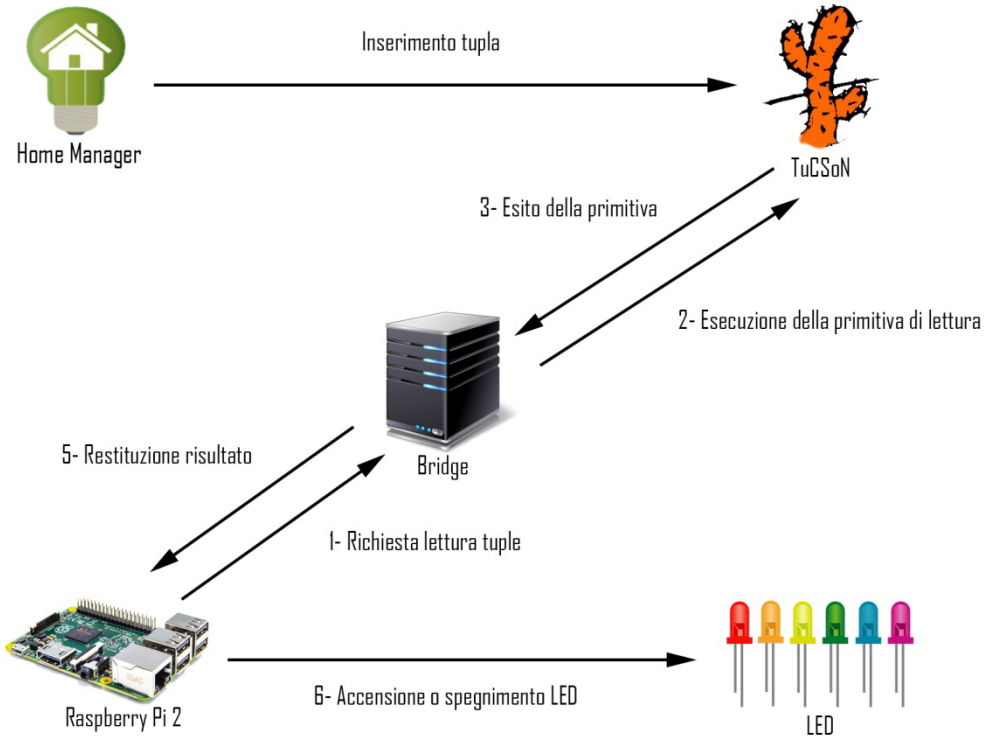


Figura 22: Architettura di comunicazione

Nella figura 22 è mostrato come l'entità Bridge, sviluppata in Java, si occupi da un lato dell'interazione diretta con il centro di tuple TuCSoN e dall'altro della comunicazione con l'agente scritto in linguaggio C# su Raspberry Pi 2.

In questa architettura l'agente C# formula richieste al Bridge, il quale interpreta tali richieste e crea un agente TuCSoN fittizio per l'esecuzione vera e propria delle primitive di coordinazione sul centro di tuple desiderato. Ottenuto l'esito delle primitive, il Bridge inoltra tale risultato all'agente su Raspberry, che si occuperà dell'accensione o dello spegnimento dei relativi LED.

Allo scopo di mantenere coerenza, il protocollo di comunicazione tra l'agente C#, chiamato *AgentTucson*, ed il Bridge, sviluppato in linguaggio Java, consiste nella medesima sintassi astratta descritta nella documentazione TuCSoN:

tname @ netid : portno ? op

Sul canale di comunicazione vengono scambiati i byte che compongono una *string* in formato UTF-8, nel rispetto dell'ordine *BigEndian.*

E' necessario inoltre introdurre il file *config.xml*, dal quale AgentTucson carica i dati necessari per il suo funzionamento, strutturato come segue:

```
<config>
  <server-ip>192.168.1.103</server-ip>
  <port>10252</port>
  <tcname>default</tcname>
  <netid>localhost</netid>
  <netidport>20504</netidport>
  <command>in</command>
  <tuple>update_physical_device_light_curr_st(R, X, Y)</tuple>

  <room>
    <name>hall</name>
    <pin>26</pin>
  </room>
</config>
```

Figura 23:Struttura file config.xml

server-ip e *port* indicano rispettivamente l'IP e la porta del Bridge, *tcname* è il nome del centro di tuple di destinazione, *netid* l'identificativo di rete o l'IP del nodo su cui risiede il centro di tuple *tname*, *netidportno* il numero di porta, *command* la primitiva di coordinazione e *tuple* l'argomento accettato dalla primitiva sopra riportata. E' necessario infine conoscere l'identificativo *name* di ogni LED da gestire, ovvero il nome della stanza dove il LED si trova, ed il numero del pin del Raspberry al quale è collegato.

