University POLITEHNICA of Bucharest
Automatic Control and Computers Faculty,
Computer Science and Engineering Department

BACHELOR THESIS

Ambient Intelligence for Task Integration on Android

Scientific Adviser:
S.I. Dr. Andrei Olaru

Author:
Drăgoiu-Prună Cătălin

I would like to thank my project supervisor,
Andrei Olaru, for the tremendous help and constant
guidance offered throughout the project
that allowed me to design and create this work.

Abstract

In a world where we need to remember, process and share more information than ever before, AmIciTy is an application that hopes to make day to day activities easier by offering a simple and easy way to organize them. The project tries to achieve this goal by using the powerful Android platform and the hardware currently available in smartphones and tablets. This document details the functionality and architecture of the application. It also includes specific implementation details, a description of the Android API used, as well as future plans for expanding the project.

KEYWORDS: Android API, Separation of Concerns, Camera, Documents, Data Sharing, Contacts, Notifications, open-source, Custom UI

Contents

List of Figures

Notations and Abbreviations

API - Application Programming Interface
UI - User Interface
URI - Unified Resource Identifier
JSON - JavaScript Object Notation
UUID - Universally Unique Identifier

Chapter 1

1. Introduction

1.1 Motivation

For people in the 21st century, completing personal and professional tasks in a timely manner is one of the most important things for a stress free and successful life. Many of these activities require interactions with the environment and with other people, and it is easy to see how this process can be improved using modern technology.

Using the AmIciTy application the user should be able to connect the description of the task with a visual source of information, with different files stored on his devices and with other people involved in the activity, ensuring quick access to all these resources. Assigning priorities and setting reminders at certain points in time are also features highly sought after.

Another aspect of the AmIciTy application is that all information related to a task can be easily shared with other people using Email or Bluetooth file transfer. This allows improved collaboration and reduces the effort of searching, grouping and sending the data to a single touch of a button.

In recent years, smart mobile devices such as phones and tablets have become more widespread than personal computers, offering the advantages of reduced size, long battery life and increased connectivity. This portability satisfies the needs of a task organizer because the device is close to the user at all times and thus, the task data is always just a few taps away.

The mobile devices market is dominated by three operating systems, Google's Android, Apple's IOS and Microsoft's Windows Phone. Android was chosen as the development platform because it is more suitable for free, open source applications by offering a reduced deployment price and a simpler development process.

AmIciTy benefits from other advances in mobile devices like increased storage size, that allows large number of files to be stored in the persistent memory, quality camera hardware, and increased wireless internet speeds. Advances in cloud storage now allow sharing tasks that contain multiple resources by offering higher limits for the size of email attachments and high speed Bluetooth allows almost instantaneous transfer of tasks between paired devices.

Another important goal for the application is the research of various Android APIs and the creation of a documentation describing their core concepts and usage. As the project is open source, working usage examples can be obtained from the projects Git repository [4].

1.2 Features

A list of features offered by the **AmIciTy** Android application is listed below:

- Expose an interface for the creation and deletion of tasks.
- Take a photo with the camera and add the photo to a task.
- Add a file from the device external or internal storage to the task.
- Add a contact stored in the device or in the SIM card to the task.
- Add a notification at a future point of time to serve as a reminder.
- From the application, files and photos will be opened by clicking on them using the application that supports their format.
- When clicking on a contact, a page offering details to call, send a message or an email will appear.
- Allow the user to share a task using email or Bluetooth.
- Sharing a task will also share all resources attached to it.
- A received task can be imported in the AmIciTy application if it is installed or it can be used

These features are offered to the user in an intuitive and easy to use way that is consistent with the design recommendations for the latest version of the Android Operating system [7]. For example, task assets are displayed using a visual representation when available, such as a photo thumbnail or a contacts profile picture, in agreement with the "**Pictures are faster than words**" [1] and "**Real objects are more fun than buttons and menus**" [1] principles. The "**Keep it brief**" [1] recommendation is respected as the application has very little text, relying on icons to trigger actions and "**Give me tricks that work everywhere**" [1] is implemented in the task list's swipe to dismiss functionality.

AmIciTy is a long term project that is designed to make the development of new features fast and easy by using a modular design, with expandable components such as the sharing module or the task object

[1] http://developer.android.com/design/get-started/principles.html

itself. Adding new modules is also made easy by Androids design architecture that provides a clear separation between the user interface component and application logic.

A detailed description of each feature, as well as screenshots and usage guides are presented in chapter 4.

Chapter 2

2. State of the Art

2.1 Presentation of top task organizer applications

The request for task organizer applications is high, and several advanced solutions already exist. In chapter 2 we try to offer a brief description of the top applications in this field, emphasizing their strengths and weaknesses.

2.1.1 Any.DO

Any.DO offers a simple and intuitive user interface to create and dismiss tasks. In this application, a task can contain a title and several notes, it can schedule reminders and share a task using the Any.DO network or using Android Beam. Other features provide search engine results for the title of the task, and integration with a Calendar using the dedicated Any.DO calendar application.

The strengths of this application are the ease of use, the clean looking user interface and the synchronization with the server side component.

2.1.2 Google Keep

Google Keep is another very popular lightweight task organizer that ships with the Android operating system. For this application, a task is either a title or a photo captured using the camera. It only supports adding reminders to tasks and does not offer any sharing capabilities.

The main strengths of the application are the multi column view, where each tile represents a task. The view supports columns of different dimensions, leaving the impression of a very efficient use of the screen.

2.1.3 Todoist

Todoist aims to be a more complete solution to the task organizer problem, offering the possibility to set subtasks, due dates and priorities to a task. Files, images and contacts cannot be linked to a task and tasks cannot be shared with other people.

Support for Google accounts and applications available on most modern platforms and operating systems represent Todoist main strengths, as well as the task tree design.

2.1.4 Task Radar

The Task Radar application distinguishes itself by presenting the task list as a radar display. The tasks are represented as points that move closer to the center of the display as their deadline approaches.

