AALBORG UNIVERSITET
STUDENTERRAPPORT

aSchedule

Visual Schedule for Android

Aalborg University
Group f11s601e
Spring 2011

Title:
aSchedule

Topic:
Application Development

Semester:
SW6, Spring 2011

Group:
f11s601e

Students:

Jens Kristian Geyti

Jeppe Pihl

Kenneth Fuglsang Christensen

Supervisor:
Saulius Samulevičius

Copies: 5
Pages: 102
Published: May 27th, 2011

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg Øst
http://www.cs.aau.dk

Synopsis:

GIRAF is the overall system developed in this project. The system's main responsibility is to, by simplifying the user interface, make the Android platform available to children with limited mental capabilities. Four development teams have in cooperation developed the system. The contribution from the authors of this report is a digital visual schedule for children with autism, called aSchedule. This report describes the process and development of both GIRAF and aSchedule, with main focus on aSchedule. The report covers an overview of previous research regarding visual schedules, which is used as the basis for the analysis and design of aSchedule. The report presents the Android platform before subsequently presenting the implementation of aSchedule. Finally, based on an external evaluation of aSchedule, a recapitulation is provided to conclude the project.

Preface

This project concerns the collaborative development of a mobile Operating System (OS) overlay targeted children with autism, called GIRAF. The system consists of an application launcher, a number of applications and an administration backend. The application launcher and the applications are minded on the children, and have been developed to support their needs and limitations. The administration backend has been developed with the caretakers in mind, this will provide them with the possibly to customize the settings of the OS and its applications.

The project features a complex organizational development process, since the different parts have been developed by separate, small teams of 6th semester software engineering students at Aalborg University in spring 2011. The focus of this report is on a visual schedule application, called aSchedule, developed by the authors of this report.

It should be noted that the report requires a basic knowledge of Java, as several code samples will be presented. The code samples will not necessarily compile out of context. Further some code, which is not considered relevant, has been removed. When this is the case, it will be denoted with "...".

A DVD is supplied together with this report containing the source code of the developed project. The supplied source code is fully functional and represents everything documented in this report. Though a more current version is available on Google Code[1] where the project is published as open source.

As per regulation, a danish summary of this report is found in Appendix C.

We would like to thank Ulrik Mathias Nyman, Assistant Professor at Aalborg University, for proposing the project and further provide ongoing feedback throughout the project.

We would also like to thank department manager Tove Lund Thomasen from Birken, a kindergarten for children with special needs, for providing feedback on the finished aSchedule application.

[1] http://code.google.com/p/sw6android/

Contents

Part I

Introduction

This part will initially provide an introduction to the problem at hand. By examining relevant research, an understanding of what autism is, what visual schedules are, and how these can improve the every day life of people with autism, will be provided in Chapter 1.

After this, the part will proceed with an evaluation of a number of development methodologies. This provides the basis for selecting the most suitable method to be used in the overall project, GIRAF, as well as in our individual project concerning the development of aSchedule. This is described in Chapter 2.

Finally, the part will introduce the development platform which has been have chosen for managing the collaborative development of GIRAF, in Chapter 3.

1 Problem Research

Today's smartphones have evolved far beyond what traditional mobile phones used to offer. Powerful processors and large capacitive multi-touch displays have made it possible to provide advanced computing, entertainment, and connectivity to users. Moreover, touch and gesture-based OSs have made advanced functionality simple enough for everyday users. While the growth in demand for powerful smartphones has entailed tremendous development within this area, the market for software for children with special needs is still dominated by classic desktop software, such as Boardmaker[1]. Only few smaller applications targeted at children with special needs are available in the Apple App Store and Android Market[2], even though studies suggest that children with autism aid from using computers in everyday tasks [HHM+10].

This chapter provides a basic understanding of what autism is as well as what visual schedules are. This basic knowledge is used to argue for how and why electronic visual schedules are useful, and especially why a smartphone or tablet based visual schedule may aid children with autism. Additionally, a basic understanding of visual schedules is required for understanding certain design decisions introduced in later chapters.

1.1 Autism

Autism is a disorder which affects the neural development of a person. While the definition of a person with autism is very broad, the diagnosis is based on some of the typical symptoms associated with autism [Rey06]:

- Lack of social skills.
- Lack of communicative skills.
- Repetitive and compulsive behavior (e.g. repetitive movement or strictly following rules).

Other symptoms that are not essential for diagnosis, but still typical for people with autism are the following:

- Restricted interest (i.e. on certain subjects).
- Increased perception and attention.
- Sensory abnormalities (both under- and over-responsiveness).
- Difficulty grasping vague concepts, such as the notion of time passing.

The actual cause is still unknown, but studies have shown that the disorder might have a connection to certain genes. Research has shown that a complete cure is not feasible, but several types of medication and assistive tools such as visual schedules have been developed to aid people with autism in their everyday life.

[1] Boardmaker is developed and distributed by Mayer-Johnson, http://www.mayer-johnson.com.
[2] Apple App Store and Android Market are the official application repositories of mobile Apple and Android devices respectively.

1.2 Visual Schedule

A visual schedule is a tool which can help children with autism structuring their everyday lives. In the simplest form a visual schedule consists of a series of visual cues where each symbolize an activity. These cues are often represented as pictographs, but they could as well be photographs, items, written words, or basically anything which, in some way, describe the activity. The foundation of visual schedules is based on the strategy of *"first-then"*, an example of the strategy could be as the following: "***First*** *take a shower,* ***then*** *dry yourself using a towel*". This section is based on [Dav11].

1.2.1 Using Visual Schedules

The usage of a visual schedule is depicted in Figure 1.1. The figure shows a schedule where the first activity has just been finished and is therefore moved to the "finished activities" column. From here, the child advances to the next activity, called "second activity". For some reason the child runs into a problem and requires help. When the child has received help from a guardian, he or she can continue with the activity and at some point finish it. This will continue until all activities has been finished.

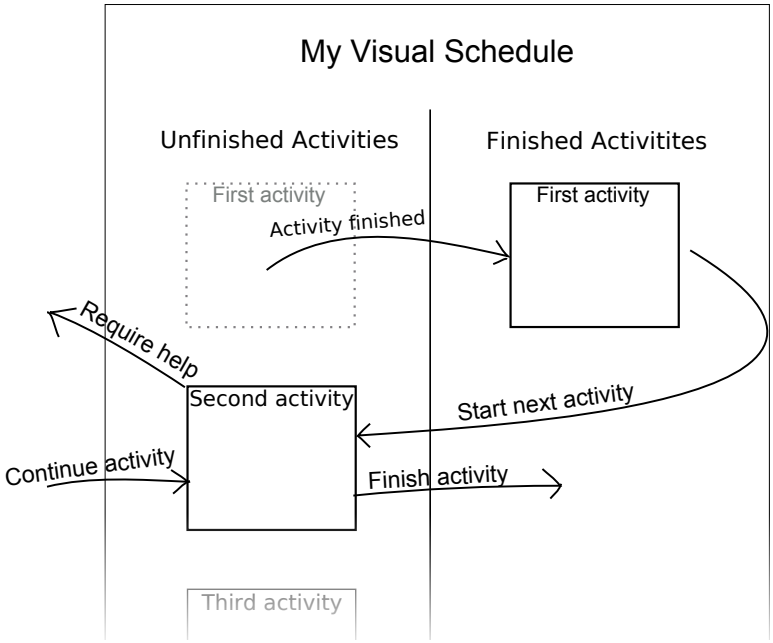


Figure 1.1. Shows the interaction with a simplified example of a visual schedule.

As mentioned earlier, visual schedules can have many appearances, and this is just an example of one, but the main chain of actions displayed in Figure 1.1 is universal.

1.2.2 Benefits of Using a Visual Schedule

Visual schedules have the following beneficial qualities:

- They provide a structured environment to the child. This is achieved by informing the child of what time, and in what order the activities of the day is to be performed.
- They do not require the child to understand speech, as this is overcome by the use of visual cues.

- Many children with autism have trouble with sequential memory and time organization. The use of visual schedules can ease the complexity of such tasks.
- They reduce the child's anxiety level, as it enables him or her to anticipate, and possibly organize, their activities on a day to day basis or even longer time spans.
- They aid the child's transitioning from one activity to another, both in speed and by enabling the child to do so more independently. This is achieved by the previously mentioned format, as this encourages the movement from one activity to another.
- They have the potential to help the child in social interactions. This can be achieved by including social gestures as activities in the schedule, this could for instance be a greeting, when the child is arriving at some destination.
- They can contribute to the child's motivation for doing less desirable tasks, if the child can see that it is followed by a more preferable one. Also, children with autism often find computers to be a strong motivator.

1.2.3 Creating a Visual Schedule

A visual schedule is a tool which is required to be used consistently and is meant to help the child on a daily basis. The longer the child is to use the schedule, the better - it is even usable beyond childhood into adult life. Initially, when using a visual schedule, it should be managed by the child's guardians. This responsibility can, however, gradually be given to the child over time [MK99]. A number of prerequisite skills are required for a child to use visual schedules. This includes the ability to:

- Identify the essential parts of a picture, and thereby not be distracted by the background. This is necessary when following a photographic schedule.
- See a correspondence between pictures and physical objects.
- Accept manual guidance, as this will allow the guardians to help the child when he or she is having trouble with an activity.
- Perform some activities without guidance, as it is preferred if the visual schedule contains some activities which the child will be able to do without guidance.

When creating a visual schedule, the following guidelines are recommended [MK99]:

- The schedule should have a consistent layout.
- The schedule should either move from left to right or from top to bottom.
- The activities of the schedule should have a clear ending.
- It should have a method to indicate for the child that an activity has been finished.
- It should provide the child with an overview of at least two activities at a time.
- The schedule should be designed to fit a specific child's needs. What is meant by this is the way activities are represented - for instance, some children might prefer photographs, while others tends to like symbols better.
- If photographs are used, they should not be under- or overexposed and they should only show the target on a plain background.

When creating and using a visual schedule, one should be aware of a range of complications which can emerge when a child is trying to finish an activity, in order to be able to handle these complications timely. The following are examples of plausible complications the child might experience:

- The child does not understand what he or she is supposed to do.
- Something has triggered a tantrum, and the child needs to be calmed down.
- The activity encompasses some physical task which the child is incapable of doing.

1.3 IT as Assistive Technology

Digital visual schedules have a number of advantages over the physical counterparts that are typically used. As noted by Tom Babinszki[3], people with autism typically process visual information better than auditory information [Bab11]. The possibilities for combining pictures, video and audio makes computers an efficient tool for presenting information in a way that is easier for a person with autism to grasp. The authors of "Activity Schedules, Computer Technology, and Teaching Children With Autism Spectrum Disorders" argues that *"students [with autism] often find spending time on the computer reinforcing"*, and that *"children with autism preferred instructions presented by a computer to that presented by a teacher"* [SKKT06]. Additionally, Thorp argues that a computer is a strong motivator for children who are difficult to engage and by targeting motivation, the child's interest in learning a variety of different skills can be increased [Tho11].

While this in itself is a motivation for using electronic aids, an electronic visual schedule also has the possibility of handling some of the challenges that are involved with making physical visual schedules. For instance images, audio and video can all be presented on a single device, and daily schedules can be generated based on calendar-entries (e.g. having an activity only on Mondays at 13:00 etc.), making it easier for guardians to plan a schedule.

A computerized visual schedule does have a number of drawbacks compared to an physical schedule. It is not very portable, and using mouse and keyboard as input devices may limit some users from being able to interact with it effectively. Our hypothesis is that a mobile tablet device or smartphone, is able to overcome those problems that makes a visual schedule on a computer less usable. A smartphone or a tablet is very portable, and the touch interface of these devices is likely to be easier to learn to use than a mouse and keyboard. Also, should a visual schedule on a smartphone prove usable, a such device is socially acceptable to bring and use in almost all public spaces.

1.4 Summary

There are many different impairments associated with autism, all related to the mental capability of a person. These are generally related to the person's capability to interact with other people as well as to organize a series of actions or events.

A visual schedule is a widely recommended tool for helping people with autism, mostly children, with organizing their every day lives. Traditionally, these are physical schedules made of paper and printed pictures or symbols. These suffer from the difficulty of minimizing the required effort from a guardian to effectively use the visual schedule on a daily basis.

It is important to notice that the rather short description of what autism is, presented in this chapter, is sufficient for the scope of this report. This is due to the fact that this project is based on existing research in the area of what visual schedules are and how they can be used to aid people with autism in their every day lives. We can hereby simply define a person with autism as a person who:

- Is incapable of managing tasks that must happen or occur during a day.
- Has normal fine motor control but may be unable to read.
- May struggle with getting instruction from others, but generally acknowledge instructions from computers.

Thus, the focus of this project is to use the existing research and our knowledge of the existing variants of visual schedules in an attempt to create an electronic version based on modern tablets and smartphones.

[3]Owner of Even Grounds, an accessibility consulting company.

2 Development Methodology

As mentioned in the preface, this project consists of a common platform, GIRAF, and an application, aSchedule, developed by the authors of this report. The GIRAF project is developed by four development teams, each developing individual parts that constitutes the complete GIRAF system. Even with limited knowledge of the project to be developed, many of the challenges introduced by developing in teams can be handled by introducing an effective development methodology, and a set of common guidelines for the teams to follow. This chapter discusses these problems as well as how they are handled in this project.

The four developer teams consists of 3-5 developers, in all 12 developers. Teams have been formed in the start of the semester, based on working style, development experience and the like. The common objectives of the development teams are defined by the study regulation:

The student must *"demonstrate knowledge on central techniques on the process of developing applications that solves realistic problems [and] demonstrate skills on analyzing, designing, programming, trying out, and testing applications that are part of a complex organizational subject."* [DI09].

This essentially lists the most essential software engineering disciplines. A development method is a framework used to structure, plan, and control the process of software development. Over the years many methods have evolved, each with its own strengths and weaknesses. The available methods are each meant for specific types of projects based on a variety of technical, organizational, and team-related considerations. Choosing the right method and customizing it to the individual project's needs is an important step towards a successful project.

This chapter will describe the concept of Object-Oriented Analysis and Design (OOAD), and further briefly describe and compare three iterative and incremental methods to allow us to decide what method is most suitable for developing the aSchedule application. The most central aspects will be covered for each of the methods: Unified Process (UP), Extreme Programming (XP), and Scrum, followed by an evaluation and reasoning for choosing a combination of OOAD and XP over the remainder.

2.1 Object-Oriented Analysis and Design

OOAD is an object-oriented toolbox that can be used to model a system as a set of interacting objects. Each object modeled is characterized by its state and behavior. OOAD defines various models to show e.g. the static structure and dynamic behavior of these objects. These models are commonly represented using Unified Modeling Language (UML).

OOAD is combined from two smaller parts: Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD). OOA is related to the modeling of the functional requirements of the system whereas OOD is focused on producing implementation specifications.

OOAD is built upon four core analysis and design principles [MMMNS01, p. 18]:

Model the surroundings Usable systems suit the surroundings. Therefore, it is important to model both the problem domain and the application domain.

Emphasize the architecture An easy-to-understand architecture supports the collaboration between designers and programmers. A flexible architecture makes it easy to refine and further develop the system.

Use patterns The usage of established ideas and well-tried components enhances the quality of the system and the effectiveness of the development process.

Customize the method Any project has its own set of special challenges. OOAD must therefore be customized to the concrete needs in a given analysis and design situation.

OOAD concerns the four perspectives in its four main activities as show in Figure 2.1. Analysis and design is a strongly iterative process since thoughts in one perspective yields new ideas for other perspectives. Every project has a different focus and each perspective might be characterized by different insecurities. For example, one project could be based on new technology that the developers have not previously had experience with, and another project will be highly-distributed and require a challenging architecture. The priority and focus of the main activities of OOAD is based upon the individual needs of the project at hand.

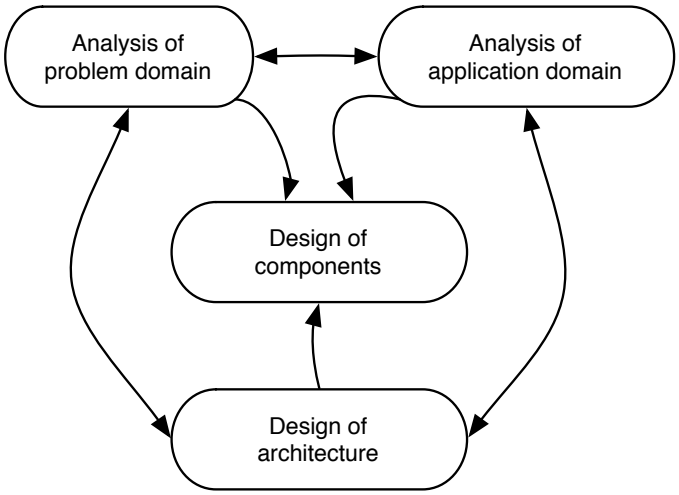


Figure 2.1. The main activities of OOAD.

The most important result of OOAD is the documentation that helps control the development activities. It can be used to collect intermediate results and aid further development of the project as well as maintain important agreements among any involved party in the development process.

The documentation is thus an important tool to create a coherent project of high quality.

2.2 Methodologies

Choosing the right tool for the job is always important, regardless of how small the job is. Using a sledgehammer to hang pictures might leave a rather large mark on the wall. In much the same way, it is important in software development to choose the right development methodology when approaching a project. Otherwise, the project may very well end up being too costly, error-prone, delayed or even fail completely.

2.2.1 Extreme Programming

XP is a popular agile method based on five core values: communication, courage, feedback, respect, and simplicity.

XP evolves around fourteen core practices in no particular order [Lar03]:

- Whole team
- Small, frequent releases
- Acceptance testing
- Test-Driven Development
- Release planning game
- Iteration planning game
- Simple design
- Pair programming
- Frequent refactoring
- Team code ownership
- Continuous integration
- Sustainable pace
- Coding standards
- System metaphors

While this is a relatively large number of practices, many of which may not make sense separately, each of them contributes to a larger synergy effect. Since XP focuses a lot on communication and being the whole team together at all times, it is best suited for small teams.

Work Products

XP is radically different from other methods in not requiring detailed work products except for program code and tests. However, it does not disallow other detailed work products. Although agile methods avoid detailed up-front specifications and plans that span the entire life cycle, most of these methods encourage writing down details for at least the next iteration. In contrast, XP emphasizes oral communication for requirements and design. An example of this is that a feature is often discussed, summarized, and hand-written on a *story card*, not to be compared to use-cases since they are much shorter and contains *user stories*. The story cards are further used to document fixes and non-functional requirements that the customer wants.

The planning game is an ongoing activity where a *task list* is created which contains all of the tasks to be implemented in the upcoming iteration. The task itself is very simple and is done e.g. by moving cards from one whiteboard to another. Another highly recommended work product of XP is *visible graphs*. These graphs should illustrate daily metrics of the progress and quality. The graphs should be present in the project room to ensure that the team always has an idea of the project's current situation and react on this.

Project Life Cycle

The XP life cycle is divided into five phases [Lar03]:

Exploration During the exploration phase, some key functional and architectural requirements are defined. Further, some story cards may be written, with rough estimates.

Planning In the planning phase (referred to as the *release planning game*), the customers and developers complete the story cards and rough estimates. Afterwards, it is decided which stories go to the next release.

Iterations to first release The goal of this phase is to implement a working system ready for the next release. The phase consists of testing, task writing and estimating, which is planned in the *iteration planning game*. The planning game is held for each iteration to pick a number of stories to implement based on their status and priority.

Productionizing During the productionizing phase, the goal is to deploy the software to the customer. The phase includes documentation, training, marketing, etc.

Maintenance The maintenance phase is an ongoing phase during the remainder of the project's life time. New errors might occur which have to be fixed, and stories may have to be improved. This might result in doing all of the phases again for incremental releases.

2.2.2 Unified Process

UP is an iterative and incremental development process best described as an extensive, customizable framework. The manager has to refine this into a detailed process description for an organization or project, choosing what parts of the framework to use. A process description is called the *development case* of the project and consists of *disciplines* which are groupings of *artifacts* (work products).

UP is risk focused meaning that the project team has to address the most critical risks early in the project's life cycle. The deliverables after each iteration must be selected to ensure the greatest risks are addressed first. This focus strives to build, test, and stabilize the core architecture of a project early to have a solid foundation.

Work Products

The UP contains as much as 50 optional work products (artifacts) that can be chosen from for the project's development case [Lar03]. Each artifact is targeted at a specific discipline. This section introduces a small, but essential, subset of these.

Requirements Discipline Key requirement artifacts include:

Vision Summary of objectives, features, and business cases.

Use-case Model Stories or scenarios of using the system. Emphasizes functional requirements.

Design Discipline Key design artifacts include:

Design Model Object model (static and dynamic) of the packages and objects.

Software Architecture Document Short learning aid to understand the system.

Project Management Key project management artifacts include:

Software Development Plan Overall plan of milestones, resources, etc.

Iteration Plan Detailed plan for the upcoming iteration (not all iterations).

Risk List Risks ordered by severity.

In UP projects, there is typically a lot of documentation and diagrams to help describe the system. Diagrams such as conceptual classes, sequence diagrams, class diagrams and so on are very common. As described, UP is supposed to be custom tailored to each individual project, but it dictates some dependent ordering of the activities that undergo during the course of the project.

Project Life Cycle

UP divides a project into four phases. Each phase is divided into a number of time-constrained iterations which each result in an increment, which is a release of the system that contains new or improved

functionality compared with previous releases. UP does not impose any specific iteration lengths. Most iterations will include work in multiple disciplines (e.g. requirements, design, implementation, and testing) but the relative focus will shift over the course of the project.

Inception The inception phase is the shortest phase of the project. Typical goals of the phase is to establish a business case for the project, outline the use-cases and key requirements, outline candidate architectures, identify risks and define a preliminary project schedule.

Elaboration The elaboration phase should document the majority of the system requirements, address already known risks, and establish and validate the system architecture. The architecture is validated primarily through the implementation of an *executable architecture baseline*, a partial implementation of the system including the core and components which have architectural significance. The result of the elaboration phase is a plan of cost and schedule estimates for the construction phase. The plan should at this point be accurate and credible since it is based on experience gained through the elaboration phase.

Construction The construction phase is the largest phase of the project. During the phase, the remainder of the system is built on the foundation laid in the elaboration phase. Functionality is implemented in a series of time-constrained iterations. Each iteration is commonly driven by a use-case and often contains the construction of a number of UML diagrams.

Transition The final phase is the transition phase in which the system is deployed to the end-users. Feedback received from an initial release may result in further improvements to be incorporated during multiple transition phase iterations.

2.2.3 Scrum

Scrum is an iterative and incremental development method. At first hand, Scrum appears simple, yet it has practices that significantly influence the work experience. A key theme in Scrum is its emphasis on being empirical rather than defined processes.

