POSEiDON

Pers**O**nalized **S**mart **E**nvironments to increase **I**nclusion of people with **DO**wn's syndrome

Deliverable D4.2

Adaptive Tablet Interface

Call:	FP7-ICT-2013-10
Objective:	ICT-2013.5.3 ICT for smart and personalised inclusion
Contractual delivery date:	31.10.2016 (M36)
Actual delivery date:	31.10.2016
Version:	Final
Editor:	Dean Kramer, MU Lars Thomas Boye, Tellu Silvia Rus, Fraunhofer
Contributors:	Juan Carlos Augusto, MU Riitta Hellman, Karde
Reviewers:	Andreas Braun, Fraunhofer
Dissemination level:	Public
Number of pages:	27

Content

1 Executive summary

As part of the POSEIDON project, we are aiming to develop context-aware applications to assist our primary users in their daily lives. One aspect of this is the context acquisition and reasoning, but the other aspect is how the application should react to specific contexts, and adapt itself. The purpose of this deliverable is to document the adaptivity and personalisation aspects of the POSEIDON application interfaces.

In the report introduction, we give a brief introduction to the document, and explains that since the project proposal we have adjusted the screen sizes we are considering for the mobile device. We should consider all types of screen sizes that the users may wish to use. We then define the difference between adaptive and adaptable UIs, since these terms can be mixed, and used incorrectly.

Adaptivity and personalisation within HCI are discussed in Chapter 3. We consider the requirements we have received previously with advice from the DSA organisations that adaptive UIs can cause disorientation and confusion. This we feel means that for now, we should stick to mostly notifications and prompts. We next introduce different approaches to adaptive UIs in literature and discuss some usability studies that have been conducted with adaptive UIs.

On the topic of personalisation we introduce different usability studies. In general, it would appear that personalisation is likely to be more beneficial to our primary user group.

Following this, in Chapter 4, we describe some of the contexts and personalisation options we have implemented for Pilot 1.

In addition to the previous version of this document we address in Chapter 5 feedback from pilot 1 and how adaptation has been focused on a few use-cases from the main POSEIDON App and how personalisation is implemented at all stages of the POSEIDON system.

In Chapter 6 we introduce the prototype being assessed in Pilot 2. It is designed and implemented using Tellu's modular mobile framework. Different screenshots of the app are explained, with items such as personalisation and adaptivity described more in detail.

We conclude this document by stating a few lessons learned.

2 Introduction

The goal of this deliverable as stated in the Description of Work was to “provide an interface which is intelligent and adapts to the user and context as well as it allows the user to set it up to prioritize the special needs of a specific individual.” The title uses the word “tablet” as at the time of writing the proposal it was perceived to be the mobile platform of choice. Along the gathering of requirements it became clearer that actually we need to look at a wider range of devices of different sizes, and that the primary target is a large phone or small tablet (screen size 5-7 inches). Hence the right way to interpret this deliverable is that it is looking at the design and development of adaptive interfaces for whatever mobile support the primary user prefers whilst out of home.

Within this deliverable we are discussing UIs that within the project are not static. We therefore should highlight that non static UIs are historically considered either adaptable, or adaptive (Stuerzlinger *et al.*, 2006). Adapt*able* UIs can be described as UIs that can be altered at runtime by the user. Adapt*ive* UIs however, are UIs that can be altered at runtime by the system automatically. In terms of this deliverable, we define all adaptable elements of our work as a form of personalization.

The remainder of this document is broken down into the following sections: Chapter 3 discusses adaptivity and personalisation in HCI, including approaches and usability studies from literature. Persoanlisation (former Chapter 4) is now included in Chapter 3. Chapter 4 now presents ideas for personalisation and adaptation and also states which of those have been implemented for pilot 1.

Chapter 5 formerly covering details of the updated system architecture have been removed because content will be moved to D5.2 Protoypic systems The Current Chapter 5 presents how we addressed feedback from pilot 1 regarding the main user app and how we implement personalisation throughout the POSEIDON components developed for pilot 2. Chapter 6 describes the pilot 2 implementation of the POSEIDON main app regarding adaptivity and personalisation. Lessons learned are presented in the conclusions, Chapter 7.

3 Adaptivity and personalisation in HCI

In this section, we shall introduce and discuss application adaptivity and personalisation in terms of HCI. We firstly reflect on our previous requirements before examining current approaches and usability studies.

Our previous gathering of requirements indicated different elements of flexibility in the system and especially personalization are valued by the potential users:

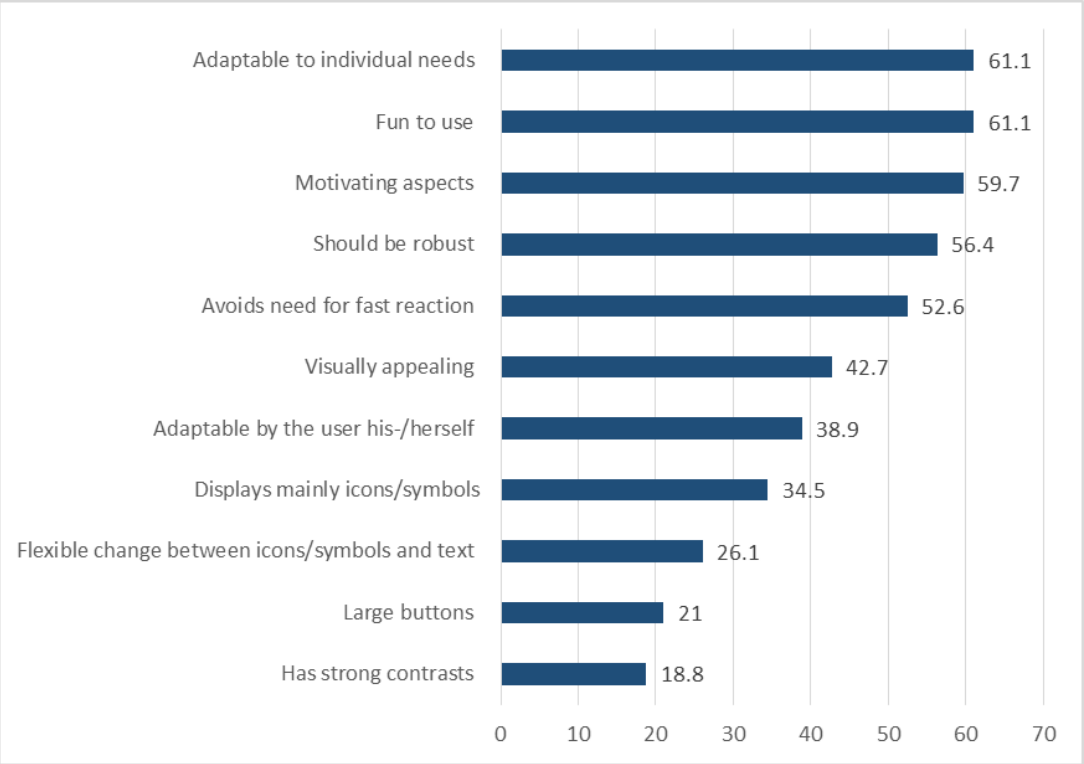


Figure 1: Design aspects being important for people with DS ("very important" in percent, N varies between 377 and 385)

At the same time we have been reminded on several occasions by our DSA organizations taking part in the project that change can disorientate some of our primary users hence the issue of what, if any, part of the system can be dynamic is still under investigation. Similar issues were found in dynamic UI usability studies carried out on people without cognitive disabilities (Holzinger *et al.*, 2012).

This justifies why we decided not to implement this features for the month 20 prototype and instead we decided to use that event to explore some possibilities and measure the level of receptivity to those proposed adaptivity features.

3.1 Adaptivity approaches

Many different approaches have been proposed to develop adaptive UIs. These approaches include model-based UIs, stepwise composition, domain specific languages, model-driven software engineering and rule based approaches.

A middleware based approach for mobile applications was proposed by David *et al.*(2011). This approach makes use of the Context-Oriented Programming language ContextJ (Appeltauer *et al.*, 2011) for implementing UI changes in the application. This approach was found to reduce development and testing time, while also being implementable with fewer lines of sourcecode.

