Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Systémová klávesnice pro OS Android se zaměřením na přístupnost

Diplomová práce

Autor: Bc. Tomáš Klapal

Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Brno srpen 2017

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Filipa Malého Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

V Brně dne 15. srpna 2017 Tomáš Klapal

Poděkování

Na tomto místě bych rád poděkoval panu doc. Ing. Filipu Malému Ph.D. za jeho vstřícnost a čas při vedení diplomové práce. Dále panu Ing. Zoltánu Zemkovi za zapůjčení testovacího zařízení a panu Mgr. Radku Pavlíčkovi s panem Bc. Matějem Plchem ze střediska Teiresiás za jejich pomoc při vytváření zadání a uvedení do světa nevidomých uživatelů mobilních zařízení.

Anotace

Tato práce se zabývá možnostmi zadávání uživatelského vstupu do moderních digitálních telekomunikačních zařízení pro zrakově hendikepované osoby. Současně popisuje vlastní způsob řešení této problematiky – implementaci aplikace Sense Keyboard. Sense Keyboard, neboli softwarová klávesnice pro intuitivní zadávání textu, je aplikace určená pro platformu android. Jejím hlavním námětem je snaha o usnadnění zadávání textů na zařízeních s dotykovými displeji nevidomým a slabozrakým. Současná řešení ne zcela pokrývají potřeby nevidomých a zadávání textu standardními klávesnicemi s využitím našeptávače je velmi pomalé. Ve spolupráci se střediskem pro pomoc studentům se specifickými nároky Teiresiás vznikla myšlenka staronového, intuitivního způsobu zadávání textu.

Annotation

Title: Software keyboard for OS Android focusing on accessibility

This diploma thesis deals with the possibilities of entering user input into mobile devices for visually impaired persons. At the same time, it describes implementation of possible solution - Sense Keyboard. The Sense Keyboard, or a software keyboard for intuitive text input, is an application for OS android. Its main purpose is to make it easier to enter input text on touch screen devices for the blind and partially sighted people. Current solutions do not entirely cover the needs of the blind people. Text input using standard keyboards supported by TalkBack is very slow. In collaboration with the Center of Assistance - Teiresiás, the idea of an old-fashioned intuitive text input method was created.

Klíčová slova

Android, přístupnost, klávesnice, aplikace

Keywords

Android, accessibility, keyboard, application

Obsah

Kapitola 1

Úvod

V dnešní době zaměřené na moderní technologie je mobilní komunikační zařízení nedílnou součástí všedního života téměř každého jedince. Rozvoj technologií umožnil již před několika lety značnou redukci rozměrů a nárůst výpočetního výkonu. V tuto chvíli se nejmodernější mobilní telefony velikostí RAM pamětí, počtem jader CPU a jejich frekvencí, přítomností dedikovaných grafických čipů, displejů s vysokým rozlišením a dotykovým ovládáním či další výbavou plně vyrovnají slabším sestavám osobních počítačů. To z nich dělá při jejich kapesních rozměrech a dostupnosti velkého množství aplikací velmi užitečné pomocníky pro každodenní použití.

Díky rapidnímu rozvoji mobilních technologií a velkým benefitům, které tato moderní zařízení oproti starším verzím mobilních telefonů s hardwarovými klávesnicemi nabízejí [1], pronikají tyto i do světa zrakově hendikepovaných osob. Jejich možnost manipulace ze zařízeními je však limitována a současná situace dostatečně neuspokojuje jejich potřeby. Dotykové ovládání díky nutnosti vizuální interakce uživatele se zařízením tvoří v tuto chvíli největší barieru pro nevidomé. Výrobci a vývojáři se snaží tyto překážky alespoň částečně eliminovat a operační systémy mobilních telefonů tak vybavují integrovanými verzemi odečítačů displeje a úpravou způsobu ovládání těchto zařízení. Například operační systém Android má integrovanou aplikaci TalkBack – odečítač displeje. Ten dokáže svému uživateli pomocí emulátoru řeči zprostředkovat nejet ovládání telefonu nebo manipulaci s aplikací pro objednání jízdenek, ale třeba i obrázek na webové stránce, pokud je tento k tomu uzpůsoben.

Zatímco pro datový tok ze zařízení směrem k uživateli již v celku použitelné techniky existují a je jen na vývojářích aplikací nebo kodérech webových stránek, aby je plně využívali, tak v opačném směru (zadávání informací uživatelem do zařízení) je situace o poznání horší. Přenosné braillské řádky jsou stále poměrně drahé, zadávaná pomocí hlasu zase vli-

[1] Moderní telefony s HW klávesnicí na trhu existují (zejména od výrobce Blackberry), ovšem jedná se o SW klávesnici typu QWERTY

vem okolního šumu stále ne příliš přesné (zejména pro některé národní jazyky). Mobilní zařízení jsou tedy odkázány na svá vlastní řešení. Tím je v případě systému Android integrovaná softwarová QWERTY klávesnice, která je pro nevidomé uživatele odečítána jíž zmíněným TalkBack-em na základě přejíždění prstem přes jednotlivé znaky v řádcích (díky funkcionalitě nazývané Explore by Touch).

Existuje rovněž několik profesionálních systémových klávesnic třetích stran. Většina těch zdařilejších je však placená a zatím co běžnému uživateli přinášejí tyto klávesnice některé příjemné funkce pro usnadnění psaní, nevidomým uživatelům příliš výhod oproti nativní androidí softwarové klávesnici neposkytují, viz SwiftKey a Google Keyboard (kapitoly 4.2 a 4.3). Některé z renomovaných klávesnic dokonce do nedávna neposkytovaly podporu pro odečítače vůbec. Jiní vývojáři zase nabízejí kompletní řešení správy obsahu a ovládání mobilního telefonu pro nevidomé, tzv. Launchery (např. BlindShell). I zde se však metody zadávání textu obvykle namísto inovativního řešení omezuje pouze na rozložení tzv. QWERTY klávesnice. To má jednu zásadní nevýhodu a totiž tu, že je pro nevidomé nesmírně pomalé. Uživatel má totiž k dispozici až 10 tlačítek na řádku, ve 4 řadách těsně vedle sebe a musí postupným přejížděním prstem přes display hledat, kde se nachází tlačítko se znakem, který chce právě zadat.

Výše zmíněný současný stav byl hlavní motivací pro vytvoření vlastního řešení – nové systémové softwarové klávesnice, která by poskytovala uživatelům větší komfort a zejména pak umožňovala rychlejší zadávání znaků, než je tomu doposud.

