UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

MOBILE SYSTEM-WIDE ASSISTIVE TECHNOLOGY

André Filipe Pereira Rodrigues

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2014

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

MOBILE SYSTEM-WIDE ASSISTIVE TECHNOLOGY

André Filipe Pereira Rodrigues

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

Trabalho orientado pelo Prof. Doutor Tiago Guerreiro

2014

Acknowledgements

In one year my life turned upside down. From the student that relies on his parents for tuition and support, to the researcher, part-time teacher, student and independent individual. I would like to thank first are foremost my thesis adviser Doctor Professor Tiago Guerreiro for taking me under his wing and guiding me through this journey that is the master degree. My family for supporting me throughout the years, by providing stability and the opportunity to pursue my desires. They were always a motivational force in my life and will always be. I own them my drive to accomplish any goals I set out to. To Ana my girlfriend that accompanied me throughout my academic journey for being there whenever I needed. I thank her for making me growth for giving me the occasional motivational “slap” that I needed and for dragging me out of the house ever so often. To my colleagues that through the year check up on me and help me with my academic pursuits. Five years ago I made a choice to enrol in FCUL and I do not regret a moment of it. A thank you to the institution for being the perfect place for my first five years of my academic course. Thank you to the Lasige research group that received me with open arms and provided me with the tools and opportunities to succeed. To the friends that went out of their way to check up on me and help me clear up my head and enjoy myself over the year. Last but not least, to the Fundação Raquel and Martin Sain, in particular to Dr. Carlos Bastardo, Miguel and his parents for without them this work would not be possible.

To my parents.

Resumo

Hoje em dia o uso do telemóvel é fundamental em diversos aspetos da nossa vida. Tornou-se uma ferramenta essencial para a comunicação, convívio, consulta de informação e até mesmo para o entretenimento. Os telemóveis tornaram-se mais do que simples ferramentas para fazer chamadas e receber mensagens. Eles são os nossos assistentes pessoais, são a forma como nos mantemos em contacto com os nossos amigos e colegas, e muito mais.

Infelizmente, os sistemas operativos destes dispositivos ainda apresentam muitas debilidades em termos de acessibilidade. No entanto, as suas capacidades representam uma enorme oportunidade na criação de tecnologias assistivas. Os sistemas operativos móveis não são suficientemente adaptáveis e configuráveis para pessoas com múltiplas deficiências poderem usufruir deles. Na verdade, nas últimas versões dos sistemas operativos móveis mais populares (iOS e Android) tem sido feito um esforço para incluir mais opções de acessibilidade. No, entanto ainda não são o suficiente para suportar todo o tipo de necessidades.

Uma das abordagens possíveis para permitir o uso das aplicações por pessoas com deficiências passa pela criação de aplicações acessíveis. No entanto, esta abordagem não é escalável e enfrenta graves problemas. Ao fazermos apenas a aplicação acessível estamos a limitar o utilizador a apenas essa aplicação, em vez de estarmos a criar as ferramentas que permitem o controlo e acesso total do sistema á semelhança de um utilizador sem deficiência. Estas aplicações têm usualmente preços elevados, justificados pelo longo desenvolvimento em tecnologias que não serão usadas pelas massas. O problema das aplicações acessíveis é a especificidade da aplicação. Quando tornamos uma aplicação acessível a utilizadores cegos não estamos a torná-la acessível a utilizadores com deficiências motoras. O problema cresce quando começamos a ter em consideração utilizadores com múltiplas deficiências. A nível aplicacional é extremamente dispendioso criar uma aplicação acessível a utilizadores com múltiplas

deficiências. A preocupação é: ao tornamos uma aplicação acessível vamos torná-la acessível a que conjunto de deficiências ou pessoas? Devido à enorme variedade de requisitos de acessibilidade, é necessário todo um sistema configurável e adaptável.

Existem muitos estudos na área das tecnologias assistivas, a maioria delas tem como foco a acessibilidade do Desktop. É fundamental para a integração de pessoas com algum tipo de deficiência o acesso a um computador. Hoje em dia é necessário mais do que isso, é também necessário permitir o acesso ao telemóvel. Os *smartphones* têm imensas possibilidades de uso como tecnologias assistivas. Podem permitir desde o acesso á internet, ao controlo de uma cadeira de rodas, ou até ser um controlo remoto do ambiente em volta do individuo (televisão, ar condicionado, eletrodomésticos, etc).

Atualmente, se desenvolvermos periféricos com o objetivo de controlar um *smartphone* vamos deparar-nos com uma enorme quantidade de restrições. Para desenvolver tecnologias assistivas realmente usáveis e abrangentes, é necessário ter acesso a dois níveis de controlo, nível aplicacional e nível de sistema. No entanto, nos dispositivos móveis não temos acesso ao nível de sistema, apenas os produtores do Sistema Operativo (SO) o têm. É necessário criar uma camada intermédia que consiga de alguma forma ter acesso a informações de sistema de forma a possibilitar um controlo preciso e fino sobre todos os eventos de entrada e saída de dados.

Esta dissertação encontrou motivação no caso do Miguel. Miguel é um individuo com múltiplas deficiências que enfrenta severas dificuldades para interagir com o seu telemóvel. O trabalho desenvolvido foi em grande parte motivado pelas dificuldades diárias que ele enfrenta em manter-se socialmente ativo e com o maior nível de independência possível. Com as tecnologias atuais, o Miguel está debilitado e não consegue usufruir de todas as capacidades do seu dispositivo. Assim, estabelecemos como objetivos criar uma tecnologia que permitisse: controlo total a nível de sistema dos mecanismos de entrada; desenvolvimento ágil de tecnologias assistivas; suporte a utilização de tecnologias assistivas externas; criação de soluções extensíveis e adaptáveis; controlo sobre o conteúdo e navegação a nível de sistema.

Nesta tese apresentamos SWAT, uma biblioteca extensível e adaptável que permite o acesso ao nível de sistema aos conteúdos e à informação dos eventos de entrada. A biblioteca foi desenvolvida para Android e providencia uma plataforma estável sobre

a qual é possível estender e adaptar as suas capacidades. Com SWAT, os programadores têm a possibilidade de aceder a eventos de baixo nível e a funcionalidades que não teriam de outra forma. Este controlo permite superar as barreiras impostas pelo tradicional sistema operativo móvel e abre as portas à criação de tecnologias assistivas mais adaptáveis e baratas. A Biblioteca possui uma API (Interface de Programação de Aplicações) simples que providencia tudo o que é necessário para um fácil acesso a estas funcionalidades.

Esta tese tem como principais contribuições:

- Biblioteca SWAT: permite a criação de tecnologias assistivas de forma rápida e eficaz, com controlo sobre o conteúdo e eventos de entrada de baixo nível.
- Macros assistivas: uma aplicação que permite gravar e reproduzir macros num sistema operativo móvel;
- Logger: um mecanismo de gravação a nível de sistema;
- Auto-Nav: um protótipo aplicacional que permite varrimento automático das opções disponíveis num telemóvel de forma a permitir o acesso via interruptor;
- Leitor de ecrã multi-toque: um leitor de ecrã que permite feedback auditivo de vários pontos em simultâneo.

Com o objetivo de validar a biblioteca criada, apresentamos dois casos de estudo onde reportamos duas aplicações criadas recorrendo à mesma. A primeira tem como foco o caso do Miguel, um utilizador com múltipla deficiência. À data da introdução desta nova aplicação o Miguel apenas conseguia aceder a uma aplicação desenhada especificamente para ele e nada mais. Neste caso de estudo reportamos a tecnologia desenvolvida para permitir ao Miguel aceder a qualquer aplicação do sistema assim como uma avaliação informal com ele e os seus pais. No segundo caso de estudo validamos a biblioteca pela criação de um leitor de ecrã multi-toque a nível de sistema. Foi feito um estudo com 30 utilizadores cegos numa tarefa de entrada de texto onde resultados foram analisados e reportados. Os resultados obtidos sugerem que pessoas cegas podem beneficiar de interfaces que aproveitem de forma mais eficiente os ecrãs multi-toque e que aproximem o seu uso do já realizado em interfaces físicas (teclado tradicional).

Com este trabalho, demonstramos as limitações impostas pelos sistemas operativos móveis na área de tecnologias assistivas. Desenvolvemos uma Biblioteca para dar resposta a estas restrições providenciando uma base de trabalho para futuras aplicações e tecnologias assistivas. Com os casos de estudo, mostrámos as capacidades do sistema

desenvolvido. SWAT é uma biblioteca prometedora oriunda deste trabalho e começa a ter ramificações nesta e outras áreas de desenvolvimento.

Palavras-chave: Acessibilidade, “System-wide”, Tecnologias, Assistivas, Telemóvel

Abstract

Mobile devices are a fundamental tool in different aspects of our lives. In the last decade, we have witnessed an explosion of the capabilities of our mobile devices. With these improvements, they became more valuable to us than the old house phones. They no longer serve only the purpose of making/receiving calls and text messages. They are our personal assistant, our way to connect to the social media, our multimedia entertainment and much more. Indeed, they are also, and increasingly more so, the prevalent communication artifact at one's disposal. Mobile operating systems have evolved to provide increasing accessibility capabilities. However, mobile application developers are still restricted to deploy custom-made accessible applications or to extend limited and stereotyped accessibility services. In current mobile devices if we develop an external peripheral to control the device we will face several restrictions since there is not a system-wide service that a developer can access to create its own input method. A regular user has the ability to navigate and explore the device without limitations. In order to truly give disabled people the same features as other users have, we need to create a system-wide accessibility service that allows users to freely navigate and interact with their devices. Motivated by the limitations described above we set out to provide more control over the system input and output mechanisms to allow the creation of powerful system wide assistive technologies. We developed SWAT (System-wide Assistive Technologies) for the Android platform. To validate our system we performed one case study with a multi-impaired person and another one with 30 blind users. Results showed that SWAT enabled the creation of application tailored to their needs and thus fostering their inclusion.

Keywords: Mobile Accessibility, Assistive Technologies, Extensible, Framework, I/O.

Content

Figure List

Chapter 1 Introduction

Mobile devices are a fundamental tool in different aspects of our lives. In the last decade, we have witnessed an explosion of the capabilities of our mobile devices. With these improvements, they became more valuable to us than the old house phones. They no longer serve only the purpose of making/receiving calls and text messages. They are our personal assistant, our way to connect to the social media, our multimedia entertainment and much more. Indeed, they are also, and increasingly more so, the prevalent communication artifact at one's disposal.

1.1 Motivation

In the latest years we have witnessed the growth of the market share of smartphones. From the 6 billion mobile subscriptions worldwide, 1 billion are smartphones and it was estimated that in 2013 1 billion smartphones were sold, meaning that one-third of the mobile subscriptions are smartphones[1] (i.e. sold devices normally replace old ones).

Users are greatly empowered when they have full access to such device. Being deprived of such commodity can have a negative impact in ones' social life. This is an extremely important factor especially if we focus on users with some kind of impairment. Conversely, acquiring access to such devices can empower their users in so many ways going beyond the frontiers of the device (e.g., controlling other devices).

Mobile devices are now coupled with a set of input, output and communication capabilities that can potentially allow for the adaptation to their users' capabilities and needs. Unfortunately, the Operating Systems (OS) of these devices suffer from a lack of accessibility options. They do not provide enough customizations to allow different types of disabled people to properly interact with the devices. Mobile operating systems have

[1] http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/e, Last visited in 26/08/2014

evolved to provide increasing accessibility capabilities. However, mobile application developers are still restricted to deploy closed custom-made accessible applications or to extend limited and stereotyped accessibility services. This is especially true when developing for people with multiple impairments [23].

We already faced a similar problem when the personal computer became a mainstream technology. Nowadays personal computers are almost fully accessible given the right set of assistive technologies. The main issue with mobile devices is the lack of support for the creation of system-wide solutions. The personal computer (PC) already went through the process of providing system-wide tools that allow assistive technologies to have full system-wide control over the system. They can, for example, emulate the mouse and keyboard behaviour in order to control all applications on the desktop. Such is still not possible in mobile OS.

Studies about mobile accessibility stress its limitations compared with the desktop. One particular study delves deep into the problem behind it focusing on people with multiple disabilities [22]. In this study the main problem reported is the lack of interoperability and extensibility of the mobile OS. Unlike desktop OS, mobile ones only have system-wide assistive technologies that are created by manufacturers. This means that developers are only given the chance to develop assistive technologies at an application level which lacks the proper permissions to take over the input and output systems. It is argued that mobile OS lack the proper system-wide layer in order to permit people with multiple disabilities to properly parameterize the device to their specific needs.

1.2 Mobile Accessibility

Currently there are two major approaches towards mobile accessibility. Creating a custom-made application or resorting to a system-wide solution. Custom-made applications seek to replace all the look and feel of the device. This is the case of applications like Mobile Accessibility[2] or the one reported by Nicolau et al [22]. These types of solutions have the advantage of being tailored specifically for a target audience, which guarantees to a certain degree an improved user experience to those that fit the

[2] http://www.codefactory.es/en/products.asp?id=433, Last visited in 26/08/2014

aimed stereotype. However, they lack the extensibility and adaptability to cover users with a different set of (dis)abilities. By creating a single or a set of applications specifically designed for a group of users we are consciously neglecting all other users with different abilities. Developing applications to users with different abilities from scratch is a costly and time consuming process. Furthermore, in a time where new applications are available every day, creating custom-made access tools is too restrictive for the user as novel versions of the access tool are needed every time a new functionality is desired.

The first steps in mobile accessibility were taken by feature phones, for example, Nuance Talks[3] is a screen reader with zoom option that allows blind and low-vision impaired individuals to take advantage of the same applications sighted people use. Blind users had now the opportunity to buy a wide variety of phone models and were no longer restricted to the expensive custom-based solutions. With the shift from feature phones to smartphones it is crucial to continue research and development in the accessibility area. Smartphones provide a different set of challenges then its ancestor. While feature phones possess physical keys that usually have navigational keys (e.g. up, down) the same cannot be said about smartphones. The lack of physical cues and navigational keypad are the one of the first challenges of smartphone accessibility that especially blind users face.

In the quest for mobile accessibility system-wide solutions, the advent of Apple's iPhone is a relevant mark. Since the introduction of the system-wide screen reader by Apple (i.e. VoiceOver[4]), all the major mobile operating systems have started taking advantage of the device's potential and have been coupled with successful accessibility features (e.g., Zoom[3], TalkBack[5], Switch Control[6], Narrator[7]). These are all services developed by the OS developers.

Only a couple of accessibility solutions are not, in fact create by them. As an example, TeclaAccess is an input method that is able to provide a different navigation mechanisms (i.e. it allows up/down/left/right navigation scheme paired up with a switch controller). Despite these solutions success and possible parameterizations, they are

3 http://www.nuance.com/for-individuals/mobile-applications/talks-zooms/index.htm , Last visited in 26/08/2014

4 https://www.apple.com/accessibility/ios/ , Last visited in 26/08/2014

5 http://developer.android.com/design/patterns/accessibi lity.html , Last visited in 26/08/2014

6 http://support.apple.com/kb/HT5886 , Last visited in 26/08/2014

7 http://www.microsoft.com/enable/products/windows8/ , Last visited in 26/08/2014

heavily stereotyped making assumptions about their possible users. They tend to focus on one kind of disability which leads them to neglect users with multiple disabilities.

Other developers are restricted in their approaches towards mobile accessibility. They can create custom-made solutions that replace the device core applications and have full control of the system, but by doing so, they limit the device use to those applications. Unlike the iOS (Apple mobile OS) and Window Phone 8, Android developers are able to develop Input Methods which relies on making existing applications accessible by providing different means of navigation and input. Unfortunately they are restricted to simple types of navigation and do not possess on screen content information. Developers can also create their own system-wide accessibility services. In Android 4.0+ systems, one can implement an accessibility service and adapt the way in which the content is presented to the user. The accessibility services provide the much needed content information. As an example, TalkBack allows a blind person to painlessly explore the screen by touching it. Selection is made by double-tapping or split-tapping, or by releasing the finger from the screen in advanced mode.

Accessibility services can access the onscreen view hierarchy, accessibility events and perform system wide actions through the accessibility APIs (e.g. back, click on an icon). They operate in between application and system levels, and by doing so, allow developers to be aware of system-wide context changes (i.e. changes that trigger accessibility events). For example, every touch triggers an accessibility event, all the registered accessibility services will be notified of the event. Through this event we are able to access all the view objects.

Unfortunately accessibility services still present several drawbacks. One major issue is they are still largely unexplored. This leads to a lack of support for the creation of technologies that take full advantage of the services capabilities. Accessibility APIs are complex and are nothing short of a challenge when trying to develop any kind of assistive technology. For example, to access the current on screen content, the accessibility service created will receive an accessibility event. Through this event we are able to get the source node, through this node we are able to get both its parent and its children. By navigating through these nodes we are able to create the full hierarchical view of the screen. This type of information should be easily attained through the APIs and not through a multiple step process.

Accessibility services are difficult to master and are mostly focused on the contents presented: the relationship between content and input control is overlooked. TalkBack provides little control over input. It is mostly a single touch interface with a handful of pre-conceived gestures that trigger special events. One problem is that the set of gestures we are able to detect through an accessibility service are predefined. As such, it restricts the developer options when it comes to input. They are only able to redefine the command associated with the gesture and not the gesture itself. They have no control over touch events, they cannot monitor, block or inject them system-wide. Touch events are only available at an application level for developers which makes it conventionally impossible to create system-wide solutions that require touch adaptation. For example, developers are not able to create multi-touch accessibility services at will. As an example, why should not developers be able to create a multi-touch screen reader? A screen reader that could simultaneously read multiple selections. Or create a system-wide gesture recognizer? As another example, although several adaptation models for touch screens have been devised [3, 5]) these limitations make it impossible for them to be deployed system-wide in a mobile device.

1.3 Mobile System-Wide Assistive Technologies

Miguel is a multi-impaired users that faces severe limitations when interacting with a mobile device; his case is reported in Chapter 4. Motivated by his struggle and the limitations described above we set out to provide more control over the system input and output mechanisms to allow the creation of powerful system wide assistive technologies. We developed SWAT (System-wide Assistive Technologies) for the Android platform due to its open source nature, system wide capabilities and market share. In order to achieve our goal we focused in creating a Library that would accomplish the following objectives:

- Full system wide control of input mechanisms (e.g. touch screen);
- Easy and fast development of assistive technologies tailored to the user's capabilities and needs;
- External assistive technology support;
- Creation of inexpensive and adaptable solutions;
- Full system wide control over content and navigation;

SWAT is a library that aims to be the go to tool for system wide applications with a special focus on assistive technology development. The struggles of developers when faced with a situation such as Miguel, where the only solution was the creation of custom made applications, are no longer. With SWAT, developers are able to access low-level events and functionalities; this access allows developers to create system wide assistive technologies that overcome the limitations described in the previous section. Combining this with a simple to use API greatly improves the development process. SWAT works as an Accessibility Service in order to have control over current onscreen content.

This is the basis for the creation of system-wide solutions. SWAT is coupled with some key modules/features that are possible by using the combining power of accessibility services and low level event information:

- System Wide logging mechanism
    - Logging navigation steps - what was touched (e.g. "Settings/Applications")
    - Logging user interaction - where did the user touched (e.g. "down x:50 y:50")
- Control Interfaces – provides fundamental navigation mechanisms for the creation of assistive technologies (i.e. touch icon, highlight icon, navigate to next)
- External Device Control – provides the tools to navigate the device relying on an external;
- Assistive Macros – recording and automation of user interactions system-wide

These modules will be described in depth the third chapter.

SWAT[8] is an Android Library, this enables developers to develop based on the framework, extending its contributions. We provide a couple of interfaces that ensure full and easy control of the system. By extending these interfaces developers are able to customize and adapt their solutions. At its core it is an extensible library that strives to allow complete control over input and content.