Hanumansetty (2004) proposed an approach to context-aware UIs using model-based UIs. This approach includes a framework which handles context processing and interface adaptation of web applications off device. All contexts are collected by the device, before being sent to the context server for aggregation and interpretation. The business components and interface server then receive context events from the context server. Using different task model adaptation rules, the task model of the UI is regenerated for the UI based on the context. Following the regeneration of the task model, the abstract UI is then generated using the dialogue model, and the concrete UI is generated using the presentation model. This adaptation approach was applied to XHTML documents.

A toolkit for handling UI migration and adaptation was proposed by Grolaux (2007). This toolkit was developed for the Mozart Programming System[1] using the Oz language. It supports dynamic adaptation, and different representations of each UI widget, which can then be switched based on the context at runtime. These representations are supported by individual UI renderers, which each represent a particular runtime variant of that UI widget. This toolkit can support adaptation to the entire screen, individual UI widgets, and different pixel areas.

Model-driven software engineering (MDSE) approaches to adaptive UIs include the works of Criado *et al.* (2010) and Rodriguez-Gracia *et al.* (2012). These approaches use model-to-model (M2M) transformations to adapt architectural models at runtime to suit a given context. Criado *et. al* propose an approach based on two specific processes. The first process handles interface architectural M2M transformations. These interface architectural models are made up of abstract UI components, which create abstract UI models during M2M transformation. The second process involves creating a concrete UI by getting the appropriate widgets from a widget repository.

Adapting the UI dynamically using stepwise composition was proposed by Savidis and Stephanidis (2010). In this approach, static UIs are adapted using a refactoring process, using OOP languages including C++. This approach is broken down into three distinct steps. In the first step, user requirements are analysed, and roles and requirements are identified. Then, the interface profiles are modelled, expressing variations of the user-interface behaviour. The step then ends with the identification of adaptation logic where context event and adaptation rules are specified. In the second step, adaptation alternatives are encapsulated, by use of a general superclass, and putting alternative components into subclasses. During the adaptation, components are terminated, with their substitutes being activated, with state from the original component being passed to the substitute class constructor.

While most adaptation approaches have been proposed to change what UI elements may or may not be on the screen, other work has been proposed to help deal with runtime UI scaling for variable device screen sizes (Behan and Krejcar, 2012). The authors suggest that for mobile applications, graphical elements should use Scalable Vector Graphics (SVG). By using SVG for each UI element on the screen, the screen can scale more easily to meet the needs of different screen sizes. The approach overrides the standard UI drawing methods and uses vector graphics instead of raster graphics.

Other approaches to handle adaptive UIs includes the use of Dynamic Software Project Lines (Kramer *et al.*, 2014). This work considers how UI adaptation can be handled statically and dynamically for document-oriented UIs. It proposes to extend the use of Feature-Oriented Programming (Batory, 2004; Apel *et al.*, 2013) to implement UI adaptation in a single unified way. The approach then creates all possible valid configurations of the UIs, which can be chosen by the system at runtime depending on the context and other system features.

[1] https://www.mozart-oz.org

3.2 Adaptivity and Usability

Different studies have been carried out to check the usability effects of adaptive UIs. One such study includes Paymans *et al.* (2004). In this study, the authors claim that users can feel a lack of control when using adaptive UIs because the interface seems unpredictable. The authors carried out an experiment to see how building greater mental models of the system can affect the usability. Using students from Utrecht University, it was found that having an improved mental model was not always needed. Depending on the domain, it was suggested that an understanding of the system's design was not always necessary for ease of use.

Other later studies include usability evaluations of adaptive UIs used for in-vehicle systems (Lavie and Meyer, 2010). This study examined four different factors including different tasks, routine and non-routine situations, age groups, and different levels of adaptivity. Two different experiments were carried out. One for younger users with an average age of 25.7, and one for older users with an average age of 58.6. Different tasks were executed by the user including reading and sending SMS messages, reading email and new updates, and changing the CD. This study found that intermediate levels of adaptivity can be more beneficial to the user, and that adaptivity does not benefit every situation. Furthermore, it was found that for routine user tasks, fully adaptive systems were more beneficial.

During the last decades Virtual Reality (VR) has become a popular technology and its added value for learning has been recognized. However, its flexibility and richness in representation can sometimes overwhelm the users, who become lost in the Virtual Environments (VE). In order to deliver relevant information, it is important that the system dynamically adapts to the personal preferences, skills, learning goals or personal context. Augmenting a VE meant for learning with adaptive information has many advantages (Chittaro and Ranon, 2007). Thus, the VE can become more effective in guiding users who exhibit noticeable individual differences and in preventing the potential conflict between usability and learning.

Possible adaptations in a VE dedicated to learning are presented in (Troyer *et al.*, 2010). The authors classify the adaptations depending on the number of VE components they apply on: *adaptation types* (they apply on single component: objects, behaviours, avatars) and *adaptation strategies* (they involve more than one component). For instance, when we consider the interaction with an object, the usability of the VE can be enhanced if the visual appearance of the object can be changed (e.g., to visually indicate that the object has not been studied, it can be highlighted, made smaller or even be hidden). Examples of adaptation strategies can be related to the way the user navigates in the VE (e.g., with restricted interaction, restricted behaviour, with or without suggestions) or to the adaptation of a group of objects (e.g., filter objects that allows selecting the objects that should be available in the VE). However, these ingredients of an adaptive VE need to be managed by an adaptation process.

In designing an adaptation process, Troyer (2010) identified three different approaches: an author-driven approach (the author of course is given full control over the adaptation process), a teacher-driven approach (similar to author-driven approach, but adaptation is done at run time, while the learner is in action) and a model-driven approach (adaptation driven by intelligent algorithms). Because of the richness of adaptation possibilities in the VE, designing an efficient adaptive system for learning is a challenge that needs further investigation.

3.3 Personalisation and Usability

Some of the earliest customization studies included those carried out on word processors (Page *et al.*, 1996). This study considered five different categories of customisation: *functionality, access to interface tools, access to functionality, visual appearance,* and *general preferences.* A total of 101 participants from the United States joined the study, with 92% of participants carrying out some form

of customisation. This study suggested that most users want to tailor their software to fit their needs, and that customisations should be easy and fit the work.

Other later studies including McGrenere *et al.* (2002) and Findlater and McGrenere (2004) have shown that the usability performance of a GUI can be improved by allowing it to be customised by the user. Moreover, by allowing for customisation, a higher sense of control and identity can be found with the application (Marathe and Sundar, 2011).

More recent advanced UI personalisation and customisation user studies exist (Zeidler *et al.*, 2013). This study looks at advanced UI layout and functional customisations. The authors introduce systems to handle these customisations and carry out a study with 18 technical users. This study found that technical users are able to use the different customisation systems, and would use them in practice. The results of this study suggest customisation techniques can be beneficial to the user.

4 Pilot 1 - Adaptivity and Personalisation

In this section, we explore the different adaptable and personalisation elements of POSEIDON. We are discussing some possibilities for personalisation and adaptivity within POSEIDON .

4.1 Personalisation within POSEIDON

Personalisation may be a safer option for the project to explore. We still have to determine which aspects of personalisation will be more appreciated by our users and we aim to gather more feedback from users after finishing the first pilot. While personalisation has been shown to be beneficial to users without DS, we are yet to understand the effects of personalisation on people with DS.

For pilot 1 the possibilities for personalisation in the mobile application include:

- Allow font size change: this seems to be the safest option as there are often vision problems within the primary users which can benefit with bigger fonts available, especially with the current trend of using smartphones with small screens as the most commonly used mobile platform.
- Allow colour change: there is less evidence for its need however we want to find out whether specific colour codes can be effectively used to enhance visibility of the elements in the screen.

For Pilot 1 we have implemented both.

Home Navigation Training:

- Allow the selection of a certain type of information: people with DS have different cognitive capacities and a preference for specific types of information (visual, audio, 3D, 2D)
- Allow the selection of interaction devices (keyboard/mouse, interactive table, kinect, etc.)