Kapitola 2

Zrakově hendikepovaní a moderní informační technologie

Tato kapitola pojednává o problému zrakového hendikepu v souvislosti s každodenním používáním moderních výpočetních technologií. V podkapitole 2.1 jsou rozebrány stávající možnosti předávání informací od zařízení směrem k uživateli, zatím co podkapitola 2.2 se věnuje směru opačnému. Na závěr jsou v podkapitole 2.3 zmíněna slabá místa současných řešení. V textu se často objevuje sousloví "elektronické" čí "výpočetní" zařízení, případně jiná synonyma. Rozumíme tím zde zejména osobní počítače a mobilní zařízení jako tablety, smartphony a další.

2.1 Výstupní komunikace

Výměna informací mezi uživatelem a moderními výpočetními zařízeními probíhá prostřednictvím komunikačních kanálů. Většina těchto zařízení využívá jako hlavní výstupní komunikační kanál některou z forem grafické reprezentace zobrazované skutečnosti. Tento fakt má své opodstatnění, protože průměrný člověk vnímá zhruba 80% všech informací o svém okolí právě zrakem. Zrakový smysl je tedy velice vhodný i pro získávání dat z výpočetních zařízení.

Lidé s těžkými poruchami zraku a nevidomí nebyli díky absenci tohoto smyslu schopni standardními prostředky s výpočetními zařízeními pracovat. S rozvojem moderních technologií a jejich postupnou integrací do běžného života vzrůstala potřeba překonání této bariery a zpřístupnění výpočetní techniky v pokud možno co nejvyšší kvalitě i lidem se zrakovým postižením. Do většiny operačních systémů osobních počítačů, tabletů i smartphonů začal být proto postupem času integrován například program umožňující odečítání obrazovky a hlasový syntetizér.

Odečítač obrazovky dokáže alespoň textovou část zobrazované informace, případně pří-

tomnost některých prvků (u internetových stránek například vstupní pole, checkboxy, formuláře, obrázky a jiné) převést na informaci zvukovou. Hlasový syntetizér je pak tzv. TTS[1] nástroj, tedy program umožňující převod psaného textu do lidem srozumitelných zvuků imitujících lidskou řeč pomocí hlasové syntézy.

Dalším, doposud nezmíněným, avšak přínosným krokem v tomto směru bylo pak vytvoření Braillského řádku, jinak též nazývaného hmatový zobrazovač. Jak již jeho název napovídá, cílem tohoto zařízení je zprostředkovat uživateli výstupní informace pomocí hmatu. Konkrétně je využito reliéfní podoby Braillova bodového písma, obvykle v osmibodové podobě. Ta je použita z důvodu snazší reprezentace v digitálních zařízeních pomocí jednoho bytu (= 8 bitů). Braillský řádek se k výpočetnímu zařízení připojuje v závislosti na typu buď pomocí standardních konektorů, nebo bezdrátově prostřednictvím bluetooth. Některé přenosné varianty tohoto řádku umožňují jeho použití i s mobilními zařízeními ve venkovním prostředí, kdy je řádek zavěšen na popruhu na krku uživatele a ten jej může ovládat i ve stoje. Uživatel je s ním díky sadě navigačních tlačítek schopen také celé mobilní zařízení ovládat. Přenosnou variantu staršího modelu Braillského řádku, jehož vzhled ani funkcionalita se však do současnosti příliš nezměnila, je možné vidět na obrázku 2.1.

Ačkoliv je možné každou z těchto technologií používat samostatně, nejlepších výsledků dosahují při jejich použití současně.

2.2 Vstupní komunikace

Předchozí podkapitola byla věnována datovému toku od zařízení směrem k uživateli a nyní bude popsán směr opačný. Výstupní zařízení umožňují uživateli pasivní přijímání informací, ovšem výpočetní technika obvykle vyžaduje nějakou formu interakce s uživatelem. Ta bývala do nedávna realizována nejčastěji formou hardwarové klávesnice ve vlastním zařízení buď přímo integrované, nebo připojené před podporované rozhraní. V posledních letech je ovšem stále častěji k vidění trend dotykového ovládání a v případě zadávání textu tzv. softwarové/systémové klávesnice.

Zatím, co původní hardwarová zařízení dávala i uživatelům s vadou zraku určité možnosti (například na počítačové klávesnici se dá psát "z paměti"), v případě softwarové klávesnice je situace daleko komplikovanější. Softwarová klávesnice pracuje na principu zobrazení sady znaků (písmen, číslic, interpunkčních znamének atd.) na výstupním zobrazovacím zařízení. Uživatel je tedy nucen dotknout se prstem přesně určitého místa na displeji, aby byl výstupem požadovaný znak. Vzhledem k počtu znaků zobrazovaných softwarovou klávesnicí a velikosti displeje například u smartphonů (běžně 4-5,5"), může být velikost tlačítek, reprezentujících jednotlivé znaky, velmi malá. Pro člověka s těžší zrakovou

[1] TTS - Text To Speech

Obrázek 2.1: Přenosná varianta Braillského řádku vhodná pro mobilní zařízení [21]

vadou je toto ovládání tudíž nevhodné.

Z tohoto důvodu vzniklo několik podpůrných technologií. Jednou z nich je již v předchozí podkapitole zmíněný Braillský řádek. Ten může být, kromě možnosti zobrazování textu a navigace v operačním systému připojeného zařízení, vybaven také sadou dalších tlačítek dovolujících zadávání znaků uživatelem. Na obrázku 2.1 je to horní řada černých kruhových tlačítek a využívají se ve stylu obouručního Pichtova psacího stroje.

Rovněž druhý přístup vychází z již dříve zmíněné podpory znakového výstupu, konkrétně tedy odečítače. K němu jsme výše uvedli, že kromě předčítání textových polí umožňuje také identifikaci některých základních prvků. Pokud jsou tyto prvky vhodně doplněny například speciálními komentáři, může odečítač poskytnout uživateli detailní popis každého takového prvku zvlášť. Informuje jej pak například, že jeho prst či kurzor myši se právě nachází nad písmenem "L" na zobrazené softwarové klávesnici. V konečném důsledku tedy umožňuje uživateli i tento typ klávesnice využívat.

Zatím, co obě předchozí uvedené technologie zadávání vstupu jsou využívány i při datovém výstupu, třetí v dnešní době používaná metoda využívá lidského hlasu. Ten je zachycen mikrofonem elektronického zařízení a následně je specializovaným programem převeden na

text. Případně je takto možné detekovat i speciální povely, kterými lze v zařízení vyvolat určité předem definované akce. Rozpoznáváním řeči se již několik let zabývají přední světové firmy (Google, IBM, Microsoft, ...) či řada technických univerzit (v České republice třeba Fakulta informatiky při VUT v Brně). Díky tomu dosahuje tento přístup v poslední době stále lepších výsledků.

