DIGITAL MASS

workshop di cultura digitale *a cura di* ninux.org

Android: da Hello World all'interazione con un robot in sole 4 ore

Antonio Martino e Daniela Ruggeri

Per l'installazione veloce, visitate il seguente sito:

- http://developer.android.com/sdk/index.html

da cui potrete scaricare il bundle (eclipse + SDK + ADT) specifico per il vostro sistema operativo.

Occhio all'architettura!!!!

Scelto il compresso e scaricato, scompattate il tutto in una qualsiasi posizione nel vostro filesystem. Infine aprite il file eseguibile in /cartella_bundle/eclipse/eclipse per avviare l'ambiente di sviluppo.

Se invece avete già eclipse:

potete scaricare l'SDK di android, settare i permessi in impostazioni dandogli il PATH del SDK, infine scaricare l'ADT tramite le utility di eclipse.

Classi e metodi

Activity

- Le Activity rappresentano il presentation layer di un’applicazione Android
- L'Activity potrebbe essere associata ad una semplice schermata. In realtà le Activity possono essere visualizzate come finestre di dialogo oppure essere fatte girare in background.
- Un’applicazione Android può avere diverse Activity.

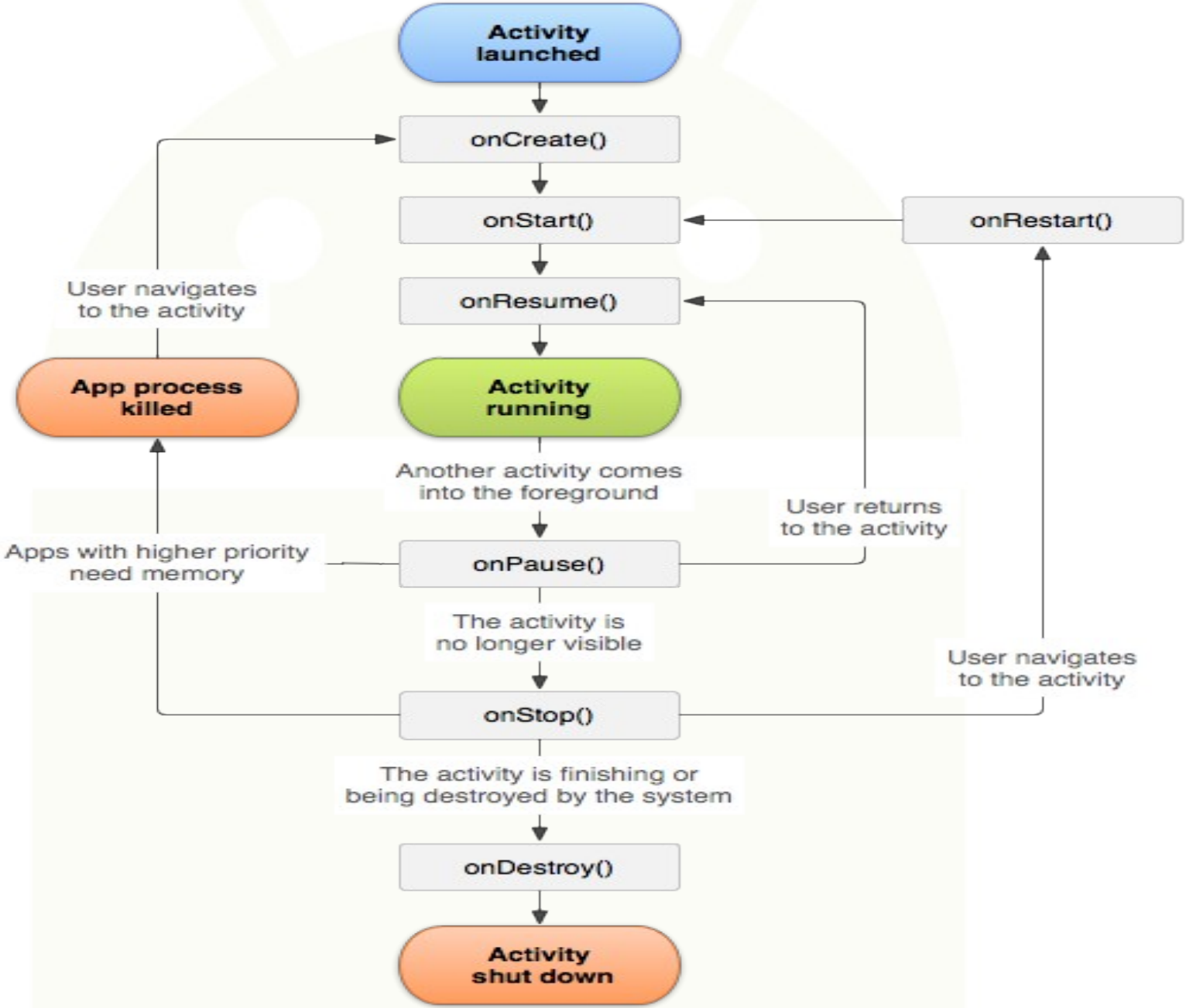
Activity

- Le Activity del sistema sono gestite nello stack.
- Quando una nuova Activity viene avviata, essa viene posta sulla sommità della pila e va in esecuzione
- L'Activity precedente rimane sempre sotto nella pila, e non va in primo piano nuovamente, fino alla chiusura della ultima Activity.

Activity

- Un’Activity ha essenzialmente quattro fasi:
- Se un’Activity in primo piano dello schermo (in cima alla pila), è attivo o in esecuzione.
- Se un’Activity ha perso fuoco ma è ancora visibile (cioè, una nuova Activity non-full-size o trasparente ha ottenuto il focus, è in pausa
- Se un’Activity è completamente oscurata da un’altra Activity, allora è in stato di arresto.
- Se un’Activity è in pausa o arrestata, il sistema può rimuovere l’Activity dalla memoria o chiederle di finire, o semplicemente uccidere il suo processo.