Task can also be assigned priorities and tags such as work or home. Notifications are created when the task approaches the deadline.

The main strengths of the application consist of the custom, radar display like user interface and the ability to add tags to tasks.

2.2 AmIciTy exclusive features

In addition to the basic functionality offered by the presented applications, AmIciTy aims to provide a much more complex task structure that can link to multiple elements such as files, images and contacts. Sharing a task also shares all linked information to that task, another unique feature for AmIciTy .

The Android ecosystem is made of a very large number of apps that offer specific functionality. Using the Intent architecture, AmIciTy aims to provide a high level of integration with other apps on the users device, such as the gallery application for viewing images, the pdf and document reader, the various email applications and the Google drive storage system.

Chapter 3

3. Application Architecture

3.1 Android Activity Flow

Activities are the main component of an Android application. An activity usually composes of a user interface that offers the user information and handles user events like tap and swipe. The user interface of an activity is composed of one or more fragments, that act like reusable UI code modules that can be displayed according to characteristics of the device like screen size and resolution.

The Android intent represents an abstract description of an operation and is used to start android activities. Intents can contain data and can be used to start activities using a known activity class, or using the android intent filters to set an activity as a handle for a specific intent. AmIciTy can be started using both these mechanisms.

Designing an Android application must start with defining its Activity classes. The model of these classes is a direct result of the expected features and user interface, with links between Activities representing the logic flow of the application.

The following class diagram illustrates two AmIciTy activities and their most important methods.

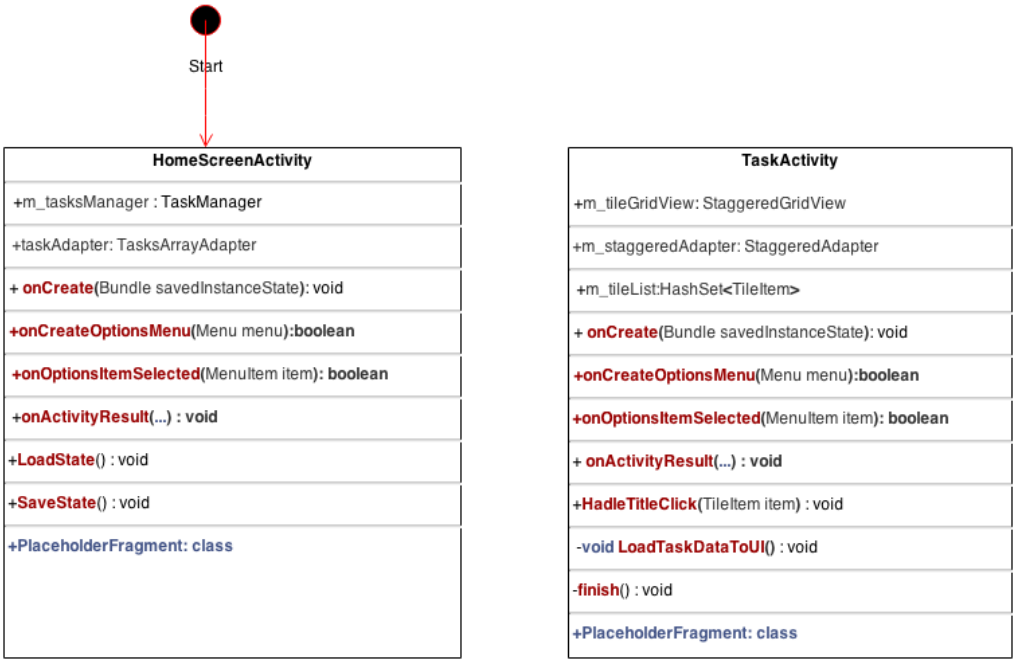

Figure 3.1 - Activity class diagrams

In the following paragraphs you will find a description each AmIciTy activity.

3.1.1 Home Screen Activity

The Home Screen Activity is set as the application launch handler for AmIciTy. Its main responsibilities are:

- Checks if the application has a previous state stored on the device.
- Loads the previous state
- Presents the task in a list layout.
- Launches the task edit activities that handle individual tasks.
- Syncs the application state with the stored state on the device.
- Allows the deletion of tasks.

The Home Screen Activity contains a Task Manager objects that offers a wrapper offer a task list data structure.

Data persistence is achieved by saving all tasks to a file in JSON format. The JSON is modern, easy to parse format that can contain complex structures like objects and arrays. To get a JSON representation of the TaskManager object the GSON library is used. GSON is an open source library developed by Google that can transform any Java object to a JSON representation, a process we will refer to as serialization, and recreate the object from an existing JSON representation, process known as deserialization.

The internal PlaceHolderFragment class is used to load the layout of the activity. The layout consists of a listview of task layouts that display minimal information about all tasks.

3.1.2 Task Activity

The Task Activity is used for creating, editing and displaying a task. Its responsibilities are:

- Attach a file to the current task
- Attach an image to the task
- Link a contact with the current task
- Add a notification to the current task
- Share the current task with other users

Using intents, the task activity must launch the appropriate application to handle the attached assets. For example, an attached file is opened using the application that registers itself as a handler for its specific extension and MIME type and a contact is opened using the Android Contact application that offers specific functionality such as calls, messages or emails.

There are two ways the user can interact with the task activity. The first one is made on of action bar buttons that can be used to add assets to the current task. The action bar menu is defined in the action_bar_menu.xml file and consists of icons from the Android Design Icons pack provided by Google.

The second way of interacting is specific to assets already linked to the task. These are displayed in a grid view in the form of tiles. The grid view is custom made to support variable height columns. Tapping on a tile triggers an action appropriate for the asset type such as the gallery or the contacts application. More information on the custom grid view can be found in section 4.3.2 from the custom UI elements chapter.

The activity uses the TaskShareManager class to share tasks with other users. More details about this class can be found in section 3.2.2.

3.2 Logical Modules

AmIciTy aims to follow the separation of concerns design practice as much as possible and uses modules to encapsulate different functionalities. A module can contain one or more classes and exposes a public API for other modules. We will now look into some of the AmIciTy modules.

