Cloud-based UI personalization
ACCESSIBILITY MADE EASIER
CHRISTOFFER FRIBERG
Cloud-based UI personalization: accessibility made easier

Svensk titel: Molnbaserad personanpassning av användargränssnitt

Examensarbete inom datalogi

Program: Civilingenjör, datateknik

Författare: Christoffer Friberg, chfr@kth.se

Handledare: Olle Bälter

Examinator: Jens Lagergren

Uppdragsgivare: Omnitor AB

Datum: 2014-02-06
Abstract

This project had two primary goals. The first was to use existing knowledge about mobile accessibility, smartphone accessibility and touchscreen usability to improve and extend the usability of two Total Conversation applications for Android. Total Conversation is the combination of video, audio and real-time text conversations frequently used by the deaf and hard of hearing. Total Conversation has wide-spread support in both the mobile and PC realms, but the primary benefit is that it’s also supported by standalone video phones. The second goal was to integrate these applications with the EU-funded Cloud4all framework. The purpose of Cloud4all is to develop a framework for automatic cloud-based user interface customization based on the individual needs and preferences of each user. Both goals were met and functional prototypes of both applications were developed and tested. A Cloud4all user can log in and have their cloud-stored preferences applied with no further action taken. Supported Cloud4all preferences are font size and high contrast theme selection. Support for icon size adjustments is implemented but not yet supported by the Cloud4all framework.

Sammanfattning

Contents

1 Introduction .................................................................................................................. 1
  1.1 Project goals ........................................................................................................... 1
    1.1.1 eCmobile and eCtouch .................................................................................. 2
  1.2 The Cloud4all project and GPII .......................................................................... 2
    1.2.1 Authentication methods ............................................................................... 3
    1.2.2 Architecture overview .................................................................................. 3
  1.3 Previous work ....................................................................................................... 5
2 Needs and requirements ................................................................................................. 6
  2.1 Test cases ................................................................................................................ 7
    2.1.1 A deaf user ..................................................................................................... 7
    2.1.2 An elderly user ............................................................................................... 7
    2.1.3 A deafblind user ............................................................................................ 8
  2.2 Specific needs identified ........................................................................................ 8
3 User interface customization ......................................................................................... 10
  3.1 Default eCtouch UI .............................................................................................. 10
  3.2 Default eCmobile UI ............................................................................................. 12
  3.3 Solving problems with adjustments ....................................................................... 15
    3.3.1 Text size is too small .................................................................................... 15
    3.3.2 Icons and buttons are too small .................................................................... 16
    3.3.3 Other issues .................................................................................................... 16
  3.4 Limits on customizability ....................................................................................... 17
4 Integrating with Cloud4all .......................................................................................... 17
  4.1 Matching needs and preferences .......................................................................... 17
    4.1.1 The transformer module .............................................................................. 17
    4.1.2 The matchmaker module ............................................................................. 19
    4.1.3 Default preferences ...................................................................................... 19
    4.1.4 Custom preferences ...................................................................................... 20
  4.2 Accessing Cloud4all preferences ........................................................................... 20
  4.3 Preference sets for test cases .................................................................................. 20
5 Programmatic UI tweaking .......................................................................................... 22
1 Introduction
Mobile technology has become a staple of everyday life for a large number of people in recent years. Our mobile devices are becoming more powerful, more useful and more affordable. With the growing adoption of tablets and touchscreen based smartphones, mainstream handheld devices are moving away from physical buttons and knobs. Many early Android based devices used up to four physical buttons for navigation, while the iPhone and iPad series of devices have been using a single button since their inception. Nowadays most modern Android devices have one or zero physical buttons, relying instead on software buttons. Buttons for volume control and power on/off remain in hardware, but for the most part modern mobile devices are almost entirely touch based. For the general public, this may seem like a harmless development. For many, it's been a step forward in usability and productivity. For a certain set of users, however, these developments may cause more problems than they solve. For users with visual or motor impairments, touchscreen devices can pose certain problems. A blind or low-vision user cannot feel their way to the correct button like on a device with a physical keypad. A person with a motor impairment (brought on by age, disease or other factors) may have difficulties accurately and quickly touching the intended part of the screen. Other potentially difficult situations may arise for users who are hard of hearing, notably some issues related to sound volume. These issues can stem from both hardware and software limitations, but nonetheless they may necessitate the use of various peripherals to alleviate them. Some hearing aids can be directly connected to mobile devices like a standard headset; other solutions can involve the use of a small external amplifier to boost the volume. Similar problems may be faced by deaf users, but with some key differences. A deaf user relies even more on peripherals for various functions that would otherwise be performed by sound signals. Alerting systems using flashing lights and vibrations are used to indicate things like incoming calls and messages.

Among the elderly, many individuals may have a combination of these impairments, although possibly to a less severe degree. When these issues occur simultaneously, it requires assistive technologies to interoperate and complement each other in order to provide a usable experience.

1.1 Project goals
The goal of this project is to investigate and implement a prototype system for automatic user interface (UI) customization of two Android applications (apps),
based on the specific needs of the user. The apps in question are eCTouch and eCMobile, the tablet and smartphone versions of Omnitor AB's Total Conversation apps. Total Conversation is a standardized concept similar to traditional video calls, but with added real-time text chat. It allows users with visual and hearing impairments to communicate using a computer, smartphone, tablet or dedicated Total Conversation device. The UIs of these apps will be automatically customized to suit their particular needs, as specified by the user in the Cloud4all framework. A blind user's needs are different from those of a low-vision user, which are both different from the needs of a user with impaired motor control. The user identifies his- or herself to the system, which then performs the required interface customizations before the app is launched.

1.1.1 eCMobile and eCTouch

eCMobile and eCTouch are two Android apps developed by Omnitor AB. They implement the Total Conversation concept and can communicate with a wide array of other Total Conversation devices. The interfaces of these apps have proven successful and usable, but it stands to reason that something designed for everyone can't be ideally suited for each individual. As such, integrating with the Cloud4all project was a way to further improve the usability of the apps and take one step further to accommodate those with particular needs.

1.2 The Cloud4all project and GPII

Cloud4all is a project funded by the EU, in particular the European Commission FP7 program (EU-FP7, 2011). The Cloud4all project aims to build a framework for interface customization based on user needs, across a wide range of different platforms. Smartphones and tablets are two of the target platforms, but the project also aims to target things like home computers, automatic teller machines, ticket machines, and more. In a video produced by the Cloud4all group, the intended functionality of the framework is demonstrated (Cloud4all, 2012). The user identifies herself to the ticket machine with an RFID ring that either contains her preferred interface settings or some identifying credentials that allows the ticket machine to fetch her settings from the cloud. The necessary UI changes are applied, in this case turning the screen to black and white, and she goes on to purchase her ticket.

The Cloud4all project is tasked with building and testing key components of the Global Public Inclusive Infrastructure (GPII), namely the components that together make up the main technical parts of the GPII. The GPII was designed by the Raising the Floor Consortium, a group of organizations working towards ensuring the usability and accessibility of modern computer systems. In this report, any reference to the Cloud4all project, platform or framework refers to
the combined system of components being developed by the Cloud4all participants as part of the GPII.

1.2.1 Authentication methods
The Cloud4all platform supports several different types of login methods, in order to suit the individual needs of users with different capabilities and limitations. The first and most well-known login method is of course a user name and password combination. While that would work well on a computer or other keyboard equipped device, it would be impractical and slow on a ticket machine. In addition to that, there is no guarantee that the user is able to use a standard keyboard effectively (or at all). The use of a name and password combination also requires the user to memorize and recall these bits of information in potentially stressful situations, which can be challenging to users with certain cognitive impairments.

The RFID token shown in the video is an elegant solution for many scenarios, but it too has some drawbacks. Privacy is a concern since the RFID tag could be read by anyone with a compatible reader, meaning that whatever data is on it must not reveal personal information or other sensitive details. Device compatibility is also an issue, since RFID support is still rare outside of the smartphone realm. While most modern Android smartphones have RFID (NFC) readers, the same cannot be said for tablets. At the time of writing, the only line of tablet devices to consistently ship with RFID support is Google’s Nexus series. RFID is a good solution for public terminals since it makes the system more resistant to vandalism, because there are no buttons, connectors or other external features that can be damaged or destroyed.