For Pilot 1 we have implemented:

- Written text and visual (2D and 3D) information. However, due to limitations by Google street view, the 3D view was not available in every country.
- Used interaction device was keyboard and mouse. The interactive table will be connected for pilot 2.

Interactive Table:

The interactive table shows potential of personalization in the large set of interaction patterns. The different options are illustrated as the following:

- Custom gesture set - the user can select from a set of available gestures for interaction
- Custom touch set - the user can select which touch gestures are supported
- Gesture scaling - the gestures can be adapted to the individual interaction speed
- Multi-user - the system can adapt based on the number of users currently interacting
- Gesture training - the set of gestures can be expanded or reduced by user training

However, for pilot 1 only taps on the table are available, due to reason explained in deliverable D4.3 – Interactive Table. Hence, the interaction pattern is highly restricted. However, the taping sensitivity can be set by the application, allowing the user to filter out light touches. Developing the table prototype further will allow for a greater set of interaction patterns, which can then be built into the according applications.

For pilot 1 we implemented the gesture scaling, meaning that altering the tap sensitivity is possible. All other personalisation options occur on application level, at the Moneyhandling Training.
However, for pilot 2 the interactive table could extend following personalisation options:

- Allow to select from different gesture sets based on best practice/piloting
- Allow to select different interaction patterns with or without touch or using different elevation levels

4.2 Adaptivity within POSEIDON For Pilot 1

The project offers the opportunity to interact with primary and secondary users and to assess the suitability of some ideas which we are aiming to assess in Pilot 1. As discussed earlier, due feedback received in Mainz on app adaptivity, and existing usability studies into adaptive UIs, we plan to predominately use specific user prompts, notifications, and reminders based on the context. Below, we illustrate some of the adaptive elements within POSEIDON for pilot 1, and those still to be assessed.

Mobile application:

For pilot 1, we have implemented following notifications and prompts:

- Start the showing the user navigation instructions when the user has started a navigation route only after the user is found to have gone outside.
- Prompt the user if s/he needs assistance if they deviate from a route either in a severe amount, or if they continue to make many smaller navigation errors.
- Notify the user when the device battery is low.

Outside of the pilot, we considered the following possibilities:

- Icons on screen change size according to contexts where they are less or more relevant: this will allow some icons to become more visible when it matter.
- Icons on screen change size according to how frequently they are used historically irrespective of the current context: this will make those applications considered more useful by the primary user prominent, and therefore easier to locate.
- Icons on screen (both in primary and secondary user interfaces) change colour, for example red for contexts where the system perceives a worrying scenario developing.

Context-Awareness

For the UI to be adapted automatically by the system, the system needs to gather and interpret different context and usage information. These different contexts are being developed as a reusable context library to be used across all prototypes. In Pilot 1, we had the following ContextObservers:

- **BatteryContext:** This context keeps track of the current remaining battery capacity.
- **ExternalStoreSpaceContext:** This context tracks the remaining public storage space on each device.
- **LightContext:** This monitors the ambient light conditions surrounding the device.
- **TelephonyContext:** This monitors the different connection properties of the device including whether it is on a 3G connection, and roaming status.
- **WifiContext:** We use this to monitor the connection status of the device to WIFI point.
- **GPSIndoorOutdoorContext:** This context calculates if the device is indoors or outdoors using GPS receivers.
- **CompassContext:** This context monitors the current direction of the device.
- **UserMoodContext:** This context is used to keep track of the user's mood.

- **WeatherContext:** This context monitors the current weather conditions of specifically important locations, included those in an upcoming event.
- **DeviationContext:** This context monitors when the user is deviating from the navigation route, and counts how many times within a timeframe the user deviates.

These contexts are available in the centralised context reasoner we have developed. This context reasoner will be able to supply context information to all POSEIDON compatible applications.

Home Navigation System

For pilot 1 we have not implemented any adaptivity.

Outside of the pilot, we considered the following possibilities:

- Restricted navigation – allows restricting the navigation to specific objects in the scene or to a particular navigation behaviour (jumping, walking). The criteria can be represented by the learner's knowledge.
- Tour Guide – allows taking the learner through a tour in the VE that can help the user gain general information about the environment he will interact with.
- Providing suggestions – allows the learner to navigate in the VE guided or not by using suggestions that can be offered by using marking (e.g., highlight)
- Behaviour speed – allows specifying the speed of a behaviour (being able to slow down will allow the learner to observe better what is happening)
- Interaction limitation – allows limiting the interaction time with a specific object by means of a condition
- Reinforcement learning – allows the environment to take actions to maximise some notions of cumulative reward

Interactive Table:

For pilot 1 we have not implemented any adaptivity for the Interactive Table.

5 Pilot 2 – Adaptivity and Personalisation

The Poseidon component which was realized for pilot 1 using a mobile device is the Poseidon App. It offers two main functionalities: support of daily planning (enhanced calendar) and outdoor navigation support.

5.1 Feedback from pilot 1 regarding POSEIDON App UI

Deliverable D6.3 describes the feedback regarding these two components in Section 3.1 Calendar and 3.3 Navigation. In the following we present feedback regarding the UI for the calendar functionality and for the navigation functionality. For each point we mention if and how it has been addressed for pilot 2:

Feedback regarding the UI for the calendar functionality:

- the calendar has a good design which is true for font, font size, contrasts and readability
- Also the time bar seems to be a good idea even if some of the PUs are not able to understand it completely.
  - → For pilot 2 development we considered replacing the bar with a watch, which was suggested by the SUs and the representatives of the Down Syndromes Associations. In the end the filling bar has been replaced by a circular bar.
- When entering the start and end time of an event, the colon could already be added by default. Two separate fields for entering hours and minutes would also be helpful.
  - → It has been addressed. Two different time fields can be scrolled to set the time.
- Add preferred views (day vs. week. vs. month).
  - → Has been considered. The implementation of a monthly view was worrying, since the space on the mobile phones screen is very limited. However, a good solution was found. Results with screenshots can be seen in D4.1 Interface Strategy Chapter 10.1.
- Support more icons instead or in addition to text
  - → We added more icons for the pilot 2 implementation, where possible combined with written text.

Feedback regarding the UI for the outdoor navigation functionality:

- Almost all PUs experience problems by using the app because of the multitasking which the app demands. Paying attention to the traffic, taking a look at the map and following the instructions are tasks which the PUs have to do simultaneously. People with Down's syndrome are more driven by visual influences and therefore constantly focus on the screen.
  - → This point has been addressed by using more audio messages in addition to images. The thorough change of the navigations function, being able to set own decision points gives the carer the chance to configure instructions at the difficult points e.g. to look at the street.
- One PU had great difficulties to switch between navigation map and instruction and then the map again.
  - → This has been addressed by using audio messages for navigation.

Feedback regarding Context-awareness functionality:

- Generally, PU and SU's found weather, and navigational assistance contexts helpful.

- There was comments that they would like to be able to personalise these contexts to suit their needs more
  - → This was accomplished through the development of R-Settings that allow users to set particular context parameters to meet their needs.
- Users would also like to see historical data about the primary user, so they can tell if they are getting to school/work on time.
  - → We developed the learning module, which through the Carer's web enables users to get access to this data and visualise it.

5.2 Adaptivity for Pilot 2

From this list of observations, we conclude that primary users suggest most changes regarding personalisation options, usability and functionality. In Section 6.3 the adaptability use cases are presented. For all other POSEIDON components adaptivity has not been further considered for pilot 2, partly due to the lack of useful use-cases and due to the need to focus more on the personalisation aspects.

5.3 Personalisation for Pilot 2

Starting from the user profile stored in the POSEIDON account, applications adapt to the settings stored in the user profile. This triggers the personalisation of the apps content and user interfaces. In Chapter 6 we focus on describing the adaptivity and personalisation of the main POSEIDON mobile app while in the following sections we address personalisation in the different auxiliary apps and the other POSEIDON components.

5.3.1 Moneyhandling Training App

The Moneyhandling Training Apps is started from the main POSEIDON app. Once it starts, the language is loaded from the user profile. According to the currency stored in the user profile, different sets of coins are used for training. The content of the app is also personalised according to user settings. The app downloads the configures shopping list and price tags and uses this information to create the game.

