Università degli Studi di Bologna – Campus di Cesena

Relazione relativa alla progettazione di RssCollect: un' applicazione Android per aggregare i feed relativi ai podcast radiofonici.

Giunchi Massimiliano

Corso di Programmazione ad Oggetti - A.A. 2014/2015

Corso di laurea in Ingegneria e Scienze Informatiche

Indice

Capitolo 1

Analisi

1.1 Requisiti

Si vuole realizzare un'applicazione Android che permetta di aggregare i feed relativi ai podcast provenienti da alcuni network radiofonici.
Le funzionalità necessarie sono le seguenti:

- Visualizzare una lista di emittenti radiofoniche con relativo logo e caricamento dell'immagini in maniera lazy.
- Selezionata l'emittente, visualizzare la lista delle trasmissioni con la data del podcast disponibile più recente e l'immagine della trasmissione (se presente).
- Selezionata la trasmissione, visualizzare gli episodi disponibili con alcune informazioni (descrizione, titolo, data pubblicazione).
- Riprodurre il podcast dell'episodio selezionato.
- Aggiungere/rimuovere una trasmissione ad/da una lista dei preferiti
- Visualizzare la lista dei preferiti con possibilità di modifica/cancellazione.
- Ricerca del titolo di una trasmissione a partire da una descrizione libera.

1.2 Problema

Uno dei primi problemi che è stato necessario affrontare è quello relativo alla sorgente dei dati: se da una parte si sarebbe potuto memorizzare all'interno dell'applicazione un elenco di emittenti con relative trasmissioni e per ognuna l'url del feed, questo avrebbe comportato la necessità di disinstallare l'applicazione, eseguire le modifiche al codice ed eseguire la re installazione ogni qual volta l'elenco delle emittenti o la programmazione avesse subito una variazione; di conseguenza si è deciso di implementare un semplice db (MySql) in un server remoto al quale l'applicazione si collega ad ogni avvio con conseguente scaricamento in locale dei dati; questa soluzione tuttavia comporta la necessità di dover interrogare il db in remoto e di organizzare la struttura dati che dovrà contenere le informazioni rese dall'interrogazione (problematica comune anche alla funzione di ricerca tramite testo libero); inoltre si deve decidere

se l’interrogazione dovrà essere globale (si scarica l’intero db) oppure se recuperare di volta in volta solo le informazioni necessarie.
Un secondo problema è quello della rappresentazione per mezzo di una lista “scrollable”, delle emittenti/trasmissioni: o meglio per ognuna si vuole visualizzare il logo dell’emittente e nel caso in cui sia disponibile sostituirlo con quello della trasmissione selezionata (nel caso della lista delle trasmissioni).
Sempre per le trasmissioni, avendo scelto per la visualizzazione della data dell’ultimo podcast pubblicato, è necessario trovare una metodologia per analizzare il file XML associato in background con la conseguente estrazione della data più recente; questo problema si è di fatto riversato in quello di dover implementare il parsing del file XML.

Capitolo 2

Design

2.1 Architettura

Si è applicato il pattern MVC separando il modello, dal controllore e dall'interfaccia utente; quest'ultima è stata realizzata tramite classi che estendono `Activity` e `ListActivity`; la separazione tra i vari strati del pattern è stata realizzata tramite l'utilizzo di interfacce.

Di seguito riporto i componenti principali:

Modello: contiene le seguenti interfacce:

- `IEmittente`: definisce un' emittente; la struttura rispetta quella del db remoto dalla quale è stata prelevata.
- `ITrasmissione`: definisce un'emittente; anche in questo caso la struttura rispetta quella del db sul server.
- `ITrasmissioneXML`: definisce un'emittente rappresentata nel file XML associato al feed (rispetto la precedente quest'ultima descrive una trasmissione rappresentata all'interno di un file).
- `IHttpClient`:contiene i metodi per il collegamento al server remoto.
- `IDownloadXML`: scarica un file XML.
- `IGetDataXML`: esegue il parsing del file XML (metodi per poter estrarre i valori dei tag di interesse).

Controllore: contiene le seguenti classi/interfacce:

