Amiko Doctor's App – Documentazione

Azienda: *Amiko*

Autore: *Daniele Prestianni*

Revisore: *Alessandro Fato*

Date: *06 Maggio 2016*

Last update: *17 Maggio 2016*

Versione: *v.1.0*

Table of Contents

Raccolta dei requisiti

Si vuole realizzare un'applicazione standalone (APP), pensata per sistemi desktop, capace di offrire dei servizi di alto livello, costruiti utilizzando l'API esposta dai dispositivi Amiko.

Requisiti funzionali

Di seguito una ista di requisiti funzionali, emersi nel corso dei meetings con il CTO di Amiko:

1. Ricerca dei dispositivi BLE 4.0 prossimi al DONGLE BLED112
2. Visualizzazione aggiornata della lista dispositivi BLE 4.0 disponibili; il massimo intervallo ammesso tra due aggiornamenti successivi è di 10 secondi
3. Connessione al dispositivo Amiko individuato
4. Disconnessione al dispositivo Amiko correntemente in uso
5. Capacità di ristabilire, automaticamente, una connessione all'ultimo dispositivo Amiko, solo in caso di disconnessioni inattese e non volute dall'utente
6. Visualizzazione del livello di carica della batteria, con aggiornamenti ogni minuto
7. Visualizzazione della data corrente, indicata dal dispositivo Amiko, con aggiornamenti ogni minuto
8. Possibilità di impostare la data corrente all'interno del dispositivo Amiko
9. Possibilità di specificare una qualsiasi data valida all'interno del dispositivo Amiko
10. Cancellazione della memoria Flash, interna al dispositivo Amiko
11. Download (recupero) e visualizzazione tabellare di tutti i dati/eventi contenuti all'interno del dispositivo Amiko
12. Download-recovery da realizzarsi in caso di inattese disconnessioni dell'APP, dal dispositivo Amiko
13. Esportazione dei dati/eventi acquisiti dal dispositivo Amiko su file .CSV
14. Possibilità di definire il codice seriale associato al dispositivo Amiko
15. Generazione di logs su files

Requisiti non funzionali

Di seguito una ista di requisiti non funzionali, emersi nel corso dei meetings con il CTO di Amiko e a seguito di un'attività di scouting tecnologico:

1. L'APP dovrà essere sviluppata per sistemi desktop (Windows e Linux), dovrà essere standalone e dovrà presentare un'interfaccia utente user-friendly
2. L'APP dovrà supportare piattaforme a 32-bit e a 64-bit
3. L'APP dovrà supportare sistemi Windows e Linux
    a. **Windows**: rappresenta la piattaforma principale perché utilizzata dagli utenti (medici) coinvolti nelle fasi di test del prodotto Amiko.
    b. **Linux**: rappresenta la piattaforma in uso dal team di Amiko, pertanto se ne richiede il supporto
4. L'APP dovrà interagire con i dispositivi Amiko tramite il DONGLE BLED112 sviluppato da Bluegiga[1]
5. L'APP dovrà implementare il protocollo di comunicazione utilizzato dai dispositivi Amiko; per maggiori dettagli si rimanda al paragrafo associato. Per convenzione, la codifica dei dati utilizzata è di tipo little-endian.

[1] A seguito di scouting tecnologico, il dispositivo BLED112 della Bluegiga permette di accedere allo stack BLE 4.0 tramite qualsiasi piattaforma (Windows, Mac e Linux). Avendo selezionato Java come tecnologia per l'implementazione della APP ed avendo come obiettivi lo sviluppo di un'applicazione cross-platform, la soluzione del DONGLE USB si è rivelata come una valida soluzione. Il vincolo più stringente ci è stato imposto dall'assenza di un'unica libreria BLE 4.x compatibile con tutte le piattaforme target.