[8] https://github.com/AndreFPRodrigues/Mswat , Last visited in 04/09/2014

1.4 Contributions

Users are greatly empowered when they have access to smartphone. There is a lack of system-wide approaches to mobile accessibility due to some severe OS restrictions. More so the assistive technology field tends to approach accessibility in a disability by disability case with custom solutions to specific users, this leads to an inflation in cost and the neglecting of users with multiple disabilities. Our main contributions with this dissertation are:

- **SWAT Library:** an extensible library for developers that enables the creation of solutions with an extreme focus on assistive technology, by proving a finer control over input and content on a system wide basis. With a simple API that accelerates the development and the cost effectiveness of integrating and creating new assistive technologies;
- **Assistive Macros:** an application rooted in the SWAT library that enables the recording, reproduction and management of macros on a mobile device. Through it we can create shortcuts that can overcome accessibility barriers of some interfaces;
- **Logger**: a system wide logging mechanism that allows third party developers to rely on SWAT for all the logging needs. Its system wide capabilities allows the logging across all application and menus, with the ability to log touches and context;
- **System-Wide Scanning Control Prototype:** using SWAT we developed an auto scanning system that enables a multi impaired user to navigate any and all applications on a smartphone using an external device;
- **Multi-Touch Screen Reader Prototype:** using a spatialized audio library [15] in conjunction with SWAT, we developed a multi-touch screen reader with simultaneous feedback. It is an exploratory system with multiple layout divisions and audio feedback options that was used to perform a multi touch text-entry study on a tablet device;

1.5 Publications

The contributions provided in this dissertation were accepted for publication in one international peer-reviewed conference:

- André Rodrigues, Tiago Guerreiro, "*SWAT: Mobile System-Wide Assistive Technologies*", HCI 2014 - 28th International British Computer Society Human Computer Interaction Conference, SouthPort, UK, September, 2014 (CORE A conference)

1.6 Document Overview

Assistive technology has great potential use when coupled with a mobile device. In this dissertation we focus our attention developing a library that allows the creation of system wide assistive technologies in mobile devices. In the next chapter we report the different kind of control interfaces with a closer look at the ones used in the latter chapters. We review the state of the art of accessibility features of the current top 2 market shareholders in the mobile device space with a detailed analysis on the Android platform for which our system was developed. We also report solution in the adaptable interface and logging interaction fields as groundwork for the system features and discussions that ensue in the later chapters. The amount of information acquired both at the presentation and input level along with the capability to simulate and inject events into the OS pave way for several scenarios where SWAT can be useful for application developers and researchers. In the fourth chapter we will present SWAT followed by two studies made possible only through the use of SWAT. This chapter presents the library developed with a detailed description of its architecture and features, it serves as introduction to developers who wish to work with SWAT and understand its underlying mechanisms. From it we understand the baseline from which the two system developed for the studies spawned. In section 4.4 we present the case of the Multi-Impaired user that motivated this dissertation. We did a formal evaluation of the system developed that motivated us to pursue the approach, refine it and use it in different accessibility fields. Chapter 5 we report the use of SWAT to create an exploratory multi-touch screen reader. The system was used to evaluate blind user performance in a text-entry task. We conclude with Chapter 6 where we look how SWAT is being used in other research fields, we discuss the limitations associated with our approach and outline steps for future research.

Chapter 2 Related Work

Mobile assistive technology is still in its early stages when we compared it with the desktop one. In this chapter we make a brief introduction to the concept, range and consequences of assistive technologies. Control interfaces are a fundamental part of these technologies, as such, we take a closer look at several types namely touch, speech and switch and how they are being used in the mobile context. We cannot mention assistive technology and overlook the work done with adaptable interfaces. They are one approach towards an inclusive design that takes into account user abilities. We then review the state of the art of accessibility features of iOS and the Android platform. Interaction logging is a crucial problem of any HCI research, assistive technologies are no different. The problem grows when we take into account the limitations developers have in logging and simulating these technologies actions. We conclude the chapter with a discussion of the current deficiencies of the mobile OS' and how they can be met.

2.1 Assistive Technologies

Assistive technologies (AT) come is many different sizes and shapes. They can be everything from a walker or a cane to a computer controlling a wheelchair. AT is a concept that includes assistive, adaptive and rehabilitation devices. All these devices have two things in common, they all have the same goal: to promote user independence, self-esteem and over-all quality of life; and they are aimed at people with some sort of impairment, either permanent or temporary. These impairments or deficits can be physical, mental or emotional. There are hundreds of assistive technology devices available, each has its own purpose and target audience.

The increasing availability of mobile technologies provides a more affordable assistive technology solution for many people with disabilities. Nowadays mobile devices are extremely powerful, with them we are able to virtually do everything a personal computer was able to do a few years ago. The ability to operate a computer is crucial, especially to people with disabilities. If they can properly control a computer it means they can do everything a computer allows them to do. Smartphones and tablets are nothing less than a small powerful computer.

When developing an assistive technology that allows disable people to properly operate a smartphone we are in fact giving them a tool that is a *jack of all trades*. They will be able to access the Internet, send/receive calls and text message, access social networks, listen to music, watch a movie, play games, with the appropriate adaptations, even use them as environmental controllers around the house or even controllers for a powered wheelchair. Using a mobile device instead of a PC to provide all this features to a disable individual has obvious benefits in terms of mobility.

2.2 Control interfaces

"*According to (Cook and Hussey, 2002), the human/technology interface is composed by three elements that contribute to the operation of the device: the control interface, the selection set and the selection method."* [10]

The control interface is the hardware used to interact with the device (e.g. keyboard, mouse, switch Fig. 1), the selection set is the list of selectable items while the selection method or interface scheme is how the user selects using the control interface.

Fig. 1 - Switches

There are two types of interfacing schemes: direct selection and indirect selection. Direct selection is the one we normally use to interact with an interface. There is a one to one correspondence between what we input and what it is selected on the screen. Clear examples of this are the mouse and the keyboard. This method is usually faster and more accurate than its counterpart. Direct selection relies on the user to properly operate the control interface in order to be accurate. This poses a problem if an individual has some sort of physical impairment that inhibits his ability to use the control interface to its full potential.

Indirect selection provide us with alternatives for people that cannot properly operate a direct selection interface scheme. It relies on techniques such as scanning and coded access in order to provide the user with all selection alternatives.

Scanning offers a solution to those who are only able to operate a switch. The selectable items are sequentially focused and can be selected by pressing the switch (i.e. in a one switch, auto scanning setup). Scanning is an interaction method that mostly addresses people with severe motor impairments. It can be used across many different context such as personal computers, mobile devices, and environmental control for smart homes and ambient intelligence environments [23].

2.2.1 Touch

With the rise of touchscreens different problems and opportunities emerged. Traditional input methods (e.g. keyboard, mouse, and keypad) require some strength in order to be operated and provide great precision when handled properly. Due to its properties, touchscreens are a lot less physically demanding than keypads. However, touchscreens suffer from a lack of relief, which makes it harder for people to accurately select their targets, particularly disable people [28]. The accessibility of mobile devices depends on the correct use of the touchscreen. The touchscreen provides us with a platform less physically demanding than traditional input methods and opens a door for new systems that may benefit people with motor disabilities that lack both strength and control on their upper limbs [14]. Touchscreens lack the physical cues physical keyboards provide but the lack of a physical input also presents itself as an advantage. If input is not reliant on physical key it means we can adapt it, as an example, we can personalise keyboard layout and use classification models to interpret key presses in order to dynamically adapt to each user's typing patterns [6]. Part of the problem is to understand each user's needs and adapt accordingly. In a study by Guerreiro et al [16] it is showed "*that tetraplegic users are likely to benefit from a better understanding of their abilities and challenges*". Allowing the customization of input would not only improve accessibility of simple tasks like navigation but would also have impact on complex ones like playing a game [17]. As an input method, touchscreens should be customizable and personalisable. We should be able to adapt and configure its properties across the system. Currently that is only possible at an application level.

2.2.2 Speech

There are two major approaches to speech interfaces, one is to have a speech recognizer that allow the user to directly select by saying the proper command to that particular selection. The second it tries to mimic the mouse and/or keyboard [5, 28, 19]. An effective speech-based interaction must provide both support for text entry and cursor control.

Although a speech-based solution is a valid input mechanism, it is still in many ways inferior to the traditional keyboard/mouse or to touch input. It is slower than keyboard input [19] and it is not adequate for navigation-oriented activities if other control interfaces can be used [25, 26].

Speech provides nowadays a powerful solution for feedback. Text-to-Speech (TTS) systems are a fundamental part of accessibility. Most smartphones now provide a TTS system that allows blind users and users that are temporally impaired from looking at the screen to interact with the device.

2.2.3 Switches and keyboard

When choosing a control interface for people with disabilities, therapists will always choose the simpler, more conventional solutions with minimal modifications. Using a conventional keyboard is always the first priority [29]. The problem is most people with disabilities cannot access a conventional keyboard, being it a physical one or a soft keyboard (in smartphones), due to lack of coordination or motion range of upper extremities. To use the keyboard some users require special devices that allow them to press only one key at a time, switches. Switches are usually pressure activated devices. Due to the wide variety of user disabilities, switches come in many different ways, they range from simple button switches to head, foot or even breath controlled ones (e.g. Physiological). In section 2.4 we will describe mobile technologies that take advantage of switch interfaces (e.g. HouseMate, Switch Control and TeclaAccess).

2.3 Adaptable Interfaces

Interfaces can be difficult to navigate, especially for users with some kind of disability. Designing interfaces that are usable and accessible to everyone is a tough

challenge. One approach to improve accessibility is to change the underling interface adapting it to each user.

The variety of individual capabilities among users is massive, especially when we considering people with multiple disabilities. Due to such variety it is not practical or scalable to manually design an interface for each [27, 12, and 9]. As such there are systems that tackle the issue by automatically generating tailored interfaces or adapting them. In one particular system, the researchers studied if it was better to generate interfaces based on the users' preferences or by doing a test to assess their abilities. The users were 26.4% faster and did less 73% errors on interfaces generated based on their capabilities. Both forms of generating adaptable interfaces had better results than the baseline interface [7].

One proposal to improve accessibility of the mobile touchscreen technology is to create a shared user model to be used across applications and devices. This solution collects information about the user during application usage and continually updates the model. The interfaces generated are then created accordingly with the updated model. Bearing in mind that accessibility needs may change across a session, it is fundamental to continually collect user interactions. With all the touchscreen data collected it is possible to calculate optimal properties for objects on a per user basis across screen locations within the device [20].

Supple [8] is a system developed with the purpose to adapt UI controls on runtime in order to provide a personalised view of the contents on a pc. The system allows users with motor impairments to have their interface adapted accordingly with their specific needs.

2.4 Mobile accessibility solutions

Currently, the smartphone market is dominated by Android with a staggering 80% market share followed by Apple iOS with 14.4% in the third quarter of 2013[9]. The usage of smartphones by the disabled community has also seen a large increase in the latest years. This is supported by a series of 4 surveys done about their preferred screen reader. They were made over a span of nearly 3 and a half years. One of its remarkable findings

[9] http://www.idc.com/getdoc.jsp?containerId=prUS24442013 , Last visited in 26/08/2014

is the increase from 12% in 2009 to 72% in 2012 in the use of screen readers on a mobile device[10]. This data suggests a shift from "feature phones" to smartphones. Feature phones have a closed OS, it is developed by the manufactures of the device and all of its functionalities come out of the box. Normally, this type of device does not allow the creation of third party software. Due to the open creation of third party software in smartphones, we focused our attention in these devices. In this section, we will discuss some of the more relevant accessibility functionalities of the iOS and Android due to their dominating position on the market and a clear trend in the adoption of smartphones.

2.4.1 iOS Accessibility

The iPhone as the first truly successful exclusively touch screen interface faced many accessibility challenges. Apple, with the advent of the iPhone 3GS model in 2007 made a step forward towards mobile accessibility. It introduced Voiceover, the screen reader that was previously used exclusive in the Mac OS. Voiceover was more than a copy from its desktop counterpart, it came with a unique set of touch gestures to control the mobile interface. Using Voiceover, we can explore the screen by tapping elements or moving over them, when we do so, audio feedback is provided from the element we touched. A user can also do flick motions to navigate between next and previous content without any knowledge of the on screen location of the content. To select an item the user simply has to double tap the screen. Voiceover was the first accessibility feature introduced in the iOS. Since then, Apple has continually improved its device with new features (e.g. Fig. 2). Here are some of the most relevant features:

Fig. 2 - iOS accessibility settings

[10] http://webaim.org/projects/screenreadersurvey4/, Last visited in 26/08/2014

- Zoom[11]– it is a built-in magnifier that works system wide, by doing a double tap with three fingers a user can instantly zoom screen content. Zoom can be adjusted anywhere from 100 to 500 percent;
- Siri[12]– it is a speech recognition system that allows you to retrieve and create information through speech;
- Switch Control[13] – allows you to control the OS using a single switch or multiple switches. It has three basic methods: item scanning (i.e. auto navigation of the items until the switch is pressed), point scanning (i.e. crosshair to select a screen location), and manual selection (i.e. manually navigate items, needs more than a single switch). One of its key features is the ability to use different switches, from the use of the touch screen as a switch to the use of external Bluetooth popular switches.

Accessibility in the iOS relies greatly on built-in features. Third party developers are heavily restricted by the control they can achieve over the system.

2.4.2 Android

Android is an open-source mobile operating system developed by Google. Android, much like the iOS, provides a couple of built-in accessibility features. One big difference is the approach towards mobile accessibility. While Apple focused on providing themselves all the accessibility solutions, the Android OS allows the creation of accessibility solutions. In Android, accessibility solutions can be designed in one of two ways: custom made solutions or system-wide solutions that can be divided into accessibility services or input methods.

11 https://www.apple.com/accessibility/ios/, Last visited in 26/08/2014
12 https://www.apple.com/ios/siri/, Last visited in 26/08/2014
13 http://support.apple.com/kb/HT5886 /, Last visited in 26/08/2014

As mentioned in section 1.2, custom made applications seek to replace all the look and feel of the device. One example of this is a commercial screen access package, Mobile Accessibility (Fig. 3). It consists of a set of 10 applications that provide an alternative to Android core applications. It has a screen reader and supports braille displays and input via several control interfaces (e.g. keyboard, trackball, touch screen gestures). ClickToPhone is an application that, paired up with HouseMate[14] hardware, creates an accessible solution for physical impaired users to control their mobile devices and the environment surrounding them. HouseMate is a Bluetooth switch controller and environmental controller. It communicates the switch presses to the ClickToPhone application. The ClickToPhone is an Android app that has been custom made for switch access. Much like Mobile Accessibility, this solution focus on one particular set of users. Another example of a custom made solution is EasyPhone [22]. The application was created to enable to performing simple tasks (e.g. make calls, clock, battery) to a user with multiple disabilities. This study raised several issues of current mobile OS shortcomings in respect to "system-wide" accessibility options, at both a user and developer level.

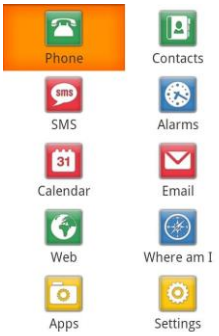

Fig. 3 - Mobile Accessibility app

TeclaAccess is an android input method that can be paired with a switch controller (Tecla Shield) to enable external switch access to Android. "*The Tecla Shield DOS™ is a wireless device that lets you control a smartphone, tablet or computer (PC & Laptop) using your external switches or the driving controls of your powered wheelchair*".

Fig. 4 Tecla Overlay

TeclaAccess provides automatic scanning, manual scanning, inverse scanning, adjustment of the scanning speed, navigation on-screen keyboard, typing on-screen, system-wide voice assistance and it is multi-platform. To scan and allow navigation on

[14] http://click2go.ie/wp-content/uploads/2013/04/housemate_5pages.pdf, Last visited in 26/08/2014

the Android platform Tecla controls the device by emulating a keyboard. Navigation can be done in four directions through an on screen keyboard overlay (i.e. Fig. 4), it increases the keyboard capabilities by adding specific functions to access system wide options (e.g. voice assistance, back). Tecla Shield is a solution that costs 349$[15], it allows the use of all applications on an Android as long as they are prepared to be accessed via keyboard which, many times, is not the case, while in iOS Tecla Shield can be used as an external switch.

One Android accessibility service is TalkBack. Originally, TalkBack was a service that gave audio feedback of the results of the actions performed, notifications and events. TalkBack was only able to announce results of nodes that gained focus through a keyboard or a four-way direction pad. With the release of Android 4.0, users saw its functionality expanded through Explore by Touch. This option of TalkBack allows the user to explore the contents of the screen in a very similar way as VoiceOver does. A recent contribution to mobile accessibility was made by the accessibility service JustSpeak [30]. JustSpeak is a voice control solution that enables non-visual access to the Android OS. The service creates a set of available voice controls based on the content descriptions retrieved from the screen application context. It has a speech recognizer that processes the voice command and matches it with one of the current content descriptions. One of its novel features is the ability to do multiple commands using a single utterance.

2.5 Interaction Logging

Being able to record interactions system wide is a key feature to provide researchers with the capabilities to do repeatable HCI evaluations. Input observer [2] is a system that quietly observes user interaction with a personal computer. The popularity of mobile technologies and interactions has led to an increase of app development and HCI research for these devices. Despite the prevalent nature of these technologies, the platforms are lacking in guidance and tools to support testing and simulation of user interactions. It is critical to develop a solution that allow the creation of similar systems to WebAnyWhere ABD [3]. This system permits the recording of accessibility problems at the time user experience them. This in conjunction with the ability to reproduce the problem can

[15] http://komodoopenlab.com/tecla/ , Last Visited in 26/08/2014

provide a powerful tool for testing and simulation. Lookback[16] is a simple recording interface that can be installed into developers' applications. Currently is only available for iOS and provides developers with the capability of recording the user experience (i.e. screen capture and recording user reaction through the iPhone camera). There is clearly a need to reproduce user experiences in mobile settings in order to perform HCI evaluations.

2.6 Discussion

Mobile accessibility has been improving over the years, with the introduction of smartphones and tablet devices new challenges and opportunities arouse. For example, we can take advantage of these devices by creating a multitude of control interfaces or developing adaptable interface. The iOS and Android both have similar fundamental accessibility features, like screen readers, speech recognition and zoom. The iOS has great built-in accessibility features, unfortunately third party developers are heavily restricted in their control over the system. Custom made applications are the only approach available to them. However, Android open source nature empowers developers, they are able to create input methods, accessibility services and also custom made applications. These third party custom applications aim to replace core system applications like phone, contacts, calendar, with custom made accessible ones. The key word here is for who? Depending on the target user group, these applications will be different. These solutions are costly and time consuming to develop. Input methods although they possess system wide capabilities they greatly relied on the application being accessible via keyboard directional pad and also overlook the content that is being navigated. Accessibility services are a step in the right direction, unfortunately they provided limited control over input and the relation between content and it is severely overlooked. Navigation and selection methods should always be customizable in order to address specific user needs with. Creating system wide control interfaces as a third party developer is a massive challenged due to all the restrictions.

Adaptive interfaces is one way to improve accessibility. The key is to adapt to every single different user which could benefit immensely user with disabilities. The answer is

16 https://lookback.io/, Last Visited in 26/08/2014

to develop these interfaces automatically with some sort of user ability record. Unfortunately, current OS only allow the customization of interfaces of your own developed applications and not across third party or stock ones. Another issue is how to record user performance on interfaces and adapt accordingly if we cannot have input control outside our applications. Interaction logging is currently lacking in the mobile context when we compared to the apparatus and procedures already found on its pc counterpart. There needs to be a simple way to record and simulate interactions on a mobile device in order to further the field of HCI research.

Chapter 3 Enabling Mobile System-Wide AT

The previous chapter review the current state of the art in assistive technologies in a mobile context. We discuss their shortcomings by analysing the top 2 OS systems of the markets and reporting their capabilities and deficiencies. In this chapter we will present SWAT a system wide library that targets the development of assistive technologies. We describe several use case scenarios that inspired the creation of SWAT. To contextualize our application we discuss the Android OS architecture and capabilities, since our library will take advantage of it. We delve into the SWAT library with a detailed view of the system requirements, architecture, components, features and uses.

3.1 Use Case Scenarios

Miguel was one of the inspirations for this project, he has limited neck movement and only has residual arm movement. Miguel had an accident that left him blind, tetraplegic and with a stutter speech.

Miguel wants to send a message.

Miguel is sitting in his chair and wants to contact his friend André. He wishes to send him a message wishing him good luck for his thesis defence. He is not sure of hour of the defence and as such he does not want to call and risk interrupt it. He starts by pressing a switch which enables the auto-scanning of the options on his smartphone, every time a new option is focused an auditory feedback is provided. He waits for the option Message to be read to press the switch. He then auto navigates the Message menu and since he has traded messages with André before he presses the switch when André name is read. Next he selects the "Write message" and a keyboard is enabled. The keyboard is navigated automatically and Miguel presses the switch whenever the letter he wants is read. After writing it he closes the keyboard, navigates to the send message option and completes his wish.

Miguel's wants to use a new contact application that his friend told him about.

Miguel friend came over and told him about this great application on the play store that he is using, that organizes the contacts in a way that is much quicker to access then.

Miguel asks his friend to install this application on his phone since this task requires a little digging and a lot of menu navigation. His friend stops the auto-navigation, gets to the play store and installs the new application. He then resumes the auto-navigation for Miguel to try it out. Miguel travels through the menus and selects the new contact application. The system reads in a row-column scheme the options available on application. After a week of use Miguel's feels comfortable using this new application and his able to quickly navigate the new menus and contact whoever he wishes faster.

