AALBORG UNIVERSITET
Student Report

Department of Computer Science
Selma Lagerløfs Vej 300
DK-9220 Aalborg Ø
Telephone (+45) 9635 8080
http://www.cs.aau.dk

Title:
Android Application Management System for Children with Disabilities

Project Theme:
Application Development

Project period:
Spring Semester 2011,
Feb. 1st to *May 27th*

Project group:
s601c

Authors:

Nikolaj Andersen

Anders Frandsen

Rune Jensen

Michael Lisby

Supervisor:
Saulius Samulevicius

Print run: 6
Pages: 101
Appendices (number, type):
#1: Overall system definition
#2: Test documents
#3: CD with source code

Abstract:

This project report describes the development of a system that allows a mentally hindered child to use a touch screen device in an environment that serves to help the child in the use of the device. The development of the modules we created is described, as well as our interaction with the other groups of the project.
An Android home screen application, and an application distribution platform, named GirafPlace, were developed by us. Both provides application filtering in regard to user capabilities. The home screen furthermore allows filtering based on the current location of the device.
To ensure the performance of the developed components, several tests were conducted during development. Integration testing were performed to ensure full functionality between the components of the entire system. The components developed by us were also tested individually, and the transfer protocol developed was verified using UPPAAL.
The entire system supplies a child with a platform for using a touch screen device in a safe environment, and supports installation of system-specific applications through a hidden interface for parents or caretakers of the child.

PREFACE

This bachelor report, has been written by group S601c, consisting of four software engineering students at the Department of Computer Science at Aalborg University in the spring of 2011.

The project has been developed in cooperation with three other groups, from whom we received several bug reports regarding the developed code. We thank these for invaluable help with hunting down bugs in our components of the system.

Also, we would like to thank our supervisor, Saulius Samulevicius, for input and suggestions throughout the course of the project.

Finally, we would like to thank Ulrik Nyman, who initially suggested the project to us, and the other groups, for help on specific requirements for autistic children.

Reading instructions While reading the report, the reader should be aware of the following:

- A guardian mentioned in this report, is either a parent or caretaker of a child.
- Commercial products like Google, Android, Java and Skype are trademarks of their respective companies. To increase readability in the report, these products and companies will not have a ™by their name.
- When referring to various types of literature, a number notation will be used. For instance, the reference to the book **Software Testing** will look like [1]. This number can be checked in the bibliography at the end of the report to see which book it refers to.
- All images inserted in the report will be numbered in ascending order with a reference to the chapter, in which the image is inserted. For instance, if chapter 6 contains five images, the

number of the fifth image would be 6.5. All inserted images will also contain a short description.

- The source code of our part of the developed system can be found on the CD-ROM attached to the report. An *.apk* file containing the GIRAF application, which includes our launcher and the administration group's administration module, is also found on the CD-ROM.

CONTENTS

CHAPTER 1
INTRODUCTION

Making user friendly devices has been a major focus for the electronics industry for decades. User friendly devices attract a larger user base and thus generate more money for the company selling them. However, that which is user friendly to one person, might be unintuitive to another. Especially, some user groups might feel that what is normally considered intuitive and easy to use, is in fact cumbersome and annoying.

One such user group is autistics, and especially autistic children. A parent of an autistic child, made us aware that autistic children can have trouble using a normal PC. Their hand-to-eye coordination may not be sufficient to percieve the link between the mouse and the pointer on the screen. They have a much easier time using a touchscreen interface though, as there is a direct connection with what they touch and what happens on screen. This could give autistic children an opportunity to use a computer in much the same way as other children do. However, most touchscreen devices still pose some problems for many of the autistic children.

These problems are mainly associated with the fact that using a phone in recent years has been increasingly demanding on ones understanding of electronics and operating systems. New phones allow users to control and adjust almost every aspect of the device, even so far as installing new applications to handle basic functions as messaging and phone calls. Such a wide range of options often serve to confuse autistic children as they comprehend the system differently from others.

While the subject of autistic children and their usage of electronic devices has been essential to the project, the cooperation with other groups have been the main focus of our development. This is mostly due to the subsystem that we developed mostly being aimed at the other developers.

As certain aspects of this project were new to us, mainly the Android platform, and certain aspects of software testing, a short description

of these subjects has been included in the report. They will make up the first two chapters, and it is suggested that readers who likewise does not know much about the subjects, reads these. Readers familiar with these topics are suggested to skip these chapters, as the content is not directly relevant for the project, but rather background theory we found was required for the project.

CHAPTER 2
ANDROID

This chapter describes the Android platform used in this project. The underlying architecture will be explored and the virtual machine used by the Android Operating System will be briefly explained. Finally, the structure of applications for Android will be described, as well as how the different parts of an application interact with each other. The chapter is based on [2], the official Android developer web page, where extensive information on the platform is freely available.

2.1. System Foundation

Android, developed by Google, is an open source software solution stack for mobile devices. Behind these buzzwords hides the fact that Android is not just one piece of software, but rather consists of a number of subsystems that together constitutes the final product. The architecture of Android can be divided into five parts: The Linux Kernel, Libraries, Runtime entities, Application Framework and Applications.

Linux Kernel The core of the system. Includes services for managing resources for the system, such as memory or processes. It also includes the drivers needed by the system in order to control the display, audio entities, wifi, etc.

Libraries Android comes bundled with a range of libraries. This includes 3D, 2D, SQL and media libraries among many others. A large part of the Java class library is included too[1].

Android Runtime The Dalvik Virtual Machine (DVM). This is a special version of the Java Virtual Machine(JVM). See section 2.1.1 on the following page for more information on the DVM.

Application framework A framework that makes it possible for developers to create applications, by giving them access to the system via

[1]Certain specific classes has been omitted as they have no relevance for mobile devices.

specific APIs. The architecture of the applications is designed to make it possible to reuse individual components of applications, including allowing other applications to make use of them. It also includes a set of services and subsystems that complement each application developed for the system.

Applications A set of standard applications for the system. These are shipped with the system to provide basic functionality such as messaging and browsing.

2.1.1. Dalvik Virtual Machine

The DVM is a special variant of the JVM. The difference is that the JVM is stack-based, whereas DVM is register-based. This means that DVM uses register operation codes, which is comparable to byte code instructions. Thus, DVM requires fewer instructions than JVM as it has no need for instructions to load and manipulate data. Each instruction however, is larger than the corresponding instruction in JVM, since it must encode the source and destination registers.

DVM is used in Android mainly because it uses less space and allows multiple instances of itself to be executing concurrently, one for each process. This is a good thing for security reasons, since applications will thus execute completely independent of each other.

2.1.2. Application

Each Android application runs in its own instance of the DVM. This instance will be available for the application until the application is closed, or until the Operating System (OS) deems it necessary to use the resources of the instance for another instance of the DVM. The execution of each application is thus separated from other applications, so that if one application crashes, it does not affect any other applications.

When an application is installed, a user is created for the application. Since this results in one user for each application, the user id is used as an identifier, along with the package name[2]. Creating a user for each application provides a security advantage, as the files of an application can only be read or written to by the user of the application.

[2]No two installed applications are allowed to have the same package name.

On most operating systems, an application has a single entry point, the main function. An Android application can have several entry points. To achieve this, applications have been redefined to revolve around five concepts: activity, service, content provider, broadcast receiver and intent. All these are declared for each application in its Android manifest file, see Section 2.1.2 on page 17.

Activity

The activity concept is the idea that an application is not a single entity, but rather a collection of activities. A very simple music player, for instance, consists of two activities:

- Select a song from a list of available songs on the system.
- Play a song.

The first activity will browse the "*Music*" folder on the device, and display all the songs currently available in a list. Once a song is selected by the user, the second activity will start and provide the user options to manipulate the playback of the song, such as play and pause. Notice that the actual playback of the track is not handled by the activity, as it is normally done by a service, see section 2.1.2 on the following page for details.

Activities are only used to display or manipulate data, and should never be used to store data. Data can be stored momentarily, while it is being used or edited in some way by the user, before again being stored properly. Storing the data in the activity is not only "bad style", but it can result in the data being lost. This is due to the way the activity lifecycle, seen in figure 2.1 on page 13, works.

All of the methods listed in the lifecycle can be overridden in any class extending the `Activity` class. Android handles multitasking, and these methods allows the activities to adapt to application switching. Only the functions that are needed to be handled in a special way by the activity should be overridden, as the standard implementation handles anything else needed by the system. Because of this, an overridden function must call the super-method as the very first statement.

When rotating the device, and thus the view, the interface needs to be redrawn. To make sure this is done properly, the entire lifecycle is started from *onCreate* when the phone is rotated, since this method

should handle UI-setup. This will cause any data stored in the activity to be lost. Data can be saved and reused, but the mechanism that provides this is only designed to handle small amounts of data, like the content of the text box in the SMS application, not entire system states.

Service

A service is a component that provides background logic for Android applications. Unlike an activity, a service will, once started, only stop when it is done executing or when it is forced to stop by the system due to a resource shortage. The background logic provided by a service must not be confused with multi threading. Instead, it means that a service has no user interface. Quite to the contrary, a service runs in the main thread of the component that started it.

Since this often causes the user interface of that component to become unresponsive, most services use another thread for performing the actual task. In fact, this behavior is so common that the Android class library contains a class, `IntentService`, that provides a worker thread amongst other helpful, default implementations.

Services are classified as being either started or bounded - or both. A started service is normally used to perform a single task, like playing a song, while a bounded service provides a more generic functionality that can be used by many, different components. Like activities, services have a lifecycle, albeit simpler than the lifecycle of the activities. Each classification of a service has its own slightly different lifecycle, both of which can be seen in figure 2.2 on page 14. If a service is both bounded and started the methods `stopSelf()`, used by the service itself, and `stopService()`, used by other components, will not actually stop the service before all clients have been unbounded.

Content provider

The content provider component is a data abstraction. It provides common data manipulation functions such as querying, insertion and deletion. The underlying database can be almost anything usable to store data. However, it must be able to be abstracted as a table form, with at least an _ID column for numeric primary keys. Android provides additional helper classes for implementing a content provider

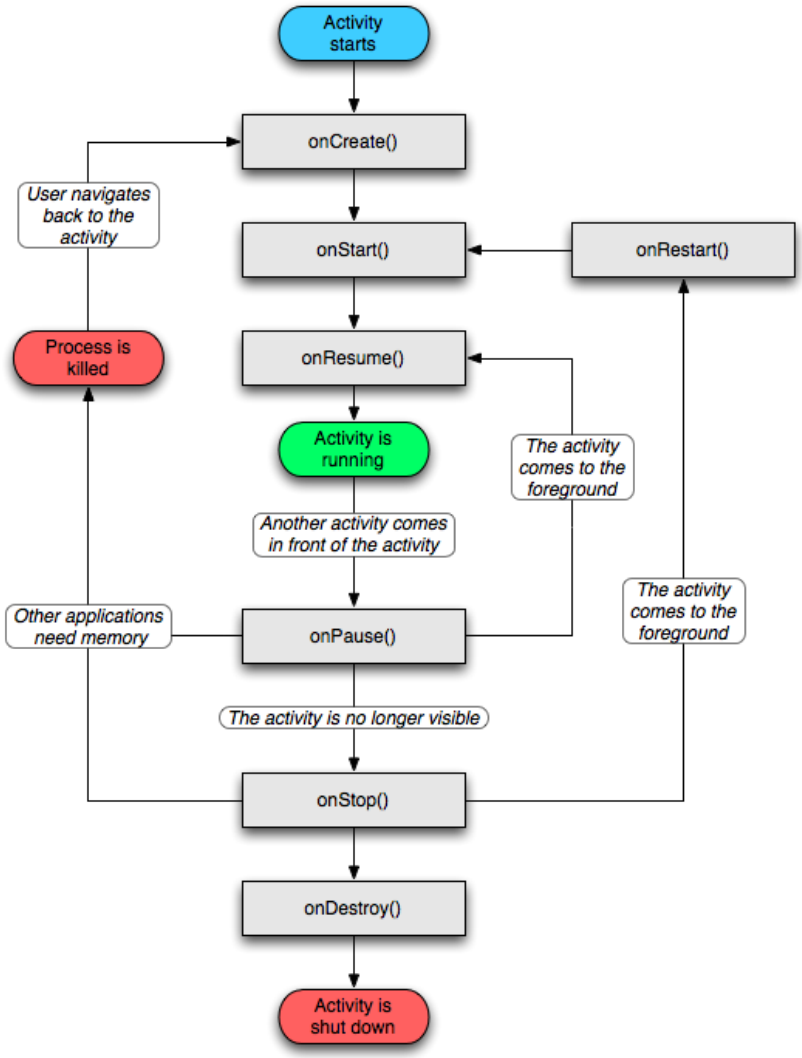

Figure 2.1.: The Activity Lifecycle[2]. The sharpcornered boxes are methods that should be handled the events in the white boxes if the default handling is not adequate.

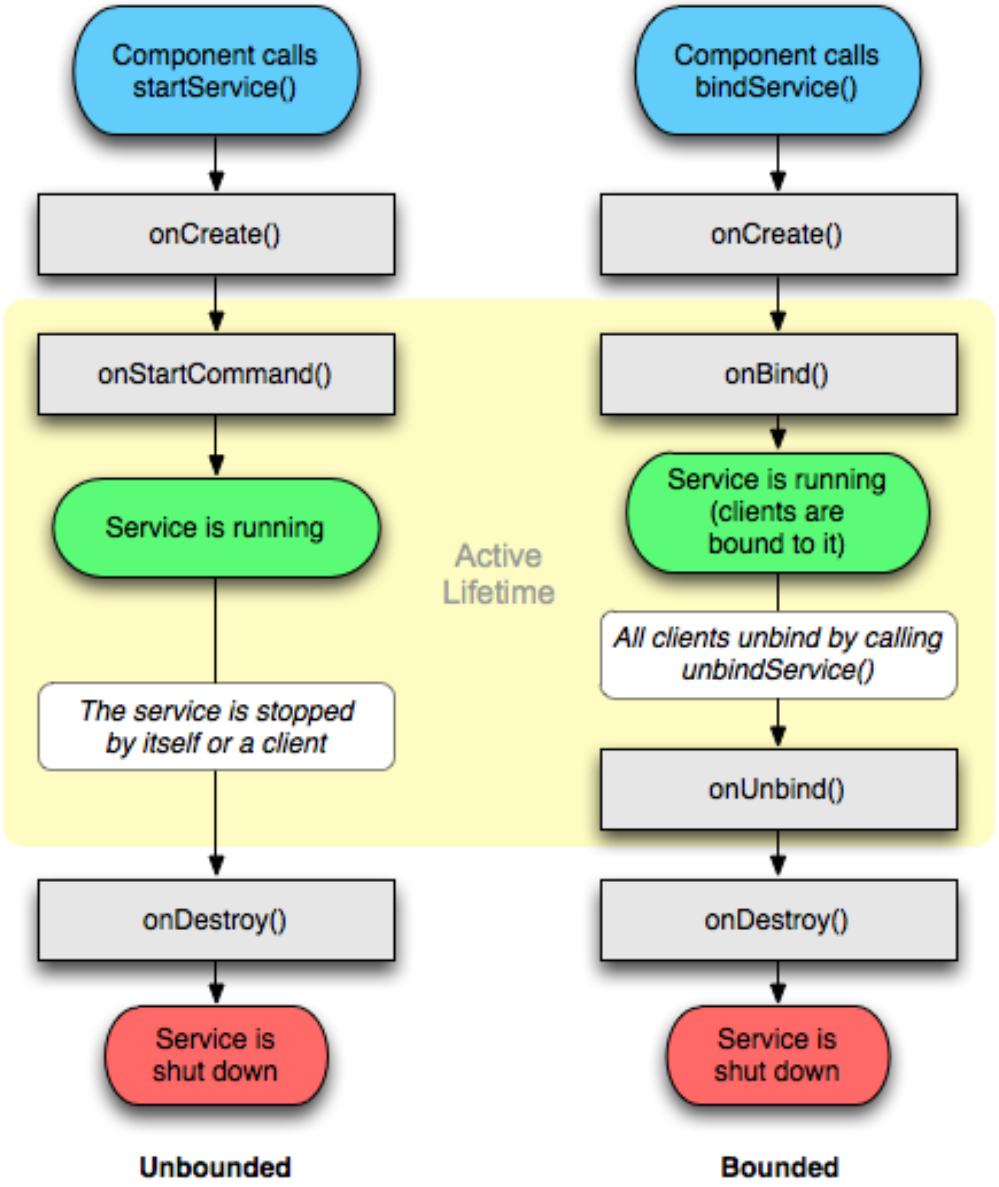

Figure 2.2.: The two service lifecycles[2]. The sharpcornered boxes are methods that can be overridden to handle specific events.

