AndroidTV Project Report
CMSC461

Matt Fioravante James Lee

May 11, 2008

Contents

List of Tables

List of Figures

1 Project Overview

AndroidTV provides the user with accurate, up to date television listings. The user can view listings using a tabular interface, or they can search for a particular program based on several different criteria. The user can mark their favorite programs and receive alerts when programs are about to be come on.

To update their TV listings the user can use the Schedules Direct service. AndroidTV will connect to the Schedules Direct servers and download updated TV listings. Whenever the listings become too old, the application will automaticly prompt the user to sync again.

2 Developer Contributions

Both developers put in an equal amount of work. The following is a list of the specific parts of the project each worked on.

Table 1: Station Schema

Stations	
channel_number	INTEGER
callsign	TEXT
name	TEXT
logo	BLOB

Shared Contributions

- ER diagram
- Database Schema
- General UI layout and design

James Lee's Contributions

- Schedules Direct Importing Code (network and xml parsing)
- Search Interface
- AndroidTV logo

Matt Fioravante's Contributions

- Listings Table
- Info Page

3 Database Design and Implementation

The ER diagram is given in Figure 1. The actual implementation of database deviated a little from the original diagram. Sqlite3 does not support foreign keys, so we just had to handle that behavior manually. For dates and times, there is an attribute type called DATETIME but it is basicly just an alias for TEXT. The attribute names are given along with their types and any other modifiers. The primary keys are underlined.

3.1 Stations

TV stations are represented by one relation. The schema is given in table 1. Data Direct supports multiple different lineups. One user could receive listings for Baltimore County and/or Howard County cable. To simplify things, we limited AndroidTV to only support one lineup. Because of this decision, it was simplest to make the channel number the primary key. The logo field was