For private or less public devices (such as a library or internet café), USB dongles can be practical. Since they require no specialty hardware (unlike RFID, for instance), they can be used with any computer and are almost as easy to use as an RFID tag. A few other methods of user identification are also being considered, such as barcodes (or QR codes) and facial recognition. When it comes to facial recognition especially, privacy is a very important issue because you’d need to keep a database of facial data without running afoul of the world’s various privacy laws.

1.2.2 Architecture overview
The Cloud4all system consists of eight components, each performing a separate task.

The Flow Manager is responsible for passing information between components and orchestrating the whole process from start to finish. When a user authenticates to a computer, smartphone, public terminal or other supported
device, the **User Listener** picks it up and sends the user’s information to the Flow Manager. The Flow Manager queries the **Preferences Server** to find out if the user is known and to retrieve the user’s preferences. The Flow Manager then asks the **Device Reporter** for information about the current device, such as display resolution and supported input methods. The Flow Manager then asks the **Matchmaker** to find a compatible solution (either from the local device or online) that suits the user’s needs. The Matchmaker interacts with the **Solutions Registry**, which keeps track of available solutions and their capabilities, and returns the result to the Flow Manager. The Flow Manager then uses the **Lifecycle Manager** to first configure the solution in question (for instance a screen magnifier) via a **Settings Handler**, and finally launches the solution.

The components are platform independent to as high a degree as possible, to this end most of them are written in JavaScript intended to run on the Node.js platform. Some components are platform dependent by necessity, especially those that interact with operating system specific APIs. The Settings Handlers that interact with the Windows registry or GNOME GSettings are platform dependent by definition, and likewise for Device Reporters. Reading detailed information about the various systems’ capabilities requires software tailored to that specific platform.

The cloud based nature of the system may not be immediately obvious from this description, and indeed all of these components can run on a local device, completely offline. The modular nature of the system combined with the way components interact does allow for some components to run on a remote server. The Flow Manager, Preferences Server and Solutions Registry are all meant to be run on a remote server, administrated by some central authority. This authority can be device manufacturers, the GPII consortium or anyone else who wants to offer this type of service. The other components run on the local device by necessity, since they need to interact with the device itself.

The Android version of the Cloud4all architecture is the same in terms of components, but there are some differences to note. All the JavaScript components can run on an Android device via the Anode framework, a customized port of the Node.js platform for Android. It allows Node.js scripts to run as they normally would, but also to interact with the Android device via bindings to the underlying Java system. Furthermore, the Android system provides a very flexible way for apps to interact with each other in arbitrary ways, using so-called Intents. This allows the Settings Handlers and Lifecycle Manager to perform their tasks in a clean and well documented fashion.
1.3 Previous work

Accessibility and usability of computer programs is a well-researched area, the field of human-computer interaction (HCI) has existed in some shape or form for over 20 years (Savage, et al., 2005) and is continually refreshed by new innovations in both hardware (new input methods, display devices) and software. On the other hand, mobile HCI is a much younger field, owing to the fact that mainstream mobile devices have been in a mature state for a much shorter time. The field of usability and accessibility research aimed specifically towards mobile touchscreen based devices is even more in its infancy. Mobile touchscreen based devices truly entered the mainstream with Apple's 2007 launch of the original iPhone, and with the relatively full-featured operating systems and applications that such devices bring, there is a large amount of research to be done. The small form factor combined with a fully touch-based input method makes these devices quite different from their predecessors.

Touchscreens themselves have been studied in non-mobile devices, of particular interest here are studies regarding the usability of touchscreens for users with some type of disability. Hwangbo et al. (2013) studied the effects of target size and spacing on touchscreen pointing accuracy in elderly users, as well as different feedback methods to indicate a successful hit. They found that pointing performance improved as target size and spacing increased, and their test subjects indicated that larger sizes eased cognitive strain rather than physical strain. Chen et al. (2013) similarly found that individuals with motor control disabilities benefited strongly from increasing target sizes in their touchscreen study. Target sizes up to 30 mm square were tested, and while non-disabled participants showed little improvement when going above 15 mm, the disabled participants showed improvement with every increase in target size.

There is a large number of age related ailments (Schieber, 2003) that can negatively impact the ability to use small mobile devices, ranging from vision and hearing to memory and attention span. Naturally, hearing loss can negatively impact the ability to use mobile devices for people of any age. As technology has progressed, these modern devices have opened new doors for communication for users with some degree of hearing loss. In a study with a group of volunteer deaf participants, Liu et al. (2010) found that 78% of the interview participants sometimes communicate in sign language via video calls. Furthermore, a study by Chiu et al. (2010) identified a number of unfulfilled needs of deaf users of mobile phones. Among others, relatively small displays make it more challenging to see small details, an important aspect when trying to read sign language. Additionally, study participants found it challenging to conveniently communicate with non-disabled people. The latter point hints at a need for streamlined integration with a relay service, either sign-to-speech (in the case of video calls) or text-to-speech. There are also environmental or
situational factors to consider when it comes to mobile devices and accessibility. Kane et al. (2009) found in their study that low-vision users experienced problems reading the displays on their devices under anything but ideal lighting conditions. It’s interesting to note, in a study by Harper et al. (2011), they found many similarities between the difficulties faced by users with disabilities and non-disabled people who are *situationally impaired*. Situational impairment refers to challenges imposed by the circumstances and environment of the user. Examples would be relatively common situations like trying to read a dim phone display in bright sunlight, or talking on the phone next to a noisy road. Every non-disabled person has found themselves in a similar situation at some point, and it offers an illuminating glimpse into an otherwise unfamiliar world.

Using interface customization to adapt to user needs has been previously studied in various ways. Sloan et al. (2010) set up a small theater play for an audience of older computer users and demonstrated how an automatic interface adaptation system might work. They then interviewed the audience members to determine how different solutions were received. They found a generally positive attitude towards the proposed system, but some users felt that the automatic nature of it took away too much control. Gajos et al. (2007) demonstrated a system for automatically generating a modified UI based on an individual performance test. Their work took into account both low-vision users as well as motor impaired users, and a combination of both. They found that by tailoring the size and appearance of controls, the participants reached an average of 20% time savings in completing a predefined task.

## 2 Needs and requirements

Every user has different needs and requirements depending on what disabilities they may or may not have. A common way to accommodate users with disabilities is to use presets for various groups of people who are deemed to have similar needs. An example would be to provide presets like "blind", "color blind" or "large controls needed" and applying a set of changes deemed appropriate by the developer. While this is better than no such adjustments at all, it is not fine-grained enough and unduly forces a user to shoehorn him- or herself into a label that may not be accurate at all. An elderly user with some degree of motor function impairment could be well served by having a "large button" mode, but if the same user also has some type of color vision deficiency there may not be a way to enable that mode simultaneously.

The Cloud4all model hopes to avoid such issues by allowing each user to specify their needs and requirements in a central place, rather than choosing between
a pre-determined set of disability labels for each device. A user would then choose what adjustments they want, such as "large text", "high contrast text" or "loud volume", and these would be applied individually by participating devices and applications.

This way of handling it avoids the often sensitive issue of labeling a person with a diagnosis which may not be wanted or even accurate.

2.1 Test cases
To simulate real world uses of the intended functionality, several test cases have been constructed to emulate how different people (with varying needs) would use and benefit from the Cloud4all integrated system. These test cases represent a small subset of those people who may not fit very well within the regular pre-defined labels that are often used to provide accessibility. They are based on the target audience for eMobile and eTouch, which means that they mainly involve users with some degree of hearing loss. These cases will be used throughout the report to showcase examples of results and the processes of arriving at said results. Each of the described needs is assigned a number to easily refer to them later in the report, the numbers are of the form User Need 1, User Need 2 (UN1 and UN2 for short), etc.