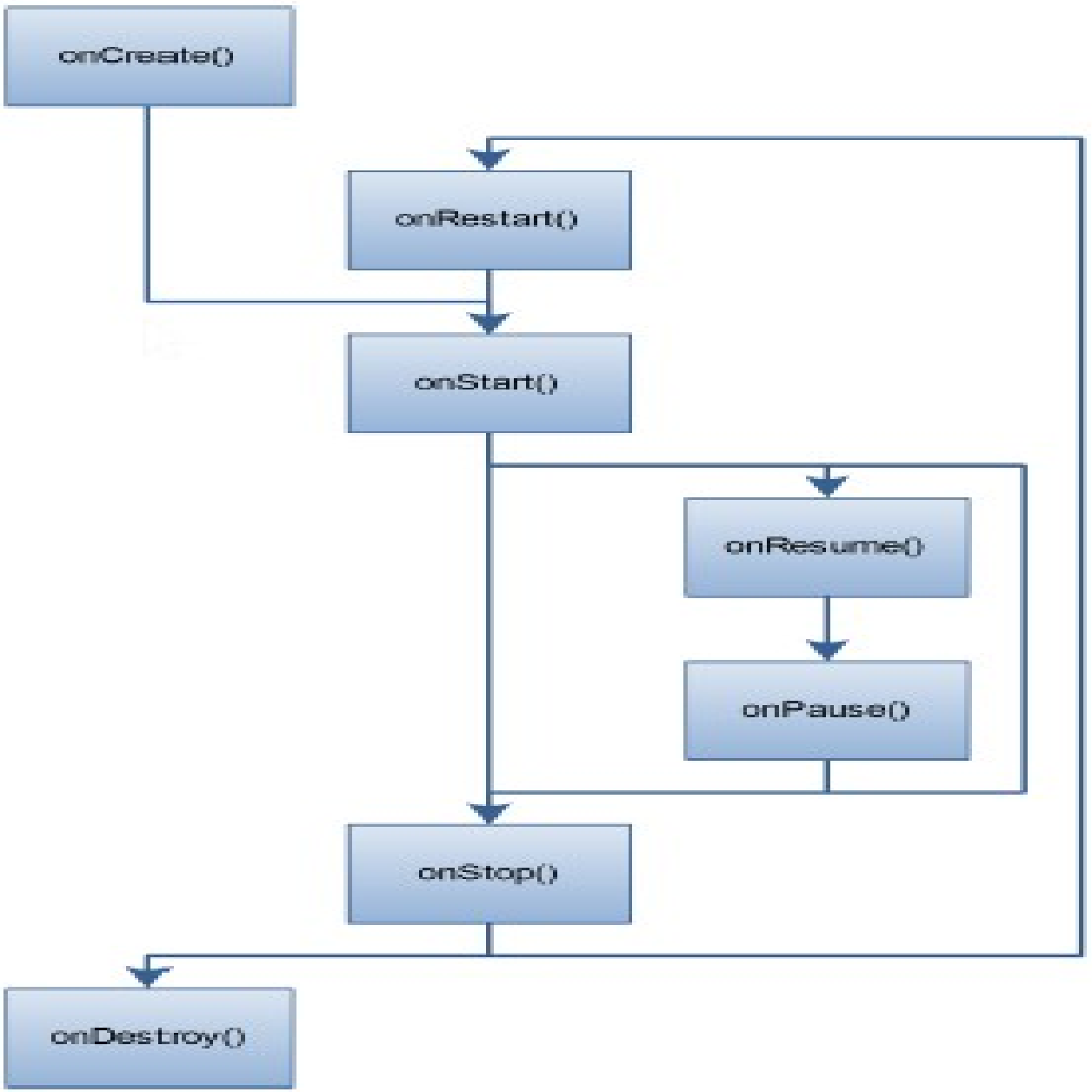
Activity


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity

- public class Activity extends ApplicationContext {
  - protected void onCreate(Bundle savedInstanceState);
  - protected void onStart();
  - protected void onRestart();
  - protected void onResume();
  - protected void onPause();
  - protected void onStop();
  - protected void onDestroy();
- }

Activity


onCreate()
onRestart()
onStart()
onResume()
onPause()
onStop()
onDestroy()

Activity

OnCreate()

- Chiamata alla creazione della Activity. Sempre seguita da onStart().

OnRestart()

- Chiamata dopo l'arresto della Activity e prima che venga riavviata. Sempre seguita da onStart()

OnStart()

- Chiamata quando l'Activity diventa visibile all'utente. Seguito da onResume() se l'attività viene in primo piano, o OnStop() se diventa nascosta.

OnResume()

- Chiamato quando l'Activity inizia l'interazione con l'utente. A questo punto l'attività è in cima allo stack, Sempre seguita da onPause().

Activity

- OnPause()
  - Chiamata quando il sistema inizia a riprendere un'Activity precedente. Viene in genere utilizzata per l'invio delle modifiche non salvate, per interrompere animazioni e altre cose che possono essere CPU burst.
  - Le implementazioni di questo metodo devono essere molto veloci perché l'Activity successiva non verrà ripresa fino alla conclusione di questo metodo. Seguito da onResume() se l'attività ritorna in primo piano, o da OnStop() se diventa invisibile per l'utente.

Activity

- OnStop()
  - Chiamato quando l'Activity non è più visibile all'utente, perché un'altra Activity si è ripresa ed è in primo piano. Ciò può avvenire sia perché una nuova Activity è stata avviata, o una esistente viene portata in primo piano, o è la Activity corrente viene distrutta. Seguita da onRestart() se l'Activity torna ad interagire con l'utente, o OnDestroy() se sta uscendo.

Activity

- OnDestroy()
  - La chiamata finale si riceve prima che l'Activity venga distrutta. Ciò può avvenire o perché l'Activity sta uscendo (qualcuno ha chiamato finish() su di essa, o perché il sistema sta temporaneamente distruggendo questa istanza per salvare spazio. È possibile distinguere tra questi due scenari usando il metodo isFinishing().

Services

- I Services eseguono una attività in background senza fornire un'interfaccia utente.
- Possono informare l'utente tramite (ad esempio) il quadro di notifica in Android.

Views

- Le Views sono widget nell'interfaccia utente (ad e.s. pulsanti o campi di testo).
- La classe base per tutti le Views è android.view.View.
- Le Views spesso hanno attributi che possono essere utilizzati per cambiare il loro aspetto e il loro comportamento.

Views

- È possibile aggiungere Views sia attraverso il codice sia specificandone la struttura in uno o più file di layout XML.
- Ci sono molte sottoclassi specializzate di View che agiscono come controlli o sono in grado di visualizzare testi, immagini o altri contenuti.