5.3.2 Shopping App

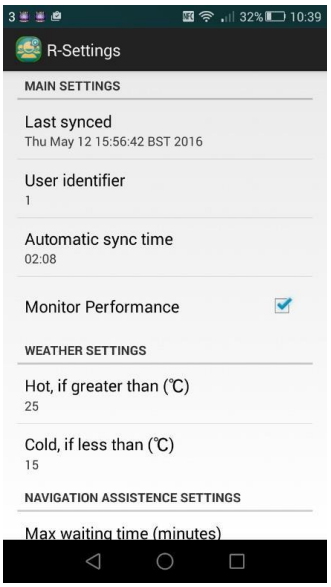
The shopping app starts only from the main POSEIDON app. Here user profile information is transferred to the app. Accessing the currency of the shopping list and the shopping products and price tags the content of the shopping app is displayed. The language selection occurs by detecting the languages settings of the Andorid version on the phone. The user inputs the content of his wallet, exactly the coins and bills he has. This was designed like this to assure maximal personalisability.

5.3.3 Route Creator App

The Route Creator App is an app which helps design the routes for the navigation functionality outdoors. The app itself does not have many personalisable features, however it also adapts to the language settings of the phone. It contributes to create a highly personalisable route, by giving the user the choice of which medium to use to convey the instructions for the decision points the user wants to create.

5.3.4 Context Settings

Based on the feedback from Pilot 1, we added the ability for the user to personalise different contexts in the reasoner to suit their personal preferences. This includes the ability to alter temperature preferences for hot and cold, and navigational assistance. Navigational assistance settings include the number of small deviations and how long they can be standstill before the user is offered to call the carer. Depending on the type of journey, the secondary user can customise these settings so the system is not too sensitive or unresponsive.



Other settings include privacy settings on whether context data can be used by the learning module by the secondary user in the POSEIDON Carer's web.

After this quick overview of the personalisation for pilot 2 we look in more detail into the personalisation and adaptivity for the main POSEIDON app.

6 Main primary user mobile prototype

This chapter presents the user interface of the main primary user mobile application. This application has been in development throughout the project, starting with a limited tracking and navigation test in the first iteration, expanding to a fully connected pilot application also including a calendar interface, with further improvements and the addition of video instruction list in the third iteration. Additional mobile applications, for money training and shopping assistance, joined the system in the third iteration. These are available from the main app menu, and share some of the user interface design principles, but are separate applications not discussed in this chapter. A Route Creator mobile application was also created in the third iteration, but this is a secondary user application for creating content, and so does not have a focus on personalisation and adaptivity.

The final prototype of the main mobile application has the following main features:

- A user profile with preferences stored in the POSEIDON infrastructure. The application establishes a connection to the infrastructure with user name and password.
- Calendar showing events, reminders and instructions, as well as allowing entering and editing of events. This is synced with the online calendar store.
- A tracker part with sensor modules, primarily tracking the user's position and making sure this is transmitted to the infrastructure for monitoring.
- Connection to the POSEIDON context reasoner middleware, sending events there and subscribing to changes to provide context sensitive aid.
- Navigation with guidance for routes created by secondary users. Starting a route can be done by selecting it from a list, or be scheduled in the calendar.

The application is connected to the POSEIDON infrastructure which is also used for the web and other applications. For a detailed description of the user interaction and functionality, see the technical manual, which is reproduced in deliverable D4.5. The prototype systems are described in D5.2. Here in this chapter we present the user interface of the application, with a focus on adaptivity and personalisation.

6.1 Application architecture

The application is developed by Tellu, and is built using Tellu's Android application code framework. This forms a layer between the Android APIs and the application-specific code, and consists of patterns and reusable code for making advanced Android applications. It is developed in research projects, and enabling adaptivity in apps is one of the overall design goals, so it adds mechanisms for dynamic variability in an app on top of what is provided by the Android APIs. It has been further developed in the POSEIDON project. The mobile application architecture and implementation is described in detail in deliverable D5.2. Here we just give a summary of what is relevant for adaptability and personalisation.

One of the most important principles of Tellu's framework is modularity. An application is seen as having two levels, an app level which represents the application as a whole and provides shared services to modules (an app infrastructure), and a module level with the various functionalities. A module has a set of Views. A View is a user interface belonging to a module, and when shown to the user will take up some rectangular area of the screen, possibly all of it. A View is referenced with a string ID, and this is all that is needed to display a View. This means that what Views to show when and where is very flexible and can be reconfigured easily. It is possible to control which View to show from a remote server. The framework includes a generic module for making menus. This takes an abstract menu definition in the form of View IDs and other object forms of app commands, along with icons

and labels, and creates a menu screen with the necessary controller code. Since menus aren't hard-coded, they can easily be reconfigured. This system is used for the main menu of the POSEIDON app.

A module definition can include a list of properties, which the module will typically use for its configuration. This is also fully integrated in the Tellu framework. The app level takes care of persisting property values and monitors changes to the values, amongst many other things, notifying a module when one of its properties has changed value. The framework supports changing these values in several ways, for instance through SMS messages for devices which support it. The framework also has code to provide abstract definitions of user interfaces for data input and generate the interface dynamically. This is used by a framework module which provides a user interface to interact with the configuration properties. The developer just has to provide a definition for the configuration View, and the framework handles the rest.

There are many possibilities for adaptability here. Modules can be highly configurable, and changing the configuration could be done from a server or by the user directly on the device. As the configuration interface is based on a definition, it is very easy to change dynamically, for instance changing which properties the user has access to in the interface.

6.2 User Interface

The user interface has been refined through the prototype iterations. While the first version had a rudimentary UI, later versions have a UI based on the design and UI principles and guidelines compiled in the project, and the feedback from primary users both in the design phase and in the pilots. An important focus has been to keep the user interface clear and simple, with a few large elements on screen at a time. Figure 2 shows the main menu in iterations 1, 2 and 3. System functions were hidden from the main menu in the second version, as these should not normally be used by primary users. From version 2 to 3 we can see an increase in the number of functions (the Map is removed as a separate item, as it is only really used as part of the navigation available from the Routes item), as well as a change in how the time left until the next calendar event is visualised.

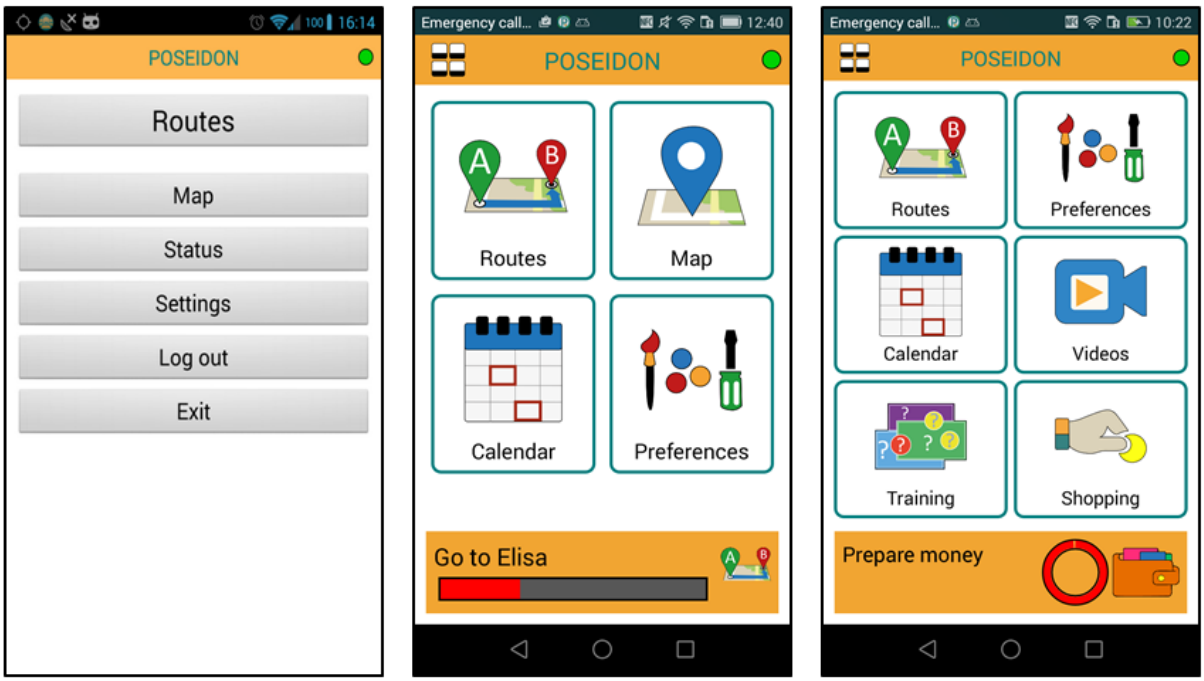


Figure 2: Main menu in prototype 1 (left), 2 (centre) and 3(right)

On initial startup, the user is first met with a login screen. Logging in can be done by a carer or by project personnel when installing the application, and thereafter be persistent, so it need not be done by the primary user. The main menu is the starting point where functionality is selected. The user can always return to this with the icon in the top left. Note that it also shows the current/next event from the calendar.