3.2.1 The Task class

The task class contains a Universally Unique Identifier (UUID) and containers for different types of assets. The UUID identifies the task and is used to prevent cases when a task is imported several times into the application. Two tasks are considered equal if they have the same UUID. This check is done in the overridden equals method. Assets are added to a task from many places in the application code such as action result callbacks and anonymous listeners, thus access to the assets is done via getters and setters to provide an easy way of debugging misuses of the class.

The main components of the Task class are presented in the following diagram.

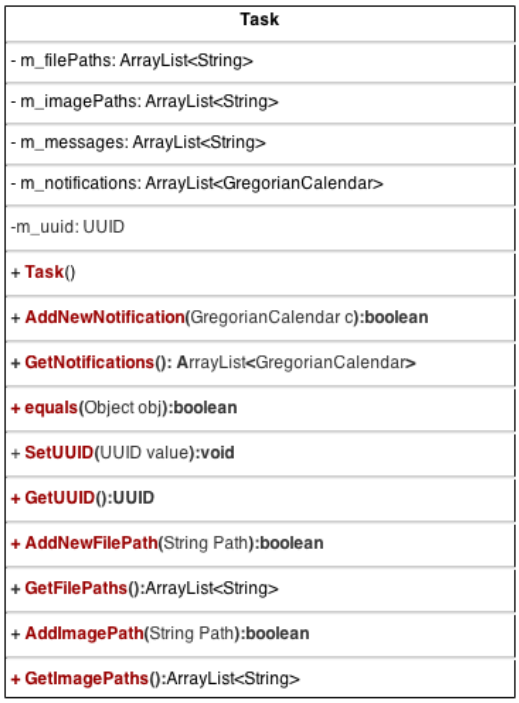

Fig 3.2 - Task class diagram

3.2.2 The Task Manager class

The Task Manager class is a basic wrapper of an ArrayList of Tasks intended to offer some additional functionality besides the one from the container, as well as to provide an easy point for debugging task list related problems.

3.2.3 The TaskShareManager class

All processes involved in sharing tasks are contained in the TaskShareManager, such as creating and exporting the task file, launching sharing applications and import external tasks received by the application .

Chapter 4

4. Implementation Details

4.1 Access to resources

As the core of the application is represented by its high level of integration with the resources offered by the Android platform, a significant part of the application is dedicated to handling various Android APIs. In this chapter we will explore each of these APIs, presenting the way they are used in AmIciTy.

4.1.1 File API

The Android Operating System does not expose a standard file picker. In response to this problem, the FilePicker class was created. The class allows the user to browse through the file system and select a file that is added to the task.

The java.IO.File class is used to obtain the contents of a directory, that is displayed in an android listview. When the user taps on the name of the folder, the file picker will load the content of that folder. If the user selects a file, the path of the file is returned in the data intent that is passed to the OnActivityResult method of the TaskActivity class. The file picker is launched from the TaskActivity class showing the top level of the external storage directory. The diagram below shows the File Picker logic flow.

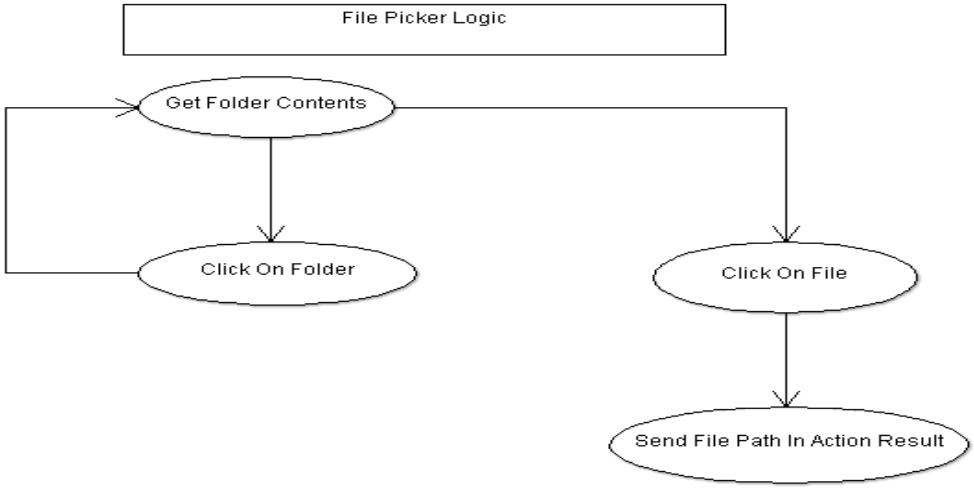

Figure 4.1.1 - File Picker Logic Flow

After the file is selected its absolute path is stored in the task object and it is displayed to the user in a tile from the grid layout. When the user clicks on the tile a new ACTION_VIEW intent is created and launched. The intent contains the file path and its MIME type, data which the Android system uses to find application that can display the file. The user is presented with a dialog where all applications that handle the file are displayed, if a default one is not already selected for the file type.

4.1.2 Android camera API

The camera activity can be started using an ACTION_IMAGE_CAPTURE intent. It enables the user to take a picture, review it and select it for inclusion in the task object. It also allows the user to discard the picture and try again if he deems it necessary.