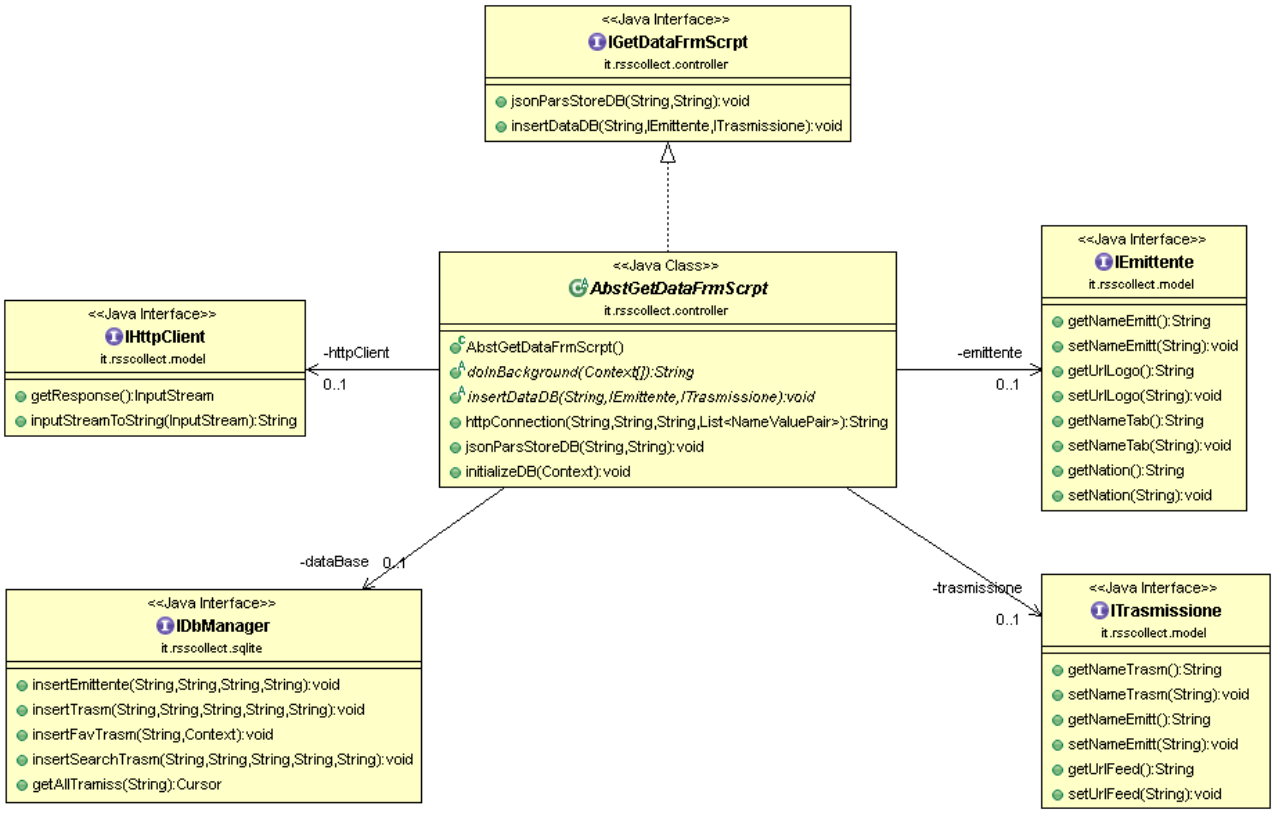
- `AbstGetDataFrmScrpt` classe astratta che implementa `IGetDataFrmScrpt` ed è preposta a settare le url (o meglio i parametri che compongono le url) per le richieste al server remoto; si specializza in `GetDataFrmScrpt`, utilizzata per recuperare le emittenti e le trasmissioni e `SrchDataFrmScrpt` che invece gestisce le ricerche, ossia le trasmissioni che rispondono ai criteri di ricerca.
- `IGetDataFrmScrpt`: esegue il parsing delle risposte del server che sono in formato JSON ed inserisce i dati ricevuti nel db locale.

- `AbstractListViewAdptr`: classe astratta che estende la classe `ArrayAdapter` e si specializza nelle classi `CstmListViewAdptr` che gestisce le liste `TransmList` e `SearchList`, mentre `CstmListFavViewAdptr` gestisce i preferiti, ossia `FavList`
- `IGetDataFrmSQL` esegue query nel db locale che restituiscono strutture dati (`List<>`) per gli `ArrayAdapter`.

View: contiene le seguenti classi:

- `SplashActivity`: `activity` di apertura che introduce l'utente alla `MainActivity`: in realtà, dopo il controllo della presenza di una connessione e l'autenticazione,esegue in background ad una progress bar visualizzata per l'utente, alcune importanti operazioni quali il collegamento al server con il conseguente scaricamento della tabella delle emittenti.
- `MainActivity`: `activity` principale che visualizza l'elenco delle emittenti e dalla quale si accede alle funzionalità principali dell'applicazione nonché alle altre `activity`.
- `GenerictList`: classe che estende `ListActivity` e rappresenta una lista "generica" nel senso che raccoglie le caratteristiche comuni delle liste in cui si specializza.
- `TransmList`: visualizza l'elenco delle trasmissioni associate ad un'emittente.
- `SearchList`: visualizza l'elenco delle trasmissioni corrispondenti ai criteri di ricerca.
- `FavList`: visualizza l'elenco delle trasmissioni preferite.
- `XmlParseActivity`: visualizza l'elenco degli episodi e selezionato quello di interesse, apre il file mp3 associato (la puntata).

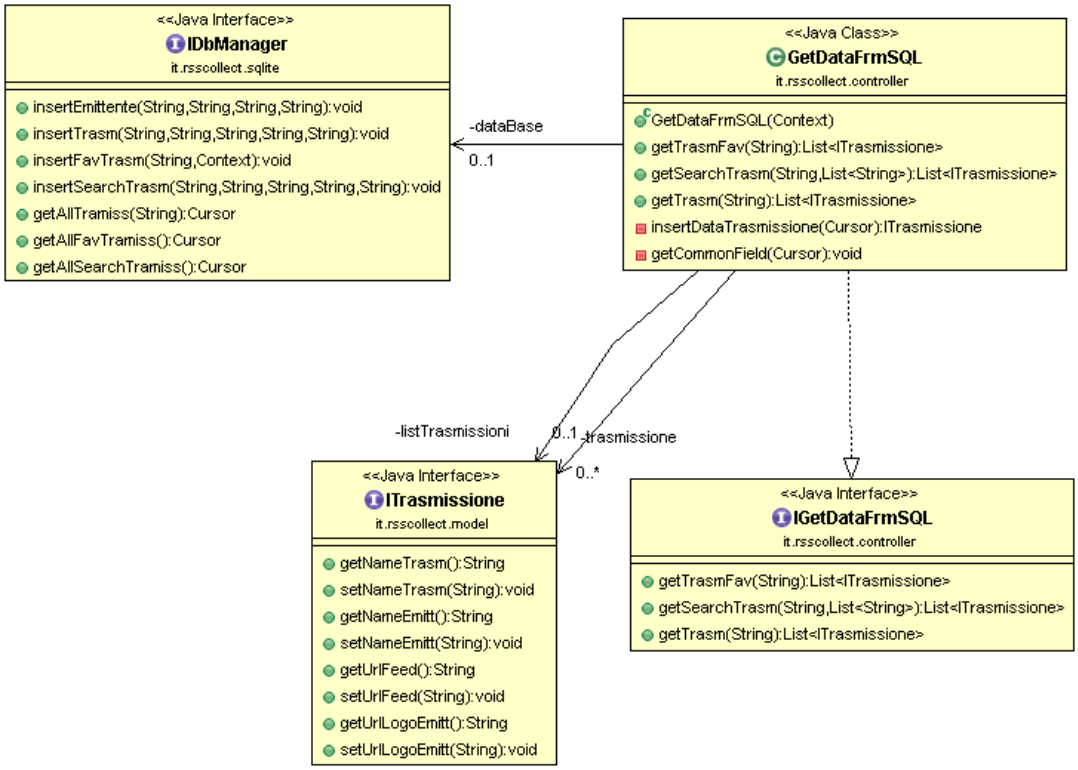
Il diagramma UML 1 seguente mostra i legami tra le classi/interfacce che si adoperano all'interrogazione del server, lo scaricamento dei dati e la relativa archiviazione in locale.