Le tuple scambiate, per essere considerate valide, devono rispettare la seguente sintassi:

update_physical_device_light_curr_st(Room, X, Y)

Room rappresenta il nome della stanza identificativo di uno specifico LED, *X* è lo stato da raggiungere e *Y* lo stato precedente.

5.4 Implementazione

Per prima cosa AgentTucson carica i dati necessari per il suo funzionamento da file *config*.xml. In accordo a quest'ultimi, vengono poi generate istanze della classe *Room* che si occupano di inizializzare i pin GPIO nella maniera corretta.

Successivamente l'agente si mette in contatto con il Bridge, invocando il metodo *execute*. Tale metodo utilizza la classe *Network* per stabilire la connessione realizzata tramite una *StreamSocket* e di procedere all'invio dei byte relativi alla richiesta. Successivamente l'agente si pone in ascolto della risposta sulla socket aperta.

```
public async Task<string> connect(string host, string port, string message)
{
    HostName hostName;
    string res;

    using (socket = new StreamSocket())
    {
        hostName = new HostName(host);
        socket.Control.NoDelay = false;

        try
        {
            await socket.ConnectAsync(hostName, port);
            await this.send(message);
            res = await this.read();
        }
        catch (Exception exception)
        {
            switch (SocketError.GetStatus(exception.HResult))
            {
                case SocketErrorStatus.HostNotFound:
                    throw;
                default:
                    throw;

            }
        }
    }
    return res;
}
```

Ricevuto l'esito dell'esecuzione della primitiva di coordinazione, il metodo *execute* ricostruisce una o più istanze di *LogicTuple*, classe creata appositamente per riprodurre la struttura di una tupla, utilizzando il metodo statico *Network.parse*. Infine viene controllato che le tuple così ottenute rispettino la relativa sintassi. Qualora il nome della stanza sia presente nella lista inizialmente caricata in memoria e gli stati X e Y siano diversi, si provvede al cambiamento di stato del LED corrispondente, in accordo allo stato X.

```
public async void execute(string host, string port, string message, Network net)
{
    while (true)
    {
        string res = await net.connect(host, port, message);

        if (!res.StartsWith("["))
        {
            this.print("Error: " + res + "\n");
        }
        else
        {
            List<LogicTuple> tuples = Network.parse(res);

            foreach (LogicTuple l in tuples)
            {
                if (l.Arguments.Count > 2)
                {
                    foreach (Room c in rooms)
                    {
                        if (l.Arguments[0].ToLower() == c.Name.ToLower())
                        {
                            if (l.Arguments[1].ToLower() != l.Arguments[2].ToLower())
                            {
                                if (l.Arguments[1].ToLower() == "on")
                                {
                                    c.Pin.Write(GpioPinValue.High);
                                    this.print(c.Name + " LED: ON" + "\n");
                                }
                                if (l.Arguments[1].ToLower() == "off")
                                {
                                    c.Pin.Write(GpioPinValue.Low);
                                    this.print(c.Name + " LED: OFF" + "\n");
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

E' importante sottolineare che AgentTucson può essere eseguito sia in modalità headed sia in modalità headless. Per mostrare all'utente ciò che avviene, è stato quindi predisposto un evento *print*. Qualsiasi funzione che rispetti la sintassi dettata da *printDelegate* può essere registrata presso tale evento. Nonostante sia stata implementata solo la stampa su video collegato tramite cavo HDMI, in questa maniera il sistema rimane aperto ad estensioni, permettendo l'utilizzo di numerosi dispositivi, ad esempio display LCD.