Miguel wants to listen to radio on his smartphone.

Miguel is tired of listening to the TV he much prefers to listen to a radio station. He decides to try the radio application installed on his smartphone. He starts the auto-navigation and selects the radio application. With the auto-scanning he is able to travel through different channels and choose a desire station to start playing. After a while he gets tired and decides he wants to close the radio station and listen to the TV since its news time. He navigates through all the menus but cannot find the stop or close radio station. Unfortunately the stop button is embedded in an inaccessible section of the application. He calls for his mother to try and resolve the issue. Knowingly he might want to turn the radio on for latter, his mother instead of simply turning of the radio, she records the action of her stopping the radio. As expected latter on Miguel enables the radio station again. It is football game time and Miguel wants to stop the radio, since he knows his mother recorded her action he navigates through the list of recorded actions and selects the radio off option and is now ready to watch the game.

Miguel's mother notices he takes a while when he wants to send a message to his friend Rui.

This week Miguel has a lot of stuff arranged with Rui. Rui is in Portugal for the week and he is not a usual contact for him. Every day for the whole week Miguel would contact Rui during the day several times. Miguel's mother noticed Miguel took a long time to start writing a message to Rui since he had to go through several screens with many options. She took his phone, pause the automatic navigation and recorded the whole process from the home screen until the "write a message to Rui" screen. Miguel had now an icon on his home screen called "Write to Rui" upon pressing it he would quickly face

the task of writing a message, overcoming the otherwise tedious process of getting to this stage.

3.2 Android Operating System

In order to understand the library developed it is important to have a mental model of where it is operating. The Android OS consists of different stacked layers, where each layer provides services to the layer above.

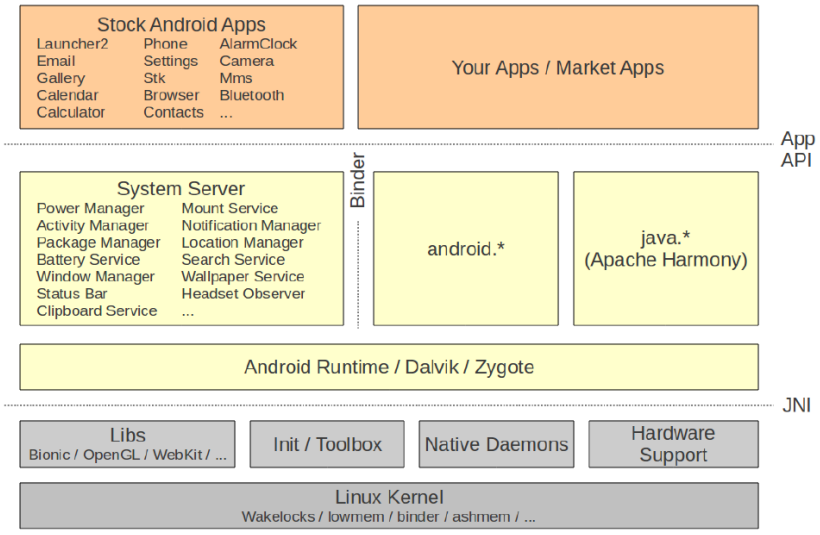

Fig. 5 - Android Architecture

As we can see in Fig. 5 on the top layer we have the Stock Android apps and the third party developed ones. Both share the same capabilities so any developer can create applications that serve as an alternative to stock ones or simply design new ones, always relying on the layers bellow. The System Server layer is composed of the components that handle applications cycles, manage data sharing between applications and so on. The Android Runtime layer is where the Dalvik Virtual Machine (e.g. a JVM) runs applications, this JVM allows multiple instances to be created simultaneous providing among others: isolation, security and memory management. The Libraries layer consists of the native Android libraries. These libraries are crucial for the system and some of them are: OpenGL (e.g. graphical rendering) and SQLite (e.g. database engine). The layer bellow is the Linux kernel where the Android OS is built upon.

"Android is a privilege-separated operating system, in which each application runs with a distinct system identity"[17]. The Linux, thereby isolates each application from one another and from the system itself. Together with a permission system that enforces restrictions on specific operations, these systems are the core of security in Android. Each Android application operates in a process sandbox, they can only share data and resources if they explicitly do it. This sandbox is created across all types of applications developed being it Java, native or hybrid.

The Android NDK allows developers to create solution that resorts to native-code of the device. This can be extremely useful since it allows developers to reuse native libraries not available otherwise. Normally applications that take advantage of such setup are CPU-intensive ones. Using the Java Native Interface (JNI) we are able to through our Android Java application interact with native code developed.

Apart from the Activities developers are also able to create Services. A Service is in fact a component of an application that can perform long-running operation in the background without relying in a user interface. The Services can run even if the application that initialize it is out of focus. They can be used to perform network transitions, perform file I/O among others.

Fig. 6 - Talk Back accessibility service

Accessibility Services although they share the tag service they are a different species. Unlike a simple service, an Accessibility one can be a standalone version or bundled with a normal application. They were designed to provide alternative navigation feedback to the end user. An accessibility service has the permission to communicate with the user on the application's behalf, it can provide text to speech, or haptic feedback. Starting on API level 14, the accessibility services can perform actions on behalf of the users (e.g. they

[17] http://developer.android.com/guide/topics/security/permissions.html, Last Visited in 26/08/2014

can change the input method, perform selection). The range of as been expanding actions (e.g. scrolling list, interacting with text fields) since its methods creating a richer environment for accessibility developers. Through the accessibility services we are able query the view hierarchy for more context. Since API level 14 we have the ability to query the view whenever an AccessibilityEvent that we are listening for is triggered. Through the event we able to retrieve the UI component that generated it and from it we can gather its parent, children, bounds, available actions, text and. Android provides information to the accessibility services mostly through the onAccessibilityEvent() call-back method that passes an AcessibilityEvent object. This object has information about the event that was triggered and its UI source, possible events include:

- TYPE_VIEW_CLICKED: represents the event of clicking in a View;
- TYPE_NOTIFICATION_STATE_CHANGED: event showing a notification;
- TYPE_VIEW_TEXT_CHANGED: triggered when an EditText is changed;

When an accessibility service is developed and installed in a system it must be manually enabled through the accessibility option on the system settings. Examples of this are services like TalkBack or JustSpeak.

3.3 SWAT

SWAT is a system wide Android library that empowers the development of assistive technologies, but due to its capabilities it enables the creation of powerful system wide services. At its core it provides developers with a control of the system that normally they would not have at a system level.

3.3.1 System Requirements

The development of this library was preceded by this list of requirements.

- **Cross application view Hierarchy access**: access view hierarchy across all applications and system windows;
- **Low level input access**: enable monitoring, blocking and injecting of all the system internal devices (e.g. touchscreen, keypad, accelerometer);
- **External control of the device:** fully control the device through an external interface;

- **Easy coupling and development:** simple coupling of new control interfaces;
- **Output parameterization:** different adaptable output options, namely sound, TTS and visual;
- **Adapting content:** the ability to personalize content, independently ownership of the application across both system and third party ones;
- **Adapting input:** the ability to in real time monitor input and adapt it, for instance using touch adaptive models;
- **Navigational control:** ability to navigate all device applications programmatically and easily create new navigation methods;
- **Perform actions system-wide:** be able to perform actions independently of the current application in focus, action such as back, home, select, scroll, slide and text-entry;

3.3.2 System Architecture

From reading the section 3.2 we understand the restrictions imposed on the cross application access and manipulation. SWAT is built upon an accessibility service, as such it is able to retrieve and disseminate all view hierarchical information.

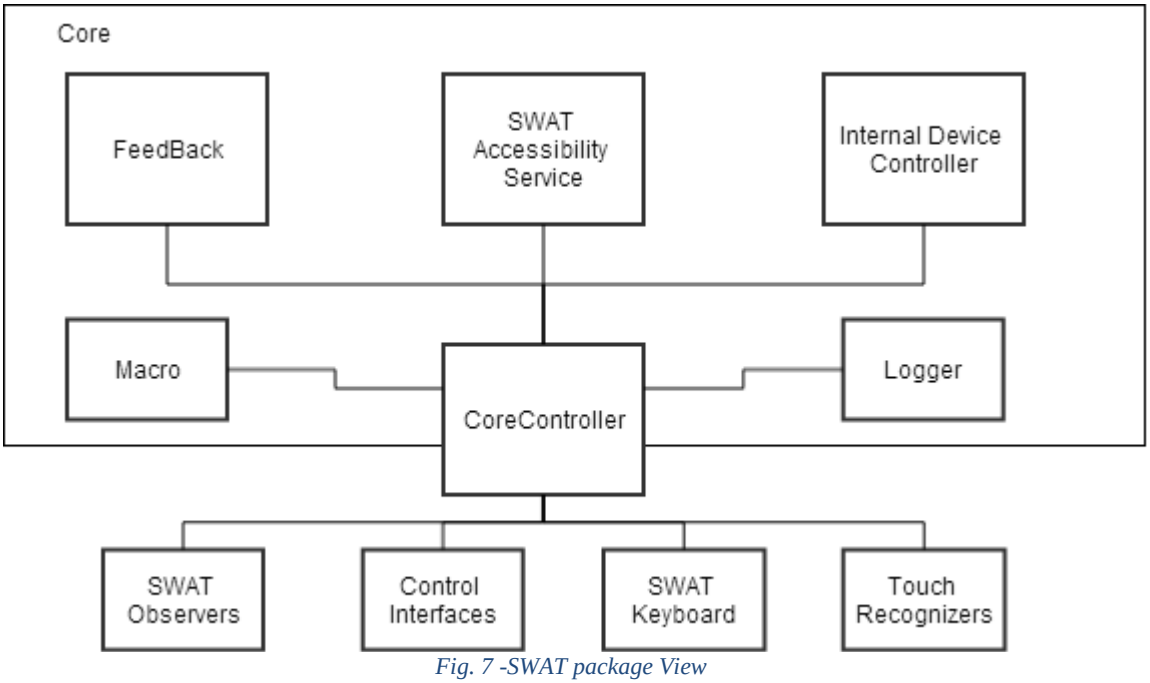

Fig. 7 -SWAT package View

With the accessibility services we overcome it in regards with content information, but there is one missing link, input. Through the accessibility services we are able to

perform actions directly on content but we have no control over the input created by the user. Using the Android NDK (i.e. native development kit) in conjunction with a rooted device SWAT is able to control the internal devices providing a finer control over input and output. The root allows us to change the permissions of access to the system and manipulate the internal devices directly by acquiring the input events before they are process by the respective drivers and later the OS, we are operating in the bottom layer, the Linux kernel (e.g. Fig. 5). We create an opening in the sandbox protection in order to manipulate input system-wide. As a library should be, it was developed to be extensible and adaptable to developer's needs. All of SWAT components were created to fulfil a specific need in the development of Assistive Technologies.

SWAT has a BUS like architecture. In Fig. 7 we have an overview of the package view of SWAT. There are three critical modules to the library: Core Controller, SWAT Accessibility Service and Internal Device Controller.

Core Controller

The Core Controller handles all communication between the different core components as well as external ones. All communication with the library is done through it, in Fig. 8 we can see a part of the architecture behind the *CoreController*. He is responsible for gathering information of the other two core modules and disseminate their information. The *HiearchicalService* is the main class behind the SWAT Accessibility module, while the Monitor is the one behind Internal Device Control. The external classes mostly communicate with SWAT through the available interfaces like the *IOReceiver*. The broader range of operation available from *highlight(…)* (e.g. highlight a specific area of the screen) to *getDevices()* (e.g. get the name of all the internal devices by driver) are perform by making a static call to the Core Controller.

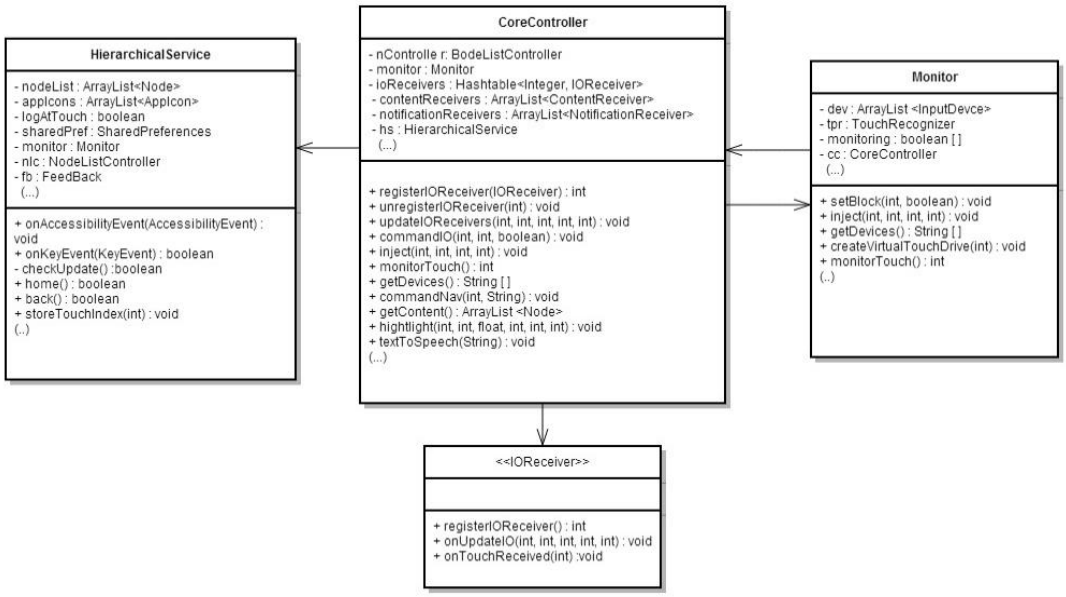

Fig. 8 - Core Controller preview

SWAT Accessibility Services

This module responsible for gathering information from the Android Accessibility Services to provide the required view hierarchical of the on screen content. It creates an internal representation of the on screen content which can be access through the Core Controller or by the SWAT Content Observer. It is composed of 4 classes (e.g. Fig. 9):

- *HierarchicalService*: the main class of the service, extends the accessibility service and handles all the content updates;
- *Service Preferences*: creates a preference window for the SWAT Accessibility Service. Through it we are able to enable and disable the built-in features of SWAT;
- *NodeListController*: control over an internal representation of the content, has available a set of navigational operation over the content (e.g. navNext());
- *Node*: internal representation of each view hierarchical node;
- *AppIcon*: internal representation of all the application start icons;

As mentioned before SWAT heavily relies on the information that is able to retrieve from the accessibility services. SWAT listen to all accessibility events and every time one is triggered it cross checks it with our internal representation to see if the view hierarchy was updated. If it was, it updates our internal representation and pushes the update to the

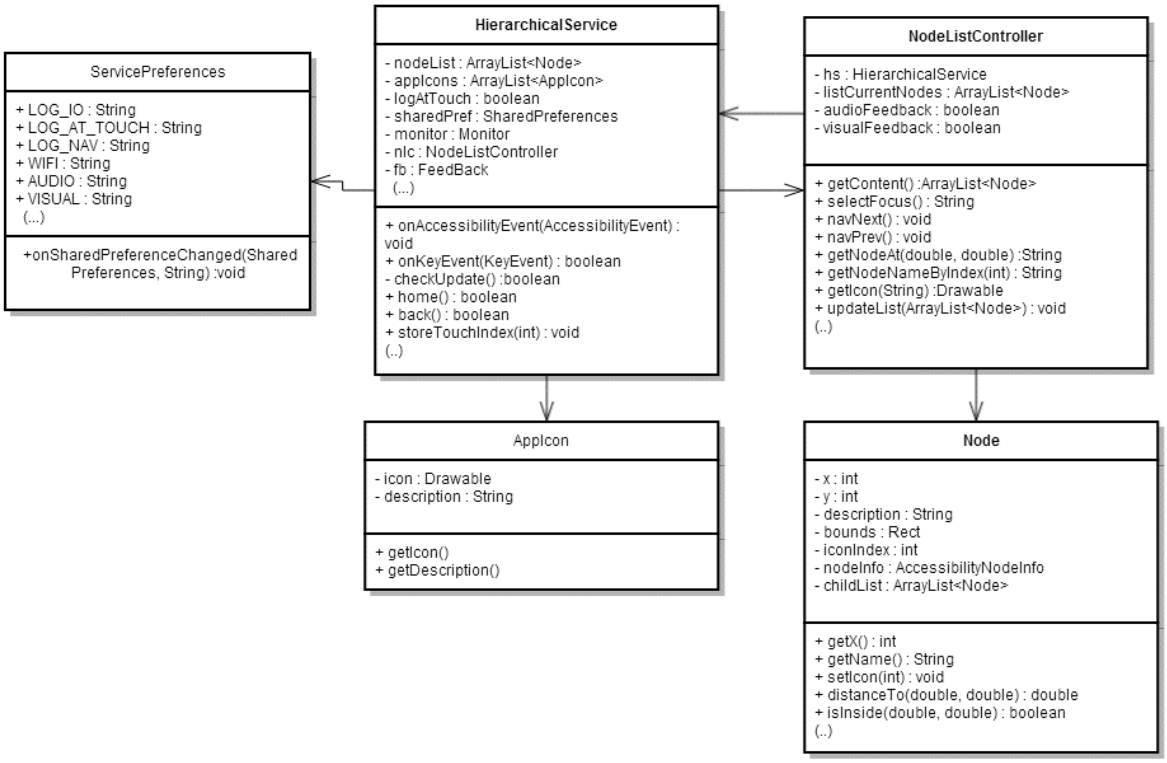

Fig. 9 - SWAT Accessibility Service

Core Controller which is responsible to propagate it to the registered content observers as shown in the sequence diagram bellow (Fig. 10).

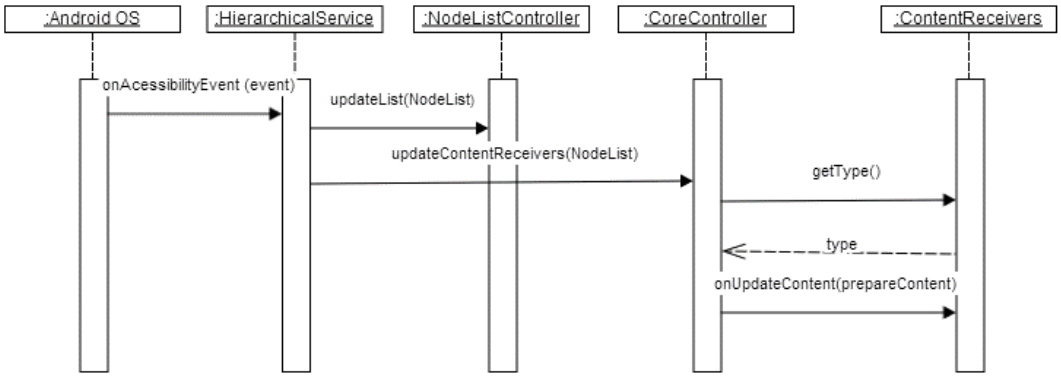

Fig. 10 - Content Update Sequence Diagram

Our internal representation tends to use content descriptions or names to label the content, some features can be compromised if android applications do not comply with accessibility norms. These content nodes created in our internal representation enable us to navigate the system. It is through the Accessibility services that we are able to click content programmatically and perform global actions:

```
public boolean back() {
    return performGlobalAction(AccessibilityService.GLOBAL_ACTION_BACK);
}
```

The accessibility events triggered by the Android system provide us with key information, there are some fundamental events that SWAT uses and they are the Click, Notification and events that originate from any edit box, they were described in section 3.2 :

- Click - used to track user activity which allows the creation of log mechanisms;
- Notification – listen for notifications, similar to the process shown in Fig. 10, the notifications are pushed to the *CoreController* that disseminates it to the registered Notification Observers;
- Edit Box –we track all events and filter the ones that originate from edit boxes. Through the accessibility services it is not possible to determine key presses. By analysing this events we are able to track changes in any non-password edit box and log the text entry;

Content information is gathered from the Android Accessibility Service and stored in an *ArrayList* structure of *Nodes* (e.g. Fig. 9). The key information present in each node is: position and bounds, class and package name, description and text, actions (i.e. actions that we are able to perform on the node, click, scroll, etc), parent and children. Developers are able to retrieve content information through the use of the Content SWAT Observer fulfilling the first requirement, **cross application view hierarchy access.**

Internal Device Controller

Using a rooted device together with the NDK enabled us to surpass normal permission levels given to developers. Normally we cannot access internal devices states and listen to events at a low level. To accomplish this we use the NDK in order to be able to handle the input event nodes. We started with a library that uses JNI (Java Native Interface) create by Radu Motisan[18] that allow us to inject/monitor internal devices. This library starts by opening the input events nodes of the system, if it does not has permission

18 http://www.pocketmagic.net/2013/01/programmatically-injecting-events-on-android-part-2/#.UiW619Kfjz8/, Last Visited in 26/08/2014

to do, it changes the file permissions resorting to the *Shell* class. The *Shell* class is able to execute shell commands with system level permission.