2.3 Nevýhody současných řešení

Předchozí podkapitoly se věnovaly současným možnostem řešení otázek vstupně/výstupní komunikace z pohledu zrakově hendikepovaného uživatele. Nyní je však nutné zmínit, že každý z výše uvedených způsobů interakce s elektronickým zařízením má svá úskalí. Braillský řádek by se mohl zdát být v rámci možností ideální variantou, neboť v sobě spojuje možnosti čtení, psaní i ovládání připojeného zařízení. Ovšem ceny takovýchto zařízení se v současné chvíli na trhu pohybují v rozmezí cca 70.000,- až 150.000,-, některé přenosné bezdrátové varianty je pak možné získat v cenové kategorii okolo 40.000,-. Uživatel se tak ve venkovním prostředí vystavuje nebezpečí odcizení, případně poškození takového zařízení.

Softwarová klávesnice, tedy program do vlastního elektronického zařízení, je oproti tomu buď zcela zdarma, nebo se jeho ceny pohybují v rádu do několika set korun v případě některých specializovaných aplikací. Uživateli tedy odpadá nutnost nosit zároveň dvě samostatná zařízení, přičemž jedno z nich je velmi drahé. Pokud je jako obsluhované elektronické zařízení zvolen smartphone, objeví se však hned několik negativ. Uživatel je bez Braillského řádku odkázán pouze na hlasovou odezvu zařízení (odečítače), zatím co s Braillským řádkem byla tato doplněna ještě hmatovým výstupem. Ten může být v některých situacích přehlednější a navíc nepodléhá vnějšímu rušení, zejména hluku z okolního prostředí, jak je tomu u samotného hlasového výstupu. Analýzou samotného proces zadávání vstupního textu uživatelem lze zjistit, že tento je velice zdlouhavý. Zrakově hendikepovaný uživatel totiž nedokáže na malém displeji smartphonu zpaměti lokalizovat polohu jednotlivých písmen softwarové klávesnice a je tak nucen postupným posunem prstu po klávesnici a poslechem odečítače hledat každé písmeno zvlášť.

Jako poslední bylo zmíněno hlasové ovládání a zadávání textu. Moduly určené k rozpoznávání hlasu dosahují lepších výsledků, pokud jsou cíleny na jednoho uživatele. Verze, které dokáží pracovat s hlasem libovolného řečníka, jsou tedy obecně méně efektivní i při ideálních podmínkách. V praxi je ovšem nutné počítat s místy i intenzivnějším okolním hlukem. Tato metoda má tedy pro nevidomého uživatele svá omezení vzhledem k vnějšímu prostředí. O to více tím, že je uživatel nucen odříkat převáděný text nahlas, což může být na veřejnosti nežádoucí s ohledem na soukromí a bezpečnost. Standardní převaděče hlasu navíc potřebují pro svou efektivní činnost poměrně velké soubory vzorových zvukových dat.

Některé z nich proto pracují v online režimu, kdy zvuková data od uživatele jsou odesílána na server, kde je provedena jejich analýza a nazpět je uživateli odeslán převedený text. Vyžadují tedy datové připojení, které je ne vždy a pro všechny uživatele dostupné.

Kapitola 3

Platforma Android

V této kapitole je rozebírána problematika platformy Android. Nejprve je uveden její obecný popis a seznam jednotlivých verzí (viz podkapitola 3.1), poté se text zaměřuje na OS Android z hlediska přístupnosti a tato podkapitola 3.2 se dále člení na menší úseky popisující jednotlivé verze systému počínaje verzí 4.1. Ke každé z nich jsou uvedeny jejich nejvýznamnější novinky v oblasti přístupnosti a pokud má některá z uvedených featur potenciální vliv na vytvářenou klávesnici, jsou zde zmíněny i některé její implementační detaily. Poslední podkapitola (3.3) pojednává o architektuře systému Android z pohledu jednotlivých vrstev a jejich rolí při vývoji a běhu aplikací. Každé z pěti vrstev architektury je pak věnována samostatná část. Kapitola čerpá z [20] a [14].

3.1 Obecný popis platformy

Android bývá ve většině odborných literatur definován jako open source[1] mobilní operační systém postavený na linuxovém jádře. Vlastníkem je firma Google, která rovněž řídí další rozvoj této platformy. První release Android 1.0 SDK byl vydán 23. září 2008 [1]. Za téměř 9 let svého vývoje prošel systém mohutným rozvojem, který vedl k představení aktuálně nejnovějšího Androidu verze 7.1.2, označovaného též Nougat. Tato verze byla oficiálně vydána v dubnu roku 2017. Systém je možné vidět nejen v dotykových mobilních telefonech a tabletech, ale rozšířil svoji působnost i na zařízení, jako jsou digitální kamery, přehrávače a televize, "chytré" náramkové hodinky a herní konzole. Díky standardu Android Auto dokáže systém spravovat takto kompatibilní základní řídící jednotkou automobilu. Navíc je od verze 5.0 Lollipop použitelný i pro 64-bitové verze architektur ARM, MIPS a x86.

Android prošel během svého vývoje několika stádii, z nichž ty nejdůležitější popisuje tabulka 3.1. V ní je kromě názvu a data vydání konkrétní verze[2] možno dočíst se také API

[1] ovšem ve většině zařízení je doplněn proprietárními komponentami

[2] Vždy první release u daného AP

level. Je to číslo jednoznačně identifikující revizi framework API, které jednotlivé verze Androidu nabízejí. Toto číslo je velmi podstatné, neboť jsou za ním skryty systémové featury a funkce, které programátorovi daná verze operačního systému dovoluje použít. Na internetových stránkách [3] vývojářské části OS Android je k dispozici přehledný nástroj pro vyhledávání změn v API mezi jednotlivými verzemi, případně je zde možné dohledat, od jakého API levelu je konkrétní funkce dostupná.

Verze platformy	API level	Kódové označení verze	Datum releasu
Android 7.1 - 7.1.2	25	NOUGAT	4.10.2016
Android 7.0	24	NOUGAT	22.8.2016
Android 6.0, 6.0.1	23	MARSHMALLOW	5.10.2015
Android 5.1 - 5.1.1	22	LOLLIPOP_MR1	9.3.2015
Android 5.0 - 5.0.2	21	LOLLIPOP	12.11.2014
Android 4.4W - 4.4W.2	20	KITKAT_WATCH	25.6.2014
Android 4.4 - 4.4.4	19	KITKAT	31.10.2013
Android 4.3, 4.3.1	18	JELLY_BEAN_MR2	24.7.2013
Android 4.2 - 4.2.2	17	JELLY_BEAN_MR1	13.11.2012
Android 4.1 - 4.1.2	16	JELLY_BEAN	9.7.2012
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1	16.12.2011
Android 4.0 - 4.0.2	14	ICE_CREAM_SANDWICH	18.10.2011
Android 3.2 - 3.2.6	13	HONEYCOMB_MR2	15.7.2011
Android 3.1	12	HONEYCOMB_MR1	10.5.2011
Android 3.0	11	HONEYCOMB	22.2.2011
Android 2.3.3 - 2.3.7	10	GINGERBREAD_MR1	9.2.2011
Android 2.3 - 2.3.2	9	GINGERBREAD	6.12.2010
Android 2.2 - 2.2.3	8	FROYO	20.5.2010
Android 2.1	7	ECLAIR_MR1	12.1.2010
Android 2.0.1	6	ECLAIR_0_1	3.12.2009
Android 2.0	5	ECLAIR	26.10.2009
Android 1.6	4	DONUT	15.9.2009
Android 1.5	3	CUPCAKE	27.4.2009
Android 1.1	2	BASE_1_1	9.2.2009
Android 1.0	1	BASE	23.9.2008