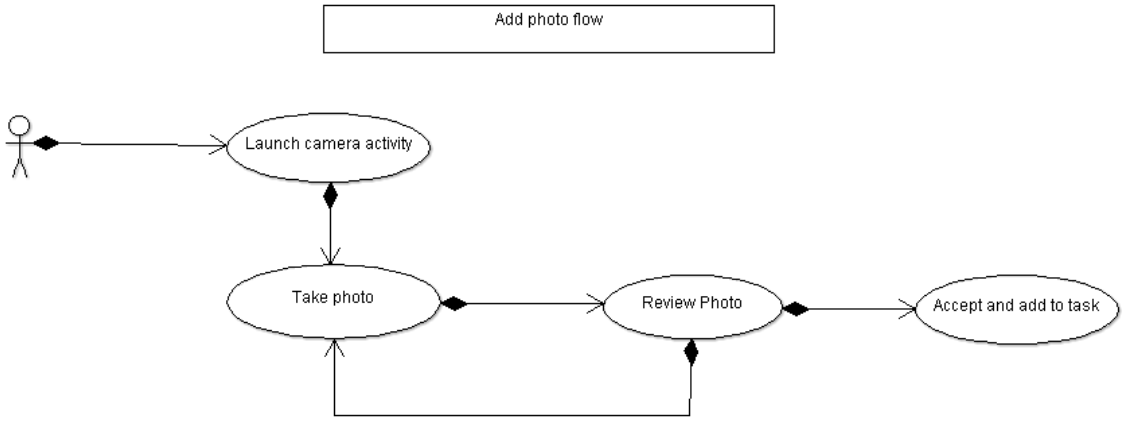

Figure 4.1.2 -Add photo to task flow

The photo can be obtained from the camera activity as a low resolution thumbnail contained in the data returned in the activity result. Obtaining the full image requires passing a file as the parameter EXTRA_OUTPUT to the ACTION_IMAGE_CAPTURE intent. The file is created before the intent is launched and is included as a Uniform Resource Identifier (URI). When the photo is accepted by the user, the image data is stored in the provided file and can be accessed after the camera activity is closed, in the OnResult method.

The received photo is displayed in the grid view of the task activity. Opening the photo is done in the same way files are opened, using the ACTION_VIEW intent. The gallery application is usually the default android image viewer.

4.1.3 Android contacts API

The ContactsContract class defines a contract between the contract provider and other applications that require contacts information. The contract data is stored in an extensible database and is structured in a three tiered data model[2].

- ContactContract.Data is a table that can store any kind of contact data, such as the phone number or email address. Applications can add their own data types for a contact, in addition to the standard ones shared by all contacts.

- ContactContract.RawContacts contains the data for a contact associated with a single user account.

- ContactContract.Contacts table represents an aggregate of the ContactContract.Contact entries describing a person. The data is synced with each individual RawContacts entry when a change occurs.

In AmIciTy, after the add contact button is tapped, a new intent is launched with the action parameter ACTION_PICK and ContactContract.Contacts.CONTENT_URI unified resource identifier. This launches a contact picker showing all the phones contacts.

When a contact is chosen in the picker, an URI pointing to the contact is returned in the data field. Using this URI, a cursor to the contact entry in the database table can be obtained with a ContentResolver query. The cursor enables the retrieval of the contact ID, a unique identifier that can be used in queries to retrieve any information about the contact. The contact ID is stored in the task object as an integer.

The contact is presented in the grid layout as a tile containing the profile picture, if this feature is available for a contact or as a default contact image. A label is also added that contains the contact name.

The following diagram illustrates the steps in the contact management flow in the AmIciTy application, from launching the contact picker to getting contact information like profile picture, phone number or email address.

[2] http://developer.android.com/reference/android/provider/ContactsContract.html

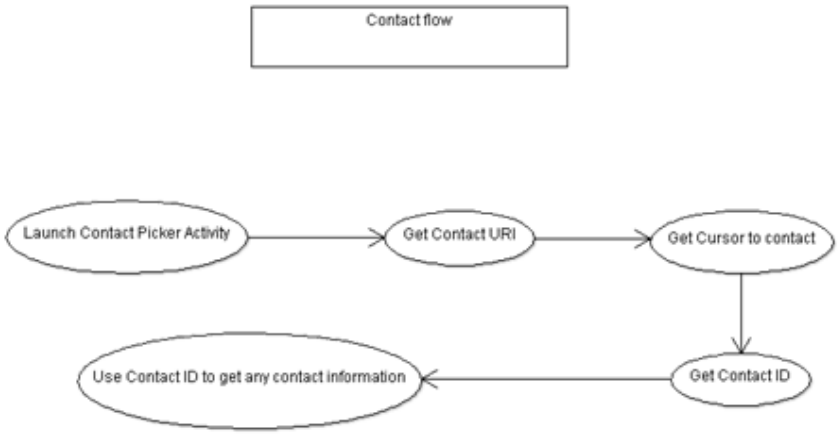

Figure 4.1.3 - Add Contact Logic Flow

To further explain the ContactsContract API we will explain the process of acquiring the contact name and contact photo using the contact ID stored in the task objects.

For the contact name, we first obtain an URI for the contact using the ContentUris class.

```
Uri contactUri = ContentUris.withAppendedId(ContactsContract.Contacts.CONTENT_URI,
                                            contactId);
```

Using the URI we acquire a cursor with a query containing the URI:

```
ContentResolver cr = getContentResolver();
Cursor cursor = cr.query(contactUri, null, null, null, null);
```

A cursor can, in theory, contain multiple entries. However, we made a query with a unique contact ID so we will only receive one entry. To get this entry, the cursor must be set to the first entry of the returned set.
We then get the name column index and use it to retrieve the name as a string.

```
if (cursor == null) {
             return "";
        }
        if (cursor.moveToFirst()) {
               contactName = cursor.getString (cursor.getColumnIndex
                              (ContactsContract.Contacts.DISPLAY_NAME));
        }

return contactName;
```

To retrieve the profile photo we reduce the logic by using the openContactPhotoInputStream method of the ContactsContract.Contacts class. Using the contact URI obtained as in the previous example we call:

```
InputStream photo = ContactsContract.Contacts.openContactPhotoInputStream
                        (getContentResolver(), uri, true);
```

If the contact has no photo added to his profile, the received input stream will have a null value. We must check for this case and, if a photo is available, return it as a drawable asset. In Android, the drawable object represent abstraction over an underlying visual resource.

```
if(photo == null)
       return null;
return Drawable.createFromStream(photo, null);
```

The drawable is then displayed in a tile from the grid view. A click on the contact tile will launch a ACTION_VIEW intent that will be handled by the contacts activity. This activity provides detailed contact options, such as call or email.