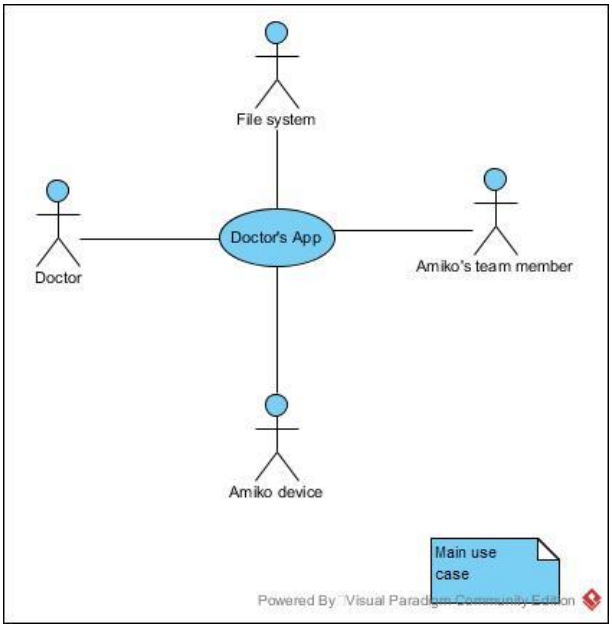
6. L'interazione tra l'APP ed il device Amiko segue il paradigma client/server, dove l'APP riveste il ruolo di client; in questo modello, l'APP potrà inviare le proprie richieste in modo sequenziale: non è ammesso l'invio di richieste contemporanee. Una gestione mediante code permetterà di soddisfare il requisito. Prevedere un meccanismo di retry qualora una richiesta non dovesse ricevere relativa risposta, all'interno di un intervallo di tempo predefinito (timeout delle richieste)
7. In caso di estrazione del DONGLE durante la normale esecuzione dell'applicazione, questa dovrà chiudersi automaticamente.

Analisi dei requisiti

Attori

Da una analisi dei requisiti, possiamo identificare i seguenti attori:

- **User**: utente utilizzatore dell'APP
  - **Doctor**: rappresenta l'utente medico che utilizzerà l'applicazione per acquisire i dati medicali dei propri pazienti
  - **Amiko's team member**: rappresenta il ricercatore/collaboratore/dipendente di Amiko, coinvolto nell'implementazione della soluzione hardware/software
- **Amiko device**: dispositivo medicale con il quale dover interagire e dal quale acquisire i dati/eventi associati ai pazienti
- **File-system**: lo inseriamo in quanto rappresenta un sistema esterno con il quale interagire



Scenari

Casi d'uso

Protocollo di comunicazione

L'APP dovrà essere compatibile con il protocollo di comunicazione ad oggi implementato nei dispositivi Amiko. In assenza di un versionamento, andremo a riportare le specifiche di tale protocollo all'interno dell'Appendice A.

L'analisi dello stesso è fondamentale per arrivare ad un design dell'intero sistema (System design) che sia estendibile.

Il requisito di estendibilità potrebbe tradursi nella possibilità di future implementazioni dell'APP, capaci di interagire con i dispositivi Amiko attraverso link layers differenti a quelli correntemente in uso (BLE 4.0 tramite stack BGAPI della Bluegiga).

Modello del sistema

La definizione del modello (entità e relazioni) parte dall'analisi del protocollo Amiko. Le entità individuate sono le seguenti:

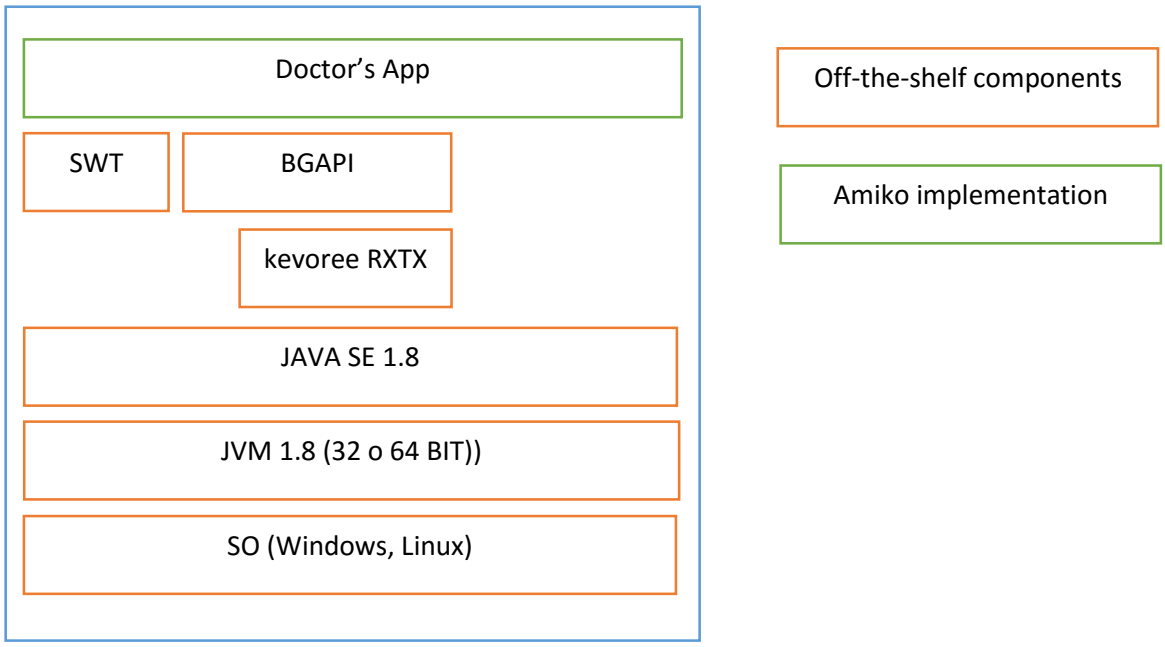
- **Amiko Command**: rappresenta il comando che è possibile inviare ad un dispositivo Amiko
    - **Device mode**: indica la modalità di interazione da utilizzare
    - **Command type**: specifica il tipo di comando tra quelli previsti dal protocollo Amiko
    - **Parametri**: lista di parametri, se ammessi, associati al comando
- **Amiko Command Response**: rappresenta la risposta ad un comando, ricevuta dal dispositivo Amiko
    - **Response type**: definisce il tipo di risposta, tra quelle ammesse dal protocollo
    - **Response status**: contiene lo stato della risposta
    - **Result**: contiene il risultato associato alla risposta
- **Record**: rappresenta il contenuto associato ad una risposta contenente uno dei record contenuti all'interno della memoria Flash di un dispositivo Amiko
    - **Index**: identifica il record
    - **Timestamp**
    - **Event ID**: permette di identificare l'evento definito dal record
    - **DataX (X variabile da 0 a 5)**: ciascun evento si caratterizza per un set di informazioni che può variare da 0 a 6 elementi informativi; l'analisi dell'Event-ID permette di determinare cosa recuperare dalle proprietà DataX