Tabulka 3.1: Přehled verzí OS Android a API level jimi podporovaný [12]

3.2 Android a přístupnost

Pokud hovoříme o přístupnosti v kontextu digitálních zařízení, myslíme tím z pravidla jejich podporu pro ovládání hendikepovanými uživateli. Jelikož tato práce je věnována problematice ovládání elektronických zařízení při absenci vizuálního kontaktu, bude zde cíleno na technologie zlepšující přístupnost z pohledu osob s vadou zraku. V této souvislosti je pak za přístupnost nejčastěji považována spolupráce uživatelského rozhraní (UI) systému a jeho aplikací s odečítačem obrazovky, případně pak systémová podpora ovládání pomocí gest, hlasu apod.

Operační systém Android prodělal v průběhu svého vývoje zdokonalení nejen v oblasti podpory rozličných typů zařízení, jejich periferií, výkonu a vzhledu, ale postupně integroval i několik prvků zlepšujících jeho přístupnost. Vývoj probíhal postupně a vyšší API levely přinášely postupně dokonalejší a mohutnější podporu přístupnosti. Zařízení s Androidem nižším než verze 4.1 je již v dnešní době méně než 1.5% [11] a stále jich ubývá.

Poslední dostupná statistika navíc ještě nezahrnuje přicházející Android 8.0, který bezesporu ukrojí další díl na úkor zastoupení zařízení se systémy v4.0.4 a starší. Postupně tedy padá nutnost udržování zpětné kompatibility se zastaralými verzemi OS. Z těchto důvodů se zaměříme na změny a inovace v jednotlivých API levelech až od verze Android 4.1 Jelly Bean (API level 16+). Následující text čerpá z [19].

3.2.1 Android 4.1 Jelly Bean (API level 16)

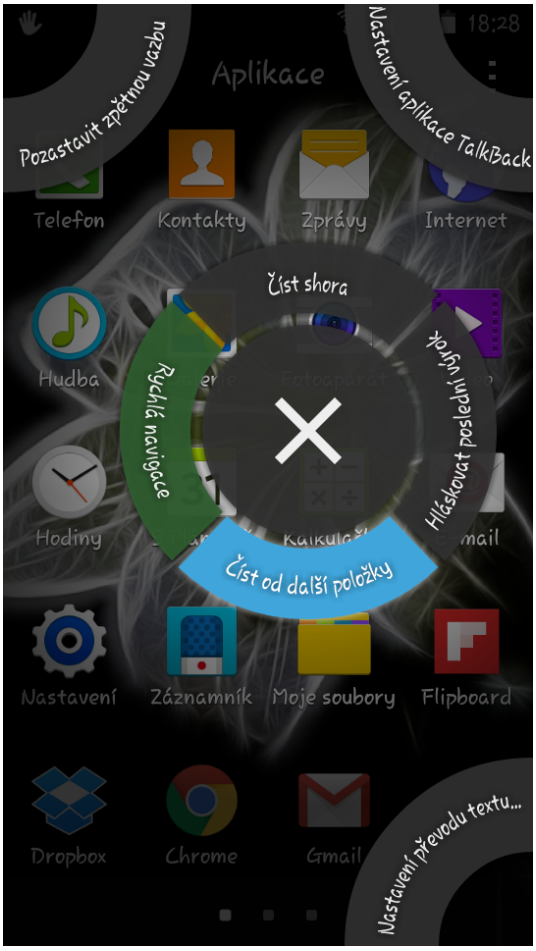
OS Android je od verze 4.0 vybaven již plně integrovaným systémovým odečítačem obrazovky TalkBack. Jeho featurou je režim nazývaný "Explore by Touch", tedy doslovně přeloženo "prozkoumat dotykem". Ten se využívá tak, že uživatel přejíždí postupně prstem přes displej a odečítač jej informuje, co se v tu chvíli pod jeho prstem nalézá.

Jelly Bean přináší nově gesta spolupracující právě s odečítačem. Gesta jsou z pohledu přístupnosti výraznou inovací a přínosem při uživatelském pohybu ve veškerých seznamech, ať už se jedná o ikony, menu aplikací[3] či systémová nastavení. Mezi gesta patří zejména tzv. Swiping, nejčastěji překládaný jako "švihání". Rychlým přejetím prstem přes displej v určitém směru způsobí uživatel posun v nabídce na předchozí (následující, zanořenou, nadřazenou, ...) položku ve směru pohybu prstu. Název nebo typ této položky, případně další informace k ní jsou uživateli ihned sděleny pomocí odečítače, respektive hlasového syntetizéru JustSpeak přijímajícího vstup od odečítače a přehrávajícího na zvukový výstup zařízení požadovaný text.

Dalším z gest je přejetí prstem směrem dolů a doprava, která vyvolá globální kontextové menu. Funguje ovšem pouze v případě, že je aktivní odečítač obrazovky (TalkBack).

[3] za nutného předpokladu, že aplikace je vytvořena právě s ohledem na přístupnost

Globální kontextové menu je znázorněno na obrázku 3.1, kde je možné vidět rozložení tlačítkových oblastí. V rozích jsou umístěna tlačítka pro nastavení TalkBacku a jeho dočasné pozastavení. Rovněž je přes toto menu možné ovládat čtení položek na stránce.



Obrázek 3.1: Globální kontextové menu režimu TalkBack na telefonu Samsung A3 [18]

Další novinkou Androidu verze 4.1 je začlenění Eyes-Free Keyboard, tedy klávesnice umožňující zadávání textu poslepu. Funguje tak, že uživatel za pomoci TalkBacku s aktivovaným režimem Explore by Touch přejíždí prstem přes jednotlivá písmenka na virtuální klávesnici, prostřednictvím odečítače je informován o jejich znacích a pokud chce nějaké písmenko použít, pouze zvedne prst z displeje.