UML 1

Il seguente diagramma UML mostra i legami tra le interfacce del db locale `IDbManager`, `IGetDataFrmSQL` che setta le liste dati (contenenti le trasmissioni) per gli adapter utilizzati nelle user interface che estendono le `ListActivity`.

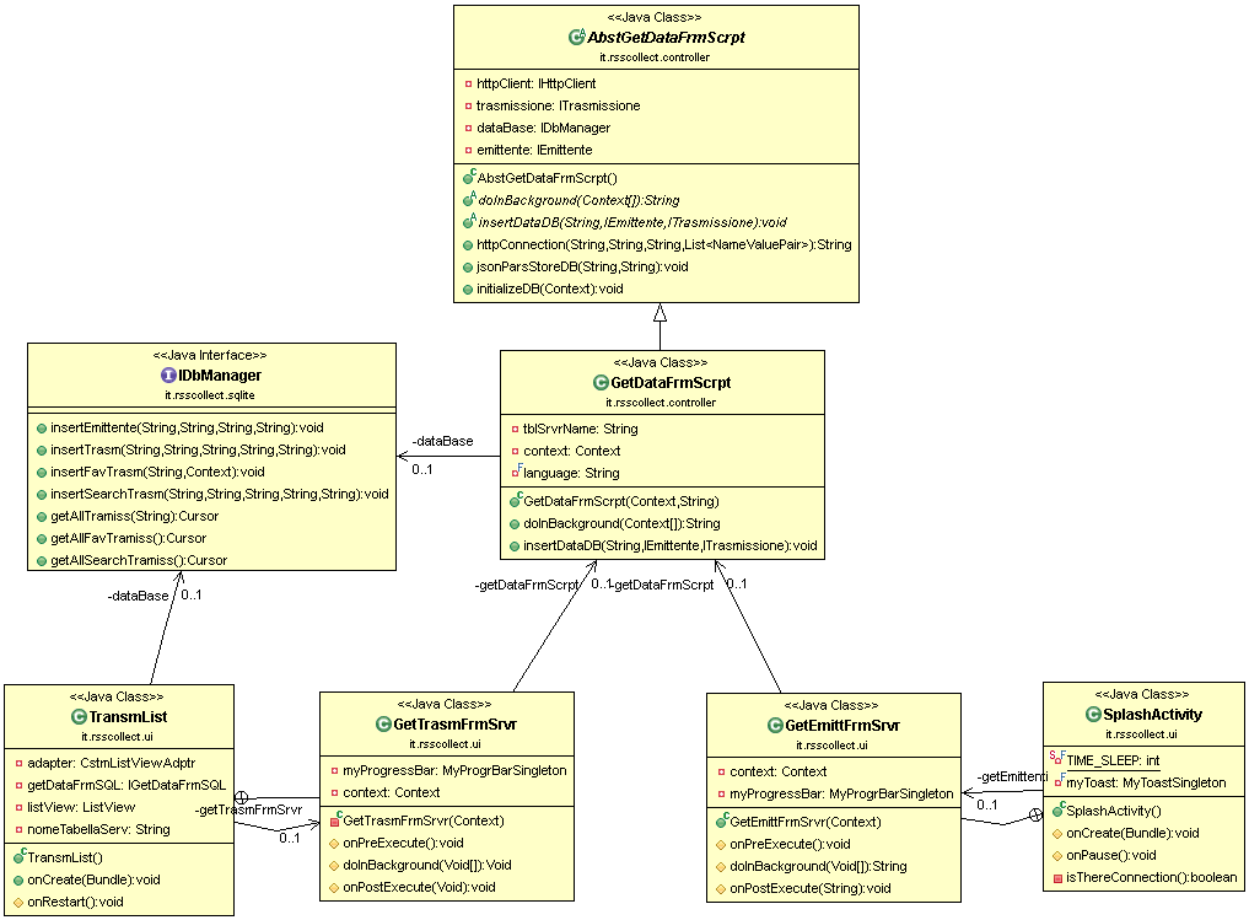
`IDbManager` è l'interfaccia che implementa le query per il Db locale, la classe `GetDataFrmSQL` si occupa di implementare i metodi dell'interfaccia `IGetDataFrmSQL` restituendo delle liste (`List<ITrasmissione>`) che saranno poi "date in pasto" agli `adapter` utilizzati per le `ListActivity` che visualizzano le emittenti, le trasmissioni ed i preferiti.


<<Java Interface>>
IDbManager
it.rsscollect.sqlite
insertEmittente(String,String,String,String):void
insertTrasm(String,String,String,String,String):void
insertFavTrasm(String,Context):void
insertSearchTrasm(String,String,String,String,String):void
getAllTramiss(String):Cursor
getAllFavTramiss():Cursor
getAllSearchTramiss():Cursor
-dataBase
0..1
<<Java Class>>
GetDataFrmSQL
it.rsscollect.controller
GetDataFrmSQL(Context)
getTrasmFav(String):List<ITrasmissione>
getSearchTrasm(String,List<String>):List<ITrasmissione>
getTrasm(String):List<ITrasmissione>
insertDataTrasmissione(Cursor):ITrasmissione
getCommonField(Cursor):void
-listTrasmissioni
0..1
-trasmissione
0..*
<<Java Interface>>
ITrasmissione
it.rsscollect.model
getNameTrasm():String
setNameTrasm(String):void
getNameEmitt():String
setNameEmitt(String):void
getUrlFeed():String
setUrlFeed(String):void
getUrlLogoEmitt():String
setUrlLogoEmitt(String):void
<<Java Interface>>
IGetDataFrmSQL
it.rsscollect.controller
getTrasmFav(String):List<ITrasmissione>
getSearchTrasm(String,List<String>):List<ITrasmissione>
getTrasm(String):List<ITrasmissione>