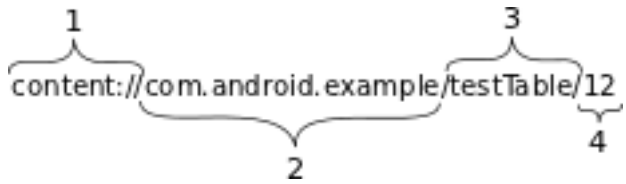

Figure 2.3.: The four parts of a content URI: (1) The scheme identifying the URI as a content URI, (2) the authority, identifying the content provider, (3) the table and (4) the id of a specific item.

over popular databases, for instance SQLITE, as they fit the table abstraction and make use of the query, insert and delete functions directly. In this case the content provider is used to validate the queries before executing them on the underlying database.

Only the system interacts directly with a content provider. When another component, such as an activity, requires data stored in a content provider, it uses a content resolver. The content resolver is a system object, provided by the *getContentResolver()* method. It has the same data manipulation methods as the ones defined in the content provider, but they also take a single URI as a parameter, to identify the table on which the query should be executed. The URIs used to identify content are of the form seen in figure 2.3. The id in the figure can be omitted when a query should affect the entire table, and not only a specific entry. It must be omitted when inserting.

A content provider can either be private to the application that declares it, or available to all applications. An example of a public content provider is the Android contact content provider.

Broadcast Receiver

A broadcast receiver executes a piece of code when it receives a broadcast from the system. It can be listening for broadcasts at all times, or it can alternate between listening and not listening. This is done by letting it alternate between being registered or unregistered while an application is running. Standard broadcasts could be for events such as installation of new packages or a low battery warning. It is also possible to send custom broadcasts, which will be system wide, just like the standard broadcasts. To increase security, it is possible

to require an application to have permission to listen for a broadcast. This is used by many of the standard broadcasts, which are only used by the system, and cannot be registered by non-system applications.

Intents and intent filters

An intent filter is the specification of when an activity, service or broadcast receiver should be started by the system. They are based on strings of the following variations:

Action A string naming the action that is to be performed.

Category A string containing more information about the kind of component that should handle the intent.

Data A URI that points to data needed by the activity. This is often, but not always, a content URI.

Using the *getExtra()* and *putExtra(String,Value)* methods on the intent, it is possible to transfer primitive data types, such as numbers and strings that are stored using string indexes, from the starting component to the newly started component. To avoid spelling errors in the intents, which can be hard to detect as they will compile with no problems, the `Intent` class contains a number of strings containing the correct spelling for many of the actions, categories and extras used by Android.

An example of an intent filter, the launcher intent filter, used by most applications, can be seen below. All applications displayed in the launcher of the device have an activity with an intent filter specifying an action and a category. The parentheses contains the variable name of the string in the `Intent` class:

Action android.intent.action.MAIN (ACTION_MAIN)

Category android.intent.category.LAUNCHER (CATEGORY_LAUNCHER)

Broadcasts are in fact intents as well. There is no actual difference between an intent used by an activity, such as ACTION_MAIN, or an intent used by a broadcast receiver, such as ACTION_BATERY_LOW. But listening for a broadcast on ACTION_MAIN would be pointless[3], as the system will never broadcast this intent.

[3]It can be succesful by broadcasting the intent oneself, but in that case a custom intent should be used instead.

Activities can be started using the *startActivity(Intent)* method. If this method is called with an intent that multiple activities match in their intent filter, the system will ask the user to select which it should execute. This is so deeply incorporated into the system that even entering a number and pressing dial may result in the user being asked to choose between the default phone application or a phone application such as Skype, if installed. It is possible for the user to select a default application that will then be used in the future, in case of multiple activities matching.