The preferences screen is shown in Figure 3. Two user preferences have been made available to the primary user in the app: controlling whether position tracking data is made available to those with account access, and the colour theme which we will discuss in section 6.4.3.

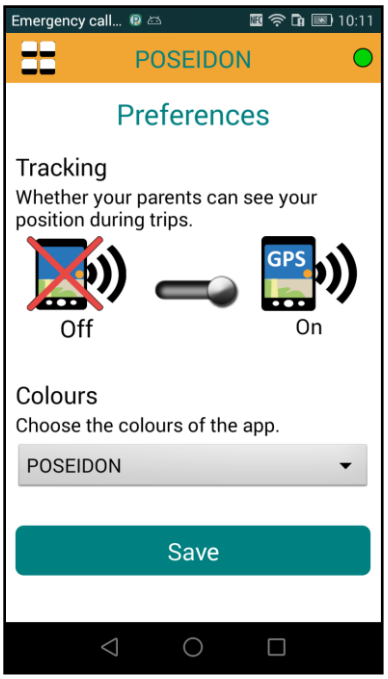


Figure 3: Preferences screen

Figure 4 shows examples of the main navigation screen in the three main iterations. The first prototype simply drew the route on the map and placed a marker at the current location. In the second version, in addition to better visualisation of the route and position, the map would switch between a 2D and a 3D view depending on the angle at which the phone was held. The 3D view would rotate to match the direction the phone was facing, using phone sensors. It was hoped that this would provide a less abstract and more understandable map view. However, the switching between modes and the fact that the sensor-based map direction could sometimes be inaccurate meant the 3D mode was found to cause some confusion in the first pilot. It was therefore disabled for the second pilot, instead focusing on 2D map improvements in the third iteration.

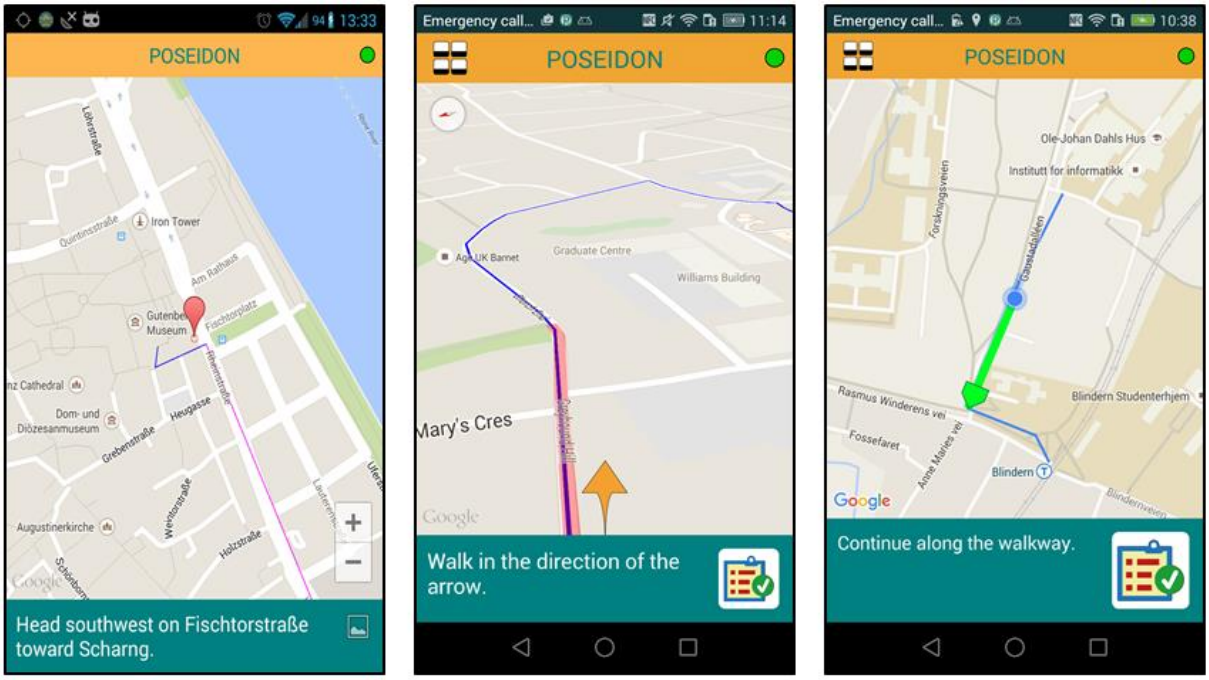


Figure 4: Navigation map in prototype 1 (left), 2 (centre) and 3 (right)

Figure 5 shows some screens from the calendar interface. The first image is the month view added in the third iteration – a calendar with all the days in a month. Dates with events are highlighted, and the dates can be pressed to see the date view listing events for this day. Note the colour-coded weekday labels. This colour-coding is different for each country. The centre image is an example of the date view, listing events. Pressing an event brings up the event detail view, as shown on the right. From here it is possible to edit or delete the event. The calendar also has other views, such as for editing an events or listing the next upcoming events.

We see that icons are used extensively in the calendar interface. This is a mix of built-in icons and user content. Each event can have an icon, entered creating the event through the web interface. The icon serves as a representation of the event, like its title, and is shown together with the title when listing events. Used correctly, it makes it possible to identify events without reading. Dates are labelled with the built-in colour-coded weekday icons. When an event is starting in less than one hour, a circle indicates the time left, filling with red colour to indicate how much of the hour is up. There are also icons to indicate that an event has a route for navigation, or that an event has a list of instructions.

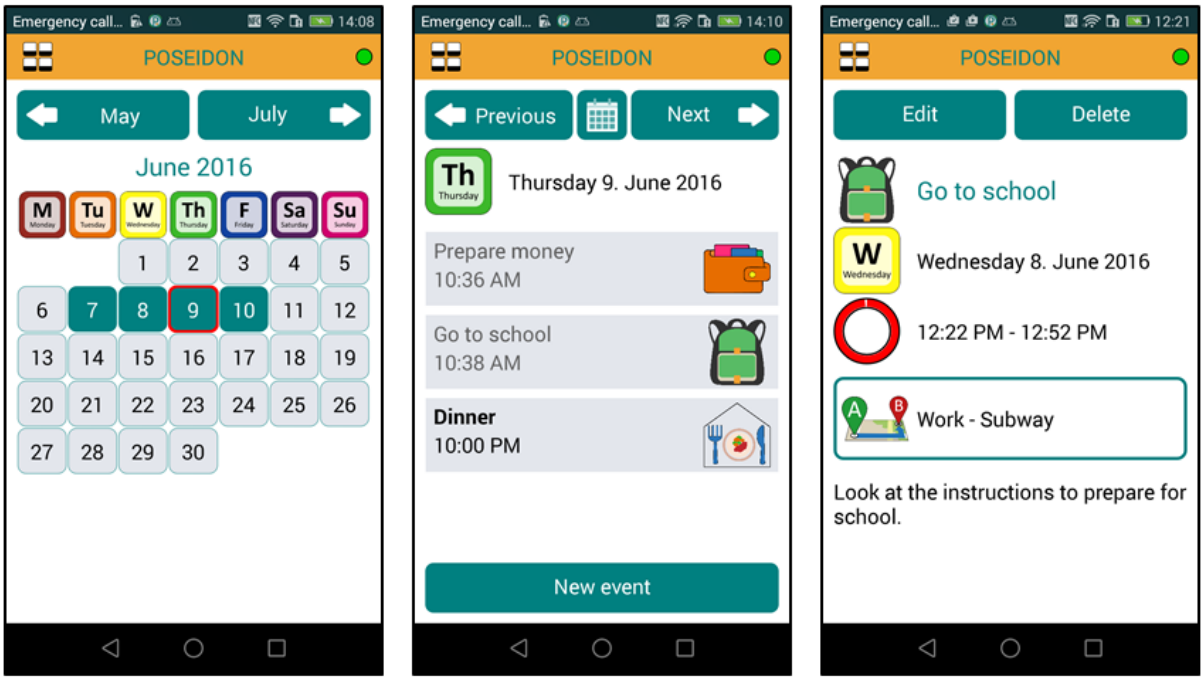


Figure 5: Screenshots of the calendar interface: month view, date view and event details