Views

- Dopo aver creato delle View è possibile...
- ...impostare le proprietà: ad esempio l'impostazione del testo di una TextView. Le proprietà e i metodi disponibili variano tra le diverse sottoclassi di View. Si noti che le proprietà che sono fissate a priori possono essere impostati nei file XML di layout.
- ...settare il focus: il framework sposterà il fuoco in risposta all'input dell'utente. Per imporre lo stato attivo per una View specifica, chiamare requestFocus().

Views

- ...impostare i Listeners: le Views consentono di impostare Listener che verranno notificati quando succede qualcosa di interessante alla View. Per esempio, tutte le View consentono di impostare un Listener per essere informato quando la View guadagna o perde il focus.
- È possibile registrare un Listener utilizzando setOnFocusChangeListener(android.view.View.OnFocusChangeListener).

Views

- Altre tipologie di View offrono Listener più specializzati. Ad esempio, un Button espone un Listener per notificare i client quando il pulsante viene premuto. Inoltre è possibile nascondere o mostrare View usando setVisibility(int).
- Nel file Xml è possibile associare alla view un ID, utile per recuperarla dal codice:
  - Button myButton;
  - myButton = (Button) findViewById(R.id.my_button);

Views da Xml

Un esempio di view

- <Button
  - android:id="@+id/my_button"
  - android:layout_width="wrap_content"
  - android:layout_height="wrap_content"
  - android:text="@string/my_button_text"/>

Views da programma

- View linearLayout =findViewById(R.id.info);
- TextView valueTV = new TextView(this);
- valueTV.setText("prova");
- valueTV.setId(5);
- valueTV.setLayoutParams(new LayoutParams(LayoutParams.FILL_PARENT,LayoutParams.WRAP_CONTENT));
- ((LinearLayout)linearLayout).addView(valueTV);

Ciclo di vita di una View

- Accade un certo evento. Un handler intercetta l'evento e in base alla tipologia di questo fa delle operazioni sulla view.
- Se nel corso della elaborazione dell'evento, i limiti della vista devono essere modificati, la view chiamerà requestLayout(), oppure invalidate() in caso di modifica.
- Se requestLayout() o invalidate() vengono chiamati, il framework si prenderà cura delle misurazioni, del collocamento, e di disegnare la struttura a seconda dei casi.

Listener

- Su Android, c'è più di un modo per intercettare gli eventi di interazione dell'utente con l'applicazione.
- Per quanto riguarda gli eventi all'interno dell'interfaccia utente, l'approccio è quello di catturare gli eventi dell'oggetto View con cui l'utente sta interagendo.
- La classe View ci fornisce gli strumenti necessari.

Listener

- OnClick(),
- OnLongClick()
- OnFocusChange()
- OnKey()
- OnTouch()

Intent

- Gli Intents sono messaggi asincroni che consentono ad una applicazione di richiedere funzionalità da altri componenti del sistema Android, (es. Service o Activity).
- Un'applicazione può chiamare un componente direttamente (explicit Intent) o chiedere al sistema Android di scegliere tra i componenti registrati di una certa Intent (implicit Intent).
- L'applicativo potrebbe implementare la condivisione di dati tramite un Intent e tutti i componenti che permettono la condivisione dei dati sarebbero di conseguenza disponibili.

Intent

- Pertanto gli Intent permettono di combinare i componenti del sistema per eseguire determinati compiti.
- Gli Intents possono essere usati per avvertire il S.O. che un certo evento è accaduto o per registrarsi ad essi per esserne notificati
- Gli Intents sono istanze della classe android.content.Intent

Intent

- Esistono due tipi di intent:
- esplicito
  - Intent i = new Intent(this, MyActivity.class);
  - i.putExtra("Value1", "Uno");
  - i.putExtra("Value2", "Due");
  - startActivityForResult(i, 10);
- implicito
  - Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));
  - startActivity(i);

Context

- La classe android.content.Context fornisce i necessari collegamenti al sistema Android.
- Essa è l'interfaccia per le risorse e le proprietà globali circa l'ambiente dell'applicazione.
- Context fornisce anche l'accesso ai servizi Android, ad esempio il servizio di localizzazione GPS.
- Activity e Service estendono la classe Context e possono quindi essere utilizzati come Context.

Strumenti di Android

ADT

- Google fornisce gli Android Development Tools (ADT) per sviluppare applicazioni Android con Eclipse.
- ADT è un insieme di plug-in che estendono l'IDE Eclipse, per poter permettere lo sviluppo di applicazioni Android.
- L'ADT contiene tutte le funzionalità necessarie per creare, compilare, eseguire il debug e distribuire le applicazioni Android da IDE Eclipse e dalla riga di comando.
- L'ADT offre inoltre un emulatore dei dispositivi Android.

ADT

- Le applicazioni Android sono scritte inizialmente in linguaggio Java.
- I file di origine Java vengono convertiti in file di tipo Java .class dal compilatore Java.
- Android offre uno strumento chiamato dx che converte i file di classe Java in un (Dalvik Executable) file dex.
- Tutti i file di classe di un'applicazione vengono inseriti in un .dex file compresso.
- Durante questo processo di conversione le informazioni ridondanti nei file di classe sono ottimizzati nel file .dex

ADT

- Per esempio, se la stessa stringa si trova in file di classe diverse, il file .dex conterrà solo un riferimento alla stringa.
- Il sistema Android utilizza una speciale macchina virtuale, la Dalvik Virtual Machine per eseguire applicazioni basate su Java.
- Dalvik utilizza un bytecode proprietario che è diverso dal bytecode Java standard.
- Pertanto non è possibile eseguire un bytecode standard di Java su Android.

Android Manifest

- I componenti e le impostazioni di un'applicazione Android sono descritti nel file AndroidManifest.xml.
- Per esempio tutte le Activity e i Service dell'applicazione devono essere dichiarati in questo file.
- Il Manifest deve anche contenere le autorizzazioni necessarie per l'applicazione.
- Per esempio, se l'applicazione richiede l'accesso di rete deve essere specificato qui.

Android Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.pdm.gpp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".GesturePinActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

R.java

- La directory gen in un progetto di Android contiene i valori generati nell'applicazione (risorse).
- R.java è una classe (generata automaticamente) che contiene riferimenti a determinate risorse del progetto.
- Queste risorse devono essere definite nella directory res e possono essere file XML, icone o immagini.
- È possibile ad esempio definire valori, menu, layout o animazioni tramite file XML.