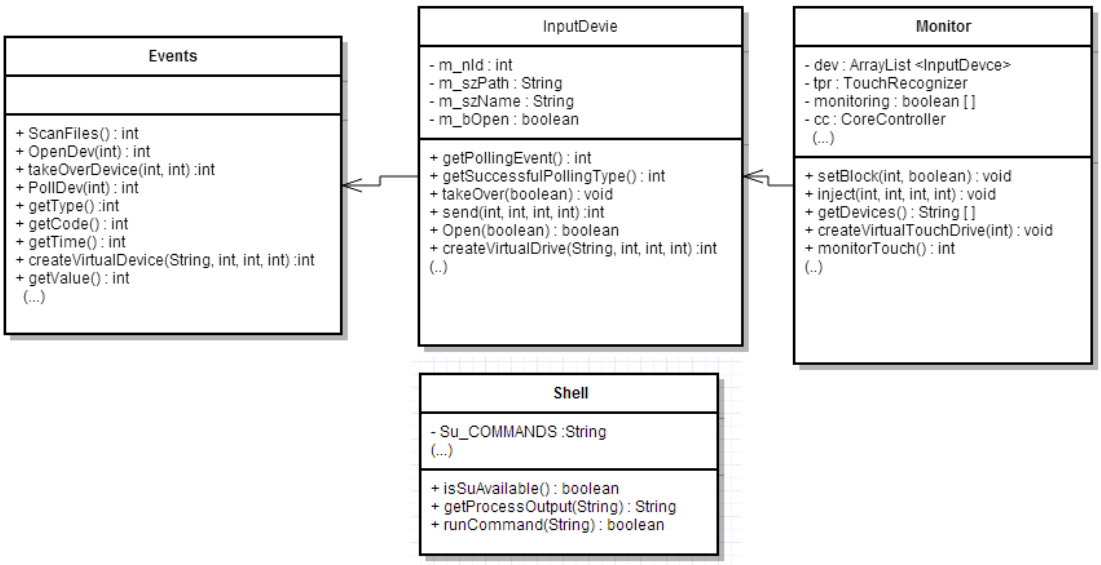

Fig. 11 - Control Devices Architecture

Through the JNI we are able to create *InputDevice* instances that correspond to the internal devices of the system. These are the instances that we manipulate in order to monitor/inject events into these devices. We expanded this library (e.g. Fig. 11) in order to accommodate our specific needs. It is now also able to block all events (e.g. block all touches from being interpret) and specifically for the touchscreen we are able to create a virtual drive.

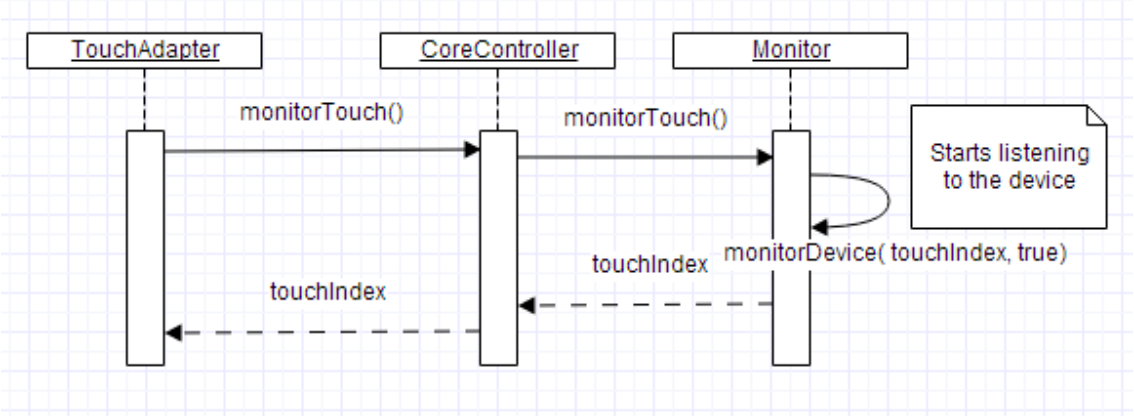

Fig. 12 – Starting the monitoring process

In Fig. 12 we can observe the sequence behind start monitoring the touch device. As an example, this virtual touchscreen drive can be used to adapt touches in real time, by blocking the default touchscreen input while monitoring it we are able to inject the adapted touches into the virtual drive which interprets and performs them. Even though our presented studies do not use this feature we believe it is a novel and powerful tool for developers and researches especially in the field of touch models.

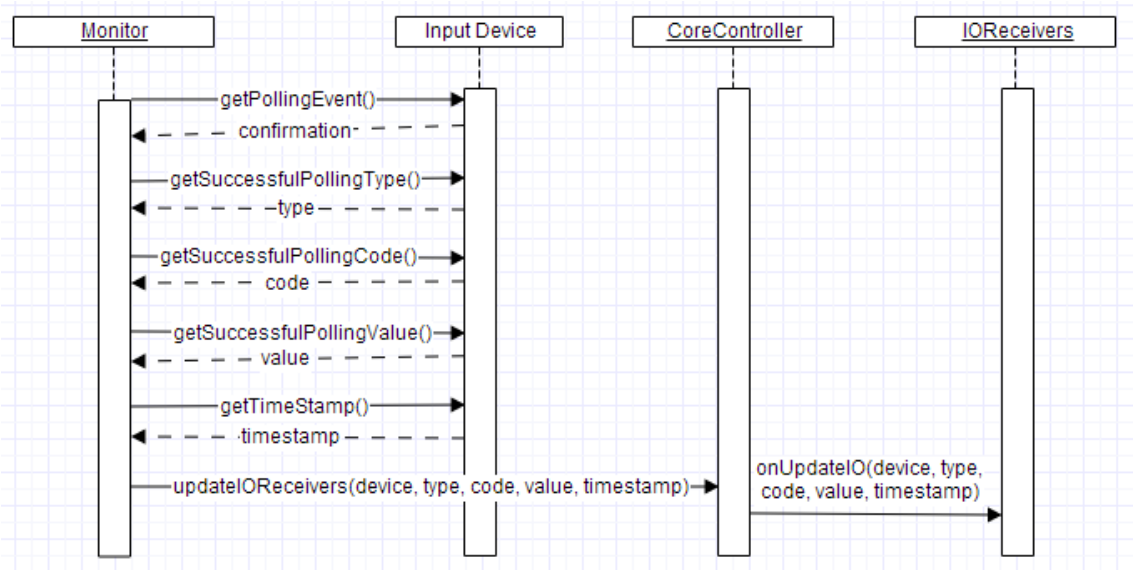

Fig. 13 - Monitoring sequence

The Internal Device Controller is the package responsible for all of this plus the propagation of the events of the monitored devices to the Core Controller which then pushes the events to the registered observers (e.g. monitoring sequence is shown in Fig. 13). To receive this events outside our library all the developer has to do is implement the *IOReceiver* interface and register it. With this module we successfully allowed **access to low level input** and provided the ability to **adapt it**.

3.3.3 Specific Components:

The fundamental components of SWAT were already described, but alone they accomplish very little. SWAT was coupled with 3 more components to achieve its goals.

Touch Recognizers

We are intercepting low level input before they are process by the system. This means we get raw data from the sensors (e.g. touch screen, accelerometer) like the one displayed in Fig. 14 (e.g. input data from the touchscreen that represents a tap).

```
5,3,53,142,23818538
5,3,54,505,23818538
5,3,58,59,23818538
5,3,48,3,23818538
5,3,57,0,23818538
5,0,2,0,23818538
5,0,0,0,23818538
5,0,2,0,23818666
5,0,0,0,23818666
```

Fig. 14 - Raw data

Although we allow developers to garnish this data, many developers are only interested in a higher level of knowledge of input. To accommodate all needs we created the Touch Recognizers. They process this low level events from the touchscreen and categorize them accordingly (i.e. down/move/up or touched/slide/long press).

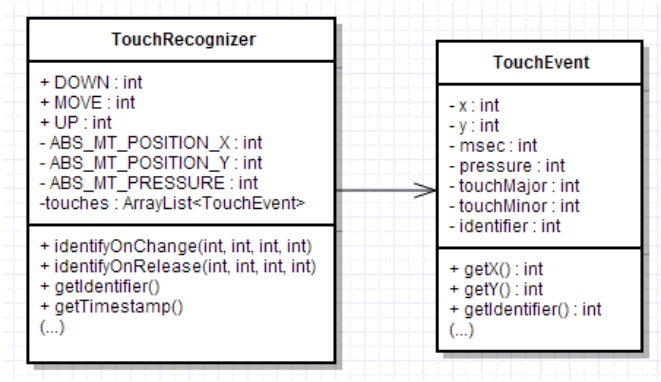

Fig. 15 - Touch Recognizer

These events can vary from touchscreen to touchscreen as such there are several touch recognizers implemented that can be activated through the preferences of the service. The package (Fig. 15) is composed by the abstract class *TouchRecognizer* and the object that represent a touch input even (e.g. *TouchEvent*). The recognizers created extend the abstract class and do a personalized implementation of the method *identifyOnChange(…)* to categorize touch in down/move/up.

Feedback

One crucial part of the development of assistive technologies is the feedback provided back to users. In our library we pre-built two feedback mechanism that developers can take advantage. SWAT resorts to the Android TTS (i.e. Text to Speech) to provide audio feedback to the users. The two main feedback types used in the mobile context are visual and auditory, to cover the first SWAT has a highlight function that allows to add highlights to a specific region, with a specific colour and transparency using a simple overlay. With both this mechanism we ensured the **output parameterization** we required.

SWAT Keyboard

Through the accessibility services we are not able to fill text edit boxes, nor can we simulate key presses. To allow the creation of custom keyboards through SWAT we must first install a SWAT input method.

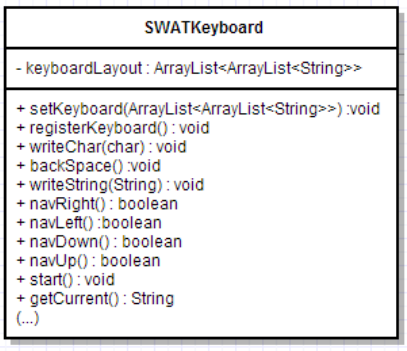

Fig. 16 - SWAT Keyboard

This SWAT input method is nothing more than the traditional on screen keyboard, with one slight difference, it listen for a specific broadcast to write characters/phrases. After we select this input method as the default one on Android we are ready to create our own custom keyboard on our SWAT solution. SWAT Keyboard (e.g. Fig. 16) is an abstract class that can be extended to create our own custom keyboard. To create it we have to specify the key layout in a row-column scheme (e.g. *setKeyboard(…)*) and register the keyboard as the default one for SWAT. We can personalize its behaviour on start, show, hide and update. By default every keyboard on SWAT possesses a 4 way navigation scheme to allow an easy pairing with assistive technologies. One example of such keyboard will be described in the Multi-Impaired case study.

Interfaces:

We have seen all of the SWAT fundamental modules and components, the missing link is how we can take advantage of the library developed. We created a set of interfaces that developers are able to extend and implement in order to take full advantage of SWAT.

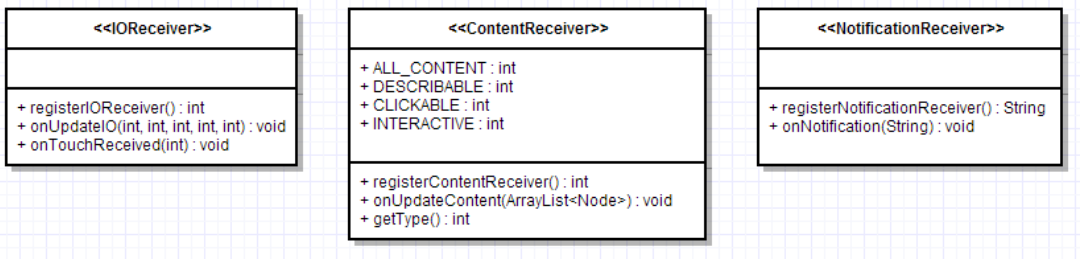

Fig. 17 - SWAT Observers

The SWAT Observers package (e.g. Fig. 17) is composed of three interfaces, Content Receiver, IO Receiver, and Notifications Receiver. By implement the Observer interfaces developers are able to get updates of the respective category (i.e. content updates, low level device events and notifications). The Observers are share the same registration process but accordingly with their purpose they have different capabilities. Through the *ContentReceiver* we are able to acquire all the hierarchical view configuration. Most of the times we are not interested on the full tree since many nodes serve only to organize layout (e.g. linear layouts, relative layouts) and contain no real content. To unclutter the content retrieved in the receiver we created a filtering system for what type of content we are interested in observer:

- All Content – retrieves all content information available;
- Describable – only nodes with either a description or text are retrieved;
- Clickable – exclusively nodes that are clickable;
- Interactive – nodes that are simultaneous clickable and describable;

To start receiving content updates all developers have to do is implement the Content Receiver Interface, specify the type of input there are interested by returning the desired type in *getType()* method and register the observer. After those simple steps are completed we are ready to receive content updates. The content is provided in an ArrayList of Nodes through the *onUpdate()* method. Since we are able to filter layout modes it is important to keep track of which regions are scrollable, as such, if there is any scrollable content on the view hierarchy we add at the end of the array list a node with the description "scroll" to provide an easy way for developers to be aware of the scrolling and be able to perform it.

The Notification Observer of all three is the simpler one. Its only purpose is to provide all the notifications to the developer even if they are not related with their application (e.g. message notification, alarms). To start receiving all notifications all developers have to do is implement the *NotificationReceiver* Interface and register the observer. Notifications are received in a String format by the *onUpdate(notification)* method.

The IO Receiver has the same registering process as the previous two. It gets its update from the *onUpdate(...)* method which contains the low level input event information:

```
public abstract void onUpdateIO(int device, int type, int code, int value,
        int timestamp);
```

When the device monitored is the touchscreen we are able to process this events using the Touch Recognizers previously described. This observer possesses one extra method the *onTouchReceived(touchType)* this allows external sources to send touch types to be interpret by the IO Receiver. This is usefull when we map touch types to specific actions, if we develop a switch enabled assistive technology we can use this method to translate the switch presses to touch types.

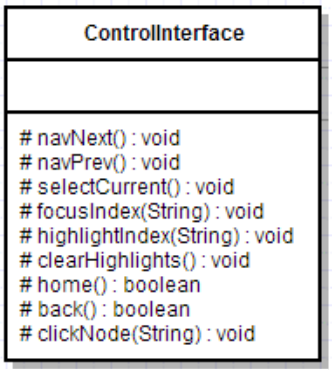

Fig. 18- Control Interface

Control Interfaces (e.g. Fig. 18) allow developers to navigate through content programmatically and easy coupling of new interfaces. The Control Interface class is one of the key elements of SWAT. Much like the SWAT Observers this abstract class is intended to be extended by developers. It is responsible for all the navigation capabilities of SWAT. Through it we are able to click, focus, navigate to the next/previous element and perform global actions like back and home. By extending this class developers have an easy and fast way to map their assistive technology to a simple navigation mechanism supported by SWAT. Using the Control Interface in conjunction with the Content Observer developers can create custom solutions that navigate the content as they see fit. With this approach we accomplish **easy coupling**, **navigational control** and we have the ability to **perform system wide actions**.

One example of this is the Multi-Impaired case study. To briefly explain, it gathers the information from the Content Observer, it filters the desired content and creates a custom navigation mechanism, when the user selects an option it is through the Control Interface that the selection is made. Both our case studies use extensively this class. Wi-Fi Control is another class of this package and it takes advantage of the Control Interface

by extending it. Wi-Fi Control is a server within SWAT that listen for navigation commands. All the Control Interface capabilities are enable via Wi-Fi, meaning we can control our device solely through the network. It has an added capability over the typical Control Interface, we are also able to send touch types to our IO Observers. This feature was design to simulate touches via Wi-Fi in order to be able to map not only commands to your assistive technology (e.g. next option, select) but also touches (e.g. touch, slide). Wifi-Control is both an example of an external controller and a possible implementation of the Control Interface. With it we created an easy to develop solution for **external control of the device.**

3.3.4 Native features

We have reviewed the system capabilities and components. In these sections we described two native features of SWAT that take advantage of its prowess in the creation of system-wide solutions.

Logging

We now know that through SWAT we have click events which we can determine what was clicked. With the Internal Device Controller we are able to monitor the touchscreen, using the proper Touch Recognizer we can identify the types of touches performed. With the combination of the two abilities above we developed three types of logging mechanism:

- Log IO – logs all touch events categorized in down/move/up;
- Log Navigation – logs all the navigation steps, what was clicked (e.g. applications->settings->Wi-Fi);
- Log Interaction – combination of the two previous loggers, logs all touches and navigation steps;

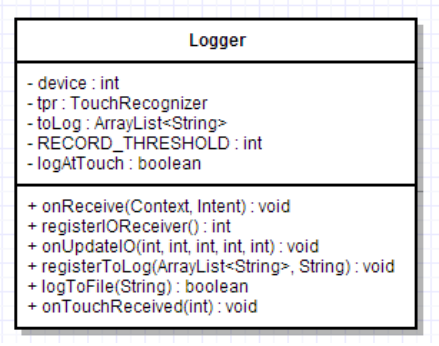

Fig. 19 - Logger class

SWAT provides this three logging capabilities by default. They can be enabled/disabled through the service settings activity. Since it might not be optimal to start logging when the service is first launched, it is possible to start the logging proceed recurring to a broadcast signal, which means the logging process can be initialized by another application. The logging process is deeply embedded in the library, while the IO information it requires can be access from the outside the navigational and keystroke information cannot. The relevant information to log tends to be very different from application to application. The Logger (Fig. 19) is a built-in feature that be adapted in one of two ways. You can activate the mode you require and manually send more information to the log using *registerToLog()* or you can extend the Logger class and developed customized logging mechanisms.

Macros

In our Multi-Impaired case study we soon realized the need for simple and short commands due to the nature of the navigation. When thinking about a solution we realized that we had at our disposal all the mechanisms necessary for the creation of macros system-wide. The Macro module consists of an application that is installed onto the Android System when SWAT is installed. This part of SWAT is design as a finished product and it is not intended to be extensible. As such this application targets final users of any SWAT enabled service/application. Our macros possess two recording modes, you can either record your navigation steps (e.g. applications -> settings -> Wi-Fi) or you can record touches (e.g. down x y, move x y…), while recording you can switch at any time between modes. This allows the creation of complex macros that reproduce touches and navigation steps.

To create a macro the user has to open the macro application and give a name to start recording. After doing so the application will close and return to the home screen where it will start recording. In the top right corner it will show the command bar for macro recording with functions to stop recording and changing mode. When we end the macro recording an icon will appear in the home screen with the macro name, by pressing this icon while the SWAT service is active it will perform the recorded macro.

In the background when a macro is being recorded we are in fact listening for click events in our Accessibility Service (i.e. when in navigation step mode) when one is triggered we store the source of the click. If we are in touch recording mode we are monitoring the touch device and storing all the low level events triggered. When the recording ends the macro is saved to a local file even though it is kept in memory. When SWAT service is started it loads all the available macros from the local file.

