Elaborato di progetto

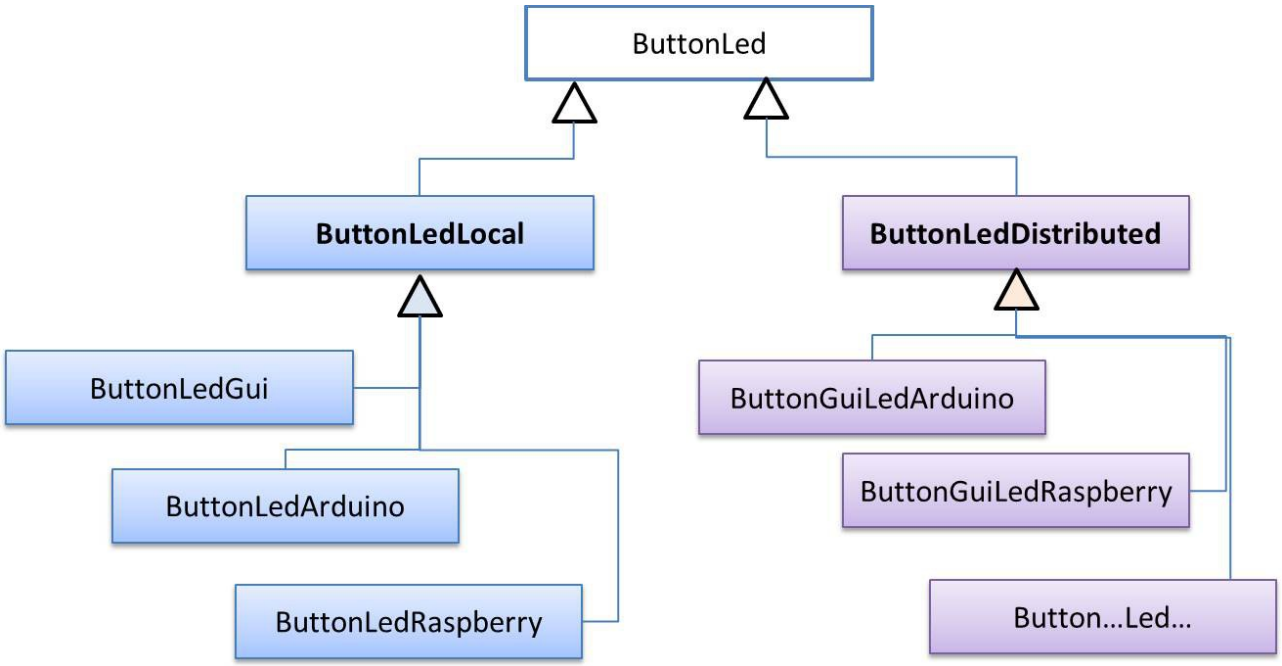
per il corso di

Ingegneria del Software

Stefano Agarbati

Requisiti

1. Progettare e costruire un sistema software (ButtonLedSystem o BLSystem) in cui un Led viene acceso e spento ogni volta che un Bottone viene premuto (da un utente umano). Il sistema dovrebbe eseguire su un singolo supporto, cioè PC convenzionale, un RapsberryPi, un Arduino.
2. Successivamente al primo passo, modificare il sistema software in modo tale che il Led possa essere acceso e spento da un programma remoto. Cioè il BLS dovrebbe avere diverse configurazioni locali e distribuite come mostra la figura sottostante:

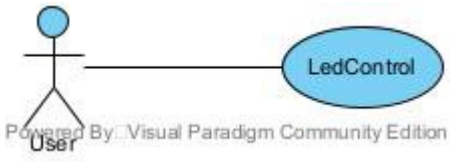


Procedo alla costruzione del sistema seguendo un processo di sviluppo iterativo caratterizzato dai seguenti passi:

1. Analisi dei requisiti
2. Analisi del problema
3. Progettazione
4. Implementazione

Analisi dei Requisiti

Cominciamo con l'analizzare il testo che descrive i requisiti. La lettura è rivolta all'individuazione dei principali casi d'uso e dei concetti principali del dominio applicativo.
Leggendo il testo si possono notare due concetti necessari alla soluzione del problema buttonLed: quello di led e quello di bottone. Inoltre il testo ci dice che il bottone viene premuto da un utente umano che per noi diventa un attore esterno al sistema e che con esso interagisce. Il risultato dell'utilizzo del sistema che l'utente può osservare e che è di valore per esso è l'accensione e lo spegnimento del led. Un possibile caso d'uso, che possiamo chiamare LedControl, può essere descritto attraverso il seguente diagramma UML



Il caso d'uso individuato può essere descritto dal seguente scenario:

ID(Nome)	UC1 – Controllo led
Descrizione	Come utente vorrei che un led si accenda quando un bottone venga premuto e si spenga quando premuto di nuovo. Un led inverte il suo stato ogniqualvolta una bottone venga premuto (user story)
Attori	Utente
Scenario principale	Led inizialmente spento. L'utente preme il bottone e il led si accende. Successivamente l'utente preme di nuovo il bottone e il led si spegne.

Ora dobbiamo chiarire con il committente alcune cose.

Cosa si intende per Led? Cos'è un led?
Cosa si intende per bottone?
Che cosa significa premere un bottone?
Che cosa significa accendere e spegnere un led?

Led

Un led è un dispositivo di uscita, un attuatore luminoso. Può essere un dispositivo fisico oppure un dispositivo virtuale. Per virtuale si intende un componente grafico parte di una interfaccia grafica (GUI). Un led fisico, quando attraversato da una corrente di 15-18 mA comporta un caduta di tensione di circa 1-2 volt ed emette luce. La luce emessa può avere diversi colori come rosso, giallo, verde, bianco ed altri. Un led virtuale è un componente grafico implementabile in diversi modi: ad esempio potrebbe essere modellato come un pannello che assume due colori distinti associati uno allo stato spento e l'altro allo stato acceso. Quindi indipendentemente dal fatto che un led sia virtuale o fisico, esso ha sempre due possibili stati: acceso e spento. Il suo comportamento può essere descritto mediante una macchina a stati finiti avente due soli stati che potremmo chiamare On ed Off.
Gli eventi che comportano transizioni di stato sono generati dall'esterno. Un led fisico viene pilotato da un sistema di controllo, rappresentato in genere da un microcontrollore.

Bottone

Un bottone è un dispositivo di ingresso. Esso può essere sia di natura fisica che di natura virtuale, dove per natura virtuale si intende un bottone che sia parte di una interfaccia grafica (GUI). Serve ad acquisire un valore dall'ambiente esterno. Il valore restituito da un bottone è binario: 1 o 0, alto o basso, premuto o rilasciato, vero o falso ecc……. Premere o rilasciare un bottone significa modificarne il suo stato interno. Un bottone può essere visto come una macchina a stati finiti dotata di soli due stati, premuto e rilasciato

Le transizioni da uno stato all'altro sono innescate da un utente esterno. Ciò non esclude il fatto che lo stato di un bottone possa essere modificato da un altro programma, da una macchina. Un bottone fisico è, in genere, collegato ad un microcontrollore cioè un dispositivo che fornisce la capacità di capire quando un bottone venga premuto oppure rilasciato (campionando il segnale da esso emesso).

Modelli del led e del bottone

Modello del led

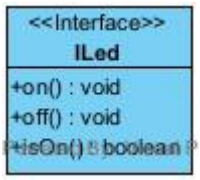
Un led, dal punto di vista logico, è un'entità con stato binario: ha solo due possibili stati che possiamo indicare con acceso l'uno e spento l'altro. E' passivo, cioè il suo stato interno può cambiare solo a seguito di una interazione con qualche altra entità.

Struttura

Il led può essere formalizzato usando l'interfaccia ILed. ILed ci dice che un led è un'entità dotata di stato modificabile attraverso opportune operazioni (due stati possibili). Per rendere comprensibile ad una macchina il modello, possiamo usare un linguaggio di programmazione per descrivere un led:

```
public interface ILed {

	void on(); //modificatore

	void off(); //modificatore

	boolean isOn(); //predicato

}
```

In UML

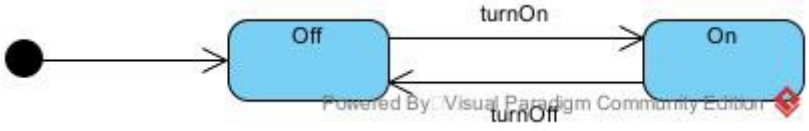


Interazione

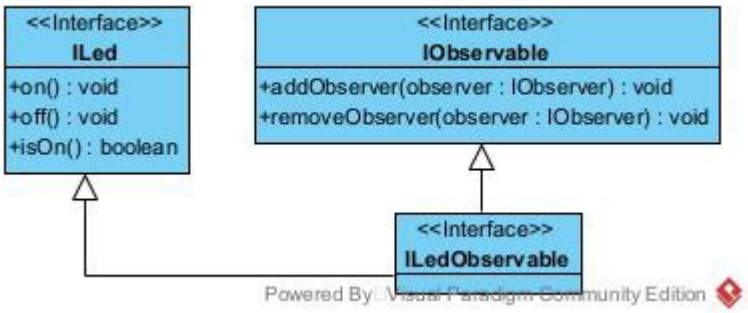
E' possibile interagire con un Led usando una delle operazioni messe a disposizione dalla sua interfaccia (ILed).

Comportamento

Un led può essere visto come una entità passiva, cioè una entità che svolge un lavoro solo se la richiesta è esplicita (non agisce in maniera autonoma). Il suo comportamento è determinato da tre operazioni (un predicato e due modificatori) e può essere descritto mediante una macchina a stati finiti dotata di soli due stati:



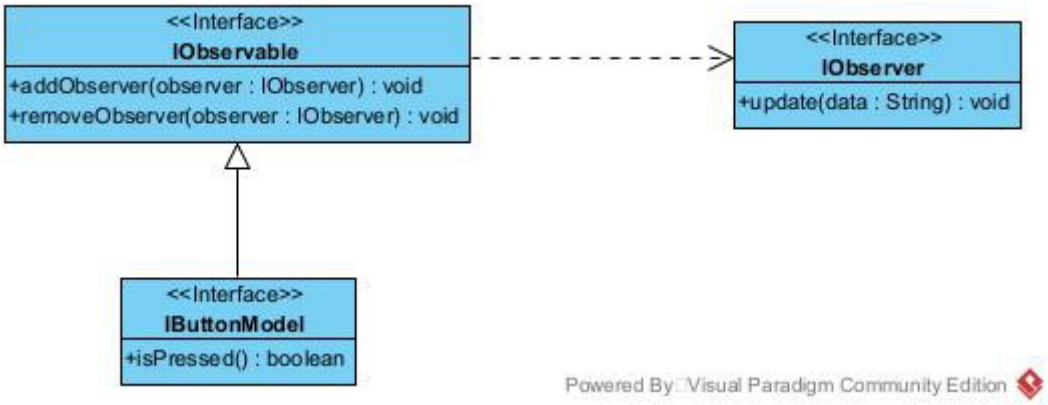
Il comportamento del led può essere specificato formalmente anche attraverso un test espresso in java con l'aiuto di Junit.
Il test serve a specificare meglio la semantica delle operazioni presenti nell'interfaccia ILed.
Un led potrebbe essere modellato anche come entità osservabile (gof) visto che è dotato di uno stato che nel tempo può cambiare.



Modello del bottone

Un bottone, dal punto di vista logico, è un'entità con due stati possibili (stato binario del tipo alto/basso, premuto/rilasciato, 1/0, acceso/spento....). Lo stato interno dipende dalle azioni compiute sul bottone da una entità esterna (un utente umano oppure una macchina). Questo dovrebbe indicare l'assenza di modificatori (per modificatore si intende un' operazione la cui esecuzione porta ad effetti collaterali cioè comporta una modifica dello stato interno dell'entità a cui l'operazione viene applicata).

Struttura



Il bottone potrebbe essere modellato formalmente usando la seguente interfaccia

```
public interface IButtonModel extends IObservable {

}
public interface IObservable {

	void addObserver(IObserver anObserver);

	void removeObserver(IObserver anObserver);

}
```

Interazione

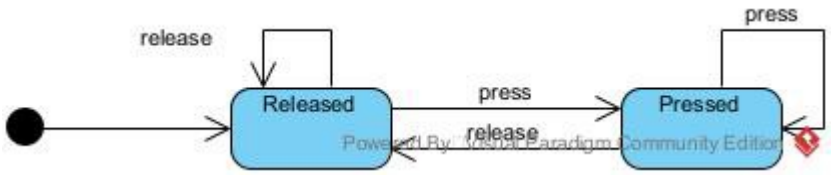
Un generico dispositivo di input, può interagire con altre entità principalmente in due modi:

- polling: si chiede al dispositivo, continuamente, il suo stato interno
- observable: un dispositivo di input può essere modellato come un observable GOF che trasmette il proprio stato a tutti gli interessati registrati ogni volta che ci sia un cambiamento

Esistono anche altre modalità

Comportamento

Il comportamento di un bottone può essere descritto attraverso una macchina a stati finiti dotata di due soli stati:



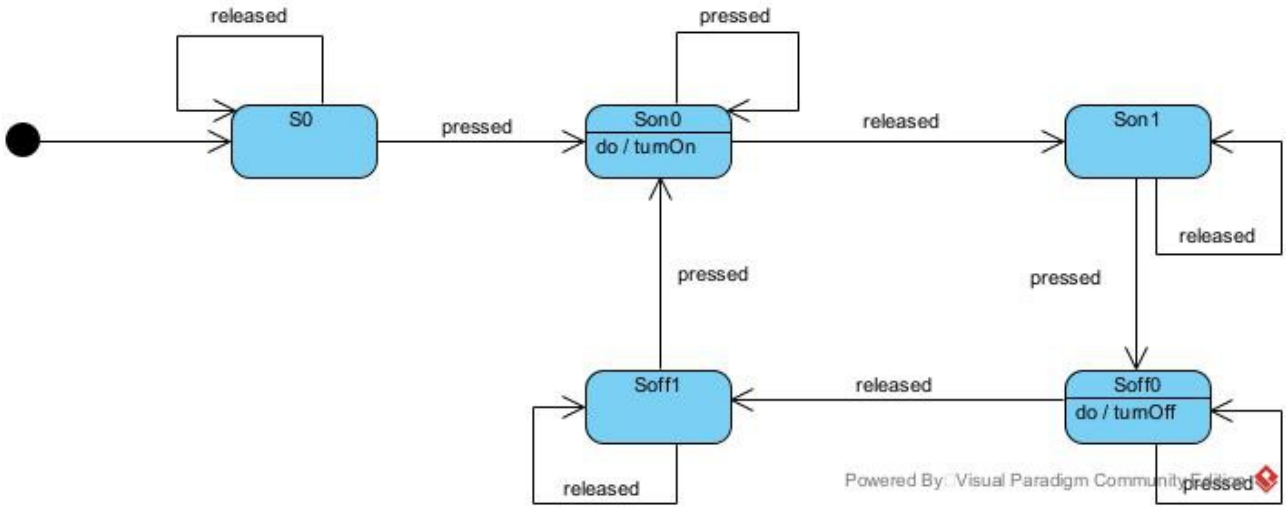
Il comportamento di un bottone può anche essere specificato formalmente attraverso un test.