4.1.4. Alarm Services

The systems alarm services allow alarm scheduling at future point of time. This functionality can be accessed using the AlarmManager class. The class can be acquired from the application context in the following way:

```
AlarmManager alarmManager = (AlarmManager) getApplicationContext().
                                    getSystemService(Context.ALARM_SERVICE);
```

The AlarmManager allows us to set an intent that will be called at a given point of time. The intent must be contained in a PendingIntent object, which contains the intent and the description of the target action that will be performed with it.

The pending intent allows another application to perform the operation specified with the right normally reserved to your application, thus it should be used with care as it may expose security holes.

To set an alarm using the AlarmManager class we use the set method, as demonstrated in the following example:

```
Intent intent = new Intent(this, AmIciTyAlarmReceiver.class);
intent.putExtra("description", text);
PendingIntent pendingIntent = PendingIntent.getBroadcast(getApplicationContext(), id,
                                intent, PendingIntent.FLAG_UPDATE_CURRENT);

alarmManager.set(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), pendingIntent);
```

The RTC_WAKEUP value instructs the system to wake up the device when the alarm is triggered, if it is sleep mode.

The intent contains the AmIciTyAlarmReceiver, an extension of Androids BroadcastReceiver class. This class contains an OnReceive method that is called when the broadcast occurs. The BroadcastReceiver class exists only while the instruction pointer resides in the OnReceive method. When this method returns, the object is considered unused and is marked for destruction.

Security is important when working with broadcasts, because any other application can intercept the message. However, in this case, only a text description of a task is contained in the broadcast intent, which cannot be considered sensitive or private information.

4.1.5 Notification API

An android notification is a message that can be displayed to the user outside the applications user interface. It is displayed in the notification area as an icon and, with more in the notification drawer. The notification drawer is controlled by the operating system and can be accessed at any time by swiping down from the top of the screen.

In AmIciTy, notifications are launched from OnReceive method of the AlarmReceiver. A notification is created using the Notification.Builder class, that allows us to set parameters such as icon, text, and sound.

```
Notification notification = new Notification.Builder(context)
.setContentTitle("You need to take a look at this task")
.setContentText(notificationText)
.setSmallIcon(R.drawable.ic_action_time)
.setContentIntent(pendingIntent)
.setDefaults(Notification.DEFAULT_SOUND | Notification.DEFAULT_LIGHTS |
Notification.DEFAULT_VIBRATE)
 .getNotification();
```

After we obtain the notification objects, we can launch it using the NotificationManager, assigning an integer ID that can be used to identify the notification in future calls. This action shows the notification in the systems notification area.

```
notificationManager.notify(1, notification);
```

When the user taps on the notification, the applications HomeActivity is started and the task list is loaded and presented in the user interface.

4.1.6 Email API

To send an email from the AmiCity application we launch an ACTION_SEND intent that contains information about the emails subjects, text content and attached file. The attachment represents the compressed ".amy" file that contains all task assets and the task description. A detailed description of the creation of this file can be found in section 4.2.1.

Launching the intent will present the user with a dialog that allows him to choose the preferred email client to handle the message. After a client is chosen, the subject and content will be already filled with the task information and the task bundle will be attached. The user has the option to change any of these fields as he desires.

The following code adds all required information to the ACTION_SEND intent and starts the intent.

```
String subject = "You have received a new task";
String emailtext = taskToShare.GetDescription();
Intent emailIntent = new Intent(android.content.Intent.ACTION_SEND);
/*Add email data*/
emailIntent.setType("plain/text");
emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_STREAM,Uri.parse("file://" + arhivePath));
emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, emailtext);

m_context.startActivity(emailIntent);
```

The attachment is provided as an URI in the intent under the Intent.EXTRA_STREAM key.

4.2 Task Sharing

In this section we will look into the implementation of the task sharing features offered by AmIciTy. We will cover the creation of the asset bundle, the changes required to the task object and the send and import process.

4.2.1 The resource bundle

As we have seen in the previous chapters, a tasks can contain several files. When the user shares a task, we also want the sent data to contain these files. We therefore identify the need for a container to store these files. The solution chosen in AmIciTy is the zip archive file format because Java offers built in support for handling zip files.

The ZipInputStream class is used to add files to a zip archive. The class is best used with a BufferedOutputStream to improve performance as the output buffer reduces the number of write system calls required.

```
File outputTaskFile = new File(filename);
OutputStream os = new FileOutputStream(outputTaskFile);
zos = new ZipOutputStream(new BufferedOutputStream(os));
```

In the next step, we iterate through the files stored it the task, read their contents and add them to the archive as ZipEntry objects. The file names are preserved in the archive.

4.2.2 Saving the task object

After the assets are added to the archive, the last step consists of adding the serialized task object. This operation consists of several steps.

- Create a clone of the task object that can be modified without impacting the original object.
- Iterate through all file paths from the new object and replace the absolute paths with the file names. This is done because, on the receiving device, the absolute paths will no longer be relevant. Also, this removes sensitive information about the users file system from the task object.
- Serialize the task using a Gson object and store it in the archive as a ZipEntry with the "SerializedTask" file name. This filename is standard so it can be identified by the receiving application.
- The ZipStream is closed and the archive is ready to be sent.

4.2.3 Sending the task

After the task archive is created, it is sent using an email message. Detailed information about the email API can be found in section 4.1.6.

4.2.4 Importing the task

AmIciTy registers itself as a handler for ".amy" files using an intent filter specified in the application manifest file. This makes AmIciTy the default application for opening files with the .amy extension. The file can be opened directly from the received email, or from the file system using a file explorer application.

```
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<data android:scheme="file" />
<data android:mimeType="*/*" />
<data android:pathPattern=".*\\.amy" />
<data android:host="*" />
</intent-filter>
```

When a file is opened this way, the path to the archive is extracted from the ACTION_VIEW intent that started the HomeActivity. Using the path, the task is added to the application in the following steps:

- A folder with the tasks name is created in the AmIciTy folder from the external storage.
- The “SerializedTask” file is read from the archive and loaded in a Task object using a Gson instance.
- All other files included in the archive are extracted to the newly created folder.
- In the Task object, the contained file names are replaced with absolute paths to the extracted assets.
- The list view is refreshed and the imported task is presented to the user.