Design del sistema

Stack tecnologico

Lo stack tecnologico può cosi essere descritto:

1. **Doctor's App**
2. **SWT** (Standard Widget Toolkit): librerie Java utilizzate nell'implementazione della GUI associata all'APP. SWT fa parte degli Heavy-Weight i quali usano i widgets forniti dal sistema operativo, al fine di garantire le migliori prestazioni a livello di user-experience. SWT Integra i pregi di AWT e Swing, è possibile usare Frame Swing e AWT dentro applicazioni SWT. SWT è un progetto Open Source.
3. **BGAPI v.1.0.3** [2]: libreria per mezzo della quale interagire con i moduli Bluegiga Bluetooth Low Energy quali il BLED112
4. **kevoree RXTX** [3]: per realizzare la comunicazione seriale[4] richiesta nell'interazione con il DONGLE BLED112, BGAPI si avvale di una libreria di terze parti, RXTX; questa libreria è progettata per essere cross-platform e supportare la comunicazione seriale, accedendo ai servizi nativi esposti dal Sistema Operativo in uso.



ATTENZIONE n° 1

La libreria **kevoree RXTX** si avvale di alcune classi identiche (almeno nel naming) a quelle preenti nel seguente repository: https://github.com/rxtx/rxtx. Prestare attenzione che le due implementazioni sono differenti e non sono interscambiabili.

Da un punto di vita dello stack di comunicazione, proponiamo il seguente diagramma esemplificativo:

[2] La libreria ed i sorgenti associati possono essere recuperati su GitHub: https://github.com/SINTEF-9012/bglib
[3] La libreria ed I relative sorgenti possono essere recuperate alla seguente URL: https://github.com/dukeboard/kevoree-extra/tree/master/org.kevoree.extra.osgi.rxtx
[4] Il DONGLE BLED112 permette l'interazione con i dispositivi BLE 4.0 esponendo una porta seriale, **COMx** per i sistemi Windows e **ttyACMx** per i sistemi Linux.