Scrum evolves around some key practices [Lar03]:

- Self-directed and self-organizing teams.
- No external addition of work to an iteration once chosen.
- Daily stand-up meetings.
- Usually 30-day iterations.
- Demo to stakeholders after each iteration.
- Each iteration based on client-driven planning.

A project based on Scrum requires a few roles to be employed for the project to succeed. The *product owner* is a person capable of creating and prioritizing the *product backlog*. He will collaborate closely with the team to guide and direct them on a non-organizational level. This means that the product owner does not decide what the team is doing - the team itself is responsible for deciding how much work it can cope with in one sprint (iteration). A *scrum team* is a group of people who works on the *spring backlog* developing the product. Each team has a *scrum master*, who is half developer, half manager. The Scrum master is not a leader in the traditional sense, but ensures the Scrum values and practices are being followed. The Scrum master is also the one who conducts the *daily scrum* and *sprint review*.

Work Product

Scrum is composed of a few work products to help track the progress of the project.

Product Backlog The product backlog is a document for the entire product. The product backlog contains all items that will be built during the project. It is open and editable by anyone working on the project and contains rough estimates. The estimates are used as rough guidelines to help the product owner prioritize the items.

Sprint Backlog The sprint backlog is specific to one team, and contains information about which features the team is going to implement in the current sprint. The features are segmented into smaller units which are estimated to 4-6 hours of work. The sprint backlog is updated daily by the team members, in order to continuously track the progress.

Spring Backlog Graph The sprint backlog graph (also called burn down chart) is a visual summary of estimated task hours remaining in the sprint backlog. This is considered the most essential project data to maintain.

Project Life Cycle

A Scrum project life cycle is composed of four phases.

Planning The planning phase usually contains a limited number of meetings (*sprint planning meeting*) at the beginning of each sprint cycle. The purpose of the meeting is to create the product backlog.

Staging The goal of the staging phase is to define the system architecture. This process is done in collaboration of a few team members and some architects by reviewing the backlog. At the end of this phase, review meetings are held where the teams exchange information and present progress and current problems.

Development This phase is meant for analyzing, designing and implementing the backlog requirements into a package that is ready for release after being tested. At first, the product owner creates the sprint backlog, based on the product backlog. The sprint backlog is then further divided into 4-6 hour sessions where the actual implementation by the team occurs. During this phase, daily meetings are held to present the progress, resolving issues, adding additional items to the sprint backlog etc.

Release The last phase prepares the release by a sprint review meeting. This includes integration testing, documentation, training, marketing, etc. Finally the release is deployed to the customer.

2.3 Choosing a Development Methodology

The choice of a development methodology often has a tendency to be influenced by buzzwords and trends in the industry. When deciding on a development methodology, it is important to understand the fact that there will never be a single best process - there is no silver bullet [BT03], all methodologies have pros and cons. This section will describe how and what development methodology has been found suitable for aSchedule.

2.3.1 Balancing Discipline and Agility

As the choice of development methodologies will always be a trade-off, it is important to determine the characteristics of the project. Agile methodologies promise fast development time and ways to handle

rapid project changes. Plan-driven (or disciplined) development methodologies promise stability and predictability. [BT03] presents a number of observations to those *seeking to integrate agile and plan-driven methods in their development process.* Some of their key points are the following:

Neither agile nor plan-driven methods provide a silver bullet. Agile methods handle changeability, but do not scale up well, nor do they handle complexity well. Plan-driven methods handle conformity, but founder on changeability.

Agile and plan-driven methods have home grounds where one dominates the other, based on the following five characteristics:

- **Criticality** describes the loss due to defects.
- **Size** describes the number of developers.
- **Culture** describes the percentage of people who "thrive on chaos" compared to those who "thrive on order".
- **Dynamism** describes the rate of change in a project.
- **Personnel** describes the amount of skill among the developers. A less experienced developer is Level 1A or 1B, whereas a highly skilled developer is Level 2.

Build a method up, do not tailor it down. Unexperienced developers have a tendency to pick a method and stick with it, often at unnecessary expense. Cost/benefits should be considered when applying a development method.

Focus less on methods, more on people, communication and expectation management as corroborative evidence show that people-factors dominate other software cost and quality drivers.

With these points in mind, Figure 2.2 has been created to assess where the project currently is with respect to the five characteristics described. The hachured area shows assessment.

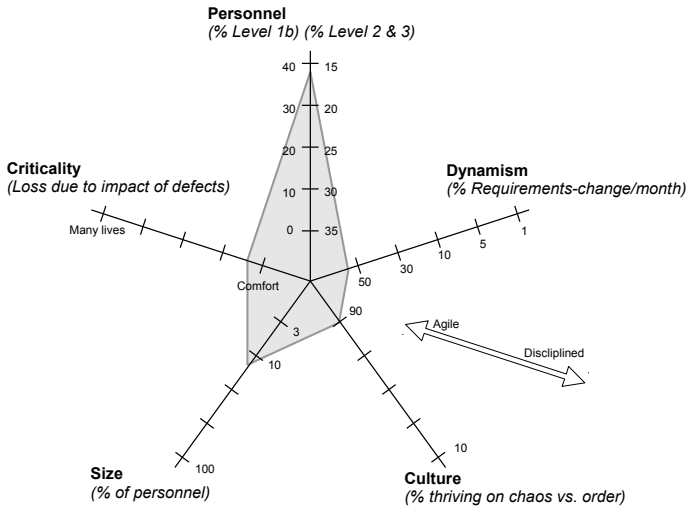


Figure 2.2. An assessment of where aSchedule is with respect to the five key axes on the figure. The figure is based on the one in [BT03].

Neither the units used on the axes, nor the exact shape of the hachured area should be considered precise. What can be drawn from the figure, is that the project itself is generally well suited by agile methods. It is very dynamic regarding requirements and changes, and most of the developers thrive on chaos more so than order. Also, there is only very limited loss due to defects in the finished product. On the other hand, the project is concerned with development of a software solution based on a mobile platform for children with autism and their guardians, all of which is unknown to the developers. A traditional, disciplined method is better at capturing and controlling this.

2.4 Our Development Method

In [MS92] different approaches for software design is suggested. The article has two key points:

- *Relying on an analytical mode of operation to reduce complexity introduces new sources of uncertainty requiring experimental countermeasures.*
- *Relying on an experimental mode of operation to reduce uncertainty introduces new sources of complexity requiring analytical countermeasures.*

When starting the aSchedule development process, the development team has no knowledge of the subject, hence every aspect of the project can be considered complex. To develop a good solution that suits the requirements of the users and works properly on the platform, we have to gain a rather thorough understanding of both before actual development can start, a process that is better captured by disciplined methods. Therefore, we will use OOAD during the beginning of the project to analyze our target group as well as to express ourselves through a number of prototypes. The prototypes both aid in gaining platform-specific knowledge but also creates a connection between our knowledge of the target group and the software we are to develop for them. The OOAD approach is also found suitable for the basic analysis of GIRAF, which will be done in cooperation with the three other development teams, as the product of this will be a well-documented base of knowledge that all teams can utilize in their individual development process.

The traditional approach to requirements engineering, and the "big design up front"-mentality will generate many ideas of how to solve the problems that will be defined during this process. As [MS92] states, this introduces a situation in which agile methods are suitable to use for experimenting, in order to reduce this uncertainty. Besides this, using a disciplined method in the programming phases of the project is not applicable nor feasible due to a number of reasons:

We do not have a specific customer Without a specific customer, we have no way of getting predefined requirements. We have to research the problem domain in order to come up with requirements. While a preliminary OOAD process does give a reasonable foundation to come up with requirements, additional ideas and approaches worth implementing may appear while developing.

We have no contract Because there is no customer, we do not need to specify a feature set and a price up front to avoid misunderstandings between what was expected, and what was developed. The only requirement is that the project is bound by a specified deadline and the study regulation defined for this semester.

Flexible criteria for success Compared to *real* projects, the goals of a semester project are very flexible. If the goals prove too hard to reach, they can be revised. If the goals are too easily fulfilled, further goals can be added.

For the very same reasons, we find an agile approach to be more fitting. The choice of development approach for the implementation phase is based on the following requirements:

The overall plan will be revised often, and development will more likely affect the plans, rather than the plans affecting the code.

Iterations will be extremely short (less than a week), and the result of each iteration will affect the overall plan greatly.

It must be easy for every team member to assess what the current overall goals are, even though they are changed often.

Tools and processes used for planning must be lightweight as the project will require a lot of re-planning throughout the development process. This will limit the time spent on planning to a bare minimum and avoid unnecessary overhead.

While these requirements are valid for all four developer teams, no single development methodology could be agreed upon. The development teams have different focus in their individual projects, and hence have different approaches to the development phase. From the very beginning of the project, it has therefore been a primary concern that the required project collaboration interferes as little as possible with the individual goals and working style of the development teams. As few rules and deadlines as possible have been enforced upon teams. Low coupling and high cohesion between areas of responsibility as well as between developed code have been the single most important quality attribute in making sure a complete and working program has been produced. By making low coupling a goal in itself, teams are free to do programming independently, and reaching the common goals is less dependent on effective leadership.

Based on the knowledge gained through our research of development methods, we feel that XP best suits our development team's requirements to efficiently manage our resources and project progress. Therefore, we will switch to XP once we have gained enough knowledge of our target group and platform.

Besides the core principles of XP, we will focus on:

- Using pair-programming extensively, to make all developers have knowledge of project changes without the need for additional meetings and communication.
- Using short iterations of work to be able to actively steer the direction of the project as desired. Short iterations will further help develop ideas that can be adopted continuously.
- Always having a usable program. A full feature set may not be implemented, but the program should be usable at the end of each iteration.
- Using continuous integration to ensure that our part of the greater system integrates into the complete system at all times.

It is not possible, nor intuitive to implement XP fully, due to the following:

- We have no on-site customer. Instead, we will explicitly take the role of the customer using role-playing in the planning game.
- Our plans are too vague to fully utilize the idea of user stories as being project goals. Instead, the XP planning process will be used to guide and steer the somewhat chaotic planning process we are suggesting.

Development Platform

3

As stated earlier, the overall system (GIRAF) is developed in individual teams of 3 to 5 developers. While this means that a large amount of man hours are devoted to the development, it also entails some organizational challenges, as the work of each group has to be integrated at some point in the development process. To cope with this, a number of tools have been applied. These tools and their use will be described in this chapter.

3.1 File Versioning

Subversion (SVN) is a centralized version control system used to handle changes in files made by multiple contributors. SVN offers version control of individual files and folders by assigning a revision number, committer's name, and timestamp to every change made to a repository. This provides centralized access to a project's files (when accessing the latest revision) as well as to any historical revision. The possibility of recovering older versions of ones files as well as examining the history of versioned items is hereby enabled. Further, the contributors are provided with the opportunity to create separate branches of the overall system, making it easier to initiate the development of a large and possibly experimental change of the system. Branches can be created and reintegrated into another branch (e.g. a "main" branch) at any time [PCSF08, cha. 4].

In this project a file repository, shared between all the development teams is used. Each project has a "trunk" branch which is used for stable code. To easier enable developers to implement unstable functionality, branching of individual sub-projects is possible. Other teams may pull files from this repository if "bleeding edge" changes are desired or needed, before they are committed to the stable branch, of which other groups normally pull files from. Whenever deemed necessary, branches are created. Whenever a branch is considered stable, the branch is reintegrated into the main, stable branch. The groups have in collaboration decided that when code enters the stable branch, it must be fully documented and adhere to the coding standards that were defined at the beginning of the project. The implementation strategy is visualized as an example in Figure 3.1.

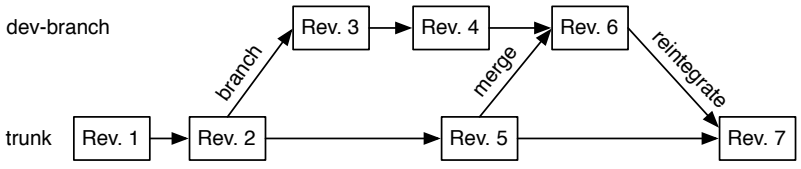


Figure 3.1. Shows an example of the SVN implementation strategy. At first the trunk is created (Rev. 1), then it is updated (Rev. 2). A developer chooses to branch the project, possibly to make some additional functionality (Rev. 3, 4, 6). In the meantime, the trunk is updated and merged into the branch. Finally the developer finishes the functionality and reintegrates the branch into the trunk (Rev. 7).

Besides offering a structured and "safe" way to store source code, the choice of a common repository has

been made to make it transparent to other groups, what and when development is being done, as well as to provide a sense of confidence in the code base and inspiration from seeing progress, not only in ones own project but the entire system.

3.2 Wiki

A wiki is a website which allows collaborative editing of a knowledge base containing any number of pages. The fact that a wiki is available to multiple users, both for reading and writing, makes it the perfect tool for knowledge sharing between the different development teams [Wik11, cha. 4].

In this project a wiki page has been has set up. The wiki enables any member of the development teams to publish relevant information. This type of information could for instance be one of the following:

- How-to guides on how to set up various development related tools.
- Documentation which should be shared among the teams, while still being easily editable.
- Contact information.
- Summaries of meetings.
- Etc.

Having a centralized location for written agreements, plans, and guidelines is crucial in a project like GIRAF, where development teams work on individual projects for prolonged periods of time.

3.3 Communication

To enable all development teams to know where the project is headed, a sufficient amount of communication throughout the project is crucial.

In this project this has been accomplished by the combined use of a shared Internet Relay Chat (IRC) channel, group emails, and meetings.

IRC chat An IRC channel is a chat room where the available developers are present. This channel provides the possibility of reaching developers of any team, instantly. Using this is a quick way of seeking help with a problem, or casually discussing the time and place of future meetings. Team members are expected to be present in the channel when committing files to the SVN repository, to make it easier to blame them for compile errors.

Team Emails The team mails provides a way of reaching either all development teams, all the developers of a single team or a single team member. This can be used for communicating important messages, such as new releases and agendas or summaries for meetings.

Meetings For meetings to be as beneficial as possible they should be held when appropriate. It was decided that once a week would be adequate, with the exception of weeks which were filled with lectures without project relation. To prevent the meetings from being too time consuming for the whole team, only a single member of each development team was selected to attend. This person was named the team leader. The purpose of these leader meetings were to discuss the subjects which have relevance to the overall system. To ensure that all team members had a say in the important decisions, an agenda, made prior to each meeting, enabled each development team to discuss the subjects in advance. Further a summary, providing an overview of what had been decided was released after each meeting. To ease the work load of the team leaders, special committees, comprised of other team members were made. These discussed more separable subjects such as the architecture and test planning for the overall system.

3.4 Standards

One of the initial tasks of the leader meetings was to ensure a certain level of interconnectedness of the overall system. This was achieved by settling on a set of standards which each development team had to follow. To keep this as simple as possible, while keeping the Android spirit, it was decided to use the Android code style guidelines for contributers. Though, some of these are too heavy for a relatively small project as GIRAF, the guidelines were deemed as not required, but merely something to strive for. The guidelines used when developing aSchedule includes the following:

Do Not Ignore Exceptions Exceptions should always be handled. This will further be elaborated in Section 11.2.

Fully Qualify Imports When importing classes, avoid using the asterisk (*) to reduce the number of import statements. By specifying exactly which classes used, it makes the code more readable for maintainers.

Use Javadoc Standard Comments Every class, and non-trivial public method should have at least a single sentence, starting with a 3rd person verb, which describes what the class or method does.

Define Fields in Standard Places Either define fields in the top of the class or above the method where the field is used.

Follow Naming Constraints The naming of fields should be dependent on its modifiers. For instance, if a field is non-public and non-static, it should always be prefixed with an "m".

These, and the remaining coding-style guidelines are described further in [Goo11b].

Part II

Analysis and Design

Visual schedules are well known and widely used by parents of children with autism [GPI+06]. The idea of developing a digital visual schedule requires limited innovative work, as best practices have already defined the basic functionality of visual schedules. This does, however, require a good understanding of how visual schedules are used in practice when deciding on a design.

The purpose of this analysis is to understand and characterize how a visual schedule is to be used in aSchedule, in order to identify the parts that can be administered, controlled, and monitored. The result of this analysis will form the basis of the actual design of the digital visual schedule.

System Definition

4

Before a system is analyzed, designed, and developed, a system definition is needed to state the fundamental properties of the development and usage. The system definition is used to define what information the system encapsulates, what it is supposed to do, in what environment it is to be used as well as specifying basic requirements.

GIRAF is the overall system developed as the result of this semester project amongst four individual development teams. As part of this system, this group has developed aSchedule, a visual schedule application for children with autism and their guardians.

An overall system definition is produced to understand the environment in which the visual schedule will be implemented, as well as a system definition for the visual schedule application.

The system definitions have been constructed by defining FACTOR(S) criteria for GIRAF and aSchedule after performing a minor problem research (see Chapter 1). It should be noted that the system definitions have been defined very early in the process, to decide on a common foundation for the different development teams to use. They have since regularly been revised to better fit the actual goals of the project. As such, they have been an ongoing agreement between development teams on what the overall goal is defined as, more than they have been system definitions in the words typical sense.

4.1 GIRAF

This section defines the system definition of the overall GIRAF project. The definition has been agreed upon amongst all teams early in the development process.

4.1.1 System Definition of GIRAF

A simple and intuitive module based single user system for Android touch devices, such as smartphones 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 smartphone or tablet OS. Guardians should be able to administer 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 device, 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 Picture Exchange Communication System (PECS) application. The system should be developed using Java and the recent version of the Android Software Development Kit (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.

4.1.2 FACTORS for GIRAF

Functionality *Describes the systems functions 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 administer common settings by need. The system should mask the normal functionalities of the device 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 device. 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 administer, 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 smartphone 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.*

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 his/her 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 device 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 (see Chapter 8 for more information about Android).

Objects *Describes the main objects in the problem domain.*

A smartphone 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 pre-installed 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.

Subsystems *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 includes a day-planning tool and a PECS application.

4.2 aSchedule

Based upon the overall system definition, this section presents the system definition of the aSchedule application developed by this team.

4.2.1 System Definition of aSchedule

A simple, adjustable, and intuitive, single user, visual schedule application for Android touch devices. The application should be responsible for aiding children with limited mental capabilities due to autism, and/or age, having difficulties managing everyday activities without guidance. Guardians should be able to configure the application settings through an administration interface on the phone. Further the application should provide the guardians with the possibility to manage the schedule by creating and maintaining the child's activities throughout the day directly from the device. The application must not be dependent of any external devices. Further, the application must attempt to minimize the required effort from guardians. The application should be developed using Java and Android SDK version 8, and support Android 2.2 and newer. Further, it is expected that the application is maintainable to such a degree, that it allows other developers to keep developing the system after this semester.

4.2.2 FACTOR for aSchedule

Functionality *Describes the systems functions that support the application domain tasks. That is, defining what the system is able to do.*
A visual schedule to illustrate the current day's activities. A step by step guide to any given activity can be specified. The application must also provide guidance to activities that are not specific to a day.

Application Domain *Concerns those parts of an organization which administrate, monitor, or control a problem domain.*
The children with autism and its guardian (parents and pedagogues).

Conditions *Covers conditions under which the system will be developed and used.*
The children may have different disabilities such as being unable to read, understand spoken language well, focus, or adhere to simple instructions. Generally aimed at children incapable of doing everyday tasks without structured guidelines. The application will be developed as a study project and it will therefore have a fixed deadline. Furthermore the application is a subsystem of a larger system and must be able to both use external information and/or functionalities and provide information to other applications.

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 device is running Android 2.2 or newer. The system should be developed using Java and the Android SDK version 8 for Android 2.2.

Objects *Describes the main objects in the problem domain.*
The Child, Guardians, Activities, Step-by-step instructions, and Settings.

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.*

To improve the everyday life of both the child, and guardians, by structuring a child's daily activities.

5 Problem Domain

A problem can be regarded from numerous points of view. By analyzing and describing the reality in which GIRAF and aSchedule is used, in the way a user will see it, a model of the problem domain can be made. The problem domain is analyzed to understand and agree on what problem the project is to solve, as well as how users will use GIRAF and aSchedule to solve it. By expressing the problem domain in class diagrams, an extremely simple and concise model is constructed. This has the advantage of limiting derailing of an extremely agile development process, in which there is no specific customer. Whenever new goals and features are suggested, the development team may use the problem domain class diagrams to consider if it aids at solving the overall problem.

5.1 Analysis Scope

While a complete problem domain analysis of GIRAF is feasible, its usefulness is limited. A common understanding of GIRAF within the development teams is defined by the system definition and FACTORs in Chapter 4. Without an actual customer or precisely defined users, a problem domain analysis of GIRAF will be no more than vague assumptions as to what reality it captures. The aSchedule application has a more concrete user base and purpose, as it is based on an existing, well known concept. As such, a problem domain analysis of aSchedule is more likely to capture the reality as potential users are supposed to see it. Additionally, since aSchedule is the part of GIRAF being developed by this team, a thorough understanding and agreement on the problem domain is a useful tool, both when developing the actual application, but also when taking on the role of the users in the planning game of XP. Therefore, only the problem domain of aSchedule is analyzed in this chapter.

5.2 Structure

The organizational structure of a visual schedule is quite simple. It consists of the schedule itself, and people using it. People may either be *children* or *guardians*. Children and guardians use the visual schedule differently. Guardians may either be the parents of the child using the visual schedule, or pedagogues who tend to the child.

A visual schedule consists of a number of activities. An activity can both be assigned to specific dates, and repeatedly on a specific time of the week, in this report referred to as a repeat pattern, e.g. "Thursdays at 2 PM". Each activity consists of at least one step, explaining the child how to execute the activity. Finally each activity can be "spontaneous". This means that the activity is instantly accessible, and not only when it is placed in the schedule. This makes it possible to help the child with unpredictable activities, such as toilet visits. This structure is shown in Figure 5.1.

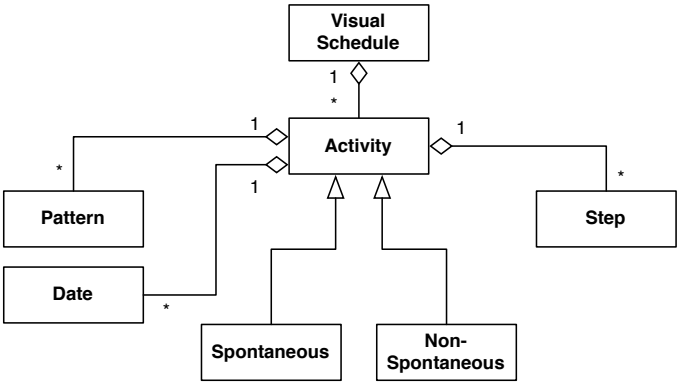


Figure 5.1. Structure of the visual schedule.