Finally, Figure 6 shows some examples of dialogue boxes in the application. On the left is a notification shown at the alarm time specified for an event, to notify of the upcoming start of the event. A slightly different notification can be shown at the start time. If the event has a list of instructions, it will then show each instruction in turn. The central image shows an example of this. The screen on the right is an instruction during navigation, showing the text and image defined for the current step of the trip. Refer to the user manual for a larger set of screenshots.

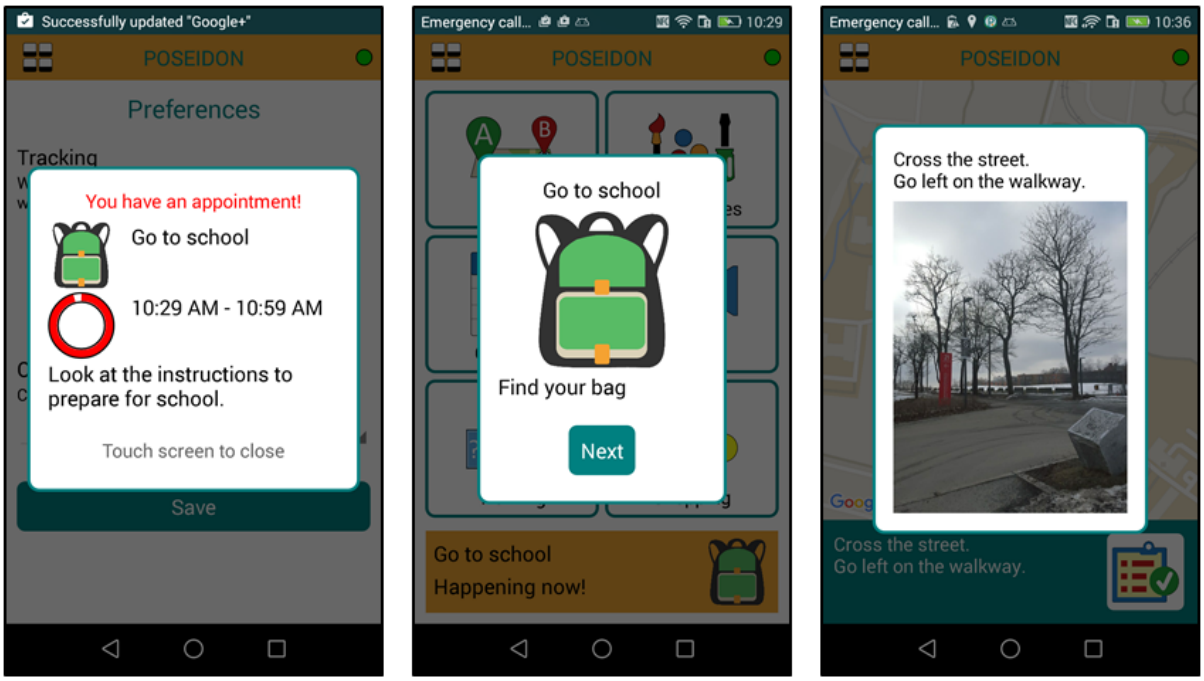


Figure 6: Pop-up dialogue boxes

6.3 Adaptivity

We describe the Android user interface system of context-dependent resources and system-wide settings in deliverable D4.5. The system provides some mechanisms for adaptiveness and adaptability, such as automatically adapting to screen size and resolution. It also places restrictions on what we can change and how, and much of it is controlled by the Android system rather than by the application.

Another set of mechanisms for dynamically changing the application is provided by the Tellu application framework, such as module properties and dynamic views such as menus. We have implemented one change in menu items based on application mode. See Figure 7. On the left is the default main menu. The top left item gives access to routes defined for the user, to start navigation. When in navigation mode, we are usually in the map screen, but it is always possible to return to the main menu. If so, we see that the two top menu items have changed. Instead of going to the list of routes, we can now return to the map screen of the navigation. We also need to be able to stop the navigation mode. We didn't want to extend the menu, as that would either make the items smaller or require scrolling. Changing preferences are a low priority and not something the user is likely to do in the middle of navigation, so we switch out the *Preferences* item for *Stop navigation* in this mode.

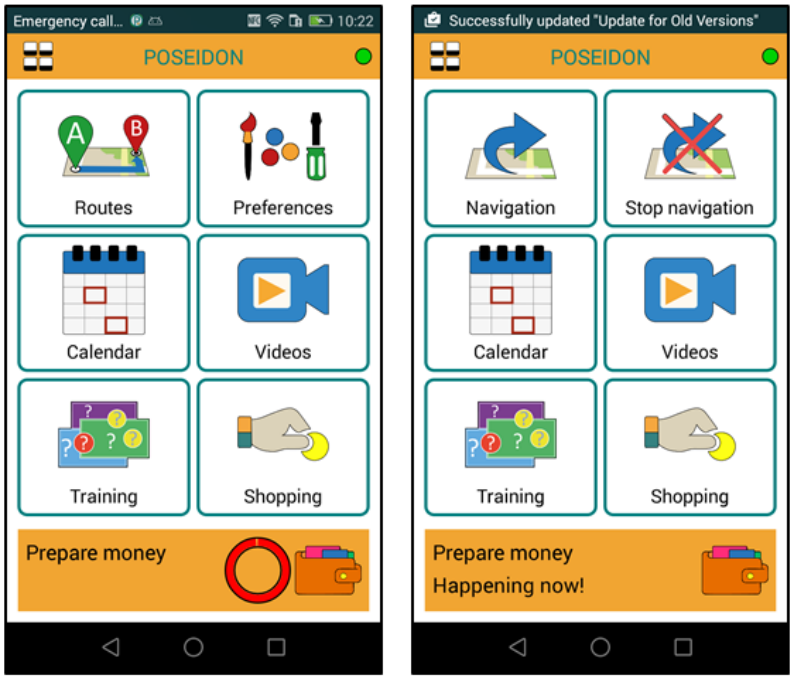


Figure 7: Menu in normal and navigation mode

Another relevant mechanism for adaptivity is POSEIDON's context awareness. The application is connected to the context reasoner, as long as this is installed on the same device. It receives context events from the reasoner, such as if the user appears to have trouble navigating or if the user is about to leave home and the weather requires attention. These are situations where the application should offer assistance to the user, and it does this by showing a notification and possibly offering some actions. Figure 8 shows a weather-based notification, giving advice. In the case of navigation trouble, the user is given options such as calling their carer or ending navigation (what we interpret as having trouble might just be because they left it on but don't intend to follow the route any more).