Analisi del Problema

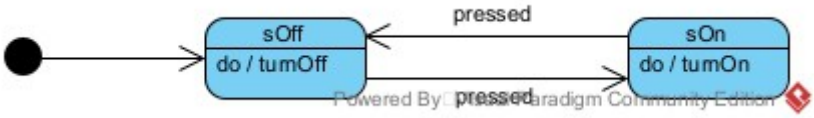
Dobbiamo analizzare il testo che descrive i requisiti e il caso d'uso (gli scenari d'uso) al fine di identificare i problemi da risolvere. Quindi si passa ad analizzare i singoli problemi identificati decomponendoli, se necessario, in ulteriori sotto-problemi (in maniera ricorsiva) e arrivare ad una architettura logica per il sistema. Per architettura logica si intende un insieme di componenti astratti che, nel complesso, realizzano i casi d'uso. Ciascuno di questi componenti risolverà un sotto-problema parte di un problema più ampio (divide et impera)

In questa iterazione ci concentriamo sulla versione concentrata del sistema lasciando la versione distribuita alle successive iterazioni.

Per poter realizzare lo scenario descritto sopra, avremo bisogno, sicuramente, di un led e di un bottone. La logica dell'applicazione consiste nell'invertire lo stato del led ogni volta che il bottone venga premuto. Tale logica non dovrebbe appartenere ai componenti del dominio ovvero non dovrà essere incapsulata ne nel bottone ne nel led ma piuttosto in un componente separato, un BLController (può essere visto come un mediator gof).
Il controller riceve dal bottone, del quale è osservatore, notifiche relative ai cambiamenti di stato del tipo “pressed”, “released”. In corrispondenza di tali notifiche, il controller aggiornerà il suo stato interno e deciderà se inviare o meno un comando al led (un comando può essere visto come un messaggio che non prevede una risposta). Il comportamento del BLController può essere descritto usando una fsm:

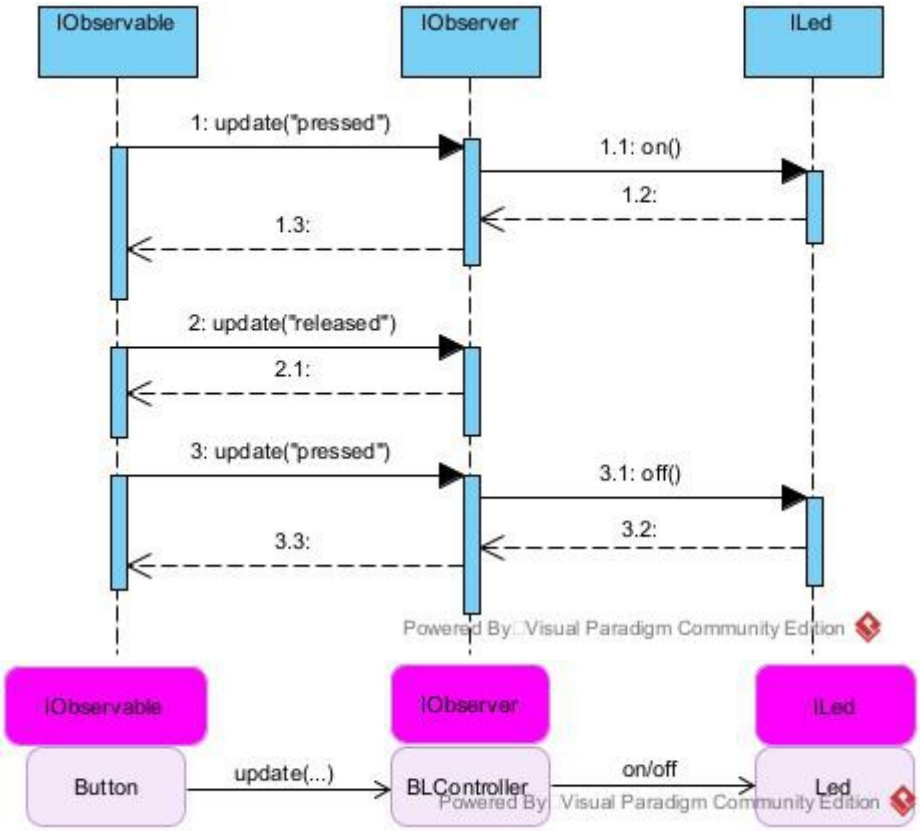


Il comportamento del controllore potrebbe essere descritto da una macchina più semplice, dotata di due soli stati :



Architettura logica

L'architettura logica del sistema può essere descritta usando il seguente diagramma di sequenza:



Il sistema è composto da tre componenti principali: un bottone osservabile, un controller observer che incapsula la logica applicativa ed un led.

Test Plan

A partire dagli scenari documentati nella fase di analisi, è possibile definire una serie di test. In realtà si potrebbe anche affermare che gli scenari vengono descritti per poter definire test che ci consentano di capire se i requisiti funzionali siano soddisfatti oppure no.
Un possibile test potrebbe essere il seguente: a partire da un led spento, procediamo premendo il bottone e verificando, successivamente, che il led si sia acceso. A partire da questa nuova situazione, potremmo premere nuovamente il bottone e verificare che il led si sia spento...e così via per alcune altre iterazioni. Il difetto di questa soluzione è che richiede l'intervento di una persona fisica per la pressione del bottone e per la verifica visiva dello stato del led. Sarebbe meglio poter automatizzare il processo di testing per quanto possibile evitando l'intervento umano. Questo comporterà l'introduzione di componenti fittizi, logici, non legati ad alcuna tecnologia particolare. Tali componenti vengono anche chiamati mock objects o test doubles.

I test possono essere formalizzati usando un framework per il testing automatico come JUnit per Java.
Possiamo provare ad impostare un primo prototipo funzionante del sistema in questione. Il prototipo può servire a comprendere meglio i requisiti e per scoprirne di nuovi.

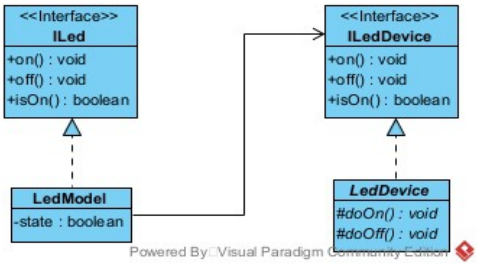
Progetto

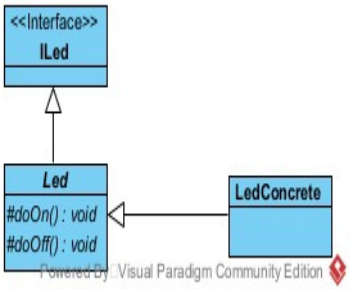
Lo scopo della fase di progettazione consiste nel raffinamento dell'architettura logica del sistema (zooming).

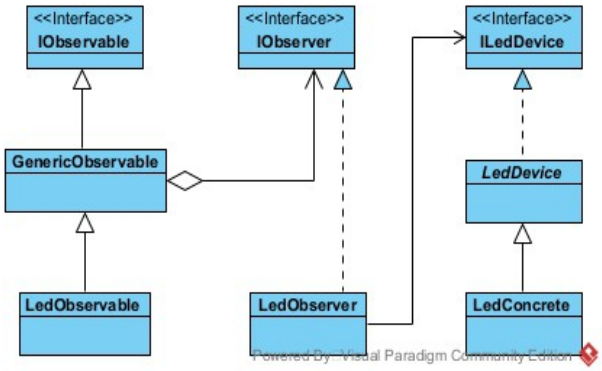
Iniziamo progettando i componenti di base del dominio.

Il led

Un led è un dispositivo di output binario, un attuatore luminoso (ha solo due stati). Introduciamo la classe LedModel che implementa l'interfaccia ILed. Questa implementazione non è legata a nessuna tecnologia particolare e rappresenta un led dal punto di vista logico. In ogni caso sarà necessario pilotare un led concreto. Seguendo il principio “favorire la composizione all'ereditarietà” possiamo iniettare nel led logico un led concreto (pattern bridge). In questo modo, la parte logica e quella concreta potranno evolvere in maniera completamente indipendente. In alternativa avremmo potuto definire il LedModel come classe astratta con metodi doOn() e doOff() astratti (template method) e lasciare l'implementazione concreta alle sue sottoclassi. In questo caso il modello e la parte concreta saranno strettamente legate. Un'altra alternativa sarebbe quella di considerare il led logico come un osservabile osservato da un opportuno LedObserver all'interno del quale sarebbe stata iniettata una implementazione concreta del led. Seguono i diagrammi strutturali relativi ai tre modelli sopra citati:



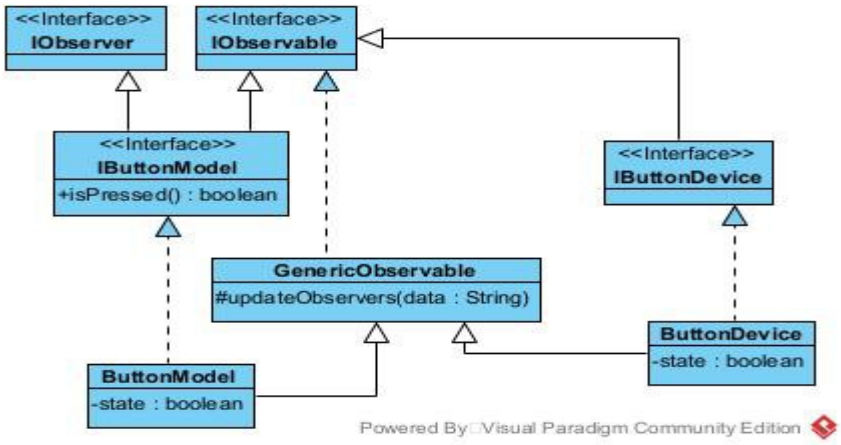




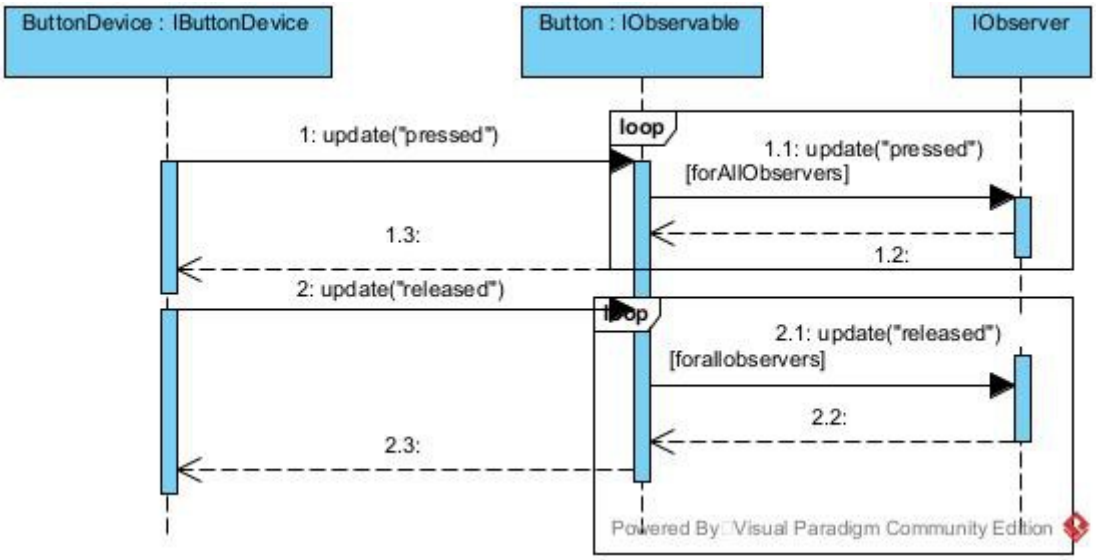
Il Bottone

Struttura

Introduciamo la classe ButtonModel che realizza le operazioni di IObservable ed IObserver. Lo stato interno è stato rappresentato usando una variabile booleana che assumerà il valore vero quando il bottone sarà premuto ed il valore falso quando il bottone sarà rilasciato. Lo stato interno dipende da quello di un dispositivo di livello più basso che può essere realizzato in diversi modi (cioè il bottone concreto, quello usato dall'utente). Per questo motivo il modello è osservatore di un osservabile che sarà un bottone concreto (parte logica e parte concreta legate attraverso il pattern observer). I bottoni concreti possono essere formalizzati usando l'interfaccia IButtonDevice che estende IObservable. Ne risulta la seguente struttura:

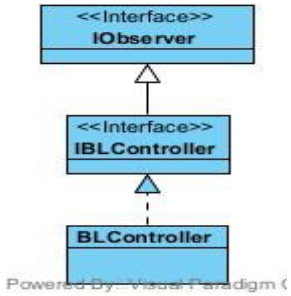


Di seguito il diagramma di sequenza che descrive le interazioni logiche fra bottone concreto, bottone logico e una qualche altra parte del sistema (un observer).