UML 2

2.2 Design dettagliato

Di seguito descrivo le classi/interfecce relativa alla parte di interrogazione del server e archiviazione in locale – v.di UML di seguito



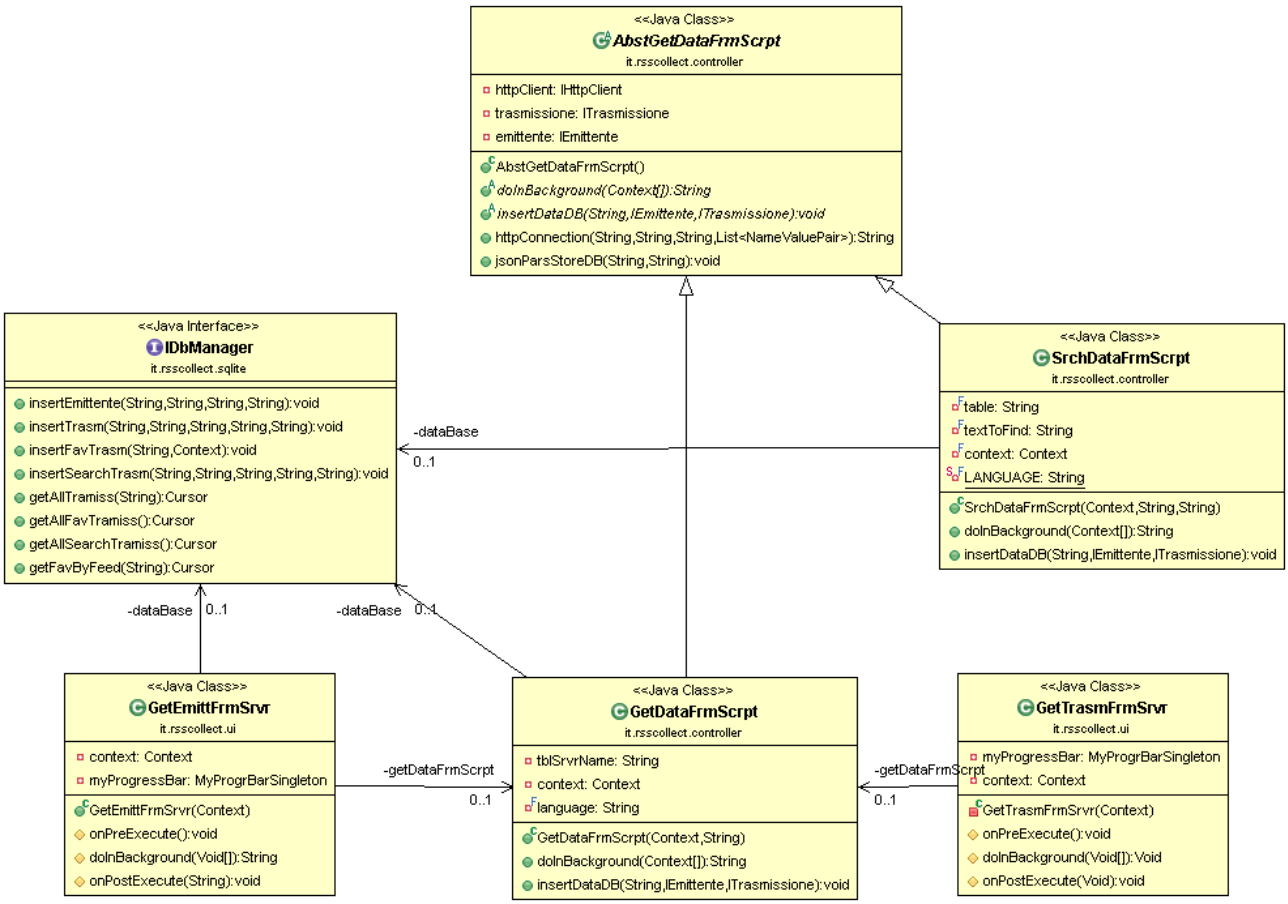
UML 3

`GetEmittFrmSrv` classe preposta al recupero delle emittenti dal server; essendo una classe utilizzata solo dall'activity `SplashActivity`, si è deciso di realizzarla in maniera innestata; recuperate le emittenti, sono memorizzate nel db locale (interfaccia `IDbManager`).

`GetTrasmFrmSrvr` è la classe preposta al recupero delle trasmissioni; essendo una classe utilizzata solo dalla listActivity `TransmList`, è stata implementata in maniera innestata (il collegamento al server avviene solo se l'interrogazione del db locale – tramite interfaccia `IDbManager` - fornisce risposta nulla); questo perché nel momento in cui viene opzionata un'emittente da parte dell'utente, viene dapprima verificato se le trasmissioni associate sono presenti nel db locale, ed in caso negativo prelevate dal server (questo al fine di minimizzare i