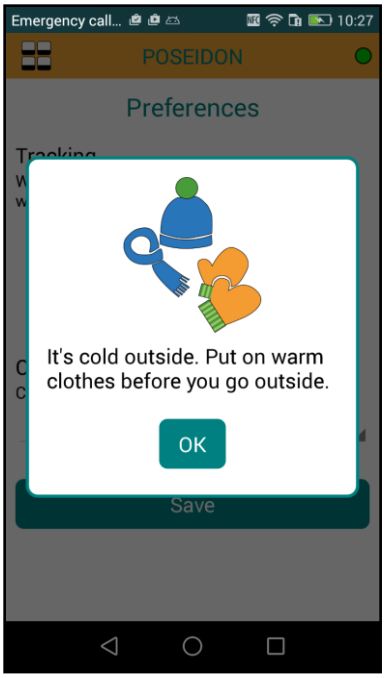


Figure 8: Weather notification based on context awareness

Other than these limited examples, we found little reason to do automatic changes in the user interface and interaction. We have been experimenting and looking for use cases, but found it difficult to find good cases for automatic changes. Such changes should only happen when and how it makes sense to the user, such as modifying the menu in navigation mode. Automatic changes which appear to come "out of nowhere" will confuse the user and be disruptive, even if the underlying system has good intentions and believe it has found an adaptation suited to the specific user. Even switching between a 2D and 3D map mode based on how the phone was held was confusing for our users. We tried it in pilot 1, but dropped it for pilot 2. We believe that controlled personalisation, such as through the user setting preferences, is much more important for our users. We discuss this in the next section.

6.4 Adaptability and Personalisation

Personalisation takes two main forms in this application, and both are supported by the POSEIDON infrastructure. One is in the configuration of the application and the device it runs on, based on personal preferences. For this, we use the POSEIDON account to store preferences, so that these are independent of the device and can be managed through the web application. We present the most relevant preferences below. The other form is personal content, typically entered into the system by secondary users. The POSEIDON framework defines data formats and provides online storage for the content.

6.4.1 Language

All three pilot languages – English, German and Norwegian – are supported by the application. All text the end user will see is provided in all three languages. This is done with Android's resource system, so that the language in the application is controlled by the language option in the device settings.

6.4.2 Font Size

As with language, font size is a system setting in Android, and an app using Android's resource system has no direct control over this font size scaling. Text size is often one of the first things to be mentioned as an attribute for which adaptability is useful. However, letting it be variable poses a challenge for designing the user interface. As people with Down's syndrome tends to have poor vision, we wanted

to make the text as large as possible while still being able to fit enough content on screen at once to avoid excessive scrolling. So we have carefully designed the layouts and default font sizes with this in mind. This is already challenging because we must account for the differences between languages – a translation may in worst case be significantly longer or shorter than the text we originally design a layout and font size for. And of course much content is provided by users, such as calendar event names and descriptions, and this we have little control over. If the user may also change the font size, it will easily break the carefully designed layouts (not in the sense that the app doesn't work, but in the sense that it will be harder to read and interact with the interface).

The application is intended for fairly large phones, with screen size around 5.5 inches, which is what we have been using in the pilots. We have tried to make sure that on such screens, the layouts work as intended even if the font size is increased one level above the default "normal" size. As this font scaling is used for the Android system and all apps which are properly designed for the system, a larger font size may be useful for the use of other apps, which is why we used the larger size as our target setting. However, we have used much larger fonts than is normal for mobile apps, so even with the system setting on normal the text is quite large. The result is that we allow the system font size to be adjusted, and our app will abide by this setting, but we recommend using either normal size or one step up.

6.4.3 Visual Theme

It is possible to implement multiple visual themes for an application, changing the user interface "skin" completely. The Tellu application framework facilitates this, with the theme being an app property which can easily be changed. While this could be used to let users choose colours based on personal preferences, we have used it to provide a high-contrast theme for those who benefit from this due to poor eyesight. The theme is one of the preferences we store in the POSEIDON account, so that it can be changed remotely. It can be changed through the web application. It can also be changed in the preferences screen of the app, so that it is easy to do for the primary users. In either case the setting is updated in the POSEIDON cloud.

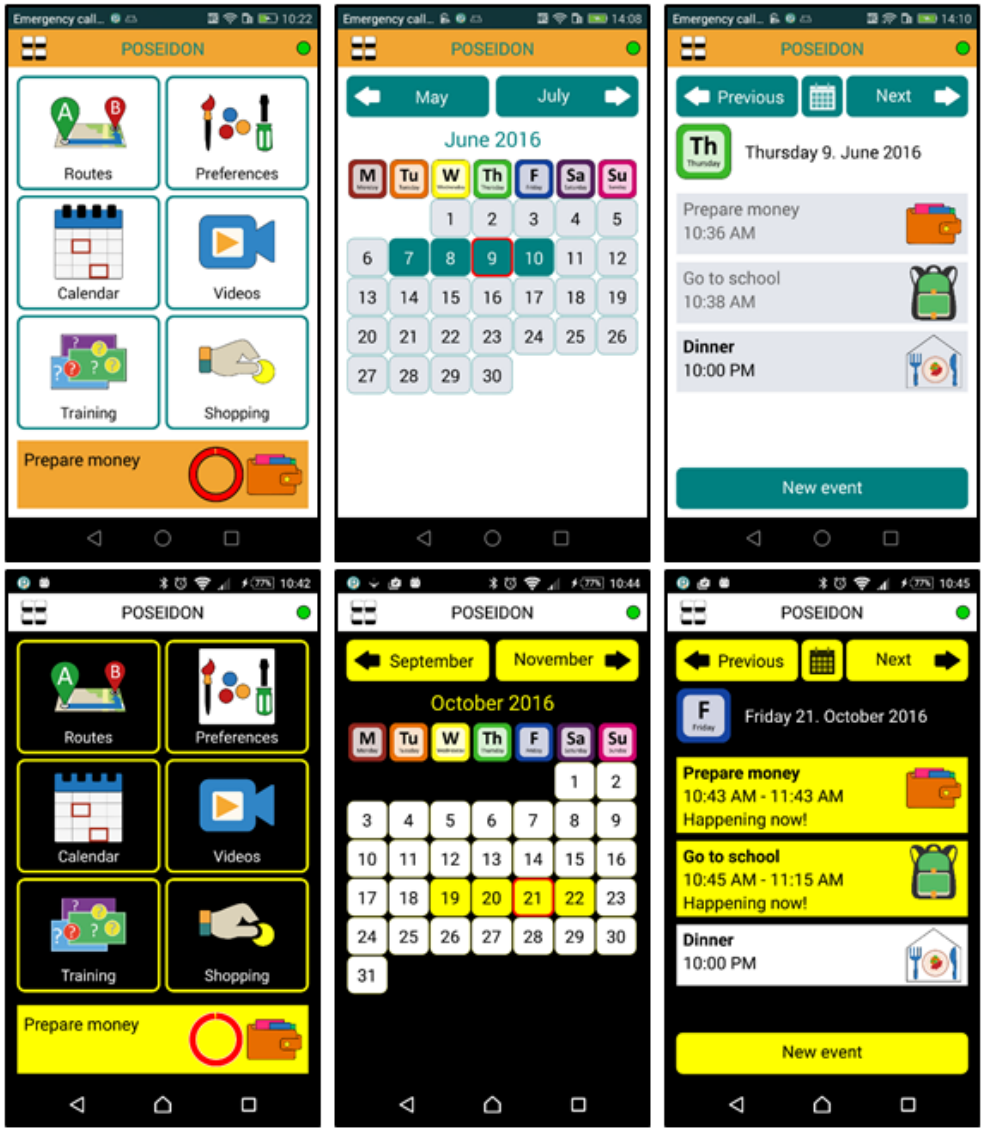


Figure 9: Two colour themes

Figure 9 shows the two themes in the third iteration prototype, with the main menu, month calendar and date screens in both themes. The high contrast theme is based on yellow and white contrasting with black. Note how some of the application icons, such as arrows and calendar on navigation buttons, are also themed. Other icons, such as in the main menu, are not. And of course user content, such as calendar event icons, are not affected by theme. This can be a challenge, as an icon which has good contrast with one background may not work so well on another background colour. Providing icons do in any case require some care.

6.4.4 Calendar alarms

There are some preferences regarding how upcoming calendar events are communicated. Keep in mind that carers can specify alarms for events or leave them without any alarms. The notification at alarm time has sound and vibration. There is a preference for whether this should be continuous until acknowledged, or less insistent. There is also a preference to specify whether the user should be notified at the start time of the event, in addition to any event alarms. If so, a notification is given at the start time of all events.