The relationship between people and the visual schedule is seen in Figure 5.2. Note the visual schedule is only to be used by one child at a time, while guardians may administer several visual schedules.

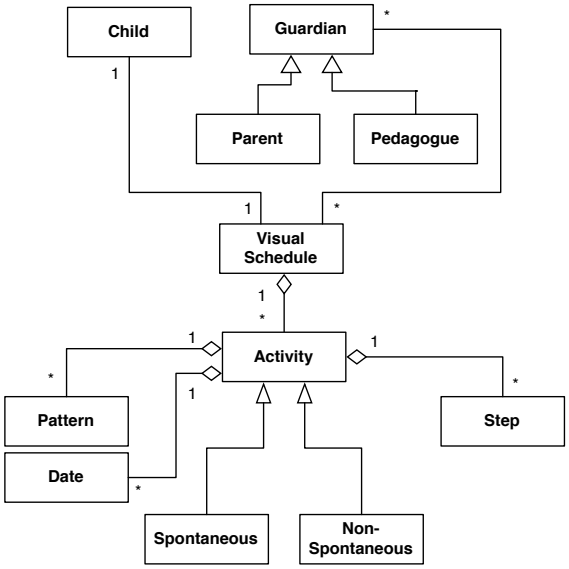


Figure 5.2. Shows the relationship between the people using the visual schedule, and the visual schedule itself.

5.3 Behavior

Class diagrams are useful tools for understanding and visualizing the problem domain. However, they only provide a simple, static overview of the problem domain. By elaborating on the changes that may happen to an object over time, a better understanding of the dynamics of the system is obtained.

The following depicts the behavioral pattern of some of the most relevant classes within the problem domain. The dynamics of the problem domain is linked to the objects within the classes *child*, *guardian* and *visual schedule*, hence the behavioral pattern of these classes are depicted in the following.

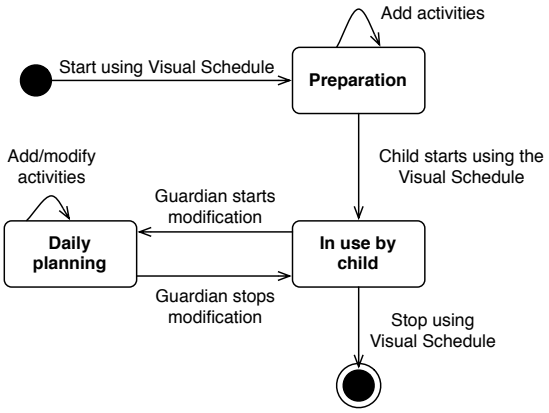


Figure 5.3. The behavioral pattern of a visual schedule.

A visual schedule is either being administered (prepared or modified) by a guardian, or being used by a child. Activities may be added and removed from it, and the activities may be rescheduled at any time, see Figure 5.3.

Figure 5.4 shows the behaviors of guardians, whom are the ones, initially creating the visual schedule. As long as a visual schedule is in use, a guardian may modify and reschedule activities and steps on it.

Guardians are also responsible of notifying a child about upcoming activities. Children (Figure 5.5) are the users of a visual schedule. A child may pick up and use a visual schedule on his own initiative, or because a guardian told him to. The child is the one who executes the activities in the visual schedule. As such, when executing an activity, steps are completed until there are no more, or until the child stops using the schedule for other reasons.

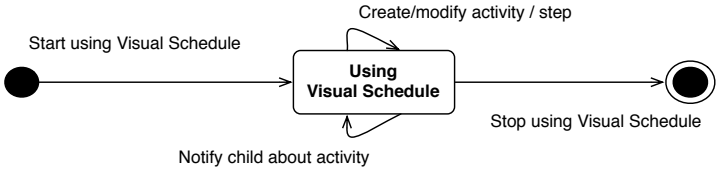


Figure 5.4. The behavioral pattern of a guardian.

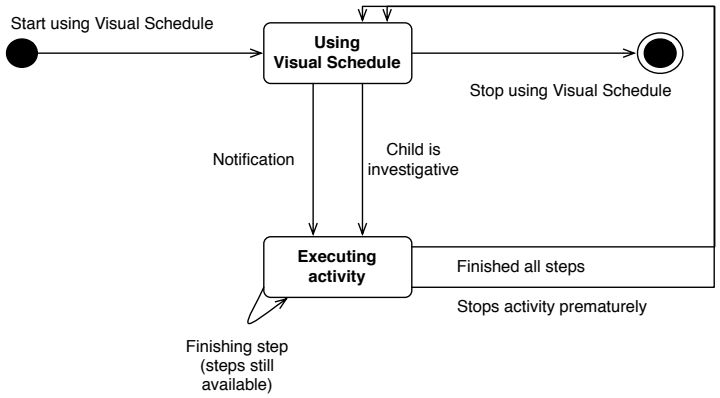


Figure 5.5. The behavioral pattern of a child.

The problem domain analysis is used to gain insight into and define how the visual schedule is used by the users. The problem domain analysis forms a basis on which the system design can be developed, ultimately leading to a complete system architecture of aSchedule and GIRAF. This will actively steer the remainder of the process.

6 Design

The system definition of GIRAF in Chapter 4 defines the fundamental properties of the system as a whole. The system definition of aSchedule and the problem domain analysis conducted in Chapter 4 and 5 defines how aSchedule is to be used, as well as who will use it. The developer teams of other parts of the overall system have conducted similar analyses of their individual projects.

System definitions and problem domain analyses are useful for gaining and categorizing information about users and the way they will use software. The scope of this chapter is to define the software design of the entire system (GIRAF), as well as of aSchedule. This design analysis is the foundation on which the actual software is implemented, and is a crucial tool for successful code integration across development teams.

The chapter will describe the overall system architecture of GIRAF before presenting prototypes and architecture of aSchedule.

Notice that this chapter assumes that GIRAF is an Android application. The reasoning for the choice of platform will, however, not be introduced before Chapter 8.

6.1 GIRAF

6.1.1 System Architecture

As mentioned in Section 2.4, low coupling between the development teams' area of responsibility is one of the most important quality attributes of the system architecture. Therefore, the common system architecture that has been developed by the development teams focus on how the area of responsibility is separated and on how different subsystems of the system interface. The precise structure of individual components is defined by the individual development teams. Generally, the system consists of the following components:

Application Launcher Is used by the user to navigate a device for starting applications.

GIRAF Place Makes it possible to install and update applications relevant to the GIRAF system (e.g. the PECS application and aSchedule). Further a backend is provided to enable developers to upload their applications and updates.

Administration Interface Is used by guardians to manage settings, including each of the applications' settings.

aSchedule Is developed by the authors of this report. It communicates with Application Launcher and Administration Interface in order to identify itself correctly as a GIRAF application.

PECS application Is, like aSchedule, a GIRAF application. Therefore it must adhere to the same guidelines as aSchedule.

Common Settings Storage Is used by applications to access the user profile of the system and to get and set predefined settings.

While this to some extend can be used to determine the area of responsibility of each module, a better understanding of how each subsystem interfaces with the rest of the system is required, if development teams are to develop towards a common platform. Figure 6.1 aids at specifying this, though, the internal structure of subsystems cannot be seen if this figure. The arrows on the figure illustrates the flow of information in the system, and does not specify which module initiates communication.

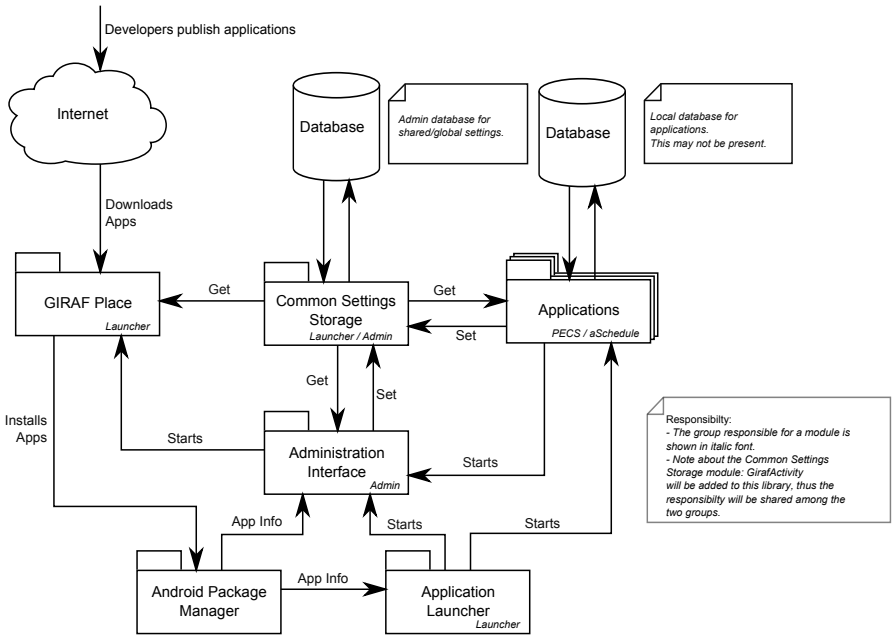


Figure 6.1. The overall architecture of GIRAF. Notice that the development team responsibility is marked in an italic font, on each component.

6.1.2 Common Settings and User Profile

The Common Settings Storage holds both application settings and the system user profile. As it is used by most parts of GIRAF, its purpose and structure has been defined in collaboration between the development teams.

Settings of GIRAF applications must be stored in the Common Settings Storage, to ensure that all system configuration is done from the Administration Interface, and not by the individual applications. This is a requirement as the Administration Interface is to be used by guardians, while the applications themselves are to be used by the child.

As implied earlier, the Common Settings Storage contains a user profile of the child. The user profile is used by the Application Launcher to only list applications usable to the child, and by applications to present themselves in a way that suits the child (e.g. by avoiding text if the child is not able to read). The user profile consists of the following settings:

- Name
- Birthday
- Gender
- Address
- Phone Number
- Can drag-and-drop
- Can hear
- Can read
- Reads digital clocks
- Reads analogue clocks
- Requires large buttons
- Can speak
- Knows numbers
- Can use keyboard
- Requires simple, non-distracting visual effects
- Has poor vision

This list is the result of a collaborative brainstorm amongst the teams, to agree on which attributes each team would like to be able to detect, and consequently adjust their application to.

6.1.3 Interface Requirements

The general overview of the system architecture in Figure 6.1 makes it possible to pinpoint how subsystems interface. Knowing this, requirements to these interfaces can be agreed upon between developers of the interfacing components.

As mentioned, the aSchedule application interfaces with the Application Launcher, the Common Settings Storage and the Administration Interface. The following explains the design requirements of these interfaces relevant to aSchedule. The implementation details of these are described in Part III.

The Application Launcher must be able to start applications This entails that applications must register themselves as a GIRAF application. To do so, the application must be located in the correct Java package and the application must extend a Java class provided by the Application Launcher developers, allowing the application to interact correctly with the Application Launcher and Administration Interface.

The Application Launcher must be able to filter applications For the Application Launcher to be able to filter applications, applications must register their user profile requirements when uploading the application through the GIRAF Place, e.g. whether an application requires the user to be able to read.

Applications must be able to get and set values in the Common Settings Storage This requires a connector that is developed by the development team responsible for the Common Settings Storage. The connector must not reveal to the application how data is persisted, nor require the consumer application to know details of this.

Applications must be able to start the Administration Interface To ensure all applications are able to do so in the same manner, all applications must extend a class that enables them to start the Administration Interface using a key combination that only guardians are supposed to know.

6.2 aSchedule

6.2.1 System Architecture

aSchedule is developed using an XP process, though a basic definition of the system architecture has been developed. This exercise is not described in the XP process, but the aSchedule architecture has been made for the following reasons:

- The developers are inexperienced Android developers. An effective agile process requires a high degree of "common knowledge" within the team, which is not present at the beginning of the project.

- Android defines an application flow that developers must abide by.
- To ensure low coupling and proper division of responsibilities is thought into the design from the very beginning of the project.

The system architecture of aSchedule strives for decoupling the User Interfaces (UIs), functionality and data. A Model-view-controller (MVC) pattern would be a good design pattern choice to enforce separation between these elements, however Android does not explicitly support such an organization of code. What Android offers is to separate static layouts from the control elements, however the controller is responsible for populating and manipulating the content of the views defined by a layout. Figure 6.2 gives an overview of the base architecture of aSchedule.

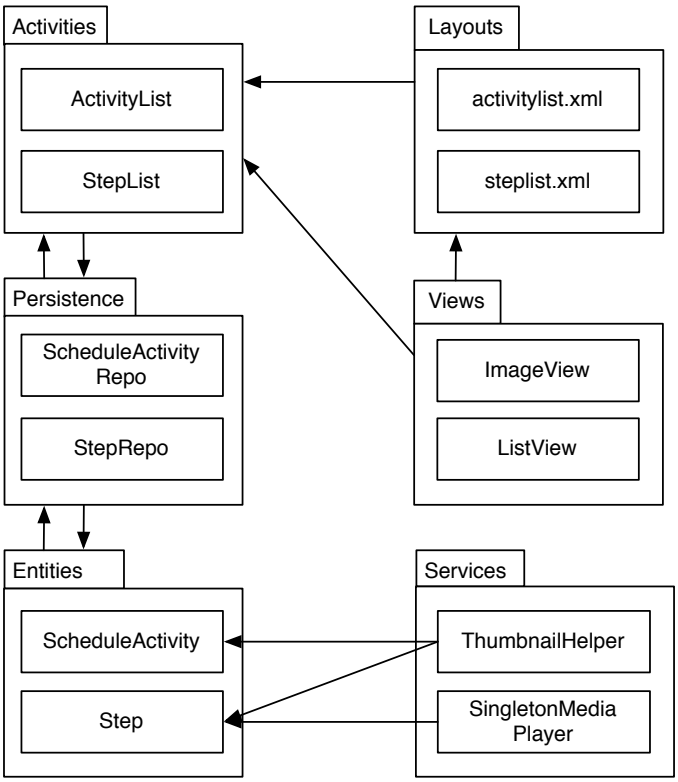


Figure 6.2. The basic architecture of aSchedule. The application contains more sub-components than visible in the figure (i.e. the components regarding the administration, are not shown). The figure explains the basic structure of components, not details of the actual implementation. The arrows illustrate the information flow in the system.

Android activities are responsible for generating the "screens" seen by the user. These are based on layout-files that define the layout of the screen. Layouts consists of Views. If elements must be added at runtime (e.g. Steps), an Activity will generate the required Views and merge them with the ones defined by the layout. An activity is also responsible for adding data to views. To separate the data from activities, they will be accessed through a model handling persistent data. To comply with business logic, the model works with objects representing the data in the database, rather than accessing data directly. This is made possible by using Object-Relational Mapping (ORM). The entities that are passed from the model to activities contain data relevant to them. If the data within them are abstract representations of complex data types, e.g. addresses to sound files, they will utilize services to present and use them in a manner that is usable to the activity (i.e. for playing a sound).

6.2.2 Prototype

The architecture of aSchedule is rather simple, as the problem it tries to control is of limited complexity. The most complex part of the project is to present data to the user in a simple, yet effective manner. A number of brainstorms and prototypes have been created before starting actual development of aSchedule. The goal of these prototypes has mainly been to come up with a user friendly interface for the child to use, but also to come up with an effective administration interface for their guardians.

The added benefit of creating a prototype, is that the required Android activities becomes known, as each "screen" is represented by an activity. This fact has been utilized extensively in the planning of the development of aSchedule. The final prototype, chosen for implementation, is shown in Appendix A.

6.2.3 Entity Relationship Diagram

As mentioned in Section 6.2.1, aSchedule uses a private database to persist the data that can be captured in the application. The database design is illustrated in Figure 6.3.

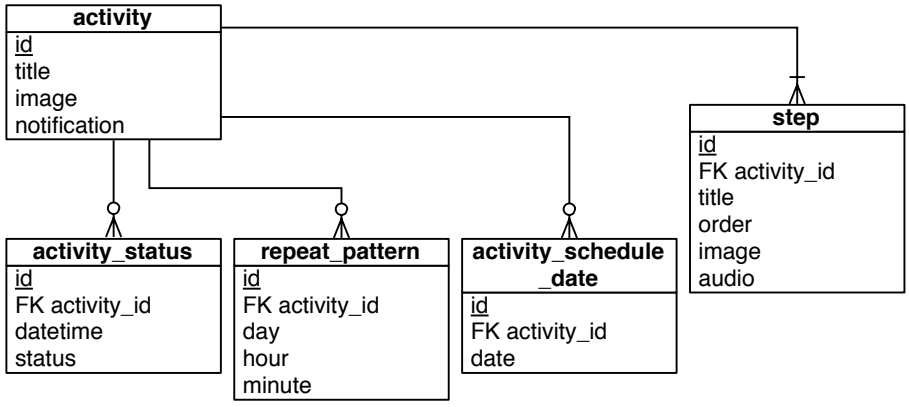


Figure 6.3. The entity relationship diagram of aSchedule.

The diagram is merely meant to give the reader an overview of the relation between entities in aSchedule. The database itself is generated by an ORM, to provide activities with object-based access to the data. The implementation details of this is explained later in the report in Section 10.3 and thus no further details will be given at this point.

The database of aSchedule is rather simple. What is noticeable is that activities are not represented as instances as they would be in a physical, traditional visual schedule. Activities are templates that contain relevant properties, and a number of steps. Based on dates and repeat patterns stored in the database, activity instances are generated that can be presented to the user of the visual schedule. Information about activity instances are stored in the *activity_status* table, where activity instances are identified by a `DateTime` value.

Summary of Analysis and Design

7

The analyses and design decisions accounted for in the previous chapters document the knowledge of GIRAF and aSchedule required for the four development teams to start implementation of their individual projects. The chapters also describe the work that has been done prior to initiation of the development phase.

Fundamental knowledge of autism and visual schedules have been accounted for, as this is an obvious requirement for being able to develop a visual schedule for people with autism.

Development methodologies have been discussed, in order to determine what development methodology and implementation strategy is better suited for the development of GIRAF and aSchedule: OOAD is used to agree on common features and requirements of GIRAF, and XP will be used by the authors of this report to implement aSchedule, as an agile method is found to be better suited at capturing the challenges within it.

The system definitions of GIRAF and aSchedule have been defined in order to determine the user base and fundamental requirements and goals. In the development phase of the project, these have served as a concise reminder to development teams of who the user is, and what is valued in GIRAF.

A problem domain analysis of aSchedule has been conducted, to limit the complexity of the project before the development phase was started. The problem domain analysis provides a concise model of how users are supposed to use visual schedules with aSchedule. Additionally, the results of the problem domain analysis are valuable when designing the database model of aSchedule, as most of the object classes within the problem domain and their internal relationships are captured by the problem domain model.

The system design has been created to ensure that a common agreement existed between development teams of what the overall system architecture should look like. Additionally, the requirements of interfacing components have been discussed, to make sure that development teams agree on areas of responsibility with regarding these interfaces. Agreeing on this is an important exercise to do before implementation starts, as development is done independently by development teams, hence well-defined interfaces are absolutely crucial for making GIRAF a complete, and working product.

Part III

Implementation

The previous two parts provided the fundamental knowledge required to develop GIRAF and aSchedule, as well as the analysis and design of these. Parts of the analyses documented in part 1 and 2 have been conducted in collaboration with all the development teams of GIRAF. As mentioned in Section 2.4, the development phase is executed individually by each development team.

This part will initiate with a description of the platform on which GIRAF will be applied. In order to give the reader enough insight to understand the implementation, the developed solution will be presented in Chapter 9 before diving into implementation details. These will include how some of the challenges in the development of GIRAF and aSchedule have been handled.

The Platform

8

Android is everywhere: phones, tablets, and TVs. Soon, Android will even be in cars and all sorts of other devices. However, the common foundation of Android devices is small screens and, most are without hardware keyboards. Though, Android will probably mostly be associated with smartphones for the foreseeable future.

Android smartphones (as well as others like the Apple iPhone) offers a rich user experience to mobile users. Internet services on mobile devices dates back to the mid 90's [Bor11]. However, only in recent years have phones capable of Internet access taken off. Thanks to trends like text messaging and products like Apple's iPhone, phones that offer Internet access are rapidly growing in popularity. Therefore, working on Android applications gives the developer experience with an interesting technology in a rapidly changing market.

Programming mobile devices has in general been a pain for developers due to the fact that they are small in every dimension:

- Screens are small.
- Keyboards, if even available, are small.
- Pointing devices (like a stylus or fingers) are inexact compared to a mouse on small screens.
- CPU speed and memory are restricted.

Moreover, applications running on a phone have to deal with the fact that they are running on a phone. This means applications must not:

- Utilize all of the (limited) CPU resources such that e.g. calls cannot be received.
- Utilize all of the (limited) CPU resources such that it affects the user experience negatively.
- Crash the operating system, e.g. by leaking memory.
- Work improperly with the operating system, such that the application does not allow other applications to run.

Hence, developing applications for a mobile device is a different experience than developing desktop or web applications.

Android makes an attempt to make up for some of these limitations, rendering a developer experience closer to that of desktop or web applications. For instance, Android can be programmed through the commonly-used programming language Java extended with an Android specific Application Programming Interface (API) in well-known tools like Eclipse. Furthermore, Android provides a fairly rigid framework in which an application needs to work to not interfere unexpectedly with other applications and operations on the device.

No matter the environment, developing applications for mobile devices will still be a different experience than developing desktop or web applications. This chapter will provide details on the concepts of the Android platform as well as how to take advantage of its capabilities.

8.1 Alternatives to the Android Platform

As of now, there are at least two viable alternatives to the Android platform which could have served as the foundation of GIRAF. These are the Apple iOS platform and Microsoft Windows Mobile Phone 7-Series.

These have, however, not been chosen in favor of Android for a number of reasons:

- iOS only runs on the Apple iPod Touch, iPhone, and iPad.
- iOS is more restricted, limiting us in how we can customize it (e.g. no custom launcher can be installed).
- iOS requires an OSX machine for development.
- WMP7 is still young and not very widespread. [Kil11]

Also, the simple fact that Android is open-source gives us the potential of investigating the source code to better understand its capabilities and any potential errors that occur during development.

8.2 Devices

The Android platform opens up for a large variety of devices from many different manufacturers. As part of the overall meetings throughout the development of GIRAF, we have chosen a common platform and a set of devices on which the system must work correctly. The selection has been made to represent a wide variety of devices and a large percentage of the available Android devices on the market.

First and foremost, it has been decided to support Android 2.2 (API level 8) and newer. This firmware version is available for most devices, see Figure 8.1, while still being quite recent as of writing this (it was released on June 29, 2010).