2.1.1 A deaf user
This case is modeled after an average deaf user. No particular disabilities aside from a complete or very severe loss of hearing, but otherwise normal vision (with or without some type of corrective lenses), normal cognitive abilities and normal motor control. In most situations, this user can use applications without any modifications, but complications arise when the applications emit sound.

- If the sound is meant to notify the user that their attention is needed, for example if an email or message has arrived, the sound should be accompanied or replaced by either vibration or visual indication, or both. (UN1)
- Applications need an easy way to turn off all sounds. An unfamiliar application might be playing sounds that the user can’t hear, potentially quite loudly. In a public setting, this can be embarrassing and in certain places (like libraries) it can cause problems for the user. (UN2)

2.1.2 An elderly user
Modeled after a user with a combination of several impairments, with varying degrees of severity. The user’s vision is impaired, to such a degree that corrective lenses allows the user to read enlarged text with minor difficulties. Motor function is lessened, partly due to tremors and partly due to decreased
muscle control. This leads to diminished accuracy when pointing and moving. Hearing is also impaired, but only moderately so. With a hearing aid, most sounds can be perceived and understood. The user is also not very experienced in using computers and modern technology.

- The application’s controls, such as buttons and checkboxes, need to be enlarged to compensate for the lack of pinpoint pointing accuracy. (UN3)
- Similarly, the text needs to be enlarged to compensate for the user’s low vision. (UN4)
- Sound controls need to be prominent to allow the user to quickly and easily adjust the volume without delving into menus and settings. (UN5)
- The language and terminology used may need to be adjusted to ensure that someone with limited knowledge and experience can still understand what all the controls and settings actually do. (UN6)

2.1.3 A deafblind user
This case models a certain type of deafblind user, in this case the user has very severe hearing and vision impairments. With this low level of sight, purely visual elements of the applications become largely superfluous due to the inability of the user to see fine details.

- Applications need to perform the same changes as for a deaf user, while also taking into account the fact the user will likely be interacting with the app via some type of assistive technology like a screen reader. (UN7)
- Purely visual parts of the application, like videos and animations, can often be removed in favor of more space for other components. The user has enough vision to make out large, high-contrast text, which needs a large amount of space to be properly displayed. (UN8)

2.2 Specific needs identified
The user needs listed above need to be translated into corresponding specific UI adjustments in order to accurately determine what’s needed to implement them.
Table 1. The relationships between specific adjustments and user needs.

<table>
<thead>
<tr>
<th>ADJUSTMENT (AX)</th>
<th>DERIVED FROM (UNX)</th>
<th>ADJUSTMENT DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>UN1</td>
<td>Enable non-audio notifications</td>
</tr>
<tr>
<td>A2</td>
<td>UN2</td>
<td>Disable all audio output</td>
</tr>
<tr>
<td>A3</td>
<td>UN3, UN5</td>
<td>Enlarge controls (buttons, volume control, etc.)</td>
</tr>
<tr>
<td>A4</td>
<td>UN4</td>
<td>Enlarge text</td>
</tr>
<tr>
<td>A5</td>
<td>UN8</td>
<td>Enable high-contrast text</td>
</tr>
<tr>
<td>A6</td>
<td>UN8</td>
<td>Disable or remove animated content</td>
</tr>
</tbody>
</table>

UN6 and UN7 are not used to derive any specific settings, because achieving those goals is done in a different way.

UN6 is most easily fulfilled by using a translation system like most applications already have. With these systems, simplified language should be available to be chosen as the application language, just like English or Swedish would. A notable example of this method being used is Wikipedia and its Simple English version†, where articles are rewritten to avoid using words and grammar that a novice English speaker might not understand.

UN7 falls under the responsibility of general good practices for developing any application, specifically to make it compatible with various assistive technologies. The W3C Web Content Accessibility Guidelines (WCAG) describe these practices in great detail (W3C, 2008). The case of using a screen reader is covered in section 1.1:

"Text Alternatives: Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language."

- W3C WCAG 2.0

At the end of the quote, they also mention the issue brought up in UN6. Adding text alternatives to various parts of the application is often very simple, since many modern frameworks provide some type of caption or description field for their built-in controls. For example, the Android platform provides a `contentDescription` field‡ that’s used by screen readers and similar assistive technologies.

---

† [http://simple.wikipedia.org/wiki/Main_Page](http://simple.wikipedia.org/wiki/Main_Page)
3 User interface customization

3.1 Default eCtouch UI
The eCtouch app has three primary views that the user interacts with. The first is the main view, from which calls, contacts and call history is managed.

![Figure 1. The main view of eCtouch.](image)

Figure 1 shows what the user first sees when the app starts up, and also the screen that’s shown when the app is idle. On the far right side is the slider for starting and ending calls (drag up to start, down to end). On far left are the controls for disabling sound, microphone and video, the screenshot shows that video is currently disabled. The large camera icon in the top left is where the user would normally see their own image if the camera was enabled. The left center panel can show either the call history (pictured, with the Swedish title “Samtalslogg”) or the contacts list. The right panel shows your own address and the address of a selected contact or a manually entered address.

A few potential issues can be identified here. For users with low vision, some of the text might be too small to comfortably read. Most notably, the grey top bar (called the Action Bar in Android terminology) and the call history panel both have important information displayed in a small text size.

For users with some degree of motor function impairment, the buttons along the left side of the screen can be too small and too close together to easily and accurately touch. The same is true to a lesser degree for the buttons in the top row, although the overflow menu button on the far right is similar in size to the left side buttons.
Figure 2 shows a call in progress. The receiver is an echo service used for testing purposes, and it simply echoes everything back to the caller. Since video is disabled by the caller, the echo service emulates the same situation and a placeholder image is displayed instead. The same buttons as in the main screen are present on the left side, with the addition of a text on/off button and a button to bring up the on-screen keyboard. The right side has the same slider as on the main screen, but here it has been dragged upwards to initiate the call. The main panel displaying the video placeholder also shows the real-time text chat. The white text (rows 1 and 3) were typed by the caller and the echo service simultaneously replied with the yellow text (rows 2 and 4). When using a separate keyboard the video call screen can remain in full screen mode while typing. If no such keyboard is available, bringing up the on-screen keyboard enables the split-screen mode shown in Figure 3. In this mode, the video display is resized to fit within the remaining space, and the text chat is displayed next to it rather than on top of it. For both eCtouch and eCmobile, the real-time text chat font size is separate from the font size in the other parts of the applications. The font size can be adjusted during a call by performing a “pinch” gesture with two fingers, like how you’d zoom in on an image on a smartphone or a tablet.

The problem with the buttons on the left side in Figure 2 and Figure 3 is the same as in Figure 1, they are potentially too small and close together. The chat text in Figure 2 appears quite legible, but users with low vision may still find it hard to read, especially when the text is overlaid on a moving video background. Increasing the size of the text can help, but that may not be
sufficient for some users. In Figure 3, the split-screen mode provides a static single-colored background for the chat text, which improves legibility.

![Figure 3. Split-screen mode for using the on-screen keyboard.](image)

3.2 Default eCmobile UI

Figure 4 shows the first of three screens of the eCmobile user interface. This screen shows the call history. The main issue here is that the text size may be too small for some users. The tab buttons along the top of the screen are already quite large, so touch issues are not a major concern here.
Figure 5 shows the main UI of eCmobile, which features a text field for entering addresses, a large call button and a group of buttons along the bottom of the screen. The row of buttons at the bottom is used to exit the application, enable or disable the microphone, sound and camera, and finally to open the preferences dialog. These buttons are relatively small and are not separated at all, which can lead to some problems for users with impaired motor control. Lastly, Figure 6 shows the contact list. It’s very similar in appearance to the call history screen, so the same problems can be found here.
When a call is in progress, there are three separate views that the user can choose from. The first, shown in Figure 7, hides the real-time text chat. The second (Figure 8) displays the text on a transparent background overlaid on the video. The third (Figure 9) displays the text on a solid single-color background, for increased legibility. In the first two views, the user can see their own image in the lower right part of the screen. In these screenshots, the camera was aimed at the ceiling. In all three views, the bottom of the screen has a row of buttons similar to those found on the main screen. They allow you to enable or disable the microphone, sound and camera. The first and the last buttons are used to end the current call and to bring up the on-screen keyboard. In these views, the main problem is the bottom row of buttons. They are small and tightly packed, which can lead to unfortunate unintended clicks. A user with impaired motor control could easily press the hang-up button while attempting to turn off the microphone, for instance.