6.4.5 Personalised content

We will not go into details on user-defined content here. It is in a sense external to the application itself, but it is a very important aspect of personalisation, so we include it in this overview. Refer to D5.1 chapter 4.4 for how the POSEIDON framework supports personalisation, or the additional deliverable R2 which is dedicated to personalisation.

Personalised content specifically supported by this application:

- Calendar events: An event can have title and description text (read with text-to-speech), icon/image and video. The icon gives a visual alternative to the title, and the video gives an audio-visual alternative to the description.
    - Alarms: Alarms can be specified for events, to notify the user ahead of time.
    - Instruction list: Step-by-step instructions can be specified for an event. Each item in the list can have text, sound, image and video.
- Routes for navigation: Routes are defined by secondary users. They consist of a number of steps, with each step giving an instruction for reaching the start of the next step. Each instruction can have text, image and recorded sound.
- Video list: A list of videos can be defined, intended to be used for instructional purposes.

7 Conclusions

In this document, we explored adaptivity and personalisation in the mobile interface. We can see that there are many different approaches already for adaptive UIs. We feel that after some investigation, it is safer to use context to drive notifications, reminders, and prompts instead of fully adaptive UIs. We have described initial adaptation and personalization ideas for pilot 1. Then we addressed UI feedback from pilot 1 and how this was addressed for pilot 2 regarding the main POSEIDON App and the other POSEIDON components. Finally we describe adaptation and personalisation features in the main POSEIDON App in detail.

As lessons learned, we conclude:

- UI adaptivity has to be used very carefully because the system might change in a for the user unexpected way.
- Mostly personalisation is requested from the end users.
- The use-cases for adaptivity of the UI have to be very carefully selected, more important are personalisation options.
- Mostly context related notification messages have been used in the identified use-cases.

8 References

Apel, S., Kastner, C. and Lengauer, C. (2013). Language-independent and automated software composition: The FeatureHouse experience. *IEEE Transactions on Software Engineering*, 39, p.63–79. [Online]. Available at: doi:10.1109/TSE.2011.120.

Appeltauer, M., Hirschfeld, R., Haupt, M. and Masuhara, H. (2011). ContextJ: Context-oriented Programming with Java. *Information and Media Technologies*, 6, p.399–419. [Online]. Available at: http://ci.nii.ac.jp/naid/130000770579/en/.

Batory, D. (2004). Feature-oriented programming and the AHEAD tool suite. *Proceedings. 26th International Conference on Software Engineering*. [Online]. Available at: doi:10.1109/ICSE.2004.1317496.

Behan, M. and Krejcar, O. (2012). Adaptive Graphical User Interface Solution for Modern User Devices. *Intelligent Information and Database Systems*, 7197, p.411–420.

L. Chittaro and R. Ranon, "Adaptive Hypermedia Techniques for 3D Educational Virtual Environments," in IEEE Intelligent Systems, vol. 22, no. 4, pp. 31-37, July-Aug. 2007.

Criado, J., Vicente-Chicote, C., Padilla, N. and Iribarne, L. (2010). A Model-Driven Approach to Graphical User Interface Runtime Adaptation. In: *Proceedings of the 5th Workshop on Models@run.time at the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems*, 2010, p.49–59.

David, L., Endler, M., Barbosa, S. D. J. and Filho, J. V. (2011). Middleware Support for Context-Aware Mobile Applications with Adaptive Multimodal User Interfaces. In: *2011 Fourth International Conference on Ubi-Media Computing*, July 2011, IEEE, p.106–111. [Online]. Available at: doi:10.1109/U-MEDIA.2011.50 [Accessed: 14 July 2014].

Findlater, L. and McGrenere, J. (2004). A comparison of static, adaptive, and adaptable menus. In: *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*, 25 April 2004, New York, New York, USA: ACM Press, p.89–96. [Online]. Available at: doi:10.1145/985692.985704 [Accessed: 23 July 2014].

Grolaux, D. (2007). *Transparent Migration and Adaptation in a Grapical User Interface Toolkit*. Universite Catholique de Louvain.

Hanumansetty, R. G. (2004). *Model Based Approach for Context Aware And Adaptive User Interface Generation*. Virginia Polytechnic Institute and State University.

Holzinger, A., Geier, M. and Germanakos, P. (2012). On the development of smart adaptive user interfaces for mobile e-business applications - towards enhancing user experience - some lessons learned. In: *DCNET/ICE-B/OPTICS*, 2012, p.205–214.

Kramer, D., Oussena, S., Komisarczuk, P. and Clark, T. (2014). Using document-oriented GUIs in dynamic software product lines. *ACM SIGPLAN Notices*, 49 (3), ACM, p.85–94. [Online]. Available at: doi:10.1145/2637365.2517214 [Accessed: 17 July 2014].

Lavie, T. and Meyer, J. (2010). Benefits and costs of adaptive user interfaces. *International Journal of Human-Computer Studies; Measuring the Impact of Personalization and Recommendation on User Behaviour*, 68, p.508–524. [Online]. Available at: doi:http://dx.doi.org/10.1016/j.ijhcs.2010.01.004.

Marathe, S. and Sundar, S. S. (2011). What drives customization? In: *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*, 7 May 2011, New York, New York, USA: ACM Press, p.781. [Online]. Available at: doi:10.1145/1978942.1979056 [Accessed: 23 July 2014].

McGrenere, J., Baecker, R. M. and Booth, K. S. (2002). An evaluation of a multiple interface design solution for bloated software. In: *Proceedings of the SIGCHI conference on Human factors in computing systems Changing our world, changing ourselves - CHI '02*, 20 April 2002, New York, New York, USA: ACM Press, p.164. [Online]. Available at: doi:10.1145/503376.503406 [Accessed: 11 July 2014].

McNaull, J., Augusto, J., Mulvenna, M. and McCullagh, P. (2014). Flexible context aware interface for ambient assisted living. *Human-centric Computing and Information Sciences*, 4 (1), Springer, p.1–41. [Online]. Available at: doi:10.1186/2192-1962-4-1 [Accessed: 22 July 2014].

Page, S. ., Johnsgard, T. J., Albert, U. and Allen, C. . (1996). User customization of a word processor. In: *CHI*, 1996, p.340–346.

Paymans, T. F., Lindenberg, J. and Neerincx, M. (2004). Usability trade-offs for adaptive user interfaces. In: *Proceedings of the 9th international conference on Intelligent user interface - IUI '04*, 13 January 2004, New York, New York, USA: ACM Press, p.301. [Online]. Available at: doi:10.1145/964442.964512 [Accessed: 14 July 2014].

Rodriguez-Gracia, D., Criado, J., Iribarne, L., Padilla, N. and Vicente-Chicote, C. (2012). Runtime adaptation of architectural models: an approach for adapting user interfaces. In: *Proceedings of the 2nd International Conference on Model and Data Engineering*, 2012, Springer Berlin Heidelberg, p.16–30.

Savidis, A. and Stephanidis, C. (2010). Software refactoring process for adaptive user-interface composition. *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '10*, p.19. [Online]. Available at: doi:10.1145/1822018.1822023.

Stuerzlinger, W., Chapuis, O., Phillips, D. and Roussel, N. (2006). User interface facades: towards fully adaptable user interfaces. In: *Proceedings of the ACM Symposium on User Interface Software and Technology*, 2006, p.309–318. [Online]. Available at: doi:10.1145/1166253.1166301.

Troyer, O. De, Kleinermann, F. and Ewais, A. (2010). Enhancing Virtual Reality Learning Environments with Adaptivity : Lessons Learned. In: *Proceedings of the 6th international conference on HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering*, 2010, p.244–265.

Zeidler, C., Lutteroth, C. and Weber, G. (2013). An evaluation of advanced user interface customization. In: *Proceedings of the 25th Australian Computer-Human Interaction Conference on Augmentation, Application, Innovation, Collaboration - OzCHI '13*, 25 November 2013, New York, New York, USA: ACM Press, p.295–304. [Online]. Available at: doi:10.1145/2541016.2541037 [Accessed: 23 July 2014].