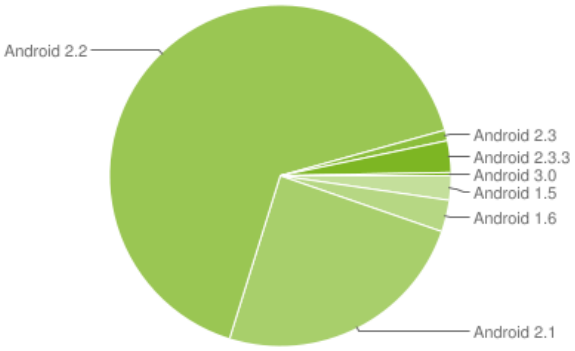


Figure 8.1. The distribution of Android versions on the market [Goo11c].

The Android 2.2 update provides a range of new developer APIs that could prove useful through development of GIRAF, such as a Media Framework and new Camera / Camcorder APIs.

Table 8.1 shows a comparison of the devices which GIRAF and aSchedule is aimed to support.

Name	HTC Hero	HTC Desire	Samsung Galaxy Tab
Image			
Screen	3.2" 320x480 pixels (Android classified as medium density *mdpi*)	3.7" 480x800 pixels (Android classified as high density *hdpi*)	7" 1024x600 pixels (Android classified as high density *hdpi*)
Processor	528 MHz ARM	1 GHz ARM	1 GHz ARM
RAM	288 MB	576 MB	512 MB
Wifi	802.11b/g	802.11b/g	802.11b/g/n
Bluetooth	2.0	2.1	3.0
Released	2009	2010	late 2010
Description	The HTC Hero is the first amongst HTC's Android-based products to feature multi-touch capability. Being an older device, this represents the new low-end, cheap devices on the market.	The Desire is a forerunner for many other devices with its fast 1 GHz processor and high density display.	The Galaxy Tab is of similar specifications as the HTC Desire, but its screen resolution yields new opportunities for what can be executed on an Android-based device.

Table 8.1. Shows a comparison of the three devices.

8.3 Android Platform

When developing a desktop application, one is largely unaware of other applications that may be running on the computer. From the application's point of view, it has its own world, and communication with other applications is mostly done through an API.

The Android platform is made up from a number of similar concepts, although packaged and structured differently to better suit mobile devices. This section will provide a quick overview of the concepts used in aSchedule to ease the understanding of the implementation part of this report.

Activities An activity is the Android analogue of a window or dialog in a desktop application. The Android platform is designed to support many short-lived, lightweight activities that the user can bring back and forth by clicking the back-button on the device, much like in a web browser. This concept will be further elaborated in Section 8.4

Services Activities are designed to be short-lived and can shut down at any time. Therefore, Android has the concept of services which are designed to keep running (if needed) independent of any activity. Often, services are used to perform background updates, although it may also be used to expose an API for other applications on the device.

Content Providers Content providers provide a level of abstraction for data stored on the device that is accessible by multiple applications. This allows one as a developer to control how the data gets accessed by other applications.

Intents Intents are a form of messages used in the Android system. The intents are used to notify applications of events, e.g. hardware state changes. The Android platform allows a developer both to listen to an intent as well as sending an intent. Intents are used to start activities, as defined in an application manifest.

Views Views are the basic building blocks of Android user interfaces. Essentially, a view is an element which occupies a rectangular area on the screen and is responsible for drawing and event handling. In Android, `View` is the base class of many subclasses, so called Widgets, like buttons or text fields.

View Groups View groups are containers for views. The `ViewGroup` class is the base class of many different layout classes, each corresponding to a different layout architecture such as a linear or tabular layout. View groups can be nested to create a view hierarchy and thus advanced user interfaces can be created by layering and positioning views according to each other. Layouts in Android are supported by Extensible Markup Language (XML) files which define the hierarchy of view groups and views for a given activity.

Storage An Android application can be packaged with data files for items that do not change, e.g. icon and image files. These are called resources. Further, most devices have a storage card attached which can be used to store larger files such as pictures, and databases or settings files that must persist e.g. system updates. Android provides a default location for applications to store settings and database files. Settings are easily stored using classes in the Android platform. It is however important to note that both settings and databases, as provided by the Android platform, are private to the individual application.

8.4 Activity Life Cycle

An Android device typically runs several applications at the same time and each application typically consists of multiple activities. Each time a new activity starts, the previous one is stopped although preserved in a stack (called the "back stack"). This stack is implemented using the Last In, First Out (LIFO) principle. This means that when the user of the device presses the "back" button, the current activity is popped from the stack and the previous activity resumes.

When an activity is stopped because a new one starts, it is notified of this state change through the activity life cycle callbacks. Depending on the specific state change, there are several callback methods that an activity might receive to be able to perform any appropriate work and ensure correct functionality of the application and system. For instance, when an activity is stopped it should release any large objects, such as database connections. If the activity is resumed again, it should be able to restore any required resources.

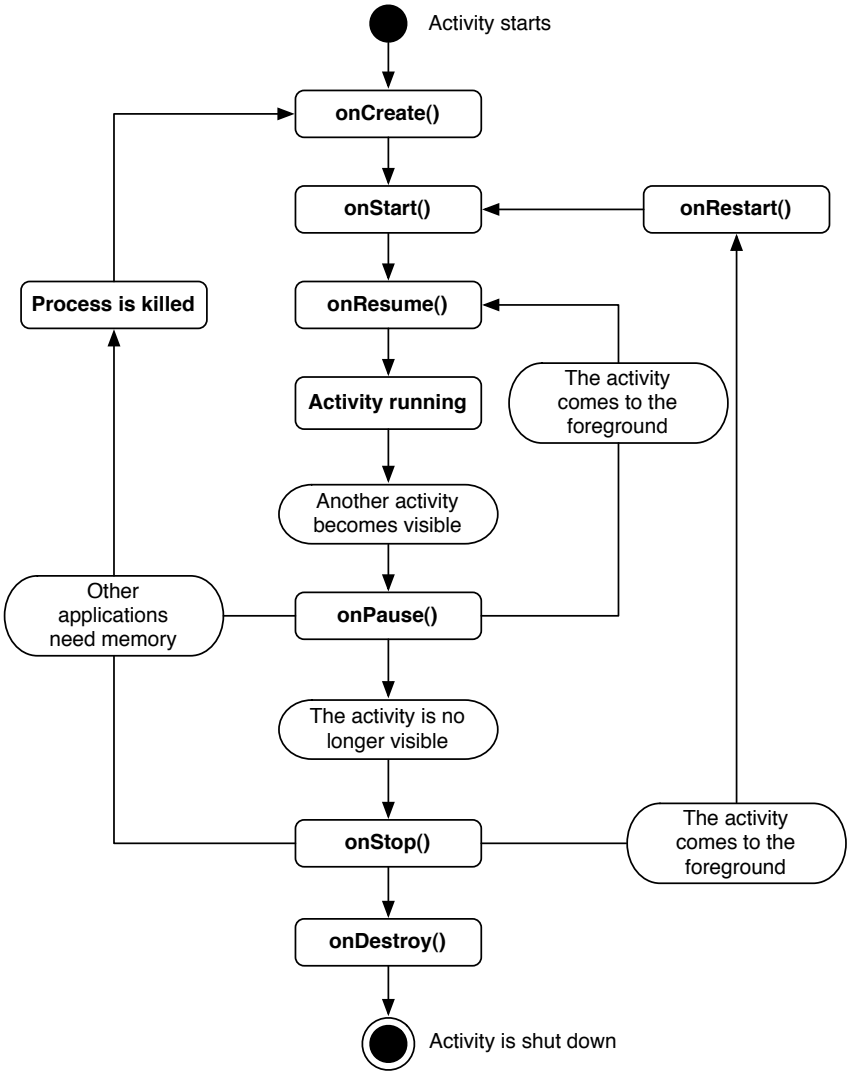


Figure 8.2. The Android activity life cycle.

An activity can basically be in three states (the entire life cycle can be seen in Figure 8.2) [Goo11a]:

Running The activity is in the foreground and has focus.

Paused Another activity is in the foreground on top of this activity, but this activity is still partially visible.

Stopped The activity is completely covered by another activity running on top of it.

These states are supported by a number of callback methods seen in Table 8.2 that gives the opportunity to save and restore the state before state changes.

In Table 8.2 it is important to note the column *Activity killable after this*. The first callback method with this attribute set to yes is `onPause()`. This means that no life cycle callback methods are guaranteed to be run after this method in case the device needs to free memory. Therefore the `onPause()` method must be used to persist any data which cannot be recovered at a later point of execution. At the same time, to provide a smooth user experience, the method should have a short execution time as it must be finished before the next activity can start.

Method	Description	Activity killable after this
`onCreate(Bundle)`	Called when the activity is initially created.	No
`onRestart()`	Called after the activity is stopped, right before being started again.	No
`onStart()`	Called just before the activity becomes visible to the user.	No
`onResume()`	Called just before the activity interacts with the user. The activity is visible at this point and on top of the "back stack".	No
`onPause()`	Called when the system is about to start another activity.	Yes
`onStop()`	Called when the activity is no longer visible to the user.	Yes
`onDestroy()`	Called when the activity is destroyed. The activity cannot resume from this point.	Yes

Table 8.2. Overview of the Android activity life cycle callback methods.

8.4.1 Saving the Transient State of an Activity

As the starting point, when an activity is either paused or stopped, it is still kept in memory and thus easily resumable. However, when the system destroys an activity to recover memory, the system cannot simply resume the activity afterwards because of lost information. Instead, the activity must be recreated if necessary.

An additional callback method, `onSaveInstanceState(Bundle)`, can be used to save any transient state information about the activity. The callback method is passed a Bundle (a mapping from string values to *parcelable* data types, i.e. data types that can be transferred inside the Android system) which can be used to store information in. If the system kills the activity and it is later created again, the system passes the same Bundle to the `onCreate()` method so that the transient state can be restored. The same system is used when starting a new related activity where you need to pass some data to. Android does this by letting you create a bundle attached to the intent, to bundle data. This should however be used with care, as referenced objects may not be accessible by the newly created activity.

It is important to note that the `onSaveInstanceState(Bundle)` callback is not guaranteed to be called and thus it should only be used for transient information. Any data that must be persisted (e.g. to a database) must be saved in the `onPause()` method.

To test an applications ability of managing the activity life cycle correctly we have developed a *Notification Monkey*, which is described in Chapter 14.

Program Presentation

9

This chapter will present GIRAF and aSchedule, in order to give a better understanding of the implementation details presented in the following chapters.

9.1 The GIRAF System

GIRAF is a system designed for children with autism. As described in Section 6.1.1, GIRAF consists of a number of modules. These modules will be presented by walking through a typical usage scenario. The scenario includes the following actions:

- GIRAF is installed on the device,
- settings are applied,
- an application is installed,
- the application is used and eventually uninstalled.

9.1.1 Install and Setup

GIRAF is installed by navigating an Android device to `http://girafplace.lcdev.dk/start/` and downloading the APK installer file. When starting GIRAF for the first time, the user is presented with a dialog, suggesting the user to set up the application before starting using it, see Figure 9.1.

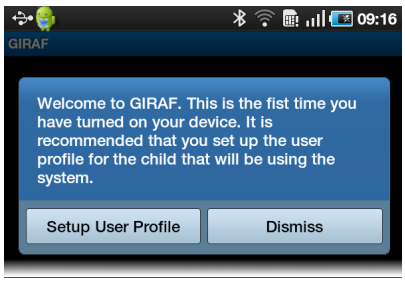


Figure 9.1. The initial window of GIRAF. This provides the user with a setup-guide.

By clicking the button labeled *Setup User Profile*, the settings screen for the user profile is shown. This is depicted in Figure 9.2. The user profile is used by the Application Launcher and GIRAF Place to filter what applications are available. Additionally, applications can modify their appearance and behavior depending on the capabilities of the child, defined in the user profile.

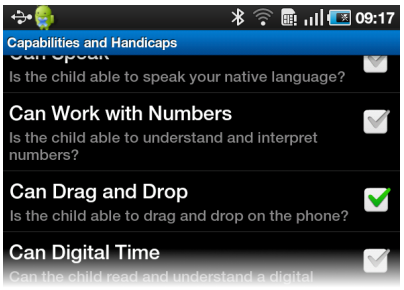


Figure 9.2. The user profile settings. This provides a way of defining the user's capabilities.

When the user profile is set up, the next thing to do is to install applications for the child to use. The Administration Interface allows guardians to start the GIRAF Place to browse and install applications. In Figure 9.3, the aSchedule application has been chosen and is about to be installed.

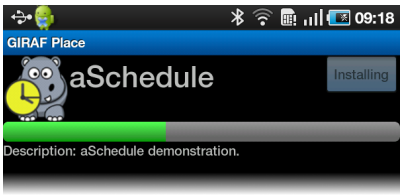


Figure 9.3. Installing the aSchedule application.

9.1.2 Using GIRAF

When applications are installed, guardians can change various settings from within the Administration Interface, see Figure 9.4. This means that guardians can modify all aspects of GIRAF from a single place, because applications, such as aSchedule, provide settings for Administration Interface to show.

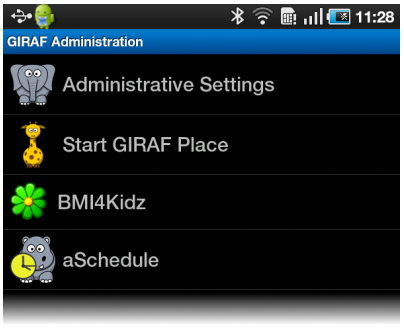


Figure 9.4. All settings can be modified from the Administration Interface, including settings of installed applications. Here the settings of aSchedule and the test application BMI4Kidz are accessible, as these are installed using GIRAF Place.

The Application Launcher is used to start the installed applications by the child, as seen in Figure 9.5. The child is only supposed to access the Application Launcher, and the installed applications. When the guardian needs to access Administration Interface, it can be started by pressing the following key combination: `<Volume Up> <Volume Down> <Volume Up> <Volume Down> <Volume Up> <Volume Down>`.

Figure 9.5. From Application Launcher installed applications can be started. In this case the applications aSchedule and BMI4Kidz are installed.

9.2 aSchedule

This section will provide a presentation of the application aSchedule; developed by the authors of this report. The presentation will follow the same strategy as the previous section describing GIRAF, where a walk through was presented.

The following scenarios will be included in this walk through:

- Starting from the guardian's point of view, an activity is first created.
- Subsequently, this activity will be scheduled for today through the planning tab.
- Switching to the child's point of view, the activity is displayed on the schedule.
- The child then performs the activity by following the attached steps.
- Finally the child needs to access a spontaneous activity.

9.2.1 Activity Creation and Management

Through the Administration Interface, the guardian accesses aSchedule's activity administration side. From here it is possible to create new activities and schedule them to specific dates. Figure 9.6 shows the initial view of the administration side of aSchedule. As the figure shows, three activities have already been created.

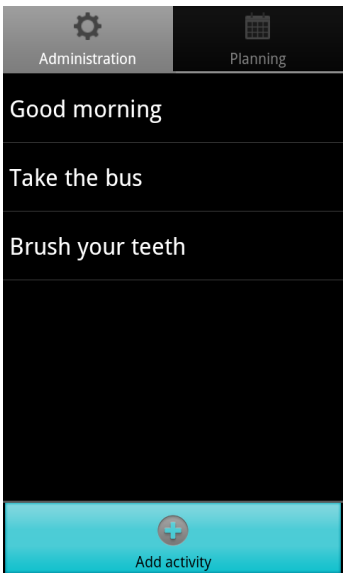


Figure 9.6. The initial view of the administration side, containing three previously created activities.

To create a new activity, the user has to click "menu" and then press the "Add Activity" option. This will present the guardian with a screen where the properties of the activities can be defined, shown in Figure 9.7. Here the properties of the activity has already been filled in; the activity concerns the task of washing hands, hence the name "Wash hands". Further an image has been specified and the activity is set to be available in the spontaneous activity menu.

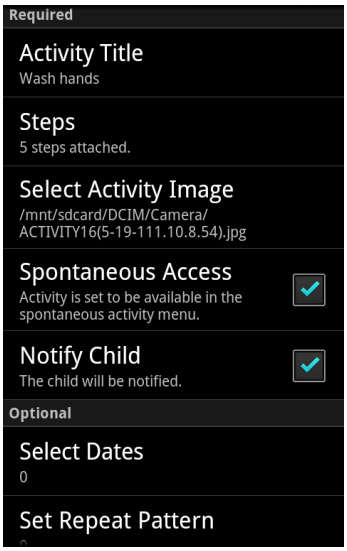


Figure 9.7. Shows the creation of an activity.

Notice that five steps are already attached to the activity. These have been created using the step creator screen, which can be accessed by clicking the list item. Doing so will make a list of all the steps appear. From here the steps can be rearranged and new steps can be attached. The creation of a new step is shown in Figure 9.8. As with the activity, the properties of this step have been filled in.

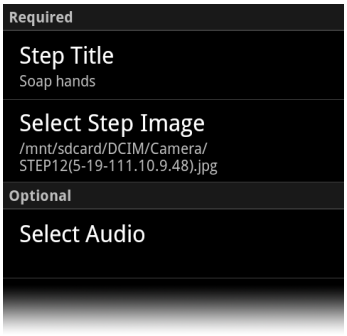


Figure 9.8. Shows the creation of a step called "Soap hands".

When the activity has been created, it is not displayed in the child's view of the application. This is due to the the fact that the activity has not been scheduled yet. This can be done from two places in the application. The first being the previously displayed screen in Figure 9.7, and the second being from the planning screen. In the create activity screen the guardian is able to both attach specific dates, and repeat patterns. The planning screen on the other hand, only supports specific dates, though a more user friendly and intuitive way is provided here.

In Figure 9.9 the process of applying an activity to a given date is shown. In the left image, the planning screen is presented. At this time two activities have been scheduled; "Good morning" and "Take the bus". To add an additional activity, the menu in the bottom of the left image, is needed. The menu shows the different functions available, and by pressing the highlighted option "Add activity to this date", an activity can be scheduled. When the option is pressed, the dialog, presented in the middle image of Figure 9.9, is shown. This enables the user to select an activity, from a list of all available activities, which is to be scheduled. When an activity has been selected, a new dialog is shown, as seen the image to the right. Here the guardian needs to specify at which time of the day the activity is to be added.

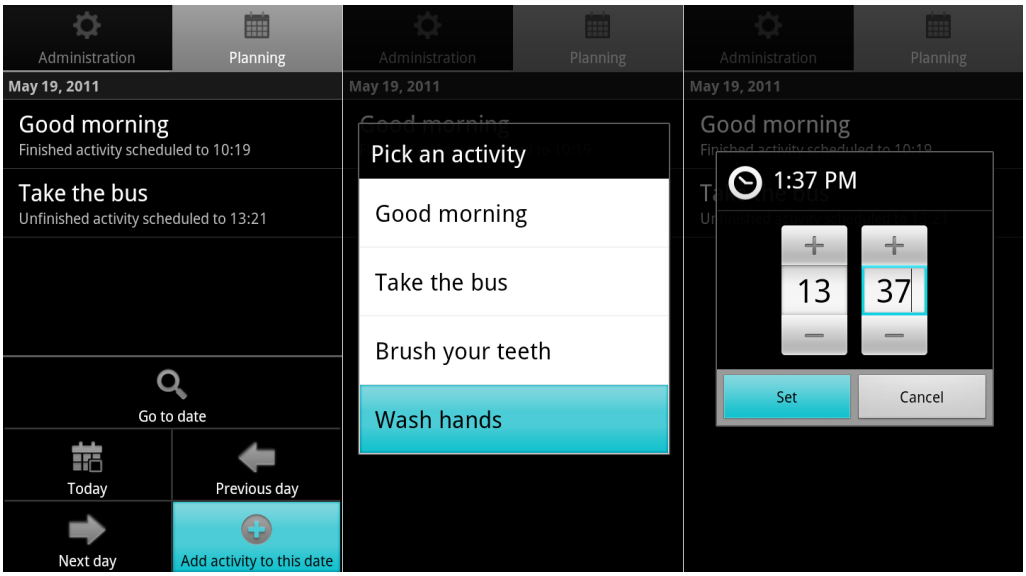


Figure 9.9. A compilation of the steps to pass when scheduling an activity, through the planning screen.

Finally, by pressing "set", the activity will be available for the child at the specified time, 13:37.

9.2.2 The Child's View

This section will concern aSchedule as seen from the child's perspective. The initial screen which the child will see is a overview of the activities of "today". As seen in Figure 9.10, two activities are at this point ready to be performed. Notice that, what the figure doesn't show is that both activities are pulsating to indicate that they are, at this point in time, ready to be performed.

Figure 9.10. Shows the child's view of the schedule.

By dragging one of the activities, a box, to the right of the screen, will appear. If the dragged activity is placed on this, the steps of the activity will be shown, see Figure 9.11.

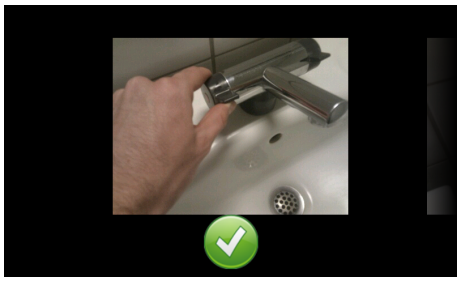


Figure 9.11. Shows the steps of an activity.

The round button with the check mark, in the bottom is used for when the child has completed the step. This will advance the screen to the next step, and so on until the last step. When this is completed, aSchedule will return to the previously displayed list of activities, though now the activity will be placed in the right-hand side of the screen, indicating it has been finished. The child can then pick the next activity, given its ready to be performed.

An additional feature accessible to the child is the list of spontaneous activities. Notice the little arrow in the right-hand side of the screen on Figure 9.10. Dragging this to the left will present the child with a list of activities which are marked spontaneous. This is shown in Figure 9.12.

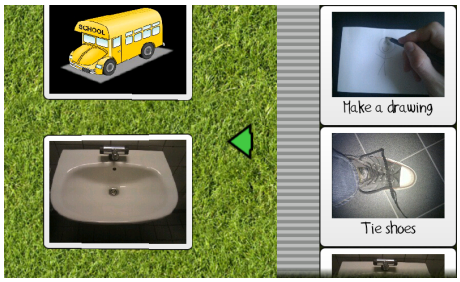


Figure 9.12. Shows the list of spontaneous activities.

By clicking either of these, the child will be met with the previously described list of steps, as presented in Figure 9.11. The difference being that they will still be available when the child returns to the list of activities, and no information on the child finishing the activity is stored. This means that guardians, or the child itself, may start spontaneous activities when a situation spontaneously arises, that the child needs guidance for executing.