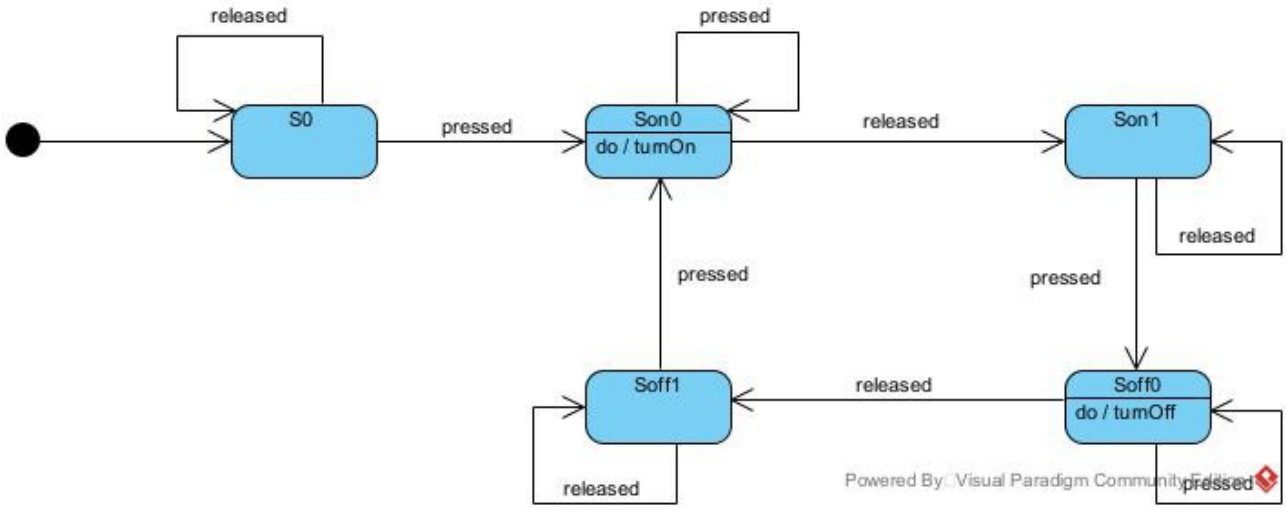


Il Controller

Struttura



Il controller incapsula la logica aziendale (core) ovvero quella cosa che da valore all'applicazione.
La classe BLController incapsula una implementazione della seguente macchina a stati finiti:



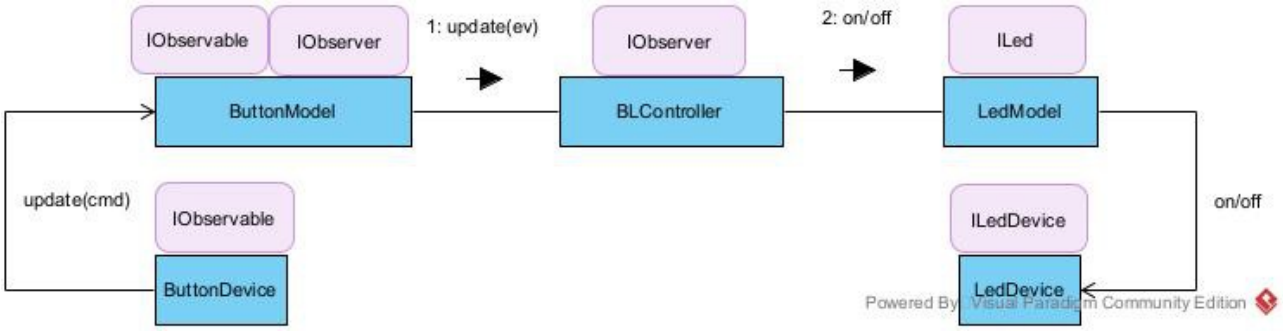
Possiamo usare direttamente un codice per esprimere in maniera formale (cioè comprensibile ad una macchina) il comportamento e la struttura del controller:

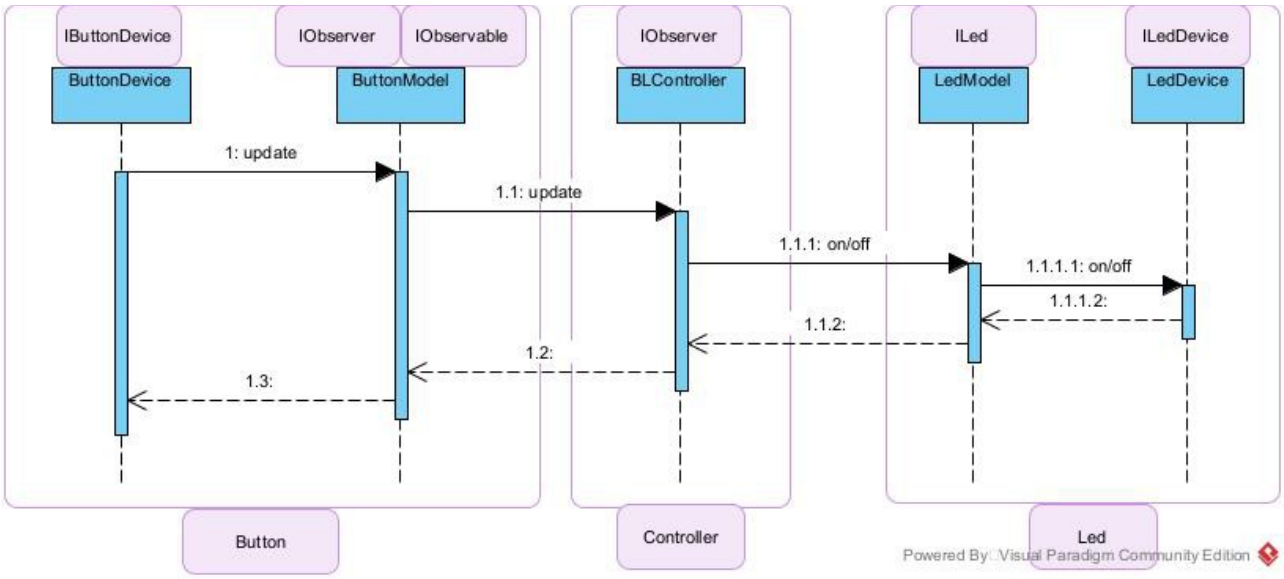
```
public interface IObserver{

	void update(String data);

}

public interface IBLController extends IObserver{
}

public class BLController extends GenericObserver implements IBLController {

	private String curState = "s0";
	private ILed led;

	@Override
	public void update(String data) {
		if(data.equals("pressed"))
			onPressed();
		else if(data.equals("released"))
			onReleased();
	}

	private void onPressed() {
		if(curState.equals("s0")) {
			turnOnLed();
			setState("sOn0");
		} else if(curState.equals("sOn1")) {
			turnOffLed();
			setState("sOff0");
		} else if(curState.equals("sOff1")) {
			turnOnLed();
			setState("sOn0");
		}
	}
	private void onReleased(){
		if(curState.equals("sOn0")) {
			setState("sOn1");
		} else if(curState.equals("sOff0")) {
			setState("sOff1");
		}
	}
	private void setState(String state) {
		this.curState = state;
	}
	private void turnOffLed() {
		led.off();
	}
	private void turnOnLed() {
		led.on();
```

Quella sopra rappresenta una possibile implementazione del controller come macchina a stati finiti. Avremmo anche potuto usare il pattern state (oppure definire un linguaggio interno per descrivere macchine a stati finiti e definire il controller usando tale linguaggio (progetto fsm)).

Prototipo del sistema button led

Modello delle interazioni

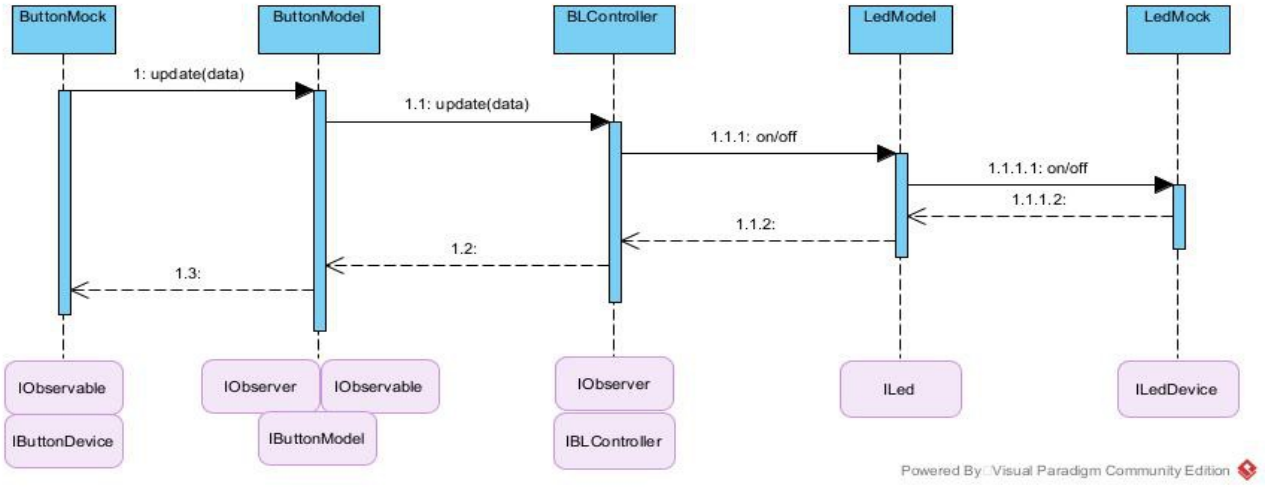




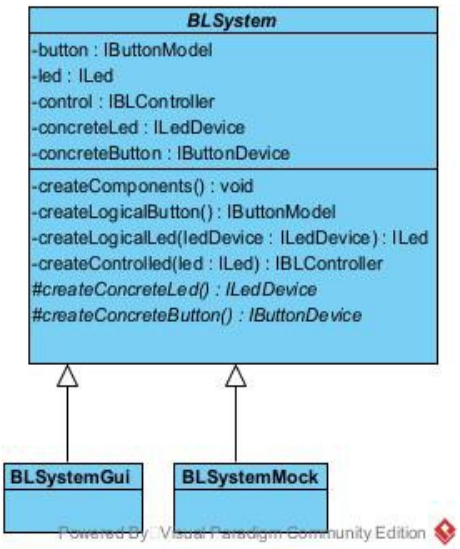
Progetto buttonLed con dispositivi mock (mock)

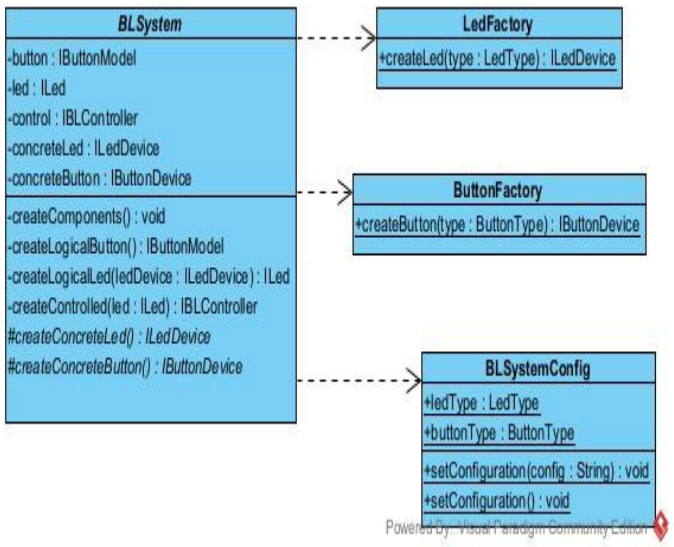
Introduco due componenti, ButtonMock e LedMock. Il led mock è un LedDevice che stampa su stdout una propria rappresentazione. Il ButtonMock è un ButtonDevice che simula il comportamento di un bottone reale.

Modello delle interazioni



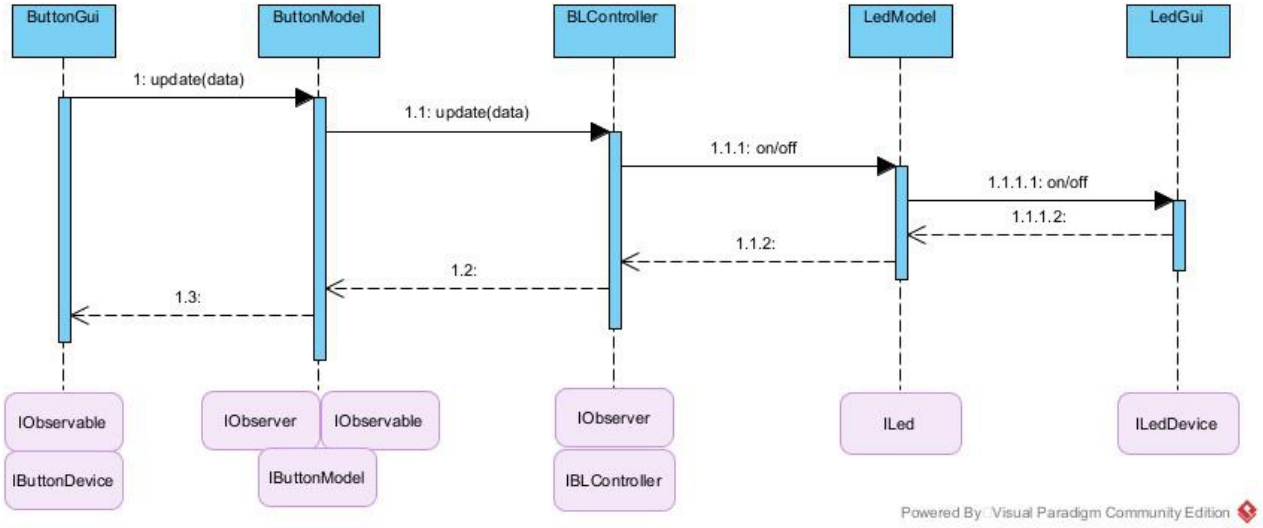
Per costruire il sistema dobbiamo prima creare i componenti concreti poi i componenti logici ed il controller iniettando all'interno di essi i componenti concreti. Questa logica può essere incapsulata all'interno di un componente che possiamo chiamare BLSystem (oppure MainBLSystem). I componenti concreti possono essere creati in sottoclassi di BLSystem attraverso un factory method (una sottoclasse per ogni possibile configurazione). Una soluzione più flessibile consiste nell'introdurre fattorie per i dispositivi ed una configurazione. La configurazione può essere salvata all'interno di un file che verrà letto all'avvio del sistema.