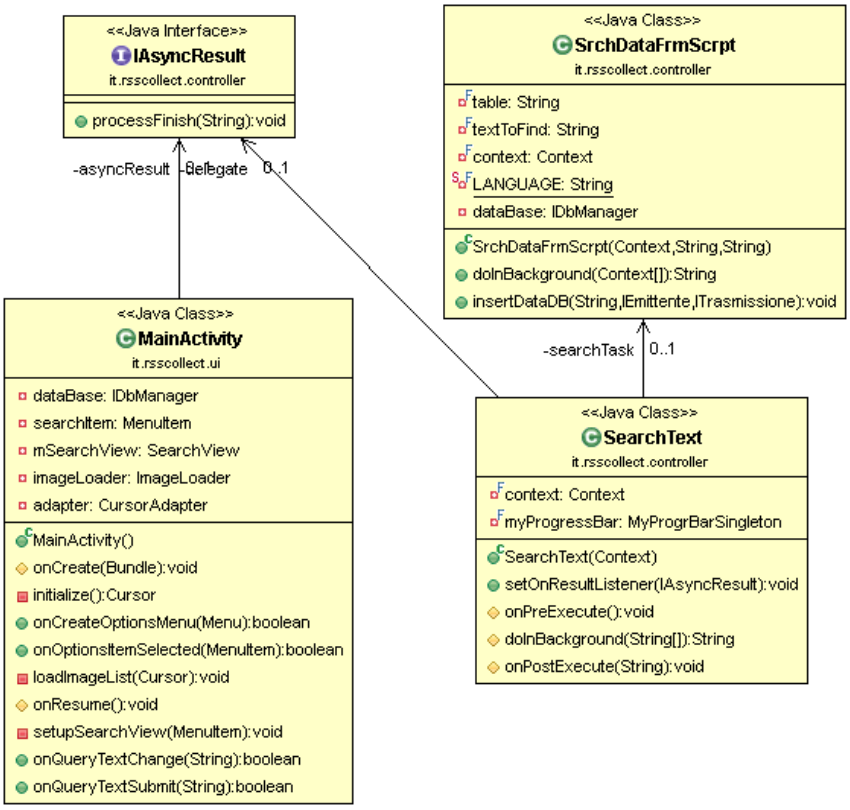
tempi di attesa in quanto sono scaricate solo le informazioni di interesse per l'utente).

`GetDataFrmScrpt` è preposta a settare le url per le richieste al server remoto ed a eseguire il parsing delle risposte che sono in formato JSON; classe analoga, ma chiamata a seguito di una ricerca è la classe `SrchDataFrmScrpt` preposta alla ricerca di trasmissioni; il metodo astratto `insertDataDB` è richiamato all'interno del metodo `jsonParsStoreDB` e specializzato nelle sotto classi (template method).



UML 4

Sempre relativamente alla funzione di ricerca, nel diagramma UML 5 è riportato il legame tra la `MainActivity` dalla quale parte la ricerca per poi essere gestita dalla classe `SearchText`: essa (tramite `SrchDataFrmScrpt`) invia la richiesta e archivia la risposta: l'interfaccia `IAsyncresult` attende che alla classe `SearchText` (che estende `AsyncTask`) sia passato il risultato (metodo `setOnResultListener` notifica che il risultato è stato settato, o meglio che il task ha completato l'operazione in background).