The Android Manifest

The Android manifest is an XML file where the components of an application is declared, along with the intent filters that will initiate them. If a component is not declared in this file, Android will not recognize it. This makes the component unusable, even if the intent that should start it declares the direct package and class name, rather than an action. Code example 2.1 on the following page shows the manifest file for the "Hello World" example that a new Android Project in Eclipse would contain.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
3       package="hello.wolrd"
4       android:versionCode="1"
5       android:versionName="1.0">
6     <uses-sdk android:minSdkVersion="8" />
7
8     <application android:icon="@drawable/icon" android:label="
        @string/app_name">
9         <activity android:name=".Main"
10                  android:label="@string/app_name">
11            <intent-filter>
12                <action android:name="android.intent.action.
                    MAIN" />
13                <category android:name="android.intent.
                    category.LAUNCHER" />
14            </intent-filter>
15        </activity>
16
17    </application>
18 </manifest>
```

Code example 2.1: : The manifest file for the "Hello World" Android project.

CHAPTER 3
SOFTWARE TESTING

Testing software has not always been a major concern for software companies. However, as the field has developed over the years it has become apparent that testing software before releasing it, is not only a good idea, but absolutely necessary in order to release a quality product. This chapter will describe the basic principles of software testing. The chapter is based on [1].

3.1. Test types

Testing software can be done in different ways. The primary distinctions between the different methods are whether they are *static* or *dynamic* and whether they are *black* or *white box* testing. Each of these different approaches has its uses, and thus, we will give a walkthrough of each of them.

3.1.1. Static Black Box Testing

Static black box testing is used in the early phases of software engineering. It is static in the way that during the test, no program is executed. The testing is done solely on the specification of the system. That is, no code has been written yet, and the test is performed on the specified features of the system. The software tester does not know how the system would behave, but instead has input and intended output values to work with. Thus the testing is done in a "black box".

This might sound like a far abstraction from an actual running program, but it has its uses. Imagine a system specification that never once states how the system should behave, if, for some reason, something does not go as planned. Such a system might look fine on paper, as one would expect it to work, but there would be no telling what

would happen to the system, or the data being worked with, if some error did occur during runtime.

"Bugs" such as these can be caught even before the code is implemented. The software tester will be able to spot that some scenario, however unlikely, has not been accounted for in the specification. The design team can then take action to correct this, before man hours are used on actually implementing the system.

These bugs might not be as obvious as the aforementioned example. Often they stem from the specification using words such as; *all*, *every*, *none*, *always*, *clearly*, *some*, *usually* or "*such as*". These are all words that one has to be wary with when using them in a specification. For instance, should the system really "*always*" return "`true`" when running some method? Should it "*never*" return a given result? What exactly is meant by "*usually*"? One can see how words like these can cause problems, as they fail to specifically describe the intended functionality of the system.

Another often occurring mistake is a specification saying "`if` **`A`** `then` **`B`**", but then forgetting the "`else`" clause. What happens if **`A`** is not true? Catching all of these errors before the programmers get to work, will save everyone in the team a lot of work finding and fixing bugs, once the system is implemented.

3.1.2. Dynamic Black Box Testing

Dynamic testing differ from static testing in the way that the system is actually executing. The tester is using the system as a user would, and will test by comparing the output from the system with the intended output given in the specification. Again, this works as black box testing, as the software tester has no idea what goes on inside the system. He can only compare input and output values.

3.1.3. Static White Box Testing

White box testing is done on the actual written code. Static white box testing is done without executing the code, but instead by examining the code and the general architecture of the system. The idea is to find areas that are problematic. That is, the test is not limited to finding bugs, but also areas where problems might occur in the future. This

is also good because it gives the software testers, doing the dynamic black box testing, an idea of where to search for errors.

3.1.4. Dynamic White Box Testing

Performing white box testing while the code is running is similar to debugging. However, dynamic white box testing is focused on finding bugs, whereas the main goal of debugging is rather fixing those bugs. This testing can be done in several different ways, but the basic idea is to look at the code and examine how it performs while executing.

3.2. Requirements for test cases

When testing software, independent of the chosen type of test, it is important that one tests the system in an orderly fashion. It is important that the test is under control at all times, and that it results in usable feedback to the developers. Thus, one needs to plan the test cases thoroughly before testing.

3.2.1. Test to pass and test to fail

Testing software to pass is used to ensure that implemented functionality works. When testing software to pass, the tester is required to use the software gracefully. This means using the software as intended.

The system should initially be tested to pass, as there is no reason to try to break it, if the basic functionality does not yet work as intended.

After ensuring that the basic functionality of the software in question is as required, it is necessary to bring the software to the edge. When testing to fail, the test cases are designed to make the program fail. This could, for instance, be done by pressing buttons unexpectedly. An example of this is to check, whether a text editing program actually saves the content, if the user closes the application while attempting to save the document.

3.2.2. Equivalence partitioning

Since it is impossible to test all possible inputs, test case developers use equivalence partitioning to test groups of input that would result in similar output.

In a simple calculator, the inputs can be partitioned into additions, subtractions, divisions and multiplications. Each of these can be further partitioned if one has a slight understanding of how a computer computes values.

Boundaries

When testing input values it is a good idea to test their boundaries. For instance, there is a natural boundary for an integer, every time its encoding is added an extra bit. If these boundaries are not handled properly, strange bugs may occur, that can be difficult to detect.

3.3. Test and Verification

It is impossible to guarantee that software will never fail. However, using verification, the model of the software can be mathematically proven to be sound.

It is important to remember that verification cannot be used to guarantee that the software, on which the model is based, is as sound as the model.

One could automatically generate the software from the model, but at this time, no such tool exists for Java. Keeping this in mind, verification can still be a helpful tool to discover problems in the model that would also exist in the software.

CHAPTER 4

PROJECT SETUP

The initial requirements of the project is discussed in this chapter. Some notions used here, like group spokespersons, are explained later in chapter 6 on page 33.

4.1. Initial requirements

The initial proposal for the project was given to us by the parent of an autistic child. We have used the conversations with this parent as a guideline as to which features the system should have. He suggested that we create a way to make "*Android applications usable by autistic children*".

In cooperation with the other groups we modified the project proposal. We want to create module based software that allows autistic as well as otherwise challenged children to use phones in a safe environment. For this, the Android platform is good, as it is completely open source.

The parent specifically asked for the children not to be able to use the phone functionality of an Android device, thus it will be a requirement to mask most common functionality of the phone.

We divided the responsibilities of such a system between the groups, to allow each group to work somewhat independently. See chapter 6 on page 33 for more details on how the collaboration was done.

The intended system modules will be:

- An administraion module.
- An application distribution module, including a launcher.
- Applications for the final system.

Administration module

The administration module should supply a user interface to allow the guardians of the child to define system settings. This includes general settings of the developed system[1], accessible by the launcher and applications, as well as application- and launcher-specific settings. This module should not be accessible by the children, but only the guardians. Furthermore the module should provide interfaces to allow other modules to interact with the user data.

Distribution and Launcher module

The distribution module should make it possible for developers to distribute their applications to all devices where the system is installed. It should do so in a manner similar to the Android Market Place.

The launcher module should provide the home screen of the system, and should function as the entry point to the rest of the system. In order to mask the administration module from the child, a special key combination will be needed to gain access to this.

Installed applications will be listed in the launcher in accordance to the user specific settings from the administration module. To execute an application, the child needs only tap them.

Applications

An application is a non-required part of the system, and could, for instance, be educational or entertaining. The applications should also allow access to the administration module by use of the special key combination. Applications should be allowed to access user data, and tailor the application content, to comply with the child's capabilities.

4.1.1. FACTOR analysis

At a grand meeting between all the project groups, the FACTORS of the entire system was discussed. The group spokespersons then used several subsequent meetings finalizing the analysis in accordance with the ideas set forth from the parent. The formal FACTORS can be seen

[1] The generic Android settings should also be accessible through this module.

in appendix A on page 77. Here, we present a list of the requirements from the FACTORS that directly affects our module, therefore the subsystems part has been omitted.

Functionality

- The system should offer installation of new applications.
- The system should mask the normal functionalities of the device to the user.
- The system should give the opportunity to control the usage of, and access to, system settings, user profile settings and application settings. This should be done in accordance to the current location and/or time.

Application Domain

- Children with limited mental capabilities due to handicap or age.
- Parents and kindergarten teachers, henceforth called guardians, administrating the system.

Conditions

- The system is being developed by a number of study groups as a study project, and thus has a hard deadline that cannot be exceeded.
- The system should be simple and intuitive to use.
- The system should be designed in such a way that it is customizable to the individual child and its disabilities.
- The system should allow guardians to limit the functionality of the system or specific applications.
- The system should be maintainable.

Technology

- The system must run on Android touch devices.
- Different hardware should be supported, although it is required that the unit runs Android 2.2 or newer.
- The system should be developed using Android Software Development Kit (SDK) version 8.

Objects

No hard requirements defined for our module.

Responsibility

- The system should provide the opportunity to install third party applications developed specifically for the system.
- Through a home menu, the system should control which applications, the user is allowed to access.

These points, including the omitted ones, are also summed up in a system definition available in appendix A on page 77.

4.2. Name of the system

In accordance with the other groups of the project, it was decided to name the developed system GIRAF. The name of the system was a working title in the beginning of the project, and, as no other suggestions came up, the name stayed.

CHAPTER 5
COMPONENT ANALYSIS

Most of the analysis is based on talks with the parent who suggested the project, as he has first hand experience with the subject. Other parts are in accordance with requirements from the other groups of the project. This chapter describes result the analysis phase conducted to define the direction of this project.

5.1. Analysis approach argumentation

Any project would benefit from conducting a complete user group analysis. However, development of our module has to begin before the application developers begin their development. If not, they would have to redo large parts of their implementation, as we continually provided more features.

As such, performing a user group analysis would cause the other groups, who relied on our project, to need to postpone their development phase until we were able to deliver most of our system. As this approach would give unnecessary delay for every single other project group, we did not find it appropriate to perform a complete analysis of the users' needs. Instead we used feedback from the previously mentioned parent, in collaboration with the other groups.

This has had little impact on our project however, as the areas of our system that would directly benefit from such an analysis are very few. The children are only interacting with the user interface of the launcher. The interface of the remaining components are either to be defined by other groups, or used by guardians who do not have any special needs, apart from those of normal touchscreen device users.

All user interfaces are therefore merely our attempt to provide an easy to use way of interacting with the system. In a final system, a user analysis should be performed, and the results hereof implemented in the system.

5.2. Users

While no in-depth user analysis has been conducted, we have still, in collaboration with the other groups, performed a simple analysis of the users of the system. They are divided into two categories:

Children The children with disabilities that are going to use the applications. They cannot be trusted with the settings of the system. They require a completely safe environment on the device where nothing they can do will break any of the software.

Guardians The parents or caretakers. As the name implies, any person who is in a position to guard the child. These users are trusted with all the settings the device offers, both the specific GIRAF settings, and the general device settings.

The initial project proposal called for a system for children with autism. Since the spectrum of autism is as wide as it is, the groups decided it would be too difficult to make the system for children simply with the diagnosis autism. Instead a list of capabilities were produced by the groups during a brainstorm. It was also decided, in accordance with the parent's wishes, that a device should only belong to a single child. Thus, no multi-user support would be needed. The capabilities are:

canRead Whether the user is capable of reading.
canDragAndDrop Whether the user understands the concept of dragging and dropping objects on the touchscreen.
canHear Whether the user is capable of hearing.
canNumbers Whether the user is capable of understanding numbers.
canAnalogTime Whether the user is capable of reading the time on an analogue clock.
canDigitalTime Whether the user is capable of reading the time on a digital clock.
canSpeak Whether the user is capable of speaking words and/or sentences out loud.
canUseKeyboard Whether the user is capable of using a hardware keyboard.
hasBadVision Whether the user has a poor eyesight.
requiresSimpleVisualEffects Whether the user needs simplistic interfaces.
requiresLargeButtons Whether the user's hand-to-eye coordination is hindered, thus requiring larger interface elements.

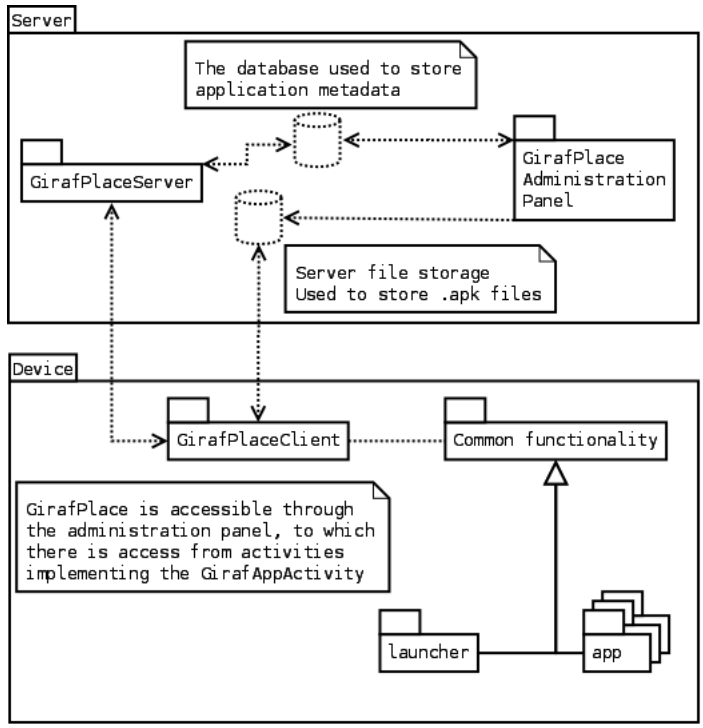

Figure 5.1.: The components intended to be a part of the final system.

5.3. System components and requirements

The system should consist of the components shown in figure 5.1. For each component, a requirements specification, derived from the system definition, will be given. These requirements will be used as foundation for the design of the individual components.

5.3.1. Overall system requirements

An initial idea when designing the overall system was to create a custom OS in order to mask phone functionality. As Android is open source, it would be possible to do so by modifying the source code to suit our needs. However, the system definition requires the system to be executable on several different types of hardware.

Therefore, it was not possible, in the scope of this project, to create a custom OS, as it would not be feasible to develop such a system

for use on several devices. It would require us to acquire drivers for all device types, and make custom builds for each device type. The responsibility of locking down system functions has therefore been moved to the launcher and each application, as common functionality.

5.3.2. Common functionality for applications

The parent requested that the system used on-screen buttons as the hand-to-eye coordination of autistic children are usually lacking. However, in his experience, they are more than capable of using touch screen interfaces.

He also requested that normal functionality of the device, including phone capabilities, was removed, as it could possibly confuse the children. Also, it is a requirement that the applications must allow access to the administration module by pressing a special key combination. Doing this in an application should allow the application developers to open the application specific settings in the administration module.