R.java

- Se si crea una nuova risorsa viene creato un riferimento nel file R.java tramite l'ADT Eclipse. Tali riferimenti sono interi statici corrispondenti agli ID delle risorse.
- Android fornisce i metodi per accedere a tali ID. Ad es. per R.string.[your_string] usiamo
  - getString(R.string.[yourString]).
- R.java viene creato automaticamente dall'ambiente di sviluppo Eclipse, modifiche manuali non sono necessarie e verranno comunque annullate dalle attività dell'ADK.

Assets

- Android offre una directory dove è possibile sistemare file che saranno inclusi, poi, nel pacchetto. Questa directory viene chiamata /assets.
- La differenza tra /res e /assets è che Android non genera ID per i contenuti in /assets.
- È quindi necessario specificare il percorso relativo e il nome file all'interno di /assets.

Layouts

- L’interfaccia utente per le Activities viene gestira tramite i Layouts. I Layouts definiscono i Widgets inclusi e le loro proprietà.
- Un layout può essere definito tramite codice Java o via XML. Si utilizza in genere il codice Java per generare il layout, se non si conosce il contenuto fino al runtime, ad esempio se il tuo layout dipende dal contenuto che si legge da Internet.

Layouts

- I Layout basati su XML sono definiti tramite un file di risorse nella cartella /res/layout, il quale definisce ogni loro attributo.
- Se una view ha bisogno di essere accessibile tramite il codice Java bisogna permettere l'accesso tramite un ID univoco(pag. 13)
  - android:id.
- Per assegnare un nuovo ID ad una View si usa
  - @+id/valore

Layouts

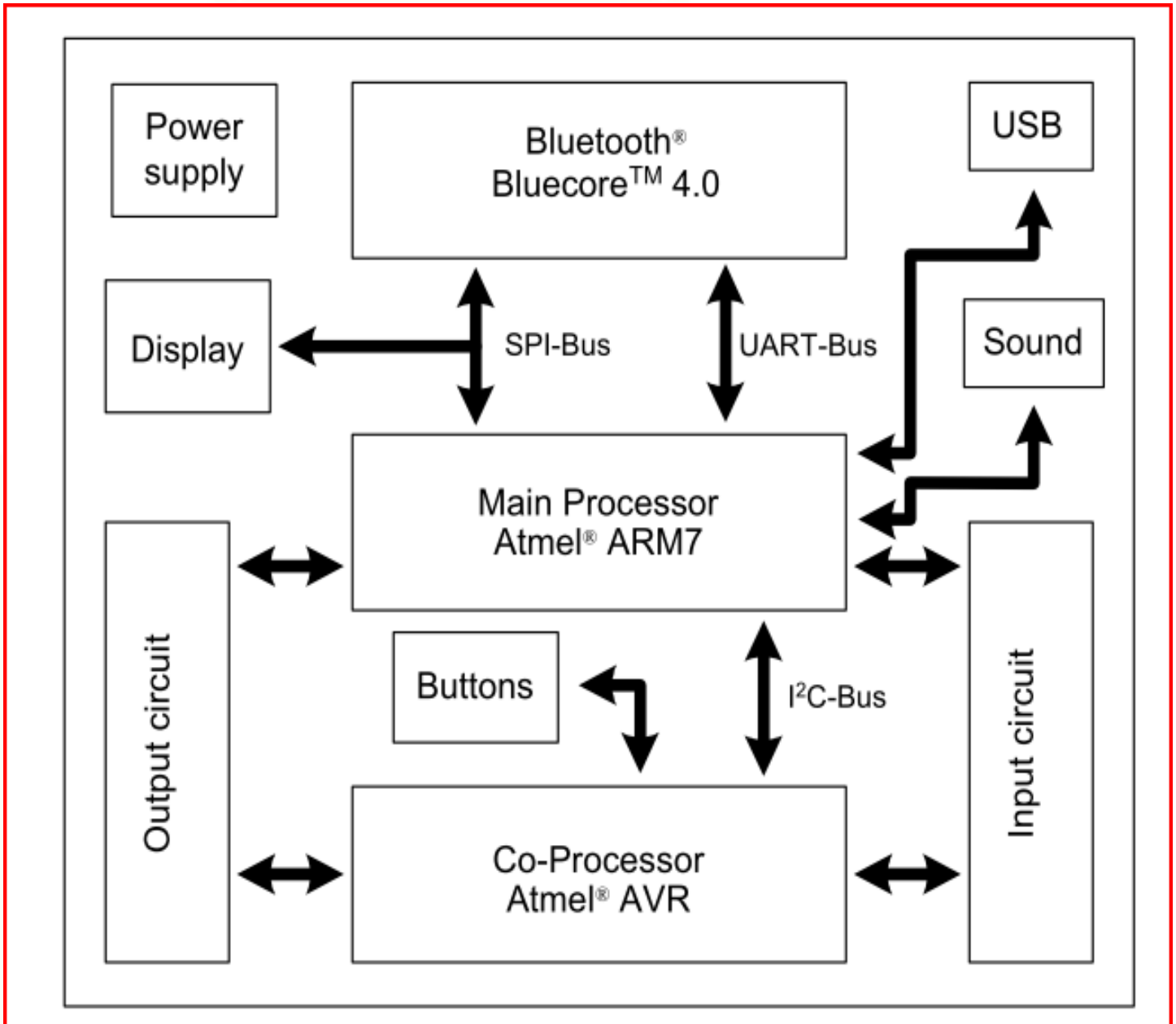
- Nei file XML, come ad esempio i file di Layout, è possibile fare riferimento ad altre risorse attraverso il simbolo @.
- Per esempio, se si vuole fare riferimento ad un colore definito in una risorsa XML, è possibile usando @color/idColore.
- Oppure, se è stata definita una stringa strHello in una risorsa XML, è possibile accedere tramite @string/strHello.