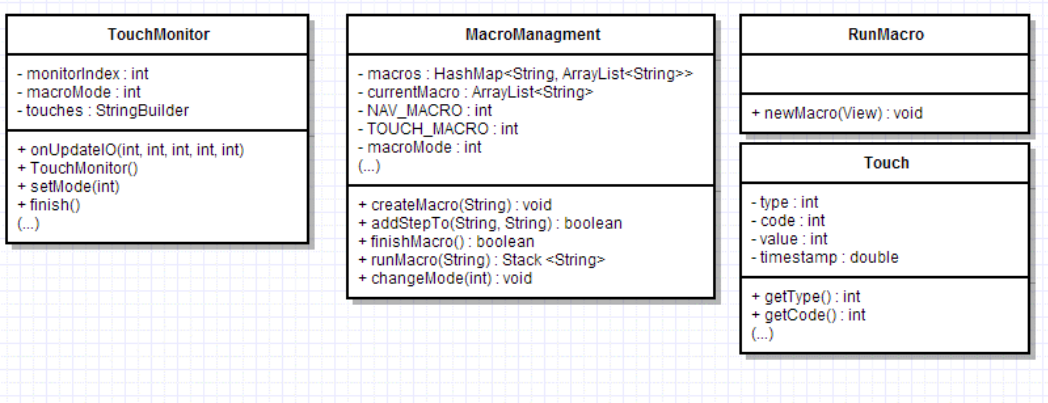

Fig. 20 - Macro Architecture

When a macro is clicked to run, the service starts the macro application (e.g. the *RunMacro* activity) which quickly informs through the Core Controller which macro should be executed. The macro execution is handled according to the recording mode, when in navigation steps, it first tries to click on the current step (e.g. applications) if it cannot find it, it begins to scroll (if scroll is available) until it finds or until it meets defined threshold. When in touch mode, the touches are reproduced with the same time intervals in between. Examples of use will be presented in the Multi-Impaired case study. The macro system is a complex part of SWAT that relies on multiple components to function, the 4 classes shown in Fig. 20 are the ones that are exclusively devise to support the macro feature:

- *MacroManagment* – responsible for handling the recording and loading;
- *TouchMonitor* – monitors and records the IO events during the macro creation process;
- *RunMacro* – this activity has two launch modes, when launched from the application menu it allows the user to start the macro creation process, when it is launched from one of the macro icons it starts the correspondent macro execution;

- *Touch* – representation of a low level input from the touchscreen, it is used to reproduce touches upon the macro execution;

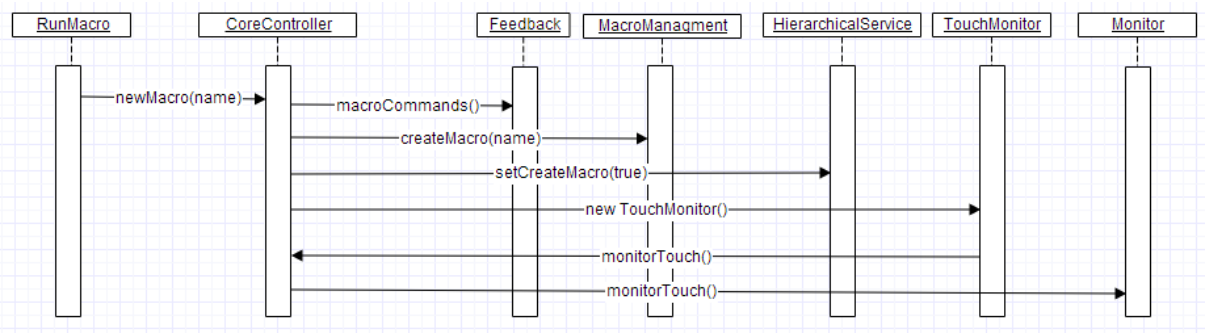

Fig. 21 - Macro recording process

The process of recording and reproducing the navigational steps is handled by the ones above in conjunction with the *HierarchicalService* (e.g. monitors and reproduces the navigation steps), the *Feedback* (e.g. handles the macro recording buttons), *Monitor* (e.g. monitors and reproduces the IO Events) and finally the *CoreController* that is responsible for the communication between all parties to ensure the proper recording and execution of the macros. Fig. 21 represents the sequence diagram that illustrates the complex process of initializing the macro recording process.

3.3.5 Using SWAT

SWAT as a library is intended to be used by others. During its development we found several motivated colleagues in its capabilities. To support the further development of our platform we made it available through a repository in GitHub[19] and created simple tutorials for each interface. To illustrate how simple it is to use SWAT we present the Control Interface tutorial.

The *ControlInterface* class provides developers with the fundamentals methods to create an interfacing scheme. By extending this class developers have access to the following methods:

- *navNext()* – navigates to the next content node on the screen;
- *navPrev()* - navigates to the previous content node on the screen;
- *selectCurrent()* – clicks on the current focused node;

[19] https://github.com/AndreFPRodrigues/Mswat, Last Visited in 26/08/2014

- *clickNode(description)* – clicks the node with the correspondent description;
- *focusIndex(index)* – focus the node;
- *highlighIndex(index)* – highlights the content index;
- *clearHighlights()* – clears all highlights;
- *home()* – performs the global action go home;
- back() – performs the global action back;

Through these methods developers can easily create and manipulate navigation system wide. To demonstrate how to use it, in this tutorial we will develop a simple control interface that will take advantage of this class and the IO Receiver mentioned in another tutorial. We will create a simple control interface that will select a node when we tap the screen and will go to the next node if we slide. To do so we create a class *TouchController* that extends *ControlInterface* and implements *IOReceiver*:

```
public class TouchController extends ControlInterface implements IOReceiver {
```

We want this the control interface to be initialised when the SWAT finishes its start-up process and we want to ensure only one control interface is initialise.

```
<string-array name="mode">
    <item name="1">None</item>
    <item name="2">Simple Navigation</item>
</string-array>
<string-array name="modeValues">
    <item name="1">null</item>
    <item name="2">touchController</item>
</string-array>
```

Fig. 22 - Service preference configuration

To be able to select the *TouchController* as our control interface we must first add it to the service preferences. To do so in the res/values/arrays.xml (e.g. Fig. 22) we add an item to the "mode" and "modeValues" array. We add the description of the control interface "Simple Navigation" and the value "touchController".

```
@Override
public void onReceive(Context context, Intent intent) {

    // Triggered when the service starts
    if (intent.getAction().equals("mswat_init")
            && intent.getExtras().get("controller")
                    .equals("touchController")) {
```

Fig. 23 - On receive SWAT init

Now the simple navigation is available to be selected in the accessibility service preference. We must select it as the active control interface when we test our project. The control interface can be initialised when SWAT finishes its start-up, we simply need to implement the *onReceive()* from the *BroadCastReceiver* and ensure that our interface is

the select one by checking the *Intent* extras (e.g. Fig. 23). With this we are in position to create our own control interface (e.g. Fig. 24).

1. We start by registering the IO Receiver (e.g. the IO Receiver tutorial is available in the attachments);
2. In this control interface we want to react to the users touches so we need to monitor and recognize them. As such we start the monitoring process and we retrieve the active Touch Pattern Recognizer (e.g. TPR, it can be selected in the service preference accordingly to the device touchscreen driver);
3. We intercept and block the touchscreen input from being process by the OS;
4. We activate the auto highlight function to get visual feedback of the item in focus;

```
// initialise touch pattern Recogniser
tpr = CoreController.getActiveTPR();

// register ioReceiver
registerIOReceiver();

// starts monitoring touchscreen
int deviceIndex = CoreController.monitorTouch();

// blocks the touch screen
CoreController.commandIO(CoreController.SET_BLOCK, deviceIndex,
        true);

// sets automatic highlighting
 CoreController.setAutoHighlight(true);
```

Fig. 24 - Initialising the Touch Controller

All that is left is to recognize the touches and react accordingly. In the *onUpdateIO() (*e.g.) we call the TPR and wait until one type of touch is recognized. Since we only want to recognize slides and taps we will use the method *identifyOnRelease()*.

```
@Override
public void onUpdateIO(int device, int type, int code, int value,
        int timestamp) {
    if(tpr==null){
        tpr=CoreController.getActiveTPR();
    }
    int touchType;
    if ((touchType = tpr.identifyOnRelease(type, code, value, timestamp)) != -1) {
        Log.d(LT, "Touch Type" + touchType );
        handleTouch(touchType);
    }
```

Fig. 25- Using the TPR

When a type of touch is recognized we send it to *handleTouch(type)*. This method uses

the most basic functions of the Control Interface (i.e. *navNext(), selectCurrent()*).

```
private void handleTouch(int touchType){
    switch (touchType) {
    case TouchRecognizer.SLIDE:
        navNext();
        break;
    case TouchRecognizer.TOUCHED:
        selectCurrent();
        break;
    case TouchRecognizer.LONGPRESS: // Stops service
        CoreController.stopService();
    }
}
```

Fig. 26 - Handle touch

We just finished creating a new control interface for our smartphone using the SWAT library. This is but one of many of the available SWAT tutorials, they are:

- Observers – since all Observers share a common structure it explains the common characteristics (registering, *onUpdate*, triggering the register mechanism);
- Content Observer - shows how to set the type of content desired and what time of information is retrieved from the *onUpdate* method;
- IO Observer- detailed tutorial on how to monitor the touchscreen, how to select the desired internal device to monitor and how to discover available devices;
- Wi-Fi Control – explains how to set a connection to the server and lists the available commands;
- Touch Adapter – demonstrates how to use SWAT to create a virtual touch device and reroute touches from the default touch device to our created one;

In our GitHub repository all of these tutorials are accompanied by the sample code of each feature. In the attachments of this dissertation you can find all of the tutorials created.

Chapter 4 Multi Impairment Case Study

Users with disabilities face a tough choice when opting for the assistive technology that enables them to control their smartphone. As mentioned before they can either go with a custom-made choice which limits their available applications and features or they can use system-wide accessibility services that can only serve a very specific kind of user, which in most cases are the blind. We came across Miguel, a multi impaired user that has limit control over his mobile device. His case was reported in a previous study [22] in which they provided him with half of the solution to his problem which will discuss in section 4.1.1. To solve this issue we developed a system using SWAT that enables Miguel to control all of the mobile OS. Miguel is a 31 year old that suffered an accident at the age of 21 that left him blind and tetraplegic. The accident also cause a speech impairment that makes Miguel stutter. Unfortunately due to this stutter voice recognition is not an option since his speech patterns are anything but consistent. He is not able to operate a keyboard or any kind of pointer, he only has residual arm and neck movement. This residual arm movement allows Miguel to control a single switch with the proper arm support. With Miguel in mind we created a system that allows him to have more control over his mobile device which results in a tremendous increase in the amount of features Miguel is able to resort to. We developed this system to empower Miguel and his caregivers. Our solution enables his caregivers to have an active role in the adaptation of the system.

4.1 Previous solution: Easy Phone

Miguel current solution is EasyPhone, a custom-made application developed and reported in [22]. This application applies an auditory automatic scanning of the available options, which are: battery, clock, received messages, contacts. Using a mouse has a switch he is able to select the option and navigate the sub-menus until he accomplish his desired task. With it he has easy access to most common phone operations. Yet it was not enough, he wants to be able to do more, he wants to send messages, select a radio station, set an alarm clock, in short Miguel is an individual that strives independence in everything he can. His current solution is not scalable, every time Miguel desires a new feature, this feature as to be built from the ground up. Another issue with this custom-made application is its lack of adaptability to future requirements. As such it is important for the system to

allow personalization to accommodate these changes. Bear in mind that due to his disabilities Miguel has to navigate all option through an automatic auditory scanning, being able to adjust the order of the menus or to create shortcuts can have a tremendous impact in the system usability.

4.2 Our solution: Auto-Nav

Our main goal with our system was to provide Miguel with a way to navigate the Android OS and the applications therein. It was fundamental that he could perform at the very least the same actions that he could with EasyPhone. We developed a system that is able to access onscreen content and control how it is navigated. In this section we report the components of our solution: navigation, notification, call management, filtering and assistive macros mechanisms. Not all Android applications are accessible and our solution relies on the onscreen content descriptions and text to navigate and give feedback. Many market applications do not follow the guidelines to support accessibility services (e.g. buttons without descriptions). Furthermore, some applications do provide descriptions but are too cumbersome for a non-visual scanning interaction. To deal with both these issues, we devised the concept of Assistive Macros, enabling caregivers to record macros for common actions on the smartphone making them available to Miguel. Macros can be a partial solution to these problems. In applications that are not accessible or have steps that are not accessible, a macro can be created to performs these steps for the user.

4.2.1 Navigation

Bearing in mind Miguel's abilities, we adapted interaction to an auditory scanning paradigm (e.g. Fig. 27) (row-column) operated with a single switch.

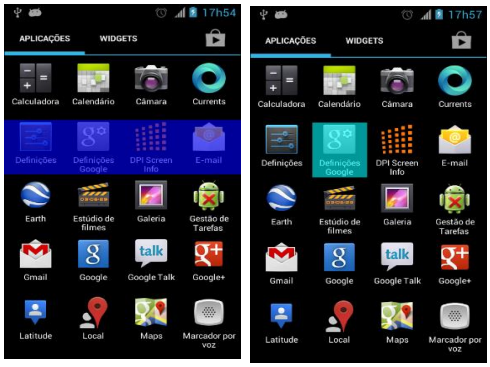

Fig. 27- Auto-Nav system

The auditory scanning when reading a row it reads its first item and “row” right afterwards to identify that row has more items (e.g. “Messages Row). Miguel has then the opportunity to click the switch to select an option in a configurable time period. If a click is made in a row with more than one option then all the options are scanned one by one, if only one item is available it is select on the row click. All clicks produce an auditory feedback to confirm selection. Our solution also handles miss-clicks by ignoring those that are not made within a minimal configurable time interval. At the beginning of every set of options (e.g., a new screen), a back option is presented to allow the user to quickly correct navigation mistakes. This row-columns layouts are created according to the content information retrieved from SWAT, this means the layouts are dependent on each item description, name and bounding boxes. To start navigating Miguel simply has to click the switch, after doing so all available rows are navigated once, then the system enters a standby mode where to start navigating again he has to press the switch again. If enough time has passed in the standby mode when Miguel resumes the navigation the system will first return to the home screen (e.g. starting point for navigation).

4.2.2 Notifications

The system also handles notifications (e.g. messages, missed calls, application notifications) when they are triggered the system stops navigating and reads the content of the notification. If the notification is received when the system is on standby, the next time the switch is pressed the notification is read.

4.2.3 Call management

It was important to simplify answering and terminating calls, if Miguel had to navigate all available rows to answer/terminate a call it would be a hassle and could prevent him from answer in time. As such when a call is received the phone rings and the caller id is read if Miguel presses the switch he answers the call. The system by default sets to speaker mode so Miguel can be heard, to terminate the call Miguel simply has to press the switch again.

4.2.4 Text-Entry

As part of the system we created a similar keyboard to NavTap [11], a keyboard built for non-visual usage. The keyboard is divided in six columns where the alphabet is

indexed by vowel meaning which row starts with a vowel and has all the following letters of the alphabet until the next vowel. The biggest difference to NavTap is exploration. The exploration of the keyboard is done using the same row-column auto scanning technique used in the navigation of system. When an edit text box is pressed our keyboard is enabled and the scanning starts to be only of the keyboard until it is closed. At the beginning of each scan the option to close the keyboard is presented, when select our system resumes the navigation of the on screen content.

4.2.5 Filtering

In an automatic auditory scanning technique where the navigation times are long it can be fundamental to restrict the available options to improve performance. We developed a simple filter that allows Miguel's care givers when starting the system can select which options should be hidden in the home screen. The home screen can be cluttered with undesirable widgets, applications or even default items (e.g. Google search, voice search) filtering out this options in the first navigation screen can drastically improve the experience.

4.2.6 Assistive Macros

We developed the Assistive Macros application to empower both Miguel and his caregivers by allowing him to perform a complex command with only one selection and his caregivers by giving them the ability to actively participate in the optimization of the system The caregivers are able to disable the auto navigation mechanism temporarily in order to create and manage the macros. To create a macro the caregivers had to start the recording macro application and name the macro, then the app would return to the home screen where it starts recording. To record the macro the caregivers had to reproduce the actions they wanted the macro to perform. The macro recording mechanism has two modes, one which records the navigation steps and another which records touches as mention in section 3.3.4 When recording a menu is presented in the top right corner that allows users to change the recording mode. This two modes are needed to overcome the lack of accessibility in some applications. If a button does not possess a description we cannot record a navigation step and reproduce it because the button has no identity, and if a button has no description or name it means the scanning technique will also not be able to reach it. By providing the ability to record touch coordinates we ensure all

applications can be accessed through our macros, assuming the inaccessible options do not change coordinates. To finish recording on the top right side an option is presented to end the process. The macro will then appear in the home screen which allows Miguel to easily and quickly access it. Since the navigation is done accordingly to the position of the icons on the screen the caregivers are able to rearrange the icon order to optimize the scanning. To delete a macro all they had to do is remove the icon from the home screen. Since the macro reproduction is a process and it is not instantaneous it is important to provide the appropriate feedback to let Miguel know the status of the reproduction. When Miguel selects a macro he receives an auditory feedback and when the macro finishes he receives another to inform him that he can start navigating again.

4.3 Leveraging SWAT

Using SWAT as a library enables us to implement key interfaces to create Auto-Nav. We implemented the three Observer interfaces from SWAT (Fig. 28).

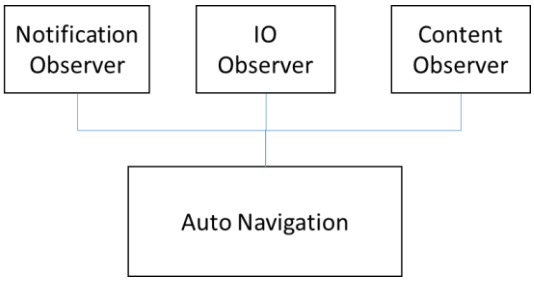

Fig. 28- Auto Nav Implementation

4.3.1 Content

The Content Observer is the vital observer of the system. After implementing the interface and registering the observer every time the on screen content is updated our system receives an update. This update contains a list of all screen content which contains the fundamental information of content name/description, type of content (e.g. scrollable content) and bounding boxes. When this update is receive all the content is process and through analysing the bounding boxes and a row-column layout is generated. After this tree is generated Auto-Nav is then able to navigate it until another update is receive, these updates are primarily triggered by selections that cause the screen to update. The filter discuss in section 4.2.5 is based on this content, when a list of content match the exact list that the filter is supposed to operate upon, the filter is activated and hides the undesired options.

4.3.2 IO input

Through the IO Observer Interface we are able to monitor all screen touches. Doing so we are able to transform the touch screen into a switch, we block all touches from reaching the OS while still monitoring them. Monitoring this touch events enable us to perform actions, in this case the click, when a type of touch event is recognized. We can control and transform multiple taps and complex gestures into a single selecting action. Miguel cannot reliably interact with the touch screen, with it in mind the IO Observer can also listen to touch events that can be injected by the SWAT system. This means we can simulate touches using this interface and allow Miguel to interact with the system with any kind of input device. In Miguel case this turn out to be a mouse that acted like a switch.

4.3.3 External Control

Due to hardware limitations the mouse was not recognized by the smartphone. To enable Miguel to use the mouse in this particular smartphone we resorted to the Wi-Fi Control module from SWAT. Using this module we are able to send touches that are then forward to our IO Observer. We developed a simple java application that established a connection via TCP to our SWAT Wi-Fi server and listened to mouse clicks, when a click was detect it was sent to SWAT. In other words, we connect a standard mouse to a pc connect to our smartphone, whenever a click was detect it was forwarded to SWAT on our smartphone which was interpreted by our IO Observer which acted accordingly.

4.3.4 Notifications

With the Notification Observer we were able to create our system wide notification alert that was discuss in section 3.3.2 To do so we simply register the observer and when an update is received it is process by Auto-Nav system to provide the auditory feedback to Miguel.

4.3.5 Control Interface

Our system extends the Control Interface abstract class mentioned in section 3.3.2 , doing so we are able to perform three fundamental actions with only a single command (e.g. back, home and click node). The click node method allows us to click a node by providing

his description. All this calls are processed by the SWAT library and the appropriate command is executed (e.g. a click node might not be a selection, it might be a scroll action when the node select is the “scroll” node).

4.3.6 SWAT Keyboard

To allow Miguel to write a message we resorted to the SWAT keyboard described in section 0. . When a touch is detect in an edit text box the AutoNavKeyboard which extends the SWAT Keyboard is called. This keyboard has the layout described in section 4.4.1 4.2.4 , to accommodate Miguel needs it has a built in automatic scanning system equal to the one provided for the core system. For the keyboard to react to the commands in the same way the system does it also implements the IO Receiver. When the keyboard is active the touches are ignored by the elementary module of the Auto-Nav system and are only processed by it. Implementing the navigation module of the keyboard separate to all other layout navigations allow us to personalize and adapt the navigation independently of the rest of the system (e.g. scanning speed, navigation mechanisms between rows).

4.4 Case Study

This system uses most of what SWAT has to offer, in this next section we report the case study of Miguel using Auto-Nav. Even though Miguel strives for independence he is cautious about new approaches for two reasons, he is afraid to lose the same level of comfort of his previous solution and to not be able to use new solutions. Knowingly we started by ensuring Miguel that what we had for him to try was a prototype and not an immediate replacement for EasyPhone. Only when the system is proven stable and with his approval will we change his current permanent solution.

4.4.1 Procedure

We had the opportunity to test our system with Miguel (e.g. Fig. 29) across two sessions, one of about one hour and a second one of about one and a half hour. After each session we discussed with Miguel and his caregivers their experience and thoughts on the system. We recorded the audio of both sessions to collect the reactions and comments to the system. Miguel is already very familiar with EasyPhone as such we used it extensively as a comparison to our system to be easier to understand.