4.3 Custom UI elements

The Android SDK offers numerous default display elements that can create any basic user interface. However, each of these elements is highly customizable to offer each application a unique interface.

For AmIciTy, two major custom UI elements have been included, a list view that supports swipe to delete functionality, similar to the one found in Google Android applications and a variable height grid view, an element that allows the display of task assets in a tile layout.

4.3.1 Swipe to delete list view

To achieve the swipe to delete functionality a custom OnTouchListener is added to the list view. In the listeners OnTouch method, a hit test is performed to detect the child element of the list view for which the swipe action occurred. A VelocityTracker object is used to record the speed of the actions and to compare it with the minimum fling velocity. If the movement qualifies as a swipe, a dismiss animation occurs and the dismiss callback is called, where the task is removed from the TaskManager object, the adapter is notified, and the change is saved in the data file.

The following code represents the onDismiss method that is attached to the SwipeDismissListViewTouchListener object.

```
public void onDismiss(ListView listView, int[] reverseSortedPositions)
{
    for (int position : reverseSortedPositions)
    {
        Task taskToRemove = taskAdapter.getItem(position);
        m_tasksManager.removeTask(taskToRemove);
        taskAdapter.remove(taskToRemove);
    }
    taskAdapter.notifyDataSetChanged();
    SaveState();
}
```

The SwipeDismissListViewTouchListener class that detects the swipe movement is created by Google and can be found at the link provided in resource [5].

4.3.2 Grid view with variable row sizes

The tile layout that displays the task assets is created using a custom Grid view that presents the data in consistent column sizes, but allows varying row sizes between the columns. This allows a better usage of the screen space because asset tiles don't always have the same size.

The custom Grid view offers all advantages of a normal list view such as recycling views that, by scrolling, disappear from the visible part of the grid. The Grid view obtains data from an Adapter in a similar way to the default list view and notifies the activity about UI events using listeners like OnItemClickListener.

```
m_tileGridView.setOnItemClickListener(new OnItemClickListener() {

    @Override
    public void onItemClick(StaggeredGridView parent, View view, int position,
    long id) {
        TileItem item = m_staggeredAdapter.getItem(position);
        HadleTileClick(item);
    }
});
return true;
}
```

The full source code of the StaggeredGridView as well as documentation and usage examples can be obtained from resource [6].

4.4 Testing and Evaluation

The application was deployed for testing purposes using the Eclipse integrated development environment with the Android Developer Tools plugin. The plugin enables the programmer to inspect the logs created by the Android log system by displaying Logcat output. Logcat is the command that can be used to view and filter the content of the circular buffers that store logs from applications and the operating system.

The application was tested on a Nexus 5 phone with Android version 4.4.1, codename KitKat. Other Android devices were used to test sharing features.

Both the applications .apk file and the full source code can be obtained from the following git repository:
https://github.com/catalindragoiu/AmIciTy

Chapter 5

5. Application Usage

5.1 Task list

The applications landing page is made up of a list view showing information about current tasks. The task description is presented in text whilst information about the attached assets is presented in the form of icons. For example, if a task has a file attached, a small attached icon will appear under the description.

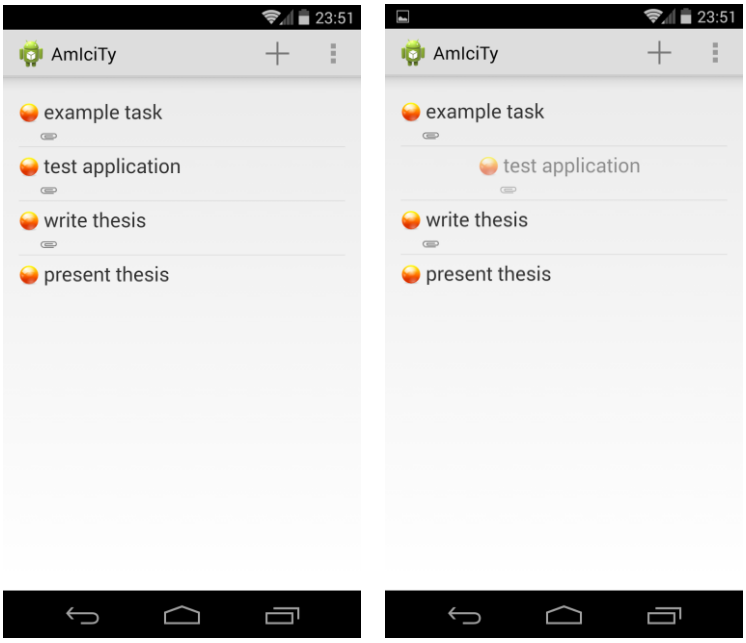

Figure 5.1 - AmIciTy landing page and swipe to delete animation

Tapping on a task opens up the task presentation and edit page. Deleting a task is done by swiping its list view entry to one side of the screen. The swipe to delete functionality is consistent with the similar functionality found in the notification are, in the Gmail email application or the Hangouts messaging tool.

The plus button opens a page that allows the creation of tasks.

5.2 Create new task

The create task page offers the possibility to add a task description, and to attach task assets. The option menu is located in the application's action bar; functionalities are displayed as icons if the space allows it or as text options in the expandable menu.