L'ora del caffè e della donazione libera

LEGO Mindstorms NXT

- Caratteristiche hardware:
  - Doppio microcontrollore:
    - ARM7 32bit, 256 Kbyte FLASH, 64 Kbyte RAM
    - AVR 8bit, 4 Kbyte FLASH, 512 Byte RAM
  - 4 ingressi, 3 uscite
  - Interfacce USB e Bluetooth
  - Display grafico 100x64px
  - Audio playback 8KHz
  - Possibilità di realizzare network di quattro NXT (master + 3 slave)

Struttura tecnica NXT



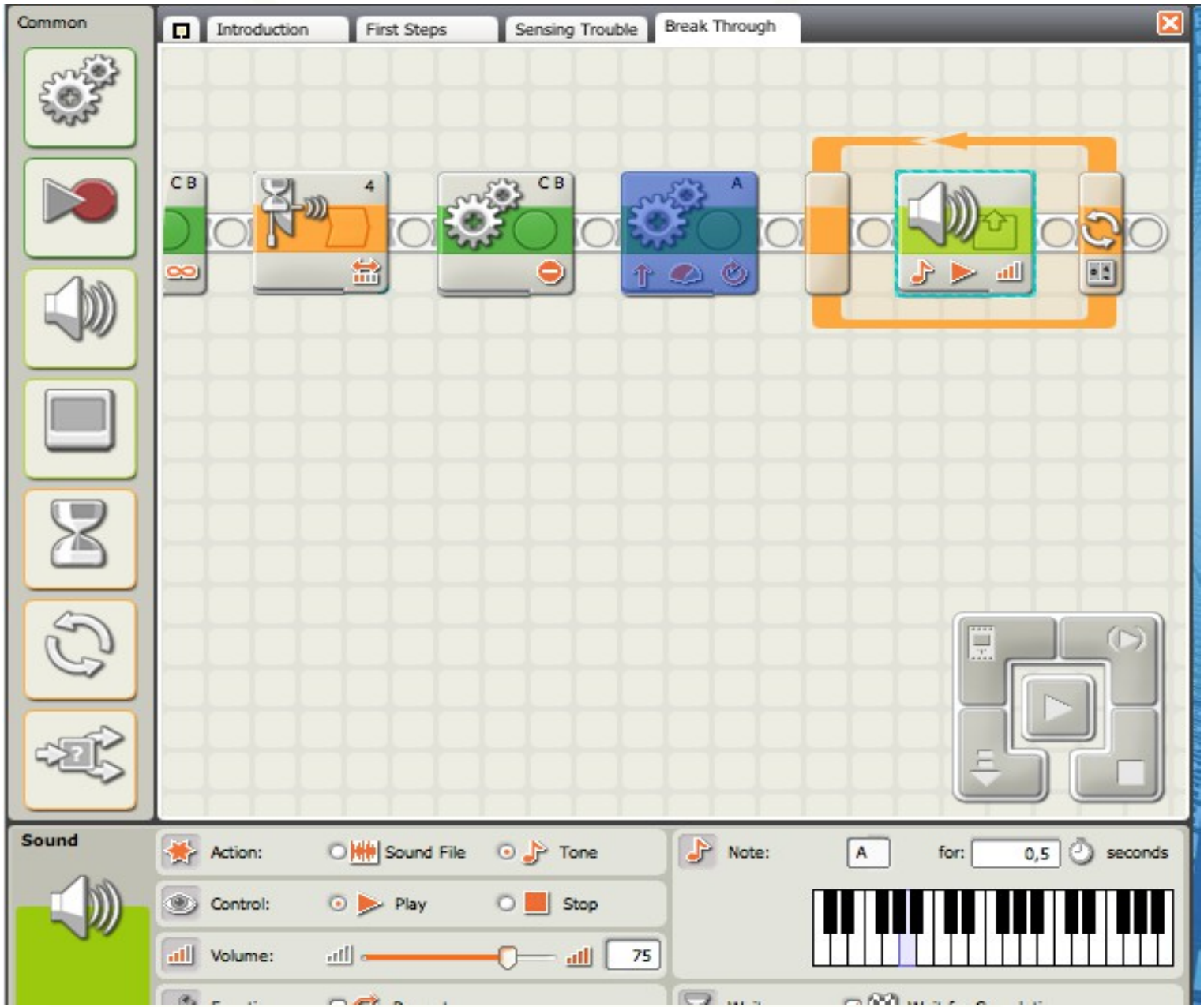
NXT'reme: è tutto open!

- LEGO mette a disposizione della comunità degli sviluppatori risorse per implementare soluzioni originali:
  - Aggiornamenti firmware
  - Documentazione dettagliata dell'interfaccia hardware e delle porte I/O
  - Documentazione dettagliata del protocollo di comunicazione attraverso Bluetooth
  - Strumenti di sviluppo

Sito:

http://mindstorms.lego.com/en-us/whatisnxt/default.aspx

Tool visuale: Labview



LEGO Java Operating System

Lejos è un progetto Open Source che ha realizzato firmware java per gestire i robot RCX e NXT LEGO ®.

Ultima versione NXJ:

leJOS_NXJ_0.9.1beta-3_win32.zip per Windows

leJOS_NXJ_0.9.1beta-3.tar.gz per **Linux\Mac**

scaricabili dal sito http://lejos.sourceforge.net/

Il pacchetto contiene l'installazione Lejos

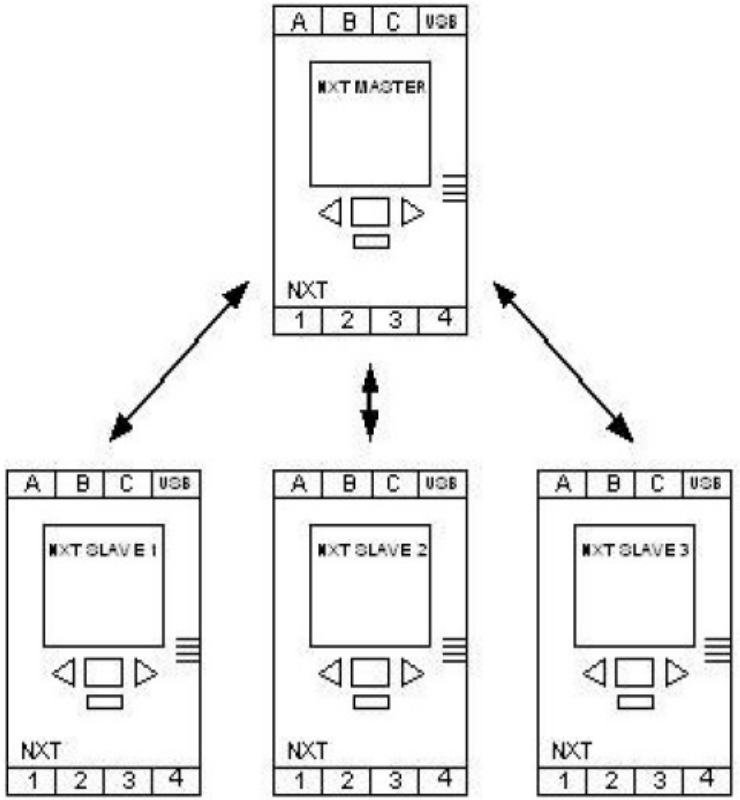
LEGO Java Operating System

In particolare lejos NXJ è un ambiente di programmazione Java per LEGO Mindstorms NXT ®. Esso consente di programmare i robot LEGO ® in Java. Comprende:

- Comandi di linea per compilare e per lanciare programmi di supporto
- API Lejos con sorgenti per creare programmi java da installare in NXT
- API Lejos con sorgenti per creare programmi java da installare su PC che comunicano con NXT
- API di altri fornitori per creare programmi java da installare su PC che comunicano con NXT
- Sorgenti di esempi da installare su NXT e su PC
- Driver USB per NXT per piattaforma MAC / Window / Linux

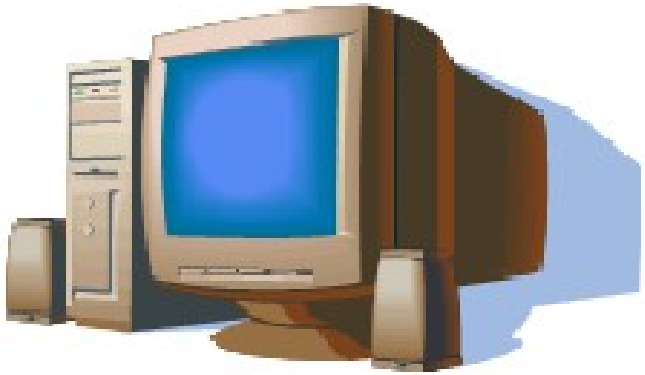
Comunicazione con altri 3 NXT

- Comunicazione Bluetooth
- Un NXT è Master
- Altri NXT sono Slaves (massimo 3)
- 4 canali di comunicazione
- Canale 0 da Slaves a Master
- Canale 1, 2, 3 dal Master verso gli Slaves 1, 2, 3
- Comunicazione 1-1

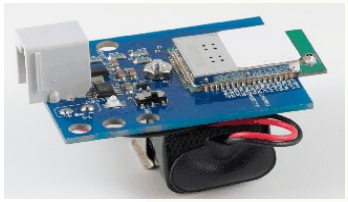


Comunicazione con dispositivi

Bluetooth



Bluetooth

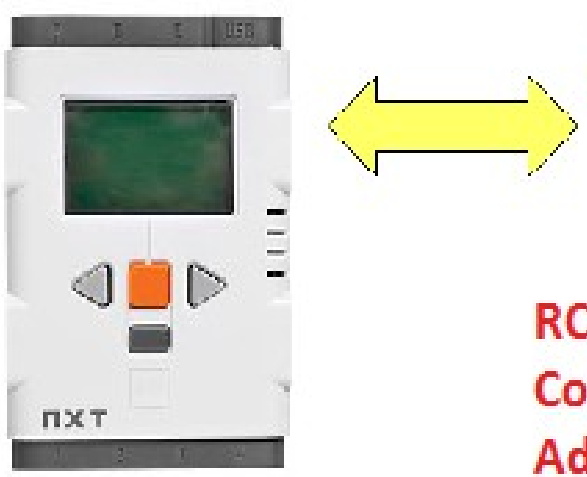


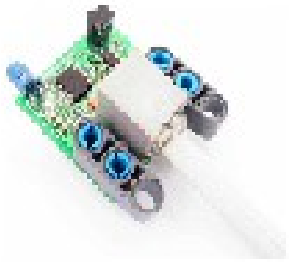
WIFI

Sensore WIFI

Altre forme di comunicazione

Ricevitore Bluetooth
GPS





RCX to NXT
Communication
Adapter (NRLink-Nx)

Tipo di comunicazione da PC

- Verso NXT usando il protocollo LCP (*Lego Communication Protocol*)
  - Con firmware originale NXT nel brick
  - Con firmware Java nel brick
- Verso e da NXT con programmi java
  - Con firmware Java nel brick
  - Programma java di spedizione dati e comandi su PC
  - Programma lejos di ricezione dati e risposta ai comandi nel brick
  - Con protocollo LCP o protocollo creato ad hoc

Classe NXTCommand

Classe che contiene tutti i comandi da/per NXT

spediti in questo formato (protocollo LCP)

- Byte 0 = Tipo comando (es. comando *sistema con risposta*)
- Byte 1 = Comando (es. *riproduci un suono contenuto in un file*)
- Byte 4...Byte n (dati informativi e necessari al comando, es. *nome del file contenente il suono da riprodurre*)

Tipi di comandi (NXTProtocol)

Tipo	Byte	Descrizione
DIRECT_COMMAND_REPLY	0x00	Comando diretto con risposta
SYSTEM_COMMAND_REPLY	0x01	Comando di sistema con risposta
REPLY_COMMAND	0x02	Comando di risposta
DIRECT_COMMAND_NOREPLY	0x80	Comando diretto senza risposta
SYSTEM_COMMAND_NOREPLY	0x81	Comando di sistema senza risposta

Comandi di sistema (NXTProtocol)

Tipo	Byte	Descrizione
OPEN_READ	0x80	Apri file in lettura
OPEN_WRITE	0x81	Apri file in scrittura
READ	0x82	Leggi
WRITE	0x83	Scrivi
CLOSE	0x84	Chiude connessione
DELETE	0x85	Cancella file
FIND_FIRST	0x86	Trova il primo file nella directory e con il filtro specificati
FIND_NEXT	0x87	Trova il prossimo file nella directory e con il filtro specificati
GET_FIRMWARE_VERSION	0x88	Ritorna informazioni sul firmware
SET_BRICK_NAME	0x98	Imposta nome NXT
GET_DEVICE_INFO	0x9B	Ottiene informazioni sul dispositivo NXT
DELETE_USER_FLASH	0xA0	Cancella le informazioni nella memory Flash
POLL_LENGTH	0xA1	Ottiene la lunghezza di messaggio di risposta ad un comando
POLL	0xA2	Ottiene il messaggio di risposta ad un comando

Comandi diretti (NXTProtocol)