Every type of device might have different hardware keys. A few keys are required to always be present on Android devices. Some of these keys unfortunately cannot be ignored if pressed.

The Home key is locked by Android to make sure that a user will always be able to return to the home screen, and not become stuck in some application. Also, the End Call button cannot be ignored, to ensure that you can always hang up. The latter should not cause a problem for us, as the phone functionality is to be ignored completely.

Initially, we wanted to ignore keypresses to the Back key. During development, however, it was discovered that this functionality was actually needed in every single screen. In order to not waste interface space on screen, it was agreed upon with the other groups, that we were to remove the already implemented block of this button.

Finally, it was decided to enforce that all applications execute in fullscreen mode. This is an indirect requirement in masking the normal functionalities of the device, as the menu bar in non-fullscreen applications on Android gives access to ordinary phone functionalities.

The requirements for the common functionality are thus:

- Ignore all hardware key presses except Back, Home and End Call.
- Force all applications to execute in fullscreen mode.

- When the special key combination is pressed, enter the administration module.
- Allow application developers to change the activity started when using the special key combination.

5.3.3. Launcher

According to the system definition, the system should control access to the installed applications through a home screen. This home screen is also known as a launcher in Android. The launcher of the developed system must, however, only list applications, the user is actually capable of handling.

As a direct corollary from that, applications not developed for the GIRAF system should not be listed, as there is no way to control, whether the user is able to handle such applications. The launcher should only display text, if the user is actually capable of reading. A user not capable of reading would thus not see the name of an application under its icon in the launcher.

The guardians should be able to decide if some applications should not be accessible in certain situations. This could, for instance, be the masking of any game applications while the child is at school.

The requirements for the launcher is thus:

- List only applications that is part of the GIRAF system.
- List only applications that the user is able to use, according to the user's capabilities.
- List only applications if their usage is not limited by the current location or time.
- Only display application names if the user is able to read.

5.3.4. GirafPlace

The system definition requires the system to offer distribution of applications. This could be achieved by requiring applications to be uploaded to the Android marketplace. However, this would not be easy to use for a guardian, as they would have a hard time finding applications specific to GIRAF. This would also make it harder to filter

applications according to the user profile. It would therefore be better to store applications in a separate market place developed by us, so we can stay in control.

This will make the filtering functionality of the launcher less useful. However, it was required from several of the other groups that the launcher functionality not be deprecated, even though there would be filtering on the marketplace. Both filters are thus present. This is not double work, as the user profile may change after initial installation.

To implement the GirafPlace system, four main parts need to be developed:

GirafPlace Application Storage To allow for exchange of applications between the Administration Panel and the Server, a shared file storage must be developed. The storage must save both files and meta-data in such a way that it can easily be retrieved when necessary.

GirafPlace Administration Panel The GirafPlace Administration Panel should allow application administrators to add their applications to the GIRAF system.

Giraf Transfer Protocol The Giraf Transfer Protocol (GTP) should be used to allow the client to request applications from the server.

GirafPlaceServer The server must use the GTP to accept requests from clients for application lists, and return this list to the client. The list of applications should only contain applications, the child using the device is capable of using.

GirafPlaceClient The client must use the GTP to contact the server and retrieve the list of available applications for installation and /or update, as well as allow users to uninstall applications. The client must use HTTP to download the installation files from the file storage on the web server. The client should be an Android application itself, accessible from the administration module.

CHAPTER 6
MULTIPROJECT

A major part of this project was working together with other groups on a larger product than we would as a single group. This chapter will describe how we worked together, including how it differs from working as a single group.

6.1. Group Structure

The project consisted of four individual groups working together. It was decided that every group should focus on a specific part of the complete system. This was mainly to make it possible for the individual groups to focus on their own assignment, instead of the broader project. Doing otherwise would have required all groups to cycle assignments between them to make sure everyone worked on everything. We did not see any advantages to that approach, and thus it was quickly decided against.

Our group worked closely with the Administration group to create efficient user interfaces for the application groups, allowing them to utilize the properties of the system properly.

6.2. Cooperational challenges

When working on a multi-group project, one can encounter some issues that would not be present in a single group project. Many of these are small, but a few have a large impact on the development of the product and on the working environment. We will explain how we tried to solve some of these issues.

6.2.1. Intergroup meetings

Managing a group of four can be challenging enough, so it was quickly decided that having all group members at inter-group meetings would not be beneficial. Instead, each group elected a spokesperson for to handle communications with the other groups. Regular meetings between the spokespersons were held, at which they would inform the other groups. Issues of mutual interest could also be discussed.

6.3. Planning

The planning of the entire system was done in several phases. This was so that each individual group could make their own drafts of documents concerning the multiproject, such as FACTORS analysis. Later, the drafts were discussed with the other groups. The discussions on the final documents were done by the group spokespersons.

It is important to note that the decisions were taken on a general level. That is, each group were, if so inclined, welcome to make their individual analysis differ from the general one. For instance, we chose to make the launcher specifically aimed at children that for some reason cannot use a regular device. Another group made an application solely intended for autistic children, and would thus have a much narrower user group than the multiproject in general.

6.4. Design

The Android platform was used throughout the entire project, and as such not much architecture design was necessary. Most of the design decisions therefore consisted of making the communication between the different modules as efficient as possible. This created some problems, as the groups were accustomed to doing things in their own way. In the end, it was up to each group to design their own architecture, as long as the components of other groups were able to communicate efficiently with the interfaces provided.

To streamline this process even further, it was decided to make the administration module of the system function as the central communication link between the components. Most components will pull

data from the database in the administration module, as well as send new data to it. Other components can then access that data in the same way. This makes it easier for the groups as they no longer need to tailor their components to all groups, but only the standardized administration module.

6.5. Integration

On two occasions there was a need to merge the code developed by us and the administration group. On the first occasion, we both provided classes that should be used as a common library for applications. Instead of requiring the applications to import both libraries, it was decided that they should be combined into a single library. This was done by creating a new Java package inside the largest of the two libraries, the administration group's *sw6.lib*, and moving the classes of the other library into that package.

The other was a larger operation, as it was three applications, the administration interface, the GirafPlaceClient, and the launcher that needed to be combined.

This was a large operation, as it was required to integrate not only classes, but also icons, layout and menus amongst other files. Each file had to be moved one by one, as moving entire folders under subversion control may cause problems if the hidden *.svn* folder[1] is included in the move.

Two files also had to be merged: The Android manifest file which should include definitions and permissions for both projects, and the launcher class file, which should include a "first run" method for setting up the user profile, the first time the launcher is executed.

6.6. Code sharing

In any coding project, code sharing is an issue which must be addressed from the very beginning. While it is an option to send files via email or something similar, most coding projects will use a version control system, as it also keeps track of the changes being made to the code.

[1] It contains meta-data about the repository.

The project groups decided to use an open source hosting site, which will allow other developers to help expand the project, after the semester is over. It was decided to use Google Code, as it offers a wiki-site along with hosting the code repositories. Google Code offers hosting both through git, a distributed version control system, and Subversion, a server based version control system. Since most groups only had experience with Subversion, it was chosen for the repositories.

6.7. Android Versions and Devices

Since the most common version of Android at the beginning of this project was version 2.2, this version was decided as the target for development. Aalborg University provided a number of devices for development and testing, both phones and tablets. The different devices are:

HTC Hero Popular phone among early Android adaptors that has been supported and updated from version 1.6 to 2.1. The device does not support Android 2.2, and as such has provided some problems to work with.

HTC Desire HTC flagship Android model for version 2.1 that has been updated to version 2.2.

Samsung Galaxy Tab Tablet device, which include a GSM adapter, making it possible to use it as an ordinary phone. Runs Android version 2.2, and is for the most part the same as a phone running Android, but with a larger screen.

A number of other devices has been used by the groups involved in the project. Most notably, the Samsung Galaxy S was used by us for debugging and testing, as we were provided an HTC Hero.

CHAPTER 7
DESIGN

Some components of the system requires thorough planning before initiating implementation. The components in question is the GTP and the database used to store application meta-data. This chapter will describe the design of these components. Finally we will show that GTP can be verified using UPPAAL.

7.1. GirafPlace Application Storage

A server is hosting both the GirafPlaceServer and the GirafPlace Administration Panel. When uploading an application through the administration panel, the *.apk* file, the applications icon, as well as the meta-data for each application, will be stored on this server.

The application meta-data will be stored in a database. The database must be supported by the programming languages selected for the GirafPlace Administration Panel and the GirafPlaceServer - PHP and Java respectively.

The application meta-data stored is:

- Package name of an application.
- Version code and version name of an application.[1]
- Name of an application.
- Description of an application.
- Capabilities required of the user in order to use an application.
- Errors encountered while uploading an application.

[1] Android differs between the version code, which is an integer, and the version name, which is treated as a string. This makes it possible to use letters in versions, for instance '0.5b'.

The *.apk* and icon files stored on the server needs to be examined and unpacked during upload. Thus, it was decided to store the file directly on the server, instead of converting it to binary data and storing it in the database.

7.1.1. Database design

An ER-diagram of the required database were made in accordance with the requirements. The diagram can be seen in figure 7.1 on the facing page. Converting it to a database table overview gives the following tables:

- `admins`
- `applications`
- `errors`

Note that, as well as the attributes defined in the ER-diagram, an identification attribute is added to each table as a primary key. In addition, the tables use foreign keys to ease referral. In the ER-diagram, these are seen as a one-to-many relationship. The final database table layout can be seen in figure 7.2 on the next page.

7.2. GirafPlace Transfer Protocol

The GTP is designed to simplify data transfer between the GirafPlaceServer, and the potentially large number of devices running the GirafPlaceClient. Connecting directly to the database would require it to accept connections from any IP. This would provide an opening for hackers, causing a security problem. As it can be difficult for the client to test whether the layout of the database has changed since the client was last updated, a data integrity problem would also become prevalent.

A library, included on both the server and the client, ensures that the information stored in the database about an application can be transferred from the server to the client, even if the table layout has been changed. This library contains a class that contains only a single, constant integer, the `Version` class. This is used to ensure that both the server and the client include the same version of the library. Using

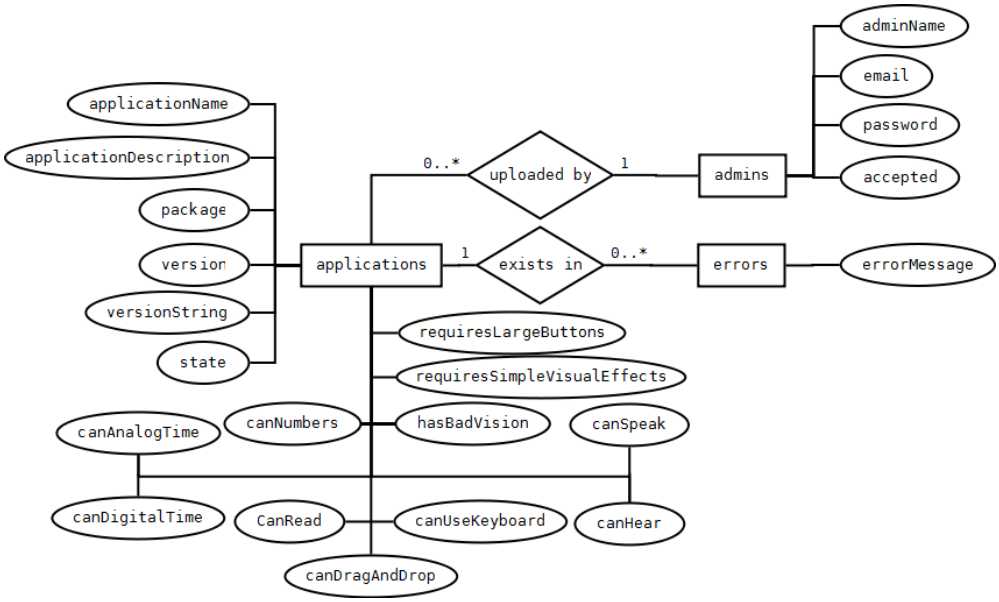

Figure 7.1.: The ER-diagram of the database.

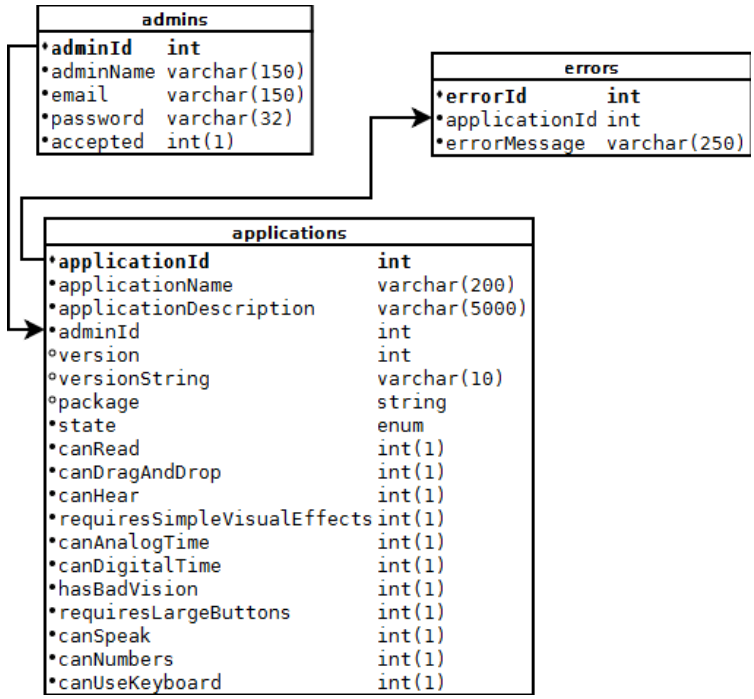

Figure 7.2.: The database table layout.

this, it is possible to gracefully inform the user that an upgrade of the GirafPlaceClient is needed, preventing a crash.

The protocol utilizes transferring of strings using the Java object streams, `ObjectInputStream` and `ObjectOutputStream`. The used strings are:

HANDSHAKE Written by the client when a connection is first made. This will make the server request the version of the library of the client, before allowing the client to make any other interactions, such as requesting applications.

VERSIONPLX Written by the server when a handshake is initiated, and the server is ready to receive the library version of the client.

WELCOME Written by the server when a version check is passed. After this has been written, the server will allow further interaction with the client.

WRONGVERSION Written by the server when a version check has failed. If the client receives this it should[2] inform the user to update the application, and stop the connection with the server.

GETAPPS Written by the client when it wants the list of applications available for download from the server. This is followed immediately by the client sending the user profile to make sure only applications usable by the user are presented.

SIGNOUT Written by the client when a connection should terminate.

KTHXBYE Written by the server as acknowledgement of the signout message. When the server writes this message, it does not expect further messages from the client, and closes the client thread.

As well as strings, the protocol also transfers the following objects:

Version An integer with the library version of the client.

Profile An object of the type `UserProfile` from the library. The client constructs this using the lazy loading functions from the administration library on the device.

Applications An `ArrayList<Application>`. The `Application` class is from the library. The list contains the applications that match the provided user profile.

[2] Whether or not the user is actually informed depends on client implementation.

7.2.1. GTP Verification

To ensure that it is not possible to trick the server to deliver applications without the version check, a UPPAAL model of the protocol has been created. UPPAAL is a model verification tool created in collaboration between the universities of Uppsala and Aalborg.[3]

The model contains a few abstractions, for instance the version check. The client starts by selecting either an integer representing the correct version or an integer representing a wrong version.

The global definitions of the model, such as the integers used to represent the object being transmitted, and the integers representing the in- and output streams are available in code example 7.1 on the following page.

The transition model for the server can be seen in figure 7.3 on page 43 and the model for the client can be seen in figure 7.4 on page 44. They are constructed from the implementation of the `ConnectionThread` class of the server, rather than just the description of GTP above, so they provide a good insight of the internals of the implementation.

Using the UPPAAL verification tool we determined that there is no way through the computational tree of the models, in which the server output will be APPLICATIONS while the **connected** variable is false. We also determined that the **connected** variable never will be true if the version of the client is wrong.

We can conclude that the model of the GTP only allows for retrieval of a list of applications if the version of the library used by the client is the same as the version of the library used by the server. This means that there will never be any problems sending the `ArrayList<Application>` object through the Java object streams[4].

These guarantees only holds for the implementation, if the model created with UPPAAL is a correct modeling of the implementation. It is, however, not possible to prove this, and as such the verification of the model can only be used as a proof of concept.

[3] Readers unfamiliar with this tool are encouraged to visit the website http://www.uppaal.com.

[4] At least not problems caused by mismatched classes.