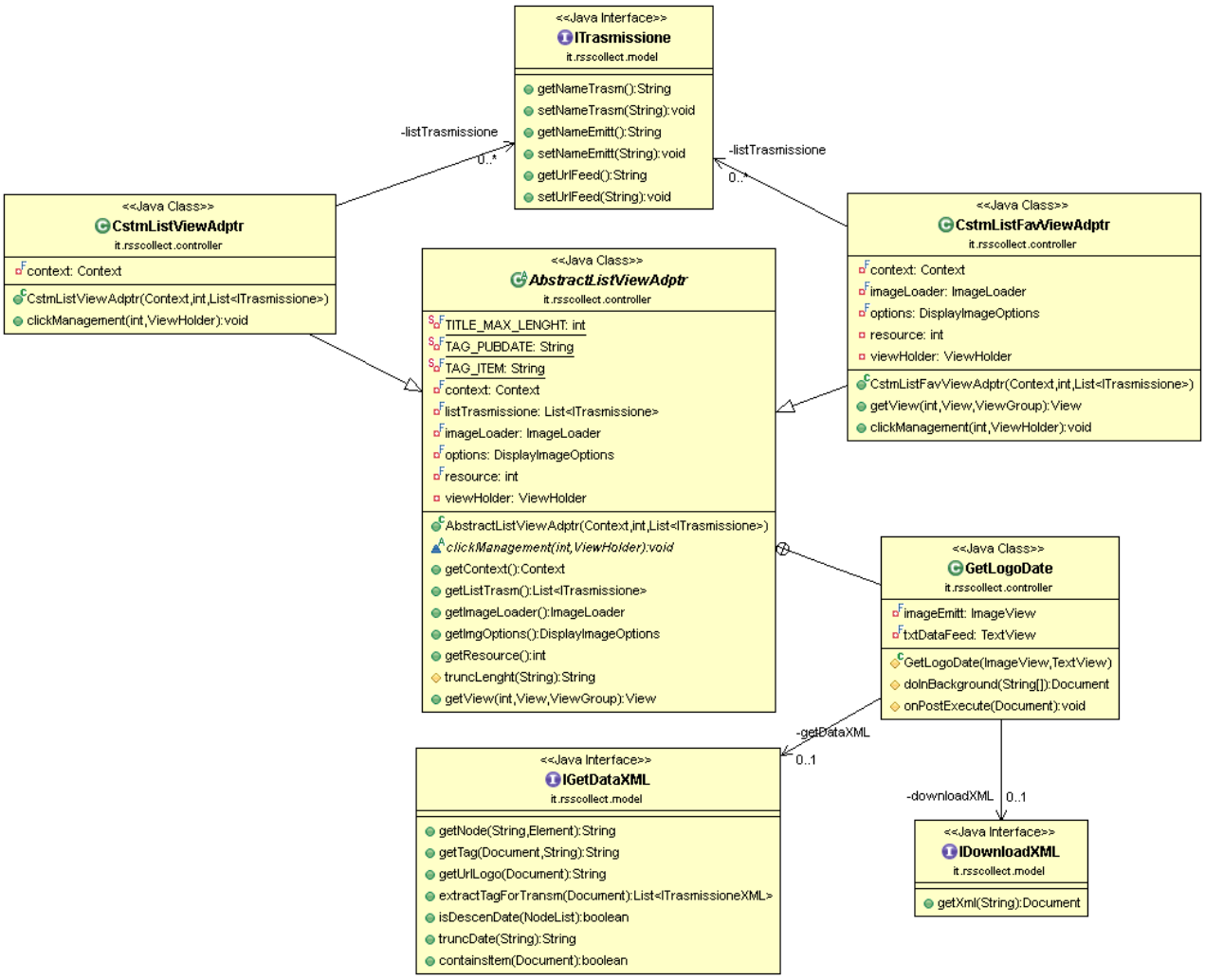
UML 5

La classe `MainActivity` si compone della classe `ImageLoader` la quale appartiene una libreria esterna - Universal Image Loader - che permette di visualizzare le immagini in maniera lazy, ossia sono caricate (e memorizzate nella cache) solo quando visualizzate dall'utente; tale classe è utilizzata ogni qual volta si debbano visualizzare immagini, ossia in tutte le `Listactivity`.

Illustro ora come sono stati progettati gli adapter che fungono da "intermediario" tra le `ListActivity` (`TransmList`, `SearchList`, `FavList`) ed i dati estratti attraverso la classe `GetDataFrmSQL` (UML 6); come citato nel primo capitolo, una caratteristica comune a queste liste è la funzionalità di estrarre la data del podcast più recente e l'immagine della trasmissione; questo è stato realizzato tramite la classe `GetLogoDate` che è preposta a ricercare per ciascun elemento della lista, la data dell'ultimo podcast pubblicato e l'immagine della trasmissione ove disponibile (altrimenti si lascia il logo dell'emittente); dovendo eseguire queste operazioni in background, essa implementa `AsyncTask`; anche in questo caso, essendo utilizzata solo dalla classe `AbstractListViewAdptr`, si è deciso di crearla innestata.

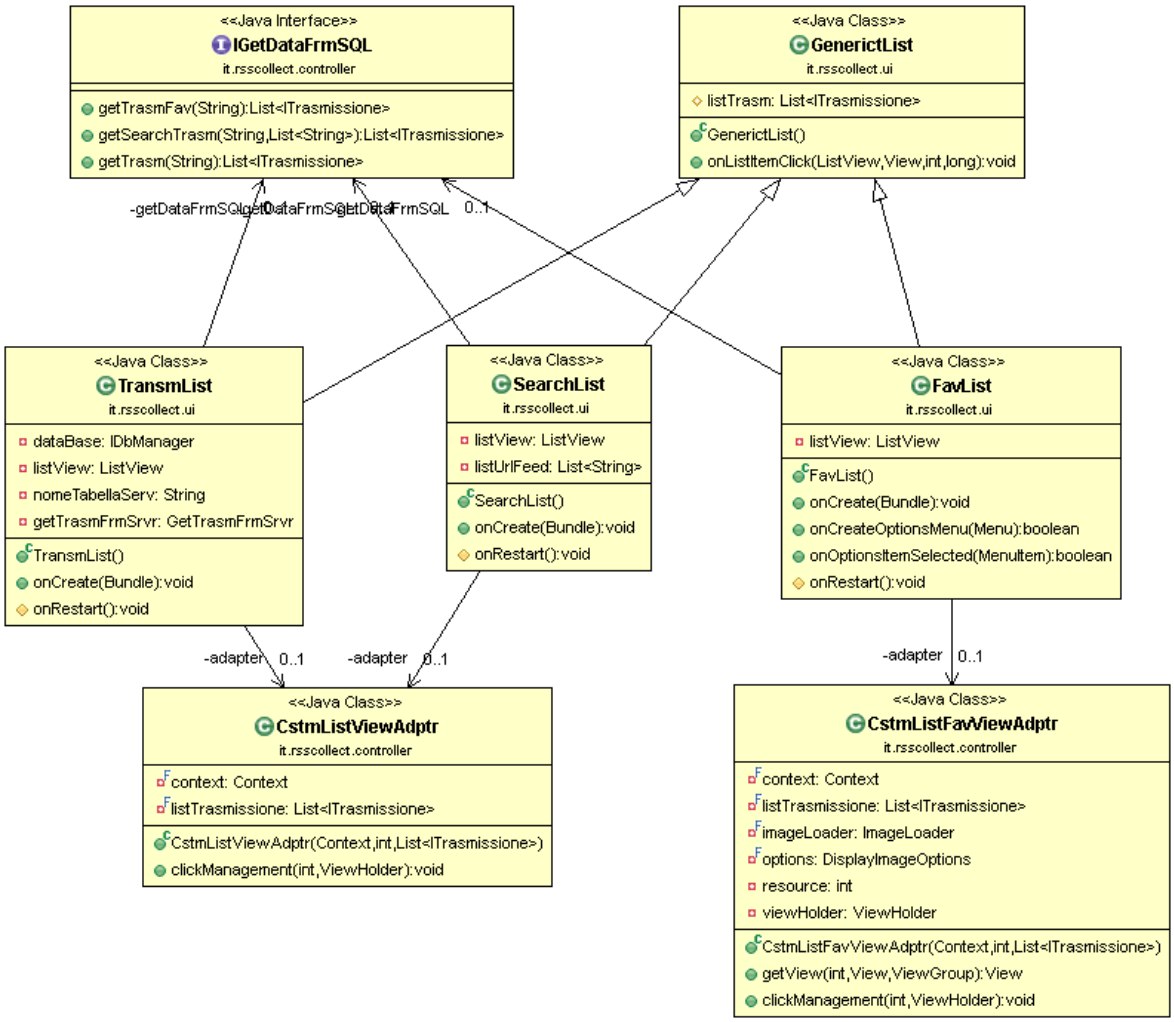
La classe `AbstractListViewAdptr` è una classe astratta che estende `ArrayAdapter` e si specializza in `CstmListViewAdptr` che rappresenta i dati per le liste `TransmList` e `SearchList`, mentre `CstmListFavViewAdptr` gestisce i preferiti, ossia `FavList`; questa differenziazione si è