Tipo	Byte	Descrizione
START_PROGRAM	0x00	Lancia un programma nell'NXT
STOP_PROGRAM	0x01	Ferma un programma nell'NXT
PLAY_SOUND_FILE	0x02	Riproduce un suono di un file
PLAY_TONE	0x03	Riproduce un tono
SET_OUTPUT_STATE	0x04	Imposta parametri per un motore
SET_INPUT_MODE	0x05	Imposta parametri per un sensore
GET_OUTPUT_STATE	0x06	Ottiene parametri di un motore
GET_INPUT_VALUES	0x07	Ottiene parametri di un sensore
RESET_SCALED_INPUT_VALUE	0x08	Azzera impostazioni di un sensore

Comandi diretti (NXTProtocol)

Tipo	Byte	Descrizione
MESSAGE_WRITE	0x09	Spedisce un messaggio ad una casella inbox NXT
RESET_MOTOR_POSITION	0x0A	Azzera contagiri del motore
GET_BATTERY_LEVEL	0x0B	Ritorna il livello della batteria
STOP_SOUND_PLAYBACK	0x0C	Ferma la riproduzione di un suono
KEEP_ALIVE	0x0D	Lascia acceso l'NXT
LS_GET_STATUS	0x0E	Ritorna lo stato del sensore a ultrasuoni
LS_WRITE	0x0F	Imposta informazioni al sensore a ultrasuoni
LS_READ	0x10	Legge informazioni di un sensore a ultrasuoni
GET_CURRENT_PROGRAM_NAME	0x11	Ottiene il nome del programma che sta girando

Es. Leggi versione firmware

Comando spedito

- Byte 0 = 0x01 tipo comando ***sistema con risposta***
- Byte 1 = 0x88 Comando GET_FIRMWARE_VERSION

Comando ricevuto

- Byte 0 = 0x02 tipo comando ***risposta***
- Byte 1 = 0x88 Comando GET_FIRMWARE_VERSION
- Byte 2 = Codice di ritorno: =**0** – OK ; >**0** KO
- Byte 3 = 0x02 (Versione Minor del protocollo)
- Byte 4 = 0x01 (Versione Major del protocollo)
- Byte 5 = 0x03 (Versione Minor del firmware)
- Byte 6 = 0x01 (Versione Major del firmware)

Versione Protocollo 1.2 e versione firmware 1.3

Specifiche per un'APP che comunica con Alex

- L'app deve inviare comandi LCP per mandare avanti Alex e fermarlo
- L'app deve spedire e ricevere dati in un pacchetto Lejos ad programma in attesa in Alex

Architettura

TrheadAlex

Clic su Avanti

Comando LCP
Motor.B.forward()
Motor.C.forward()

Comando LCP
Motor.B.stop()
Motor.C.stop()

Clic su Stop

Waiting

Programma
Java
BTReceive

Scrittura/lettura
Pacchetto Lejos
su socket
Bluetooth

BTSend

Clic su Spedisci

Maschera Alex

Bottone “Avanti” richiama **ThreadAlex** e manda avanti il robot

Bottone “Stop” richiama **ThreadAlex** e ferma il robot

INIZIO

Casella di Testo impostata a “INIZIO” cambia con l'avanzare dei programmi

Bottone “Spedisci” richiama **BTSend** e inizia la comunicazione con **BTReceive** in attesa dentro il robot

Librerie necessarie

Per l'app:

- log4j-1.2.15.jar - Libreria che gestisce i messaggi del log
- pccomm.jar – Libreria che contiene le classi di comunicazione da pc a NXJ
- lejos.pc.comm.NXTCommAndroid – classe aggiuntiva per gestire la comunicazione bluetooth verso e da sistema Android

Per NXJ:

- org.lejos.sample.BTreceive – Classe compresa nel pacchetto lejos

Contenuto APP

- ninux.robotic.MainActivity - Un'activity principale
- ninux.TreadAlex - Una classe estensione di un thread che gestisce i movimenti di Alex con il protocollo LCP
- AndroidManifest.xml – Contiene i parametri generali dell'APP
- activity_rob_alex.xml – Layout grafica della maschera
- string.xml – elenco valori utilizzati nell'app
- drawable-hdpi, drawable-ldpi, drawable-mdpi, drawable-xhdpi – dir contententi le icone dell'app nelle varie dimensioni

AndroidManifest.xml

```
 xmlns:android="http://schemas.android.com/apk/res/android"
    package="ninux.robotic"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
   <uses-permission
android:name="android.permission.BLUETOOTH"></uses-permission>
      <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"></uses-
permission>
 <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-
permission>
    <application
        android:allowBackup="true"
        android:icon="@drawable/alex"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
       <activity
            android:name="ninux.robotic.MainActivity"
            android:label="@string/title_activity_alex" >
             <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
       </activity>
```

activity_rob_alex.xml
Layout per bottone e testo

```
<Button android:id="@+id/button1"
android:text="@string/test_1"
android:textSize="30sp"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:onClick="comandaAlex" />

   <TextView
       android:id="@+id/messaggio"
       android:text="@string/testo"
       android:layout_width="fill_parent"

android:layout_height="wrap_content"
       android:layout_marginBottom="10dip"
       android:layout_marginTop="10dip"
       android:textSize="40sp"
       android:textColor="#FF0000"
       android:textAppearance="?
android:attr/textAppearanceMedium" />
```

string.xml

```
<resources>
    <string name="hello">Alex</string>
    <string
name="app_name">Alex</string>
    <string
name="test_1">Avanti</string>
    <string name="test_2">Stop</string>
    <string
name="test_3">Spedisci</string>
    <string
name="testo">INIZIO</string>
</resources>
```

MainActivity: Esecuzione connessione

```
public static NXTConnector connect(final
CONN_TYPE connection_type) {

   NXTConnector conn = new NXTConnector();

   switch (connection_type) {
      case LEGO_LCP:
      conn.connectTo("btspp://Alex",
NXTComm.LCP);
      break;
      case LEJOS_PACKET:
      conn.connectTo("btspp://");
      break;
   }

   return conn;

}
```

MainActivity: Avanti e Stop

```
    public void comandaAlex(View view) {
        textView = (TextView) findViewById(R.id.messaggio);
        try {
            textView.setText("avviato");
            ThreadAlex.avviato=true;
            if (alex==null) {
                alex = new ThreadAlex();
                alex.start();
            }
        } catch (Exception e) {
Log.e(TAG, "Mancato avvio ComandaAlex:" + e.getMessage(),
e);
        }
    }
    public void stopAlex(View view) {
        textView = (TextView) findViewById(R.id.messaggio);
        try {
            textView.setText("stop");
            if (alex!=null) {
                ThreadAlex.avviato=false;
                alex=null;
            }
        } catch (Exception e) {
Log.e(TAG, "Mancato avvio ComandaAlex:" + e.getMessage(),
e);
        }
```

MainActivity: Spedisci

```
public void sendAlex(View view) {
   textView =
(TextView)findViewById(R.id.messaggio);
        (new BTSend()).start();
   }
}
```

Creare la connessione

La connessione **leJOS** è fatta tramite il metodo standard **lejos.pc.comm.NXTConnector**.

leJOS individuerà l'ambiente Android e tenterà di caricare il programma

lejos.pc.comm.NXTCommAndroid.