While in a call, the user can bring up the on-screen keyboard. This changes the layout as shown below. Figure 10 shows the overlay mode where chat text is drawn on top of the incoming video stream and the self-image is hidden. Figure 11 shows the solid background screen, which hides both the incoming video and the self-image. Both views have the same problem, that the toolbar is displayed above the keyboard. Directly below the toolbar is the area where the keyboard shows its predictions and suggestions for the current or next word, meaning that the user is likely to press there frequently. The close proximity of
the toolbar means that there’s a risk for accidental button presses, particularly among users with reduced motor control.

Figure 10. In-call view, keyboard active in the overlay text mode.  
Figure 11. In-call view, keyboard active in the solid background mode.

3.3 Solving problems with adjustments
The adjustments listed on page 9 can be applied to solve the issues outlined above. There were a few commonly recurring problems that have similar solutions, listed below.

3.3.1 Text size is too small
Solving the problem of small text is a simple task at first glance. The solution was identified earlier as adjustment A4, “Enlarge text”. This operation must be performed with care, since the relative sizes of different text fields are used to show how they relate to each other. Headers in a document are bigger than the body text and if that relative size difference is lost or diminished, some information is lost. As such, adjustments to text size are always performed as a scaling operation. Each text field is resized as a percentage relative to its original size.

There are two other ways of doing these adjustments, the first being to define the change as an absolute change in point size. For instance, one could ask that all text fields have their text sizes increased by 10 points. While this would work relatively well, it does change the relative size difference between different text fields. The last, and worst, way of making changes to overall text size is to set all text sizes to the same absolute size. Setting the text size globally to (for
example) 30 points would destroy any contextual information given by the relative sizes, and it would also negatively impact the aesthetics of the application. There is one positive aspect of this type of operation, however. With the other two methods, the user can’t know what size the text will be afterwards; only that it’ll be bigger or smaller than before an by how much. By setting the size to a single fixed value, the user can be certain that no text will be smaller than the chosen size. This can be a somewhat useful feature for users with low vision, if they know that a certain size is the absolute minimum they can use comfortably.

3.3.2 Icons and buttons are too small
The solution to this problem was earlier identified as adjustment A3. Increasing the size of purely visual elements primarily helps users with low vision. Increasing the size of interactive elements like buttons helps both those with low vision and those with impaired motor control.

The same methods used to change text size can be used to change icon size (or button size, hereby collectively referred to as icon size), and the same considerations apply when comparing them. There is some difference, in the sense that buttons generally have similar sizes across an application. That means that there’s less risk of losing information when changing their sizes than there is with text.

3.3.3 Other issues
Adjustment A1 (“Enable non-audio notifications”) is a feature present in both applications. The applications use vibrations, flashing lights (using the camera flash LED, if available) and flashing screens to signal the user when particular events happen.

Adjustment A2 (“Disable all audio output”) is handled in one of two ways with these applications. All Android devices where these applications run have hardware volume controls, which control the sound volume for the entire device. Secondly, the sound can be disabled in the applications by using the toggle buttons on the main screens.

Adjustment A5 (“Enable high-contrast text”) has been implemented in both applications, and extended to include art assets as well. This ensures a consistent UI appearance regardless of the theme in use.

Adjustment A6 (“Disable or remove animated content”) is only applicable to the video portions of the applications, which can be turned off entirely.
3.4  Limits on customizability
The UI descriptions presented so far have only described potential problems and shown solutions without considering the implications of applying them. These applications run on mobile devices, which means screen real estate is scarce. For eMobile especially, there is very little room to expand into when performing the adjustments necessary to accommodate the users’ needs. This means that great care has to be taken to ensure that the various UI elements don’t overlap, overflow or cause other visual problems.

For this reason, it’s necessary to impose limits on the magnitude of the changes that are possible. For instance, it was found in testing that doubling the text size was the maximum value that still preserved an acceptable UI appearance.

4  Integrating with Cloud4all
The UI customizations presented so far are of little use unless the users have a way to get them how they want them and when they want them. This is where the Cloud4all platform comes into play, since it provides many of the functions needed to streamline process of individual customizations.

4.1  Matching needs and preferences
The Cloud4all architecture uses a component called the matchmaker to translate the generic needs specified by the user into application or platform specific preferences. The individual mappings are done by the transformer module, which is a part of the matchmaker.

4.1.1  The transformer module
The user specifies their needs in the pre-defined common terms, and the developers of each solution will then specify a set of rules for transforming common terms into settings compatible with their particular solution. This transformation can also perform various operations on the common terms to properly match them to solution settings. In the case of a screen magnifier, the magnification can be represented in different ways without changing the meaning of it. Either you specify it as a multiplicative factor, as in “I want 1.5x magnification”, or equivalently as a percentage: “I want 150% magnification”.
Obviously a solution that uses one can’t use the other (if it wasn’t made to do both), so a transformation is needed. The transformer module allows developers to specify (for instance) a scaling transformation that converts between the two formats. The scaling transformation scales the input value by a pre-defined factor (in this case 100 or 0.01) to yield the output value. There are several other transformation operations available to ensure that solutions
can use the settings provided with minimal changes to how they work internally.

As an example, eCtouch and eCmobile support three Common Term preferences at the time of writing. Expressed in plain English, they are font size, high contrast enabled or disabled and the name of the preferred high contrast theme. These preferences have several different names depending on context. The names and their uses are outlined in TABLE FOUR below.

### Table 2. The relationship between the names of preferences

<table>
<thead>
<tr>
<th>ENGLISH NAME</th>
<th>APPLICATION SPECIFIC NAME</th>
<th>COMMON TERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font size</td>
<td>fontsize</td>
<td><a href="http://registry.gpii.org/common/fontSize">http://registry.gpii.org/common/fontSize</a></td>
</tr>
<tr>
<td>High contrast enabled</td>
<td>theme</td>
<td><a href="http://registry.gpii.org/common/highContrastEnabled">http://registry.gpii.org/common/highContrastEnabled</a></td>
</tr>
<tr>
<td>High contrast theme name</td>
<td>theme</td>
<td><a href="http://registry.gpii.org/common/highContrastTheme">http://registry.gpii.org/common/highContrastTheme</a></td>
</tr>
</tbody>
</table>

The application specific preference *theme* appears twice because it captures the meaning of both of the Common Terms it’s associated with. If *theme* is anything but “none”, high contrast is enabled. The value of *theme* could be, for instance, “yellow-black” or “white-black” to specify a theme which primarily has yellow or white text (respectively) on a black or very dark background.

Furthermore, *fontsize* does not have the same value range as the equivalent Common Term. The Common Term preference value range is scaled up by a factor of 2 compared to *fontsize*. Since neither of these preferences have a straight one-to-one mapping from Common Term to application setting, or vice versa, transformations must be written.