```
1 // Place global declarations here.
2
3 bool connected = false;
4
5 const int WELCOME = 1;
6 const int HANDSHAKE = 2;
7 const int VERSIONPLX = 3;
8 const int OLDVERSION = 4;
9 const int WRONGVERSION = 5;
10 const int CORRECTVERSION = 6;
11 const int APPLICATIONS = 7;
12 const int PROFILE = 8;
13 const int KTHXBYE = 9;
14 const int SIGNOUT = 10;
15 const int GETAPPS = 11;
16
17
18 int serverIn;
19 int serverOut;
20
21 chan clientWrite;
22 chan serverWrite;
```

Code example 7.1: : The definitions for the GTP UPPAAL model.

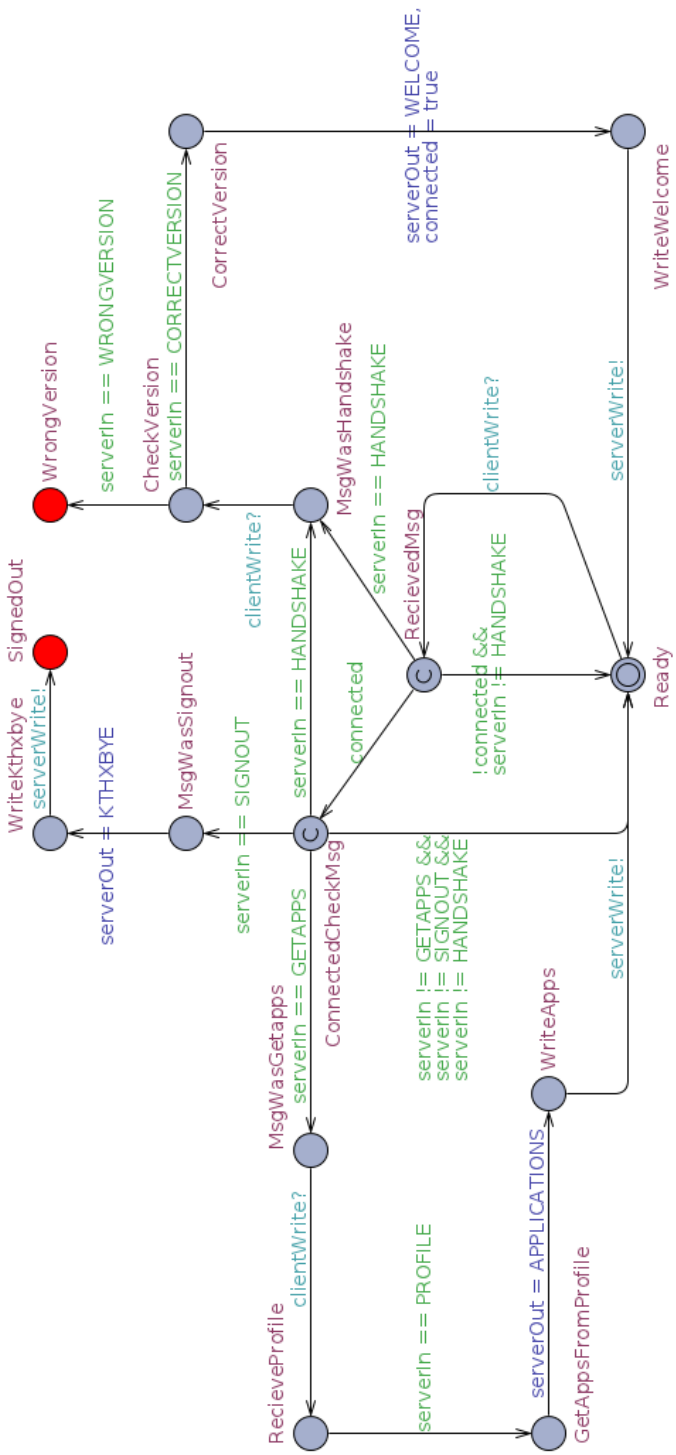

Figure 7.3.: The UPPAAL transition model for the server.

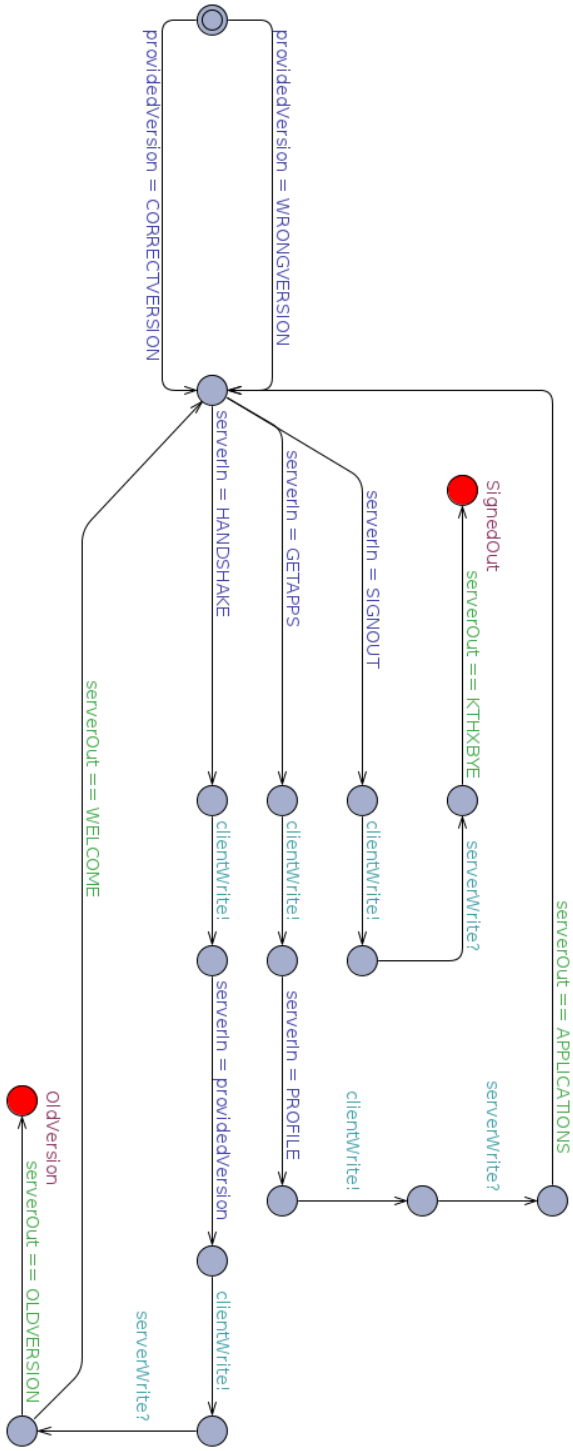

Figure 7.4.: The UPPAAL transition model for the client.

CHAPTER 8

IMPLEMENTATION

The design and implementation of our part of the complete system will be described in this chapter. We will do this by explaining the general responsibilities of each module. Interesting classes, methods etc. will be expanded upon and problems encountered during development will be described.

8.1. Architecture

The interactions between our components will be described here before we go into detail with the implementation of each component. In figure 8.1 on the next page, the architecture of the system is shown. Each component interaction is denoted by a number, which matches the number in parentheses after the title of the following sections.

GirafPlace Administration Panel and GirafPlace database (1)

From the GirafPlace Administration Panel, it is possible to upload and change application meta-data in the database. Meta-data can be retrieved from the database to the GirafPlace Administration Panel for display.

GirafPlace Administration Panel and GirafPlace file storage (2)

If the application administrator uploads a valid application, the *.apk* file is stored in the server file storage.

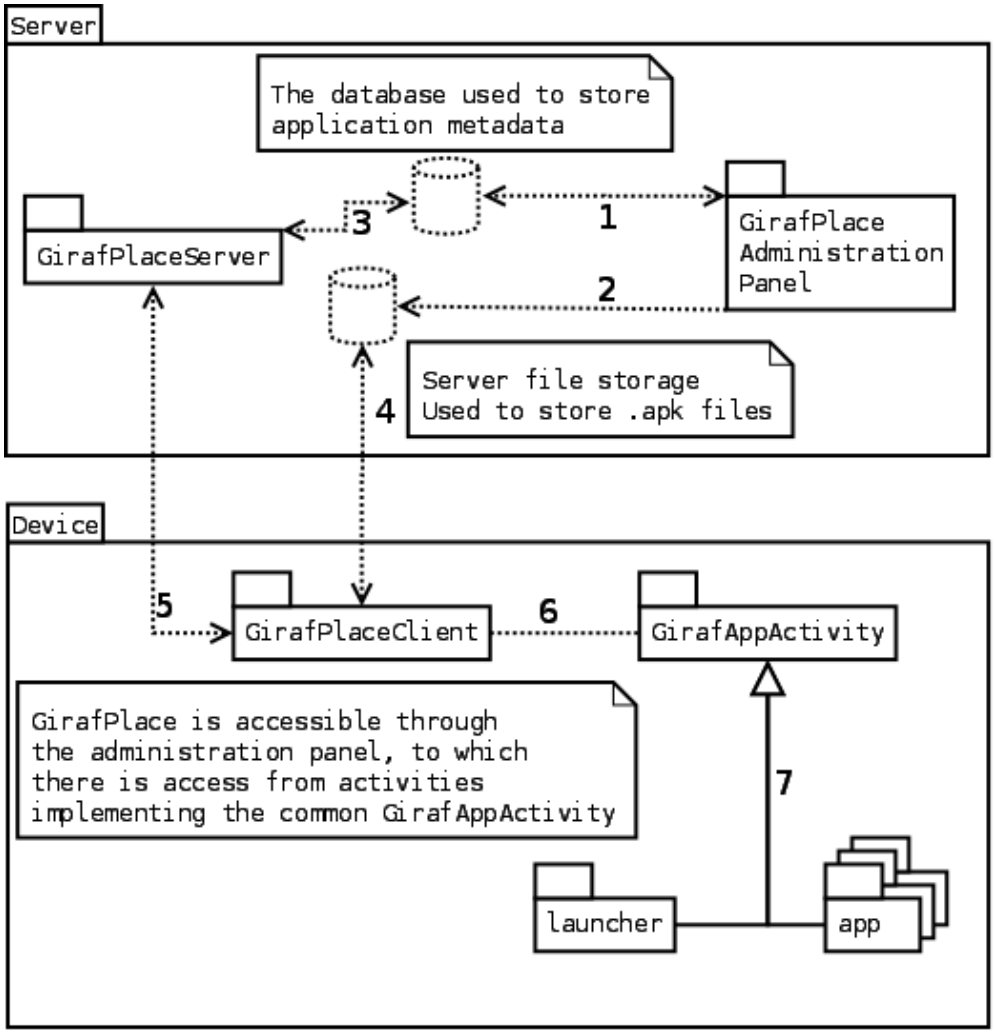

Figure 8.1.: The interaction between the components of our final system.

GirafPlaceServer and GirafPlace database (3)

The GirafPlace Server sends an SQL request to the database for applications matching the user profile provided to the server by the GirafPlaceClient[1]. The database returns the id of the application, its description and its name. The first is used to provide the link for the *.apk* file, should the user want to download it. The latter two are used to display information to the user about the application.

GirafPlaceClient and GirafPlace file storage (4)

The files stored on the GirafPlace file storage are downloaded by the client using HTTP. The files are stored in a subdirectory of the homepage called "installs", with the file name "[ID].apk", where "[ID]" is replaced with the id of the application.

GirafPlaceServer and GirafPlaceClient (5)

The interaction of these two components follows the GTP, as described in section 7.2 on page 38.

GirafPlaceClient and GirafAppActivities (6)

As described in figure 8.1 on the preceding page, these two components do not interact directly. First of all, the GirafAppActivities are abstract[2], and thus, requests made for the GirafPlaceClient stems from classes inheriting the GirafAppActivities. Furthermore, the request is indirect, as GirafAppActivities allows for opening of the administration module, which in turn allows for execution of the GirafPlaceClient.

GirafAppActivities and Launcher or applications (7)

The GirafAppActivities are inherited by the launcher and the applications of GIRAF. This makes it possible to open the administration module, by using the hardware key combination `Volume up - Volume down - Volume up - Volume down - Volume up - Volume down`.

[1] See interaction number 5.
[2] See interaction 7.

8.2. GirafAppActivities

In the analysis in section 5.3.2 on page 30, certain common functionalities were suggested. As well as suggesting this approach, the analysis also gave a list of requirements for the activities to be developed. The requirements were:

- Ignore all hardware key presses except Back, Home and End Call.
- Force all applications to execute in fullscreen mode.
- When the special key combination is pressed, enter the administration module.
- Allow application developers to change the activity started when using the special key combination.

8.2.1. Design

To enforce the required functionality, the existing Android `Activity` class should be extended. The new class should then implement the specified functionality. However, Android contains a number of classes that extend the `Activity` class, and therefore a special GIRAF version must be developed for each of them. We will only develop the classes needed by the other project groups, however, in order to save time.

To avoid redundancy in the extended classes, the functionality should, to as far an extent as is possible, be moved to a helper class, from which the functions can be called.

8.2.2. Implementation

The `GirafAppActivities` are implemented in the sw6.lib library, for easy access by all GIRAF applications. They include GIRAF versions of the `Activity`, `ListActivity` and `TabActivity` classes. The remaining standard `Activity` classes were not required by any of the application developers. The implemented classes are prefixed with `Giraf` to indicate they belong to the GIRAF system.

Most of the common functionality however, is not in these classes, but in a separate helper class called `ActivityFunction`. This class holds

the methods the `GirafActivity` classes calls when needed. It must be instantiated by all `GirafActivity` classes using it.

The disregarding of all key presses, as well as allowing access to the administration module, is implemented in the *onKeyDown* method of the `ActivityFunction` class. The method is called by the *onKeyDown* method of the `GirafActivity` classes.

The *onKeyDown* method overrides a method defined in the `Activity` class of Android. If the key combination is pressed, the parent activity is started. In most other cases, the keypress is ignored. Exceptions are described in the analysis in section 5.3.2 on page 30.

Application developers can redefine which intent to start when pressing the special key combination. This is done by calling the *setParentIntent* method of GirafActivity classes, which in turn calls *setParentIntent* method of the `ActivityFunction` class. This intent can also be opened by calling the *openParentInterface* method.

Enforcing fullscreen behavior on applications is implemented by making the *onCreate* method of the activity in question, call the *onCreate* method of the `ActivityFunction` class with the activity as argument.

8.3. Giraf Launcher

In the analysis in section 5.3.3 on page 31, the following list of requirements for the launcher was stated:

- List only applications that is part of the GIRAF system.
- List only applications that the user is able to use, according to the user's capabilities.
- List only applications if their usage is not limited by the current location or time.
- Only display application names if the user is able to read.

8.3.1. Design

The launcher must list applications available to the user according to two conditions: The settings defined in the user profile, and the cur-

rent location. These applications must be shown as a list of tappable icons, which, when tapped, opens the illustrated application.

The launcher should provide application titles below the icons only if the user is able to read. Rendering text to a child who cannot read is, in best cases, a wasted effort, and in worst case a cause of confusion for the child.

The location filtering must use an SSID blacklisting system. The idea is to allow a guardian to decide if an application should, for instance, not be used when the child is at school. Therefore the administration panel must provide access to an SSID blacklist for each application.

As the launcher is required to provide access to the administration module, the launcher must implement the `GirafActivity` class.

8.3.2. Implementation details

The GirafLauncher component consists of several classes. The `launcher` class itself is an activity, which extends the `GirafActivity` class as described above.

The `launcher` class contains an instance of the `GridView` class, which requires an adapter, in this case an instance of the `IconAdapter` class, to display applications as icons for the user to tap. The icons are represented by the `Icon` class.

Finally, GirafLauncher uses a content provider from the `ApplicationInfoProvider` class, to access application information from the database. A helper class, providing table information, `TableInfo`, is used by the `ApplicationInfoProvider` class to define the tables in the underlying database.

8.3.3. Launcher

The GirafLauncher is the home application of the GIRAF system. That means that at any time during execution, an instance of the launcher will be running in the background[3], and a click on the Home key will take the user back to it.

[3] In reality the Android system might terminate the launcher when in the background, to free up resources.

The standard Android behaviour for the Back key will take the user to the previous activity, however, pressing the Back key in the launcher should not do anything. Therefore the *onKeyPress* method has been overridden in the `launcher` class to ignore the Back key being pressed.

Aside from the Back key, the launcher must also provide access to the administration module, and block use of most other keys. This functionality is most easily implemented by have the `launcher` class extend the `GirafActivity` class where that functionality is already implemented.

Finally, the launcher defines the action to be taken when an application is clicked, with an *onItemClickListener* method. The action is to launch the default intent for the application package name. This intent is acquired using the *getIntentForPackage* method of the Android `PackageManager` class. This has the limitation that if a package has no main intent, the user will be presented with an error message, and no application will be launched.

8.3.4. IconAdapter

The `IconAdapter` class is an extension of an adapter, Androids implementation of a list interface. The `IconAdapter` class is implemented using the singleton design pattern. This means, only one instance of the class can exist. This is enforced by making it impossible to instantiate the class, and instead put the instantiation in a static method called *getInstance*. The class instantiation is, however, only called if the class is not already instantiated. In all cases, the instantiation of the class is returned.

The `IconAdapter` class also uses an asynchronous task for preparing the list of applications. The alternative to this was to retrieve applications directly in the UI thread, which would disallow update of the interface causing the UI to be unresponsive. By making the application retrieval an asynchronous task, the user can be notified that the application list is being loaded, and provide him the option to cancel it.

The `IconAdapter` class is responsible for filtering out applications that should not be displayed in the launcher, be it because the user cannot or is not allowed to use it. There are two kinds of filtering in the GirafLauncher:

Abilities The abilities defined in the user profile can make some applications unusable for a user. If a user for instance cannot read, an application requring that the user can read, should not be shown in the launcher. Applications are filtered in the GirafPlaceClient, but since a user profile may change after an application has been installed, the filtering takes place in the launcher as well.

Location Since location filtering uses SSIDs, the launcher requires the device to have wifi. The only device where this has proven a challenge has been the Android emulator. Therefore the launcher assumes that if there is no wifi device available, then it must be executing on the emulator. If this is the case a debug failsafe kicks in and makes the launcher believe that it is in proximity only to a single network, the "AAU-1x" network. It is important to notice here that the SSIDs on the blacklist are not the networks that the device is connected to, but rather any network in proximity that broadcasts its SSID.

8.3.5. Icon

The `Icon` class extends the Android class `View`, and is the class that is used to display the icon of an application. It allows for changing the application icon, as well as the application title. It is responsible for testing whether a user is capable of reading, and adding a label below the icon only if the result is true.

8.3.6. ApplicationInfoProvider

The `ApplicationInfoProvider` class is an extension of the Android `ContentProvider` class and is used to provide application info. This data is being accessed by the `Icon` and `IconAdapter` classes. The `ApplicationInfoProvider` class accesses an underlying SQLite database.

The database holds application meta-data, used by the `IconAdapter` class. The data is inserted when an application is installed, and removed when an application is deleted. The meta-data is the package name and the `Application` object containing information on, among other things, the required abilities and the SSID blacklist, represented as a list of strings.

The information stored in the `ApplicationInfoProvider` content provider should ideally have been stored in the database administered by the administration group, but the need for storing the data was discovered too late, and instead we decided to setup our own content provider.

Since this would have to be changed in a future release, we decided to omit the time constraint functionality. We did not deem it reasonable to use time developing the database backend needed, as the implementation6 would resemble that of the SSID blacklisting. The SSID blacklisting therefore exists as a proof of concept of how application filtering should be implemented.

8.4. Database

As described in section 7.1 on page 37, a database was required to hold application meta-data. The final implementation of the database was done in MySQL, as a web server already using MySQL was available for storing the programs and scripts required for the project.

Both PHP and Java, the two programming languages used in this project, has drivers for connecting to MySQL databases, which made it an easy choice.

8.5. The GirafPlace Administration Panel

As part of the distribution platform, we found that there was a need for developing a web page to add applications to the GIRAF system.

8.5.1. Design

The administration panel will save application meta-data during upload. The data consists of the package name, the application version, the application name as well as the description of the application.

The package name is used by the Android package manager to determine, whether an application is already installed on the system. If so, the application version will be used to check whether or not the new application is an update to the existing application.

If it is not, the package will be rejected when the user attempts to install it, otherwise the application will be updated on the device. The application name and description will be used whenever a user browses GirafPlace for applications.

The required functionality for the administration panel is listed below:

- Create an application administrator.
- Upload an application.
- Display errors encountered when adding an application.
- Display applications uploaded by the administrator.
- Edit meta-data of uploaded applications.
- Change administrator information and settings.

8.5.2. Implementation details

The website, being the only part of the system not used by GIRAF clients, was decided to be implemented as a proof of concept only. This means, that the solution presented is not well-considered, but merely implements the basic functionality needed.

The web page was implemented using PHP[4] since we had access to an Apache server supporting PHP. Some group members knew PHP already, and could, in a short amount of time, develop the homepage.

Create an application administrator

The creation of an application administrator resembles the user administration process at some other web pages. It is required for a new user to fill out his email, name, and password. These informations are, if entered correctly, stored in the database. After the application developer has been added to the database, page administrators must verify the signup.

[4] *PHP: Hypertext preprocessor* - a scripting language for web development.

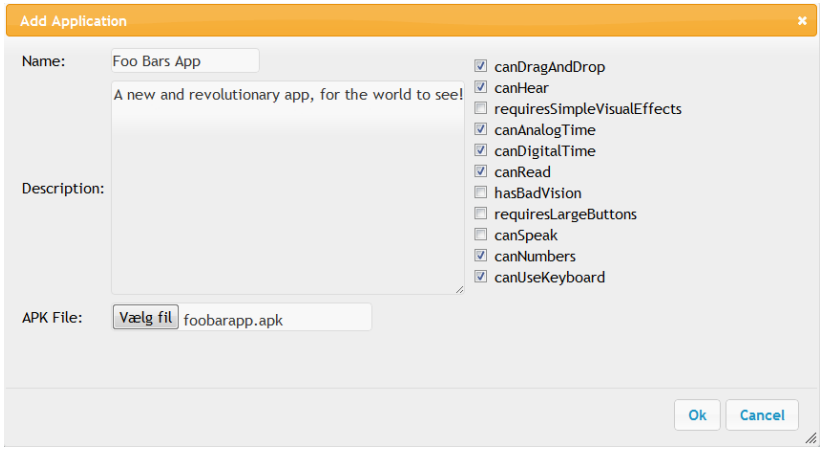

Figure 8.2.: A new application being uploaded to GirafPlace.

Upload an application

An application is uploaded to GirafPlace by clicking on ”Add application” which opens a dialog, which can be seen in figure 8.2. This dialog requires the application developer to enter information about the application. The application meta-data that must be provided is:

- Name of the application.
- Description of the application.
- The file to be uploaded.
- The capabilities the application requires the user to have.

The script retrieves further meta-data from the supplied *.apk* file, and ensures that the *settings.xml* file is present in the *.apk* file and that it is valid.

Display errors encountered when adding applications.

If an application contains an error, application meta-data is still added to GirafPlace. The application is, however, not visible from the GirafPlaceClient, until the error have been fixed.

Display applications uploaded by the administrator

The main page of the administration panel lists all applications uploaded by the currently active administrator. Also, should an uploaded

application contain errors, a button, which opens the error dialog, is displayed in the application list.

Edit meta-data of uploaded applications

The list of applications also allows for editing meta-data of uploaded applications. This is done by selecting an application, editing the meta-data required, and then uploading the changes. The interface for this functionality resembles the interface used for adding applications, though it does not have the opportunity to add a new *.apk* file.

If a developer wishes to upload a new version of his application, he must instead upload it as if it was a new application. An uploaded application with the same package name as a previously uploaded application, and a newer version, will replace the old application as the active one of that package name.

Change administrator information

To allow application administrators at least a mild form of security, they are allowed to change their email, name and password if wanted. This can be done by pressing *Edit info*.

8.6. GirafplaceServer

The GirafPlaceServer is a server program that resides on the GirafPlace web server. According to the analysis in section 5.3.4 on page 31, it must use the GTP to accept requests from clients for application lists, and return these lists to the clients. The list of applications should only contain applications the child using the device is capable of handling.

8.6.1. Design

The GirafPlaceServer will implement the server side of the GTP and offer access to GIRAF applications for clients. Since we want to employ the Java in- and output streams, both the client and server must be

implemented in Java. To ensure that a single client does not block access to the server for other clients, the server will be multi-threaded, allowing each connected client server processor time.

8.6.2. Implementation details

The server consists of four classes, of which only three are of interest as the `Main` class merely creates an instance of the `NetworkThread` class, and starts it. The `NetworkThread` class listens for new connections, and starts a new instance of the `ConnectionThread` class for each new connection. The last class, the `SQLManager` class, is used to connect to the database and extract the applications a client requests.

The server logs to STDOUT, meaning, that if it is desired, the output can be piped into a file. It is currently being executed using the Linux application "screen". This means that the output is not stored during server reboots, but it is not deemed a problem at this point, as the server load is very low.

The NetworkThread class

The task of the `NetworkThread` class is to intercept incoming connections and assign each client its own thread to handle the connection. Instantiating a new instance of the class makes the server listen to the assigned port, as well as writing to the server log.

The class runs a while(true) loop, listening for new connections, using the *ServerSocket.accept()* method. When the method returns, a new instance of the `ConnectionThread` class is started, in order to handle connection with the newly connected client. To make it easier to dissociate the individual threads in the output, the `NetworkThread` instance keeps a running integer that identify the threads. This number is incremented by one, every time a new instance of the `ConnectionThread` class is started, and is reset to 0 when it reaches MAX_INT.

The ConnectionThread class

The `ConnectionThread` class implements the GTP protocol. This means running a while loop, waiting for a string to be read from the input stream. Then, it matches said string up against the protocol

and performs the action defined therein. When the server is asked to transfer the application list, by a client writing "GETAPPS", it starts by reading a `UserProfile` object from the inputstream, then using the `SQLManager` class to get a list of applications matching that user profile.

The `ConnectionThread` class uses a convenience function for outputting debug information. Whenever the server uses this function, it will add information to the output regarding which thread number the current thread was assigned, the value of the **connected** variable, and how many threads are active at that instance. The function can be seen in code example 8.1.