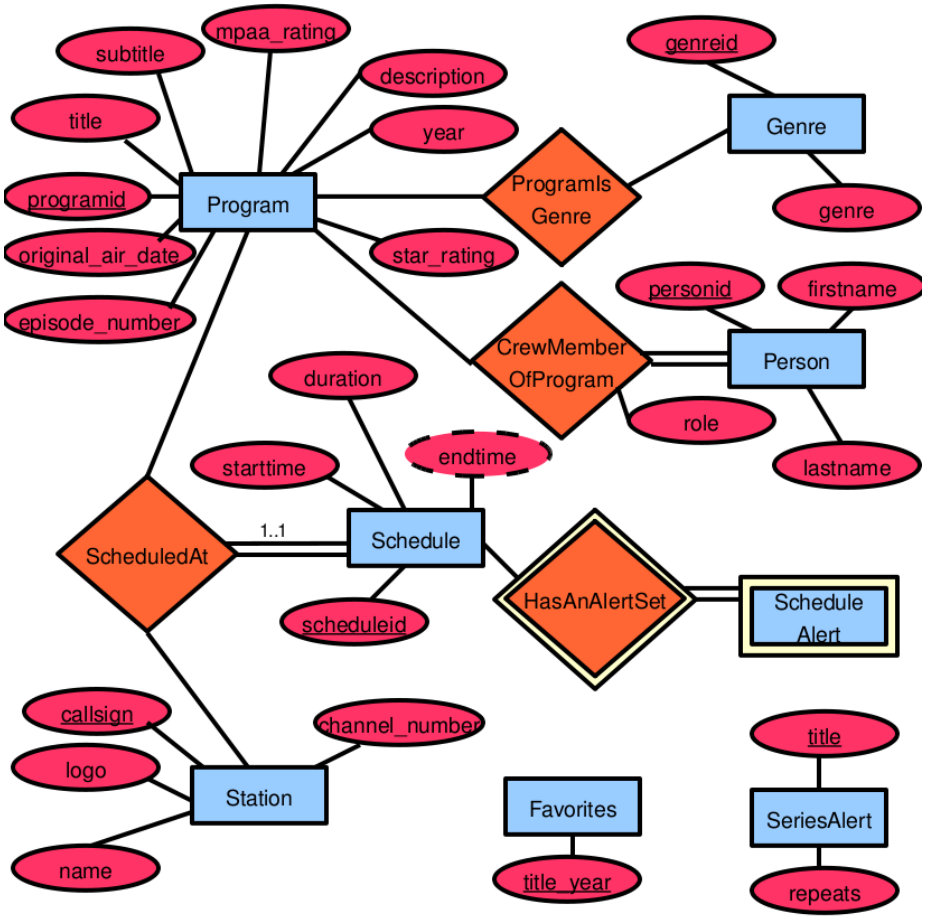

Figure 1: ER Diagram

Table 2: Program Schema

Program		
programid	INTEGER	autoincrement
title	TEXT	
subtitle	TEXT	
desc	TEXT	
mpaaratingid	INTEGER	REF(MpaaRating)
starratingid	INTEGER	REF(StarRating)
year	INTEGER	
showtypeid	INTEGER	REF(ShowType)
original_air_date	DATETIME	
episode_number	TEXT	

Table 3: Schedule Schema

Schedule		
scheduleid	INTEGER	autoincrement
channel_number	INTEGER	REF(Station)
programid	INTEGER	REF(Program)
starttime	DATETIME	
duration	INTEGER	
tvratingid	INTEGER	REF(TvRating)

meant to store a bitmap graphic of the station's logo. Currently, this feature unsupported since there aren't many easy-to-use repositories of logos.

3.2 Programs and Schedules

The schema's for programs and schedules are shown in tables 2 and 3, respectively. A program can be an episode of a show or a movie, or anything else that is shown on television. Programs are independent of the stations they appear on and the times they air. These are represented by the Schedule relation. The programid and scheduleid are arbitrary numbers that are generated using autoincrement. The other id fields will be described shortly. The rest of the fields are populated by Data Direct.

3.3 Ratings and other single value derived attributes

The schemas for ShowType, MpaaRating, StarRating, and TvRating are given in tables 4, 5, 6, and 7. All of these attributes are 1 of a specific set of strings. For example, tvrating can be "series", "miniseries", etc... To save space and reduce redundancy all of these attributes were placed in separate relations and referenced using automatically generated ids.

Table 4: ShowType Schema

ShowType		
showtypeid	INTEGER	autoincrement
showtype	TEXT	

Table 5: MpaaRating Schema

MpaaRating		
mpaaratingid	INTEGER	autoincrement
mpaarating	TEXT	

Table 6: StarRating Schema

StarRating		
starratingid	INTEGER	autoincrement
starrating	TEXT	

Table 7: TvRating Schema

TvRating		
tvratingid	INTEGER	autoincrement
tvrating	TEXT	

Table 8: Genre Schema

Genre		
genreid	INTEGER	autoincrement
genre	TEXT	

Table 9: ProgramIsGenre Schema

ProgramIsGenre		
programid	INTEGER	REF(Program)
genreid	INTEGER	REF(Genre)

3.4 Genres and other multi-value derived attributes

The schema for Genre, ProgramIsGenre, Advisory, and ProgramHasAdvisory are given in tables 8, 9, 10, and 11. These attributes are similar to the derived attributes from section 3.3. However, these attributes are multivalue. A program can have multiple genres. It can also have several advisories such as violence, strong language, nudity, etc...

3.5 Cast and Crew Members

Schedules Direct has a lot of data about cast and crew members of programs. The related schemas are given in tables 12, 13, and 14. People are uniquely identified by only their first and last names. If two people with the same name exist they will be considered the same person. AndroidTV currently imports all crew members. This is the largest amount of data in the entire database. This data takes the longest to import and it is currently not feasable to query crew members in a reasonable ammount of time. For future versions, we may consider importing only a subset of the crew members, such as only actors and directors.