```
public delegate void printDelegate(string message);
public event printDelegate print;

this.print += this.printToVideo;

 public void printToVideo(string message)
{
    textBox.Text += message;
}
```

Il Bridge consiste in un server implementato in Java, eseguibile su qualsiasi macchina che abbia una JVM. Esso può essere eseguito specificando l'identificativo dell'agente fittizio che comunicherà con il centro di tuple indicato, l'IP e la porta della macchina sulla quale il server rimarrà in ascolto di richieste da parte di AgentTucson.

Per ogni richiesta di connessione accettata, viene creato un thread dedicato. Quest'ultimo si occupa per prima cosa di leggere dal canale di comunicazione i byte inviatigli, formattandoli grazie al metodo *parseCommand*. Tale metodo permette di ottenere il nome del centro di tuple, l'IP e la porta del nodo su cui risiede quest'ultimo, la primitiva di coordinazione da eseguire ed infine i parametri per la sua invocazione.

```
byte[] command = new byte[10000];

try
{
    inSock.read(command);
    while (inSock.available() != 0)
    {
        inSock.read(command);
    }
}
catch(SocketTimeoutException ste)
{
  System.out.println("Timeout: ");
  ste.printStackTrace();
  clientSocket.close();
  return;
}
catch (Exception e)
{
  e.printStackTrace();
  return;
}
String[] parsedCommand = this.parseCommand(new String(command, StandardCharsets.UTF_8));

String tcName = parsedCommand[0];
String netid = parsedCommand[1];
String portno = parsedCommand[2];
String cmd = parsedCommand[3];
String arg = parsedCommand[4];
```

Successivamente viene eseguita, grazie il metodo *performCommand*, la primitiva di coordinazione richiesta. Infine viene valutato il successo di tale esecuzione, inviando come riposta il risultato ottenuto oppure un messaggio di errore sul canale di trasmissione aperto con AgentTucson.

```
tid = new TucsonTupleCentreId(tcName, netid, portno);
LogicTuple tuple = null;

if (arg != null)
{
    try
    {
        tuple = LogicTuple.parse(arg);
    }
    catch (InvalidLogicTupleException e)
    {
        System.out.println("LogicTuple parsing error");
        e.printStackTrace();
    }
}

ITucsonOperation op = this.performCommand(cmd, tuple);
List<LogicTuple> res = null;
```