```
1 protected void output(String output) {
2     System.out.println(String.format(
3             "[%1$d] %2$s. The connection variable is: %3$b,
4             and the number of active threads is: %4$d",
5             threadNumber, output, connected, Thread.
                  activeCount()));
6 }
```

Code example 8.1: : The *output* method of the `ConnectionThread` class.

The while loop ends in two cases: Either the "SIGNOUT" string is sent from the client, or there is some exception. Exceptions are written to STDERR, before the thread returns. Unlike normal debugging output, the exceptions are not written using the *output* method, and as such they do not have the thread number available as debugging information. This is because we use the *printStackTrace()* method, to output the information of exceptions.

The SQLManager class

The `SQLManager` class is used to connect to the MySQL database containing information on which applications are available, and which capabilities they require from the users. It has only a single public method, *getApplications*, which takes a `UserProfile` object as its only argument. Using this profile, it constructs a query to select all applications from the database that are live, and has capabilities matching the user profile.

Matching attributes in this case, means that the capabilities the user does not have, cannot be required by the application.

8.7. The GirafPlaceClient

The requirements set for the GirafPlaceClient in the analysis states that it must use the GTP to allow guardians to install, uninstall and update applications. The GirafPlaceClient should be accessible through the administration module of GIRAF, and package installation should be handled by the Android package manager.

8.7.1. Design

The GirafPlaceClient will be an Android application accessible from the administration module. The client must, using the GTP, be able to retrieve application meta-data from the GirafPlaceServer. These applications must be displayed to the user, who will then be able to install applications on the device, thus making it accessible for the child through the launcher.

The retrieved applications will be displayed to the user as a list. When tapping an item, it will be possible to install it, if it is not already installed. If an older version of the selected application is installed, it will be possible to upgrade to the newer version. If the user already has the newest version installed, it will instead be possible to uninstall the application.

Note that this means, that it is not possible to uninstall applications that are not up to date through the GirafPlaceClient. This is unfortunate, but it is still possible to uninstall them through the Android package manager available through the Android settings in the administration module.

As GirafPlace might contain many applications, retrieval of all applications from the server could potentially take rather a long time. The client will therefore let the user know that applications are currently being downloaded via a loading screen.

8.7.2. Implementation details

The user interface of the GirafPlaceClient consists of an instance of the `ListView` class. Each item in the list is tappable and tapping one will prompt an application view for the individual application. Here, one can see detailed information about the application. In the development version, this includes an application icon, a description, a name. Also, a multi-purpose button for installation, update and uninstallation is available. Future releases could feature ratings and reviews, version logs and so forth as well.

The GirafPlaceClient consists of five classes:

- `AppdetailsActivity`
- `ApplicationAdapter`
- `AppView`
- `MainActivity`
- `NetworkTask`

AppdetailsActivity

This class extends from the `Activity` class. Its functionality is to show the details of an application, in GirafPlace. These details are the name, description and the state, meaning whether it is installed or not, of the application. If the application is not installed, an install button is displayed. If there is an update for the application, and it is currently installed, an update button is displayed. Finally if the application is installed on the device, without an update being available, an uninstall button is displayed.

The `AppdetailsActivity` class also contains a private class that extends the `AsyncTask` class. This private class handles downloads of *.apk* files from the server, and updates the UI with progress information. This task downloads to the ”Download” folder on the device. The actual location of this folder can change, but Android provides an abstraction that solves this issue. However, on most devices it is stored on the SD card. If no SD card is present on the device, the download will instead be to the root of the file system. This presents a problem, as the Android activity that handles installation does not have access to the root, and a permission error will prevent installation.

Should the user choose to install an application, a broadcast is sent to the Android system, to inform of this. This is done so that the built-in package manager will take care of installing the application. The same is true, when an application is uninstalled.

ApplicationAdapter

As the `MainActivity` class extends the `ListActivity` class it needs a `ListAdapter` implementation to handle the views to be listed, in this case the `ApplicationAdapter` class. It keeps track of the list of application meta-data downloaded with the `NetworkTask` class, and provides the functions needed by the `MainActivity` class in order to display them.

AppView

The `AppView` class extends the `View` class used to display the applications in the `MainActivity` class. It consists of an `ImageView` instance and a `TextView` instance side by side, and an `AsyncTask` instance to download the icon and display it in the `ImageView` instance.

MainActivity

The `MainActivity` class shows the main view of the GirafPlaceClient. Here, all the applications available from the GirafPlaceServer are displayed. It contains code that ensures when a application icon or text is tapped, an `AppDetailsActivity` activity is displayed for that application.

The `MainActivity` class is also responsible for starting the `NetworkTask` class that downloads the list of applications from the server. It does this in the *onResume* method of the class. This means that it will update the list every time the user returns to the list. This helps to keep the list updated. An update can also be forced via the options menu.

NetworkTask

Using the GTP, the `NetworkTask` class is responsible for creating a connection to the server, and retrieve the applications from the server that matches the current user profile.

During the development of the class we found two problems with the network implementation in Android. First of all, a socket connection that fails, does not result in an exception. To counter this problem we attempted to use the *InetAddress.isReachable* method to perform a check before connecting the socket. This function provided the second problem however. It turned out to be extremely unreliable, and continually provided wrong results. Therefore we implemented our own method for checking reachability using an HTTP connection.

The function can be seen in code example 8.2. If the connection returns 200[5], then the function returns true.