Rovněž zvukový výstup se v této verzi dočkal inovace a od ní mimo jiné umožňuje rozpoznávání diktovaného textu bez nutnosti internetového připojení. Jsou zde však známy interferenční problémy mezi odečítačem a hlasovým vstupem.

3.2.2 Android 4.2 Jelly Bean (API level 17)

Ve verzi 4.2 přichází Android se dvěma významnějšími počiny v oblasti přístupnosti. Prvním z nich je Magnification Gestures, tedy systémově integrovaná lupa (Android v 4.2.2). Pokud je tato feature v nastavení povolena, je kdykoliv možné trojnásobným kliknutím jedním prstem vyvolat zvětšovací výřez a následně se s ním pomocí dotyku a tažení dvěma prsty pohybovat po původně zobrazené obrazovce. Doposud Android v základu neumožňoval cílené zvětšování zvolené části textu. Featura navíc spolupracuje s odečítačem, ovšem pouze ve statické poloze zobrazené zvětšené výseče. V případě zcela nevidomých uživatelů je však pochopitelně tato featura irelevantní.

Druhým vylepšením je upgrade některých gest použitelných v režimu odečítače. Zejména přidání možnosti kdykoliv aktivovat TalkBack podržením dvou prstů na displeji. Dle dokumentace má být tato featura použitelná i při uzamčeném telefonu. Některé zdroje [4] uvádějí, že by se měla v systému nacházet již od verze 4.1, ovšem při testech se smartphonem Lenovo P780 s OS Android 4.2.1 se tuto feature nepovedlo aktivovat v žádném z režimů. To by nasvědčovalo tomu, že byla do systému integrována až ve verzi 4.2.2. Navíc je v její specifikaci uvedeno, že gesto slouží pouze k aktivaci, nikoliv k přepínání. Dočasné pozastavení TalkBack je nově možné v globálním kontextovém menu (gesto: tah prstem dolů a doprava). Jedná se však o vícekrokovou deaktivaci, která trvá pouze do uzamčení telefonu nebo opětovného aktivování odečítače.

3.2.3 Android 4.3 Jelly Bean (API level 18)

Některé zobrazované prvky, jako obrázky nebo grafická tlačítka/nabídky není možné jednoduše interpretovat odečítačem, dokonce i v případě, že obrazec jimi reprezentovaný je ve skutečnosti číslo, slovo nebo předmět. Odečítač totiž nedisponuje logikou pro rozpoznávání obrazu. Odečítač je schopen pojmenovat element podle jeho typu, například "obrázek"nebo "tlačítko". Pokud však má být odečítač schopen sdělit uživateli více informací o elementu, třeba jaká věc se na tomto obrázku vyskytuje nebo jaký text se nachází na tamtom grafickém tlačítku, je nutné takový element opatřit labelem.

Label je textový popis elementu, který může vývojář aplikace napsat přímo do zdrojového kódu aplikace. Tato informace se nezobrazuje, proto nijak nenarušuje vzhled aplikace, ovšem poslouží odečítači při interpretaci. Problémem zejména dřívějších aplikací (ačkoliv ani v dnešní době toto není výjimkou) byla absence těchto popisů. Taková aplikace pak byla pro nevidomého člověka prakticky nepoužitelná.

Android 4.3 přichází s možností vlastního přidání textového popisu k libovolnému tlačítku aplikace, i jeho dodatečné editace. Uživatel si tak může sám přidat chybějící label, nebo upravit stávající nedostatečně výstižný popis nějakého elementu, pochopitelně za před-

pokladu jednorázové asistence vidící osoby, která poskytne popis daného prvku.

Z vývojářského pohledu pak s Androidem 4.3 přibývá u vytvářené aplikace nutnost deklarovat její funkcionalitu vzhledem k systémem podporovaným možnostem přístupnosti. Deklarace se provádí pomocí XML atributů v souboru s metadaty aplikace. Pokud zde nejsou deklarovány žádné vlastnosti, aplikace nebude spolupracovat se systémovými featurami pro přístupnost. Stane se tak proto, že aplikaci nebudou systémem doručovány eventy spojené s danou featurou a činností.

Rovněž přibyla možnost kopírovat a vkládat text i v režimu talkbacku, protože služba *AccessibilityService* nyní dokáže pracovat s událostmi započetí a ukončení výběru textu, následně pak kopírování a vkládání vybraného textu. Tato funkcionalita se stala součástí *AndroidSupportLibrary* a tak ji lze díky zpětné kompatibilitě využít i v nižších verzích Androidu.

3.2.4 Android 4.4 KitKat (API level 19)

KitKat jako největší ze svých přínosů v oblasti přístupnosti uvádí tzv. "system-wide settings for closed captioning". Jedná se o nastavení preference titulků, ovšem v rámci systému. Všechny aplikace, zejména ty pracující s videem, se pak mohou přes nově přidané metody API dozvědět, jaké vlastnosti (velikost, barva, pozice, ...) titulků uživatel preferuje. Toto rozšíření je poměrně komplexní a umožňuje celou řadu dalších nastavení, avšak z pohledu vytvářené aplikace zjevně nemá praktického využití a proto mu nebude věnována detailnější pozornost.

Pro potřeby interakce uživatele se systémem daleko zajímavější je další rozšíření, které se týká přidáni možnosti označit konkrétní části uživatelského rozhraní aplikace jako "live regions". Toto má velké využití u částí grafiky, které mění svůj vzhled. Například pokud pole určené pro zadání přihlašovacích údajů aplikace dokáže změnit svůj vzhled v případě uživatelovy neúspěšné autentizace a zobrazit text "Neplatné uživatelské jméno nebo heslo.", je vhodné toto pole označit jako "live region". Odečítač obrazovky pak bude na tomto elementu očekávat změnu obsahu a informuje o ní uživatele bez nutnosti přepnout focus na další prvek a následně zpět. Označení prvku se provede buď přidáním atributu *accessibilityLiveRegion* do XML layoutu komponenty ke konkrétnímu prvku, nebo zavoláním metody *setAccessibilityLiveRegion*().

3.2.5 Android 5.0 Lollipop (API level 21)

Dalším vydaným releasem Androidu v řadě je verze 5.0 Kitkat. Ta v sobě zahrnuje většinu dříve představených featur pro podporu přístupnosti a navíc přináší i některé novinky. Jednou z nich je "Switch Access"umožňující uživatelům se sníženou schopností pohybu

nakonfigurovat tlačítka svých externích zařízení jako klávesové zkratky poskytující okamžitý přístup ke zvoleným akcím. Tato featura je však pro naši aplikaci rovněž irelevantní.