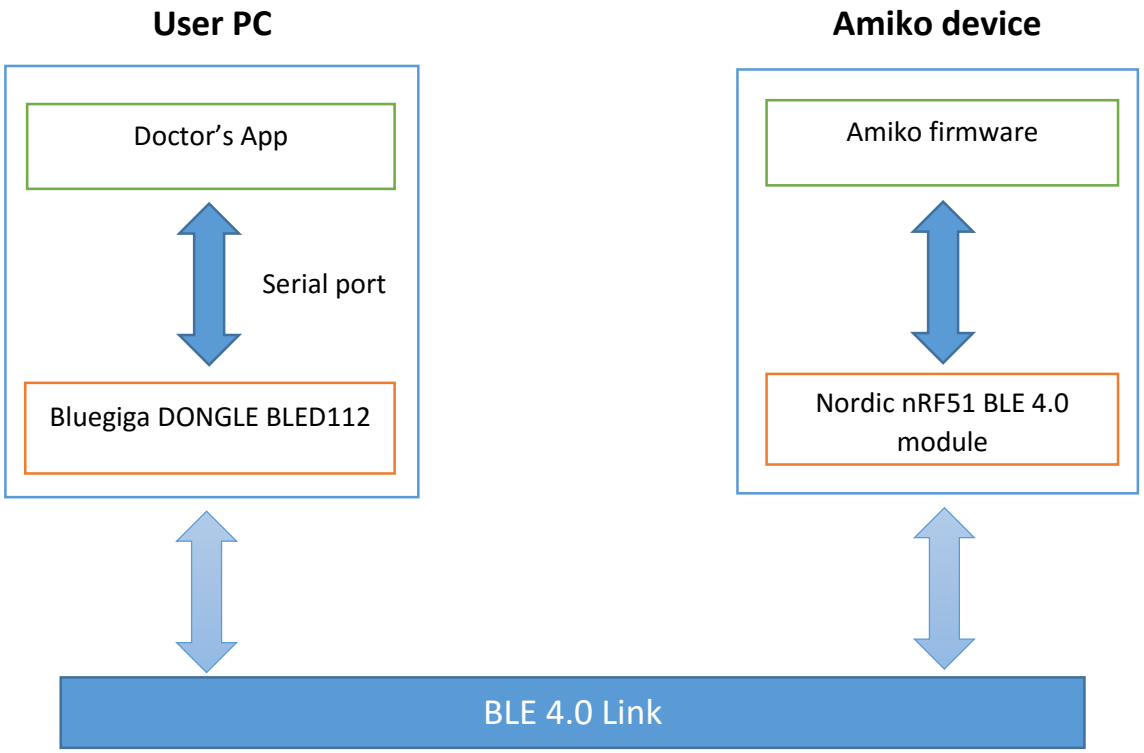


Diagramma delle classi

L'implementazione della APP si caratterizza dei seguenti macro-moduli software:

1. **Demo** (io.amiko.app.doctorsapp.demo): contiene la logica di business e di presentation associata alla APP.
2. **Devices** (io.amiko.app.devices): contiene Java l'implementazione del protocollo Amiko
    a. **BLE Driver** (io.amiko.app.devices.bledriver): contiene le interfacce e le classi madre da utilizzare nell'implementazione dei drivers
        i. **Bluegiga BLE Driver** (io.amiko.app.devices.bledriver.bluegiga): driver Java utilizzato per l'interazione con il dispositivo Amiko mediante modulo DONGLE BLED112 di Bluegiga