resa necessaria dal fatto che la lista dei “preferiti” presenta un layout differente rispetto alle altre.
Il parsing dei dati è affidato alla classe `IGetDataXML`: la realizzazione di questa classe ha comportato la risoluzione di un problema inizialmente imprevisto; la maggior parte dei file XML relativi ai feed ha, infatti, una struttura in cui le puntate delle trasmissioni più recenti (ma il discorso vale anche per le notizie, ecc.) sono poste in testa e man mano che si scende vi sono quelle meno recenti; ecco che inizialmente si era pensato di prendere in esame il primo nodo (contenente il tag <*item*> e da qui estrarre i valori dei nodi dei tag di interesse); tuttavia questa regola in alcuni rari casi è violata, ossia nel primo nodo vi è l’episodio meno recente e nell’ultimo il più aggiornato.
Questo ha comportato la necessità di implementare un metodo `isDescenDate` il quale si occupa di restituire *true* se l’ordine delle date è decrescente (primo nodo, articolo più recente) o il contrario; in base al risultato di questo controllo si è in grado di sapere da quale nodo è necessario iniziare l’estrazione delle informazioni.
Il metodo astratto `clickManagement` è utilizzato all’interno del metodo `getView` ed implementato nelle 2 sottoclassi.


<<Java Interface>>
ITrasmissione
it.rsscollect.model
getNameTrasm():String
setNameTrasm(String):void
getNameEmitt():String
setNameEmitt(String):void
getUrlFeed():String
setUrlFeed(String):void
-listTrasmissione
0..*
-listTrasmissione
0..*
<<Java Class>>
CstmListViewAdptr
it.rsscollect.controller
context: Context
CstmListViewAdptr(Context,int,List<ITrasmissione>)
clickManagement(int,ViewHolder):void
<<Java Class>>
AbstractListViewAdptr
it.rsscollect.controller
TITLE_MAX_LENGHT: int
TAG_PUBDATE: String
TAG_ITEM: String
context: Context
listTrasmissione: List<ITrasmissione>
imageLoader: ImageLoader
options: DisplayImageOptions
resource: int
viewHolder: ViewHolder
AbstractListViewAdptr(Context,int,List<ITrasmissione>)
clickManagement(int,ViewHolder):void
getContext():Context
getListTrasm():List<ITrasmissione>
getImageLoader():ImageLoader
getImgOptions():DisplayImageOptions
getResource():int
truncLenght(String):String
getView(int,View,ViewGroup):View
<<Java Class>>
CstmListFavViewAdptr
it.rsscollect.controller
context: Context
imageLoader: ImageLoader
options: DisplayImageOptions
resource: int
viewHolder: ViewHolder
CstmListFavViewAdptr(Context,int,List<ITrasmissione>)
getView(int,View,ViewGroup):View
clickManagement(int,ViewHolder):void
<<Java Class>>
GetLogoDate
it.rsscollect.controller
imageEmitt: ImageView
txtDataFeed: TextView
GetLogoDate(ImageView,TextView)
doInBackground(String[]):Document
onPostExecute(Document):void
-getDataXML
0..1
-downloadXML
0..1
<<Java Interface>>
IGetDataXML
it.rsscollect.model
getNode(String,Element):String
getTag(Document,String):String
getUrlLogo(Document):String
extractTagForTransm(Document):List<ITrasmissioneXML>
isDescenDate(NodeList):boolean
truncDate(String):String
containsItem(Document):boolean
<<Java Interface>>
IDownloadXML
it.rsscollect.model
getXml(String):Document

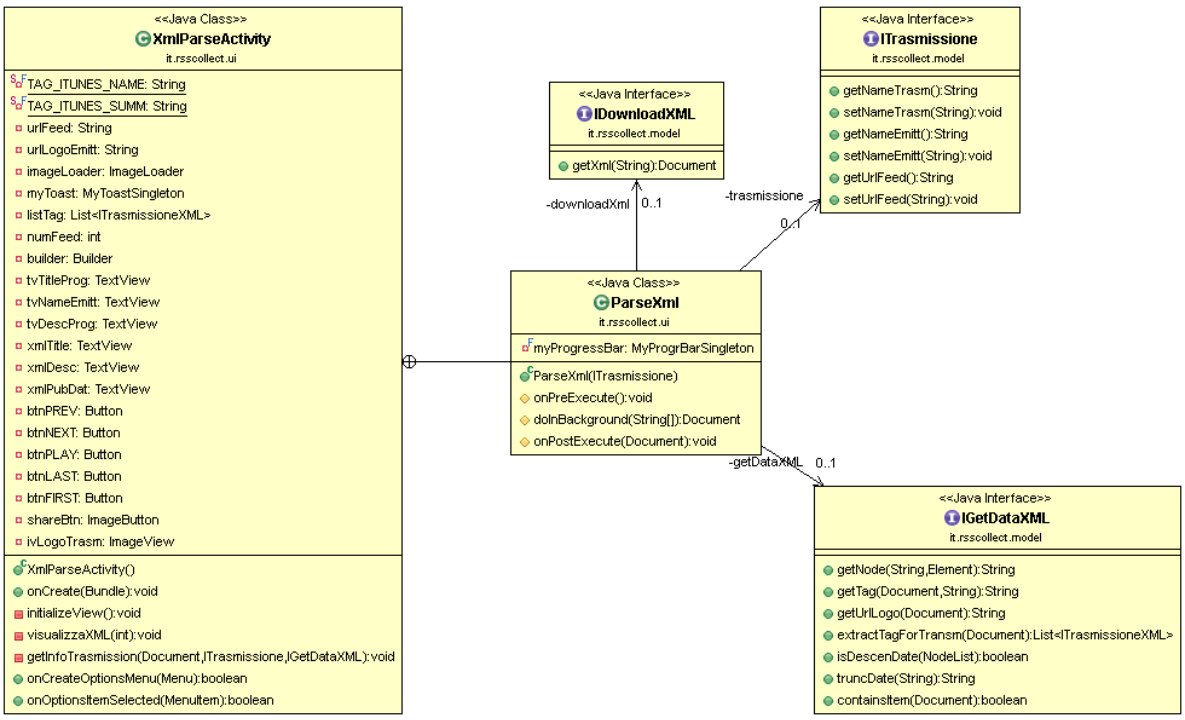
UML 6

Di seguito l'UML 7 mostra le relazioni tra le `ListActivity` e gli adapter:



UML 7

L' UML 8 descrive le relazioni tra la l'`activity` `XmlParseActivity` preposta a visualizzare gli episodi ed aprire la traccia mp3; `ParseXml` è la classe preposta ad estrarre i valori (tramite `IGetDataXML`) che devono essere settati nelle `Textview` e a prelevare l'indirizzo della traccia audio.