Firstly we explained to him the thought process behind our new system and how it was different from EasyPhone and what that meant. We explained that EasyPhone was tailored specifically for him and what he was accessing were menus that did not existed in the wild, in any other application or phone. We talk about the consequences of such design process and how adding new features presented a difficult task.

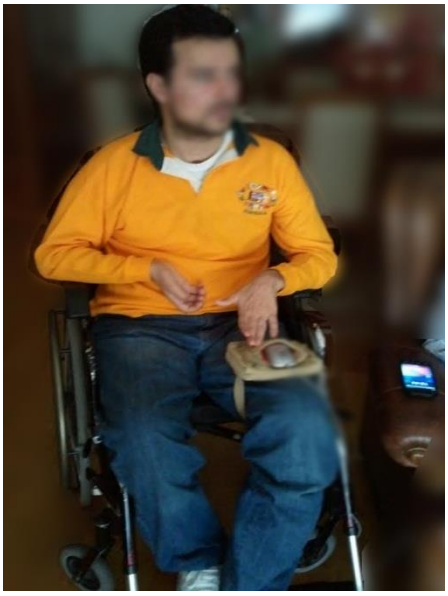

Fig. 29 - Miguel interacting with a smartphone with a mouse controller working as a single switch.

After establishing the limits of his current system we introduced him to ours. It was imperative that Miguel understood that with our system we would actually be using applications that anyone can use with their smartphone. This was important for two reasons:

- Making sure Miguel understood that some navigation limitations/problems are cause not by our system but by how each of these applications layouts are designed and that our system only allows him to freely navigate them;
- To ensure Miguel realizes the potential of allowing him to use applications that were not designed specifically for him, now he could use almost any application available;

We explained how the selection method was the same but now he actually had to navigate in a row-column basis. Every task we asked Miguel to do had a brief explanation of the steps he had to take to accomplish them, after that we let Miguel freely navigate the system while guiding him, in each window, by telling him which was the option that would allow him to perform his goal.

4.4.2 Tasks

In the first session we focused on tasks that he was already able to perform with his custom made solution or that were very similar:

- Navigate the home screen (e.g. similar to the navigation of the menu of EasyPhone);
- Select an option and perform a back operation;
- Navigating contacts;
- Making/receiving phone calls;
- Reading messages;

In the second session we wanted to observe if Miguel was able to take advantage of the new features of the system that he asked for:

- Write a message “ola como estas” (e.g. hello how are you);
- Use the radio application;
- Create a macro for the radio application;
- Use a macro;

4.4.3 First Session

In the first session, we focused on making Miguel familiar with the system. We asked Miguel to try the system and navigate the home screen where the layout only contained one tem per column in four of the five rows as shown in Fig. 30 (e.g. [Back], [15:30], [Mensag], [aContacts], [Applications], [Google Search, Voice Search]). His first task was to simply select one of the items on the home screen and then select the back option. Miguel was able to do with no difficulties.

Fig. 30 - Home screen

The next one was to perform a call. To do so Miguel had to select the "aContacts" application and had to do four selections to complete the task. He had to select the last row that contained the option from "A/Z" which opened a menu with four rows of letters order alphabetically. Afterwards he had to select the letter that corresponded to the first letter of the person he was asked to call, then a menu with all contacts that start by that letter opened were he could select the contact. After opening the desire contact information, when the number of the contact was read he could press to perform the call, which he did. He was able to speak with the person and then terminate the call. Miguel showed no difficulties in the navigation mechanism except for the need to know beforehand where the option he was looking for was. Since it was Miguel first experience with the menu of this application we had to specify in which row the option was available. It was noticeable and even suggested by him that he could benefit from some label adjustments: since Miguel was navigating ordinary applications some labels were abbreviated (e.g. Favs meaning favourites, the first option of the last row of the aContacts application).

We proceeded with the tasks that are triggered not by Miguel but by a third person (e.g. receiving calls/messages). Both these tasks had a brief explanation of what happens in each case and both of them presented no issue for him.

He then asked how he could consult the message again and we explained the steps required to do so. When he reached the message the last option read on that screen was "write message", he immediately select the option and smiled. Miguel was anxious to try writing messages, so in our first session we introduced him to the layout of our keyboard and asked him to write a simple "hello how are you". He was able to write the message and was thrilled with the possibility to send messages. However we noticed some difficulties in grasping the layout of the keyboard he consistently asked in which row the desired letter was.

In the post session comments he showed great interest in the ability to control more of the system, he added "*EasyPhone is a simpler system but with more limitations and I can adapt fast to this new one*". He referred again the desire to adapt some labels in order to adapt faster and that the keyboard navigation could be easier if the layout was different.

4.4.4 Second Session

In our second session, we asked him to write the same message but now we explained the steps needed to send the message. Now more agile with the system Miguel took less time and assistance to accomplish the task. He showed again some adversity against the keyboard layout.

This second session greatly involved his caregivers in order to test the creation and reproduction of Assistive Macros. Has discuss in section 4.2 some applications can be inaccessible even through our system to Miguel. Such a case is the radio application installed in the device which was a special request from Miguel. This particular application allowed Miguel to navigate between different stations and select the one he wanted, he could start listening to the station but he could not stop. Getting to the stations could also represent a challenge since it is a multiple steps process through multiple menus that have loading times that deteriorate the navigation experience. For this reasons we choose this application as a target for our Assistive Macros proof of concept.

We asked Miguel's mother to now have an active role in the study. Firstly we explained her how she could pause the system in order to interact with the smartphone has a normal user (e.g. using the volume down key to stop it and again to resume it). Since she had no previous experience with touchscreens we briefly explained how to interact with the smartphone (e.g. slide gesture, taps). We guided her step by step on how to reach the Assistive Macro application. Upon reaching it we explained to her and to Miguel that she was creating a shortcut that he could take in order to accomplish an action that required multiple complex steps or even steps that he could not take with the system. First she named the macro radio and then we guided her on the steps to reach the radio application and select the list of stations. She then finished the recording and went back to the home screen.

Miguel task was to use the shortcut to get to the radio station list, which he did successfully. After selecting the station curiosity took over and Miguel explored the radio application menus. We explained what each menu meant has he navigated through then and that one key feature was missing, the stop action.

We pre-recorded the stop macro of the radio application and we had it available on the second home screen. To illustrate how Miguel's caregivers could optimize his

navigation we asked his mother to pause the system and rearrange the home screen menus. To do so we explained how she could move icons on the home screen (e.g. long press) and that she could rearrange everything and even delete options (e.g. we had a dummy icon which we asked her to delete). After rearranging the icons Miguel had one row on the home screen with the radio and the stop option. He then proceeded to stop the radio station.

Both Miguel and his caregivers showed interest in this concept to such ends as writing a pre-recorded message, or go to a set of contacts (i.e. contacts that start with a "t"). Reinforcing the positive feeling about the macros, the mother said "*We never know what we need until we try it*". She showed great interest in experimenting with the application "After learning the basics we start to feel comfortable to explore new things" referring to how she would try to create different macros for different applications.

4.5 Conclusion and Future directions

One possible endeavour the Assistive Macro application might face is the lack of confidence Miguel has in the ability of his caregivers to handle the smartphone, "*I do not know if I will let you mess with the system*". To this end we have to make sure Miguel's caregivers feel comfortable and confident in using the system while ensuring the system is stable. Improving the recording and management of macros is one step in the right direction to improve trust. Due to the nature of navigating ordinary applications a problem we discovered is the ambiguity of some names and description that make the system have a higher learning curve. To overcome this issue we can develop a "Tag Adapter". Similar to how the Assistive Macro process we can create an application that Miguel caregivers can access and select existing tags (e.g. tags/names/description) that were previously navigated and attribute a corresponding name. So the next time Miguel was faced with the option "Favs" the system would read "Favourites".

The lack of descriptions is one of the big problems in accessibility. Recurring to Assistive Macros we involved his caregivers in the optimization and usability of the system. With it we overcame barriers that not a single accessibility service can overcome, the lack of respect for the accessibility guidelines by most applications. This system was only made possible by using the SWAT library, it used most of its key features to allow system-wide access. With this we have made the first steps in proving the SWAT

capabilities and range of applications. Even though this system is focused around Miguel's needs SWAT is an agnostic library that can be used to create system wide assistive technologies that are adapted to any kind of disability. In this case study we have shown how SWAT can be used to create an system wide assistive technology to enable a multi-impaired user to have full access to his smartphone by giving him the tools to interact with ordinary smartphone applications.

Chapter 5 SWAT Screen Reader

Touch screens are now common technology in our everyday life, from tablets and smartphones to public kiosks. It is therefore fundamental to understand how touch screen-based interfaces can be used by people with visual impairments. When interacting with touch screen interfaces, sighted users have the ability to quickly perceive what on screen options are available and where is there location. A visually impaired user has to go through all options using a screen reader to understand what is available. Due to the lack of tactile cues it can be hard to understand where the options are spatially, users have to rely on accessibility features such as Explore by Touch from TalkBack. As expected this is a time consuming process that requires significantly more time for non-sighted users.

Touch screens have become more powerful over the years and now allow up to 10 touch points. Unfortunately the top accessibility features for the blind (e.g. VoiceOver and TalkBack) only take advantage of its potential in the form of pre determine gestures that can be mapped to specific actions. It gets worse when we realize that when these services are active, users are restricted to use single touch input. If a user wants to write a message quickly and presses two keys simultaneously only one is read through the system and the other touch point is simply ignored. We are creating barriers and disabling devices capabilities instead of taking advantage of their potential, essentially we are crippling the device use for no benefit.

The use of touch screen is usually a visually intensive task, when we rely on accessibility services like TalkBack to translate these interfaces for blind users they need a good spatial ability and/or cognitive capabilities in order to memorize items locations [24]. This is especially true when users are presented with on screen keyboards. The lack of tactile feedback makes multi touch text-entry on a virtual keyboard a challenge [1], the problem is exacerbated when users are visually impaired. The system developed enables a visually impaired user to explore the screen using a similar technique to Touch to Explore (e.g. dragging the finger across the screen while the content is read aloud) but with multi touch and simultaneous spatialized auditory feedback. In this chapter we will discuss the system developed using SWAT and a study conducted where visually impaired users were asked to write using a single and a multi touch technique that took advantage of our system on a Tablet device.

5.1 SWAT Reader

To explore the notion of simultaneous spatialized auditory feedback we developed an Android system recurring to SWAT. The result is a prototype bi-touch screen touch reader similar to TouchToExplore with multi touch and spatial capabilities. Even though our system was developed for all Android devices our focus is on Tablets devices where the spatial recognition problem is amplified in comparison with the smartphone.

To spatialized the sound we relied on a sound framework [15] available on pc. In order to test our solution the system connected to the sound framework (that was running on a pc) via TCP/IP. Since the auditory feedback has a focus on the spatial position of the content the system is intended to be used with headphones. The system would send the desired text to be read aloud and what was its desire spatial position (i.e. the spatial position was define by angle and distance to the listener). This framework allow us to spatialized pre-recorded sound in real time, as such all the text read by our system was previously generated (e.g. all the letters, home screen options).

Our system is only a prototype of what an actual multi touch screen reader should be. We focused on creating a system that would allow us to explore a multitude of input/output solutions and perform a laboratory study. In order for the users to perceived simultaneous auditory feedback it was crucial to distinguish between several audio sources. In [4] we can verify that users can benefit from listening to different frequencies and from different positions.

With this in mind there was an important issue to address: how to spatialize the auditory feedback to translate the position of the content with simultaneous feedback? Burdened by this question we explored 5 different modes:

- **2 Way** – we divide the screen in two region (e.g. left and right), when the content is read its position determines if he either is spatialized to the left of the listener or to the right;
- **3 Way** – same solution as above with one more region (e.g. front) where the content in the middle of the screen is spatialized in front of the listener;
- **Auto Column** – by analysing the content we divide it into columns, the number of columns varies with the perceived number of columns the interface displays. We then divide the number of columns by the 180 degrees in front

of the listener and then each text is then is read accordingly to their position/column on the screen;

- **Set Column** – we divide the screen into a set x amount of columns, the content is then attributed one accordingly to their position on the screen. It has a sub-mode for testing purposes where we can manually assign text descriptions to a specific column;
- **Finger to Ear** – the idea behind this mode is to associate a finger to a particular region (e.g. left hand finger always read aloud to the left of the listener). The implemented version of this mode gives the first pointer is read to the right of the listener and the second one is read to the left;

In all the modes we only use the spatial position 180 degrees in front of the user, since the user is interacting with the device in front of him there was no reason to spatially use the 180 degrees on the back. To further differentiate between audio sources we used two voices in every mode one male and other female where the male voice displays low frequency levels and the female the opposite.

The voices could be used in a multitude of ways but we choose to use them to distinguish between contact points. In every mode the first point is read with a male voice, from thereon while the finger is not lifted all the content is read with it. The second contact point is read with a female voice, from thereon all content read with that finger is read in it. In short the first finger on screen is read in male voice and the second one is female if the first one is still on screen. All modes support multi-touch interaction but due to the cognitive load generated by simultaneous auditory feedback we restricted the system to a two pointers interaction. In all of the modes there were 3 available selection methods:

- On Up – upon lifting the finger the user selects the content that was last read with that contact point;
- Double Tap – when a finger is lifted the content last read with it is focused, by double tapping close to the focused item a selection is made. Take notice that since we are using a bi-touch approach two items can be focused at the same time, therefore the distance to the focused item is taken into account, it can also discard the double tap if the distance to the focused item is to great;
- Split Tap – to select an item the user rest a finger on it and taps in the vicinity using a second/third finger. The second/third tap must be performed closest to the desired selection target;

Every selection mode shares the same property of overlooking the physical bounds of the items. When any of the selection actions are performed the last read item is activated even if the pointer is already off the target, making selection easier and not restricted to bounding boxes. If the user drags the finger of the device in the On Up selection no action is executed. Since we have an experimental system the confirmation feedback of all selections is simply a beep. In every mode when the user enters the bounding box of an item that item is read aloud, if the user rests the finger on the content after the option has been read if he performs a move type action the content is reread. This makes it easier for users to reread any option without losing the content in focus. Since we provide simultaneous auditory feedback we wanted to ensure no the information did not overlap to unperceived gibberish. Using only two simultaneous audio sources associated with each independent pointer we ensure at most only two sounds are being played. As an example if the user drags a finger across the screen through two options and he passes through the second one without the first one being completely read the system will stop the first one and start reading the second one. The only simultaneous audio sources the system allows is one from each contact point.

We used a part of the system developed to conduct a study on single and multi-touch text entry input on a Tablet device. In it we focused on the Set Column mode with only the On Up selection available. To perform the study we tooled our system specifically for it, several parameters and features were added that will be described in the next sections.

5.1.1 Limitations

Currently the system relies on an available pc sound framework, as such the system is not truly mobile since we are relying on the pc output to provide feedback. We pre-recorded all the available items in our tests with the desired voice, as such the system is not a fully functional screen reader it is only a testing prototype. To create a functional screen reader we would have to create a Text-To-Speech that would be able to spatialize the given text and have multiple voice options, but such was not the purpose of this endeavour.

If we look at the first two modes presented they are very similar and only provide users with a distinction between left/right or left/front/right which in most cases will not

provide the depth necessary to be a relevant cue since it will only serves to help localize our finger in relation with the screen.

The Auto Column method unlike the previous two is able to distinguish between items with an undetermined set of columns that are translated to degrees. By setting the number of columns automatically the difference between two adjacent columns will get diluted when the number of columns rises which can lead to weaker spatial cues since no difference between adjacent columns is perceived.

The Set Columns resolves this issue by pre defining the number of columns available. Both column methods have to deal with the issue of skewed content, for example a keyboard does not have perfectly align columns, which can lead to an unnatural column division.

All of the modes above rely heavily on the position of the content, this presents an issue when the content presented is large enough to occupy multiple regions of the define space, in our system we use the centre point of the content to attribute it to a column/region.

The Finger To Ear does not rely on the position of the content but it neglects to give any spatial cues to its position, it serves only the purpose of facilitating the distinction between what is read by which of our touches. Another issue with this method is that ideally every time I use a finger from my left hand it should always be attributed to the same region, unfortunately today smartphones are not able to distinguish between fingers or hands as such we have to rely on the methodology of the first contact is the right/left one.

This is an issue that is presented in every mode in a different feature, the different voices (e.g. male/female) should not be attributed to the first/second pointer they should be attributed to which finger the user is currently using. Unfortunately the limitations of the device do not allow us to create this correspondence to easily distinguish between touches. One possible solution would be to attribute the voice accordingly with which side of the screen is first touched (i.e. if the user touched to the left side of the screen it was probably the left hand finger and as such it should be the woman/man voice until lifted) but this raises its own issues when the user presses with his right hand finger to the far left of the device. There is no proven correct approach and has such we adopted the first contact is a male voice, second contact it is a woman's.

5.1.2 Leveraging SWAT

Using SWAT as a library enable us to create our prototype screen reader system wide. Our system (SWAT Reader) implements SWAT IO Observer and extends the Control Interface (Fig. 31).

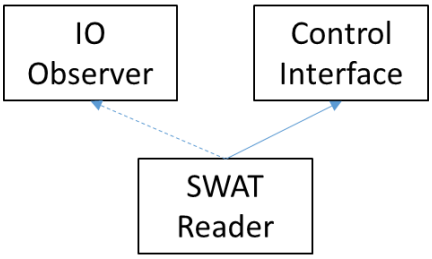

Fig. 31 - SWAT Reader implementation

The IO Observer is the fundamental feature of SWAT that allow us to create a screen reader. After implementing the interface and registering the observer we select which device we want to monitor, in this case the touchscreen. Since we want to modify how the user interacts with the device through SWAT we block all touches from being interpret by the OS. Since we are reading touches directly from the device we have to process the low-level events to translate them into touch types. SWAT possesses two Touch Recognizers able to process different types of multi-touch protocols[20]. These Recognizers are able to process the events and categorized them in one of two ways: down/move/up or tap/longPress/slide.

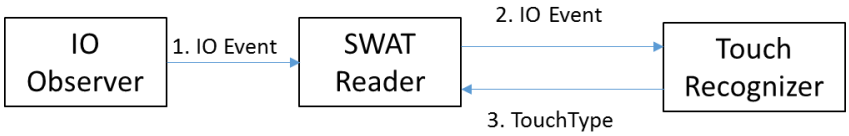

Fig. 32 – Touch recognition process

In SWAT Reader we are interested in using the traditional categorization of down/move/up. By selecting the appropriate Recognizer in the SWAT accessibility services preferences activity we can invoke him in our system. When we receive a touch event we forward it to the active Touch Recognizer from SWAT that returns us 4 possible touch types results (Fig. 32), down/move/up or undefined. We are receiving events one at a time and each event is composed by type, code, value and timestamp, for example the event with code 53 it's the code for the x coordinate so if the value is 100 that is the

20 https://www.kernel.org/doc/Documentation/input/multi-touch-protocol.txt, Last vistited 03/09/2014

x value. When processing these events only when a synchronization event code is received does the system recognizes a touch type, until them all previous calls to the recognizer return undefined since the touch is not fully defined yet. When a touch is recognized our Reader processes it accordingly. If the touch recognized is a Down or Move and it is the first or second contact point our system asks SWAT what is underneath those screen coordinates; SWAT answers and the system forwards the text to the Sound Framework with the appropriate play parameters calculated from the activated feedback mode. All the Move and Down touches change the focused item, our system keeps track of two focused items at any time from the last two independent contact points. When an Up action is performed the system checks which of the two focused items is closer to the Up coordinates and if it respects the distance threshold the item is selected.

By extending the SWAT Control Interface we are able to select items by their text description, making selection a one line command. The system was developed to explore multi-touch screen reader and perform a laboratory study, particular in the text-entry case study it was necessary to log all possible information. The relevant information to log was all IO Events in conjunction to what was read/wrote when touch types were recognized. To do so we relied to SWAT fundamental logging mechanism (Log IO) and logged custom messages about the relevant information (e.g. “0 d 355 534 9597 1400749930582”, “Wrote:r”).

The SWAT Reader is only made possible through the use of the SWAT library, by blocking the OS from processing the touches we are able to process then and react to them accordingly making it possible to create a multi-touch system-wide screen reader prototype. It relies heavily on the IO Observer and takes advantage of the Logging and Control Interface capabilities. In the next section we report a study performed using the system on a text-entry task.

5.2 Study – Bi-Manual text entry evaluation

Touch based devices present users with a great opportunities and Tablet devices are no different. Touchscreen devices constant evolution amplify the interaction possibilities, thus increasing the visual demands imposed to their users. This results in limited access and slower content exploration, as well as greater difficulties to memorize and understand the screen spatial layout.