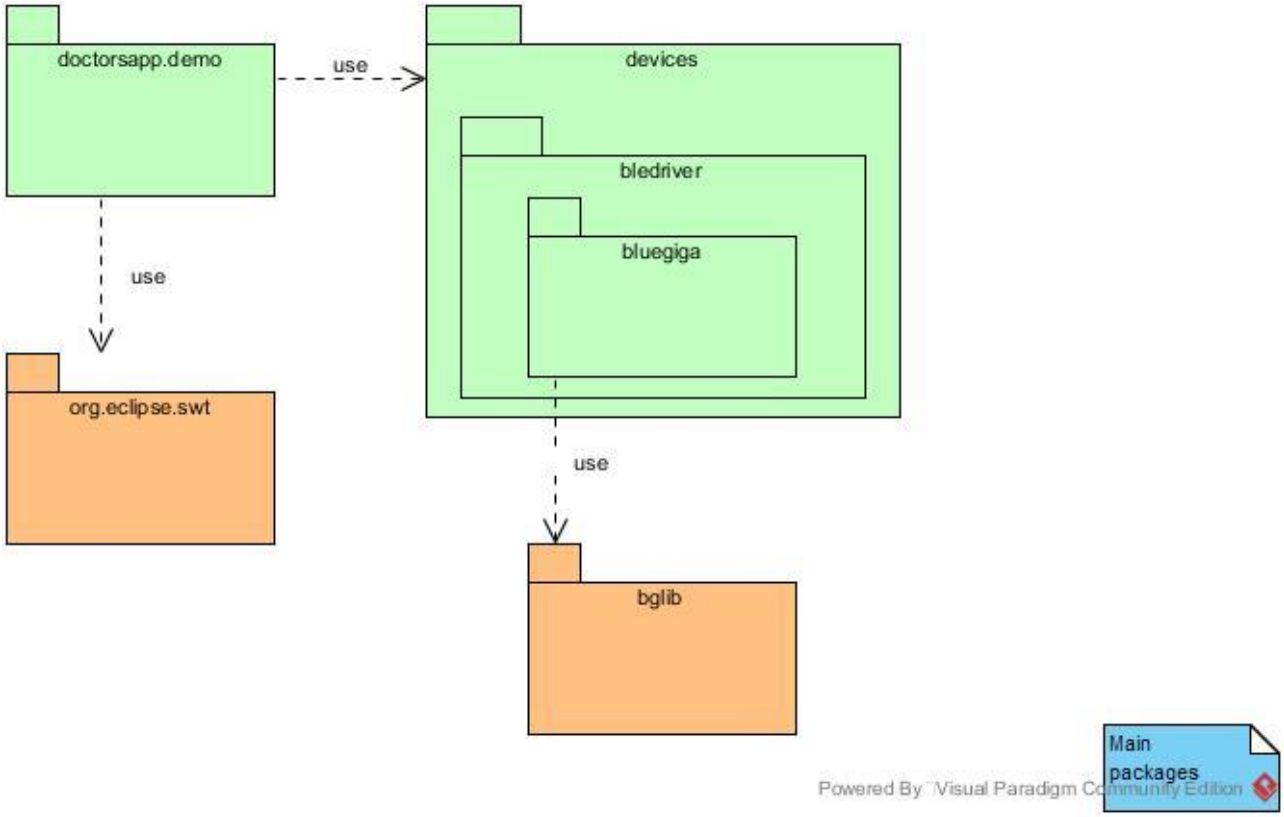


Figura 1Main packages

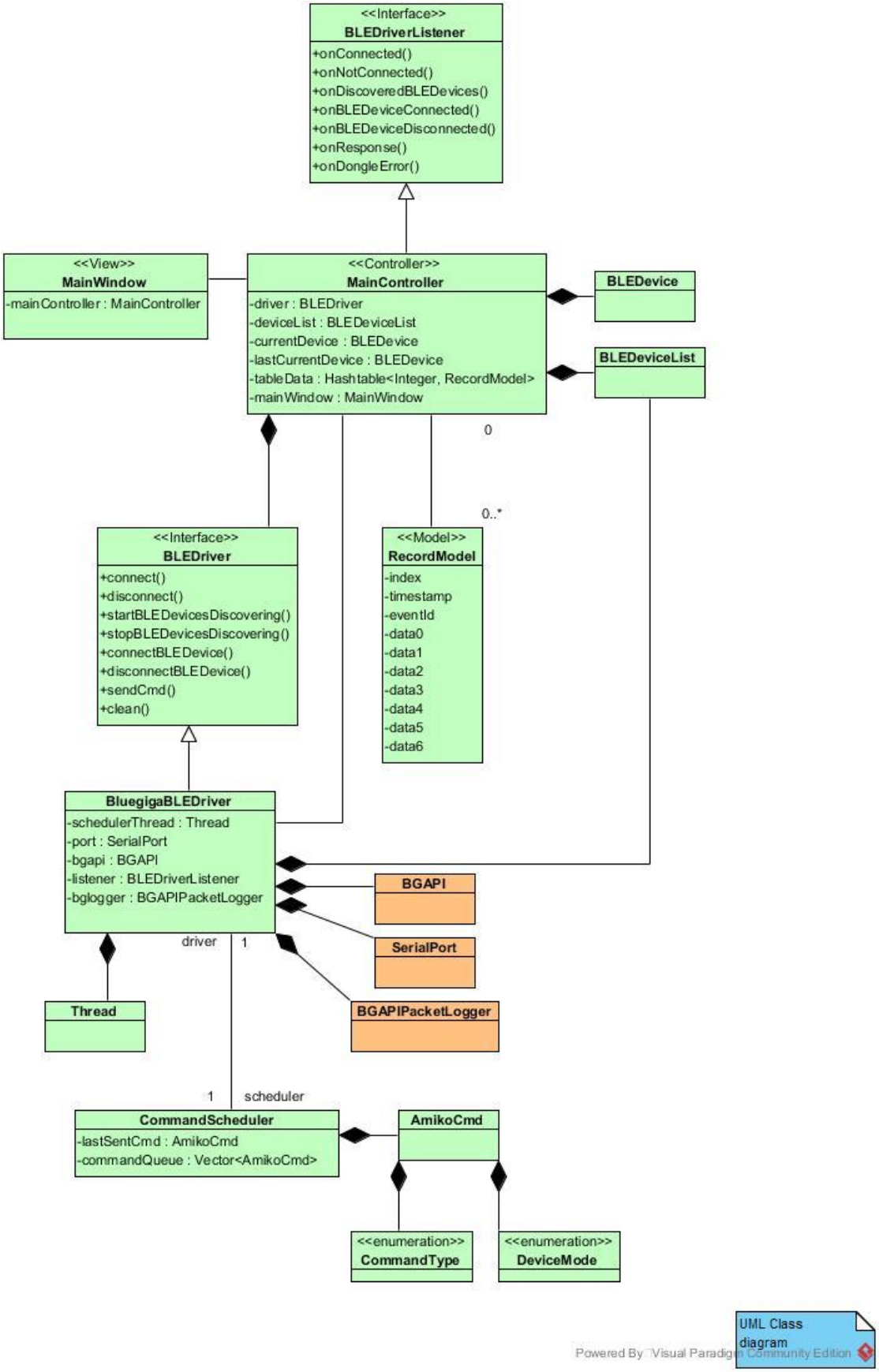


Figura 2UML class diagram

Implementazione

MVC

Il pattern MVC è stato preso a riferimento nell'implementazione della Doctor's App; il ruolo di Controller è implementato dalla classe **MainController** mentre quello di presentation dalla classe **MainWndow**.

In questo pattern, il **MainController** si occupa di:

1. elaborare gli eventi scatenati tramite interfaccia utente
2. realizzare l'elaborazione del data model (records)
3. interagire con il device Amiko [5]
4. interagire con il File-System

La **MainWindow** si occupa di:

1. presentare la User-interface richiesta da specifiche per una interazione con l'utente
2. aggiornare l'interfaccia in relazione a quanto richiesto dal **MainController**

Il model è definito dalle seguenti classi:

1. **RecordModel**
2. **AmikoCmd**
3. **AmikoCmdResp**

Libreria Kevoree RXTX

L'implementazione ha richiesto una modifica della libreria, per poter supportare la piattaforma Linux; la libreria, in fase di ricerca delle porte seriali attive (per intenderci quella interessata dal DONGLE) non era in grado di trovare le porte **ttyACMx**.

Poiché il sistema Linux associa al DONGLE la porta ttyACMx (es. ttyACM0), la Doctor's App per Linux non era in grado di individuare il DONGLE richiesto. Per far fronte a questo problema, si è rea necessaria una modifica alla classe **gnu.io.RXTXCommDriver**:

La riga evidenziata è la modifica richiesta.

```
if(osName.equals("Linux"))
{
        String[] Temp = {
                "ttyS", // linux Serial Ports
                "ttySA", // for the IPAQs
                "ttyUSB", // for USB frobs
                "rfcomm",  // bluetooth serial device
                "ttyircomm", // linux IrCommdevices (IrDA serial emu)
                "ttyACM",
        };
        CandidatePortPrefixes=Temp;
}
```

[5] In futuro si potrà implementare un controller ad-hoc per gestire la comunicazione con il dispositivo Amiko

ATTENZIONE n°2