```
if (op != null)
{
    if (op.isResultSuccess())
    {
        try
        {
            res = op.getLogicTupleListResult();
        }
        catch (Exception e)
        {
            res = new ArrayList<LogicTuple>();
            res.add(op.getLogicTupleResult());
            e.printStackTrace();
        }
        outSock.write(res.toString().getBytes("UTF-8"));
    }
    else
    {
        outSock.write("Failed command execution".getBytes("UTF-8"));
    }
}
else
{
    outSock.write("Command not found".getBytes("UTF-8"));
}
```

5.5 Collaudo

Per verificare il corretto funzionamento dell'intero sistema è necessario avviare in primo luogo avviare TuCSoN ed il Bridge sui nodi prescelti.

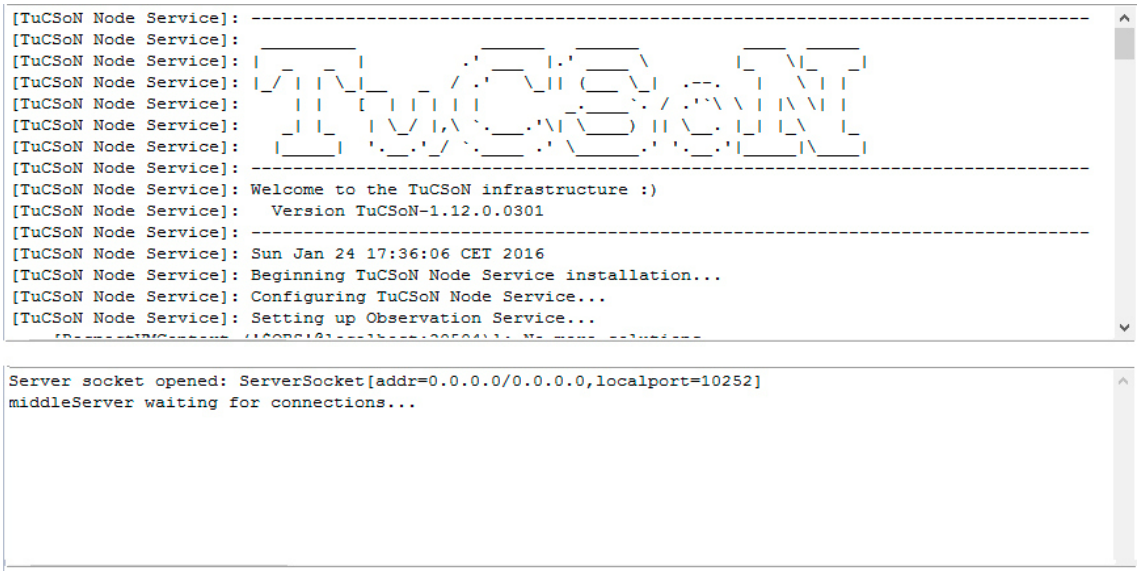
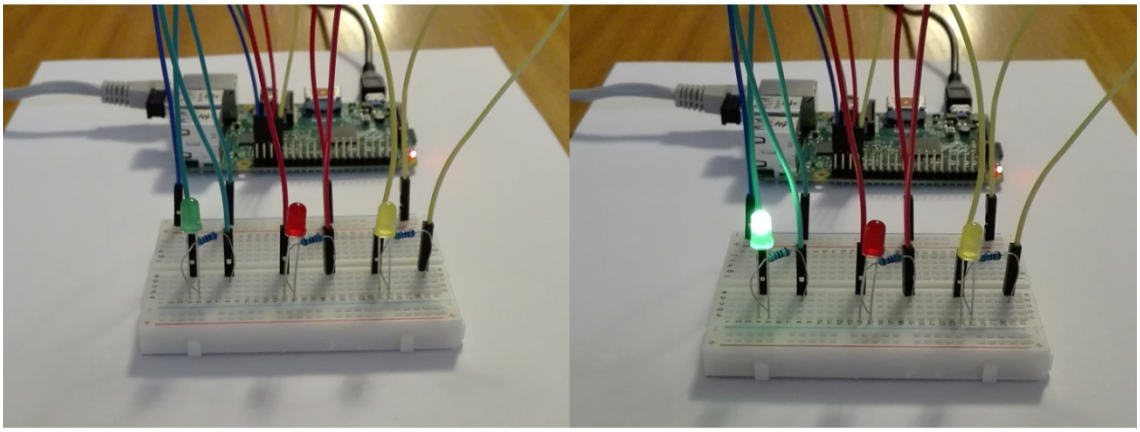


Figura 24: Avvio TuCSoN e Bridge

Successivamente si deve effettuare il deployment di AgentTucson su Raspberry Pi 2, come descritto nei capitoli precedenti. Infine, avviato anche AgentTucson, è possibile simulare l'inserimento di tuple da parte di HomeManager in un centro di tuple utilizzando un semplice agente Java, scritto sulla base delle API TuCSoN. Inserita una qualsiasi tupla valida, si può riscontrare il raggiungimento dell'obiettivo prefissato con successo.

1- [Home Manager] out(update_physical_device_light_curr_st(hall, on, off))



2- [Home Manager] out(update_physical_device_light_curr_st(dining_room, on, off))

3- [Home Manager] out(update_physical_device_light_curr_st(hall, off, on))

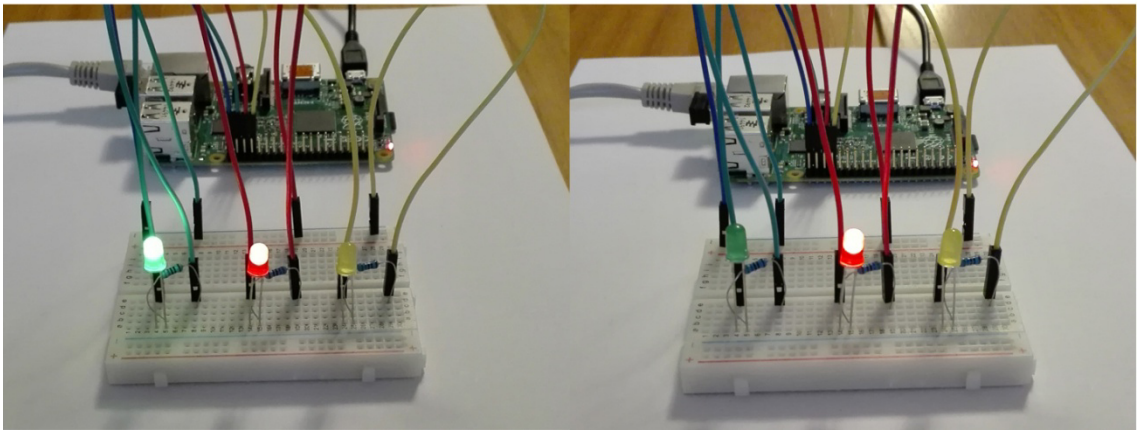


Figura 25: Collaudo accensione LED

6. Conclusioni

Questa tesi si è posta come obiettivo l'analisi della distribuzione Windows 10 IoT Core su Raspberry Pi 2, sperimentando sensori e dispositivi per valutarne l'adeguato funzionamento. Il secondo obiettivo è consistito nella realizzazione di un agente per la gestione dell'accensione e dello spegnimento di luci LED, eseguibile sulla tecnologia precedentemente studiata e collaudata.