- Connessione LCP da ThreadAlex:

```
NXTConnector conn =
MainActivity.connect(
MainActivity.CONN_TYPE.LEGO_LCP);
```

- Connessione Lejos da BTSend:

```
NXTConnector conn =
MainActivity.connect(
MainActivity.CONN_TYPE.LEJOS_PACKET);
```

ThreadAlex: Avanti e Stop

```
public void run() {
Motor.C.setRegulationMode(NXTCommand.REGULATION_MODE_M
OTOR_SYNC);
Motor.B.setRegulationMode(NXTCommand.REGULATION_MODE_M
OTOR_SYNC);
      setName(TAG + " thread");
      conn =
MainActivity.connect(MainActivity.CONN_TYPE.LEGO_LCP);
      Motor.B.forward();
      Motor.C.forward();
      while (avviato) {
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            Log.e(TAG, "Thread.sleep error", e);
         }
      }
      Motor.B.stop();
      Motor.C.stop();
      Sound.buzz();
      closeConnection();
}
```

BTSend: Leggi e Scrivi in socket Bluetooth

```
public void run() {
      Log.d(TAG, "BTSend run");
      conn =
MainActivity.connect(CONN_TYPE.LEJOS_PACKET);
      dos = new
DataOutputStream(conn.getOutputStream());
      dis = new DataInputStream(conn.getInputStream());
      int x;
      for (int i = 0; i < 100; i++) {
         try {
            dos.writeInt((i * 30000));
            dos.flush();
            yield();
            x = dis.readInt();
   Log.d(TAG, "spedito:" + i * 30000 + " ricevuto:" +
x);

MainActivity.sendMessageToUIThread("spedito:" + i *
30000 + " ricevuto:" + x);
            yield();
         } catch (IOException e) {
            Log.e(TAG, "Error ... ", e);
         }
```

BTReceive: Leggi e Scrivi in socket Bluetooth

```
while (true)     {
   BTConnection btc =
Bluetooth.waitForConnection();
   DataInputStream dis =
btc.openDataInputStream();
   DataOutputStream dos =
btc.openDataOutputStream();
   for(int i=0;i<100;i++) {
      int n = dis.readInt();
      LCD.drawInt(n,7,0,1);
      LCD.refresh();
      dos.writeInt(-n);
      dos.flush();
   }
   dis.close();
   dos.close();
   Thread.sleep(100);
}
```

Uso della classe Handler

- Le **Views** in Android sono a singolo thread e i controlli di integrità possono dar luogo ad un comportamento imprevedibile e vi è il rischio di perdita di memoria
- Un **Handler** consente di inviare ed elaborare oggetti **Message** e **Runnable** associati ad un oggetto **MessageQueue** di un thread.
- Quando si crea un nuovo **Handler**, questo viene associato al thread che lo ha creato e a una coda di messaggi
- Da quel momento in poi, messaggi e runnables andranno nella coda di messaggi e saranno serviti appena possibile in ordine di arrivo.

Classi usate nell'APP per la classe Handler

- **android.os.Bundle** è una classe atta a mappare valori tramite utilizzo di chiavi univoche (similmente ad una tabella hash).
- **android.os.Message** è una classe che definisce un messaggio contenente una descrizione e un arbitrario oggetto che può essere spedito ad un Handler.

MainActivity: Uso della classe Handler

```
   public static UIMessageHandler
mUIMessageHandler;
   public static void sendMessageToUIThread(
                              String message) {
      Bundle b = new Bundle();
      b.putString(MESSAGE_CONTENT, message);
      Message message_holder = new Message();
      message_holder.setData(b);

mUIMessageHandler.sendMessage(message_holder);
   }

class UIMessageHandler extends Handler {
   public void handleMessage(Message msg) {
     textView.setText((String)
           msg.getData().get(MESSAGE_CONTENT));
      }
   }
```

MainActivity: Uso della classe Handler

```
   public static UIMessageHandler
mUIMessageHandler;
   public static void sendMessageToUIThread(
                              String message) {
      Bundle b = new Bundle();
      b.putString(MESSAGE_CONTENT, message);
      Message message_holder = new Message();
      message_holder.setData(b);

mUIMessageHandler.sendMessage(message_holder);
   }

class UIMessageHandler extends Handler {
   public void handleMessage(Message msg) {
     textView.setText((String)
           msg.getData().get(MESSAGE_CONTENT));
      }
   }
```

MainActivity: Uso della classe Handler

```
   public static UIMessageHandler
mUIMessageHandler;
   public static void sendMessageToUIThread(
                              String message) {
      Bundle b = new Bundle();
      b.putString(MESSAGE_CONTENT, message);
      Message message_holder = new Message();
      message_holder.setData(b);

mUIMessageHandler.sendMessage(message_holder);
   }

class UIMessageHandler extends Handler {
   public void handleMessage(Message msg) {
     textView.setText((String)
           msg.getData().get(MESSAGE_CONTENT));
      }
   }
```

Evoluzione di Lego Mindstorms

Nella seconda metà del 2013 uscirà

Lego Mindstorms EV3 con le seguenti caratterisiche:

- Può essere programmato direttamente nel Brick
- Più memoria e potenza di elaborazione
- Firmware Linux
- Porte USB e slot espansione SD
- Supporto WIFI
- Piena compatibilità con piattafoma IOS e Android
- Manuale di costruzione 17 robot diversi

Link

- http://lejos.sourceforge.net
- http://mindstorms.lego.com/
- http://www.youtube.com/watch?v=yOUW_l66iqw

Per informazioni: ruggeridany@yahoo.it

Domande?