The transformations are written as part of the application’s entry in the Solution Registry, in the form of a JSON file. Each installed application has an entry in this file and it specifies a several different things about each application. Among other things, the Solutions Registry entry lists the version of the application, the required operating system version, how to start and stop the application, and some other properties. The transformations are defined in a block called capabilitiesTransformations:
"capabilitiesTransformations": {
  "map\.string\.fontSize\.$t": {
    "transform": {
      "type": "fluid.transforms.linearScale",
      "valuePath": "display.screenEnhancement.fontSize",
      "factor": 2
    }
  },
  "map\.string\.theme\.$t": {
    "transform": {
      "type": "fluid.transforms.condition",
      "conditionPath": "display.screenEnhancement.-provisional-highContrastEnabled",
      "true": "yellow-black",
      "truePath": "display.screenEnhancement.-provisional-highContrastTheme",
      "false": "none"
    }
  }
}

The first transformation says, if paraphrased into English, to take the value of `display.screenEnhancement.fontSize` (provided by the system, from the user’s preferences) and scale it linearly by a factor 2, and then store the result in `map\.string\.fontSize\.$t`. The strange format is due to the fact that it’s an encoded way of specifying a position in an XML tree. This is used by the XML Settings Handler module to format the output XML file in the correct way.

The second transformation is a bit more complicated. First it checks the value of `display.screenEnhancement.-provisional-highContrastEnabled`. If true, output the value of `display.screenEnhancement.-provisional-highContrastTheme` if it exists. If it does not, output “yellow-black”, which is the default high-contrast theme. If the original condition is false, output “none”.

4.1.2 The matchmaker module
The matchmaker module itself has the capability to utilize a user’s specified needs and preferences to suggest solutions that can provide the best fit with those particular needs. In a basic example, two news reader applications are both registered solutions with Cloud4all, but only one of them implements support for high contrast text display. A user whose only requirement is the ability to use high contrast text will only be recommended the applicable solution by the matchmaker. With more complex situations this becomes a much less trivial matching process, and because of that there are several different matchmaking strategies that can be used.

4.1.3 Default preferences
In order to ensure that everyone involved can communicate about needs and preferences efficiently and without ambiguity, the Cloud4all project has compiled a list of Common Terms (see Appendix A), listing the preferences that will be used. This means that a preference like “font size” has a well-defined
meaning, and these common terms are also used as the basis for the default set of user-changeable preferences.

4.1.4 Custom preferences
The Cloud4all common terms registry covers a wide range of preferences, but it would be infeasible to compile a list of preferences that cover every configurable aspect of every registered solution. The common terms are aptly named, in the sense that their purpose is to only include those terms that are commonly used and shared by many solutions. For the purposes of eCmobile and eCtouch, there are some wanted preferences that are not included in the common terms registry. The first of these has to do with relay services, specifically the user’s needs regarding the type and provider. Relay service preferences are specific to solutions implementing total conversation, which make up a very small part of the overall GPII ecosystem. As such, including it in the common terms registry is potentially unwise since it could make the registry bloated and hard to manage.

The Cloud4all system can still handle those preferences, but without the ability to transform them into something usable by other solutions. These are called application specific preferences. They are passed as-is to the solution, since the framework has no knowledge of how to use them. Application specific preferences also include application unique preferences, which are only useful to that specific solution. An example of an application unique preference could be the position or size of a particular UI element, like the free-floating toolbox dialogs often found in image editing programs.

4.2 Accessing Cloud4all preferences
User preferences are stored either in the cloud or locally on the device. The user doesn’t need to know which, since the Preferences Server handles those things. The user initiates a login by any of the supported methods, for instance by scanning a QR code containing her unique user token. Her preferences are retrieved, transformed and (in the case of eCtouch and eCmobile) written to an XML configuration file on the device. The framework then launches the application, which reads the configuration file and adjusts itself accordingly.

4.3 Preference sets for test cases
The test cases defined earlier can now be expressed in terms of which specific needs and preferences they have.

The first test case (the deaf user) does not have any needs that would lead to UI customizations in this prototype version. Non-audio notifications are a standard feature and a deaf user would likely have all sounds turned off on an operating system level regardless of these apps and Cloud4all.
The second test case (the elderly user) needs enlarged text and icons to make it easier to read and to allow easier touch navigation. So one possible preference set for this user could be as shown in Table 3.

Table 3. Example preference set for an elderly user.

<table>
<thead>
<tr>
<th>PREFERENCE</th>
<th>DEFAULT VALUE</th>
<th>USER’S VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>font size</td>
<td>24</td>
<td>40</td>
</tr>
<tr>
<td>icon size</td>
<td>70</td>
<td>90</td>
</tr>
</tbody>
</table>

In order to translate this preference set into Cloud4all Common Terms, an equivalent preference must be found among the Common Terms. If the Common Term preference does not share the same value range as the preference set the application exists, the aforementioned transformations must be used to construct a mapping from one to the other. Furthermore, since there is no Common Term preference for icon size, button size or anything of that sort, there is (at the time of writing) no way to use that preference via the Cloud4all framework. Future versions of the framework will have better support for mixing Common Terms with application specific preferences, and it is likely that a preference similar to icon size will be made into a Common Term. With that in mind, this is the full preference set expressed in Common Terms:

```
{  
    "http://registry.gpii.org/common/fontSize": [{ "value": 20 }]
}
```

The preference set here is expressed in the JSON format, which is what the Cloud4all framework uses for all inter-module communication. The Common Term for font size is "http://registry.gpii.org/common/fontSize".

Using the transformation rules outlined earlier, a preference set like this is translated into the following XML output:

```xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="fontsize">40</string>
</map>
```

This is the XML code that eCmobile/eCtouch read on startup.

The third test case (the deafblind user) has similar needs, but due to low vision even larger text and icons are preferred. Additionally, a high contrast theme is used to aid in readability. A possible preference set for this user could be as shown in Table 4.
Table 4. Example preference set for a deafblind user.

<table>
<thead>
<tr>
<th>PREFERENCE</th>
<th>DEFAULT VALUE</th>
<th>USER’S VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>font size</td>
<td>24</td>
<td>50</td>
</tr>
<tr>
<td>icon size</td>
<td>70</td>
<td>105</td>
</tr>
<tr>
<td>theme</td>
<td>none</td>
<td>yellow-black</td>
</tr>
</tbody>
</table>

This preference set corresponds to the following set of Common Terms:

```json
{
    "http://registry.gpii.org/common/fontSize": [{ "value": 25 }],
    "http://registry.gpii.org/common/highContrastEnabled": [{ "value": true }],
    "http://registry.gpii.org/common/highContrastTheme": [{ "value": "yellow-black" }]
}
```

Which in turn gets transformed into the following XML:

```xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="fontsize">50</string>
  <string name="theme">yellow-black</string>
</map>
```

## 5 Programmatic UI tweaking

### 5.1 Android UI

When developing Android apps, there are three main ways of designing the UI. The first is to write XML layout files manually, specifying each UI element and its properties. The second is to use the graphical “what you see is what you get” (WYSIWYG) UI editor that’s available as an Eclipse plugin. This tool allows the developer to drag-and-drop UI elements to build their desired UI, and the tool then automatically generates the corresponding XML layout files. For many tasks, a combination of these two methods is quite practical. You insert all the necessary UI elements roughly how you want them and then manually tweak the auto-generated XML files. The third method is to make the UI with Java code inside the app’s startup methods. You then have to manually create each UI element and layout containers and combine them into a finished UI, an often arduous process considering you have to re-compile and run the app in order to see the effects of the changes you’ve made. In contrast, the first two methods let the developer see the effects directly in the UI editor without compiling. The third method allows the developer to create arbitrary layouts at runtime, a measure of freedom that is not possible using the other methods.
5.2 Performing tweaks
Many aspects of the UI can be altered at runtime, but certain elements are not intended to have that capability. For instance, the so called ActionBar visible along the top edge of the screen in Figure 1 has its appearance defined by a theme and its associated styles. These are not intended to be changed while the application runs, which places some restrictions on how our UI customizations can be performed. For this reason, preferences are only read and applied on startup.

5.3 Applying profiles
To illustrate what happens when a user logs in, here is an overview of the events that take place.

1. User logs in (via NFC, QR code, web login, etc). The relevant User Listener reacts.
2. The login event is passed through the Cloud4all architecture (see Architecture Overview for more details) causing an XML configuration file to be written and eCtouch/eCmobile is launched.
3. The application reads the configuration file immediately on startup, before the UI is constructed.
4. The application checks if the ActionBar has to be changed and applies the correct theme. This can only be done before the UI has been created.
5. The application creates the UI like it normally would, by inflating it from the XML layout files.
6. The application applies any other changes specified in the configuration file. This is the point where text size and icon size are changed. These can only be performed after the UI has been created.
7. The application is now ready to use.