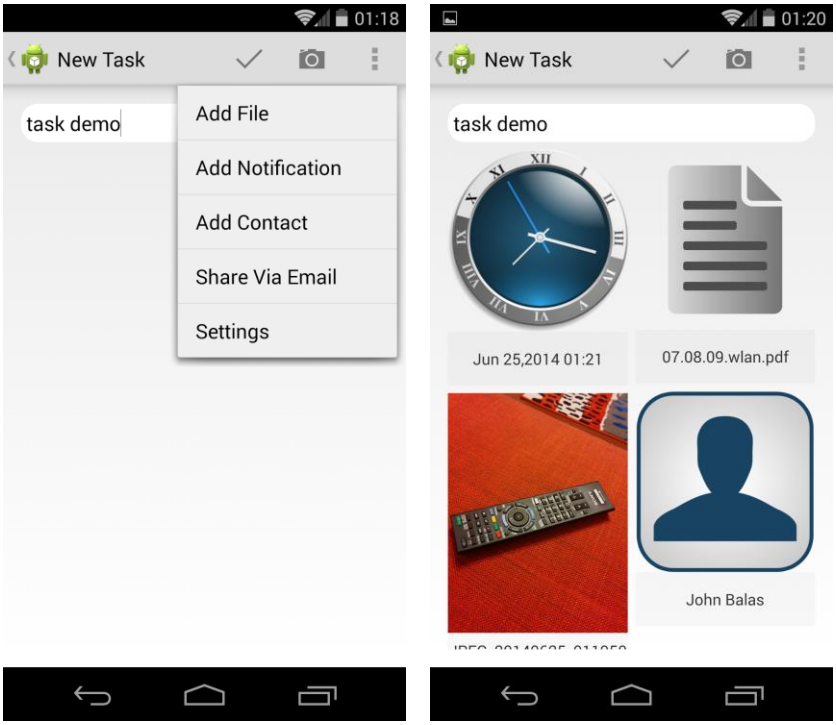

Figure 5.2 The new task menu, and assets added to a new task

The Add File button opens up the custom file picker that enables the selection of files. Tap on the name of a file to add it, tap on a folder to open it, or tap on the back button to exit the file browser. A special file is added to each page of the file picker that, when clicked, moves the picker up a level in the folder tree. The file name is "..".

The Add Notification button opens a date and time picker that enables the user to set a date and time for a notification. After the data is selected, a notification tile will appear on screen, with the date and time of the alarm presented in the tile's label.

The Add Contact button opens the contact picker displaying all contact stored on the device. Selecting a contact will add it to the current task in the form of a tile containing the contacts profile picture. If no picture is available, a default contact image will be presented. In both cases, the contact name is displayed in the tile's label.

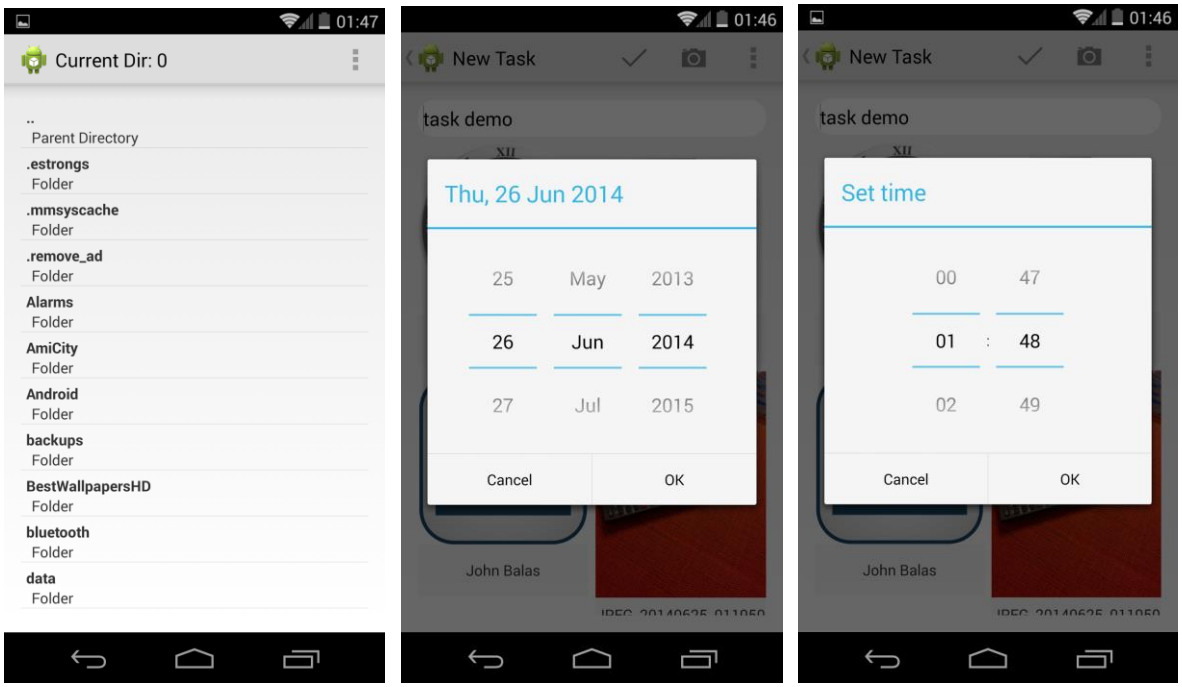

Figure 5.3 - The file, date and time pickers.

5.3 Access task assets

Selecting a file in the tile grid view will open the file using the appropriate Android application. Images will be opened using an Android image viewer. Selecting a contact will open up a page with contact options, such as call or send message.

If the user presses the back button from any activity that was started from the AmIciTy application, the tasks page that started the application will be displayed.

5.4 Share task

Tapping the share task button will open an application that supports the sending or storage of files, like Gmail or Google drive. For example, in the Gmail application, an email draft will be presented to the user with the archived task already attached.