3.6 Favorites

Table 15 shows the schema for the Favorites relation. Favorites are implemented in a somewhat non-intuitive way. A favorite is meant to be a movie, or a TV show or some series. Marking a favorite episode does not make much sense. The

Table 10: Advisory Schema

Advisory		
advisoryid	INTEGER	autoincrement
advisory	TEXT	

Table 11: ProgramHasAdvisory Schema

ProgramHasAdvisory		
programid	INTEGER	REF(Program)
advisoryid	INTEGER	REF(Advisory)

Table 12: Person Schema

Person		
personid	INTEGER	autoincrement
firstname	TEXT	
lastname	TEXT	

Table 13: Role Schema

Role		
roleid	INTEGER	autoincrement
role	TEXT	

Table 14: CrewMemberOfProgram Schema

CrewMemberOfProgram		
personid	INTEGER	REF(Person)
programid	INTEGER	REF(Program)
roleid	INTEGER	REF(Role)

Table 15: Favorites Schema

Favorites		
title	TEXT	REF(Program)
year	INTEGER	REF(Program)

Table 16: ScheduleAlert Schema

ScheduleAlert		
scheduleid	INTEGER	REF(Schedule)

Table 17: SeriesAlert Schema

SeriesAlert		
title	TEXT	REF(Program)
repeats	INTEGER	

easiest way to reference at this level is to reference the title. TV shows generally leave the year field as null. For movies on the otherhand, remakes had to be accounted for, so the year field is also used to uniquely identify a favorite.

3.7 Alerts

The relations used to implement the alert system are shown in tables 16 and 17. A schedule alert is an alert that occurs when just one program at one time on one station is about to start. A series alert occurs everytime a particular series is about to start on any station. For series alerts, the option to ignore repeats is also available. These features have yet to be implemented in AndroidTV. It is likely that when they are implemented, the schema may be changed.

4 Application Design and Implementation

4.1 Database and Application Interfacing

Since the database was going to be used by all parts of our progam, and there is only one database, we decided to make our SQLiteDatabase a singleton object and expose some querying functions through a static class. This way, we did not have to worry about passing our database object between classes. Our static database adapter also handled database creation, opening, and closing.

If the database failed to open, our class would assume the database does not exist and create the database. We stored our database's schema as a text file asset which we could open using Android, then read statements using the Scanner class. Program data such as ratings and genres are stored as Enumerations so that we could access them from our program quickly without querying the database, but they are also inserted into the database after it gets created.

4.2 UI Layout and Design

The main menu pictured in figure 2 was designed to be usable on any screen size (by using a ScrollView) with large buttons to be easy to use with fingers. The

Figure 2: Screenshot of Main Menu

TV Listings button launches the Listings activity (see section 4.3), the Favorites button launches a search of the database (see section 4.6), the Search button launches the Search activity (see section 4.5), and the Settings button launches a dialog where you can enter your Schedules Direct username and password (see section 4.7).

4.3 TV Listings Table Screen

The Listings Table allows the user to scroll vertically by channel number and horizontally by time to view TV Listings. A screenshot is given in figure 3. The user can click on any program in the table and they will be taken to the Info screen to see more details about the particular program (see section 4.4).

Currently, the table is incomplete. Vertical scrolling does not work. The table takes a long time to load because the database has to query all of the schedule and program information. Currently, only a limited number of stations' schedules are loaded into the table to avoid running out of memory. Later on, the table needs to be redone so that it only reads in a portion of the data, and reads in more in small ammounts as the user scrolls left and right.

Getting the table elements to align properly was also a challenge. Originally a TableLayout was used. The idea was to have the listing textboxes have fixed width and span multiple columns if needed, but this turned out not to work. Instead, an approach using a vertical linear layout that contained horizontal linear layouts was used and turned out to be effective. Another limitation was that the textboxes that contain each of the listings had to be set to one line only even though they can hold two lines. The automatic word wrapping would ruin the alignment of the entire table otherwise.