The remaining feature to be presented is notifications. When an activity, which is set to notify, is ready to be executed, a notification dialog will pop up, regardless of which application is currently running. An example of such a notification is seen in Figure 9.13. An activity notification can either be dismissed or accepted, if accepted, the aSchedule application opens.

Figure 9.13. A notification is shown, when it is time to do an activity.

Application Database

10

aSchedule has a database used to persist the entities that are represented in the visual schedule. This chapter will provide details on which technology and techniques have been used to implement this.

10.1 SQLite

SQLite is a Database Management System (DBMS) and as the name suggests it uses a dialect of Structured Query Language (SQL). This includes "`SELECT`" for queries, "`UPDATE`" and "`INSERT`" for data manipulation, and "`CREATE TABLE`", etc, for data definition. Like most DBMS's it has its deviations from the SQL92 standard. The main advantage of SQLite is that it is compact enough to run to its full extend on the Android platform. Though this comes with a price of a limited support of data typing, joins, trigger support, and table altering. Most notable is the missing data typing support. While it is possible to choose a data type for a column in a `CREATE TABLE` statement, SQLite does not enforce this type upon data manipulation. This means that it is possible to store a string value in an integer column, and vice versa. SQLite defines this as "manifest typing" [SQL11]:

> "In manifest typing, the data type is a property of the value itself, not of the column in which the value is stored. SQLite thus allows the user to store any value of any data type into any column regardless of the declared type of the column."

The authors of SQLite argues that this is by design and that the use of manifest typing has proven in practice to make SQLite more reliable and easier to use, especially when used with dynamically typed programming languages such as PHP or Python.

Despite its limitations compared to the SQL92 standard, SQLite serves as a solid foundation of the data storage in aSchedule. Since Android 2.2, the Android Platform has been bundled with SQLite version 3.6.19, which introduces foreign keys. These will inevitably be useful to enforce data integrity in the application.

10.2 Repository Pattern

Many applications uses some sort of data storage, such as databases or web services. If the data storage is accessed directly within the business logic, it can result in one or more of the following:

- Duplicated code.
- Higher potential of programming errors.
- Weak typing of entities.
- Difficulty in centralizing data-related policies like caching.
- Inability to test business logic independently from the data storage.

A design pattern is a general, reusable solution to a common software engineering problem, such as the one just described. A pattern is a template for solving a particular problem and thus cannot be transformed directly into code. Thinking in patterns allows one to define high-level solutions that are independent of implementation details. Using established patterns leads to solutions that are more easily comprehensible, testable and maintainable.

The repository pattern is a design pattern used to prevent the above mentioned problems. This is done by following these objectives [HM11]:

- Isolation of the data layer to better support unit testing.
- Centralized consistent and managed data access rules and logic.
- Improvement of the code maintainability and readability by separating business logic from data access logic.
- Strongly typed business entities where problems can be identified at compile time rather than run time.

The way to do this is to use a repository, which separates the logic that retrieves the data and maps it against the business entities, from the business logic that acts on the model. The business logic should be unaware of the source of the data, e.g. it should not matter if data are retrieved from a database or web service.

The repository mediates between the data layer and the business layers of the application. It queries the data source for the data, maps it to a business entity, and persists changes in the business entity to the data source. This is depicted in Figure 10.1. In this way, the repository pattern is a specialization of the Facade pattern - it takes a complex system (persistence to the database) and wraps it in a simpler interface (as a collection). Further, the repository pattern adds persistence ignorance to the solution. This is an attribute that we strive for in a solution where we want low coupling between subsystems. The users of the repositories should be completely unaware of how and where data is persisted; they should just expect it to happen.

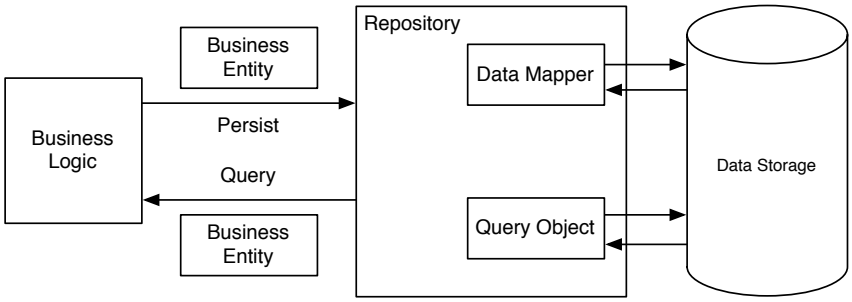


Figure 10.1. The interaction of the repository pattern.

The repository pattern, like any pattern, increases the level of abstraction of the code. This may make the code more difficult to understand to any developer unfamiliar with the pattern. Although the pattern generally reduces the amount of redundant code, it has a tendency to increase the number of classes that must be maintained.

Since we are only three developers on aSchedule, we do not see it as a negative asset that it may be more difficult to understand for developers unfamiliar with the pattern.

10.3 Object Relational Mapping

The basic building block of any object-oriented programming language, like Java, is the object. As such, an application is more or less a large collection of objects which interact. In its own world, this works perfectly

fine, but when joined with a completely different paradigm, such as that of relational databases, the beauty of any object-oriented solution quickly vanishes.

The basic purpose of ORM is to allow an application written in an object-oriented programming language to interact with the information it manipulates in terms of objects, rather than any database-specific concepts like rows or columns.

At a high level, one needs to perform the following activities to use an ORM framework:

- Design the domain model as plain objects.
- Derive the database schema from the domain model.
- Create a mapping description of how to map between the two.

Like many techniques or frameworks, one has to bare in mind that introducing an ORM is not necessarily a good choice for the project. Given the above list of tasks that must be done, one can argue that an ORM does in fact burden the development process unnecessarily.

From the list, the first item might be accepted since the domain model may be known to the project. In most cases, this domain model can be represented in a relational database. However, to derive a database schema from a good domain model does not necessarily result in an optimal database schema. Thus, the introduction of ORM can potentially yield new problems such as an inefficient database. Any persistence issue, when using ORM, is most likely more difficult to find and resolve, due to the higher level of abstraction. There is no direct access to the raw queries that are performed against the DBMS, and no easy way to modify these.

Though, even with these considerations in mind, we do feel that it would be beneficial to introduce ORM in the development of aSchedule. The most minimalist framework available, which even has a Android specific package, is ORMLite. ORMLite is an open-source ORM framework providing simple, lightweight functionality for persisting Java objects to SQL databases without adding the complexity and overhead of many other ORM frameworks (such as Hibernate). Further, ORMLite uses the native Android API's to access databases which makes us more confident that it will function correctly on the platform.

The only lacking functionality of ORMLite, regarding aSchedule, is database cascades. This can, however, easily be managed from Java code so this is not considered a viable reason for not using it (as shown later in Listing 10.6).

ORMLite works by letting you annotate classes that defines how you want the object mapped against the database, this is illustrated in Listing 10.1, together with a connection management system.

```
1  @DatabaseTable(tableName = "accounts")
2  public class Account {
3    @DatabaseField(id = true)
4    private String name;
5
6    @DatabaseField(canBeNull = false)
7    private String password;
8
9    Account() {
10     // all persisted classes must have a no-argument constructor with package visiblity.
11   }
12 }
```

Listing 10.1. A class Account annotated with ORMLite annotations.

As mentioned, ORMLite provides connection management as well as other helping utilities, which will be introduced in the following section about the actual implementation.

10.4 Implementation

Data storage is a central aspect of the aSchedule application and many thoughts are required to come up with a good, maintainable solution. This section will describe our realization of the technologies and techniques described previously.

Our implementation is built upon the functionality provided by ORMLite using a repository pattern to access a SQLite database.

This yields a solution where:

- an `OrmLiteSqliteOpenHelper` is implemented to create, upgrade, and seed the database.
- a number of annotated entity classes defines the desired database mapping.
- an abstract repository class `AbstractDataRepo<T>` is implemented to provide basic Create, Retrieve, Update, Delete (CRUD) data operations.
- a number of concrete repository classes are implemented for each entity aggregate.

10.4.1 Implementation of `OrmLiteSqliteOpenHelper`

The abstract `OrmLiteSqliteOpenHelper` class comes from ORMLite's Android package and overrides a number of methods from the abstract `android.database.sqlite.SQLiteOpenHelper` from the Android framework. This means that it does not replace the built-in SQLite implementation in Android but rather extends it.

Most notably, this abstract class provides a new method which is used by the remaining ORMLite implementation: `getDao(Class<T>)` which returns a Data-Access Object (DAO) object for a specific entity.

The `SQLiteOpenHelper` is meant to be sub-classed in order to specify a database version number and the methods `onCreate(SQLiteDatabase, ConnectionSource)` and `onUpgrade(SQLiteDatabase, ConnectionSource, int, int)`. These methods are called accordingly by the SQLite implementation when the database is accessed within an application. This controls the flow of creating and upgrading the database file in the correct directory. In Android, SQLite databases are private to the individual application and will correspondingly be deleted when an application is deleted.

Our implementation of the `OrmLiteSqliteOpenHelper` is called `DatabaseHelper` and has the following implementations of create and update behavior.

```
1 @Override
2 public void onCreate(final SQLiteDatabase sqliteDatabase, final ConnectionSource
      connection) {
3   TableUtils.createTable(connection, ScheduleActivity.class);
4   TableUtils.createTable(connection, Step.class);
5   TableUtils.createTable(connection, DbDate.class);
6   TableUtils.createTable(connection, RepeatPattern.class);
7   TableUtils.createTable(connection, ActivityStatus.class);
8 }
```

Listing 10.2. The `onCreate(SQLiteDatabase, ConnectionSource)` method of `DatabaseHelper`.

```
@Override
public void onUpgrade(final SQLiteDatabase sqliteDatabase, final ConnectionSource ↵
    connection, final int oldVersion, final int newVersion) {
  TableUtils.dropTable(connection, ScheduleActivity.class, true);
  TableUtils.dropTable(connection, Step.class, true);
  TableUtils.dropTable(connection, DbDate.class, true);
  TableUtils.dropTable(connection, RepeatPattern.class, true);
  TableUtils.dropTable(connection, ActivityStatus.class, true);

  onCreate(sqliteDatabase, connection);
}
```

Listing 10.3. The `onUpgrade(SQLiteDatabase, ConnectionSource, int, int)` method of `DatabaseHelper`.

As it can be seen from Listing 10.2 and Listing 10.3, the behavior we have implemented as of now is that tables are dropped on upgrade before being recreated.

This behavior reflects the status of the project right now, where the application has not been deployed on any devices, and no upgrades have been made. When an upgrade to aSchedule's database structure is made, the `onUpgrade(SQLiteDatabase, ConnectionSource, int, int)` method should be altered to consider the parameters `oldVersion` and `newVersion`. From these parameters, a sequence of operations should be made on the database to transform it into the new layout. It should be noted though, that SQLite's `ALTER TABLE` command only supports a subset of the functionality defined by SQL92. Amongst others, it is not possible to rename or remove a column. Neither is it possible to add or remove any constraints from a table.

From ORMLite's point of view, the importance here is to call `TableUtils.createTable(ConnectionSource, ↵ Class<T>)` which tells ORMLite that we want object mapping of this entity.

10.4.2 Annotated Entity Classes

As illustrated in Listing 10.1, our entity classes have been annotated with the ORMLite annotations in order to allow them to be persisted to the database.

These annotations informs ORMLite of the desired mapping behavior, e.g. database field name, index, or nullable.

An example of an entity in aSchedule is the `ScheduleActivity` entity, shown partly in Listing 10.4.

```
@DatabaseTable(tableName = "activity")
public class ScheduleActivity implements Comparable<ScheduleActivity> {
  public static final String ID_FIELD = "id";
  public static final String TITLE_FIELD = "title";
  ...
  public static final String NOTIFICATION_TIME_FIELD = "notification_time";

  ScheduleActivity() { }

  public ScheduleActivity(final String title) {
    mTitle = title;
    mSpontaneous = false; // default value
    mNotificationTime = 10; // default value
  }

  @DatabaseField(columnName = ID_FIELD, generatedId = true)
  private Integer mId;
  @DatabaseField(columnName = TITLE_FIELD, canBeNull = false)
  private String mTitle;
  @ForeignCollectionField(eager = false)
  private ForeignCollection<Step> mSteps;
  ...
  private ActivityStatusType mInstanceStatus;
  ...
  public Integer getId() {
    return mId;
  }
  public String getTitle() {
```

```
29      return mTitle;
30    }
31    public void setTitle(final String title) {
32      mTitle = title;
33    }
34
35    ...
36  }
```

Listing 10.4. The `ScheduleActivity` entity class.

We decided to use a set of static strings to define the database field names we wanted. These proved useful also when writing the repository classes, described in the following sections.

By defining the desired mapping, we also have the possibility of having non-persisted fields in the entities. Our `ScheduleActivity` entity has an instance status which is an individual object's status (new, skipped or finished). This property is populated at runtime when a specific day's activities are fetched from the database. This relates to the fact that we have decided to think of activities as templates, which are realized by a number of dates and repeat patterns. These are matched on runtime against a specific date, so we do not need to store multiple copies of an activity in the database.

All of the fields of the entity classes are made available to other objects through getters and setters. This design will be taken advantage of when attaching images to activities and steps as described in Section 11.3.

10.4.3 The Abstract Repository Class `AbstractDataRepo<T>`

To give basic CRUD functionality to each of the repositories that are implemented, they are all based on the `AbstractDataRepo<T>` class which is shown in Listing 10.5. The class itself is an extension of `OrmLiteBaseService<DatabaseHelper>` which provides basic ORMLite functionality, such as getting access to the DAO object. Notice that the `OrmLiteBaseService<T>` class is provided with the type of `DatabaseHelper` which ties our create and update behavior and connection management to ORMLite.

```
1   public abstract class AbstractDataRepo<T> extends OrmLiteBaseService<DatabaseHelper> {
2     private final Class<T> mType;
3     private Dao<T, Integer> mDao;
4     private static final String LOG_NAME = "AbstractDataRepo";
5
6     public AbstractDataRepo(final Class<T> typeT) {
7       super();
8       mType = typeT;
9     }
10
11    public T getById(final int itemId) {
12      T result = null;
13
14      try {
15        result = getDao().queryForId(itemId);
16      } catch (SQLException e) {
17        Log.e(LOG_NAME, e.getMessage());
18      }
19
20      return result;
21    }
22
23    public List<T> getAll() { ... }
24    public void delete(final T obj) { ... }
25    public void delete(final int itemId) { ... }
26    public void update(final T obj) { ... }
27    public void create(final T obj) { ... }
28
29    protected Dao<T, Integer> getDao() {
30      if (mDao == null) {
31        try {
32          mDao = getHelper().getDao(mType);
33        } catch (SQLException e) {
34          Log.e(LOG_NAME, e.getMessage());
```

```
35        }
36      }
37
38      return mDao;
39    }
40
41    ...
42  }
```

Listing 10.5. The abstract `AbstractDataRepo<T>` class.

Besides providing the same basic functionality for each concrete repository it catches the SQL exceptions that might occur during runtime. This is based on the idea of persistence ignorance, where our repository consumers should not be aware of how data is persisted; it should just work. The exceptions here are caught and logged to the error logging mechanism on the device, which allows us to see the exception details.

Last but not least, the class caches the type-specific DAO. This ensures the same instance is used from the repository consumers and thus resources are saved.

10.4.4 Concrete Repository Classes

In aSchedule we have implemented one repository per entity. This is not exactly how the pattern is defined, since the pattern suggests that there should be one repository per aggregate root in the domain. Though, we found that the aSchedule application needs the basic CRUD functionality for each of its entity types. Therefore we implemented a concrete repository for each type, to benefit from the functionality provided by the `AbstractDataRepo<T>` class.

As an example, the `ScheduleActivityRepo` is shown in Listing 10.6.

```
1  public class ScheduleActivityRepo extends AbstractDataRepo<ScheduleActivity> {
2    private final StepRepo mStepService;
3    private final DateRepo mDateService;
4    ...
5    private static final String LOG_NAME = "ScheduleActivityService";
6
7    public ScheduleActivityRepo(final Context base) {
8      super(ScheduleActivity.class);
9      this.attachBaseContext(base);
10
11     mStepService = new StepRepo(base);
12     ...
13   }
14
15   @Override
16   public void delete(final ScheduleActivity obj) {
17     try {
18       TransactionManager.callInTransaction(getConnectionSource(), new Callable<Void>() {
19         @Override
20         public Void call() throws Exception {
21           for (Step s : obj.getSteps()) {
22             mStepService.delete(s);
23           }
24
25           for (DbDate d : obj.getDates()) {
26             mDateService.delete(d);
27           }
28
29           mActivityStatusService.delete(obj.getId());
30           superDelete(obj);
31
32           return null;
33         }
34       });
35     } catch (SQLException e) {
36       Log.e(LOG_NAME, e.getMessage());
37     }
38   }
39
```

```
40    protected void superDelete(final ScheduleActivity obj) {
41      super.delete(obj);
42    }
43
44    public List<ScheduleActivity> getActivities(final Date date) {
45      final List<ScheduleActivity> activities = new ArrayList<ScheduleActivity>();
46
47      activities.addAll(mDateService.getActivities(date));
48      activities.addAll(mRepeatPatternService.getActivities(date));
49      Collections.sort(activities);
50
51      final List<ActivityStatus> statusEntries = ↵
            mActivityStatusService.getStatusEntries(date);
52
53      final HashMap<String, Integer> positionMap = new HashMap<String, Integer>();
54      for (int i = 0; i < activities.size(); i++) {
55        positionMap.put(activities.get(i).getId() + ↵
              activities.get(i).getInstanceDate().toString(), i);
56      }
57
58      Integer currPosition;
59      for (ActivityStatus statusEntry : statusEntries) {
60        currPosition = positionMap.get(statusEntry.getActivity().getId() + ↵
              statusEntry.getDateTime().toString());
61        if (currPosition != null) {
62          activities.get(currPosition).setInstanceStatus(statusEntry.getStatus());
63        }
64      }
65
66      return activities;
67    }
68
69      public List<ScheduleActivity> getSpontaneousActivities() { ... }
70  }
```

Listing 10.6. The `ScheduleActivityRepo` class.

Listing 10.6 is a good example of the concrete implementation of a repository. It uses the basic functionality provided by the abstract superclass and extends this with functionality relevant to the entity type.

From the code example, several interesting points can be seen:

- A repository can use other repositories.
- The `delete(ScheduleActivity)` method is overridden from the base functionality.
- A number of entity type relevant query methods are provided.

The repository pattern is all about putting up a facade to simplify access to complex logic. Therefore, there is no reason why one repository should not be able to take advantage of the abstraction provided by another repository.

The usage of other repositories is seen in the overridden `delete(ScheduleActivity)` method. The method has been overridden in order to make up for SQLite's lack of cascades in the database. Therefore, we have to manually clean up related entities in the database when deleting the parent entity. The method ensures data integrity by performing database operations in an atomic transaction while at the same time benefits from having SQL statements sent to the DBMS in a batch rather than one by one.

Finally, it can be seen how complex logic can be encapsulated to the repository consumer in `getActivities(Date)`. The method is used to build the list of activities shown for a particular date. The method retrieves these from both the date and repeat pattern tables, sorts these and populates the results with instance dates and statuses. Further the code example it is a good example that complex logic can be implemented using a high level of abstraction, but potentially at a cost of performance. Since we have implemented the repository pattern, we can easily change the implementation of `getActivities(Date)` at a later point in case we see a need for optimization, e.g. by using a raw SQL statement.

aSchedule

11

aSchedule is the product of this semester project. This chapter will discuss a few of the overall decisions made during development of aSchedule. This includes; to make aSchedule support multiple languages, to manage exceptions, and finally to optimize a certain part of the application which was found to be running slowly.

11.1 Localization

Children with autism, and their parents, are obviously located all around the world. This makes localization an important part of aSchedule, as this will enable aSchedule to reach as many users as possibly.

Android supports the concept of resources (text, audio, images, dates, etc.) and resource-switching based on a large variety of attributes, such as the user's locale.

As a consequence, every text, date and time string is formatted through a localized resource in aSchedule. Text strings are stored in an XML file called strings.xml accessed through the `R.strings` namespace (an example of a strings.xml-file is shown in Listing 11.1). Android provides the capability of placing this, and other resources, in folders that are targeted against specific resource attributes. Further, this provides a separation of actual text from the UI code, since no text constants are present in the Java code.

To demonstrate this capability, aSchedule supports both Danish and English language, with English being the default language for other locales.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <resources>
3     <string name="add_activity">Add activity</string>
4     <string name="remove_activity">Remove activity</string>
5     ...
6 </resources>
```

Listing 11.1. An example strings.xml file.

While this does work, we do not necessarily see it as the best solution to localization. When the need of adding a new localized text string arises, this has to be added to every locale file. The Android toolkit only verifies that a specific string is present in at least one resource file. This means that a localized string may be missing in one language, in which case the default language is used. We feel that it would have been better to give the developer a compile-time error (or at least warning) rather than just hiding it until at run-time.

11.2 Exception Handling

Exceptions are part of most software development projects and are often the root of discussion since many different opinions of best practice exist.

Before starting development of aSchedule, we agreed upon a few simple rules for exception handling.

If an exception cannot be handled it should be left unhandled to give higher layers the chance to handle it. This may mean that no layer handles the exception thus causing a run-time crash. We argue that this behavior is "desired" since no layer was able to handle the exception it must be something not foreseen and thus a critical error.

Any try/catch region should only catch the most specific exception that it handles, to not mistakenly catch a broader range of exceptions than what is handled.

One of the most common mistakes in an application is unhandled null-pointer exceptions, e.g. caused by missing input to a user input field. It makes it easier for the developer to spot potential errors in an application by not hiding exceptions that are not being handled.

11.3 Optimization

Mobile devices are quite different from desktop or web applications in terms of performance requirements. By todays standards, the hardware in mobile devices is still relatively limited. This is due to both heat production and power consumption.

In general, developers should be careful with excessive resource usage. This is even more important when developing applications for a mobile platform. If poorly optimized code is used, not only will this presumably cause a bad user experience, but the battery will also be drained unnecessarily.

During development of aSchedule we experienced the limited performance of the mobile devices we had available.

The aSchedule application itself is packaged with different built-in images for the various pixel densities of the devices' screens. The thought behind this is not to increase the performance, but to support images in a higher quality for screens with a higher pixel density.

The initial screen, which the child sees when opening the application, is made up of a number of draggable images. These images are represented in two linear lists which are scrollable. The images are located on the storage card attached to the device, originating from an external source, or captured with the built-in camera. The images though, are often stored in a resolution several times larger than the screen of the device.

We did not find the problem until quite late in the development process, since we had used a set of low-resolution test images during development. When preparing the application for a demonstration of GIRAF, the problem appeared when we started using photos taken with the built-in camera.

The problem was caused by images being scaled at runtime which caused the list of activities to load very slowly and scrolling to become unsmooth.

To resolve the issue we decided to generate thumbnails of the images in the correct size for the application and device. The functionality has been implemented in a new helper class `ThumbnailHelper`, which is called by a new method `getImageBitmapThumbnail(Context)` method in both the `ScheduleActivity` and `Step` entity classes (as seen in Listing 11.2). This means the functionality is centralized and the UI code just needed slight refactoring to take advantage of the new functionality.