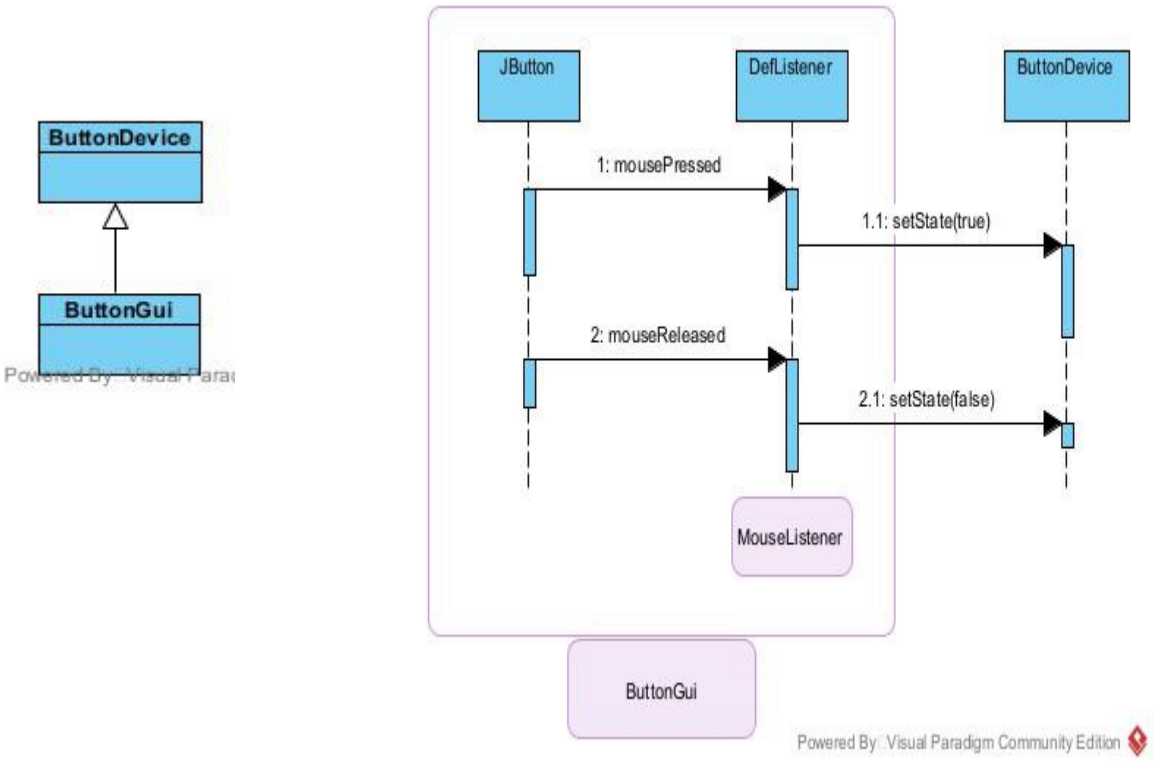


Progetto buttonLed con dispositivi virtuali (gui)

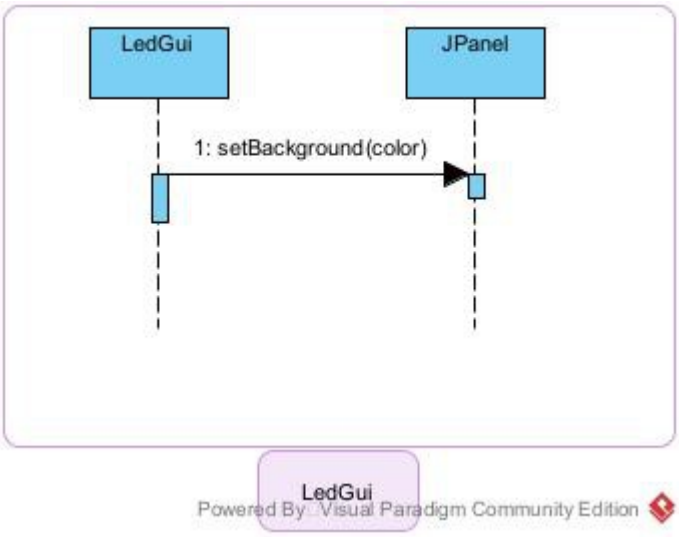
Per costruire un bottone grafico (in java) si dovrà utilizzare un framework per la costruzione di interfacce grafiche. Il framework aumenta il livello di astrazione rispetto alla macchina e dunque contribuisce a ridurre il gap fra il problema e la macchina. Il framework Swing mette a disposizione la classe JButton che realizza un bottone grafico come osservabile. E' possibile associare un listener ad un JButton in modo tale che il bottone possa notificarlo ogni qualvolta il suo stato cambi (pattern observer). Tutta la logica relativa alla creazione del bottone grafico potrebbe essere incapsulata all'interno del componente ButtonGui che estende ButtonDevice.



I componenti swing possono essere incapsulati all'interno della classe ButtonGui.



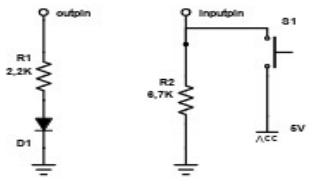
Il led virtuale, può essere modellato come un pannello che cambia colore in base al suo stato interno. Quando acceso assumerà il colore onColor mentre quando verrà spento esso assumerà il colore offColor. In Java è possibile costruire un tale pannello riusando la classe JPanel del framework swing e la classe Color del framework awt. Quando il led riceve il comando on() si imposta il colore del pannello ad onColor. Quando il led riceve il comando off() si imposta il colore del pannello ad offColor. Questa logica può essere incapsulata all'interno dell'astrazione LedGui:



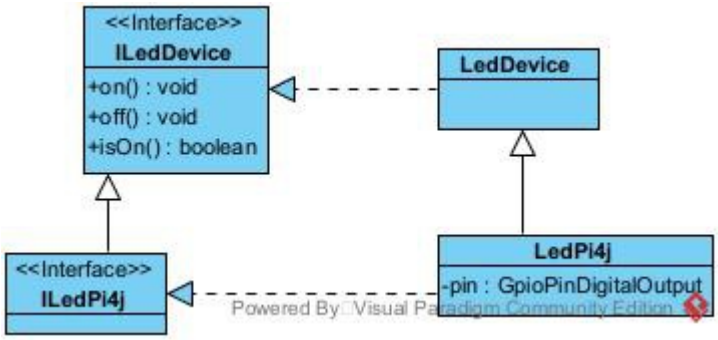
Per avviare un tale sistema è sufficiente modificare il file di configurazione impostando i tipi dei dispositivi come “gui”.

ButtonLed System per piattaforma raspberry

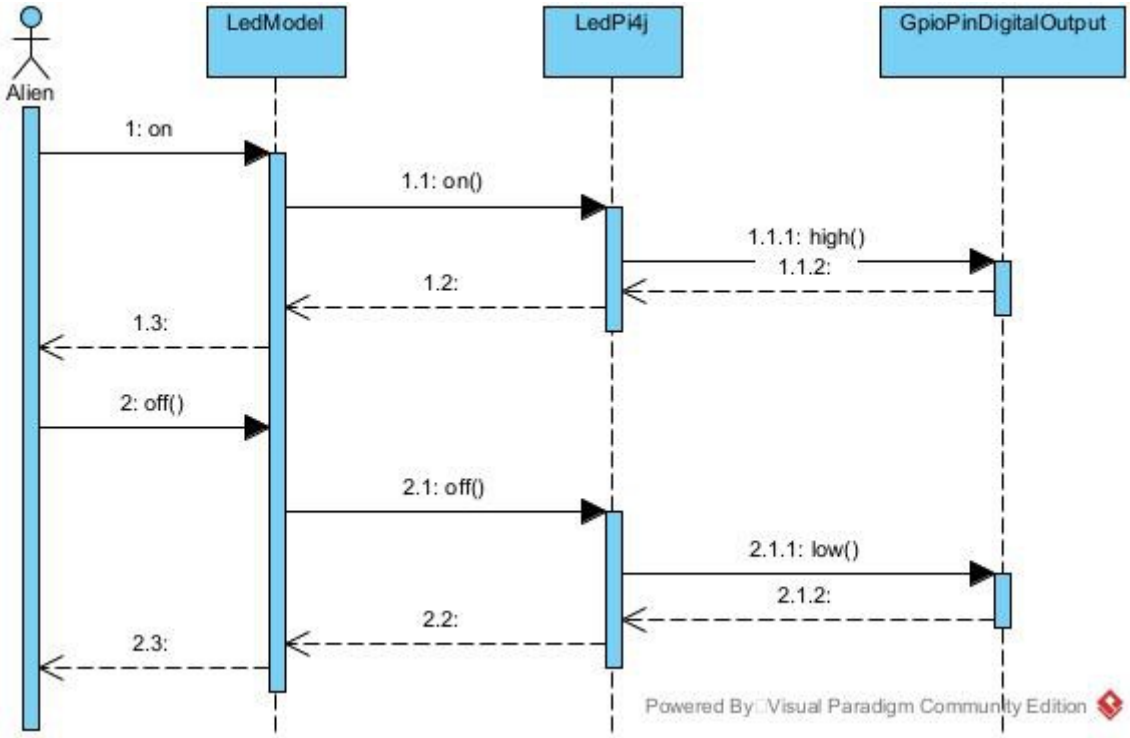
Un qualsiasi bottone fisico deve essere collegato ad un pin di un microcontroller mediante un qualche circuito di interfaccia. Stessa cosa vale per un led fisico. La cosa vale anche per tanti altri dispositivi, come sensori, motori, display i quali vanno collegati ad uno o più pin di un controller attraverso un circuito di interfaccia. Segue lo schema del circuito usato nel progetto:



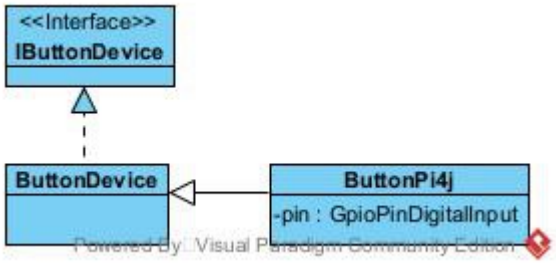
Che cosa bisogna fare per controllare un led fisico? Basta modificare lo stato logico del pin al quale è attaccato. Per accedere al GPIO di un raspberry, è possibile fare uso della libreria Pi4j (costruita sopra la libreria wiringPi scritta in C). Tale libreria funge da intermediario fra le applicazioni e il sistema operativo (riduce il gap verso la macchina, aumenta il livello di astrazione rispetta alla macchina). La documentazione di pi4j ci dice che per interagire con la libreria bisogna creare un'istanza di GpioController. Questo può essere fatto usando la fattoria GpioFactory che consente di creare una istanza di GpioController (interfaccia). Una volta ottenuto il controller, questi ci permette di creare diverse tipologie di pin (in out digitali, in out analogici, pwm). A noi interessano le astrazioni GpioPinDigitalOutput per il led e GpioPinDigitalInput per il bottone. Dunque per accendere un led è sufficiente chiamare il metodo high( ) di un gpioPinDigitalOutput e per spegnerlo basterà invocare l'operazione low( ) (le due operazioni servono a modificare lo stato logico del pin associato). Questa logica può essere incapsulata all'interno della classe LedPi4j secondo il seguente diagramma:

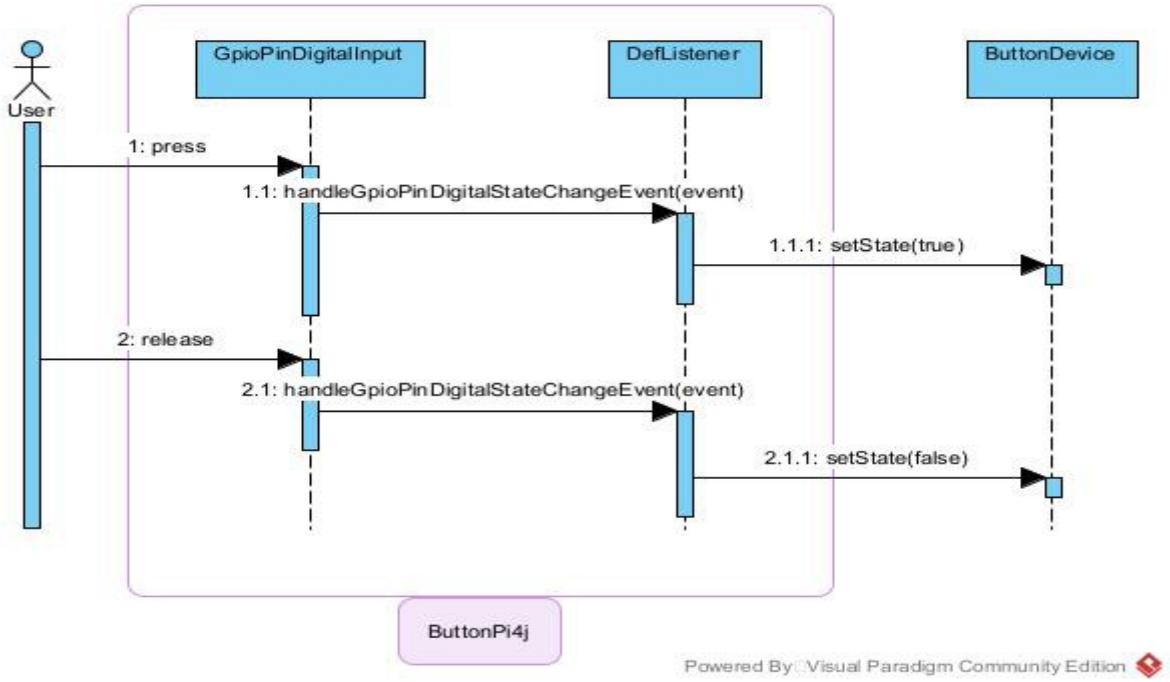


Seguono le interazioni



Ora consideriamo il bottone. Quando premuto, lo stato logico del pin associato diventa alto mentre quando il bottone viene rilasciato lo stato del pin assume il valore logico basso. In generale, un pin di input può essere gestito a polling oppure ad interrupt. La libreria pi4j ci consente di registrare un listener ad un GpioPinDigitalInput, il che equivale ad una gestione ad interrupt. Analogamente a quanto fatto per il led, otteniamo dapprima una istanza di GpioPinDigitalInput mediante la factory GpioController. Dopodiché possiamo registrare presso il pin un listener DefListener che si occuperà di modificare lo stato interno dell' IButtonDevice. Di seguito gli schemi, strutturale e interattivo:





A livello pratico, il bottone crea problemi dovuti al rimbalzo del segnale (problema noto). Quando la tensione di un pin viene modificata improvvisamente (transizione alto-basso oppure basso-alto) , il segnale comincia ad oscillare per un po' prima di stabilizzarsi verso il livello desiderato, rendendo, di fatto, lo stato del pin flottante (cioè ne alto ne basso). Il rimbalzo può essere gestito via hardware mediante un apposito circuito, via software oppure usando una soluzione ibrida hardware software. Useremo un approccio software (una volta rilevato un cambiamento nello stato del pin, si esegue una lettura, si attende un pochino (20-30 ms), quindi si effettua una nuova lettura. Le due letture devono coincidere affinché la transizione sia valida.).
Per istanziare il sistema per piattaforma raspberry, una volta codificati i 2 nuovi componenti, sarà sufficiente modificare il file di configurazione specificando come tipo 'raspberry', sia per il bottone che per il led (sono state aggiornate le factory dei device del dominio).

ButtonLed System per piattaforma arduino

In questo caso siamo costretti a cambiare linguaggio di programmazione passando da Java a C++ (oppure C). La struttura del sistema rimane immutata rispetto ai casi precedenti. Per pilotare un led la piattaforma arduino fornisce le funzioni pinMode e digitalWrite. La funzione pinMode prende due parametri in ingresso che specificano il numero del pin da impostare e la direzione INPUT o OUTPUT (nel caso di un led avremo pinMode(ledPin, OUTPUT)). Una volta configurato il pin come pin di uscita, possiamo, attraverso digitalWrite, modificare il suo stato logico impostandolo ad HIGH per accendere il led oppure LOW per spegnere il led. Questa logica è incapsulata all'interno della classe LedArduino che estende LedDevice. Il bottone è stato gestito facendo uso delle interruzioni. Le funzioni di nostro interesse sono pinMode, digitalRead e attachInterrupt. Il pin a cui il bottone viene connesso dovrà essere impostato come pin di ingresso usando la funzione pinMode. Dobbiamo poi fare in modo che ad ogni cambiamento dello stato del pin venga generata una interruzione e mandata in esecuzione una opportuna ISR. Questo risulta possibile usando la funzione attachInterrupt passando tre parametri fra cui un puntatore alla funzione ISR. All'interno della ISR è stato implementato l'antirimbalzo. Questa logica è stata incapsulata all'interno della classe ButtonArduino che estende ButtonDevice.

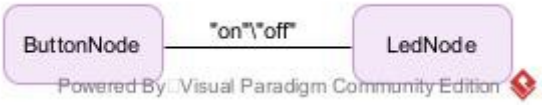
Sistema ButtonLed distribuito

Consideriamo adesso il seguente requisito espresso come storia:
Come utente vorrei poter premere e rilasciare un bottone in modo che un led remoto si accenda e spenga ad ogni pressione del bottone. Da led spento quando il bottone viene premuto il led si accende. Premendo di nuovo il bottone il led si spegnerà e così via...

Che cosa si intende per led remoto? Per led remoto si intende un led situato in un nodo computazionale diverso da quello in cui si trova il bottone. In questo caso, il sistema sarà composto da due nodi computazionali: in uno risiede il bottone e nell'altro risiede il led e dunque risulta essere un sistema distribuito.

Analisi del problema

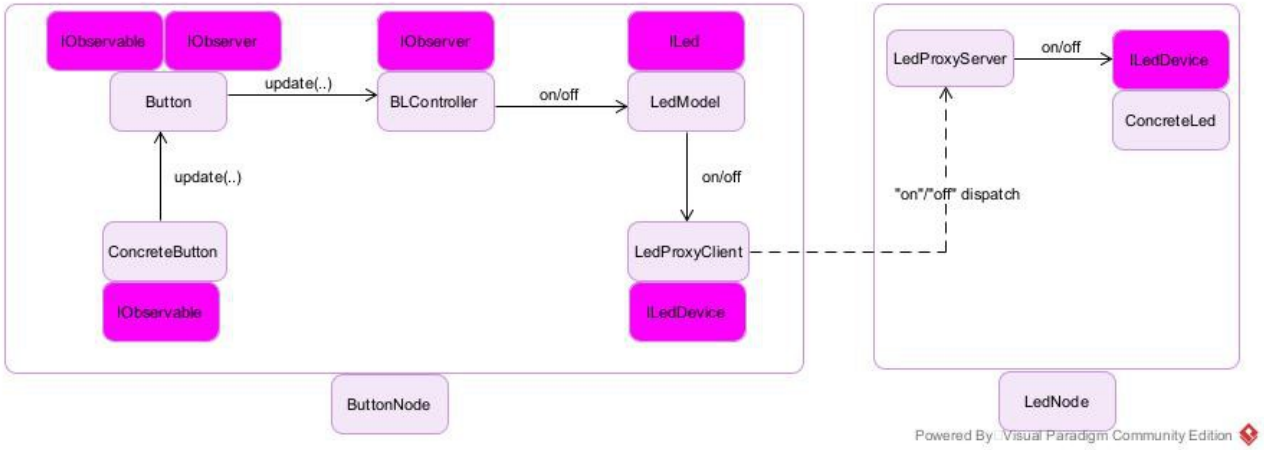
Consideriamo il caso in cui il led sia remoto. Dobbiamo analizzare il problema buttonLed con questo nuovo vincolo. Siamo in presenza di due nodi computazionali, in uno dei quali si troverà il bottone e nell'altro il led. Dove conviene allocare il controller? Abbiamo due possibilità: nel nodo del bottone o nel nodo del led. Per minimizzare le comunicazioni fra i due nodi, conviene allocare il controller nello stesso nodo in cui si trova il bottone.



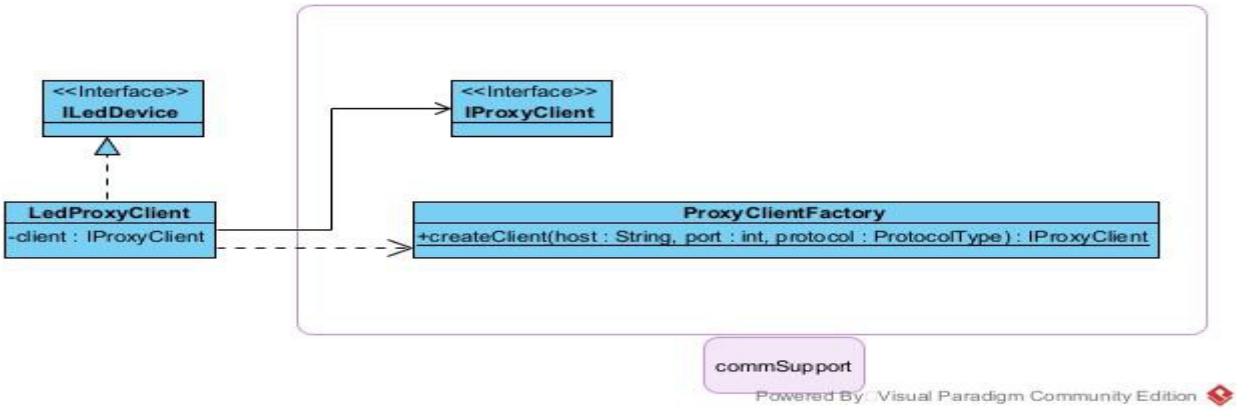
Consideriamo nuovamente la nostra architettura logica:

Nel caso concentrato, i componenti comunicano fra loro mediante chiamata a procedura. Nel caso distribuito, la chiamata a procedura non funziona più. La comunicazione dovrà avvenire per mezzo di messaggi (message-passing). Il problema principale da risolvere è quello legato alla comunicazione fra le due parti remote.
Mentre nel caso concentrato la comunicazione fra il controller ed il led avviene mediante chiamata a procedura, nel caso distribuito questo non sarà possible. I due componenti dovranno comunicare scambiandosi messaggi. La comunicazione può avvenire in diversi modi: direttamente, usando un protocollo standard come TCP o UDP oppure indirettamente,usando un intermediario (un middleware di comunicazione, una coda di messaggi) che dovrà risiedere da qualche parte ovvero in uno dei due nodi del sistema oppure in un nodo terzo. Inoltre dovrebbe esserci accordo sulla struttura delle informazioni scambiate fra le parti. Questo potrebbe essere parzialmente vero in quanto è possibile introdurre degli opportuni trasformatori (che convertiranno un linguaggio in un altro). In ogni caso, da qualche parte, le chiamate ad on( ) e off( ) dovranno essere trasformate in "messaggi" e questi ultimi dovranno poi essere trasferiti, attraverso l'infrastruttura di rete, al nodo remoto. I componenti a nostra disposizione non sono capaci di comunicare attraverso messaggi ma il nostro problema lo richiede. Questo crea un gap fra il problema e la macchina che deve essere colmato in qualche modo. Il problema spinge verso la necessità di disporre di componenti capaci di comunicare attraverso messaggi. Il nostro led remoto potrebbe essere wrappato all'interno di un

componente capace di ricevere messaggi da un client remoto. Dall'altra parte del mare il nostro led dovrà inviare alla sua controparte remota messaggi del tipo on off. Possiamo introdurre il componente LedProxyClient di interfaccia ILedDevice (pattern proxy)
Una delle due parti assumerà il ruolo di server mentre l'altra il ruolo di client. Ad offrire il servizio di accensione e spegnimento è il led remoto e dunque sembra ragionevole assegnare il ruolo di server al nodo led.
Il led remoto accetta due messaggi, on e off che possono essere formalizzati usando semplici stringhe come ad esempio “on” e “off”.
Il led device come proxy, viene suddiviso in due parti capaci di comunicare tra di loro usando un protocollo di qualche tipo (pattern half object + protocol).
Ora passiamo ad analizzare il problema LedProxyClient. Il proxy client dovrà essere in grado di trasformare le chiamate a procedura in messaggi e quindi di inviare questi messaggi alla parte remota usando un protocollo di comunicazione. Si tratta di risolvere il problema dell'invio (ed in generale anche della ricezione) delle informazioni attraverso la rete. Il seguente diagramma di interazione illustra quanto detto sopra:

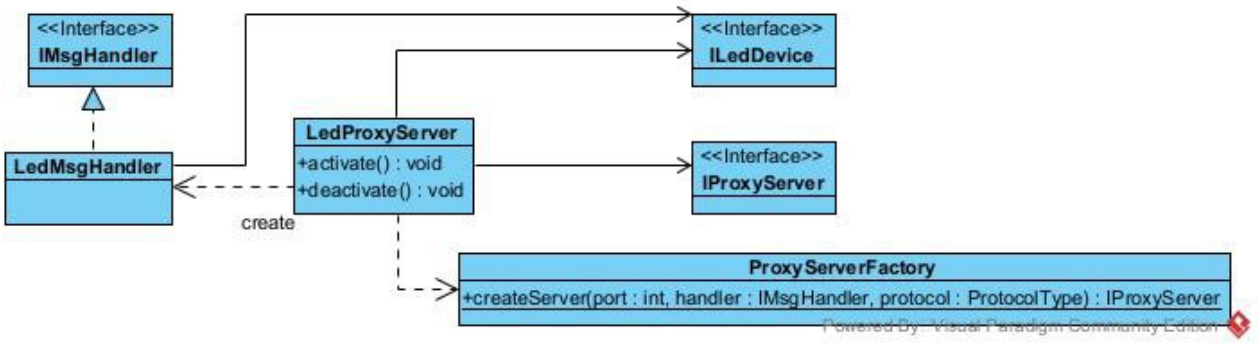


Dopo alcune iterazioni di progettazione/implementazione si è giunti al seguente modello:

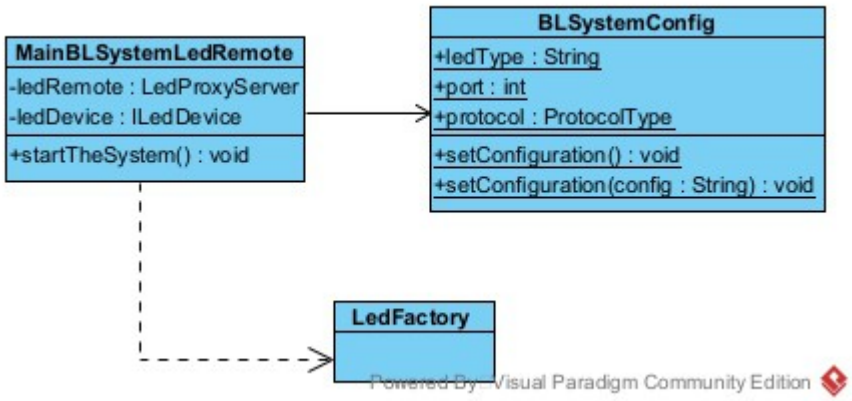


E' stato creato un nuovo progetto commSupport il quale fornisce supporto alle comunicazioni.
Ora passiamo alla parte server e cioè al LedProxyServer. Dovrà essere un componente attivo (dotato di un flusso di controllo proprio) che attende richieste da parte del led client e le trasforma in comandi per il led di basso livello.