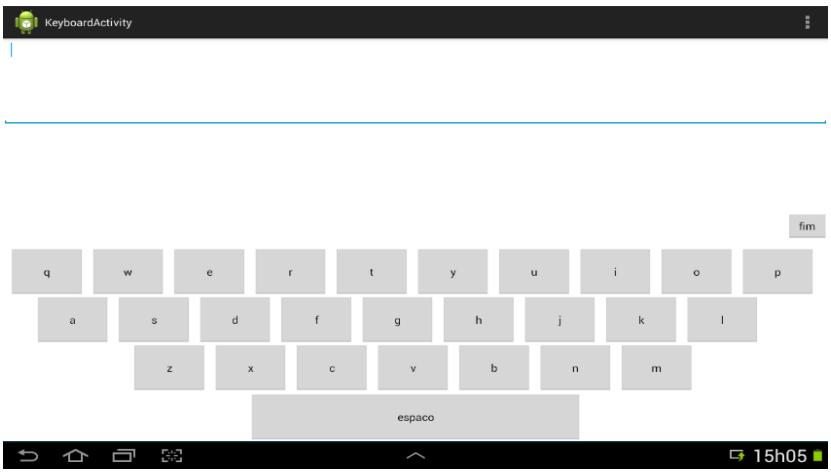

Fig. 33 - Text entry application

Text-entry stands not only as one of the most common tasks performed with these devices, but it is also one of the most visually demanding. Standard QWERTY keyboards remain the dominant method and the one provided by mainstream accessibility solutions. However, previous studies have shown that text-entry in smartphones is still slow compared to what sighted people experience [24]. In contrast, there is little knowledge about blind people's usage of QWERTY keyboards in tablet devices. Actually, their dimensions are more similar to a physical QWERTY keyboard, which may empower a similar use and therefore enable users to transfer their previous knowledge. We propose a solution that takes advantage of bi-manual interaction in a Tablet device resembling the use of a physical QWERTY keyboard. We set out to investigate if given similar capabilities to a physical keyboard would the users have a bi-manual interaction and would we observe a performance improvement. To do so we used the SWAT Reader and created a test application that suited our needs.

The application (e.g. Fig. 33) was composed by a text entry box and 27 buttons that represented each letter of the alphabet plus the space bar. The buttons were positioned in the same position the default virtual keyboard displays them. This application also

handled the experiment logs (e.g. target phrase, transcribed phrase, time). A java application was created to control the flow of the experience and was connected with the application via TCP/IP. We added to SWAT Reader a single touch mode that initially used the double tap selection similar to VoiceOver text-entry. On this mode only the male voice is presented and the buttons do not have a spatialized auditory feedback.

5.2.1 Approach

We conducted a preliminary study with five users to examine the audio feedback, and target selection methods of the bi-manual system. In our first design the selection method available was Double Tap. However, we consistently observed that character selection by releasing the finger on the key to be the assumed and preferred solution by the participants, therefore we opted to support this method exclusively. Releasing the finger to write is the default writing mode on the Android TalkBack service and it is available also on the iOS VoiceOver under the flag "Expert Mode".

During our preliminary study users struggled with repositioning the finger since lifting it would write a character. To tackle the issue after receiving audio feedback for a character, users have 2 seconds to insert that character by lifting their finger. This allows users to have both fingers on the screen and lift any of them off to reposition quickly without inserting an undesired letter. To prevent users from inserting characters in error from a quick tap on a key, we introduced a 200ms delay between insertions. These threshold values were reached empirically through analysis of the interaction log, where we observed multiple miss inputs of this nature by users with no previous touchscreen experience, which rendered the phrase unreadable.

The QWERTY keyboard does not have the keys perfectly align in a column row fashion, they are skewed. The first row has 10 columns while the second one has 9 and the last one 7 if we only count the 26 letters of the alphabet. Since we wished to give the users a good grasp of where they were on the keyboard it was fundamental to give more than a 3 way division of the space. To conduct our study we used the auditory feedback mode Set Column. Using the Set Column we defined the number of columns to 10 and its using sub-mode we manually attributed each letter to the desired column. When asked to drag the finger across a row from left to right users noticed the difference between them and how the sound was shifting. We pre-recorded all the letters and the space bar sounds

in both a high frequency voice (woman) and a low frequency one (male). In the single touch only the male voice was used.

5.2.2 Evaluation

With the preliminary study completed and our parameters sorted into their final form we set out to evaluate visually impaired users performance with the bi-manual approach on a tablet device using the single text-entry method as a basis for comparison. To our knowledge there are no comprehensive studies on blind user's performance on a QWERTY text-entry method on tablet devices. Nor any attempt with simultaneous audio feedback with multi-touch input has been reported. In detail, we aim to answer the following research questions:

- How a visually impaired user performs using a QWERTY text-entry method on a Tablet device?
- Do users take advantage of the bi-manual input?
- Can multi-touch input with simultaneous audio feedback improve user performance?

5.2.3 Participants

We conducted the study with 39 blind participants (with light perception at most) recruited mainly from a formation centre for the visually impaired. Of the 39 participants only 30 were taken into account for this study. From the other 9 showed severe difficulties interacting with a touch device and were not able to write a readable string of their name with either method after 10 minutes of practice.

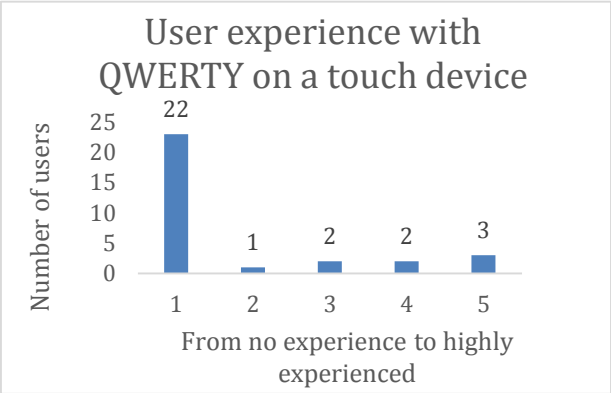

Fig. 34 - User experience with QWERTY text entry method on a touch device

The 30 participants group was composed of 6 females and 24 males, with ages ranging from 22 to 65 (M=45, SD=11).

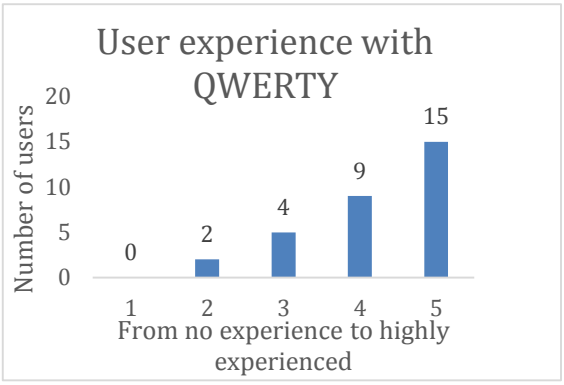

Fig. 35- User experience with QWERTY text-entry method

All participants completed a background questionnaire and self-assessed their experience with the following technologies: QWERTY text-entry on touch devices (1:No Experience to 5:Highly Experienced), five (16%) graded themselves with experience above three; 23 (74%) reported no previous experience with touchscreen devices(e.g. Fig. 34); and 15 (48%) regarded themselves as highly experienced with QWERTY text-entry on physical keyboards(e.g. Fig. 35).

5.2.4 Apparatus

We used a Samsung Galaxy Tab2 with a multi-touch 10.1 capacitive touchscreen. The device was fixed to a table and connected to a laptop computer via WI-FI connection to relay user interactions, and enable the research to monitor and control the evaluation stimuli. Ozone Onda ST over-the-ear headphones were used to provide audio feedback to the participants, running the spatialized auditory feedback sound framework previously discussed.

5.2.5 Procedure

The study comprised one session that was conducted within a training centre for visually-impaired people, to evaluate with two text-entry interface conditions. The session took approximately 45 minutes and included: an oral questionnaire about demographic data and touchscreen proficiency and a text-entry task. Moreover, we explained that the purpose of the study was to understand and test two approaches for text-entry in Tablet devices. In the evaluation session, the participants tried one of two

different conditions: one of the currently used methods and our Multi Touch (Bi-Manual) approach. Participants were assigned randomly to one of the conditions. After which they were given a 10 minute practice with the system, here the researcher explained how the condition behaved. The participants were encouraged to perform a simple text-entry task such as typing their name, this allowed them to become familiar with touchscreen interactions, the systems auditory confirmation of characters and key selection. During this period the user was guided through several tasks:

- Find the letter “A”;
- Find the letter “L”;
- Write the letter “A”;
- Write any letter;
- Perform a space;
- Write your name;

These tasks were common to both methods, the letters the participants were asked to write and find changed from subject to subject. In the Bi-Manual method the user’s started by performing the tasks with 1 finger for the first 3 tasks after which they were instructed to use both hands to perform the rest. In this method we also asked users to drag their finger across a row of the keyboard in order to understand the spatialization of sound that was taken place. We explained how the simultaneous (including voice differences) and spatialized feedback works.

During the actual text-entry evaluation the participants completed five trials, the order of these were again counterbalanced across the participants. All of the phrases were comprised of five words, with an average word size of 4.48 characters. These phrases were gathered from a written language corpus, having at least a 0.97 correlation with the language. For each trial the researcher would read aloud the stimulus phrase, then the participant would repeat the phrase back to the researcher to ensure they understood what was said. The participants were instructed to write the sentences as fast and as accurate as they could. In the Bi-Manual method while we emphasised the use of simultaneous contact the participant was not obliged to do so. The participant would then attempt to type the phrase using the current interface condition, error correction was disabled during these trials to ensure that all errors and typing behaviours were captured. Once the participant finished all five trials they were asked to complete an oral questionnaire to obtain subjective opinions regarding the interface condition.

5.2.6 Design and Analysis

This study used a 30 between-subjects design with one independent variable: interface condition. Phrases were randomized for each interface condition and the participants completed all trials: 5 phrases x 1 interface condition x 30 participants = 150 trials. We analyse the users' performance in terms of both text-entry speed (words per minute - WPM) and accuracy (MSD Error Rate). We used a keyboard matrix based on the Manhattan distances between keys; in which we account for keyboard row offsets. Using this matrix we calculate the precision of the user precision (e.g. how far from his final target did he land in each insertion). We applied Shapiro-Wilk normality test to observed values in all dependent values. Parametric test were applied for normally-distributed variables and non-parametric test otherwise.

5.2.7 Results

The goal of this study was to understand user performance using a QWERTY keyboard on a tablet device with a bi-manual method. We start by analysing the different performance metrics of each method.

Performance

Fig. 36 shows the WPM scores per condition. A Mann-Whitney U test revealed that users did not wrote significantly more WPM with the Bi-Manual (M = 3.00, SD=1.17) interface than in the Uni-Manual (M= 2.38, SD 0.82) one (U = 75.0, z = -1.25, p = .124).

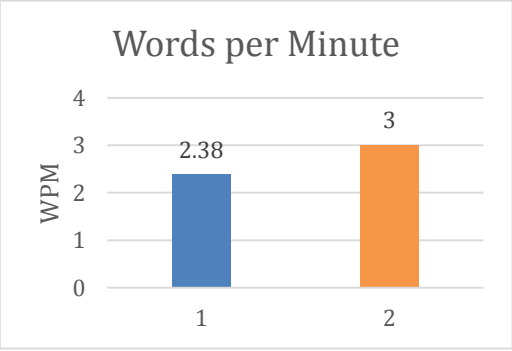

Fig. 36 - Words per Minute

This suggests that touchscreen novice users are able to take advantage of two simultaneous interaction points and match the results of a Uni-Manual interaction.

In Fig. 37 we can observe the evolution of the means of the WPM across the 5 phrases where we can notice a slight edge in the Multi touch method across all phrases. The difference between methods in the first phrase showed not to be statistically significant. We found the precision to be significantly higher (U=61.00, z=-2.12, p=0.034) in the Bi-Manual (M=1.59, SD=0.98) interface than the Uni-Manual (M=2.29, SD=0.98) when we performed a Mann-Whitney U test.

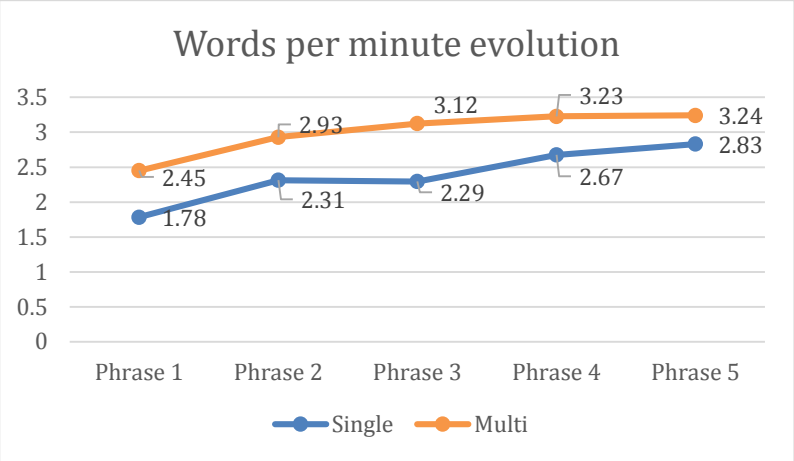

Fig. 37- Words per minute evolution

A Mann-Whitney U test showed that there were no differences in the MSD Error Rate (e.g. Fig. 38) between Uni-Manual (M=9.95, SD=6.54) and Bi-Manual (M =11.52, SD=3.90) conditions (U = 74.0, z = -1.58, p = .114). This result highlights the fact that Bi-Manual interface input comes at no cost regarding the error rate.

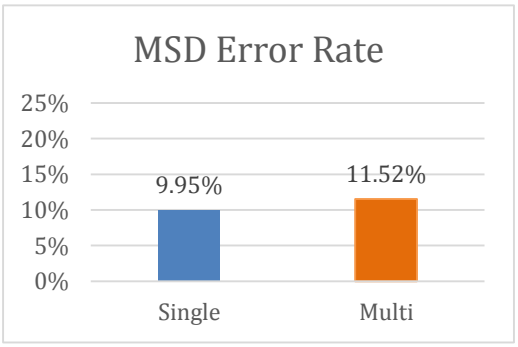

Fig. 38 - MSD Error Rate

Users needed to travel between fewer keys with the Bi-manual condition (M=2.36, SD=0.66) than with the Uni-manual (M=2.97, SD=0.96) one (U = 62.00, z = -2.079, p = .038). This result suggests that users were able to leverage their two-handed interaction in physical QWERTY keyboards, which gave them a greater understanding of spatial key locations.

Strategies

Fig. 39 represents the average time the user spent in the multi touch method with one or two contact points on screen. The users spent an average of 503 (M=503.37, SD= 231.89) seconds using one contact point and 128 using 2 (M=127.72, SD=135.42), ranging from users that spent 5 seconds using 2 contact points to 661 seconds.

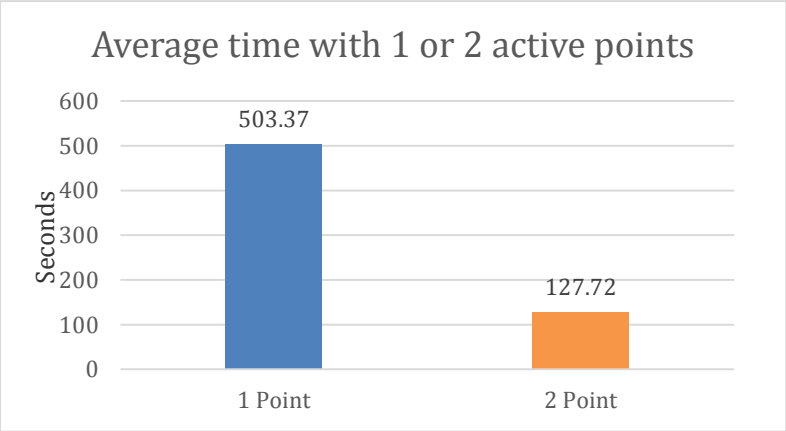

Fig. 39 - Average time with 1 or 2 active points

Not every user took advantage of the Multi Touch method, some spent 5 seconds of their session with 2 contact points while others spent 10 minutes.

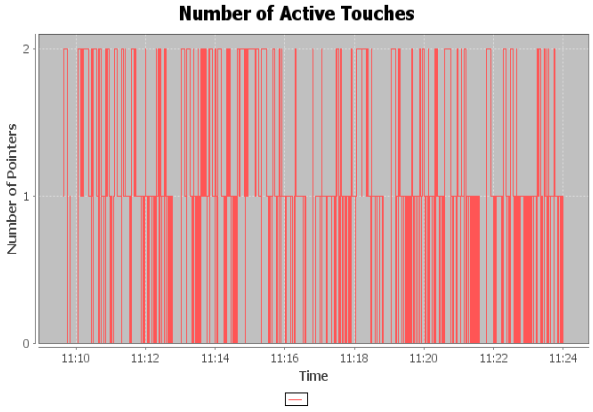

Fig. 40 - Number of active touches in the multi touch session

There are clearly two types of user in the Multi Touch session as we can see from Fig. 40 and Fig. 41 where we can see the number of contact points that are active across the session. The first one shows a user that took advantage of the multiple touches while the second one only used one contact point at a time. Different participants used different techniques when it came to the Multi Touch session.

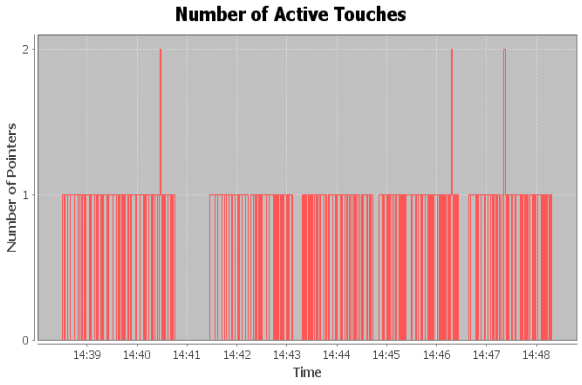

Fig. 41 - Number of active touches in a multi touch session

If we look at Fig. 42 and Fig. 43 we are seeing all the contacts performed that triggered a reading character, in blue we see the second contact point and in red a single contact point.

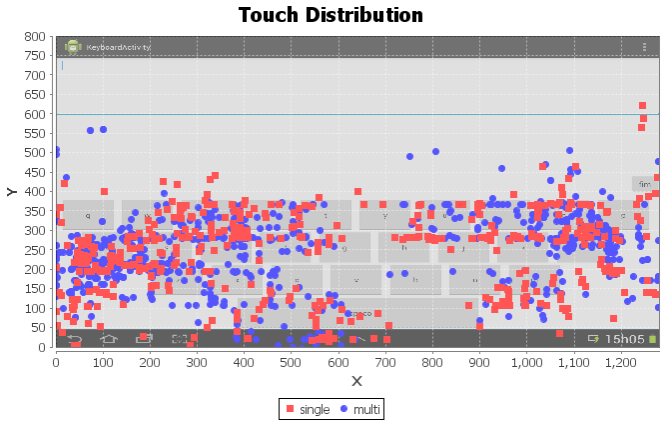

Fig. 42 - Touch distribution 1

The user from Fig. 42 clearly used both contact points anywhere on the screen, when searching for a character he would start from the either the left or right side of the screen but never the centre (e.g. if he did we would notice a significant larger grouping of points in the centre of the screen).

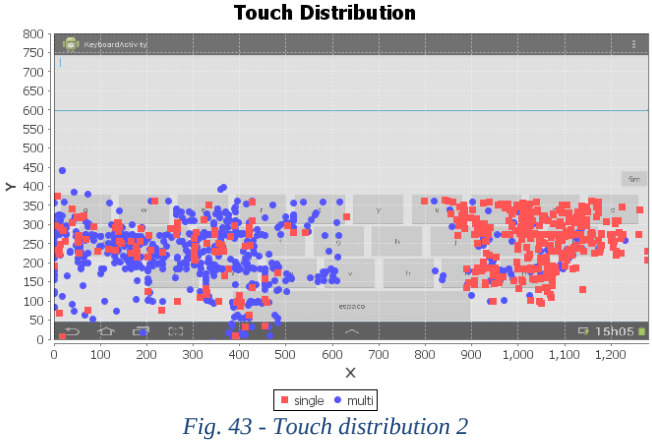

Fig. 43 - Touch distribution 2

The user Fig. 43 had a completely different approach. He used a single contact point whenever he wanted to write a letter from the right side, when he wanted to write a letter from the left side he used 2 contact points.

Opinions

After each session we performed a brief questionnaire. This questionnaire was composed of two statements to classify using a seven-point Likert scale.