```
1 public Bitmap getImageBitmapThumbnail(final Context ctx) {
2   Bitmap bitmap = ThumbnailHelper.getImageBitmapThumbnail(mImage);
3   if (bitmap == null)
4     bitmap = BitmapFactory.decodeResource(ctx.getResources(), R.drawable.default_image);
5   return bitmap;
6 }
```

Listing 11.2. The `getImageBitmapThumbnail(Context)` method inside `ScheduleActivity` and `Step`.

The helper will return null in case the thumbnail cannot be generated (e.g. if the original file does not exist anymore). This gives the caller the possibility to choose an alternative image.

Thumbnails are maintained by the helper class to ensure thumbnails are removed as soon as they are not needed anymore. The thumbnail generation itself is trivial and will not be documented further in this report.

Implementing this has drastically improved the user experience of the application at the cost of a slightly longer delay when choosing image for an activity or step.

12 Integration

Several subsystems are developed simultaneously in GIRAF which must be integrated. The integration is a critical point in the development process since it will be the first time where early design choices are brought to life. This chapter describes the integration with other subsystems from aSchedule's point of view.

12.1 Common Settings Storage

The Common Settings Storage is basically a content provider which contains both application specific settings, and a user profile. aSchedule uses both of these to adjust its features to better match the users capabilities. Listing 12.1 shows how the `Settings` class is used to retrieve the user profile. Depending on whether the user has reading capabilities the `stepView` makes its `TextView` visible and sets the text to the title of the step entity.

```
1 if (Settings.getUserProfile(mContext).canRead()) {
2   stepView.getTextView().setVisibility(View.VISIBLE);
3   stepView.getTextView().setText(step.getTitle());
4 }
```

Listing 12.1. Shows how the stepview is adjusted to match the users ability.

As for the application specific settings, aSchedule enables the guardian to select a background of choice. This requires a bit more code than the above, but it is still rather simple. The applications of GIRAF is required to contain a file called `settings.xml`. This file is used to specify the settings which should be accessible from the Administration Interface. In Listing 12.2 an excerpt of the `settings.xml` in aSchedule is shown. More precisely, it shows how the background setting is defined.

```
1 <enum varName="Background">
2   <element realName="Black" value="0"/>
3   <element realName="Sky" value="1"/>
4   <element realName="Grass" value="2"/>
5   <element realName="Wood" value="3"/>
6   <element realName="Fur" value="4"/>
7 </enum>
```

Listing 12.2. Shows an excerpt from the `settings.xml` file of aSchedule, concerning the background image.

From the `settings.xml` file, an entry will automatically be generated in the Common Settings Storage and displayed in the Administration Interface. Here, the background of aSchedule can be chosen by the guardian. As shown in Listing 12.3, the value of the setting is then retrieved using the `Settings.getEnum(Context, ↵ String)` method. It is, however, only possible to get the value of the setting, which means that the developer is required to maintain the order of the settings both in the `settings.xml` file and the code.

```
switch ((int)Settings.getEnum(this, "Background")) {
case 0:
  mBackgroundImageView.setBackgroundColor(Color.BLACK);
  break;
case 1:
  mBackgroundImageView.setBackgroundDrawable(getResources().getDrawable(R.drawable.skystyle));
  break;
...
case 4:
  mBackgroundImageView.setBackgroundDrawable(getResources().getDrawable(R.drawable.doggystyle));
  break;
default:
  mBackgroundImageView.setBackgroundColor(Color.BLACK);
  break;
}
```

Listing 12.3. Shows how the background image is applied from the data of the `Settings` library.

12.2 Publishing GIRAF Applications

The publication of applications is done through the subsystem GIRAF Place. This is a system similar to the Android Market, which is the default software store provided in the Android OS.

GIRAF Place provides an interface for the developers to upload their applications. When uploading an application, the developer must specify which capabilities it requires of the user (this list is specified earlier in Section 6.1.2). This specification is later used when the application has been installed and is to be listed in the Application Launcher, further elaborated in Section 12.3. On the device, GIRAF Place is accessed through the Administration Interface. From here new applications can be installed, or currently installed applications can either be uninstalled or updated, given an update is present.

12.3 Application Launcher

The responsibility of the Application Launcher is to provide the child with access to the installed applications on the device. When an application has been installed it will not necessarily be listed in the Application Launcher. If an application is to be shown, it has to comply with two criteria. The application's user requirements must be fulfilled in terms of required user capabilities being selected in the user profile. Taking aSchedule as an example; it specifies that it requires the user to be capable of drag and drop. If the user profile doesn't specify the child the be able to do this, aSchedule will not be shown in the Application Launcher. The second criterion is that applications can have certain WiFi SSIDs blacklisted, meaning that the application will not be shown if the SSID is available (this, for instance, provides the possibility to ban games and other unwanted applications if the child is at school).

The activities of the applications, such as aSchedule, which is to be used as a part of GIRAF, must all inherit the class `GirafActivity` instead of the Android class `Activity`. `GirafActivity` is a class which itself inherits from the Android `Activity` class. The specialized `GirafActivity` defines that activities should run in full screen mode and provides access to the Administration Interface.

Part IV

Test

News about software problems or security breaches is unfortunately not rare incidents in todays modern world: a Mars lander lost in space, or hackers gaining access to millions of credit card numbers in a global gaming console network. As software gets more complex, gains more functionality, and becomes more interconnected, it becomes increasingly complicated to test the software and even mathematically impossible to verify the correctness of said software.

Software testing is an expanding area with many available techniques and continued research. This part will describe some techniques used in the development of GIRAF and aSchedule to strengthen the quality of both. Further, an evaluation of aSchedule by the kindergarten Birken, specialized in children with special needs, is presented.

Unit Testing

13

In most software development models, a code-test-fix based loop can repeat multiple times before software is released. When developing a feature, one may have to run tests a dozen times before it works correctly. Further, when developing a new feature, one have to test that the new feature does not break previously implemented functionality, and thus might have to re-run the tests once again. This is a monotonous and time-consuming process that to a large extend can be replaced by automated tests.

Automated test tools generally provide [Pat05, ch. 15]:

Speed Manually running test cases is a time-consuming task. It is obviously considerably faster to leave this job to a computer.

Efficiency While manually running test cases, the developer/tester is restricted to doing this. It is not possible to do other things at the same time. If tests are automated, there will be more time to plan more or better tests.

Precision The attention span of a human is limited and this might affect the precision of testing.

Trust If a developer works on a feature, it provides a high level of trust in the system, if he can verify instantly that the changes do not break functionality.

Throughout development of aSchedule, a number of unit tests have been written to verify the correctness of the database persistence and model layer. Unit testing is only a small set of the available test automation techniques. Another very relevant technique would be to automate the UI tests of the application. Google has created a tool for the Android platform called *monkeyrunner* to perform automated UI tests. Some experimentation with it showed that it was very time consuming to set up tests of the UI which worked as intended. Further, since the UI has developed gradually throughout the project, we would have had to continuously maintain the tests. Therefore, we decided to revert to manual UI tests.

The unit tests are based on the JUnit framework. The Android SDK provides a number of classes that can be used for unit testing an Android application. Mainly, this is to support the concept of application and activity contexts which have to be simulated during unit testing.

The tests implemented verify the correctness of the persistence and model layer. The tests are all based on the concept of recreating the database before each test, to ensure that an empty database exists at the beginning of each test. This can be seen in the `setUp()` method of Listing 13.1.

```
1  public class ScheduleActivityTest extends AndroidTestCase {
2      private ScheduleActivityRepo mDb;
3      private StepRepo mStepDb;
4      ...
5
6      @Override
7      protected void setUp() throws Exception {
8          super.setUp();
9          DatabaseHelper x = new DatabaseHelper(getContext());
10         x.recreateDatabase();
11         mDb = new ScheduleActivityRepo(getContext());
12         mStepDb = new StepRepo(getContext());
13         ...
14     }
15
16     public void testInsertActivity() {
17         ScheduleActivity orgActivity = new ScheduleActivity("Test Activity");
18         mDb.create(orgActivity);
19         assertNotNull(orgActivity.getId());
20
21         ScheduleActivity newActivity = mDb.getById(orgActivity.getId());
22         assertEquals(orgActivity.getId(), newActivity.getId());
23         assertEquals(orgActivity.getTitle(), newActivity.getTitle());
24     }
25     ... //More test cases
26 }
```

Listing 13.1. An excerpt of the `ScheduleActivityTest` test class.

Listing 13.1 shows a simple test of inserting an activity into the database through the implemented repositories.

Other tests exist of the other repositories in aSchedule, which test functionality like adding steps to an activity, getting activities for a specific date, etc. In total, this adds up to 25 automated tests that have been checked after every development change in the data layer of aSchedule. During the development phase, this added a solid foundation for us to implement the UI on top of. Further, it has served as a reduction in time spent on debugging, since we were able to trust the data storage.

Notification Monkey

14

As described earlier in Section 8.4, managing the activity life cycle correctly is one of the most fundamental tasks when developing Android applications. If not done correctly, the application might crash unexpectedly in situations that can be difficult to spot or even provoke manually. aSchedule includes a notification service that may pop up at any time, notifying the user that it is time to execute an activity. This means that any GIRAF activity may be interrupted, requiring developers to ensure all activities can be resumed correctly.

To aid us, and the other development teams of GIRAF, we developed the *notification monkey*, which is a tool to assist in debugging and verifying management of the activity life cycle. The notification monkey is a background service that can be installed and run like other applications on the device. Once started, the notification monkey will start a dialog on top of other applications every 30 seconds. This causes other applications to change their state, triggering their `onPause()` callback method (and potentially `onStop()`). The dialog, displayed in Figure 14.1, presents two buttons: one to close the dialog and one to shut down the notification monkey completely. Once the dialog is closed, the previous application on the "back stack" is resumed. If the activity cannot resume it means the life cycle is not managed correctly.

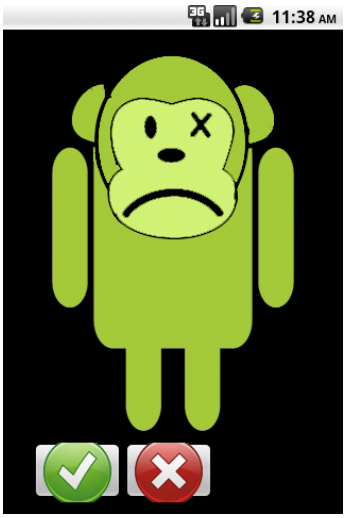


Figure 14.1. The dialog shown by the notification monkey every 30 seconds.

The notification monkey consists of a few classes: `MainActivity` which is started from the launcher to activate the notification monkey, `NotificationService` which is the background service that spawns the dialog `NotificationActivity` every 30 seconds. The `NotificationService` class is shown in Listing 14.1.

```
1  public class NotificationService extends Service {
2    private Timer mNotificationTimer;
3
4    @Override
5    public int onStartCommand(Intent intent, int flags, int startId) {
6      setUpdateTimer(30 * 1000);
7      return START_STICKY;
8    }
9
10   @Override
11   public void onDestroy() {
12     if (mNotificationTimer != null) {
13       mNotificationTimer.cancel();
14       mNotificationTimer = null;
15     }
16   }
17
18   @Override
19   public IBinder onBind(Intent intent) {
20     return null;
21   }
22
23   private void setUpdateTimer(int millisecondsToUpdate) {
24     mNotificationTimer = new Timer();
25     mNotificationTimer.schedule(new NotificationTask(), millisecondsToUpdate);
26   }
27
28   class NotificationTask extends TimerTask {
29     @Override
30     public void run() {
31       Intent intent = new Intent().setClass(NotificationService.this, ↵
             NotificationActivity.class);
32       intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
33       startActivity(intent);
34
35       setUpdateTimer(30 * 1000);
36     }
37   }
38 }
```

Listing 14.1. The `NotificationService` class of Notification Monkey.

As seen in the code example, the primary aspect of the background service is to start a timer which calls a `TimerTask` once its delay expires. A private inner class `NotificationTask` has been created to implement the abstract `TimerTask` class.

To start the `NotificationActivity` dialog, an intent is created and passed to `startActivity(Intent)`. Here, it is important to note that we add the flag `Intent.FLAG_ACTIVITY_NEW_TASK` which means that the activity should become a new task on the stack. Therefore, the dialog is not dependent on the continued execution of the background service.

Integration Testing

15

Throughout the development process, each group has been responsible for testing their individual subsystems, such as aSchedule. At the end of the development process, a committee consisting of a member from each development team, has been responsible for developing and executing an integration test that ensures subsystems integrate into GIRAF, and that GIRAF functions as a complete system. This chapter documents the integration test, as well as the result of it.

Test documents are attached in Appendix B that show the details of each test conducted.

15.1 Strategy

Each group is completely responsible for the the stability and functionality of their individual projects. The starting point of the integration tests were that all subsystems function individually. To test whether all subsystems integrate into GIRAF, dynamic black-box tests have been conducted on multiple devices.

Dynamic black-box testing is using software the way an end user would use it, without looking into the underlying code. The reasoning for this approach is based on the assumption that the part of the system most likely to cause errors is the UI, and the way it is handled by the underlying software. This assumption is based on empirical knowledge gained throughout the development of GIRAF.

15.2 Objectives

As the goal of the test is to ensure all subsystems integrate into GIRAF, the focus has been on testing the interfacing parts of each subsystems. To determine these, the test plan has been based on the flow that can be deducted from the architecture diagram in Figure 6.1, by focusing on how modules interact, and when it happens in the entire life cycle of GIRAF. This has led to the following test objectives. Notice that test objectives are labeled according to the test document they refer to.

TD00 Ensuring GIRAF applications can be deployed to GIRAF Place, and that non-GIRAF applications and corrupted GIRAF applications are rejected.

TD01 Testing initial installation of GIRAF, and that the user is presented with necessary welcome dialogs.

TD02 Ensuring applications can be installed on devices using GIRAF Place. To ensure that the Application Launcher and applications are able to use and store settings in the Common Settings Storage, as well as abiding to the navigation flow for opening the Administration Interface, that has been agreed upon.

TD03 Installing and upgrading GIRAF and GIRAF applications.

TD04 To test the back button behavior of GIRAF and GIRAF applications.

TD05 To test whether the home key handling works as intended.

TD06 To test whether applications can be deleted from GIRAF Place.

TD07 To check if applications are filtered according to user profile.

Notice how these test objectives describe the entire application life cycle in GIRAF. Applications must be added to GIRAF Place, downloaded, started, used, integrated with the system correctly and uninstalled. The goal of the tests has not been to find specific bugs, but rather to ensure that GIRAF works as a complete system, and is fully usable from a user's perspective.

15.3 Execution and Results

The tests were conducted on four different devices, to test on both slow and fast devices, as well as on devices with different resolutions and pixel densities. The test procedures are deliberately written to be slightly vague, in order to make testers try to force GIRAF into failing states themselves. This does however require testers to be both experienced developers and Android users, in order to have a knowledge of how and when Android applications typically fail.

The tests showed that while programs generally worked as intended, many minor issues were present. The bugs and issues discovered are documented in Appendix B, and issue reports were created in the common issue tracker[1] used throughout the project.

[1] Located at http://code.google.com/p/sw6android/issues/list

External Evaluation of aSchedule

16

The aSchedule application has been developed with physical visual schedules as a role model. This chapter highlights key points of aSchedule from private communication with department manager Tove Lund Thomasen from Birken, a kindergarten for children with special needs. aSchedule was explained and presented to Tove, who gave her professional opinion on how a such system can be used and what parts of the system that supplement their current working procedure and the children's everyday life.

16.1 Mobility

Compared to physical visual schedules, aSchedule is very portable. Their current implementation of day planners include numerous schedules placed at different locations in the kindergarten. This has the advantage of "pushing" instructions to the user, e.g. by placing toilet instructions on the door to the toilet. But the missing mobility of the day planner poses a problem when pedagogues and children are e.g. at the playground. It is obvious that very young children would not be able to carry a mobile phone or tablet everywhere, but for older children, this would be a huge advantage.

16.2 Socially Acceptable

Visual Schedules are important aids to many people with autism, even adults. A tablet device or a smartphone is a socially acceptable device in almost any public space. This is an important asset of aSchedule, as it means children in schools are less likely to be picked on for their use of aids, and therefore less likely to give up using them.

16.3 Efficient Activity Administration

The possibility of administering one or more visual schedules using a PC interface would be an advantage for the pedagogues and teachers. An effective administration interface could for instance be used by teachers to plan a complete school timetable with pictures and instructions.

The fact that a mobile device can contain and categorize thousands of images is also a very valuable feature of an electronic visual schedule, as the current process of printing, laminating and categorizing images would be obviated.

16.4 Steps in Activities

Compressing too much information into a physical visual schedule may require a very large board that can potentially confuse the user. By presenting only activities at first, and then presenting steps when an activity is started, complex activities become easier to illustrate. The concept is implemented in physical schedules at Birken, by using multiple schedules "linked" together by an image (e.g. when picking the car image on one schedule, the child finds the car-schedule with additional instructions on it). This does however require the child to understand the system, and it makes the visual schedule even less mobile.

16.5 Limited Flexibility

Compared to physical visual schedules, aSchedule is very inflexible. The physical schedules used at Birken comes in many variations to suit children individually. E.g. some children like to get an overview of all steps involved in an activity, rather than having them presented in a step-wise fashion. Additionally, there are situations where steps are not sequential, for instance if telling a child that a meal consists of both eating and drinking.

Additionally, aSchedule is not efficient at handling mistakes. If a step is marked as finished, it cannot be redone, even though the child mistakenly marked it. The same is the case with finished activities - when done, they cannot be moved back to the left "undone" lane in the activities view. A visual schedule is an aid that children enjoy to use, and there is no real use for controlling and limiting how the child uses it to the extend aSchedule does.

16.6 Limited Support for Developing Skills

While the goal of visual schedules is to support and aid the children with their everyday life, the pedagogues continuously change children's schedules to make them adjust to simpler schedules with less illustrations and instructions. One of the methods is to keep making images smaller and text larger, until the image can be replaced entirely by text. aSchedule has little support for gradually changing the user interface to accommodate this technique, although such is easily extended by the usage of the common user profile.

16.7 Overall Evaluation

Tove's overall impression of aSchedule is that it captures the essence of how visual schedules will be used in the near future. On one hand, efficient administration interface with easy access to media makes the everyday work easier. The children gets a device that is completely mobile, socially acceptable to use, and that supports the way they plan their day. The application obviously has a number of flaws and limitations, especially compared to the flexibility of a physical schedule. The possibility to notify a child of activities as well as providing an understandable overview of the day, no matter where the child is, are some of the advantages that aSchedule already has over physical schedules.

Part V

Recapitulation

The purpose of this part is to reflect on the project. The part will initiate with a reflection upon the development process and on mobile development with the experience gained through the project. Hereafter, Chapter 18 will sum up the results of the project and draw a conclusion to examine whether the results comply with the project and study goals.

Finally, in Chapter 19 some plausible future improvements for GIRAF and aSchedule will be described.

17 Reflection

During development of GIRAF and aSchedule, numerous lessons have been learned, especially because the semester project as a whole contained challenges that we had not encountered before. Developing to mobile platforms, in particular Android, and developing a larger common project in multiple development teams were two of the main challenges. This chapter reflects on some of the choices that were made throughout the project.

17.1 Development Methods

At the beginning of this project we had no prior knowledge of the subject nor the platform of the project, hence every aspect was considered complex. To develop a good solution that would suit the requirements of the users, and work properly on the platform, we had to gain a rather thorough understanding of both before the development could start. To incorporate this, we decided to use OOAD during the beginning of the project to analyze our target group, and define a common agreement on the overall goals of the project. This process was done in close collaboration with the other development teams. The product of this was a well-documented base of knowledge that all groups could utilize in their individual development process.

The traditional approach to requirements engineering, and the "big design up front"-mentality of OOAD generated many ideas of how to solve the problems. However, as the article Principle of Limited Reduction [MS92] states, this introduces a situation in which agile methods are suitable to use for experimenting, in order to reduce uncertainty introduced by traditional research. Therefore, once we felt confident that we had a thorough knowledge of our problem at hand (limited complexity), we switched to an XP process executed within our own development team. This decision was made to speed up our development process while at the same time being very flexible. In a project like this, with multiple collaborating groups, we felt it was necessary to be ready for change at any time. For instance if a group decided to change its goals, we would potentially be forced to adapt to these changes. The agile approach was useful for experimenting with solutions to these problems, hence reducing the uncertainty within the project. As there is no actual customer, XP only introduced little additional complexity that had to be analyzed and understood.

Figure 17.1 illustrates the entire development process, and the idea behind it.

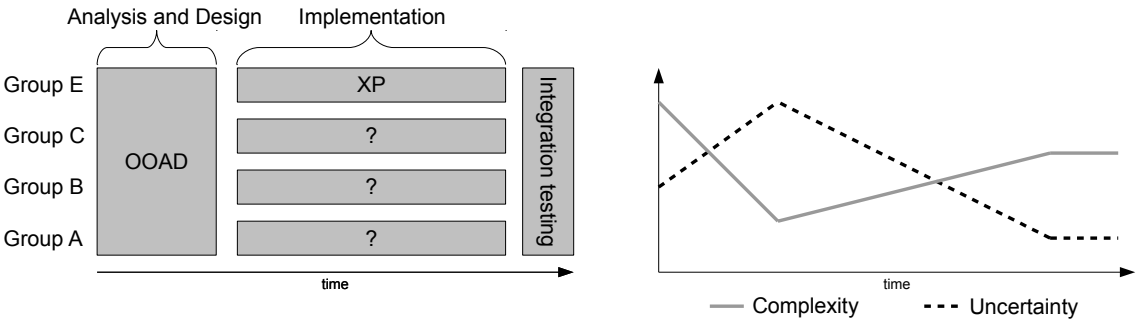


Figure 17.1. Shows the development process of the project. The figure on the left shows that a common OOAD process used to determine core problems of the project was replaced by individual development phases, in our case, as Group E, an XP process consisting of four iterations. The graph on the right shows what was gained from this approach, in terms of the amount of complexity and uncertainty within the project.

The traditional OOAD approach in the beginning of the project proved to be useful for capturing and categorizing the problems of the project, hence reducing complexity. All development teams seemed to agree on the overall goals of the project, and no groups' projects had tendencies to diverge from the rest. However, one of the reasons the OOAD process successfully captured the problems of the project, was that no real customers were involved in the project. This obviously makes it easy to make the analysis correct from the very beginning, as its correctness is defined by the groups themselves. Developing to a "real" customer would most definitely have required the basic understanding of the problem to be revised throughout the development process.

While the OOAD process was effective at reducing complexity within the project, it produced uncertainty in terms of how to actually solve the problems defined by the analytical OOAD approach. Because groups developed individually, we decided from the very beginning to put a lot of attention to make our sub-project as independent of the other parts of the common project as possible. Throughout the development phase, we have had multiple occasions where our suspicion proved to be true, when a group changed their system, or chose to leave out features agreed upon. These situations were quite neatly handled since we were able to quickly adapt to the changes in the next, short XP iteration, with minimal effort because of the low coupling between subsystems. Even though other groups did change their systems, we have not had any time where aSchedule did not work independently. Thus, external changes did not cause our development process to come to a halt.

17.2 Mobile Development and Android Specific Challenges

Developing to a mobile platform introduces different challenges from typical software engineering. Compared to the plethora of programming languages and techniques available on desktop platforms, a mobile platform typically limits one to a single language and methodology defined by the device or platform manufacturer. Compared to many other mobile platforms, Android offers a rather high level of abstraction through the Java API used for developing Android applications. Nonetheless, developing for Android still introduced numerous challenges that were tackled - some better than others.

17.2.1 Separating User Interface and Business Logic

Android applications consists of activities, whose design elements are defined through XML layout files. However, as many activities do not have a static layout, additional view elements has to be added to the activity at runtime. Because the activity is responsible for both reading the layout, defining additional views and populating all the views with data, business logic and UI elements have a tendency to start interweaving.

In VisualSchedule.java, the main drag-and-drop activity of aSchedule, the following example (Listing 17.1) of interweaved business logic and UI manipulation is found:

```
1  if (activity.getInstanceStatus() == ActivityStatusType.FINISHED) {
2    imgLeft.setVisibility(View.INVISIBLE); //imgLeft is a UI element (a View)
3    //User profile fetched from the Common Settings Storage
4    imgRight =
5      new LinkedActivityView(mContext, activity, bitmap, ↵
           Settings.getUserProfile(mContext).canRead());
6  } else {
7    imgRight = new LinkedActivityView(mContext, activity, bitmap, ↵
         Settings.getUserProfile(mContext).canRead(), true);
8    imgRight.setVisibility(View.INVISIBLE);
9  }
```

Listing 17.1. Example of interweaving UI manipulation and business logic found in VisualSchedule.java.

Listing 17.1 also introduces another design problem we encountered. Because a single activity is the entrance to all the functionality seen on a single screen, the sheer amount of code lines can make the class rather large and complex[1]. When we started to encounter this problem, we started to refactor our code by offloading tasks from the activity to the views that encapsulates these tasks. `LinkedActivityView` is an example of such a view. A `LinkedActivityView` represents an `ScheduleActivity` from the model as a white box with the activity image and title. To avoid placing the code that combines these elements into a single view in the main activity, it was placed in the custom view, `LinkedActivityView`.

The problem is that offloading work from the activity to a view requires the application context, `mContext` to be passed to the view. The context is required because the view uses application drawables (images) that are only accessible from the context of aSchedule. These drawables are available through the context since Android offers resolution-specific images, and thus the context is responsible for selecting the correct image.

Passing the application context to other layers in the application poses a few problems. First, it means that UI logic becomes available far down in the layers of an application, which makes it easy to write subpar code. Secondly, it means that a large amount of code easily becomes dependent on the platform and specific context. As an example Listing 17.2 shows how the `ThumbnailHelper` is dependent of the context to provide a fallback image in case an image does not exist or the thumbnail cannot be generated. As was shown in Figure 6.2, the `ThumbnailHelper` is far down in the stack which means many layers must carry on the application context.

```
1  public Bitmap getImageBitmapThumbnail(final Context ctx, final int maxWidth, final int ↵
       maxHeight) {
2    Bitmap bitmap = ThumbnailHelper.getImageBitmapThumbnail(mImage, maxWidth, maxHeight);
3    if (bitmap == null) {
4      bitmap = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(ctx.getResources(), ↵
           R.drawable.note), maxWidth, maxHeight, true);
5    }
6    return bitmap;
7  }
```

Listing 17.2. An example from the `ThumbnailHelper` that shows the dependency of an application context far down in the application layers.

[1] Extremely large and complex activities are actually the norm rather than the exception in the activities in the Android source itself.

17.2.2 Leaking Memory and Limited Heap Size

Passing the context around between classes proved to be a bad design choice. But late in the development process, it became evident that this caused another problem. When views and activities rely on the context of the program, the garbage collector is in many cases not able to remove them from the heap. This did not become evident to us, before late in the development process. The leaks have mostly been eliminated in aSchedule, but few are still remaining in the application's current state. This is evident by a class we have made to log heap usage throughout the applications life cycle. By calling `DebugHelper.logHeap(Class)` (see Listing 17.3 in `onPause()` and `onResume()`, an activity's current heap usage can be read from the device log.