4.4 Program Information Screen

The Info Screen gives detailed information about a particular program. A screenshot is given in figure 4. Using the Menu button, the user can toggle whether this program is a favorite. They can also set an alert. This screen is reached by either using the listings table or performing a search.

Currently, the Crew Members section is not implemented. There is too much crew data it is not feasable to query it all. Please see section 3.5 for details.

4.5 Search

The search activity (pictured in figure 5) provides a generic interface to enter data. This activity doesn't actually do the searching; it passes its input to another class.

The fields are populated by an Enumeration called SearchField, which defines a human readable description, the field in the database to search, and what values the user may choose from. Here are a few of the SearchField enumerations used by the Activity to populate the form:

```
TITLE("Title", "upper(title)", new SearchRange[]{SearchRange.CONTAINING,
        SearchRange.EXACTLY}, QueryType.EDIT_STRING),
```

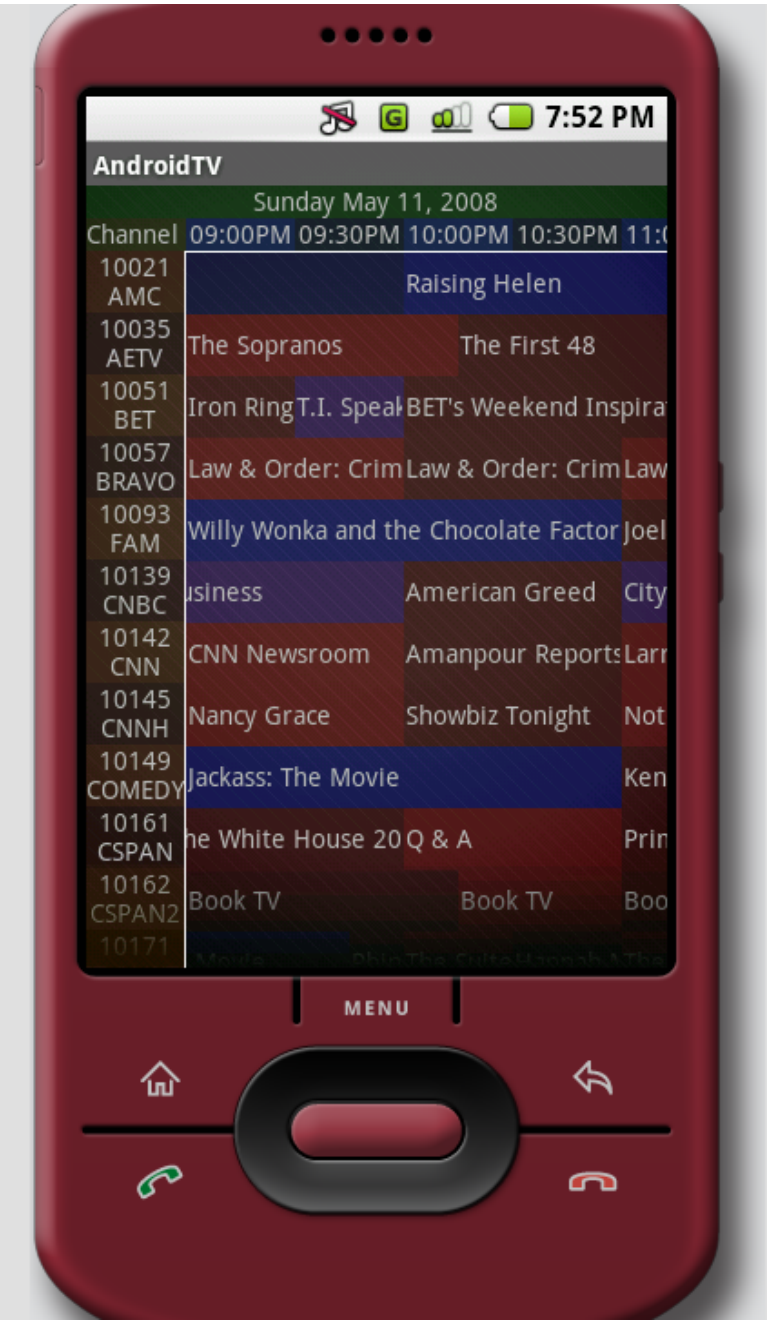

Figure 3: Screenshot of TV Listings Table

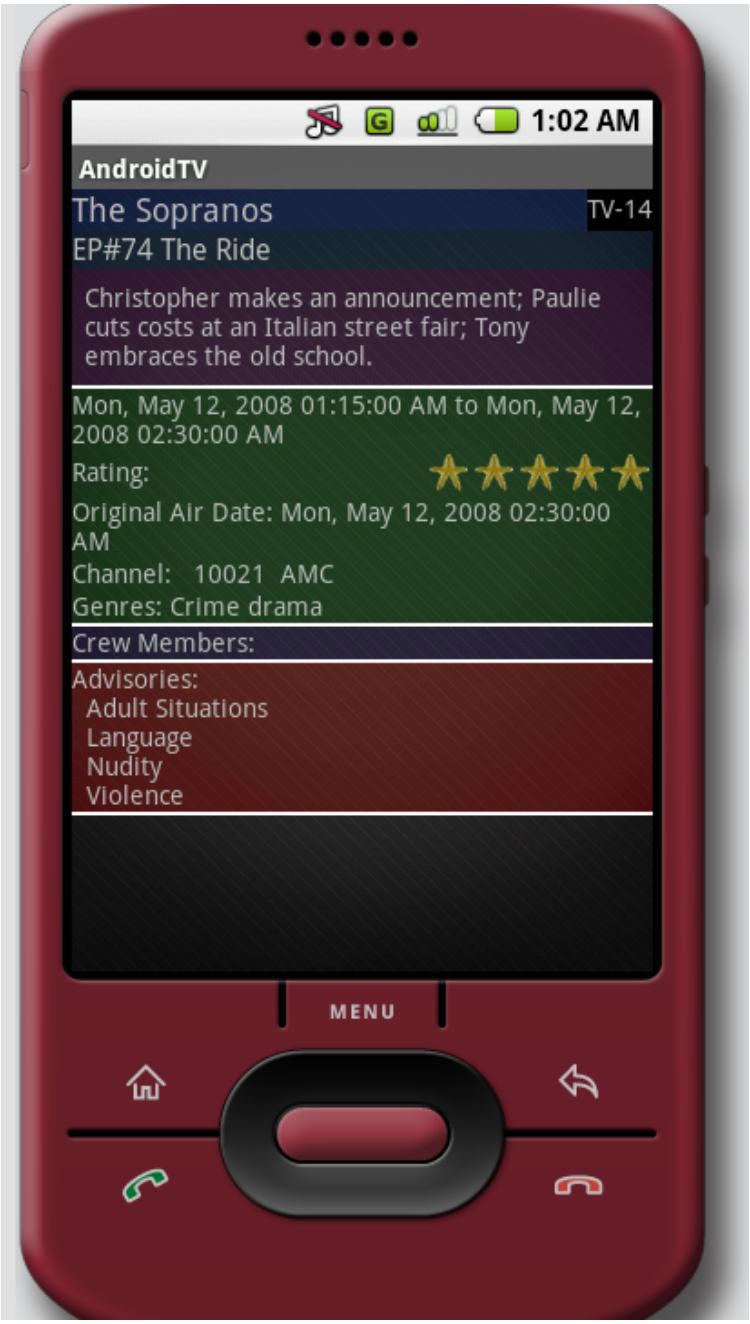

Figure 4: Screenshot of the Program Information Screen

Figure 5: Screenshot of the Search Screen

```
TV_RATING("TV_Rating", "tvratingid", new SearchRange[]{SearchRange.EXACTLY,
        SearchRange.AT_MOST, SearchRange.AT_LEAST}, QueryType.CHOOSE_STRING,
        ProgramData.TV_RATINGS),
SHOW_TYPE("Show_Type", "showtypeid", new SearchRange[]{SearchRange.EXACTLY},
        QueryType.CHOOSE_STRING, ProgramData.SHOW_TYPES);
```

Here we would say: define a search form that allows a user to select from searching "Title", "TV Rating", or "Show Type". If they pick "Title", "upper(title)" (in the database) may either contain or exactly match a string that they enter. Likewise for the rest of the fields. Finally the search activity has a function to generate a SQL query from the user's choices. This query gets passed to the Searcher.