An example of the share task flow is presented in the following screenshots

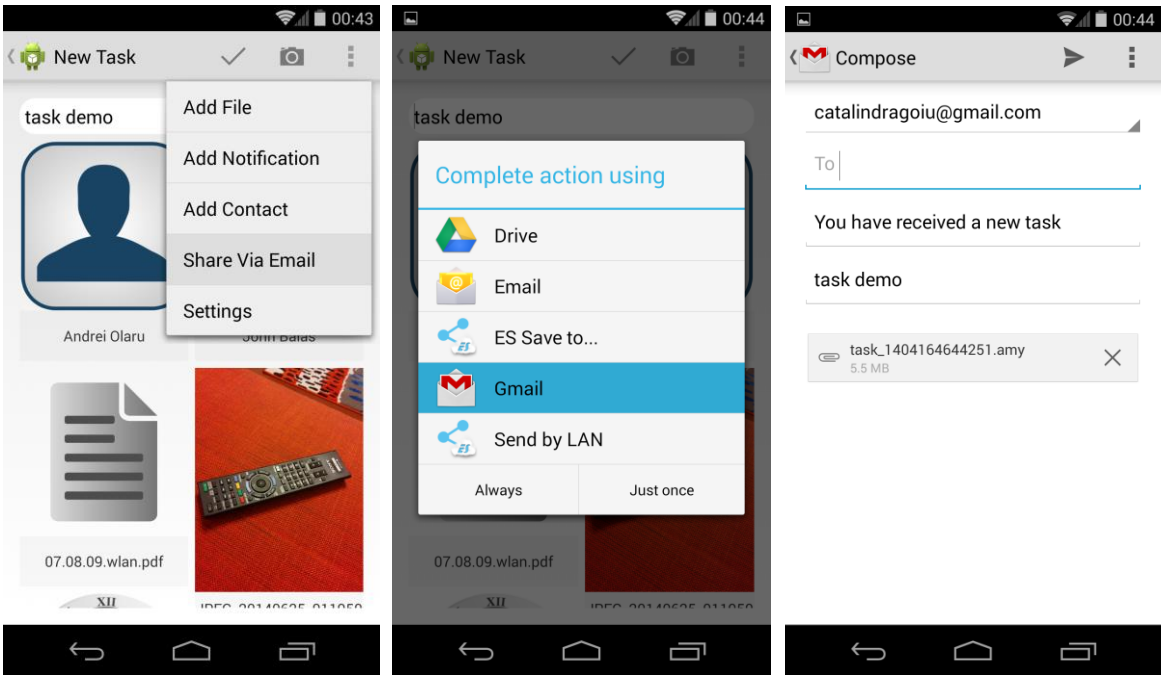

Figure 5.4 – The share task by email flow.
Tap Share Option -> Select Application -> Send Email

As you can notice in the screenshots, selecting the Share via Email option also allows other options in addition to dedicated email clients. This is because, in Android, applications can register for any kind of intent. In the above case, the Google Drive and ES application are registered for the ACTION_SEND intent and are added to the application list.

Chapter 6

6. Conclusion

The project resulted in an open source Android application that can be used to efficiently manage day to day tasks. The main achievements of the project consist of:

- The application has a high level of integration with the Android Operating System features. The following elements have been directly used in development: File Storage, Camera, Contacts, Notifications, Alarms, and Emails.

- Tasks can be easily shared with other users using Email, Google Drive or Bluetooth. Sharing a task also shares all attached resources.

- The project resulted in a documentation that contains information about the creation of an Android application with usage examples of various Android APIs, as well as the creation and usage of custom user interface modules.

All the projects main goals have been achieved; however, there are many new features that could benefit the AmIciTy project. We will explore future development ideas in the next chapter and present the challenges that must be overcome in their implementation.

Chapter 7

7. Further Development

The application development can follow three paths, improving the functionality and the general user experience:

- The integration of SMS, Calendar and other APIs to increase the number of use cases for the application. The new Google Cloud Storage API can also be used to store tasks for usage on multiple devices.
- Adding new share options, such as Android Beam and direct application to application sharing using a dedicated web service. This required the addition of user accounts to the application.
- Increasing the level of intelligence in the applications with features like scheduled tasks, intelligent reminders, obtaining data about the time spent on each task and using machine learning techniques to provide optimizations to the user's schedule. An analysis of user productivity can also be created using this data.

For the first path, a useful feature could allow the user to schedule the sending of a SMS at a certain point of time. This could also be implemented for emails or any other kind of messages that could be sent without human intervention. Integrating AmIciTy with the calendar application would allow the user another way of viewing future tasks. When a task is shared with another user, the calendar data should also be shared.

Cloud storage requires a unique way to identify the user. Creating a special account for the AmIciTy application adds problems such as security and privacy, and forces the user to remember another set of credentials. Using the Google account or the Facebook account for login is a much simpler solution both from the implementation and ease of use viewpoints.

To increase the ability of the application to assist with task advices, the user should be able to use signal when he begins working on a task and when he stops working. This would allow the user access to statistics like the total time spent on a task and the number of interruptions. Another interesting feature would be auto-managing a task, setting predefined breaks at specific intervals to improve the users productivity.

Porting the application to other platforms and to other operating systems should be a top priority as in users rarely use a single device on a day to day basis. Specific applications need to be developed for the IOS and Windows phone operating systems to cover mobile devices market. For desktop applications a web based solution is desired, offering a website that can be accessed from any type of PC regardless of

the operating system. To offer notifications on personal computers the application can be presented as browser extension. These extensions would allow the interaction with the user outside the main webpage.

AmIciTy, as it is now, creates a strong base platform, both of ideas and implementation, on top of which new functionality can be added as the product matures.

Bibliography

[1] Android Documentation. Online.
https://developer.android.com/

[2] Scenarios for ambient intelligence in 2010, Ducatel, Ken and Bogdanowicz, Marc and Scapolo, Fabiana and Leijten, Jos and Burgelman, Jean-Claude, 2001, http://www.ist.hu/doctar/fp5/istagscenarios2010.pdf

[3] Git Setup Guide. Online.
https://help.github.com/articles/set-up-git

[4] Git Project Repository. Online.
https://github.com/catalindragoiu/AmIciTy

[5] Custom OnTouchListener for swipe to dismiss functionality. Online.
https://code.google.com/p/dashclock/source/browse/main/src/com/google/android/apps/dashclock/ui/SwipeDismissListViewTouchListener.java?r=a7968ff883ad627869130ec6ec7c31554ca9954b

[6] StaggeredGridView repository and documentation
https://github.com/maurycyw/StaggeredGridView

[7] Android design principles. Online.
http://developer.android.com/design/get-started/principles.html