```
1  public Boolean isConnected(String host){
2      try {
3          URL url = new URL("http://"+host);
4          HttpURLConnection urlc = (HttpURLConnection) url.
               openConnection();
5          urlc.setRequestProperty("Connection", "close");
6          urlc.setConnectTimeout(1000 * 30); // mTimeout is in
               seconds
7          urlc.connect();
8          if (urlc.getResponseCode() == 200) {
9              Log.d(tag,"getResponseCode == 200");
10             return new Boolean(true);
11         }
12     } catch (MalformedURLException e1) {
13         e1.printStackTrace();
14     } catch (IOException e) {
15         e.printStackTrace();
16     }
17     return false;
```

Code example 8.2: : The *isConnected* method that tests whether a connection to the server, can be established.

[5]The HTTP status code meaning that the request was fulfilled, (OK).

CHAPTER 9
TESTING

This chapter describes how we chose the test approaches that we used. After that, each test will be described and the results of the test will be discussed. The actual test documents are placed in appendix B on page 81.

9.1. General Test Information

Note that our tests of the server, launcher and girafAppActivities have been omitted. This is due to the fact that the other tests cover their purpose. The test of the GTP is mostly concerned with the server, and the multi-project test covers girafAppActivities, as well as the launcher.

Another detail to notice is that the test documents concerning our own components, are not of the latest version. All the final test runs we performed passed, thus, they would not give as much insight in the work flow of the project, as the earlier, failing ones, would.

9.2. Comparing the test approaches

The theory in chapter 3 on page 19 describes several different approaches to testing ones software.

An analysis of the pros and cons of these approaches, seen in table 9.1 on the following page, has lead us to use the following test approaches as described:

Static white box testing During development, we have conducted reviews of the developed code. We reviewed in an informal way, and have not documented the review sessions, as the review sessions were lengthy discussions during the development phase.

Dynamic black box testing We have used the developed software to ensure it behaved as specified. This testing has been structured, as the actions to be performed were specified, along with the expected outcome. These have been compiled to several test documents, which have been used as foundation for iteratively conducted tests throughout the project.

Approach	Pros	Cons
SBB	- No code involved (Only looks at specification) - Detects missing error handling	- Requires complete system definition - Requirements may change
DBB	- Create specific cases - Easy to execute - Use the system as end-user	- Hard to cover entire system - Time consuming
SWB	- Not only bugs are found (troublesome areas also found) - Looks at code only	- Inclined to fix bugs on the go - Incline to forget ”obvious” bugs
DWB	- Similar to debugging (easy to use)	- Similar to debugging (inclined to fix bugs on the go)

Table 9.1.: Pros and cons of testing methods. Used abbreviations: *SBB*: Static black box testing, *DBB*: Dynamic black box testing, *SWB*: Static white box testing, *DWB*: Dynamic white box testing.

9.3. GirafPlace Administration Panel

As the GirafPlace Administration Panel has only been developed as a proof of concept, the quality of this component cannot be expected to be as high as other components. However, the panel is still part of the system, and it is therefore necessary to ensure that it does indeed work as intended. Therefore, the basic functionalities must be tested.

The final test document for testing the GirafPlace Administration Panel can be seen in appendix B.1 on page 81.

9.3.1. Xml validator update missing

At the time of testing, test case 9 were skipped, and test case 8 was completed using an old version of *sw6.xmlvalidator*. Test cases similar to these were also tested in the multi-project test, where the use of the validator failed.

The error was first of all caused by, as stated, an old version of the xmlvalidator, and secondly, an erroneous setup of file permissions and an unresolved error in the relative file path. The latter was fixed by changing the file path to use the absolute path. New tests were performed and the test case 8 and 9 now passed.

9.4. GirafPlace Transfer Protocol testing

As well as verifying GTP using UPPAAL, in section 7.2.1 on page 41, we also wanted to ensure, the implementation of GTP actually followed the protocol. This was part of a hand-in in the project course Test and Verification. As this hand-in had a deadline long before the project, the tested version is an old version of GTP.

The test report can be seen in appendix B.2 on page 84.

9.4.1. Missing update of connection variable

Testing GTP resulted in one of the test cases, test case 2, not passing. In particular, the problem was that the **connection** variable for the client was not set to false as intended. This only meant that the connection status was not updated properly as the connection was closed beforehand.

This was only seen as a minor inconvenience, as the server closed the thread immediately after the connection was closed. However, the issue was fixed in a later version of GTP, which also included improved functionality, as described in the implementation in section 8.6 on page 56 and 8.7 on page 59.

9.5. Test of GirafPlaceClient

To make sure that the GirafPlaceClient does not crash at inconvenient times, it is important to test the client to fail. That is, executing test cases designed specifically to crash the client. Thus, we have designed the test cases described in the test document in appendix B.3 on page 88. The test cases can be divided into three major areas, where we deemed the GirafPlaceClient would be prone to errors. The issues encountered, as well as whether they have been fixed, are described in the following subsections.

9.5.1. Using the Back key while updating application list

Pressing the Back key should, at any point, close down the currently executing activity. It could, however, prove a problem being allowed to do so while retrieving applications for the client. And, as can be seen in test case 3, it is.

Pressing the Back key will stop the update process if it is currently running. Doing this when the client is first started, or any other time where the list is still not populated, will result in an empty list shown on screen. The system can still function, and one can simply ask to restart the update process in order to finish the update. However, it is not intuitive to the average user, as one would expect the Back key to return one to the previous activity.

One could argue that the previous activity is indeed the empty list of the client, but that is not what the average user would presume, as the update activity executes in the same screen as the client. Instead, one would expect the client to close and return to the activity from which the GirafPlaceClient was launched[1]. A correction has been made to ensure that this happens.

9.5.2. Rotating the device

Rotating the device has been a cause of problems for all groups involved in the project. The cause of this is that the life cycle of an application is restarted on rotation. Only if the current instance had

[1]We did not perform any user analysis to come to this result, however, the other groups commented on this issue the very first time they used the client.

been saved before rotating, could it be possible to avoid starting the activity from scratch. Testing whether rotation causes errors are executed in test cases 4 to 7.

Restarting the life cycle causes the application to redraw the interface, and icons for the application list are retrieved as part of the interface. Thus, rotating the device causes application icons to be redownloaded. This is a minor inconvenience, but should perhaps be fixed to reduce bandwidth usage.

Another issue, not covered by the test, is that rotating the device while downloading an application, removes the download progress bar. The Install button is also available again, allowing the user to download the application once more. This happens because the download is executing as an asynchronous task, thus not allowing the GirafPlaceClient to keep track of it.

The first download is not stopped, and one can thus have several simultaneous downloads running, while only being able to view the progress of the latest. Once the first download is finished, the package manager will prompt the user to install the package. Installing the package will take priority, and thus, no other prompt will be displayed when subsequent downloads finish.

After installation, however, the package manager will prompt the user for the installation of the next download, until no further downloads remain. This is a major problem, as the user have no chance of knowing what is currently going on, as they are continuously prompted by the package manager.

Since fixing this issue requires redesigning the entire `AppDetailsActivity` class, we have decided not to fix this issue during the course of this project, as it was discovered too late.

9.5.3. Missing Internet connection

An issue not considered, while developing the GirafPlaceClient, was whether the phone actually was connected to the Internet. Therefore, we decided to test, whether a missing Internet connection would pose a problem. This is tested in test cases 8 to 11.

Entering the GirafPlaceClient while not connected to the Internet, unfortunately causes the client to simply wait for a response from the server. Since this response is not coming, at some point, the user will

cancel the activity by pressing the Back key. This causes the client to crash, as the cancellation does not, correctly, stop the update of the application list.

What happens is that the client hangs when trying to open a socket to connect to the server. Since the server never acknowledges this socket the client will simply wait for it. Cancelling this task causes a failure due to the way sockets are implemented in Android, as mentioned in section 8.7.2 on page 62, where the fix is also described.

The user experience of the client is severely lowered by this, as no error message is ever given to the user as to why the client crashed.

9.6. Integration testing of multiproject

As well as testing our own components to ensure their inner functionality meets the requirements set forth, the entire project has been tested to determine if the components work well together too.

These tests were conducted in cooperation with the other project groups. The test cases were devised amongst representatives of each group, and after a review session with the rest of their members, the tests were performed.

The tests performed in these test cases are intentionally not testing the individual component functionality rigorously. As it is described in the test document, the tests were performed on the assumption that the groups had tested their own components before the joint tests. These multi-project tests thus only tested the interaction between components.

The test report can be seen in B.4 on page 90. It is composed of several test documents, as opposed to the approach used in our previous tests. The most prevalent difference is that this test document only describes the expected behavior in the test results, not in the procedure as we have previously done.

The tests are named Test Document 00 to 07, abbreviated TD00 to TD07. All tests except TD00[2] were executed on four different devices, using different GIRAF applications when needed.

[2]As this did not concern the parts of GIRAF used on the devices.

9.6.1. Reported issues

The issues reported after executing the multi-project test concerning our components are listed here, along with the changes needed to resolve each issue, and whether the solution was implemented or not.

Invalid settings.xml applications are accepted by GirafPlace

In TD00 it was reported that applications with an invalid *settings.xml* file are accepted, when uploaded via the GirafPlace Administration Panel. This issue is described previously in section 9.3 on page 64.

Context menu icon of GirafPlace is not scaled appropriately

In TD02 it was reported that on the HTC Wildfire, icons were inappropriately scaled. The cause of this issue is unknown, and the issue is not reproducable on other devices. The issue was therefore not fixed.

Application list issues

In TD03 several issues with the list of applications in the GirafPlace-Client were reported. One issue was that the client did not update the list of applications again, if an application was updated at GirafPlace after the client had retrieved the list of applications originally. This would cause the client to try to access the old application if the user clicked on the specific application in the list, causing a failure.

This issue was not fixed, as it very rarely occurs. The issue could be fixed by redesigning the upload procedure of the GirafPlace Administration Panel. Since this would most likely be necessary for any further development of GirafPlace, the problem would not be present in any actual deployment of the system. However, at the time the issue was found, we did not have time to reimplement the procedure.

Launcher displays two application icons

In TD03 it was discovered that if an application was updated on the device, the launcher displayed it twice in the home screen. This was due to the system not removing the old application from the list of

displayable applications when updating to the new application. This was fixed in a later build, which also fixed a known issue with the SSID blacklist being reset when updating an application.

CHAPTER 10
CONCLUSION

10.1. Future work

Several minor issues could be fixed to give an improved user experience when using not only the launcher, but also GirafPlace. Most of these fixes are trivial, or otherwise uninteresting, thus we will focus on a few key issues we believe to be the most important aspects of further development of the system. After finalizing development, requirements for the device to be used could also be set.

10.1.1. Saving application capability requirements

Currently, the capabilities required of a user is entered by the application administrator when uploading an application. This requires the application administrator to insert capabilities at each application update. Instead, the data should be saved in the *settings.xml* for each application. This would make the most sense, since it would be saved in the same database as the rest of the application-specific data.

10.1.2. Ratings

A feature we discussed adding to the marketplace, was the possibility to give ratings to applications. This would require many additions to the system, and thus we decided, as the GirafPlace is a proof of concept, that we did not have sufficient time to implement it properly. In order to give ratings, the system would have to make sure that a user could only vote once, or at least only once per device. This would require some sort of login system for the device, that should run in the background, to not allow the child to interact with it. It

would require new database tables to handle the new users and the relations between them and the rest of the system.

The rating functionality should most likely be a part of the administration module, in order to not confuse the child by the text interface for the rating. This would also allow us to let the user write a review along with the rating, as it would be the guardian performing the action. These reviews would be accessible through the store, for other users.

10.1.3. Convenience views

The developed system is in many ways a proof of concept. Most interfaces, for instance, were developed to accommodate the needs of the current features and to be easy to use. The GirafPlace user interface, for instance, is sufficient for a marketplace with a limited amount of applications.

If more than a couple of hundred applications were present in GirafPlace, it would quickly become annoying to browse through them all, to find the applications wanted.

To fix this, several convenience views should be added to GirafPlace. The most obvious would be a search function to only view the applications with a given substring in either their name or description. But it would be equally important to divide applications into categories, and let a user view only the categories wanted. In accordance with the ratings addition, it would also be feasible to add a list of popular applications, and maybe even add a monthly "*hot apps*" view.

These views would allow the user to navigate through GirafPlace easier, and would also provide the user with easier access to applications.

10.1.4. GirafPlace Administration Panel

The current implementation of the GirafPlace Administration Panel is a proof of concept only. The implementation needs to be completely rewritten, should the system be deployed for actual usage. Most important to consider is that the administration panel contains several security holes, deliberately ignored during development. Therefore these would need to be fixed.

As well as fixing security errors, some of the implemented functionality of the administration panel is not working as it is ought to. For instance, the current update functionality would be prudent to fix. It should be possible to simply update your application, instead of uploading a new application with the same package name.

10.1.5. Improved documentation for usage of the system

Even though our components of the system are documented properly, the overall system does lack some general documentation on how to use the system. This documentation should be targeted guardians of the disabled children. The documentation should be written, or at least reviewed, by all involved groups, so that the used components are described properly.

As well as ensuring the overall system usage documentation, it must also be ensured that the application developers are provided with a proper documentation for developing GIRAF applications and adding them to the system. This documentation cannot be finished before the administration panel has been redesigned.

10.1.6. Re-testing the multiproject

Since the integration test was only conducted once, it is suggested that we should conduct the test again after fixing the encountered issues. Also, it is suggested that additional tests are designed to ensure applications follows the Android lifecycle, and other design guidelines.

10.1.7. Entering the adminstration module

The current implementation allows a user to access the administration module by using the special key combination. This combination was only intended to be used for development, and no deeper thought has been put into its security, nor user friendliness, apart from it not being too easy to press accidentally.

Another group reported that during their user test of their application late in the project, one of their testers was very close to entering the module accidentally. This would have to be fixed before releasing the system, possibly by not using the volume keys after all.

10.1.8. Device requirements

When deploying the system, it would be suggested to do so on a device developed specifically for the system. The requirements for this device could be that all hardware keys except the Home and Back key are removed. Also, as the device is to be used by children, it should be made more durable than an ordinary device.

10.2. Final Conclusion

During this project, we have developed a custom launcher for the Android platform that allows dynamic filtering of applications in regards to user capabilities and the location of the device. We have also supplied the system with a custom marketplace that performs filtering of applications in regards to user capabilities. The marketplace allows the guardian of a user to download and install applications for the system.

During the project, we worked together with other groups from the university in order to develop a larger system. This has been a great learning experience for us, as we never thought it would create that many problems along the way. We spent a long time this semester figuring out how to work with the other groups. For instance, performing things, such as an analysis, took a lot longer as the different parties envisioned and valued aspects of the system completely different.

The largest problems were naturally the communication between the groups. It suffered mostly because every group already had their own way of working, not interacting gracefully with each other.

The biggest negative impact on the final product stems from the unordered development cycle used among the groups. Had this been an industrial project, the administration module and launcher would have been designed, and maybe partly implemented, before design started on further modules, and missing features in the administration and launcher module could be implemented afterwards. This was not possible here due to timing constraints.

Thus, the application groups often found themselves in need of features that neither we, nor the administration group, had time to develop. Adding to that, both we and the administration group, experienced developing features that none of the other groups needed. This

was all due to the fact that the results from the analysis and design of some groups were needed by the other groups in order to finalize their design.

Since the requirements for our part of the system were initially intended to be decided by what the other groups required, we did not see the need of a thorough analysis. This, unfortunately, proved an issue for us, as the requirements set by the other groups were provided to us at a very late stage of the project. The main cause of this was a lack of a deadline for the requirements, which is very important to note, should a similar project be initiated.

Other than that, we have had few issues with the other groups. We structured the work so that we could work as independently from each other as possible, in order to not interfere with the inner workings of each group.

We have learned the importance of agreeing on the vision of the product, even before analyzing in-depth. It would have been much more reasonable of us to finalize the product requirements before anyone were allowed to work further with the development. We could have saved rather much time in all the project groups, had we done that. This is an experience valuable for later projects.

APPENDIX A

FACTORS OF THE OVERALL SYSTEM

A.1. FACTORS

In cooperation with the other groups involved in the project, we made a FACTORS analysation of the overall system. Based on this, the actual responsibility of each subsystem could be defined. The final FACTORS is as follows[1]:

Functionality *Describes the functions of the system that support the application domain tasks. That is, defining what the system is able to do.*
The system should offer installation of new applications and make it possible to administrate common settings by need. The system should mask the normal functionalities of the unit to the user. Further, the system should give the opportunity to control the usage of and access to system- and user profile settings as well as applications according to the current time, and location of the unit. The system should be delivered with a number of pre-installed applications which is customizable to the user.

Application Domain *Concerns those parts of an organization which administrate, monitor, or control a problem domain.*
Children with limited mental capabilities due to handicap or age, making it hard for them to handle the complexity of a normal smart-phone or tablet OS. Parents and kindergarten teachers (guardians) will be in charge of administrating the system.

Conditions *Covers conditions under which the system will be developed and used.*

[1]Note that we did not write the rest of the content of this appendix ourselves, as it is a collaborated work. We have changed some of the content during the project, and as thus, this is merely submitted as it was our source.

The project is being developed by a number of study groups as a study project, and thus has a hard deadline that cannot be exceeded. The system should be simple and intuitive to use. The system should be developed such that it is customizable to the individual child and its disabilities. Further, it should allow guardians to limit the functionality of the system. To allow other application developers to continue to develop the system and further applications after this semester, the system should be maintainable.

Technology *Covers the technology used to develop the system and the technology on which the system will run.*
The system must run on Android touch devices such as smartphones and tablets. Different hardware should be supported, although it is required that the unit is running Android 2.2 or newer. The system should mainly be developed using Java and the Android SDK version 8 for Android 2.2.

Objects *Describes the main objects in the problem domain.*
A smart-phone or tablet device. The Android platform. Global system- and application specific settings. Applications.

Responsibility *Covers the systems overall responsibility in relation to its context. That is, how the system would interact with the tasks to be solved using the system.*
The system should act as an assistive tool by providing preinstalled applications developed to aid and entertain the small-aged and disabled children using the system. Further, the system should provide the opportunity to install other third party applications. Through a home menu, the system should in accordance with the location, the user profile as well as the global settings of the system control which applications the user is allowed to access.

Sub Systems *The subsystems in the overall system.*
An administration module should provide access to user profile properties as well as global and specific application and system settings. A home menu must provide access to applications in accordance with the location, the user profile and the global settings of the system. Pre-installed applications including a day-planning tool and a Picture Exchange Communication System (PECS) application.

A.1.1. System Definition

The devised overall system definition is as defined:

A simple and intuitive module based single user system for Android touch devices, such as smart-phones and tablets. By masking the normal interface of an Android device, the device should offer functionality that is suitable for the intended user.

The system should be responsible for aiding and entertaining children with limited mental capabilities due to mental handicap and/or age, having a difficult time handling the complexity of a normal smart-phone or tablet OS. Guardians should be able to administrate the system by controlling selected application-, system- and user-specific settings through an administration interface on the phone.

Based on these settings, as well as the location of the unit, a home menu should be responsible for providing access to applications that conforms to the current settings and the state of the system. It should be possible for any third party to develop and provide additional applications to the system.

Beyond that, the system must be delivered with a set of pre-installed applications consisting of a visual, day-to-day, planning tool, and a PECS application. The system should be developed using Java and the recent version of the Android SDK. It is expected that the system supports Android 2.2. Further, it is expected that the system is maintainable to such a degree, that it allows other developers to keep developing the system as well as applications to the system after this semester.

APPENDIX B
TEST DOCUMENTS

B.1. Initial test of GirafPlace administration interface

Test method Dynamic black box testing
Chosen area The administrator interface on the server (girafplace.lcdev.dk)
Testing The administrator interface on GirafPlace

B.1.1. Test cases

1. **Create new user** - After creation, the user must be informed that he has been created, and now must wait for theGirafPlace administrators to verify him, so that he can gain access to the administrator interface.