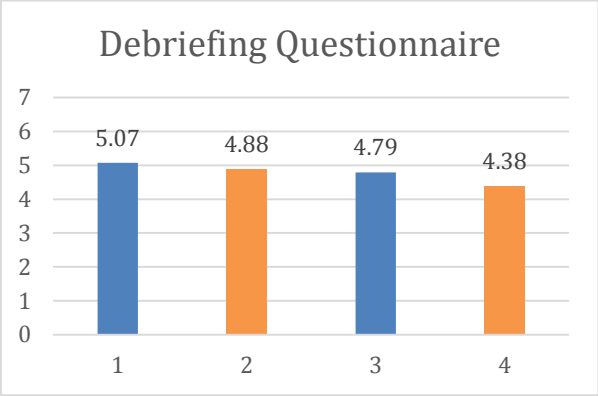

Fig. 44 - Debriefing questionnaire

The first statement asked participants to grade how easy it was to use the method from 1- very hard to 7-very easy. The second asked how fast they thought the method was from 1- very slow to 7-very fast. In Fig. 44 we can see there is little discrepancy between methods. Since the data did not follow a normal distribution we applied a Mann-Whitney U test. . No significant different was found in the easiness statement ($U = 119.00$, $z = -.301$, $p = .791$) nor with the quickness one ($U = 129.00$, $z = -.729$, $p = .498$).

There seems to be a precision advantage in writing using the Multi Touch method even when it did not translate in a WPM significant difference. One possible reason is as a participant commented "*I can use the second finger to localize myself on the keyboard, like a physical cue*" this individual started by finding the "f" and "j" of the keyboard before he started writing (e.g. the "f" and "j" possess physical cues on a physical keyboard to help blind users positioning) or as another said "*one point is a guide while the other writes"*. Another pointed out that he could "*rest his finger on screen"* instead of being obliged to never touch the screen with the second finger.

5.2.8 Discussion

Our first study goal was achieved, we report blind user's performance using a QWERTY text-entry method on a Tablet device. Participants achieved a mean of 2.4 WPM with the Single method and a 9.95% MSD Error Rate. There is no speed or accuracy difference between the two methods, even though multi has a 3 WPM and an 11.5% MSD Error Rate average. However, there is a precision significant discrepancy between the two, the Multi Touch approach is more precise with a lower value of M=1.59 to the M=2.29 of the Uni-Manual interface. From Fig. 37 we observe an improvement across the phrases with a slight edge in favour of the Multi Touch method that proved to not be statistically significant. To further understand the influence of this factor we analysed the number of words read. Again a significant difference was found.

These results lead us to postulate that users that use the Multi Touch method have a better grasp of the keyboard and perform a more ideal navigation traveling through less characters. We found that some users take advantage of a two contact point solution, solidifying the need for a deeper personalization of screen readers. We argue that even if the second contact point is not used to perform character insertion it provides a cue to the keyboard layout and overhaul size. The second contact point in conjunction with the spatialized auditory feedback provide users with a deeper understanding of their surroundings.

5.3 Conclusion and Future Directions

There is no reason why multi touch text-entry with screen readers is not allowed. We are creating restrictions where only options should exist. If a user prefers a method over the other he should just be able to enable/disable as he pleases. In this study we explored text-entry with a QWERTY keyboard and simultaneous spatialized auditory feedback. Although we explored a QWERTY input method there are other possible text-entry techniques where the multi touch with multiple audio sources might be successfully implemented. We focused on text-entry but multiple input with simultaneous audio feedback can be applied to many other tasks and we wish to do so in the future. The work on the simultaneous audio sources techniques is not complete, and although we decided to use the set column method there are other models that can be implemented and their exploration is another possible venue to proceed this line of investigation.

From a developer stand point the OS heavily restricts the control over input/output as we have mentioned before, SWAT overcomes those restrictions and allow for the creation and exploration of solutions such as the one presented in this study. Using SWAT we can proceed with this work and explore ideas for a spatialized screen reader, more specifically, we can create a system to explore text and not options. The development of this assistive technology would not be possible without recurring to the SWAT library. In this chapter we shown another application of SWAT in a completely different scenario of the assistive technology field, proving its usefulness and range of application.

Chapter 6 Conclusions

Mobile devices have become a crucial tool in our everyday lives. There has been a shift from the dated feature phones to the powerful smartphones even among the disabled community. It is therefore fundamental to provide users with the proper tools to interact with this and other mobile devices like a tablet. Unfortunately mobile OS make a lot of assumptions about their end user and do a poor job supporting the creating of assistive technology by third party developers. We cannot deny the effort that has been made in the latest years by OS developers to include out of the box with accessibility features. They are a great step in the right direction but it is not enough.

Adaptability and extensibility are the key to provide a multitude of different users with a usable interface. Current OS do not fully support this endeavour due to their restrictions on the control developers can achieve over the system core input and output parameters.

In this work we presented the current interaction techniques of assistive technologies and state of mobile assistive technologies applications, ranging from the custom made to the pre-built in. We identified the issues of the current OS approach and defined a clear objective of what our system aimed to provide. SWAT was created to provide more control over the system output and input mechanisms. More specifically we wanted to access to on screen content and the view hierarchy and we wished to be able to control touch events system wide.

SWAT was developed and accomplished all of its predefined goals. We developed an extensible and adaptable library that allowed the creation of two very different assistive technologies. The SWAT library was presented and carefully described in detail in Chapter 3.

In Chapter 4 we presented the first assistive technology created and the inspiration of this thesis. The case study of Miguel a multi-impairment individual that strives for independence and wants to be able to control as much about his life as he possible can. The new system made only possible by SWAT allowed Miguel to quickly use new functionalities without the need to custom develop a suitable application. On top of what was accomplished by a previous custom made application Miguel is now able to send messages and select a radio station using applications available on the market to everyone.

These applications are not adapted to Miguel and without SWAT Miguel would not be able to interact with them. SWAT enables him to use default Android applications and even overcome some of their accessibility problems using the Macro system of SWAT. Even though only the radio and the messaging application were tested SWAT gives control of the full system to Miguel, he is now able to navigate all the menus of the Android OS. In this chapter we present the system developed using SWAT, we describe a pre-built application of SWAT the Assistive Macro application and its applications in Miguel's case. A case study is described that consisted of two sessions with Miguel. This Chapter was a step in the right direction to substantiate the adaptability and extensibility of the library. This is not a closed chapter, we intend to continue our work to provide Miguel with a final form of the SWAT system to allow a stable use that has the proper tools to handle the accessibility problems that some applications present.

In Chapter 5 we present the second assistive technology developed using SWAT. This one was not intended as a final product but as an exploratory system of simultaneous auditory spatialized feedback in touch devices. This system could eventually become a multi touch screen reader. First we described the system enumerating all of its functionalities. With it we performed a study on the QWERTY text-entry method on a tablet device. This study showed that a multi-touch text-entry with simultaneous spatialized auditory feedback can have an impact in the user performance. More specifically a first time user will benefit from using a multi touch method, the user will grasp the layout of the keyboard quicker leading to a better performance by optimizing navigation. The study is presented in great detail, for this thesis purposes we showed another application of SWAT, this time in the realm of screen readers and text-entry. In this chapter we shown SWAT allows a faster development of exploratory systems. The developer bypasses the logging mechanism and allows SWAT to manage it speeding the development process. The study was perform in a laboratory setting and as such the application used was a custom-developed one, but nothing prevents SWAT Reader from being used system wide.

Throughout this work we have shown the systems we developed made only possible by SWAT and presented then each in a study. Although we believe SWAT to be a fundamental tool to report the deficiencies of the current OS's and what can be accomplished when we bypass their restraints, we believe mobile OS's should give developers a deeper control over the system to allow the creation of such technologies.

SWAT had an extreme focus on assistives technologies (as the name implies) but the reality is that SWAT has other venues where it can contribute.

6.1 Other Applications

SWAT was created with the intention to be used by others. We developed has a library and to support a development community to take advantage of it we created a few tutorials to introduce a new developer to it. To date, SWAT has been used in two very distinct areas by other developers and others have also shown interest in it. A testing system was explored by one developer using SWAT to perform scripts similar to monkeyrunner[21]. The intent of the system was to perform a set of actions and measure the approximate time it takes to load an application. The project is currently on hold due to the accessibility issues the target application presented.

One collaboration resulted in the creation of a system by the name AURIC. This system uses SWAT touch monitoring capabilities to record user interactions. More specifically AURIC detects if an intruder has unblocked your smartphone and starts recording the interaction taking place by taking screenshots of your interactions and snapshots using the front camera while recording all the touches. The smartphone owner can then check the AURIC application and see the intrusion, the system reproduces it with all the touches and screenshots in conjunction with the snapshots taken with the front camera. Furthermore a wizard-of-Oz mode was developed using SWAT Wifi to allow the control of the detection of intrusion via Wifi for user testing purposes.

6.2 Limitations

SWAT relies on a rooted Android device with an API level of 14 or above. SWAT is still in its infancy and as such it can have unidentified issues. Since the library overrides the system drivers when we monitor//block/inject touches we have to be aware of the touch screen capabilities and protocol. Since different devices possess different touch screens and consequently different drivers this can present an issue on the scalability of SWAT. We currently have tested we over 5 different types of devices and came across two different methods, unfortunately all the devices were Samsung. Although this is

[21] http://developer.android.com/tools/help/monkeyrunner_concepts.html

currently a limitation we can certainly adapt to new models as we need. The system heavily relies on the accessibility services consequently we rely on the respect for the accessibility norms the applications have. Unfortunately many applications bluntly ignore accessibility standards and good practices, we have developed tools to overcome some of this barriers but not all of them. Rooting a smartphone compromises some of its built in security as such this poses a risk, ideally SWAT as proven to OS's developers the need to provide this control and is now up to then to do so without endangering the end user.

6.3 Future Prospects

SWAT is still in its infancy and as shown over the course of this project to be a promising library with applications in many different fields. SWAT can be used to create a variety of assistive technologies, we only explored two on this thesis but we intend to pursue this subject. The two system presented using SWAT are definitely not on their final form, further development will be done to accommodate all Miguel's needs in the system present in Chapter 4 and new venues will be explored with the one in Chapter 5. Another possible prospect is the detection of touch patterns, the automation of actions and the detection/adaptation/correction of miss inputs. The library is currently public domain and can be used by anyone. We hope to create a developer community that relies on our tool to further progress the research in the different fields.

Chapter 7 Bibliography

[1] Bachl, Stefan, et al. "Challenges for designing the user experience of multi-touch interfaces." *Proc. Workshop on Engineering Patterns for Multi-Touch Interfaces*. 2010.

[2] Bergman, Eric, and Earl Johnson. "Towards accessible human-computer interaction." *Advances in human-computer interaction* 5.1 (1995).

[3] Bigham, Jeffrey P., Jeremy T. Brudvik, and Bernie Zhang. "Accessibility by demonstration: enabling end users to guide developers to web accessibility solutions." *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 2010.

[4] Cherry, E. C. 1953. Some experiments on the recognition of speech, with one and with two ears. The Journal of the acoustical society of America, 25(5), 975-979

[5] Dai, Liwei, et al. "Speech-based cursor control: a study of grid-based solutions." *ACM SIGACCESS Accessibility and Computing*. No. 77-78. ACM, 2004.

[6] Findlater, Leah, and Jacob Wobbrock. "Personalized input: improving ten-finger touchscreen typing through automatic adaptation." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012.

[7] Gajos, Krzysztof Z., Jacob O. Wobbrock, and Daniel S. Weld. "Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces." *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 2008.

[8] Gajos, Krzysztof, and Daniel S. Weld. "SUPPLE: automatically generating user interfaces." Proceedings of the 9th international conference on Intelligent user interfaces. ACM, 2004.

[9] Guerreiro, T., Nicolau, H., Jorge, J. and Gonçalves, D.. 2010. Assessing mobile touch interfaces for tetraplegics. In Proceedings of the 12th international conference on Human computer interaction with mobile devices and services (MobileHCI '10). ACM, New York, NY, USA, 31-34.

[10] Guerreiro, Tiago, and Joaquim Jorge. "Assistive technologies for spinal cord injured individuals: Electromyographic mobile accessibility." Proceedings of GW 2007, 7th International Workshop on Gesture in Human-Computer Interaction and Simulation. 2007.

[11] Guerreiro, Tiago, et al. "NavTap: a long term study with excluded blind users."*Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 2009.

[12] Hwang, Faustina, et al. "Mouse movements of motion-impaired users: a submovement analysis." *ACM SIGACCESS Accessibility and Computing*. No. 77-78. ACM, 2004.

[13] Igarashi, Takeo, and John F. Hughes. "Voice as sound: using non-verbal voice input for interactive control." *Proceedings of the 14th annual ACM symposium on User interface software and technology*. ACM, 2001.

[14] Jacob, Robert JK. "What is the next generation of human-computer interaction?." *CHI'06 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2006.

[15] João Guerreiro and Daniel Gonçalves. Text-to-Speeches: Evaluating the Perception of Concurrent Speech by Blind People. In Proceedings of the 16th International ACM SIGACCESS Conference on Computers and Accessibility. Rochester, USA, 2014

[16] Keates, Simeon, et al. "User models and user physical capability." *User Modeling and User-Adapted Interaction* 12.2-3 (2002): 139-169.

[17] Kim, Yoojin, et al. "Surveying the Accessibility of Mobile Touchscreen Games for Persons with Motor Impairments: A Preliminary Analysis."

[18] Manaris, Bill Z., Valanne MacGyvers, and Michail G. Lagoudakis. "Universal Access to Mobile Computing Devices through Speech Input." *FLAIRS Conference*. 1999.

[19] Manaris, Bill, Valanne Macgyvers, and Michail Lagoudakis. "A Listening Keyboard for Users with Motor Impairments—A Usability Study." *International Journal of Speech Technology* 5.4 (2002): 371-388.

[20] Montague, Kyle, Vicki L. Hanson, and Andy Cobley. "Designing for individuals: usable touch-screen interaction through shared user models." *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 2012.

[21] Myers, Brad A., et al. "Using handhelds to help people with motor impairments." Proceedings of the fifth international ACM conference on Assistive technologies. ACM, 2002.

[22] Nicolau, H., Guerreiro, J., & Guerreiro, T. (2014). Stressing the Boundaries of Mobile Accessibility. arXiv preprint arXiv:1402.1001.

[23] Ntoa, S., Margetis, G., Antona, M., & Stephanidis, C. Scanning-Based Interaction Techniques for Motor Impaired Users.

[24] Oliveira, João, et al. "Blind people and mobile touch-based text-entry: acknowledging the need for different flavors." The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility. ACM, 2011.

[25] Oviatt, Sharon, et al. "Designing the user interface for multimodal speech and pen-based gesture applications: state-of-the-art systems and future research directions." *Human-computer interaction* 15.4 (2000): 263-322.

[26] Oviatt, Sharon. "Multimodal interactive maps: Designing for human performance." Human-Computer Interaction 12.1 (1997): 93-129.[27] Ryall, Kathy, et al. "Experiences with and observations of direct-touch tabletops." Horizontal Interactive Human-Computer Systems, 2006. TableTop 2006. First IEEE International Workshop on. IEEE, 2006.

[28] Sears, Andrew, et al. "When computers fade: Pervasive computing and situationally-induced impairments and disabilities." HCI International. Vol. 2. No. 03. 2003.

[29] Wu, Ting-Fang, et al. "Computer access assessment for persons with physical disabilities: A guide to assistive technology interventions." *Computers helping people with special needs*. Springer Berlin Heidelberg, 2002. 204-211.

[30] Yu Zhong, T. V. Raman, Casey Burkhardt, Fadi Biadsy, and Jeffrey P. Bigham. 2014. JustSpeak: enabling universal voice control on Android. In Proceedings of the 11th Web for All Conference (W4A '14). ACM, New York, NY, USA

Appendix – Tutorials

For the code of every tutorial please visit us on github https://github.com/AndreFPRodrigues/SWAT_Lib

7.1 Adapt Tutorial

```
Using SWAT developers are able to monitor/block/inject events into the system
internal devices. In this tutorial we will show how you can create a virtual
touch device. We will in real-time block and adapt the touches sent to the
touchscreen device.
To do so we:
1. Create a class that implements the IOReceiver interface
2. Extend the broadCastReceiver class and check for the SWAT init intent
(check tutorial on IOReceiver)
3. Start monitoring the touchscreen
4. Block the touchscreen
5. Create the virtual drive and select the protocol*
6. On the onUpdateIO method, create the desired adaptation and call

You can try the tutorial at tutorials.adapt.TouchAdapter.
*You have to use the same protocol the default driver of your touchscreen is
using. To do so print the messages received in the onUpdateIO and figure out
to which of the protocols it corresponds. Bellow there is a link to the
protocols documentation.
https://www.kernel.org/doc/Documentation/input/multi-touch-protocol.txt
This are some of the codes that will be received in the onUpdateIO for the
touchscreen:

    To check the possible types and codes of all events go to Mswat/jni/input.h
```

7.2 Content Tutorial

Content receiver provides information about the current screen contents in the form of an ArrayList<Node>. Each node has a describable node of the screen and each node possesses the following information:

- Bounds in parent and in screen
- Class and package name
- Description and Text
- Actions
- Parent
- Children

An example of a content receiver is available in tutorials.receivers.

7.3 IOReceiver Interface Tutorial

This tutorial shows how you can use the IOReceiver interface.
IOReceiver interface is meant to allow users to get updates on the desired monitored internal device (touchscreen , keypad, ...)
To receive the updates you need to:
1. Create a class that implements the mswat.interface.IOReceiver
2. Implement the interface methods
   a. registerIOReceiver – this method is meant to register the IOReceiver in the system. To do so use

   b. onUpdateIO(device, type, code, value, timestamp) – this method will receive all the IO updates from the system. Where device is the id of the monitor device and the rest are the raw values provided from the device.
   c. onTouchReceived(type) – implement this method if you want to allow control over wifi. It receives the type of touches available to send over wifi. (Touch types are defined in mswat.touch.TouchRecognizer)

3. When you start the registering process is up to you; In this example we will start when the mswat finish the calibration process. Mswat broadcasts a init signal when it finish the calibration/initialisation process
   a. To do so start by extending the class BroadCastReceiver
   b. Implement the class method and check for the mswat init signal
   c. When the signal is received register the IOReceiver
   d. Now make sure you register the broadcastReceiver in your Android manifest
4. Now you will receive all the IO updates of all the monitored devices.

By default no devices are being monitored as such you will also need to start the monitoring process when you register the IOReceiver. To do so you have 2 options:
   Call CoreController.monitorTouch() after registering the IO receiver, if the purpose is to monitor the touchscreen
   Manually select the device to monitor

To manually select the device SWAT provides two key methods
   CoreController.getDevices() – returns a String array of the devices
   CoreController.commandIO(command, index, state) – using the command CoreController.MONITOR_DEV we can select the desired monitor device by giving its index and desired monitoring state (true/false).

In this short example we select the device that contains keypad in its name to be monitored.

You can try this example at tutorials/IOReceiverTutorial.

7.4 Receivers Tutorial

There are three types of receivers in SWAT:

Content receiver – receives updates of the current content of the screen;

Notification receiver – receives all notifications;

IO receiver – allows you to monitor any system internal device;

These interfaces are available in mswat.interfaces. These interfaces share a common structure.
To receive updates you need to:
1. Create a class that implements one of the interfaces in mswat.interface
2. Implement the interface methods
a. register[type]Receiver– this method is meant to register the receiver in the system. To do so use *"CoreController.register[type](this);"* as shown in the figure bellow;

b. onUpdate[type] – this method will receive all the updates from the system. Depending on the type of receiver.

7.5 Wi-fi Control Tutorial

Through Wifi Control we can use all the control interfaces methods plus simulate types of touches.
To do so:
1. Check Wi-Fi control in the accessibility service configuration
2. Create a TCP connection through port 6000
3. Send message in the format [type],[arg0]

The supported messages are:

```
navNext
navPrev
select
focus,[index]
touch,[type of touch*]
autoHighlight
home
back
clickAt,[description]
```

*The touch method is used to allow developers to send types of touches to their SWAT controllers that do not require coordinates, only types of touches like the example given in the tutorial TouchController. The types of touches are:
0 – down
1 – move
2 – up
3 – double click
4 – split tap
5 – tap
6 – slide
7 – long press

3. When you start the registering process is up to you; In this example we will start when the mswat finish the calibration process. Mswat broadcasts a init signal when it finish the calibration/initialisation process
a. To do so start by extending the class BroadCastReceiver
b. Implement the class method and check for the mswat init signal
c. When the signal is received register the receiver as shown in the example
d. Now make sure you register the broadcastReceiver in your Android manifest
4. Now you will receive all the updates

We provide you with an example of each receiver in the tutorials package. Also each receiver has a correspondent pdf with some additional information specific to the receiver.