La modifica richiesta alla libreria **kevoree RXTX** è valida per tutte le piattaforme Linux che si auto-identificano come "Linux"; se andiamo a guardare l'implementazione della classe **RXTXCommDriver** questa analizza il risultato ottenuto dal comando:

osName=System.getProperty("os.name");

Laddove la APP dovesse riscontrare un problema nello stabilire una connessione con il DONGLE le verifiche da effettuare sono le seguenti:

1. Identificare l'etichetta ETCH associata alla porta seriale in uso dal DONGLE, dalla distribuzione Linux in uso
2. Identificare il valore della property di sistema os.name, riportata all'interno dei logs (vedi alla fine di questa sezione)
3. Assicurarsi che la coppia di valori ETCH + os.name sia effettivamente gestita dalla classe **RXTXCommDriver** ; in caso contrario si rende necessaria una modifica.

Per assicurarsi che la modifica abbia successo, occorre fare endorsing della stessa: in fase di class-loading, occorre assicurarsi che la versione aggiornata della **RXTXCommDriver** venga caricata prima di eventuali differenti implementazioni (es. quelle contenute all'interno di JARs).

La proprietà os.name viene riportata all'interno dei logs, un esempio:

[DEBUG]-[main]-[05/05/2016 14:09:42,239]-[MainWindow]-OS=Windows 10

Libreria SWT

Windows 32-bit

Per poter esportare la Doctor's App su sistemi Windows a 32-bit, il progetto dovrà includere la seguente libreria: org.eclipse.swt.win32.win32.x86_3.104.2.v20160212-1350.jar[6]

Windows 64-bit

Per poter esportare la Doctor's App su sistemi Windows a 64-bit, il progetto dovrà includere la seguente libreria: org.eclipse.swt.win32.win32.x86_64_3.104.2.v20160212-1350.jar[7]

Linux 32-bit

Per poter esportare la Doctor's App su sistemi Linux a 32-bit, il progetto dovrà includere la seguente libreria: org.eclipse.swt.gtk.linux.x86_3.104.2.v20160212-1350.jar [8]

Linux 64-bit

Per poter esportare la Doctor's App su sistemi Linux a 64-bit, il progetto dovrà includere la seguente libreria: org.eclipse.swt.gtk.linux.x86_64_3.104.2.v20160212-1350.jar [9]