Dopo alcune iterazioni di progettazione/implementazione si è giunti al seguente modello:

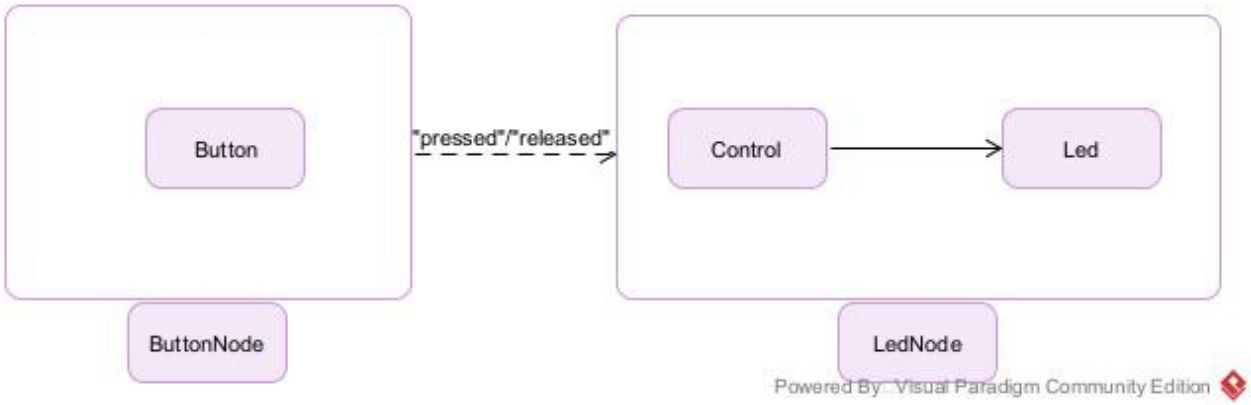


E' stato aggiornato anche il progetto commSupport introducendo il supporto lato server. E' stato aggiornato anche il progetto bls.domain, in particolare le fattorie, per poter creare i componenti proxy.
A questo punto possiamo costruire il nostro sistema buttonLed con led remoto. Vanno costruite le due parti da deployare sui due nodi. La parte principale, cioè quella in cui si trova la logica aziendale, può essere costruita semplicemente modificando il file di configurazione BLSystemConfig.json impostando a “proxy” il parametro ledType. Per ciò che concerne la parte relativa al led concreto remoto siamo giunti al seguente modello:

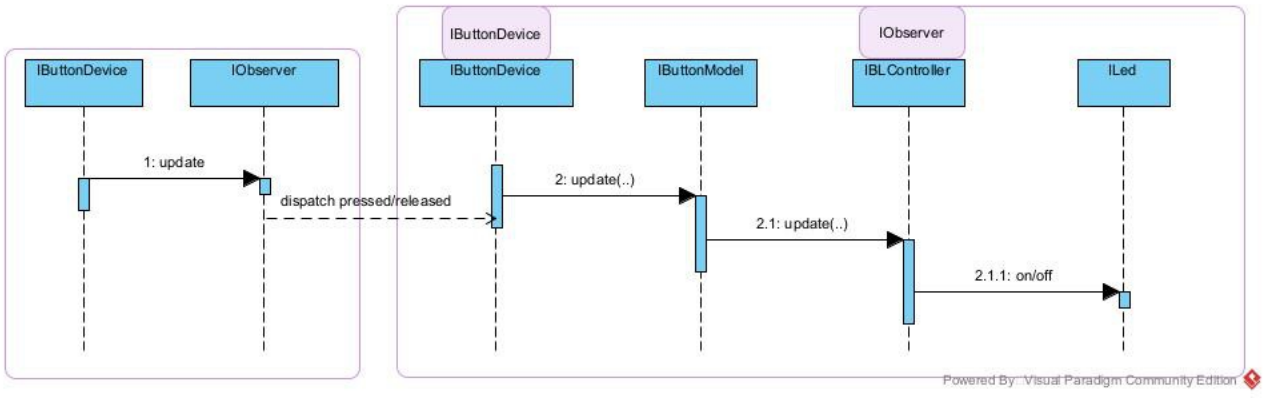


Progetto/Analisi del problema ButtonLedSystem con bottone remoto

Consideriamo il caso in cui sia il bottone ad essere remoto. Come nel caso precedente, per bottone remoto si intende un bottone situato in un nodo computazionale diverso da quello in cui è situato il led. Il sistema si troverà diviso in due parti, una che comprende il solo bottone e l'altra comprensiva di controllo e led.

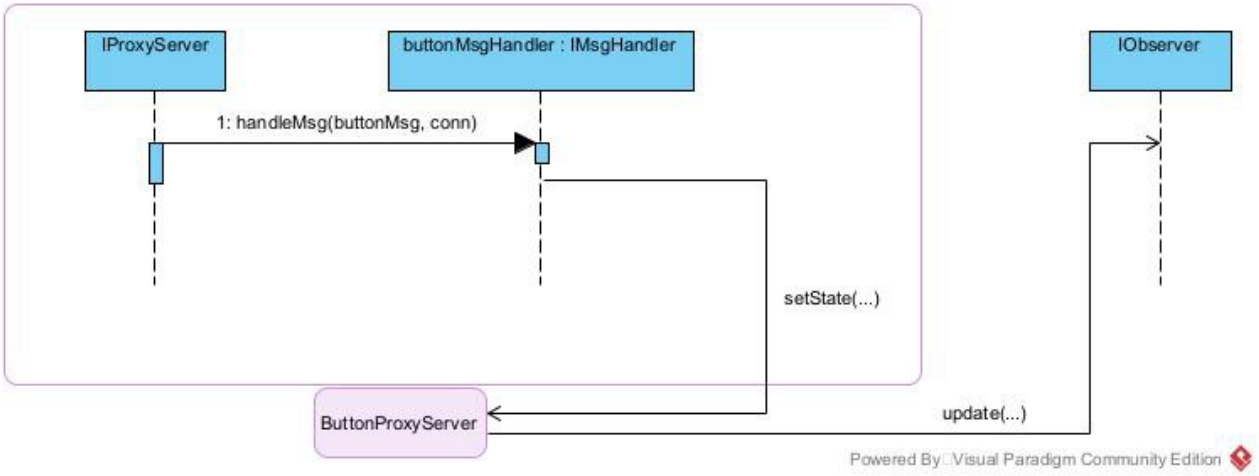


Il bottone a nostra disposizione usa la chiamata a procedura per comunicare con altri componenti. Per risolvere il nostro problema (BLSystemButtonRemote), il bottone remoto dovrà inviare verso il LedNode messaggi del tipo “pressed”, “released”. Del resto all'interno del LedNode dovrà esserci un componente capace di ricevere tali messaggi. Nel nostro sistema dovrà essere quindi disponibile un componente surrogato di un bottone, capace di ricevere messaggi, nel senso del message passing, prodotti da un bottone remoto. I componenti già disponibili non consentono di attuare tale tipo di comportamento (abstraction gap). Dal punto di vista logico la situazione può essere descritta dal seguente diagramma di interazione:

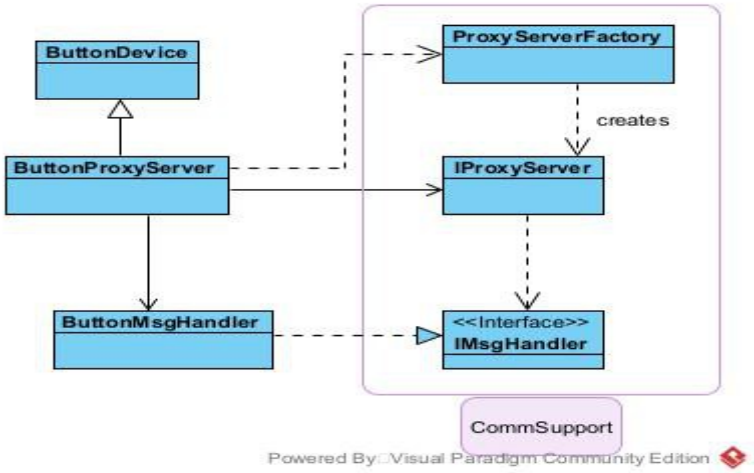


Progetto/Implementazione

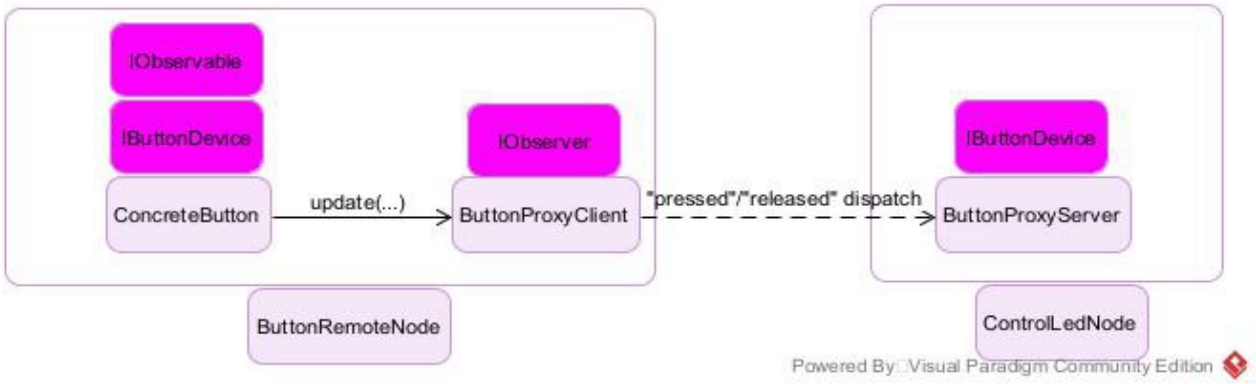
Introduciamo la classe ButtonProxyServer come estensione di ButtonDevice. Sarà responsibile di ricevere messaggi da clienti remoti, elaborare tali messaggi e notificare il modello. La sua struttura interna può essere descritta dal seguente modello di interazione:



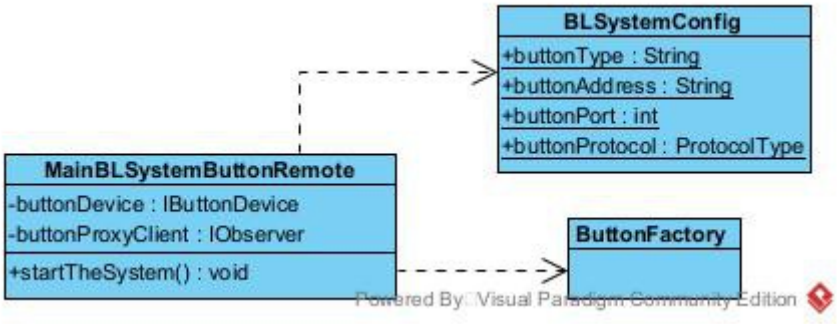
Si è fatto uso del supporto alle comunicazioni che nasconde molti dettagli e rende l'implementazione del ButtonProxyServer molto semplice.



Possiamo ora costruire il sistema intero integrando i suoi vari componenti come fatto in precedenza per gli altri sistemi. La struttura è sempre la stessa, cambiano solo le implementazioni. E' sufficiente modificare il file di configurazione BLSystemConfig.json impostando il valore del parametro buttonType a “proxy”. Per la parte remota, la struttura segue il seguente modello di interazione:

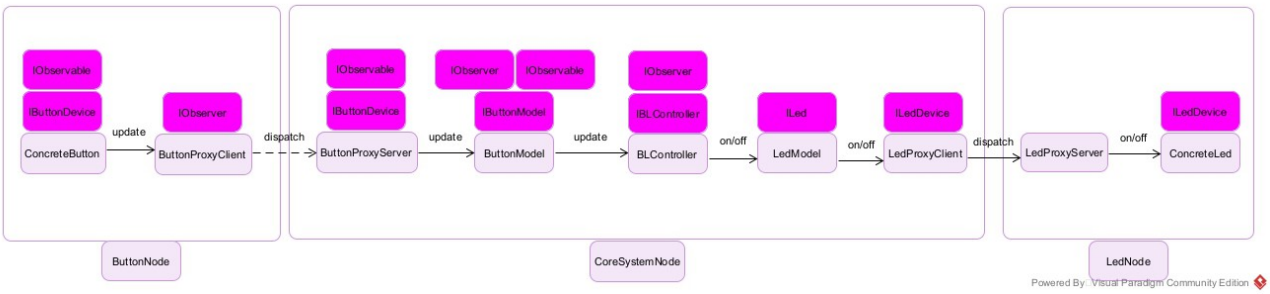


E' stato introdotto il componente ButtonProxyClient come osservatore del bottone concreto. Il ButtonProxyClient usa, internamente, un IProxyClient del supporto alle comunicazioni. Questo gli conferisce la capacità di inviare messaggi al core system. Di seguito la classe MainBLSystemButtonRemote che costruisce la parte del sistema relativa al bottone concreto remoto.