Searcher will display progress dialog and query the database with a given statement. If it returns no results, it will report the failure to the user in an alert dialog. If it returns more than 200 results, it will truncate the results and report that to the user. Next it passes the results to the search results activity.

The search results activity takes in an array of scheduleids which it then displays in a ListView as pictured in figure 6. For every scheduleid, additional information such as the show's title, subtitle, and airing times, are queried from the database. Since this takes several seconds for 200 shows, a progress bar is displayed in the title bar to inform the user of status.

Finally, when a show is clicked in the ListView, an info activity is launched as described in section 4.4.

4.6 Favorites

Favorites is just a specialization of searching. Since our Searcher class can take an arbitrary SQL statement and display its results, we can pass it a query that finds all upcoming instances of programs with titles contained in the Favorites table. In code, this looks like:

```
new Searcher(AndroidTV.this).search("SELECT_scheduleid_FROM_" +
        "Schedule,Program,Favorites_WHERE_" +
        "Schedule.programid=Program.programid_AND_" +
        "Program.title=Favorites.title_" +
        "LIMIT_" + (Searcher.MAX_RESULTS + 1), null);
```

4.7 Settings

The settings screen pictured in figure 7 is a simple username and password form for Schedules Direct. The values in this form get saved to the activity's global preferences with keys "username" and "password", respectively.

4.8 Schedules Direct Import

Note: The Schedules Direct protocols are defined at `http://tmsdatadirect.com/docs/tv/`.

All of the schedule information used by our program comes from Schedules Direct. This information can be received by quering their SOAP service. They