```
1  public static void logHeap(Class c) {
2    Double allocated = new Double(Debug.getNativeHeapAllocatedSize())/1048576.0;
3    Double available = new Double(Debug.getNativeHeapSize())/1048576.0;
4    Double free = new Double(Debug.getNativeHeapFreeSize())/1048576.0;
5    ...
6    Log.d( ... );
```

Listing 17.3. A snippet from `DebugHelper.java`. The method is used to log heap usage in activities.

17.2.3 Debugging Android Code

The `DebugHelper` class in Listing 17.3 was required because the provided Android debugging tools are not very mature, nor do they provide much useful information. For instance, the objects stored in the heap can be seen by the use of the debugger, but their relation to objects and methods within a running application cannot be seen. This makes debugging memory usage very difficult.

Debugging run-time errors in an Android application can be a hit-or-miss experience. When errors occur within ones own classes, the debugger is able to break and show a reasonable exception. But if errors occur in Android-specific methods (e.g. in `onCreate()`, `OnPause()`, etc.), even inside ones own classes, the debugger will break the execution far down in the Android source code, hiding the real problem/exception. This often made non-trivial errors hard to spot during the development of aSchedule.

17.2.4 Android Fragmentation

aSchedule relies on external activities for selecting media, e.g. images and sounds for activities and steps. The plethora of different Android versions, and especially the amount of UI overlays makes the way to fetch media inconsistent across devices. The Samsung Tab (TouchWiz UI) default audio-picker is a music player that is not able to select a file and return a URL, only play songs. Its default Gallery does not always load up completely (doesn't show all files), and occasionally crashes when loading. Requesting URLs from the camera is done completely different than requesting a URL from any other Media Picker in Android. Even though workarounds exist for all these problems, it clearly shows why many Android developers criticize the OS for its fragmentation[2].

[2]Searching Google for "Android Fragmentation" shows numerous discussions on this topic, and examples of developers having to release different versions of an application because of fragmentation issues.

17.2.5 Application Speed Regarding User Experience

One of the interesting challenges have been to create an interface as smooth and polished as an Android user expects. Android devices are extremely powerful compared to mobile phones only few generations older, but a heavy UI with lots of animations makes good use of the computational power. Especially when creating very customized activities, such as the main view the child sees for selecting activities, we saw slowdowns in the UI. Utilizing thumbnails, preloading and reusing UI elements have been some of the key elements to making the user experience acceptable.

The entire concept of optimizing UIs has been a new experience to us as developers. A slow UI in a desktop application has always been caused by inefficient algorithms or doing I/O in a UI thread - the problem has never been the UI itself.

Conclusion

18

In this project an Android application called aSchedule has been developed as part of a larger system called GIRAF, developed in collaboration between four software engineering groups at Aalborg University.

The overall system GIRAF is an OS overlay for Android-based devices, such as smartphones and tablets. The system should aid and entertain children with limited mental capabilities in their everyday lives. The children's guardians should be able to administer the system by controlling selected application-, system- and user-specific settings through an common administration interface on the device.

The product developed by this group, called aSchedule, is a visual schedule application. This application has been developed to aid children, having difficulties managing their everyday activities without guidance. The system is based on existing principles of visual schedules, and aims at transferring these to an existing mobile platform.

Throughout development of aSchedule, it has been our intention to develop an application which is almost ready for use by the intended end-users. Even though it is not covered extensively in the report, it has taken a central role in the development process to decide on UIs through multiple prototypes. This has also affected the development process, where a lot of attention has been focused on ensuring a good user experience.

According to the study regulation, the student must *"demonstrate knowledge on central techniques on the process of developing applications that solves realistic problems [and] demonstrate skills on analyzing, designing, programming, trying out, and testing applications that are part of a complex organizational subject."*. These subjects have been covered throughout this report: Part 1 and 2 documents the analysis and design of GIRAF and aSchedule, while presenting a discussion of development methodologies. Part 3 presents the finalized applications developed as well as details of the aSchedule implementation and integration with GIRAF. Part 4 presents the testing techniques used during and after development. Finally, part 5 reflects and concludes on the project before presenting a number of future improvements.

Through analysis of the problem domain of aSchedule, we gained an important understanding of which problems the developed system must solve. A model of the overall aSchedule system design was a the product of this analysis. Analysis of GIRAF was done in collaboration with the other development teams. The problem domain of GIRAF was analyzed to a lesser extend where only the system definition has been defined. This provided a common understanding of GIRAF that was used to specify the overall problems, which ultimately made us able to create a complete system architecture of GIRAF.

aSchedule has been developed with focus on separating component responsibilities and ensuring low coupling between system components. The development itself was marked by creating abstractions to ease development, e.g. by abstracting the database interface through an object mapper as explained in Section 10.4. The development process has been undertaken in an agile manner using XP. Though, this process has not been documented in the report, as the XP process has been adapted to a leaner approach, ensuring high efficiency, without entailing excessive overhead.

To test the final product of the development, GIRAF and aSchedule have been subjected to integration testing. Further an external evaluation of aSchedule was conducted by department manager Tove Thomasen

at the kindergarten for children with special needs, Birken. The conclusion of these tests is that all components of GIRAF integrates, even though a number of UI issues are yet to be corrected. aSchedule seemingly has potential for being useful in practice, although the implemented version leaves some improvements still to be developed.

The overall project, GIRAF, in its current state is released as an open-source project on Google Code under the GNU General Public License v. 2 (GPLv2) license in the hope that it can be continued by our fellow students and the community. Meeting with Birken has assured us there is an immediate need for an application like aSchedule now that a plethora of multi-touch capable devices exist and are socially acceptable to use as an aiding tool.

Future Work

19

This chapter covers a description of various ideas that would improve or advance aSchedule, and possibly how they could be implemented if we have had more time.

19.1 Video Playback

During a meeting with the personnel of Birken, a kindergarten for children with special needs, the discussion went to how video playback could improve aSchedule. The kindergarten describes that during everyday usage of their current physical visual schedule, they have a number of activities which are reoccurring. In spite of this, the pedagogues still needs to repeatedly explain how to perform these activities for the children. They feel that the step list in aSchedule is a good utility in reducing the child's dependency of a guardian. However, it would be good to have support for steps containing video playback, instead of just an image and audio file.

We agree that this is a valid improvement of the application and could be implemented in a future version of aSchedule. Android's media player, which we already use to play audio files, also supports multi-format video playback.

In Section 17.2.4, it was mentioned that Android still seems a little immature and rough in some places, especially when switching from device to device. During our experimentation with supporting video playback in aSchedule, we found that the media player's stability during video playback is very much dependent on the device manufacturer, even though all of the devices are running the same Android version. Due to this, we felt it was necessary to remove the video playback functionality in aSchedule until the Android media player reaches a more mature state.

19.2 External Administration of aSchedule

One of the key criteria for aSchedule to become successful is that it is easily managed by the guardians. As of now, the visual schedule can only be managed from the device.

A future improvement of the application would be to extend it with an external management utility, like a web interface. Having a web interface, accessible from a wide range of devices, allows us to take advantage of both larger screens, and a faster way for data input. This makes it possible to create an administration interface that gives the user a better overview, e.g. by showing a calendar-like interface. Further the guardians will more easily be able to add new content.

During the initial collaborative meetings, it was decided to create a PC interface to administer GIRAF. The proposal was to allow application developers to extend some base classes, defined by the development team responsible for the Administration Interface. Thereby, each application would have a plug-in for the PC interface, much like on the mobile device where each application's administration interface is

available through the global administration interface. Due to the limited time available, it was decided not to implement a PC interface and thus this is a future improvement, that both GIRAF and aSchedule could benefit from.

19.3 First Start Introduction

During the project, the application have been shown to possibly guardians, to take note of their opinions and reactions. We noticed that these users in general are not familiar with mobile touch devices.

A potential improvement to this would be to implement an introduction that would appear on first start. The introduction should provide an explanation of how to enter the Administration Interface, how to navigate on the device, and how to perform various tasks.

Regardless of whether aSchedule is a standalone application or a part of GIRAF, we see the need for an introduction for guardians. Both of these does, at this point, only contain little or no help for the user.

It is not difficult to imagine that both GIRAF and aSchedule could be offered to kindergartens and other institutions specializing in children with special needs. In such a case, one could offer a total solution where devices are pre-installed with the application(s) and bundled with a brief user manual for pedagogues.

19.4 Release aSchedule to Android Market

A potential future for aSchedule, and perhaps even GIRAF, would be to release it on the Android Market. Google created the Android Market as a service that makes it easy for users to find and download Android applications to their Android-based devices. Other marketplaces exist, but every Android-based device ships with a client for Google's Android Market out-of-the-box.

For aSchedule to have a real chance at becoming used in practice, it must therefore be released on the Android Market. At this point, aSchedule is in a close-to-releasable state. By this we mean that only few bugs and quirks need to be fixed before we feel that the application is ready to be released to the public.

We believe the most important thing to do in preparation for publication is testing. Preferably, this should be done on a large variety of physical devices under realistic conditions. This can be done, using the emulator to simulate a variety of screen resolutions, CPU speeds, etc. These tests should ensure that the UI elements scale properly and that the performance is acceptable on the devices we would wish to support.

Before publication, we must also ensure that we have removed any debugging code from the application, as well a debugging attribute from the manifest file, which tells the operating system to disable a number of debugging information. Last but not least, the application must be signed with a private cryptographic key, obtained from Google. This is a requirement from Google to guarantee end-users that they are in fact downloading what it appears to be.

On one of the initial collaborative meetings between GIRAF development teams, it was decided to release the entire system under GPLv2. This means that derived works, which aSchedule can be considered, can only be distributed under the same license. GPLv2 does not prohibit that any derived work can be commercialized but it requires that any derived work must as well stay open-source, even when the work is added or changed to. Additionally, the GPLv2 requires that recipients get a copy of the license along with the application [FSF91].

Should we decide to release just aSchedule to the public, no matter if it will be a free or paid application, we must keep it an open-source project and the GPLv2 license must be presented to the user for acceptance prior to installation.

Bibliography

[Bab11] Tom Babinszki. How people with autism use the computer. http://www.evengrounds.com/blog/how-people-with-autism-use-the-computer, May 2011.

[Bor11] Gaynor Borade. History of mobile phones. http://www.buzzle.com/articles/history-of-mobile-phones.html, May 2011.

[BT03] Barry Boehm and Richard Turner. Observations on balancing disclipline and agility. 2003. ISBN: 0-7695-2013-8.

[Dav11] Catherine Davies. Using visual schedules: A guide for parents. http://www.iidc.indiana.edu/?pageId=394, May 2011.

[DI09] Aalborg University De Ingeniør, Natur-og Sundhedsvidenskabelige Fakulteter. Studieordning for bacheloruddannelsen i software. http://www.sict.aau.dk/digitalAssets/3/3331_softwbach_sept2009.pdf, 2009.

[FSF91] Inc. Free Software Foundation. Gnu general public license. http://www.gnu.org/licenses/gpl-2.0.html, June 1991.

[Goo11a] Google. Android developers - activity. http://developer.android.com/reference/android/app/Activity.html, May 2011.

[Goo11b] Google. Coding style guidelines for contributors. http://source.android.com/source/code-style.html, May 2011.

[Goo11c] Google. Platform versions. http://developer.android.com/resources/dashboard/platform-versions.html, May 2011.

[GPI+06] Vanessa A. Green, Keenan A. Pituch, Jonathan Itchon, Aram Choi, Mark O'Reilly, and Jeff Sigafoos. Internet survey of treatments used by parents of children with autism. *Research in Developmental Disabilities*, 27(1), 2006.

[HHM+10] Gillian R. Hayes, Sen Hirano, Gabriela Marcu, Mohamad Monibi, David H. Nguyen, and Michael Yeganyan. Interactive visual supports for children with autism. *Personal and Ubiquitous Computing*, 14(7), 2010.

[HM11] Edward Hieatt and Rob Mee. Repository. http://martinfowler.com/eaaCatalog/repository.html, 2011.

[Kil11] Jesper Kildebogaard. Windows phone 7-salg en fiasko – kan nokia nå at redde microsoft? http://www.version2.dk/artikel/18994-windows-phone-7-salg-en-fiasko-kan-nokia-naa-at-redde-microsoft, May 2011.

[Lar03] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison Wesley, 2003. ISBN: 0-13-111155-8.

[MK99] Lynn E. McClannahan and Patricia J. Krantz. *Activity Schedules for Children with Autism*. Woodbine House, 1999. ISBN: 0-933149-93-X.

[MMMNS01] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. *Objekt Orienteret Analyse & Design*. Marko, 3rd edition, 2001. ISBN: 87-7751-153-0.

[MS92] Lars Mathiassen and Jan Stage. The principle of limited reduction in software design. *Information, Technology, and People*, 6(2), 1992.

[Pat05] Ron Patton. *Software Testing*. Sams Publishing, 2nd edition, July 2005. ISBN: 0-672-32798-8.

[PCSF08] C. Michael Pilato, Ben Collions-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, 2 edition, September 2008. ISBN: 978-0596510336.

[Rey06] Tammi Reynolds. Introduction to autism. http://www.mentalhelp.net/poc/view_doc.php?type=doc&id=8763, Feb 2006.

[SKKT06] Rober Stromer, Jonathan W. Kimball, Elisabeth M. Kinney, and Bridget A. Taylor. Activity schedules, computer technology, and teaching children with autism spectrum disorders. *Focus on Autism and other Developmental Disabilities*, 21(1), 2006.

[SQL11] SQLite. Distinctive features of sqlite. http://www.sqlite.org/different.html, May 2011.

[Tho11] Danielle M. Thorop. Computer play as a clinical intervention for children with pdd. http://www.superkids.com/aweb/pages/features/pdd/, May 2011.

[Wik11] Wikipedia. Wiki - wikipedia. http://en.wikipedia.org/wiki/Wiki, May 2011.

Part VI

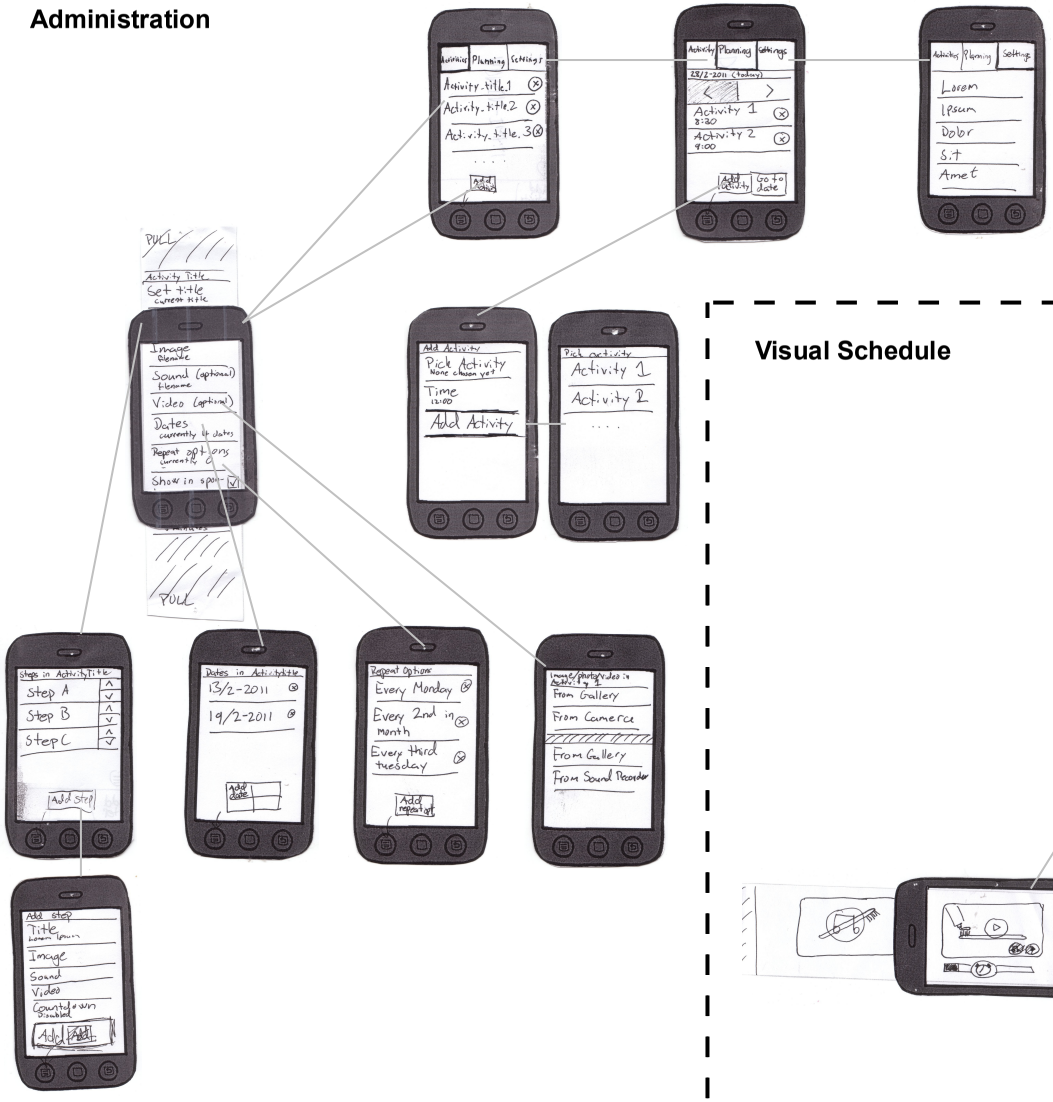
Appendix

The Final Prototype

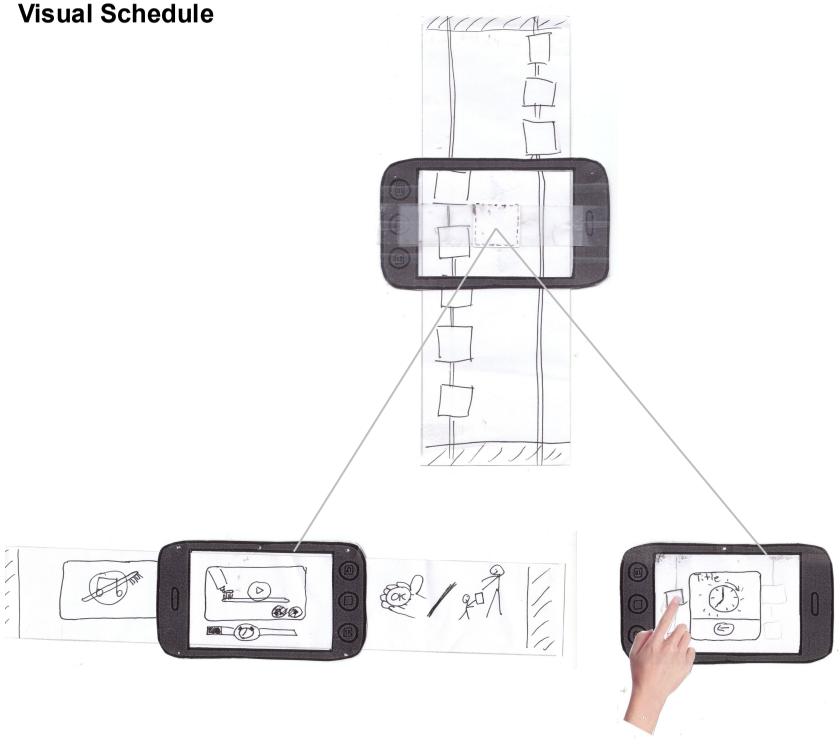
A

See next page

Administration



Visual Schedule



Integration Test Documents

B

See next page.

Multi-project Test

Purpose of the integration test

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

What is not covered is isolated tests of the individual parts of GIRAF. The requirements for executing the integration tests are 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 user interface 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 compile errors in revision 1122, hence the source code of this project is not included in the tests. However, a stable .apk of the PECS application is available, and will be used when possible.

Identifier: TD00

Program/Module/Object Under Test:

GirafPlace, application, administration panel, 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 Procedure:

Instruction	Expected Behaviour	Pass / Fail
Create an Android application with an invalid settings.xml (e.g. verify that it is invalid using the sw6.xmlvalidator tool)	The settings.xml is considered invalid by the sw6.xmlvalidator tool.	**aSchedule**: Pass **BMI4Kidz**: Pass
Upload the application to GIRAF Place.	The application is rejected by the GIRAF Place, and should report the same errors as the sw6.xmlvalidator tool.	**aSchedule**: Fail (application is accepted) **BMI4Kidz:** Fail (application is accepted)
Correct the settings.xml file, and upload the application to GIRAF Place again.	The application is accepted and stored on the GIRAF Place.	**aSchedule:** Pass **BMI4Kidz:** Pass
Verify that the settings.xml file is valid using the sw6.xmlvalidator tool	sw6.xmlvalidator reports file is valid	**aSchedule:** Pass **BMI4Kidz:** Pass
Update the application with a new change in its code. Remember to edit the version number. Upload the updated application to GIRAF Place.	The application will be upgraded, and stored on GIRAF Place.	**aSchedule:** Pass **BMI4Kidz:** Pass

Test Results and Observations:

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

Identifier: TD01

Program/Module/Object Under Test:

GIRAF.

Test Objective:

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

Test Conditions and Intercase Dependencies:

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

Test Procedure:

Instruction	Expected Behaviour	Pass / Fail
Download sw6.giraf from website (http://girafplace.lcdev.dk/start/)	The .apk file is downloaded.	**Desire:** pass **Hero:** pass **Wildfire:** pass **Tab:** pass
Install .apk file.	GIRAF is installed.	**Desire:** pass **Hero**: pass **Wildfire:** pass **Tab:** pass
Set GIRAF as the default launcher.	GIRAF Launcher is started. A first run dialog should appear.	**Desire:** pass **Hero:** pass **Wildfire**: pass **Tab:** pass

Setup user profile
- Name: John
- Address: Doe Street 1337
- Parent's Phone: +45 1234 5678
- Gender: boy
- Birthday: 01-02-2003
- Capabilities (mark following):
  - Can drag-and-drop
  - Can read
  - Can Work with Numbers
	The user profile setup is stored.	**Desire:** pass
		Wildfire: pass
		Hero: pass
		Tab: pass

Observations

All tests passed, however the following observations were made:

Rotating the user profile screen clears text fields:
- Open User Profile. Edit name. Close dialog. Edit name. Rotate screen. Name field is now empty (reported as issue 45).

Cursor not placed at the end:
- Open a text dialog, e.g. "Name". Enter the name. Close the dialog. Open the dialog again, and now the cursor is placed at the beginning, and cannot be moved to the end. This issue is observed on all HTC devices running HTC Sense UI (reported as issue 46).

Admin loads slowly
On HTC wildfire, the administration module loads slowly (reported as issue 47).

Identifier: TD02

Program/Module/Object Under Test:

Launcher, GIRAF Place, applications using “in-app” administration.

Test Objective:

- Ensuring applications can be installed on different Android devices using GIRAF Place.
- Ensuring that the launcher and applications are able to use and store settings in the administration module of GIRAF.
- Ensuring that the launcher and applications abides to the navigation flow for opening in-app administration mode that has been agreed upon.

Test Conditions and Intercase Dependencies:

Requires TD01 to be executed.

Test Procedure:

Instruction	Expected Behaviour	Pass / Fail
Start GIRAF Place.	Ensure that apps are listed and filtered according to user profile (only apps filtered according to user profile should be possible to download).	**Tab (aSchedule):** Pass **Desire (BMI4Kidz):** Pass **Hero(pecs, sched)**: Pass **Wildfire(digiPecs):** Pass
Download and install an application.	The download should start. After download has finished, verify that the Android Package Manager installer starts, and that the installation succeeds.	**Tab (aSchedule):** Pass **Desire (BMI4Kidz):** Pass **Hero(pecs, sched)**: Pass **Wildfire(digiPecs):** Pass
Navigate to the launcher home screen.	Check that only GIRAF applications are shown.	**Tab (aSchedule):** Pass **Desire (BMI4Kidz):** Pass **Hero(pecs, sched)**: Pass **Wildfire(digiPecs):** Pass
Launch an application.	Verify that the top panel is not shown.	**Desire (BMI4Kidz):** Pass **Tab (aSchedule):** Pass **Hero(pecs, sched)**: Pass **Wildfire(digiPecs):** Pass
Go into “in-app” admin mode (using the "secret" key combination).	Verify that the admin settings is the admin settings for the particular app being tested.	**Desire (BMI4Kidz):** Pass **Tab (aSchedule):** Pass **Hero (pecs, sched):** pass **Wildfire:(digiPecs):** Pass
Change a setting for the application	Check that the setting is updated.	**Desire (BMI4Kidz):** Pass **Tab (aSchedule):** Pass **Hero (pecs, sched)**: Pass **Wildfire(digiPecs):** Pass
Press back until home screen is shown.	Verify that the activities are shown in the following order: Admin panel, App, Home screen.	**Desire (BMI4Kidz):** Pass **Tab:** Pass **Hero (pecs, sced):** pass **Wildfire(digiPecs):** Fail
Start the app again.	App opens.	**Desire (BMI4Kidz):** Pass **Tab (aSchedule):** Pass **Hero (pecs, sched)**: pass **Wildfire(digiPecs):** Fail
Go into admin mode.	Verify that the changes made before are the ones shown.	**Desire (BMI4Kidz):** Pass **Tab (aSchedule):** Pass **Hero (pecs, sched)**: pass **Wildfire(digiPecs)**: Pass

Test Results and Observations:

Applications can be installed using GIRAF place. This makes them visible in the launcher and in the administration panel. If settings are stored using the “in-app” administration mode, they are stored persistently as expected.

- Press back until home screen is shown-failed with digiPecs: (running with old lib and bug is fixed) digiPecs needs to update newest lib-library.

- The refresh context menu in GIRAF Place cannot be read on HTC Wildfire. Icon is toooo big (reported as issue 48).

Identifier: TD03

Program/Module/Object Under Test:

GIRAF Place client (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. Ensure that an upgraded version of the package installed/used in TD02, has now been uploaded to GIRAF Place. The upgraded version should introduce some changes in its settings.xml file.

Test Procedure:

Instruction	Expected Behaviour	Pass / Fail
Open GIRAF Place client.	The app installed in TD02 should be able to be upgraded.	**Tab (BMI4Kidz):** Pass* **Desire (BMI4Kidz):** Pass* **Hero (pecs)**: Pass* **Wildfire(digiPecs, BMI4Kidz):** Pass*
Download the upgraded version.	The new version is downloaded.	**Tab (BMI4Kidz):** Pass** **Desire (BMI4Kidz):** Pass** **Hero (pecs)**: Pass** **Wildfire(digiPecs, BMI4Kidz):** Pass
Upgrade the downloaded application.	The Android Package Manager installer opens, and starts a “package replacement”.	**Tab (bmi):** Pass **Desire(bmi):** Pass **Hero (pecs):** Pass **Wildfire(digiPecs, bmi):** Pass
Go to launcher.	Updated application must be shown correctly.	**Tab (BMI4Kidz):** Fail*** **Desire(BMI4Kidz):** Fail*** **Hero (pecs):** Fail*** **Wildfire(digiPecs, BMI4Kidz):** Pass***
Launch the application again.	Verify that the settings have been upgraded correctly by checking the application, and the admin mode.	**Tab (BMI4Kidz):** Pass **Desire(BMI4Kidz):** Pass **Hero (pecs):** Pass **Wildfire(digiPecs, BMI4Kidz):**Pass

Test Results and Observations:

* If one forgets to update the GIRAF Place application listing, and the following assertions are true, then GIRAF Place is not able to install/upgrade the wanted application.
    - The application is not installed on the device.
    - The application has been upgraded on GIRAF Place compared to the version shown on the list on the device.

(reported as issue 49)

** If one forgets to update the GIRAF Place application listing, and the application is currently installed on the phone, GIRAF Place shows an Uninstall button (part of issue 49).

*** The application is now shown twice in the launcher. Both icons open the application though.
Install BMI4Kidz -> Verify that one icon is shown in the launcher -> Upload new version of BMI4Kidz to Market Place -> Upgrade the BMI4Kidz app -> go to the home screen -> now, two BMI4Kidz icons are visible on the home screen
(Also an issue when updating PECS)
(reported as issue 51)

Other observations:
Icons in the sw6.admin, and sw6.bmi package (high res) does not follow the Android design guideline: http://developer.android.com/guide/practices/ui_guidelines/icon_design.html
(reported as issue 50)

Identifier TD04

Program/Module/Object Under Test:

Back button behaviour, GIRAF Launcher, and applications.

Test Objective:

To test the back button behaviour of GIRAF Launcher and applications.

Test Conditions and Intercase Dependencies:

Requires TD01 to be executed.

Test Procedure:

Instruction	Expected Behaviour	Pass / Fail
Navigate to the home screen of GIRAF launcher and start Admin panel, by using the secret key combination.	Admin panel is started.	**Desire:** pass **Hero:** pass **Tab:** pass **Wildfire:** pass
Press the back button.	GIRAF Launcher home screen is shown.	**Desire:** pass **Hero:** pass **Tab:** pass **Wildfire:** pass
Press the back button again (while still in the GIRAF Launcher home screen).	Nothing will happen.	**Desire:** pass **Hero:** pass **Tab:** pass **Wildfire:** pass
Open application with more than one activity.	Application is started.	**Hero (aSchedule):** pass **Tab (aSchedule):** pass **Desire (aSchedule):** pass **Wildfire(aSchedule)::** pass
Open secondary activity from the main activity.	Secondary activity is shown.	**Hero (aSchedule):** pass **Tab (aSchedule):** pass **Desire (aSchedule):** pass **Wildfire(aSchedule):** pass
Open admin panel from the application.	Admin panel is started.	**Hero (aSchedule):** pass **Tab (aSchedule):** pass **Desire (aSchedule):** pass **Wildfire(aSchedule):** pass
Press the back button (ensure that we currently are in the root of the admin panel).	Secondary activity is shown.	**Hero (aSchedule):** pass **Desire (aSchedule):** pass **Wildfire(aSchedule):** pass
Press the back button.	Main activity is shown.	**Hero (aSchedule):** pass **Tab (aSchedule):** pass **Desire (aSchedule):** pass **Wildfire(aSchedule):** pass
Press the back button.	GIRAF Launcher home screen is shown.	**Hero (aSchedule):** pass **Tab (aSchedule):** pass **Desire (aSchedule):** pass **Wildfire(aSchedule):** pass
Press the back button.	Nothing happens.	**Hero (aSchedule):** pass **Tab (aSchedule):** pass **Desire (aSchedule):** pass **Wildfire(aSchedule):** pass

Test Results and Observations:

All step in this test passed and the back button behaves as expected.

Identifier: TD05

Program/Module/Object Under Test:

Behaviour of the home key.

Test Objective:

To test whether the home key handling works as intended.

Test Conditions and Intercase Dependencies:

Requires TD01 to be executed and passed. A valid application matching the user profile have been installed.

Test Procedure:

Instruction	Expected Behaviour	Pass / Fail
Navigate to home screen (GIRAF Launcher).	See the GIRAF home screen.	**Hero**: pass **Tab**: pass **Desire:** pass **Wildfire:** pass
Press key combo (vol up - vol down) * 3	Admin panel opens.	**Hero**: pass **Tab**: pass **Desire:** pass **Wildfire:** pass
Press home button.	GIRAF home screen is shown.	**Hero**: pass **Tab:** pass **Desire:** pass **Wildfire:** pass
Press home button.	Nothing happens/GIRAF home screen is still shown.	**Hero**: pass **Tab:** pass **Desire:** pass **Wildfire:** pass
Click an app to open it	The clicked app opens.	**Hero (aSchedule)**: pass **Tab (aSchedule):** pass **Desire (bmi):** pass **Wildfire:** pass
Press key combo (vol up - vol down) * 3	Admin panel opens.	**Hero**: pass **Tab:** pass **Wildfire:** pass
Pres home button.	GIRAF home screen is shown.	**Hero**: pass **Tab:** pass **Desire:** pass **Wildfire:** pass
Click an app to open it.	The clicked app opens.	**Hero (aSchedule)**: fail* **Tab (BMI4Kidz):** fail* **Desire (BMI4Kidz):** fail* **Wildfire:** fail*
Press home.	GIRAF home screen is shown	**Hero**: pass **Tab:** pass **Desire:** pass **Wildfire:** pass
Press back.	Nothing happens.	**Hero**: pass **Tab:** pass **Desire:** pass **Wildfire:** pass
Press home.	Nothing happens.	**Hero**: pass **Tab:** pass **Desire:** pass **Wildfire:** pass

Test Results and Observations:

*Open aSchedule or digiPecs (or BMI4Kidz). Enter admin panel. Press home. Open aSchedule. This opens the admin panel. This is unintended behaviour, as children may unintendedly enter the admin interface (reported as issue 43).

Identifier: TD06

Program/Module/Object Under Test:

Testing uninstall functionality in GIRAF Place.

Test Objective:

Test if applications can be deleted from GIRAF Place.

Test Conditions and Intercase Dependencies:

Requires TD01 to be executed.

Test Procedure:

Instruction	Expected Behaviour	Pass / Fail
Open GIRAF Place, and un-install an application.	Application is un-installed without warnings or errors.	**Hero:** pass **Wildfire:** pass **Tab:** pass **Desire(BMI4Kidz):** pass
Open launcher.	Launcher does no longer show the un-installed application.	**Hero:** pass **Wildfire:** pass **Tab:** pass **Desire**: pass
Enter admin mode, open device settings, and open package manager.	Un-installed application is not shown in the applications list.	**Hero**: pass **Wildfire:** pass **Tab:** pass **Desire:** pass
Download un-installed application from GIRAF Place.	Application is downloaded without warnings or errors.	**Hero**: pass **Tab:** pass **Wildfire:** pass **Desire(BMI4Kidz):** pass
Install application.	Application is installed without warnings or errors.	**Hero**: pass **Tab:** pass **Wildfire:** pass **Desire(BMI4Kidz):** pass
Go to the home screen of the launcher.	Application is shown.	**Hero**: pass **Tab:** pass **Wildfire:** pass **Desire(BMI4Kidz):** pass

Test Results and Observations:

The un-install process works as intended.

Identifier: TD07

Program/Module/Object Under Test:

GIRAF Place, and user profile

Test Objective:

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

Test Conditions and Intercase Dependencies:

Requires TD01 to be executed.

Test Procedure:

Instruction	Expected Behaviour	Pass / Fail
Open home screen.	Home screen is shown.	**Hero**: pass **Wildfire:** pass **Desire:** pass
Go into admin mode.	Admin mode is shown.	**Hero**: pass **Wildfire:** pass **Desire:** pass
Open user profile settings (through administrative settings).	User profile settings are shown.	**Hero**: pass **Wildfire:** pass **Desire:** pass
Edit capabilities in user profile. Set *canRead* to false.	Ensure all settings are saved when finished.	**Hero**: pass **Wildfire:** pass **Desire:** pass
Press home.	Home screen is shown. Ensure no text is shown under applications.	**Hero**: pass **Wildfire:** pass **Desire:** pass
Go into admin mode.	Admin mode is shown.	**Hero**: pass **Wildfire:** pass **Desire:** pass
Start GIRAF Place.	Ensure that only applications, that meets the user profile capabilities, or already are installed, are shown.	**Hero**: pass **Desire:** pass **Wildfire:** pass

Test Results and Observations:

Applications were filtered according to specifications.

Summary in Danish

C

Dette bachelorprojekt er udarbejdet af softwaregruppe f11s601e ved Aalborg Universitet på forårssemesteret 2011 . Projektets overordnede tema er applikationsudvikling. I projektet er fokus lagt på udvikling af mobile applikationer til børn med særlige behov. Projektet er udarbejdet i samarbejde med tre andre softwaregrupper, med det formål at skabe et komplet system der understøtter barnets behov og kompetencer. Projektet er udviklet til Android-platformen, med fokus på udvikling til tablets og smartphones. Denne gruppes ansvarsområde har været at udvikle en "visual schedule" applikation, som kan anvendes som hjælp til at strukturere det autistiske barns hverdag efter principper, som er benyttet i klassiske visual schedules.

Rapporten beskriver udviklingen af visual schedule applikationen, aSchedule, udviklet af rapportens forfattere, samt integreringen af aSchedule med det overordnede system, GIRAF.

Rapporten beskriver indledningsvis autisme og visual schedules, samt de muligheder IT giver som hjælpemiddel til autister. De begreber der defineres her, danner senere det nødvendige grundlag for resten af rapporten. Eftersom projektets fokus i høj grad er på udvikling af større softwaresystemer, bliver forskellige udviklingsmetoder diskuteret, for at fastslå, hvilken udviklingsmetodik der bør anvendes gennem udviklingsprocessen. Det konkluderes at objektorienteret analyse og design benyttes i første fase af projektet, for i samarbejde med de resterende grupper, at fastslå fælles begreber og målsætninger. Efter denne proces vil de forskellige delprojekter blive udviklet individuelt. Der argumenteres herpå for at eXtreme Programming vil blive benyttet til at udvikle aSchedule. Foruden at diskutere udviklingsmetodikker, præsenteres de udviklingsværktøjer og -aftaler, der benyttes på tværs af udviklingsgrupperne, samt hvorfor de er nødvendige.

Målet med den efterfølgende del af rapporten er at definere en forståelse for hvordan princippet om visual schedules skal benyttes i aSchedule. Dette gøres for at identificere områder, der kan administreres, kontrolleres og overvåges i det udviklede system. Baseret på denne analyse dannes det nødvendige grundlag for at udvikle prototyper, der benyttes til at definere det faktiske design af aSchedule.

Den tredje del af rapporten beskriver selve implementationen af aSchedule, baseret på designet og analyserne som er foretaget. Android-platformen hvorpå aSchedule og GIRAF bliver udviklet bliver beskrevet, hvorefter det udviklede produkt bliver præsenteret. Herefter gennemgås implementationen detaljeret, for at beskrive flere af de interessante problemstillinger og løsninger, som er udarbejdet. Der fokuseres i særlig grad på database-modellen, hvortil der benyttes en objekt-relationel mapper for at strukturere dataadgang og -manipulation samt forskellige Android-specifikke koncepter, der er implementeret i aSchedule. Slutteligt beskrives hvordan aSchedule er integreret i GIRAF, samt hvilke udfordringer der ligger heri.

Testen af det udviklede produkt dokumenteres i den fjerde del af rapporten. Denne indeholder en beskrivelse af de implementerede unit tests af database-modellen, et udviklet program til at teste hvorvidt Android-applikationers livscyklus håndteres korrekt samt en afsluttende dynamisk blackbox-test af det komplette GIRAF-system. Der afsluttes med en diskussion og vurdering af aSchedule udarbejdet i samarbejde med Tove Thomasen, afdelingsleder ved specialbørnehaven Birken.

Afslutningsvis i femte del af rapporten, reflekteres der over valget af udviklingsmetodik, både for GIRAF

og aSchedule. Der reflekteres her også over flere af de løsninger, der er valgt på problemstillinger opstået i løbet af projektet samt over de vanskeligheder, som en mobil platform introducerer i et udviklingsprojekt. Denne refleksion efterfølges af en konklusion på projektet som helhed. Et afsnit om fremtidige forbedringer af GIRAF og aSchedule præsenteres for at følge op på nogle af de problemer, der ikke er løst i projektet, men som vil blive løst, inden programmet udgives på Android Market.

Acronyms

D

API Application Programming Interface

CRUD Create, Retrieve, Update, Delete

DAO Data-Access Object

DBMS Database Management System

DDMS Dalvik Debug Monitor Server

GPLv2 GNU General Public License v. 2

IRC Internet Relay Chat

LIFO Last In, First Out

MVC Model-view-controller

OOA Object-Oriented Analysis

OOAD Object-Oriented Analysis and Design

OOD Object-Oriented Design

ORM Object-Relational Mapping

OS Operating System

PECS Picture Exchange Communication System

UI User Interface

UML Unified Modeling Language

UP Unified Process

SDK Software Development Kit

SQL Structured Query Language

SVN Subversion

XML Extensible Markup Language

XP Extreme Programming