Progetto ButtonLedSystem con bottone e led remoti

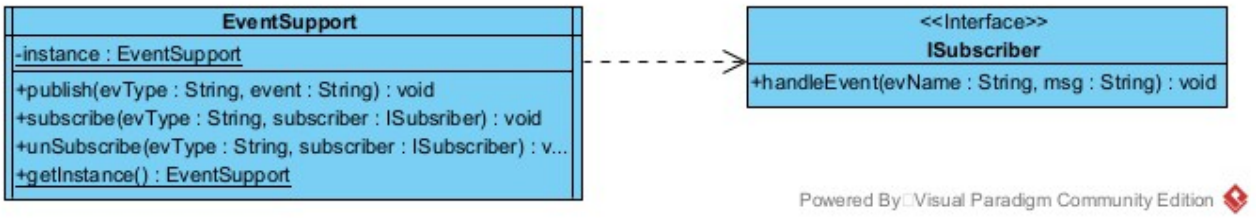
E' possibile considerare anche il caso in cui sia il bottone che il led siano remoti. L'architettura può essere descritta dal seguente modello di interazione:



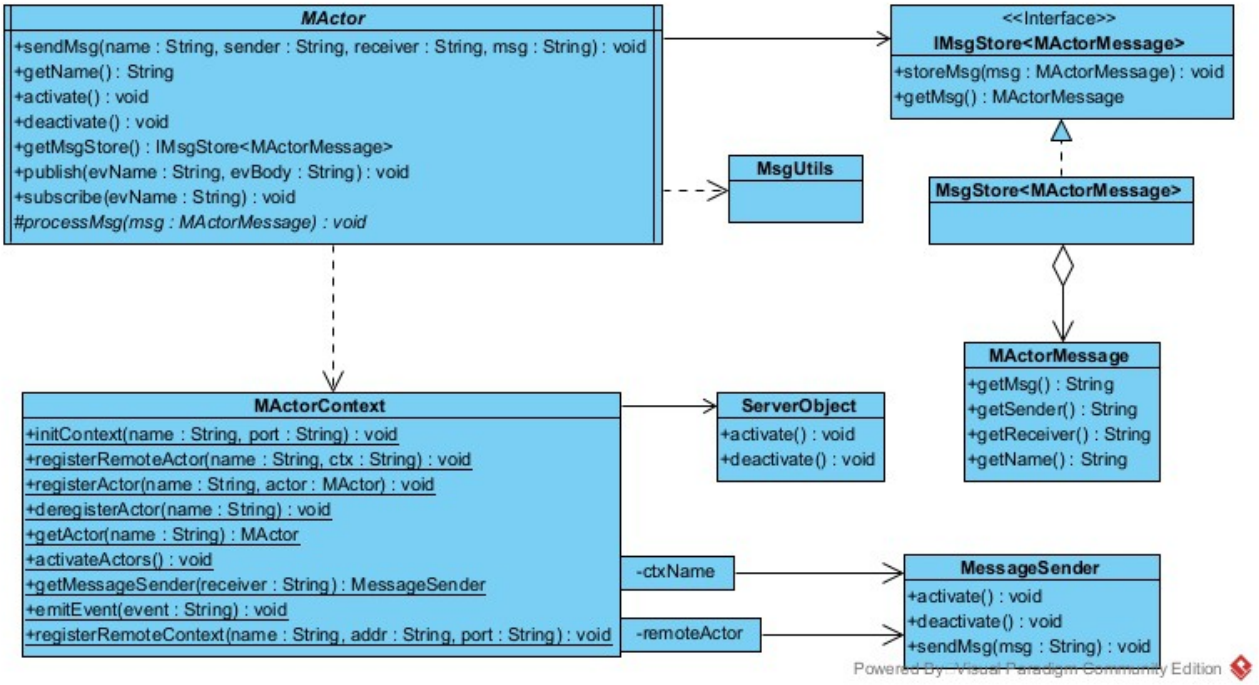
Il problema buttonLed distribuito mostra che lo spazio concettuale object oriented non è completamente adeguato alla costruzione di sistemi distribuiti. In tale spazio, i singoli oggetti possono comunicare solo attraverso la chiamata a procedura (sebbene il concetto iniziale di oggetto fosse diverso e cioè oggetto come entità autonoma capace di comunicare usando messaggi nel senso del message passing) mentre in un sistema distribuito è necessaria la comunicazione basata su messaggi. Tutto sarebbe più semplice se potessimo disporre di astrazioni, di più alto livello rispetto a quello degli oggetti, capaci di comunicare fra loro usando messaggi asincroni trasferibili sia ad entità locali che remote. Inoltre, nel caso concentrato, il flusso di controllo parte dal bottone e viene trasferito agli altri componenti. Sarebbe meglio poter disporre di entità capaci di comunicare in maniera asincrona senza trasferimento di controllo. Questo spinge verso la definizione di una nuova entità computazionale che possiamo chiamare MActor. Possiamo in qualche modo vederla come un'evoluzione naturale del concetto di oggetto (nel senso object-oriented classico).

Cominciamo dal caso locale. Un MActor è un'entità capace di comunicare con altri MActor usando messaggi scambiati in maniera asincrona (comunicazione bufferizzata). L'invio di un messaggio presuppone la conoscenza del nome del destinatario. Ogni MActor è dotato di un flusso di controllo privato e di una coda usata per tenere memoria dei messaggi ricevuti i quali verranno elaborati in sequenza, uno alla volta. Ogni messaggio è descritto da un nome che ne identifica la richiesta, dal nome del mittente e del destinatario (che possono essere vuoti in alcuni pattern di interazione) e dal messaggio vero e proprio. Un MActor può anche pubblicare eventi cioè messaggi senza un esplicito destinatario e sottoscrivere interesse a ricevere certi tipi di eventi (pattern publish subscribe – può

essere visto come caso specializzato di observer- implementato usando un singleton EventSupport).

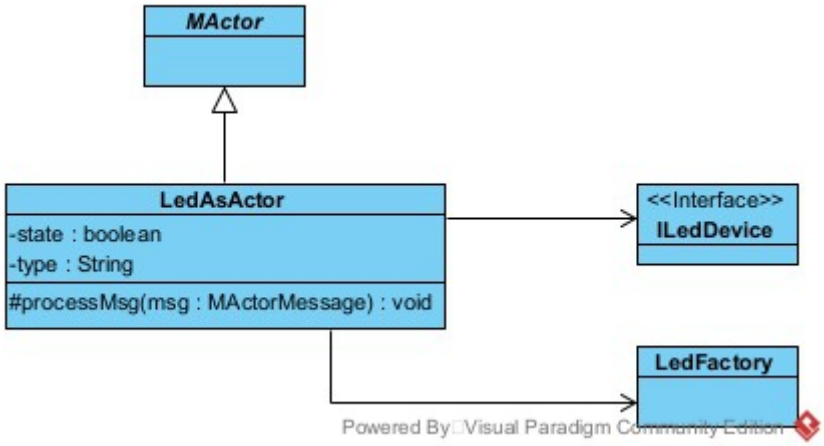


Ogni attore appartiene ad un MActorContext che rappresenta un contenitore di attori ed un nodo computazionale e fornisce operazioni di supporto.
Un attore che voglia inviare un messaggio ad un altro attore non farebbe altro che inserire il messaggio nella coda del destinatario nel caso in cui quest'ultimo appartenga allo stesso contesto. Per capirlo il mittente chiede informazioni al contesto. Se il destinatario fosse remoto allora il messaggio verrebbe propagato verso il contesto remoto a cui il destinatario appartiene usando un proxy (MessageSender). In un ambiante distribuito ogni contesto avvia un server tcp che consente di ricevere connessioni e messaggi da contesti remoti, mantiene un proxy per ogni altro contesto remoto ed una corrispondenza tra nomi degli attori e contesti di appartenenza. Gli eventi vengono pubblicati a tutti gli interessati locali e poi propagati verso tutti gli altri contesti remoti. Un sistema distribuito può essere concepito come un insieme di contesti che comunicano scambiandosi messaggi e dove ciascuno dei quali conterrà un certo numero di attori locali.

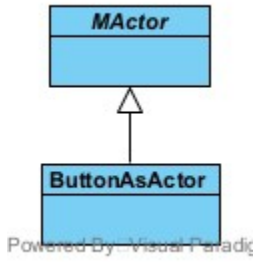


Facciamo un passo indietro e riconsideriamo il sistema buttonLed cercando di rendere il sistema asincrono e basato su messaggi. Possiamo partire dal componente Led. I messaggi a cui dovrà rispondere sono gli stessi definiti dall'interfaccia ILed. On ed off sono comandi e non necessitano di risposta mentre isOn è un predicato che prevede l'invio di una risposta al mittente (può essere visto come una richiesta alla quale deve seguire una risposta – interazione request-response). E' sempre dotato di stato binario. Questa volta è un componente attivo. Dobbiamo dare una rappresentazione ai messaggi. Possiamo usare semplici stringhe del tipo “on”, “off”, “isOn”. Quando il led riceve il messaggio “on” aggiorna il proprio stato interno e si accende. Quando riceve il messaggio “off”

aggiorna il proprio stato interno e si spegne. Quando riceve il messaggio “isOn”, valuta il proprio stato interno e invia al mittente la risposta “vero” oppure “falso”. Questo richiede la conoscenza del nome del mittente che può essere inserito all'interno del messaggio dal mittente stesso (che nel nostro caso sarà rappresentato dal controller (un BLControllerAsActor)). Questo nuovo modello di led può essere formalizzato usando la classe LedAsActor, estensione di MActor. Segue un diagramma strutturale:



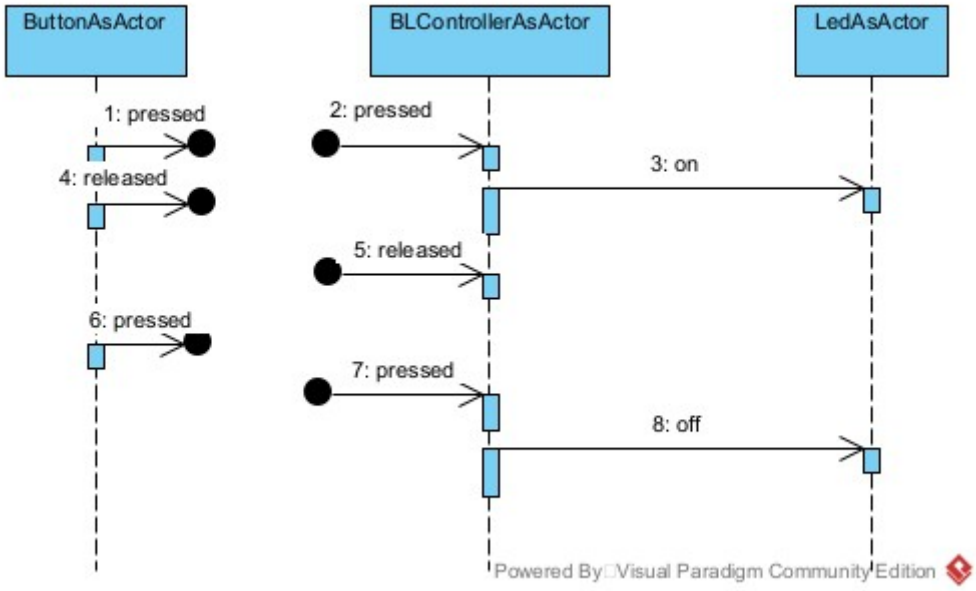
Passiamo ora al bottone. Un bottone è stato modellato come un'entità osservabile che ad ogni cambiamento del suo stato interno va propagando una notifica a tutti gli osservatori interessati. Nel nuovo modello un bottone può essere visto come un attore dotato di uno stato interno binario (come nel modello precedente) che pubblica un evento di tipo “buttonEv” asincrono ad ogni cambiamento del suddetto stato.

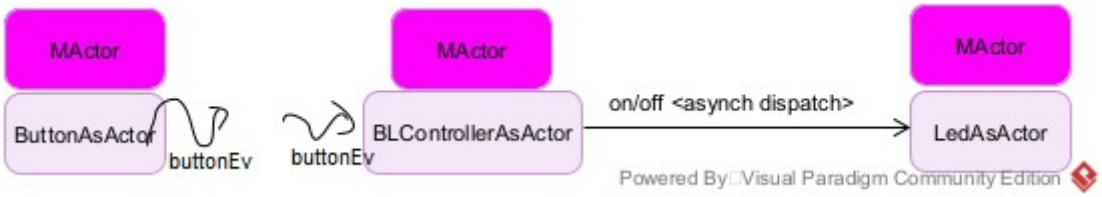


ButtonLedSystem con attori MActor

La logica applicativa del nostro button led system dovrà essere incapsulata all'interno di un apposito attore che chiameremo BLControllerAsActor. Tale attore sarà concepito non più come observer del

bottone bensì come subscriber di eventi di tipo “buttonEv” (emessi dal ButtonAsActor). Alla ricezione di un evento il controller dovrà decidere se inviare o meno un messaggio “on” oppure “off” al led. Possiamo descrivere l'architettura logica attraverso il seguente modello di interazione:

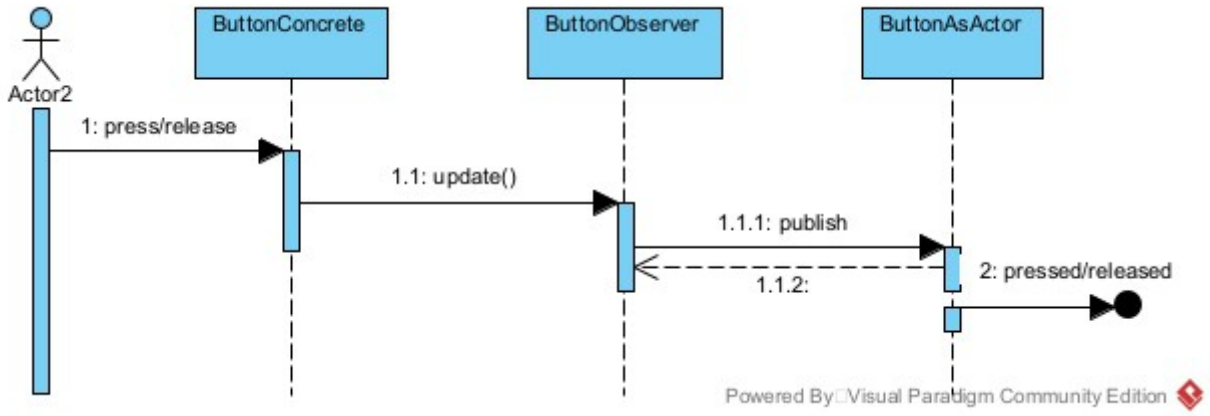


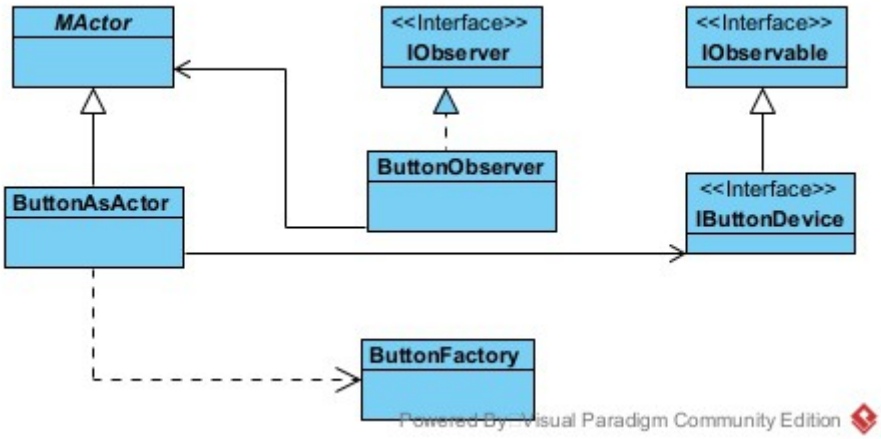


Progettazione

Progettazione bottone

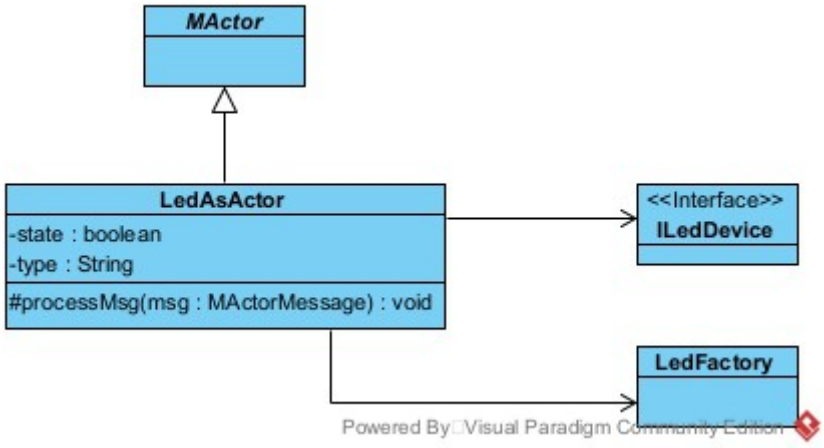
Quando il bottone viene creato, come tutti gli attori, registra se stesso presso il contesto a cui appartiene. Usando la factory ButtonFactory andrà creando l'IButtonDevice concreto ed un generico observer ButtonObserver passando se stesso come argomento. Quando il bottone concreto viene premuto o rilasciato, esso andrà notificando il button observer il quale, a sua volta, userà l'MActor ButtonAsActor per pubblicare l'evento corrispondente (pressed oppure released)

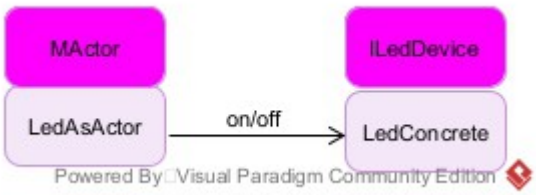




Progettazione Led

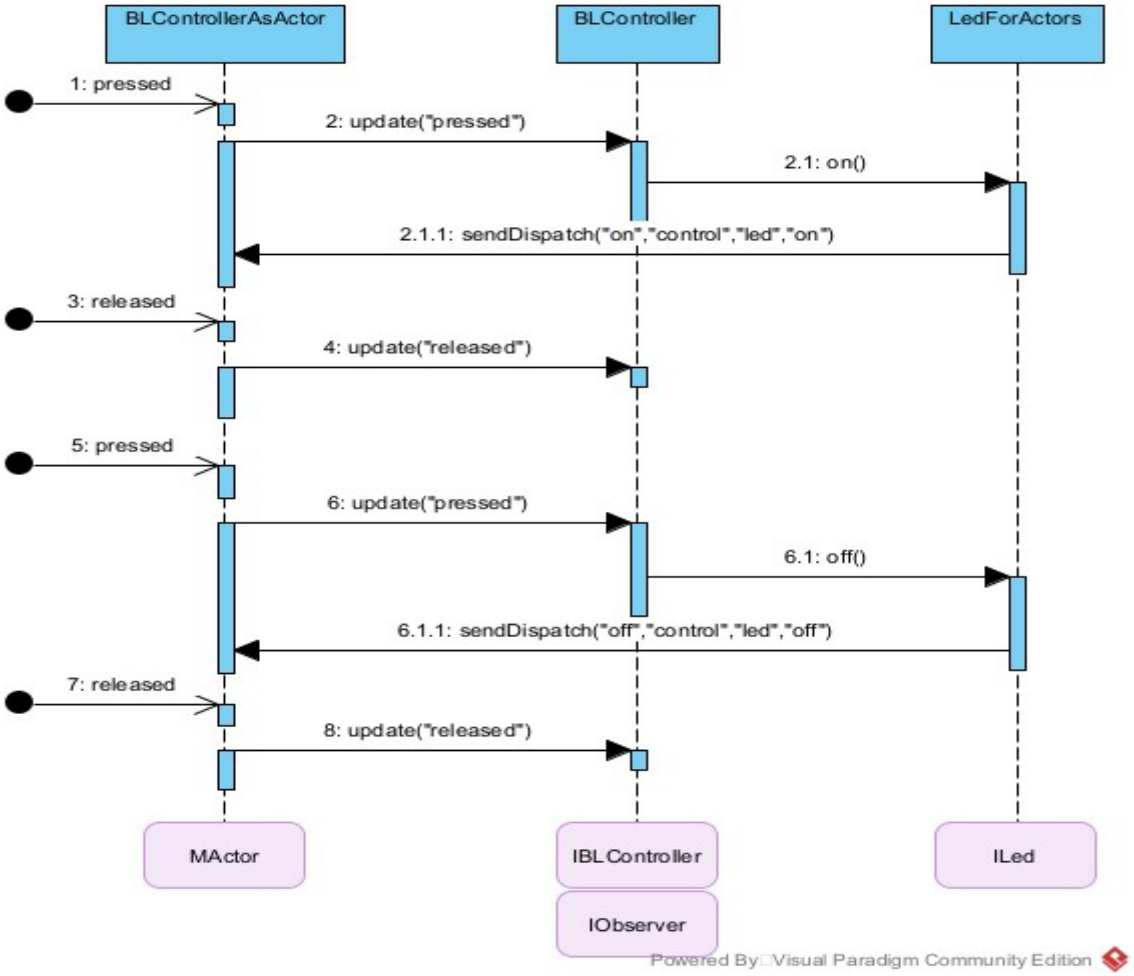
Il LedAsActor incapsula un ILedDevice (dispositivo concreto). Può rispondere ai messaggi “on” , “off” ed “isOn”. Quando riceve il messaggio “on”, cambia il suo stato interno e chiama sull'ILedDevice il metodo on(). Quando riceve il messaggio “off” cambia il suo stato interno e chiama sull'ILedDevice il metodo off(). Quando riceve il messaggio “isOn” controlla il suo stato interno ed invia al mittente la stringa corrispondente “true” oppure “false”. Il nome del mittente viene inserito dal mittente stesso nel messaggio inviato al led (interazione di tipo request-response). L'ILedDevice viene creato dal LedAsActor usando l'apposita fattoria LedFactory.

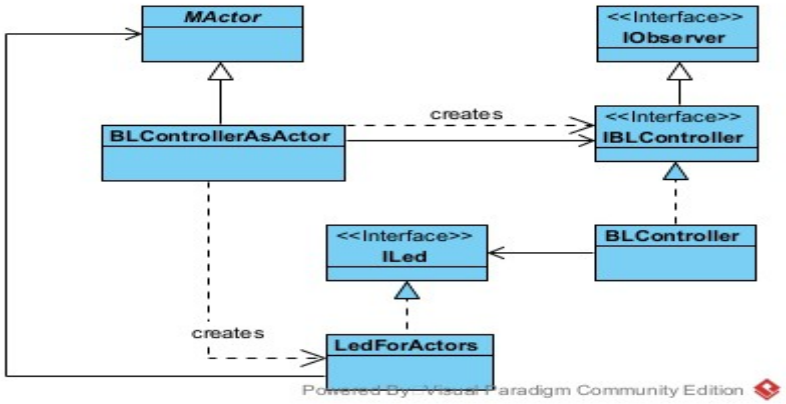




Progettazione Controller

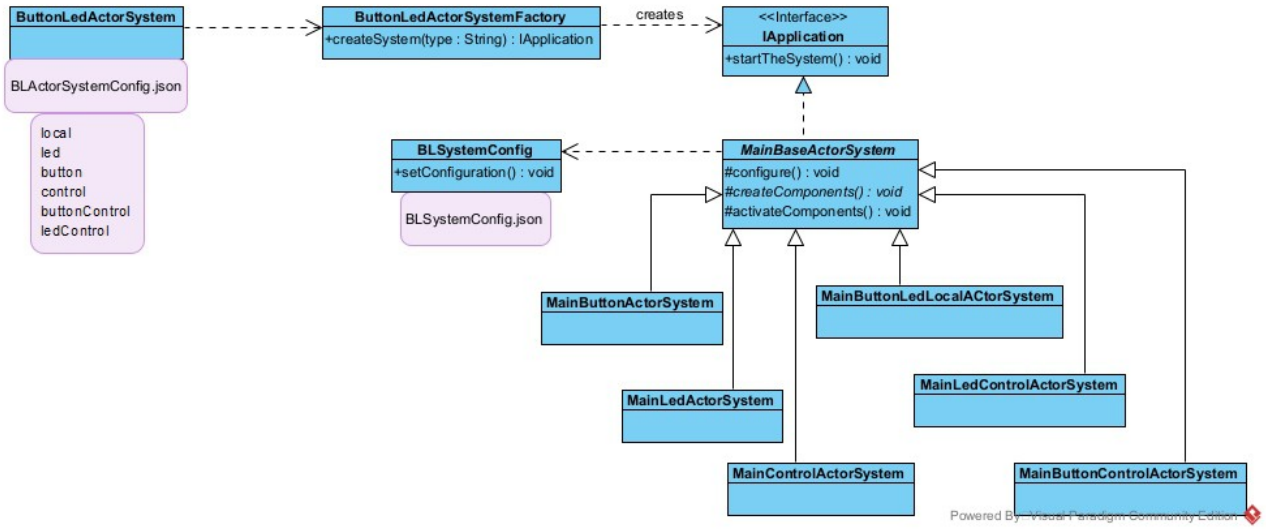
Il BLControllerAsActor incapsula la logica aziendale. Percepisce gli eventi di tipo buttonEv emessi dal ButtonAsActor esprimendo interesse alla ricezione di tali eventi mediante una subscription. Al suo interno è stata riusata la logica sviluppata in precedenza ed incapsulata nella classe BLController facendo uso di una particolare implementazione dell'interfaccia ILed, LedForActors. Quando il BLControllerAsActor percepisce un evento buttonEv, innesca la logica aziendale incapsulata nel BLController. Questi, a sua volta chiamerà sul LedForActors, implementazione di ILed, il metodo on() oppure off(). Il LedForActors (adapter gof) userà il BLControllerAsActor per inviare al LedAsActor il messaggio corrispondente “on” oppure “off”.





Progettazione ButtonLedSystem con attori

Possiamo costruire 6 diversi sistemi. L'uno differisce dall'altro per la distribuzione dei componenti. Abbiamo introdotto il componente ButtonLedActorSystem (dopo refactoring) incaricato di costruire il sistema usando un file di configurazione BLSystemActorConfig.json. Legge dal file il tipo di sistema che dovrà essere costruito e quindi userà una opportuna fattoria per crearlo. Ogni tipo di sistema dovrà implementare l'interfaccia IApplication. Abbiamo 6 tipi possibili di IApplication: locale (bottone, led e controllo nello stesso nodo), solo bottone (controllo e led in un altro nodo), solo led (bottone e controllo in un altro nodo), bottone e controllo, led e controllo, solo controllo.
Di seguito un diagramma che mostra la struttura del sistema:



Analizzando il lavoro fatto ci si può chiedere se sia possibile generalizzare la creazione di un sistema ad attori. La risposta è affermativa. E' stato introdotto il componente MActorSystem, nel progetto mactor, responsabile della creazione di un generico sistema ad attori. Inoltre sono stati aggiunti i componenti del dominio ButtonActor e LedActor. Ogni sistema ad attori può essere creato specificando l'insieme dei contesti (uno dei quali sarà locale e tutti gli altri remoti) e l'insieme degli attori all'interno di un file di configurazione. Ogni contesto sarà descritto da un nome, un indirizzo ed una porta mentre ogni attore sarà descritto da un nome, un contesto di appartenenza ed il nome della classe dell'attore (per poterlo istanziare usando la reflection). Ogni attore specifico applicativo dovrà poi provvedere a creare e recuperare tutto ciò di cui ha bisogno per adempiere alle sue responsabilità. Per i componenti del dominio l'approccio usato è quello di istanziare un supporto che dovrà, con l'aiuto di un file di configurazione, recuperare tutto il necessario per il corretto funzionamento.