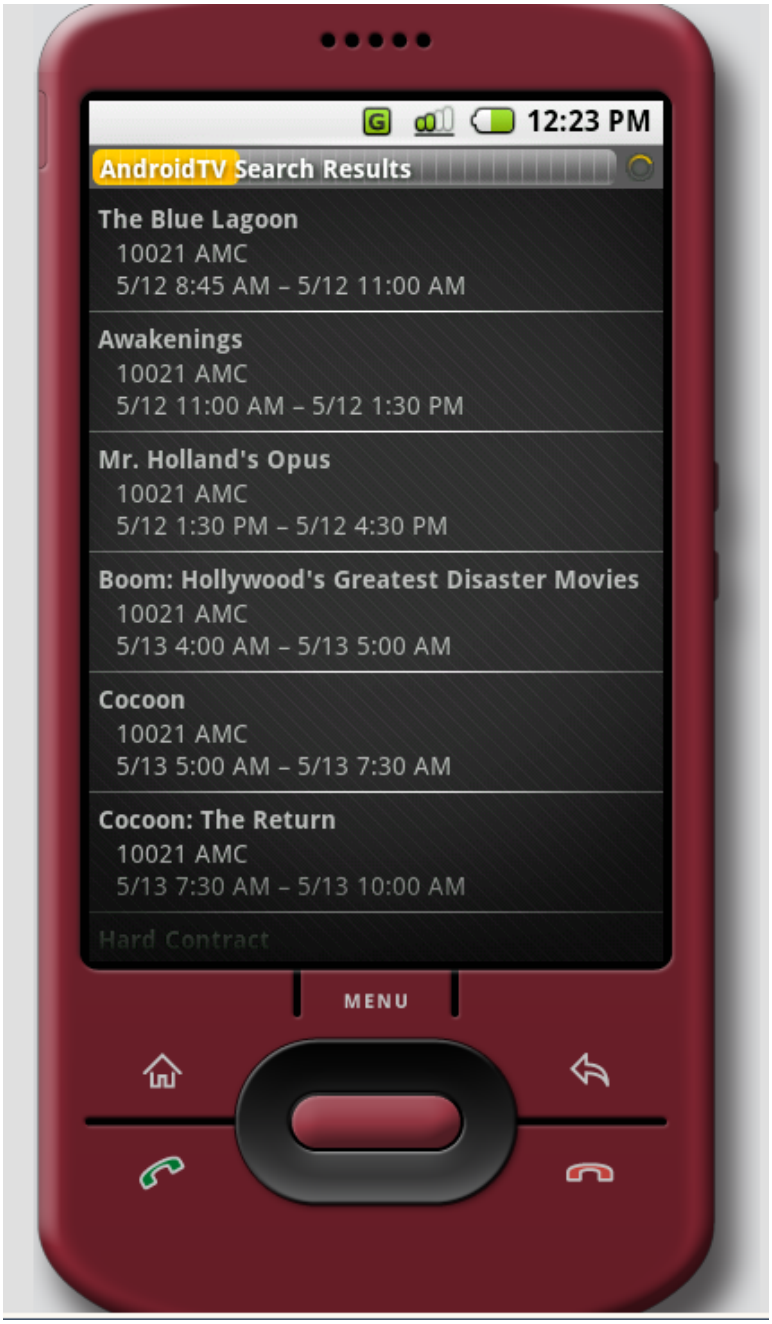

Figure 6: Screenshot of the Search Results

Figure 7: Screenshot of the Settings Screen

Figure 8: Screenshot of the Schedules Direct Import Process

protect their service using HTTP digest authentication, which is poorly supported in Java. Fortunately, Android bundles Apache's HTTP client libraries which do support digest authentication. During an import, if the server reports bad credentials, or if no credentials are stored in the global preferences, a login dialog is presented to the user (see section 4.7). Next, a SOAP envelop is sent to the server containing request information such as the amount of schedule information to receive. Due to massive amount of data contained in a day's worth of schedules, we limit this to two days. Finally, the service sends a well-defined XML response containing schedule information for the given period of time.

In Java there are two major XML parsing paradigms: DOM and SAX. DOM parsers must read in a whole document to build a model of it in memory. Then you can query or change it as you wish. SAX parsers respond to events *as* it parses the document, but can't move around it randomly. Since the Schedules Direct response is huge, we cannot store it in memory, so DOM is not an option. But SAX is a better fit, anyway; now we can insert information into the database as we parse it. Even without any processing code, parsing two days of information still takes about three minutes since the response is usually over 200,000 lines.

First, rather naïvely, data was inserted into the database as the parser received it. For example, when a subtitle was parsed, it would update the relevant program. This made sense at first since a lot of progam data is optional. However, with import times taking over twenty minutes for one day's worth of schedules, improvments had to be made. First, and most obviously, is to reduce the number of database insertions and updates. This was done by storing more data until the last possible moment, then inserting. For example, when we read a subtitle, store it in memory, then once we know we've read all of a program's data, insert whatever it has read. This, effectively, halved our import times at the expense of memory. With only 16 MB of heap space per process on Android, reading some data, such as the crew information, nearly fills memory.

Secondly, all insertions and updates were changed to use pre-compiled SQL statments. Now whenever you insert something, you just bind different data to a statement's varibles and execute it. Since the queries themselves don't ever change, we only have to compile our statements once. This also just about halved our import times.

With the improved import times, we are able to import two days of schedules.

5 Conclusions

This project was a thorough introduction to databases including design, XML, and SQL. The requirement of Android gave us many challenges which we would not have faced on traditional hardware, which is a good thing. We would not have otherwise considered time and space issues. It would be interesting to see if these issues still exist on actual hardware.