6 Method
By and large, the work done followed the agile principles loosely defined by Beck et al. (2001) in the Agile Manifesto. Abrahamsson et al. (2002) proposed a definition of agile methods that gives them four primary attributes. According to them, an agile method is incremental, cooperative, straightforward and adaptive. This means that releases are frequent, but with a high degree of flexibility to quickly react to changes. Additionally, close contact with the client is maintained in order to quickly get feedback on the releases made. For this project, it doesn’t make sense to think in terms of releases and clients. Rather, new test builds can be thought of as releases and the client is Omnitor.
The rapid iterative methodology used was chosen for several reasons. It allows for more flexibility in how you define the work and desired results. This leads to a result that is both easier to attain (since you can change or re-define a feature that turned out to be infeasible in the allotted time) and that has the potential to offer the client a better product than was initially expected. During the course of development, an improvement on a requested feature (from either the developer or the client) can be discovered. The flexibility of the agile method allows this improvement to be quickly evaluated and tested. Another reason for choosing this method was the fact that it’s well suited to a single developer working on his own, which would be the case for the development stage of this project. The reason for this is that the basic ideas of the agile methods do not include any tenets specifically aimed at developers working in teams. This may seem contradictory with the inclusion of the cooperation attribute in the previous paragraph, but the cooperation it refers to is the close contact between client and developer, not necessarily between developers themselves.

Throughout the project, iterative cycles of work were used to design, test and re-design the various parts of the applications. The work done can be roughly divided into 3 separate phases, although there is significant overlap between the different phases.

6.1 Design phase
The design phase took place after the initial research work had been completed. Working together with Omnitor’s staff, mockups were constructed to show the various changes that needed to be made to both eCtouch and eCmobile. The knowledge gained from the research phase was used to guide the process towards higher usability and accessibility, while Omnitor provided the experience needed to ensure that the final designs were in line with what current and future customers want.

The design process loosely followed the first four steps in the five-step process described as FDD, or Feature-Driven Development (Palmer & Felsing, 2002). The process is normally applied for teams of developers, but adapting it for a single developer is trivial. The first step in the process is to define an overall model of the desired product. Given the relatively small scope of this project, this step and the second step merged somewhat. The second step is to build a feature list, detailing which individual changes or additions are to be made. Examples of the type of changes or additions that were decided for eCtouch are:

- Remove the left sidebar to gain horizontal space
- Increase default text size on the ActionBar
• Add the ability to change application text size
• Add the ability to read preferences from a file
• Move version number into the menu to remove a bit of visual clutter

The third step in the FDD process is to plan by feature, produce a plan for how development of said features will take place. Normally this is where different features would be assigned to different developers, but that was obviously not done. However, this step is also where features are given different priority levels. Some features are inevitably less important than others, so this is a very important step. In the example list above, the last feature is close to insignificant compared to the other four and was de-prioritized as a result.

The final two steps of the FDD process are to design by feature and build by feature, however these two are closely intertwined in a combined, iterative development process. This leads to some overlap between the design phase and the development phase.

6.2 Development phase
The development work was done iteratively with a fast cycle of build-evaluate-fix, essentially following the final two steps of FDD. For each new feature implemented where there was room for modifications or improvements, the design team from the previous steps were consulted to evaluate the outcome and to discuss possible alterations. This sometimes led to several cycles where new improvements were discovered and implemented, ultimately leading to a better final product.

6.3 Evaluation phase
The development phase was considered completed when the applications had reached a high level of functionality together with the required new features. Since each iteration in the development cycle also involved some degree of testing, there is some overlap between the development and evaluation phases. After the development work was concluded, the applications were tested thoroughly to ensure that no functionality had been lost or changed from the original applications. While there was a substantial amount of testing done during the various iterations of development, those tests were specifically aimed at testing individual features. If one of the new additions had caused problems elsewhere in the applications, those tests were not guaranteed to discover said problems.

The methodology used for testing followed a two-step process. After each new feature was implemented, functionality was evaluated and compared to the expected functionality. For instance, after moving the sidebar buttons in eCtouch (described in section 7.1.1), certain success criteria were required to
be fulfilled. They had to be functionally equivalent to the old versions, in addition to working with the animations for showing and hiding the menu. These tests were fairly basic and only aimed to find the easily discovered issues. The most commonly used method of performing the tests was a side by side comparison with a previous version that did not have the changes being tested. This allowed for very quick discovery of unintended changes in functionality. This first step of testing meant that time could be saved by avoiding the overhead of involving another tester, while also reducing the number of development iterations for any given feature.

The second step was to involve a tester from Omnitor’s staff. Depending on the complexity of the implemented features, this could be done to review a single change or a handful of smaller changes that could be reviewed at once. Since the first round of testing had hopefully caught most of the smaller issues, the second round of tests could be primarily focused on the bigger picture. As such, the primary goal of these tests was to ensure that the new changes hadn’t impacted the overall functionality of the applications. Any deviations from the expected results were relayed back for another round of development.

The evaluation phase was concluded by resolving the issues discovered, followed by a final round of testing.

6.4 Agile over other methods
The agile method offers many benefits for this type of project, compared to more traditional methods. For instance, the waterfall model of software development is what students were encouraged to use in the Software Development course at the Royal Institute of Technology. The basic principles of this method were described by Royce (1970), although it wasn’t given its name until later. The waterfall method describes an entirely sequential workflow, where each step has to be completed before the next begins. The idea is to put a large amount of time and effort into making a thorough and detailed specification in the beginning of the project, and then following this specification strictly until development is finished. Any changes that are requested or required after the specification has been finalized require formal change requests to be submitted and approved. This makes spontaneous changes rare, and encourages the creation of a very good set of initial requirements and expectation.

For this type of project, however, the waterfall method would not have been a good choice. During the planning and design stages, there was a degree of uncertainty in what could be done technology-wise and time-wise. A developer
unfamiliar with the code base he’ll be working with cannot make accurate estimations of time requirements, nor can a designer unfamiliar with programming estimate the difficulty of implementing various requested features. For these reasons, any specification made early in the project would be unlikely to be followed throughout the course of the project, necessitating frequent change requests and further bureaucracy.

Assuming, however, that a suitable specification had been made and successfully adhered to during development, the end result would not have been of the same standard as the actual results were unless additional effort was invested. The reason is that several good ideas were only hatched after working on some related issue during development. Using the waterfall model, each of these would have required a change request to be submitted and approved, using time and effort that could have been better spent elsewhere. One could choose to disregard the need for such bookkeeping, but stepping away from the basic ideas of the model means you could hardly claim to follow it.

7 Results
The work done has yielded two fully functional prototypes that will undergo user testing and review in January 2014. The testing is performed by a part of the Cloud4all consortium as a part of the end-of-year reviews for year 2. After several rounds of internal bug hunting and usability testing, the applications are stable and reasonably polished.

7.1 Modified default UI
As part of the effort to improve the usability of the applications, both applications received minor facelifts. The changes made were decided in collaboration with the team at Omnitor and there were several goals that were strived for. Primarily, the aim was to achieve a consistent and easy-to-use interface in both applications. As illustrated previously, eCtouch had a much different UI style than eCmobile. Since these applications are essentially meant to be the same app in different form factors, UI consistency is preferred.

7.1.1 eCtouch
There were some significant changes made to the eCtouch UI (see Figure 12). First off, the grey sidebar has been removed to free up horizontal space. The old sidebar had two functions, to hold the self-image and to hold the toolbar buttons (for camera, microphone and mute). Both of these were moved to the bottom right corner, the toolbar buttons moved into a separate menu.
Figure 12. The new eCtouch main screen UI.