[6] La libreria appartiene al progetto Eclipse e può essere recuperata da una installazione a 32-bit dell'IDE Eclipse per Windows (cartella /plugins)
[7] La libreria appartiene al progetto Eclipse e può essere recuperata da una installazione a 64-bit dell'IDE Eclipse per Windows (cartella /plugins)
[8] La libreria appartiene al progetto Eclipse e può essere recuperata da una installazione a 32-bit dell'IDE Eclipse per Linux (cartella /plugins)
[9] La libreria appartiene al progetto Eclipse e può essere recuperata da una installazione a 64-bit dell'IDE Eclipse per Linux (cartella /plugins)

Logging

La generazione dei logs è stata pensata per supportare le prime fasi di test dell'APP e per supportare eventuali attività di troubleshooting.

Sono stati definiti dei contesti funzionali, all'interno dei quali le classi andranno a generare dei logs i relazione al contesto di appartenenza. In questo modo, indipendentemente dalla classe Java che lo ha generato, una riga di log apparterrà ad un contesto specifico, permettendo al personale tecnico di individuare il contesto che ha generato l'errore.

Ad oggi, sono previsti i seguenti contesti:

- [CONNECTION] – si attiva in fase di connessione con lo stack BGAPI
- [BLEDEVICE-DISCONNECTED] - lo stack BLE ha rilevato la condizione di BE device disconnected
- [BLEDEVICE-CONNECTION] – si attiva al click dell'utente sul pulsante di Connect
- [BLEDEVICE-DISCONNECTION] – si attiva al click dell'utente sul pulsante di disconnect
- [RESPONSE] – si attiva in corrispondenza di una nuova risposta da parte del dispositivo Amiko
- [SENT-REQUEST] – si attiva in corrispondenza dell'invio di un nuovo comando al dispositivo Amiko
- [RECOVERY-RECS-DOWNLOAD] - si attiva in caso di recovery-download

Developing

Per lo sviluppo della APP è stato utilizzato l'IDE **Eclipse Mars** (Version: Mars.2 Release (4.5.2)) e tre differenti progetti associati:

- **Amiko_DoctorsApp_BLE4.0_Bluegiga**: oltre a contenere l'implementazione del protocollo BGAPI, contiene un esempio di applicazione standalone preposta al discovering di BLE devices. Contiene altresì il package gnu.io.* che si è reso necessario per l'aggiornamento della libreria **kevoree RXTX**.
- **Utils**: contiene delle classi di utilità.
- **Amiko_DoctorsApp_Demo_Win**: progetto principale e dipendente dai due precedenti. Contiene i packages proprietari di Amiko

In fase di definizione del percorso di build associato al progetto principale, **Amiko_DoctorsApp_Demo_Win**, occorrerà andare ad esplicitare la dipendenza funzionale rispetto ai progetti **Amiko_DoctorsApp_BLE4.0_Bluegiga** e **Utils**.

Manuale utente

Pre-requisiti

1. JVM 1.8 o superiore (usare il comando **java -version** per ottenere la versione correntemente in uso)
2. SO: Windows 8, 8.1, 10, Linux (32-bit e 64-bit)
3. Bluegiga DONGLE BLED112
4. Almeno 10 Mb di memoria RAM disponibile
5. RAM: [50..110] MB

Avvio dell'applicazione su Windows

- Doppio click sul file .JAR
- java -jar AmikoDoctorsApp_Linux_32bit.jar

Avvio dell'applicazione su Linux

- java -jar AmikoDoctorsApp_Linux_32bit.jar

Stato di connessione/disconnessione

La GUI risulterà parzialmente disabilitata fin tanto che non sarà disponibile una connessione con il dispositivo BLE 4.0. Il pulsante di Connect si andrà ad attivare solo e soltanto dopo aver selezionato un dispositivo BLE tra quelli in elenco.

Auto-reconnect

Alla prima connessione valida con un dispositivo Amiko, l'APP ne andrà a memorizzare il nome; in caso di disconnessioni del dispositivo, NON ESCPLICITAMENTE VOLUTE DALL'UTENTE, l'APP provvederà ad ristabilire la connessione quando questo ritornerà ad essere disponibile (in elenco nella lista dei dispositivi BLE).

Download-recovery

Questa funzionalità prevede che il download dei records possa continuare ed arrivare a conclusione tra sessioni BLE differenti e successive.