Jako další inovace lze jmenovat celou skupinu nových vylepšení, které jsou všechny zaměřeny na úpravu barev a kontrastu a jsou převážně označeny jako experimentální. Jedná se o "High contrast text", "Colour inversion"a "Colour correction". První jmenovaná funkcionalita, pokud je aktivována, má za úkol zvýšit kontrast textu vzhledem k pozadí. Například všechny šedé či jinak barevné texty, pokud jsou umístěny na bílém pozadí, se automaticky změní na texty barvy černé. Druhá funkcionalita má pak měnit barevné schéma textu a pozadí mezi sebou navzájem, čímž dojde k inverzi barev zobrazených na displeji zařízení. Bílé pozadí s černým textem je tedy změněno na černé pozadí s bílým textem apod. Třetí zmíněné vylepšení, v překladu "korekce barvy", umí pracovat ve třech režimech:

- Deuteranomalie - pro osoby se sníženým vnímáním zelené barvy
- Protanomálie - pro osoby se sníženým vnímáním červené barvy
- Tritanomálie - pro osoby se sníženým vnímáním modré barvy

Z vývojářského pohledu pak ve verzi 5.0 přibyla možnost získat detailní informace o vlastnostech všech viditelných prvků, které jsou zobrazeny na aktuální obrazovce a se kterými mohou vidící uživatelé běžně interagovat. Také lze nastavit předdefinované či vlastní akce, které budou vykonány v případě detekce určitého prvku na obrazovce. Využívá se k tomu třída $AccessibilityNodeInfo.AccessibilityAction$, jež byla ve verzi Androidu 5.0 nově přidána.

Ještě stojí za zmínku vylepšená možnost kontroly nad převodem textu do řeči (text-to-speech syntéza). Byla přidána nová třída umožňující použití zvukových profilů spojených s umístěním, kvalitou, latencí a dalšími parametry specifickými pro proces transformace textu na řeč.

Na závěr pak ještě nutno zmínit informaci o novém způsobu změny výběru vstupní metody (IME – Input Method Editor). Ta má umožňovat intuitivnější a rychlejší změnu mezi jednotlivými metodami zadávání vstupního textu - obvykle mezi jednotlivými typy klávesnic, které jsou v systému aktuálně k dispozici.

3.2.6 Android 5.1 Lollipop (API level 22)

Vylepšení a nových funkcionalit z pohledu přístupnosti se v této verzi oproti Androidu 5.0 příliš neurodilo, respektive nejsou známy žádné významnější počiny v této oblasti. Za zmínku snad stojí pouze zavedení některých dílčích featur ze standardních mobilních androidích zařízení na platformu Android Wear verze 5.1.1, tedy OS určený především pro

"chytré hodinky"a podobné druhy zařízení. Jednalo se především o funkce spojené se zobrazením, tedy lupu, inverzi barev a možnost volby většího textu. Některé klíčové vlastnosti, jako je talkback, však tato platforma stále postrádá. Navíc se nejedná o primární cílovou platformu této práce, takže jí dále nebude věnován zřetel.

Zatím co mobilní platforma Android 5.1 neobdržela žádná klíčová vylepšení přístupnosti, byla zaznamenána řada negativních komentářů na některé dříve fungující vlastnosti. Například talkback a jeho problémy s checkboxy, respektive jejich stav, který neměl být nyní odečítačem prezentován. Dále pak problém s přesunem mezi obrazovkami aplikací pomocí tzv. švihání, tedy rychlého přejíždění prstem po displeji zařízení buď zleva doprava, nebo v obráceném směru. Kromě těchto dvou pak ještě několik dalších závad na funkcích pro přístupnost.

Výše zmíněné obtíže se s telefonem Oukitel K4000 vybaveným operačním systémem Android 5.1 nepodařilo ověřit, neboť všechny standardní funkce systému z pohledu přístupnosti fungovaly identicky s předchozí verzí systému. Mohlo se tak ovšem stát vlivem dodatečných oprav pomocí patchů jednotlivých inkriminovaných programů a služeb.

3.2.7 Android 6.0 Marshmallow (API level 23)

Release Androidu verze 6.0 povýšený releasem ze 7.12.2015 na 6.0.1 nese označení verze Marshmallow. Pro ni, respektive pro aplikace běžící na této verzi operačního systému, vydala společnost Google aplikaci nazvanou Accessibility Scanner. Tato aplikace je zdarma ke stažení v aplikaci Play Store[4] nebo na webových stránkách. [5]

Tato aplikace se po instalaci přidá do systémového nastavení, konkrétně do záložky "Usnadnění"[5]. Aplikace vyžaduje systémová práva pro snímání obrazovky a znemožnění uzamčení telefonu. Na oplátku nabídne možnost nechat si proskenovat a zanalyzovat, postupně jednu po druhé, všechny aktivity[6]. Výstupem takové analýzy je pak zhodnocení stavu uživatelského rozhraní skenované aplikace z pohledu přístupnosti.

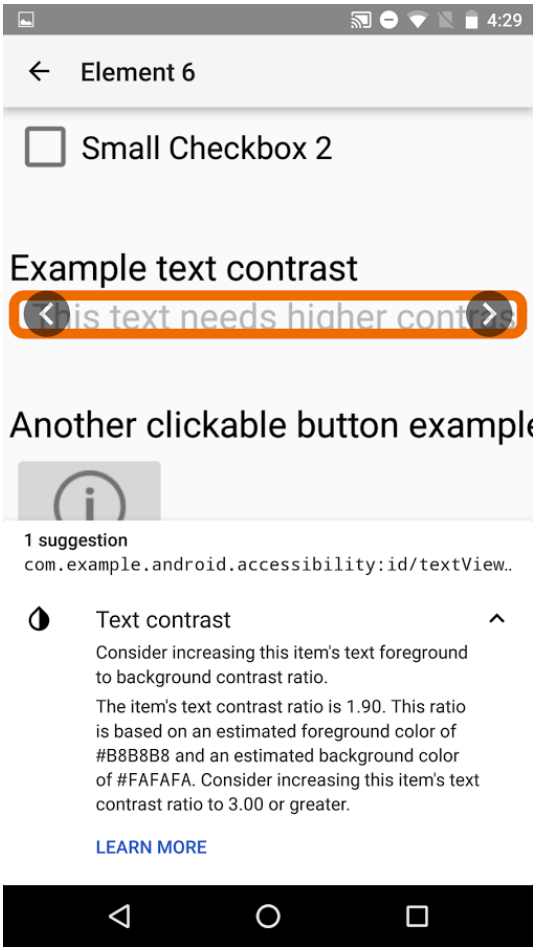
Na obrázku 3.2 vidíte ukázku možného výstupu. Aplikace navrhla zvýšení kontrastu jednoho textu, neboť ten byl napsán šedým písmem na bílém pozadí a pro osoby se zrakovými obtížemi by mohl být obtížně čitelný. Kontrolována je rovněž velikost tlačítek tak, aby tlačítka nebyla příliš malá a umožňovala pohodlné "stisknutí"(tím je samozřejmě myšleno kliknutí na zobrazený prvek na displeji zařízení) i pro uživatele s motorickými dysfunkcemi.