The ActionBar was increased in size and contrast (from grey to white background) in order to make it easier to read and to accommodate larger text sizes. The menu items in the ActionBar have been re-arranged and moved to different places in order to simplify it.

Figure 13. The new eCtouch toolbar menu.

The gear icon in the bottom right corner of the application opens a new menu (Figure 13), where you can access the main settings dialog (left-most button) and the normal toolbar buttons.
During a call, the UI is similar to the main screen in many ways. The self-image is in the same position (bottom right, partially obscured by the menu), and the menu works in the same way. There are some additional buttons in the in-call menu, from right to left:

- Change real-time text appearance (font and color).
- Alert (where supported, causes the other party’s device to flash, vibrate, or otherwise attract their attention).
- Show on-screen keyboard (eCtouch is normally used with a hardware keyboard, so this button is rarely used).
- Show/hide real-time text chat (e.g. for privacy).

7.1.2 eCmobile
The new eCmobile UI has a few notable changes. The backgrounds have been changed to match those used in eCtouch, the text on the tab buttons along the top has been enlarged as much as possible, and the bottom toolbar has been modified. Figure 15 and Figure 17 show the new background in both the call history screen and the contacts screen. Figure 16 shows the main screen, where the toolbar has been replaced by a menu bar similar to the one shown in eCtouch. The figure shows the menu in its opened state, achieved by pressing the gear icon in the bottom left.
There have also been some changes made to the in-call UI. The self-image has been made smaller and moved up. It now overlaps the incoming video in the bottom right corner, picture-in-picture. This means that the real-time text chat will never overlap the video streams, which eliminates a potential source of legibility problems. Second, the toolbar at the bottom of the screen has been replaced. The new toolbar has three buttons, from left to right: menu, show keyboard, and hang up. The new menu is shown in Figure 19, and the buttons are the same as in eCtouch.
Finally, there have been two changes made in the in-call UI with the keyboard active. The first is that the toolbar is no longer visible when the keyboard is active, which means you can no longer accidentally hang up or turn off audio or video while typing. The second is that in the transparent background mode, the self-image is now visible, as shown in Figure 21. This means that a sign language user can keep the keyboard up and not have to worry about signing out of frame.
7.2 Resulting UI for the test cases

Below, the customized UIs for the test cases are shown, for both eCtouch and eCmobile. For brevity’s sake, only selected screenshots are included. All UI customizations shown here are performed automatically, the only user input required is to initiate the login procedure by, for instance, scanning an NFC tag.

7.2.1 Test case 1

As determined earlier, test case 1 does not have any preferences that lead to a visual change in the appearances of the applications. As such, no screenshots need to be shown here.

7.2.2 Test case 2

This user’s preference set was to have the font size set to 40, instead of the default value of 24.
7.2.2.1 eCtouch

Figure 23. eCtouch main screen with font size 40 (default 24).

7.2.2.2 eCmobile

Figure 24. eCmobile call history screen with font size 40.

Figure 25. eCmobile contacts screen with font size 40.

7.2.3 Test case 3

This user’s preference set was to have maximum font size (50) and a high contrast theme (yellow on black background).
7.2.3.1 eCtouch

Figure 26. eCtouch main screen with maximum font size and high-contrast theme. The toolbar menu is open in the bottom right, showing suitably colored buttons.
Figure 27. eCtouch in-call screen with yellow icons and buttons visible, consistent with the selected high-contrast theme.

7.2.3.2 eCmobile

Figure 28. eCmobile main screen, test case 3 preferences. The menu is open at the bottom of the screen. Figure 29. eCmobile contacts screen, showing font size and theme adjustments. Figure 30. eCmobile in-call view, with the menu open.

7.3 Full customizability demonstration

Since the Cloud4all framework does not yet support icon size as a Common Term, it was not a part of any of the test cases. However, since it will most likely become a Common Term during the third year of the project, support for
it has been implemented and is fully functional. Below is an example of how both applications look when they use the largest sizes for both icon and font size, together with the yellow-black theme. In other words, this is what “maximum customization” looks like.

7.3.1 eCtouch
In both Figure 31 and Figure 32 below, make note of the enlarged icons and buttons, particularly in the right-side sidebar.

![eCtouch main screen with fully customized UI, including icon size.](image)
**7.3.2 eCmobile**

Of particular note in the figures below is Figure 35, which illustrates a common issue when trying to allow arbitrary size changes of UI elements. The bottom menu is open, but due to the small amount of horizontal space available, the buttons in the menu have been constrained in size in order to not clip outside the screen.
8 Discussion and conclusion
Overall, work on this project has progressed smoothly throughout the assigned time period. There have been no major setbacks and both the project deadline and the internal Cloud4all deadlines have been followed.

8.1 Problems faced
One of the things that caused difficulties throughout the development phase was the lack of documentation for certain parts of the Cloud4all framework. Because it’s still under active development, the documentation that does exist is often outdated or otherwise inaccurate. This led to many frustrating days of failed attempts. Luckily, communication within the project was very open and smooth. Many problems and issues were resolved after simply e-mailing the right person, but that should ideally be a last resort.

A related issue is the relative immaturity of the Cloud4all framework. At the time of writing, there are several issues that impact the usability and utility of it. There are currently very few Common Terms that apply to more than a handful applications, while most of them are specific to things like screen readers and magnifier applications. This means that developers must rely on application-specific preferences, which can’t be used in conjunction with Common Terms yet. In addition, by using application-specific preferences you lose all the benefits of the transformation system.

The most frustrating technical issue was the Android platform’s way of dealing with the ActionBar, more specifically how and when its appearance can be changed. This single limitation meant that the idea of complete on-the-fly UI customizations had to be let go. Allowing all the other parts of the application to be changed on the fly is an option, but it was decided that the inconsistent appearance was not acceptable. For instance, a user might select a dark theme for night-time use, but the ActionBar would remain bright white until the application was restarted.

When adding this level of customizability to an application that wasn’t originally designed to support it, there will be many small visual problems that require fixing. Tweaking the layout of icons, buttons and text is required to ensure that there’s no overlapping or other visual flaws, for every combination of text and icon size.

8.2 Restrictions of scope
In the original description of this project, submitted to and approved by the school before work started, there was a specification of the expected scope of
the project. During the course of the project, certain parts were cut out for various reasons. These decisions were made together with Omnitor.

8.2.1 Contextual adaptations
The original specification called for an evaluation of the possibility to perform contextual adaptations of the UI. Context could, for instance, be the time of day or level of ambient light. The purpose of using context for further UI customization is to provide an additional level of usability without the user needing to initiate it. As an example, most Android devices already have one type of contextual UI customization built-in by default. Most devices have a light sensor somewhere on the front of the device. This sensor is used to automatically change the brightness of the screen, increasing brightness in relation to the level of ambient light.

Context functionality is a planned part of the Cloud4all framework, in a module called the Environment Reporter. This module has not yet implemented in the framework, and duplicating the functionality in eCmobile/eCtouch was deemed unnecessary.

8.2.2 Peripherals
The specification also called for a review of relevant peripherals, such as headsets, keyboards and alerting systems. This was dropped for two reasons, the first being a relative lack of relevance. The main focus of this project is on UI customization and while peripherals are important to the end users, they have little relevance to the rest of the work done.

The second reason is the fact that device compatibility is an ever-changing issue. At the start of the project, eCtouch was primarily targeted towards the Samsung Galaxy Tab 2 tablet. Midway through the project, the Tab 2 tablet was becoming increasingly harder to acquire due to its age. As such, the focus was shifted to the very recently released Samsung Galaxy Note 10.1 (2014 edition). Many peripherals, such as keyboard docks, are device specific. Conducting an evaluation of such peripherals becomes problematic, since the conclusions will be outdated in very short order.