Nel corso di un download relativamente lungo, può accadere che la connessione al dispositivo Amiko venga interrotta per numerosi motivi: apertura del tappo, distanza eccessiva dal DONGLE, etc. Al fine di evitare di dover ripetere l'operazione, l'APP è stata implementata per ristabilire la connessione con il medesimo dispositivo e riprendere il download a partire dall'ultimo record correttamente acquisito.

Se la disconnessione che ha interrotto il download dei records è stata richiesta dall'utente, la funzionalità di download-recovery non si verrà ad attivare (vedi auto-reconnect).

Se il download-recovery è attivo, l'utente non potrà fare click sui pulsanti di Refresh e di Clean.

Esportazione dati su CSV

L'esportazione dei dati su file CSV andrà a generare un file con le seguenti caratteristiche:

- Il file-name segue questo formato: [DEVICE-NAME]_data_[DATE yyyy-MM-dd-HH-mm-ss].csv (es. Amiko_Tracker_Ellipta_0_data_2016-05-04-10-51-47.csv)

- Il file CSV avrà la seguente intestazione: Index, Timestamp, EventId, Data0, Data1, Data2, Data3, Data4, Data5, Data6
- Il contenuto del file è rappresentato da valori interi (nessuna rappresentazione in esadecimale è presente)

Parametri di configurazione

L'applicazione utilizza alcuni parametri da poter configurare tramite file di proprietà: **config.properties**.

Per accedere a tale fine, è sufficiente aprire il file .JAR utilizzando WinZip, 7zip o software analoghi (un file JAR - Java Archive - altri non è che un archivio compresso); recuperato il file di configurazione, potrete modificarlo in base alla proprie preferenze, per poi re-inserire (aggiornato) all'interno del JAR.

Tutti i parametri sono stati documentati, quindi non riporteremo ulteriori dettagli in merito.

Troubleshooting

Elenchiamo gli errori ad oggi riscontrati e le relative soluzioni:

java.lang.OutOfMemoryError - sebbene non si sia mai presentato questo tipo di errore, riteniamo che possa verificarsi sotto certe condizioni. In fase di acquisizione dei records memorizzati dal dispositivo Amiko, l'APP provvede a memorizzare tutti i records, all'interno di una Hashtable. Eventuali estensioni della memoria Flash potrebbero portare a saturare la memoria. La memoria Heap uata dalla JVM può variare da 5Mb a 64Mb, per poterla ampliare, occorre aggiungere i seguenti parametri di avvio della JVM

- **-Xmx**<size>: per impostare la dimensione massima che può avere l'Heap (64 MegaByte di default)
- **-Xms**<size>: per impostare la dimensione iniziale che deve avere l'Heap (circa 5 MegaByte di default)

Status: … please check if your DONGLE is installed – Le cause ad oggi rilevate sono le seguenti:

- Il DONGLE non è connesso fisicamente ad alcuna porta USB: intervenire inserendo il DONGLE in una porta USB
- Il DONGLE è connesso ma utilizzato da altra applicazione: chiudere l'eventuale applicazione cosi da liberare il device
- Il DONGLE è connesso, riconosciuto dal sistema e disponibile: verificare di possedere i permessi di accesso alla periferica. La prima operazione da fare è quella di provare ad eseguire l'applicazione come superuser; in caso di successo, provare ad acquisire i permessi utilizzando questo comando Linux: ***usermod -a -G dialout [YOUR_USER_NAME]***
- Il DONGLE è connesso ma non utilizzato da alcuna applicazione: verificare se il SO riconosce il dispositivo oppure prendere visione del box di attenzione n° 2

Unsupported major.minor version 52.0 Error: questo tipo di errore viene generato quando si prova ad eseguire un'applicazione Java compilata per la versione 1.8 utilizzando una JVM (JRE) di una precedente versione. La soluzione richiede l'aggiornamento della JVM a versione 1.8 o superiore.

FAQ

D: **stabilisco una connessione con il device BLE ma l'interfaccia non permette alcuna interazione, fatta eccezione per il pulsante Disconnect. Cosa devo fare**?

R: L'operazione di disconnessione re-inizializza lo stato della App che, evidentemente, si trova in una condizione d'errore. Facendo click sul pulsante di Disconnect, sarete in grado di uscire dalla condizione di erorre e riprendere l'interazione con il dispositivo BLE.

Appendice A

Protocollo Amiko

Appendice B – Future evoluzioni

- Utilizzo del comando di **PING**, ad oggi previsto dal protocollo, per gestire eventuali stati di inattività del dispositivo Amiko
- L'APP non supporta l'internazionalizzazione (i18n)
- Pensando ad un re-factoring della APP, possiamo suggerire le seguenti modifiche:
    - Implementare un controller ad-hoc per gestire la comunicazione con il dispositivo Amiko e quindi l'interazione con il driver Bluegiga