Ačkoliv je Accessibility Scanner vzhledem ke své funkci primárně určen pro vývojáře mobilních aplikací k otestování jejich UI, společnost Google jej prezentuje jako nástroj i

[4] V české jazykové verzi "Obchod Play"je předinstalovaná systémová aplikace určená pro stahování dalších aplikací, jejich aktualizací a jiných utilit z oficiálního uložiště do zařízení. Nacházejí se zde jak aplikace placené, tak i ty zdarma. Pro přístup a stahování aplikací je nutné internetové připojení.

[5] "Accessibility"v anglické jazykové verzi systému.

[6] V Androidu je jako aktivita označena každá obrazovka aplikace, tedy stránka zobrazená na displeji.



Obrázek 3.2: Ukázka práce s aplikací Accessibility Scanner [5]

pro běžné uživatele, kteří s jeho pomocí mohou sdělit své názory a poznatky k uživatelské přívětivosti konkrétních aplikací.

3.2.8 Android 7.0 Nougat (API level 24)

Android verze 7.0, který je označován jako Nougat, přichází v oblasti přístupnosti při nejmenším se dvěma významnými vylepšeními. Jsou jimi: Screen Zoom a Vision Settings in Setup Wizard. [13].

Screen Zoom je zcela nová systémová funkcionalita umožňující uživateli nastavit rozlišení zobrazení. Tím samozřejmě nelze ovlivnit hardwarové vlastnosti displeje, jímž je zařízení vybaveno. Lze však ovlivnit jeho proporce ve smyslu zmenšení nebo naopak zvětšení obsahu. Pokud je například na zařízení s rozlišením $800x480px$ nastavena velikost displeje pouze $640x384px$, je na stejné ploše displeje zobrazována menší část systému/aplikace a tím vlastně dochází ke zvětšení textů/tlačítek/obrázků. Při změně rozlišení displeje jsou

dotčeny jak systémové nástroje, tak i aplikace. Ty musí při zobrazení v nižším rozlišení vykreslené prvky přeskládat (toto naštěstí provádí komponenty automaticky) a pokud se například již nevejdou dvě tlačítka vedle sebe, nebo se na řádek vejde jen menší množství textu, jsou tyto vertikálně přetečeny a scrollováním je možné postupně zobrazit celý obsah.

Druhým zmíněným vylepšením je přidání možnosti aktivovat některé funkce z oblasti přístupnosti hned v úvodní systémové konfigurační obrazovce při prvním spuštění zařízení. Tím je nevidomým a slabozrakým umožněno využívat usnadňující funkce od samého počátku. Takto lze aktivovat zvětšovací gesta, velikost písma, výše zmíněné nastavení velikosti displeje a již několikrát zmíněný odečítač obrazovky TalkBack.

3.2.9 Android 7.1 Nougat (API level 25)

Zatím posledním oficiálně vydaným releasem je Android 7.1 (konkrétně minor release 7.1.2), který nese stejný název jako master release verze 7.0, tedy Nougat. API bylo povýšeno na verzi 25. V oblasti přístupnosti však na rozdíl od API verze 24 nepřináší žádné zásadní novinky.

3.3 Architektura systému z pohledu běhu aplikací

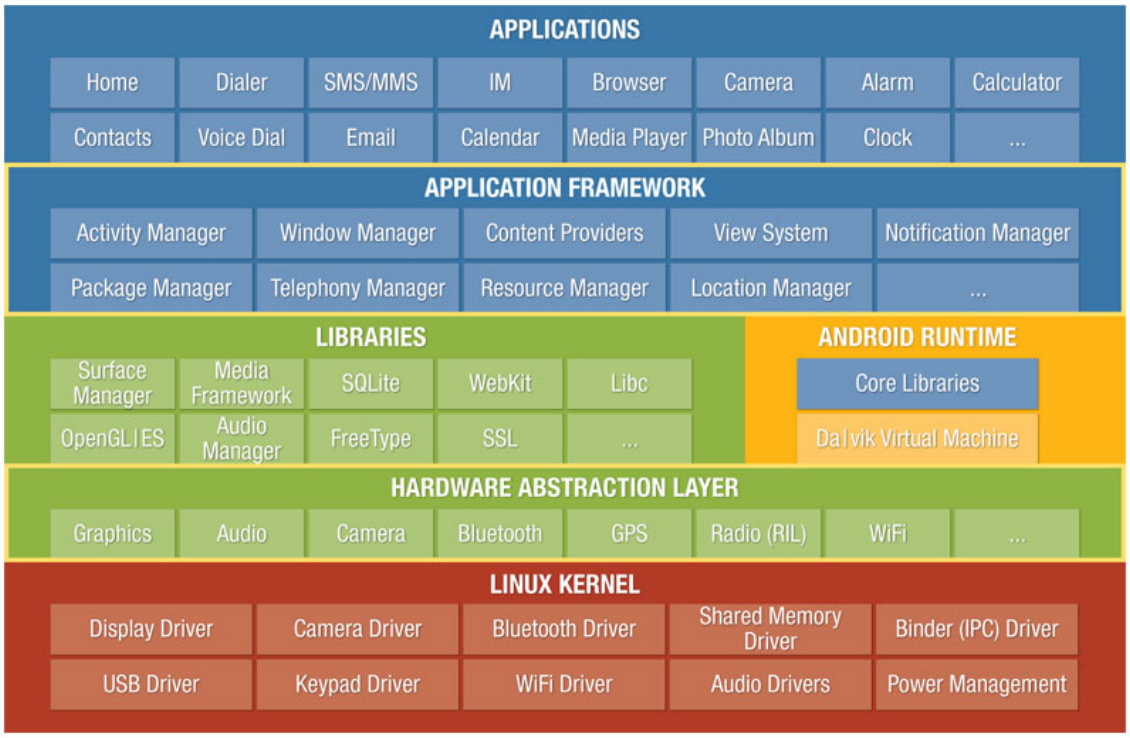
Pohled na architekturu operačního systému Android z hlediska rozložení na jednotlivé vrstvy není zcela jednoznačný. Některé literární prameny uvádějí čtyřvrstvou architekturu, jiné až šestivrstvou. Vše v závislosti na tom, jaký pohled na vzájemné propojení jednotlivých funkčních celků zaujmeme. Za nejspolehlivější zdroj dat se z tohoto pohledu jeví oficiální stránky Androidu, kde je architektura definována jako pětivrstvá. Pro lepší názornost je uveden obrázek 3.3, který sice pochází z jiného zdroje, ale názorněji popisuje celou skutečnost.