UML 8

Multithreading; il progetto a cause delle numerose operazioni in background (richieste/attesa risposte server, scaricamento file XML, caricamento immagini, parsing), utilizza classi che estendono `asynctask`: `GetDataFrmScrpt`, `SrchDataFrmScrpt`, `GetEmittFrmSrvr`, `GetTrasmFrmSrvr`, `SearchText`, `ParseXML`.

I pattern utilizzati sono:

- Singleton: utilizzato per la gestione dei toast (`MyToastSingleton`) e per le progress bar (`MyProgrBarSingleton`).
- Template Method: nella classe `AbstGetDataFrmScrpt`, `AbstractListViewAdptr`.
- Builder: utilizzato per la gestione del componente `AlertDialog` nella classe `XmlParseActivity` e `FavList` (per l'aggiunta/rimozione alla lista dei preferiti).
- Adapter: utilizzati per la rappresentazione dei dati nelle `ListActivity`.
- View Holder Pattern (pattern di Android usato nella `MainActivity`: lista emittenti).

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Durante la redazione del progetto ho appurato che la classe più critica e quella che con maggiore facilità generava errori è la classe `GetDataXML` la quale ha il compito di estrarre dai nodi del file XML, le informazioni di interesse.
Come già citato a causa delle diversità dell'organizzazione del contenuto, nonché della scelta di valorizzare alcuni nodi piuttosto che altri (scelta di chi mi mette a disposizione i contenuti), spesso si rischia di avere informazioni non valide (nodi privi di valorizzazione) o ancora peggio di generare `NullPointerException`.
Questo ha comportato la creazione di una classe `GetDataXmlTest` che esegue un test su alcuni metodi della classe; lo sviluppo ha dovuto tenere conto che i metodi della classe `GetDataXML` agiscono su un documento XML che non è disponibile localmente, ma che deve essere scaricato di volta in volta (per ogni test è passata la url).
Segue che la classe di test ha una classe innestata preposta allo scaricamento dei documenti (tramite un `Thread`): una volta che il documento è disponibile, sono testati i metodi.
Per quel che invece riguarda il test sul dispositivo è stato utilizzato l'emulatore incluso nel Android SDK con schermi da 4" pollici, 4.7" e 5", oltre ad alcuni dispositivi reali.

3.2 Divisione dei compiti e metodologia di lavoro

Per alcune ragioni di carattere personale ho deciso di affrontare questo progetto in maniera autonoma.
Il DVCS è stato utilizzato anche se gli aggiornamenti (`commit`) non sono avvenuti con quella frequenza che invece sarebbe stata opportuna nel caso di un progetto in team.
.

3.3 Note di sviluppo

Di seguito riporto le fonti esterne che ho reputato più significative:

- Libreria per il caricamento delle immagini nelle `listActivity`: https://github.com/nostra13/Android-Universal-Image-Loader
- Tecnica per il ritorno del risultato da un task asincrono: http://smartphonebysachin.blogspot.it/2012/11/how-to-return-value-from-async-task-in.html
- Parsing del file XML (in realtà è stato preso come spunto, ma poi modificato integralmente): http://www.androidbegin.com/tutorial/android-xml-parsing-dom-tutorial/
- Interrogazione di un DB Mysql da client tramite server php: http://www.androidworld.it/forum/sviluppo-e-programmazione-64/interrogazionedi-un-db-mysql-da-client-android-tramite-server-php-121001/
- ViewHolder Pattern: http://www.javacodegeeks.com/2013/09/android-viewholder-pattern-example.htm
- HTTP authentication con credenziali: http://hc.apache.org/httpcomponents-client-4.2.x/tutorial/html/authentication.html

Capitolo 4

Commenti finali

4.1 Conclusioni e lavori futuri

La naturale prosecuzione del progetto consiste nel utilizzare le API messe a disposizione da Itunes® per poter sganciare l'applicazione dal db in MySql e prelevare da qui tutte le informazioni necessarie, superando in questo modo il problema dell'aggiornamento dei palinsesti (una delle maggiori criticità dell'applicazione).
Anche la parte relativa alla sicurezza deve essere notevolmente migliorata: non è stata implementata alcuna cifratura dei dati sensibili (user e password di autenticazione) così come le connessioni non sfruttano i protocolli più affidabili in termini di sicurezza.

4.2 Difficoltà incontrate

Il progetto ha richiesto uno sforzo elevato per via delle varie tecnologie utilizzare (collegamento ad un server remoto, interrogazione tramite script php, parsing JSON dei dati in uscita, organizzazione dei dati tramite SQLlite, XML DOM parsing, multithreading, Android®); nel mio caso specifico il tutto si è reso più complicato per il fatto che per alcune tecnologie le mie conoscenze sono risultate molto superficiali, tuttavia è stata proprio la volontà di acquisire nuove competenze uno dei driver del progetto.
Questo ha comportato che complessivamente sia stato superato il monte ore previsto; anche per questa ragione l'interfaccia grafica dell'activity `XmlParseActivity` deve essere migliorata e resa più flessibile alla varietà dei dispositivi presenti.