Windows 10 IoT Core si è dimostrato essere un sistema limitato nelle potenzialità rispetto alle altre distribuzioni Unix/Linux per Raspberry, offrendo solamente alcune funzionalità di un sistema operativo completo e non garantendo completa compatibilità con molti sensori in commercio. Esso ha permesso però un efficace utilizzo di dispositivi semplici ed una grande facilità di sviluppo e distribuzione delle relative Univesal App. Essendo inoltre un sistema molto recente, Windows 10 IoT Core è in continuo aggiornamento ed evoluzione per offrire un'esperienza sempre migliore nel settore dell'Internet of Things.

Il lavoro fatto ha sottolineato come l'infrastruttura TuCSoN sia un valido strumento per una facile comunicazione ed interoperabilità tra agenti, sebbene limitata al linguaggio Java. Tale ostacolo è stato superato grazie allo sviluppo del Bridge che, allargando gli orizzonti a tutti i linguaggi di programmazione, permette al sistema HomeManager ed a qualsiasi architettura basata su TuCSoN di avere una società di agenti eterogenea.

Un importante sviluppo futuro consisterà nell'integrazione del lavoro fatto con Home Manager, allo scopo di una gestione automatizzata e intelligente delle luci di casa.

Bibliografia

[1] Tratto da Wikipedia:

https://it.wikipedia.org/wiki/Domotica

[2] Tratto da Wikipedia:

https://it.wikipedia.org/wiki/Internet_delle_cose

[3] Tratto dal sito ufficiale Microsoft:

https://www.microsoft.com/it-it/

[4] Tratto dalla documentazione ufficiale Microsoft:

http://ms-iot.github.io/content/en-US/IoTCore.htm

[5] Tratto da Wikipedia

https://it.wikipedia.org/wiki/Raspberry_Pi

[6] Prof. Enrico Denti, Ing. Roberta Calegari, “Home Manager”:

http://apice.unibo.it/xwiki/bin/view/Products/HomeManager

[7] Prof. Andrea Omicini, Prof. Stefano Mariani, “TuCSoN Home”:

http://apice.unibo.it/xwiki/bin/view/TuCSoN/

[8] Prof. Andrea Omicini, " SODA: Societies and Infrastructures in the Analysis and Design of Agent-based Systems"

http://lia.deis.unibo.it/~ao/pubs/pdf/2000/aose.pdf

[9] Tratto dalla documentazione ufficiale ReSpecT:

http://apice.unibo.it/xwiki/bin/view/ReSpecT/

[10] Tratto da Wikipedia

https://it.wikipedia.org/wiki/Windows_10

[11] Tratto dalla documentazione ufficiale Microsoft:

https://msdn.microsoft.com/it-it/library/windows/apps/dn894631.aspx

[12] Tratto dalla documentazione ufficiale Microsoft:

https://ms-iot.github.io/content/en-US/GetStarted.htm

[13] Tratto da Wikipedia

https://it.wikipedia.org/wiki/LED

[14] Tratto da Github

https://github.com/Voltrr1/LCDWin10