8.2.3 Small scale user testing
The original specification called for a small user testing session at Omnitor’s offices. The purpose was to evaluate the benefits provided by the automatic customization of the UI. However, as part of the Cloud4all project, there will be extensive user testing on a far larger scale and with much more rigor. The first third party testing scenario was performed in January 2014, at the annual review held before the European Commission. This was only a small demonstration, showcasing customizability in the various participating
applications and platforms. Actual user testing is scheduled to begin towards the end of February 2014, in Germany, Spain and Greece. The results will then be made available in a report due in May 2014. While it would be desirable to include their findings in this report, it would take too long to wait for their report. In light of these tests, it was decided that it would be a needless duplication of effort to perform local small-scale tests. Furthermore, designing a proper user testing scenario is not a trivial matter, and would have required a rather sizable amount of time.

9 Future work
One of the biggest issues to resolve in future versions is the lack of on-the-fly changes to the UI. There are no known obstacles to implementing it in eCmobile, as it does not use the troublesome ActionBar that eCtouch does. For eCtouch, there might be some type of workaround to change the ActionBar at runtime. There are different avenues to explore in pursuit of this goal, but due to time constraints they were not fully investigated after it became clear that such functionality is not intended.

Having on-the-fly changes would make the customizability much more striking and easy to see, since you can see the appearance change instantly. While the end result is identical whether or not changes happen on the fly, it is a feature that would help sell the apps and the platform. For eCtouch and eCmobile, the expected use-cases do usually not involve frequent user switching for a single device. The reason for this being that devices are sold pre-configured for a specific user, usually via the employer, Landstinget (the County Council) or Arbetsförmedlingen (the Swedish Public Employment Service). Since the devices are purchased and configured for one specific person, device sharing is rare. This means that end-users are unlikely to ever see the interface change on the fly, meaning that benefit gained would not be worth the development effort required.

Another worthwhile avenue for future improvements is the upcoming features being developed for the Cloud4all framework. As support for more common terms is added, the user experience can be improved even further with more fine-grained control over the interface. Future framework modules are also promising, such as the Environment Reporter. When it has been implemented in the framework and integrated with the applications, users can choose to customize their devices based on a variety of environmental factors. For example, eCtouch and eCmobile could be configured to go into silent mode in the evenings, but only if the user is currently at home.
10 Bibliography


### 11 Appendix A: Cloud4all Common Terms

Copied from an internal Google Docs spreadsheet\(^3\) which requires you to be granted access by the administrators.

<table>
<thead>
<tr>
<th>PREFERENCE TERMS</th>
<th>PILOTS 1 NAME</th>
<th>VALUE RANGE</th>
<th>DEFAULT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>screen reader settings</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speechrate</td>
<td>display.screenReader .speechRate</td>
<td>1 .. * (words per minute)</td>
<td>150</td>
</tr>
<tr>
<td>trackingtts</td>
<td>display.screenEnhancement .trackingTTS</td>
<td>array of strings with one or more of: mouse, caret, focus</td>
<td>focus</td>
</tr>
<tr>
<td>speaktutorial messages</td>
<td>display.screenReader .-provisional-speakTutorialMessages</td>
<td>true / false</td>
<td>FALSE</td>
</tr>
<tr>
<td>keyecho</td>
<td>display.screenReader .-provisional-keyEcho</td>
<td>true / false</td>
<td>TRUE</td>
</tr>
<tr>
<td>wordecho</td>
<td>display.screenReader .-provisional-wordEcho</td>
<td>true / false</td>
<td>FALSE</td>
</tr>
<tr>
<td>announcecapitals</td>
<td>display.screenReader .-provisional-announceCapitals</td>
<td>true / false</td>
<td>FALSE</td>
</tr>
<tr>
<td>screenreader brailleoutput</td>
<td>display.screenReader .-provisional-screenReaderBrailleOutput</td>
<td>true / false</td>
<td>FALSE</td>
</tr>
<tr>
<td>punctuationverbosity</td>
<td>display.screenReader .-provisional-punctuationVerbosity</td>
<td>none, some, most, all</td>
<td>some</td>
</tr>
<tr>
<td>readingunit</td>
<td>display.textReadingHighlight .readingUnit</td>
<td>word, line, sentence, paragraph</td>
<td>sentence</td>
</tr>
<tr>
<td>auditoryoutlanguage</td>
<td>display.screenReader .-provisional-auditoryOutLanguage</td>
<td>IETF BCP 47 language tags</td>
<td>(should depend on the user's locale)</td>
</tr>
<tr>
<td>screenreaderttsenabled</td>
<td>display.screenReader .-provisional-screenReaderTTSEnabled</td>
<td>true / false</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

---

\(^3\)https://docs.google.com/spreadsheet/ccc?key=0AppduB_JZh5EdGltZnF3dVpKdXcxSVhEZ0VjZGY1U3c&usp=drive_web#gid=0
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Display</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Text Reading</strong></td>
<td>Display the text with high contrast colors.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Highlight text with different colors.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pitch (font size)</td>
<td>0.0 .. 1.0</td>
</tr>
<tr>
<td><strong>Volume</strong></td>
<td>Display the text with TTS.</td>
<td>0.0 .. 1.0</td>
</tr>
<tr>
<td><strong>Self-voicing</strong></td>
<td>Enable or disable self-voicing.</td>
<td>true/false</td>
</tr>
<tr>
<td><strong>Volume TTS</strong></td>
<td>Enable or disable TTS volume.</td>
<td>true/false</td>
</tr>
<tr>
<td><strong>Screen enhancement</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>High contrast</strong></td>
<td>Enable or disable high contrast.</td>
<td>true/false</td>
</tr>
<tr>
<td><strong>Theme</strong></td>
<td>Enable or disable high contrast theme.</td>
<td>black-white</td>
</tr>
<tr>
<td><strong>Font size</strong></td>
<td>Display the screen with a font size.</td>
<td>0.1 .. * (in points)</td>
</tr>
<tr>
<td><strong>Cursor size</strong></td>
<td>Display the screen with a cursor size.</td>
<td>0.0 .. 1.0</td>
</tr>
<tr>
<td><strong>Magnifier</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Magnifier enabled</strong></td>
<td>Enable or disable magnifier.</td>
<td>true/false</td>
</tr>
<tr>
<td><strong>Magnification</strong></td>
<td>Display the screen with a magnification.</td>
<td>1.0 .. *</td>
</tr>
<tr>
<td><strong>Tracking</strong></td>
<td>Display the screen with tracking.</td>
<td>array of strings with one or more of:</td>
</tr>
<tr>
<td></td>
<td>Mouse, caret, focus</td>
<td>mouse</td>
</tr>
<tr>
<td><strong>Magnifier position</strong></td>
<td>Enable or disable magnifier position.</td>
<td>FullScreen, Lens, LeftHalf, RightHalf, TopHalf, BottomHalf, Custom</td>
</tr>
<tr>
<td><strong>Invert colours</strong></td>
<td>Display the screen with inverted colours.</td>
<td>true/false</td>
</tr>
<tr>
<td><strong>Show crosshairs</strong></td>
<td>Display the screen with crosshairs.</td>
<td>true/false</td>
</tr>
<tr>
<td><strong>General</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Volume</strong></td>
<td>Enable or disable volume.</td>
<td>0.0 .. 1.0</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>Display the screen with the language.</td>
<td>IETF BCP 47 language tags</td>
</tr>
</tbody>
</table>

(should depend on the user's)
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch screen settings</td>
<td>(mobile settings)</td>
<td></td>
</tr>
<tr>
<td>Haptic feedback</td>
<td>control.-provisional-hapticFeedback</td>
<td>none, always</td>
</tr>
<tr>
<td>Screen off time</td>
<td>display.-provisional-screenOffTime</td>
<td>0 .. * (seconds)</td>
</tr>
<tr>
<td>Screen dim</td>
<td>display.-provisional-screenDim</td>
<td>true / false</td>
</tr>
<tr>
<td>Screen rotation</td>
<td>display.-provisional-screenRotation</td>
<td>true / false</td>
</tr>
<tr>
<td>Screen default rotation</td>
<td>display.-provisional-screenDefaultRotation</td>
<td>0(0°), 1(90°), 2(180°), 3(270°)</td>
</tr>
</tbody>
</table>