2. **Log in with incorrect password (another user's password)** - Log in with the newly created user with incorrect password (another user's password). Must inform the user, he was not logged in.
3. **Log in with incorrect password (random password)** - Log in with the newly created user with incorrect password (random password). Must inform the user, that he has not been logged in.
4. **Log in with correct password** - Log in with the newly created user, before it has been verified, with correct password. Must inform the user, that he has not been logged in.
5. **Log in with incorrect password (another user's password)** - Log in with the newly created user, after it has been verified, with incorrect password (another user's password). Must inform the user, that that password is incorrect.
6. **Log in with incorrect password (random password)** - Log in with the newly created user, after it has been verified, with incorrect password (random password). Must inform the user, that the password is incorrect.

7. **Log in with correct password** - Log in with the newly created user, after it has been verified, with correct password. Must give access to the administrator interface.
8. **Upload a new application, with correct settings.xml** - Add a new application to girafPlace. Give the application a name, a short description, choose the corresponding *.apk* file, and set no abilities the application supports. This application should appear in a list of applications, that the user have uploaded through time.
9. **Upload a new application, with incorrect settings.xml** - Add a new application to girafPlace. Give the application a name, a short description, choose the corresponding *.apk* file, and set some random abilities the application supports. The user should be notified, that the *settings.xml* file is not correct setup, and therefore the application has not been uploaded.
10. **Upload a new application, without the settings.xml file** - Add a new application to girafPlace. Give the application a name, a short description, choose the corresponding *.apk* file, and set some random abilities the application supports. The user should be notified, that the *settings.xml* file has not been included, and therefore the application has not been uploaded.
11. **Edit an application** - Edit the added application, by changing the description, and changing what abilities the application support, and submit the changes.
12. **Edit administrator** - Change information about the user. Change the name, and submit the change.
13. **Log out of GirafPlace** - Log out of the GirafPlace, and be returned to the front page.

B.1.2. Test case results

1. **Create new user** - user *mlisby08@student.aau.dk*, password *1234* created - register page shown.
   PASSED.
2. **Log in with incorrect password (another user's password)** - Tried logging in with *mlisby08@student.aau.dk*, password *kodeord* (used by another account). Message: Login failed shown.
   PASSED.
3. **Log in with incorrect password (random password)** - Tried logging in with *mlisby08@*

student.aau.dk, password *GIVEMEACCESS* (not used by another account). Message: login failed shown.
PASSED.

4. **Log in with correct password** - Tried logging in with *mlisby08@ student.aau.dk*, password *1234*. Message: Login failed shown.
   PASSED.
   (Accepted mlisby08@student.aau.dk in the database)
5. **Log in with incorrect password (another user's password)** - Tried logging in with *mlisby08@student.aau.dk*, password *kodeord* (used by another account). Message: Login failed shown.
   PASSED.
6. **Log in with incorrect password (random password)** - Tried logging in with *mlisby08@*
   student.aau.dk, password *GIVEMEACCESS* (not used by another account). Message: Login failed shown.
   PASSED.
7. **Log in with correct password** - Tried logging in with *mlisby08@student.a*
   password *1234*. Administration interface shown.
   PASSED.
8. **Upload a new application, with correct settings.xml** - Tried uploading an application with a *settings.xml* with a settings tag within, with the name *Green Application*, and a short text description. Supported no abilities. Uploaded with no errors.
   PASSED (note, though, that the sw6.xmlvalidator was not up to date at the time of test).
9. **Upload a new application, with incorrect settings.xml** - Skipped, as a new version *sw6.xmlvalidator* was being developed at the time of testing.
10. **Upload a new application, without the settings.xml file** - Skipped. Functionality was not implemented
11. **Edit an application** - Tried editing an application. Worked as intended.
    PASSED.
12. **Edit administrator** - Changed name and password. Name was changed, and logging in was not possible with old password.
    PASSED.
13. **Log out of GirafPlace** - Clicked the logout button. Was returned to the login page.
    PASSED.

B.2. Implementation of GTP

Test method Dynamic black box testing
Chosen area Server of the GirafPlace (Marketplace for the Giraf application)
Testing The protocol for communication between the Giraf Application and the GirafPlace Server

B.2.1. Test case design

The protocol gives reason to define at least seven different test cases, one for each of the message strings - and an additional one if the server must check, if connection is true. In addition, it is relevant to check, what happens if the same, valid message is sent repeatedly, as well as check for timeout - that is, not sending anything to the server:

- The server receives ”HANDSHAKE” - the server must set *connection* to true and return ”WELCOME” to the client.
- The server receives ”SIGNOUT”, and *connection* is true - the server must set connection to false and return ”KTHXBYE” to the client.
- The server receives ”SIGNOUT”, and *connection* is false - an error must be logged.
- The server receives ”GETAPPS”, and *connection* is true - the server must return a list of applications.
- The server receives ”GETAPPS”, and *connection* is false - the server must log an error.
- The server receives a NULL value - the thread must shut down.
- The server receives any other string - an error must be logged.
- The server receives the same valid string repeatedly.
- ”SIGNOUT” - close connection on first signout, second ”SIGNOUT” should not be received, as socket is closed by then.
- ”GETAPPS” - Return applications on all GETAPPS.
- ”HANDSHAKE” - Return welcome on first ”HANDSHAKE”, ignore repeat.

B.2.2. Test cases

1. **Establish a connection to the server** - ”HANDSHAKE” is sent to the client. The response from the server is then checked (must

be ”WELCOME”), and the *variable* connection is checked to determine whether it is true.

2. **Close connection to the server** - A connection is first established. ”SIGNOUT” is then sent to the server. The response from the server is checked (must be ”KTHXBYE”), and the variable connection is checked to determine whether it is false.
3. **Close connection, without having established a connection first** - ”SIGNOUT” is sent to the server. The log is checked, to determine if a message was written to the log.
4. **Retrieve what applications that are available on GirafPlace** - The connection is first established. ”GETAPPS” is then sent to the server. The type of the response from the server is checked (must be an arraylist of applications) - it is then checked, whether the applications in the list equals the result of running the getApplications function, the function the server uses to retrieve applications.
5. **Retrieve what applications that are availble on GirafPlace, *without* having established a connection first** - ”GETAPPS” is sent to the server. The log is checked, to determine if a message was written to the log.
6. **Find out how many active treads that are currently running** - Send NULL to the server. Retrieve the number of active threads again, and compare with the first number.
7. **Try out various strings, to see how the server reacts** - Some various strings must be sent to the server, and the log must be checked, to ensure this was logged. The strings to send to the server are:

    - Lowercase of keywords - ”getapps”, ”handshake”, ”signout”
    - Partly uppercase keywords - ”GETapps”, ”HANDshake”, ”SIGNout”
    - Random uppercase keypresses - ”AA”, ”IUYTFVBNJYF”, ”WERJWHSB”, ”ZZ”
    - Numbers] - ”1”, ”9”, ”0”, ”124356789”
    - Special characters - ”#%&#&%”, ”)(/&%&/(”
    - Numbers, letters and special characters - ”23467ADF%%#”, ”1A#”
    - Keywords with additional letter/number/special character - ”GETAPPSA”, ”HANDSHAKEA”, ”SIGNOUTA”, ”GETAPPS1”, ”HANDSHAKE1”, ”SIGNOUT1”, ”GETAPPS#”, ”HANDSHAKE#”, ”SIGNOUT#”

B.2.3. Test case results

The test was executed using a custom-built client that allowed to send messages to the server, easily. Some of the test cases required a check to the internal structure of the server to confirm that the correct action had taken place.

1. **Establish a connection to the server** - The string ”HANDSHAKE” was sent to the server. On the server, we could see that ”HANDSHAKE” was received and the server responded to the client with the string ”WELCOME”. *PASSED*.
   To determine if a connection had been established, the variable *connection* was checked. For correct connection establishment the variable should be true, which it was. *PASSED*.
2. **Close connection to the server** - First a connection was established by sending the string ”HANDSHAKE” to the server. On the server side, ”WELCOME” was responded, and *connection* was set to true.
   The string ”SIGNOUT” was then sent to the server, in order to close the connection. In the serverlog, we could see that ”SIGNOUT” was received and that the server responded with the string ”KTHXBYE” *PASSED*. The *connection* variable, however, was still true, where it should have been false. *FAILED*.
3. **Close connection, without having established a connection first** - The string ”SIGNOUT” was sent to the server, without first sending handshake and thus creating a connection. In the serverlog, we could see that the string was received, and that the server had replied with the string ”The server received SIGNOUT, but the handshake has not been completed.”. The connection variable was also still false, further confirming that a connection had not been established. *PASSED*.
4. **Retrieve what applications that are available on GirafPlace** - A connection was established with the string ”HANDSHAKE”, and the server replied with ”WELCOME”. Then, the string ”GETAPPS” was sent from the client. In the serverlog we could see that the string was received, and the server replied with a list. The number of items in the list corresponded with the number of applications to be retrieved. *PASSED*.
5. **Retrieve what applications that are availble on GirafPlace, *without* having established a connection first** - The string ”GETAPPS” was sent to the server without first establishing a

connection. The server logged that it received GETAPPS while no connection was established. Thus, no response was given. *PASSED*.

6. **Find out how many active treads that are currently running** - First, we checked how many threads were active on the server before doing anything. There were currently 2 threads running. Then NULL was send to the server without having first established a connection. We confirmed that this did not create a new thread on the server as the number of active threads was still 2. *PASSED*.
7. **Try out various strings, to see how the server reacts** - During this test many different strings was sent to the server. These were variations of the generally accepted strings, but with different capitalization, casing and various other strings consisting of numbers or special signs.
    - Lowercase keywords - All strings were logged to the server - *PASSED*.
    - Partly uppercase keywords - All strings were logged to the server - *PASSED*.
    - Random uppercase keypresses - All strings were logged to the server - *PASSED*.
    - Numbers - All strings were logged to the server - *PASSED*.
    - Special signs - All strings were logged to the server - *PASSED*.
    - Numbers, letters and special signs - All strings were logged to the server - *PASSED*.
    - Keywords with additional number, letter or special sign - All strings were logged to the server - *PASSED*.

B.3. Test of GirafPlaceClient

Test method Dynamic black box testing.
Chosen area GirafPlaceClient.
Testing Whether GirafPlaceClient has issues that does not emerge under ”normal” use.

B.3.1. Test Cases

The test devised tests three areas we expect might cause problems when using GirafPlaceClient in an unintended way.

1. **Open GirafPlaceClient** - GirafPlace application list is being retrieved and displayed.
2. **Update application list** - GirafPlaceClient is updating the application list.
3. **Press back before GirafPlaceClient has finished retrieving applications** - The application should close.
4. **Reopen GirafPlaceClient** - GirafPlace application list is being retrieved and displayed.
5. **Rotate the device** - Application list is still shown.
6. **Open an application details activity** - Application details are shown.
7. **Rotate the device** - Application details are still shown.
8. **Close GirafPlaceClient** -
9. **Disable Internet connection on the device** -
10. **Open GirafPlaceClient** - The user must be notified that the application must have access to the Internet to work.
11. **Press back to exit GirafPlaceClient** - The application must be closed correctly.

B.3.2. Test results

1. **Open GirafPlaceClient** - Application list is shown. *PASSED*.
2. **Update application list** - GirafPlaceClient starts updating the application list. *PASSED*.
3. **Press back before GirafPlaceClient has finished retrieving applications** - The application stops updating the application list, and shows the last list that was fully updated. *FAILED*.
4. **Reopen GirafPlaceClient** - Application list is shown. *PASSED*.

5. **Rotate the device** - Application list is still shown, but icons are re-downloaded. *PASSED*[1].
6. **Open an application details activity** - Application details are shown. *PASSED*.
7. **Rotate the device** - Application details are still shown. *PASSED*.
8. **Close GirafPlaceClient** -
9. **Disable Internet connection on the device** -
10. **Open GirafPlaceClient** - Nothing happens. *FAILED*.
11. **Press back to exit GirafPlaceClient** - The application crashes. *FAILED*.

[1] Icons should, of course, not be re-downloaded when the device is rotated

B.4. Multi-project test

Purpose of the integration test

The integration test is made to test that the complete GIRAF system behaves according to the system definition. The main approach to this will be testing the complete lifecycle of GIRAF and its applications. This includes installation of GIRAF and fetching applications from the GIRAF place.

What is not covered is tests of the individual parts of GIRAF. The requirements for executing the integration tests is that the unit tests defined for each module all have passed. This also includes unit tests testing the couplings between modules. Testing the actual functionality of individual modules is done by groups individually.

Notice that the following test documents are based on dynamic black box testing. The purpose of the tests is not to find specific bugs, but to determine to what extent the different modules successfully integrates (if at all). The tests must be conducted by testers who are experienced android users and developers, who knows where UI elements are prone to fail.

Tested Versions

All tests are conducted on the following dev-branches, revision 1122:

sw6.admin:	/sw6.admin/branches/dev/
sw6.bmi:	/sw6.bmi/branches/dev/
sw6.lib:	/sw6.schedule/branches/dev/
sw6.schedule:	/s6w.schedule/branches/prototype1

Notice that *sw6.pecs* contained errors in revision 1122, hence the source is not included in this test. A stable `.apk` is available, and will be used when possible.

B.4.0. Test TD00

Program/Module/Object Under Test

GirafPlace application administration module, web interface.

Test objective

Ensuring GIRAF applications can be deployed to GIRAF Place, and that non-GIRAF applications and corrupted GIRAF applications are rejected.

Test conditions and intercase dependencies

The source must be available for all tested applications. The PECS source cannot be built at the time of this test execution, hence steps requiring the source has been skipped for this application.

Test cases

1. Create an Android application with an invalid *settings.xml*.
2. Upload the application to GIRAF Place.
3. Correct the *settings.xml* file, and upload the application to GIRAF Place again.
4. Verify that the *settings.xml* file is valid using the *sw6.xmlvalidator* tool.
5. Update the application with a new change in its code. Remember to edit the version number. Upload the updated application to GIRAF Place.

Expected behavior and result

The test was executed with two applications, BMI for kids and aSchedule. All test results hold for both applications.

1. The *settings.xml* is considered invalid by the *sw6.xmlvalidator* tool. *PASSED*.
2. The application is rejected by the GIRAF Place, and should report the same errors as the *sw6.xmlvalidator* tool. *FAILED*.
3. The application is accepted and stored on the GIRAF Place. *PASSED*.
4. *sw6.xmlvalidator* reports file is valid. *PASSED*.
5. The application will be upgraded, and stored on GIRAF Place. *PASSED*.

Observations

Applications with invalid *settings.xml* are accepted at GIRAF Place, this was not intended. Upgrade of packages worked as intended. The *sw6.xmlvalidator* tool reported the correct problems introduced in the *settings.xml* files. (reported as issue 44)

B.4.1. Test TD01

Program/Module/Object Under Test

GIRAF.

Test objective

Testing initial installation of GIRAF, and that the user is presented with necessary welcome dialogues.

Test conditions and intercase dependencies

The latest revision (r1122) of GIRAF should be available at: http://girafplace.lcdev.dk/start/.

Test cases

1. Download *sw6.giraf* from website (http://girafplace.lcdev.dk/start/).
2. Install *.apk* file.
3. Set GIRAF as the default launcher.
4. Setup user profile[2]

Expected behavior and result

1. The *.apk* file is downloaded. *PASSED*.
2. GIRAF is installed. *PASSED*.
3. GIRAF Launcher is started. A first run dialog should appear. *PASSED*.
4. The user profile setup is stored. *PASSED*.

[2]The user profile to set was: *Name*: `John`, *Address*: `Doe Street 1337`, *Parent's phone*: `+45 12345678`, *Gender*: `Boy`, *Birthday*: `01-02 2003`, *Enabled capabilities*: `Can drag and drop, can read, can work with numbers`.

Observations

Rotating the user profile screen clears text fields. (reported as issue 45 on the project issue tracker.)
Cursor not placed at the end of input boxes. Observed on all HTC devices running HTC Sense UI. (reported as issue 46 on the project issue tracker.)
Administration module loads slowly. (reported as issue 47 on the project issue tracker.)

B.4.2. Test TD02

Program/Module/Object Under Test

Launcher, GIRAF place, applications including ”in-app” administration.

Test objective

Ensuring applications can be installed on devices using GIRAF place. To ensure that the launcher and applications are able to use and store settings in lib, as well as abiding to the navigation flow for opening ”in-app” administration module that has been agreed upon.

Test conditions and intercase dependencies

TD01.

Test cases

1. Start GIRAF Place.
2. Download and install an application.
3. Navigate to launcher screen. Check that only GIRAF applications are shown.
4. Launch an application.
5. Go into administration module (using ”secret” key combination).
6. Change a setting for the application.
7. Press back until home screen is shown.
8. Start the application again
9. Go into administration module

Expected behavior and result

1. Ensure that applications are listed and filtered according to user profile (only applications that matches the user profile should be possible to download). *PASSED*.
2. The download should start. After download has finished, verify that the Android Package Manager installer starts, and that the installation succeeds. *PASSED*.
3. Verify only GIRAF applications are installed. *PASSED*.
4. Verify that the top panel is not shown. *PASSED*.
5. Verify that the administration module is the administration module for the particular application being tested. *PASSED*.
6. Setting is updated. *PASSED*.
7. Verify that the activities are shown in the following order: Administration module, Application, Home screen. *FAILED*
8. Application opens. *FAILED*.
9. Verify that the changes made before are the ones shown. *PASSED*.

Observations

The two failed tests only failed on when using the DigiPecs application, as it was, at the time of testing, using an old version of sw6.lib.
Also, the context menu in GirafPlace has an icon scaling error on the HTC Wildfire (reported as issue 48 on the project issue tracker).

B.4.3. Test TD03

Program/Module/Object Under Test

GirafPlaceClient (application installer and updater), as well as launcher application display (listing of applications).

Test objective

Installing and upgrading GIRAF and GIRAF applications.

Test conditions and intercase dependencies

Requires TD02 to be executed.

Test cases

1. Open GirafPlaceClient (and ensure that an upgraded version of the package installed before, has now been uploaded to GirafPlace. The upgraded version should introduce some changes in the *settings.xml* file).
2. Download the upgraded version.
3. The Android Package Manager installer opens, and starts replacing the existing package.
4. Go to launcher.
5. Launch the application again.

Expected behavior and result

1. The application installed in TD02 should be upgradeable. *PASSED*.
2. The new version is downloaded. *PASSED*.
3. The Android Package Manager installer opens, and starts replacing the existing package. *PASSED*.
4. Updated application must be shown correctly. *FAILED*.
5. Verify that the settings have been upgraded correctly by checking both in the application and the administration module. *PASSED*.

Observations

If one forgets to update the GirafPlace application listing, and the following assertions are true, then GirafPlace is not able to install the wanted application.

- The application is not installed on the device.
- The application has been upgraded on GirafPlace compared to the version shown on the list on the device.

(reported as issue 49 on the project issue tracker).

If one forgets to update the GirafPlace application listing, and the application is currently installed on the phone, GirafPlace shows an uninstall button. (Reported as issue 49 on the project bug tracker).

The reason for failing test 4 is that the application is now shown twice in the launcher. Both icons open the application though. Install an application. Verify that one icon is shown in the launcher. Upload new version of the application to GirafPlace. Upgrade the application. Go to the home screen. Two icons for the application are visible on the

home screen. (reported as issue 51 on the project issue tracker).
Icons in the *sw6.admin*, and *sw6.bmi* package (high res) does not follow the Android design guideline[2]. (reported as issue 50 on the project issue tracker).

B.4.4. Test TD04

Program/Module/Object Under Test

Back Button behavior of GIRAF launcher and applications.

Test objective

To test the back button behavior of GIRAF launcher and applications.

Test conditions and intercase dependencies

TD01 is executed and passed. A valid application matching the user profile have been installed.

Test cases

1. Navigate to the home screen of GIRAF launcher and start the administration module, by using secret key combination.
2. Press the back button.
3. Press the back button again (while still in the GIRAF launcher home screen).
4. Open application with more than one activity.
5. Open secondary activity from the main activity.
6. Open administration module from the application.
7. Press the back button(ensure that we currently are in the root of the administration module).
8. Press the back button.
9. Press the back button.
10. Press the back button.

Expected behavior and result

1. Administration module is started. *PASSED*.
2. GIRAF launcher home screen is shown. *PASSED*.

3. Nothing happens. *PASSED*.
4. Application is started. *PASSED*.
5. Secondary activity is shown. *PASSED*.
6. Administration module is started. *PASSED*.
7. Secondary activity is shown. *PASSED*.
8. Main activity is shown. *PASSED*.
9. GIRAF launcher home screen is shown. *PASSED*.
10. Nothing happens. *PASSED*.

Observations

None.

B.4.5. Test TD05

Program/Module/Object Under Test

Home key behavior.

Test objective

To test whether the home key handling works as intended.

Test conditions and intercase dependencies

TD01 is executed and passed. A valid application matching the user profile have been installed.

Test cases

1. Navigate to home screen (GIRAF launcher).
2. Press secret key combination to enter administration module.
3. Press home button.
4. Press home button.
5. Click an application to open it.
6. Press secret key combination to enter administration module.
7. Press home button.
8. Click an application to open it.
9. Press home button.
10. Press back button.
11. Press home button.

Expected behavior and result

1. GIRAF home screen is shown. *PASSED*.
2. Administration module opens. *PASSED*.
3. GIRAF home screen is shown. *PASSED*.
4. Nothing happens. *PASSED*.
5. The clicked application opens. *PASSED*.
6. Administration module opens. *PASSED*.
7. GIRAF home screen is shown. *PASSED*.
8. The clicked application opens. *FAILED*.
9. GIRAF home screen is shown. *PASSED*.
10. Nothing happens. *PASSED*.
11. Nothing happens. *PASSED*.

Observations

Open an application. Enter the administration module. Press home. Open application. This opens the administration module. This is unintended behavior, as children may enter the administration module. (reported as issue 43 on the project issue tracker).

B.4.6. Test TD06

Program/Module/Object Under Test

Testing uninstall functionality in girafPlace.

Test objective

Test whether applications can be deleted from GirafPlace.

Test conditions and intercase dependencies

TD01 is executed and passed. A valid application matching the user profile have been installed.

Test cases

1. Open GIRAF Place, Uninstall an application.
2. Open launcher.

3. Enter administration module, Open device settings, open package manager.
4. Download uninstalled application from GirafPlaceClient.
5. Application is installed without warnings or errors.
6. Application is shown.

Expected behavior and result

1. Application is uninstalled without warnings or errors. *PASSED*.
2. Launcher does no longer show uninstalled application. *PASSED*.
3. Uninstalled application is not shown in application list. *PASSED*.
4. Application is downloaded without warnings or errors. *PASSED*.
5. Application is installed without warnings or errors. *PASSED*.
6. Application is shown in the launcher. *PASSED*.

Observations

The uninstall process works as intended.

B.4.7. Test TD07

Program/Module/Object Under Test

GIRAF Place, User Profile.

Test objective

To check if applications are filtered according to user profile settings.

Test conditions and intercase dependencies

TD01 is executed and passed. A valid application matching the user profile have been installed.

Test cases

1. Open home screen.
2. Open the administration module.
3. Open user profile settings. (through administrative settings).
4. Edit capabilities in user profile. Set canRead to false.

5. Home screen is shown. Ensure no text is shown under applications.
6. Go into administration module.
7. Ensure that only applications, that meets the user profile capabilities, or already are installed, are shown.

Expected behavior and result

1. Home screen is shown. *PASSED*.
2. Administration module is shown. *PASSED*.
3. User profile settings are shown. *PASSED*.
4. Ensure all settings are saved when finished. *PASSED*.
5. Home screen is shown. Ensure no text is shown under applications. *PASSED*.
6. Administration module is shown. *PASSED*.
7. Ensure that only applications, that meets the user profile capabilities, or already are installed, are shown. *PASSED*.

Observations

Applications were filtered according to specifications.

BIBLIOGRAPHY

[1] Patton, R. (2006) Software Testing, SAMS, 2nd edition. 3, 19

[2] Android Foundation Android developer guide http://developer.android.com (2011). 9, 13, 14, 96