3.3.1 Linuxové jádro

Nejnižší vrstvou je linuxové jádro, na kterém je operační systém Android postaven. Nejnovější verze Androidu je založena na jádře verze 4.4 a to je pro potřeby mobilní platformy vybaveno řadou rozšiřujících patchů, kterých je v současné době něco přes 100. Mezi nimi například "Wake Locks"(systém pro správu paměti) či "Binder IPC driver"(Inter-Process Communication neboli mechanismus meziprocesové komunikace).

Linuxové jádro a tudíž i jádro Androidu OS obsahuje všechny nezbytné ovladače pro chod periferií, jako jsou display, klávesnice, Wi-fi, kamera a řada dalších. Rovněž procesy jako preemptivní multitasking[7] (souběžný chod více aplikací s nemaskovatelným přerušením

[7] od linuxového jádra verze 2.6



Obrázek 3.3: Architektura operačního systému Android [10]

od jádra OS směrem k procesům), správa sítí, správa běhu aplikací z pohledu systémových oprávnění či správa napájení zařízení. Nové ovladače je pak možno vytvářet stejným způsobem, jako jsou běžně vyvíjeny ovladače pro standardní linuxové operační systémy. Jádro operačního systému Android tvoří jako celek abstraktní vrstvu mezi hardwarem, ze kterého je zařízení sestaveno, a softwarovými částmi nacházejícími se ve vyšších vrstvách architektury.

Je možno si zkompilovat androidí jádro vlastní, tedy jádro, jež bude obsahovat přesně ty ovladače, které si sami zvolíme. V tom případě je nutno brát zřetel na minimální nutnou sadu ovladačů. Jádro OS je schopno nastartovat bez ovladačů periferií. Avšak pro reálný běh mobilního zařízení (byť ve velice omezeném režimu), bude několik do kompilace přidaných ovladačů nutných. Rovněž je možno použít některou ze starších oficiálních verzí systému, doporučeno je však používat vždy nejaktuálnější verzi jádra.

3.3.2 Vrstva hardwarové abstrakce - HAL

Hardware Abstraction Layer (HAL) je druhou z vrstev architektury OS Android. Tato vrstva bývá v některých architektonických modelech opomíjena, případně skryta v některých jiných vrstvách. Za validní zdroj dat se však dají beze sporu považovat oficiální stránky

Androidu pro vývojáře [14], které slouží zároveň jako dokumentace dynamicky se vyvíjejícího operačního systému, který modifikuje dílčí části své architektury poměrně rychle. Na zmíněném webu je HAL popsána jako samostatná vrstva.

HAL definuje standardní rozhraní pro výrobce hardware a umožňuje Androidu být nezávislý na implementaci nízkoúrovňových ovladačů. Pomáhá vývojářům dosahovat požadovaného chování jejich knihoven, aniž by byl ovlivněn či musel být modifikován systém Android na vyšších vrstvách. Systém pracuje s moduly ve formátu ".so"(sdílený zkompilovaný knihovní soubor), které jsou dynamicky nahrávány systémem v případě potřeby.

Z existence HAL na straně operačního systému Android vyplývá pro výrobce hardware povinnost implementovat spolu s ovladačem rovněž HAL rozhraní i na straně ovladače jejich zařízení. Vlastnosti HAL rozhraní jsou pro každou jeho verzi definovány v souboru *hardware/libhardware/include/hardware/hardware.h*, což zajišťuje jeho standardizovanou strukturu. Každé takové rozhraní může specifikovat další požadavky, které jsou nutné k chodu daného hardware.

3.3.3 Android Runtime a knihovny C/C++

Další vrstvou jsou nativní knihovny psané v jazyce C/C++ a Android Runtime, který v sobě obsahuje 2 základních části:

- Dalvik Virtual Machine
- Core Libraries

Android Runtime - Dalvik Virtual Machine (DVM)

Aplikace spuštěné v operačním systému Android, neběží jako procesy přímo v jádře, nýbrž je každému z nich přiřazena instance jejich vlastního DVM. Tato virtualizace je zavedena z několika důvodů. Jedním z hlavních je fakt, že se jedná ve své podstatě o jisté zapouzdření aplikace, která tím ztrácí možnost úmyslně či jiným způsobem zasáhnout do operačního systému a tím jej například poškodit. Právě tak nemají možnost přímo komunikovat s hardware, což je pozitivní jak z hlediska bezpečnosti, tak z pohledu efektivní správy zdrojů. Navíc to činí aplikace nezávislé na konkrétním použitém hardwarovém vybavení.

Dalvik jako JIT (Just-In-Time[8]) virtuální stroj je sám o sobě kvůli nízkoúrovňovým systémovým voláním závislý na linuxovém jádře a jedná se o jistou obdobu JVM (Java Virtual Machine). Vznikl nejen kvůli licenčním podmínkám na užívání JVM, ale i z technických důvodů, neboť je oproti JVM optimalizován pro chod na mobilních zařízeních. To znamená

[8]Překlad na instrukce procesoru probíhá v okamžiku potřeby.

lepší správu zdrojů a tudíž úsporu energie při dosažení požadovaného výkonu. Dalvik pracuje s Dalvik byte kódem, který je Dalvik kompilátorem kompilován ze standardního Java byte kódu a zabírá poloviční paměťovou velikost původního Java souboru.

Předchozí odstavec platí zejména pro verze Android OS 4.3 a menší. Od verze 4.4 se z důvodů dalšího zvyšování požadavků na výkon aplikací a úsporu energie začal používat princip tzv. AOT (Ahead-of-time compilation), tedy dopředná kompilace. Ta funguje tak, že aplikace je při instalaci zkompilována z byte kódu Dalviku do nativního binárního kódu procesoru daného zařízení. Tento krok je permanentní a při jakémkoliv budoucím spuštění aplikace není již nutná žádná další kompilace ani virtuální stroj. Pro zachování kompatibility je však původní Dalvik stále v Androidu přítomen.

Android Runtime - Core Libraries

Společně s DVM se na vrstvě Android Runtime nacházejí ještě Core Libraries. Tyto knihovny je dále možné rozdělit do 3 částí podle oblasti použití:

- Dalvik VM Specific Libraries - Knihovny specifické pro DVM
- Java Interoperability Libraries - Knihovny Java interoperability
- Android Libraries - Knihovny Androidu

Za knihovny specifické pro DVM se považuje ta část Core Libraries, která je určena pro přímou interakci s konkrétní instancí virtuálního stroje Dalvik. Tyto knihovny nebývají zpravidla vývojáři využívány. Výjimku tvoří případy, kdy je nutné získávat či modifikovat přímo některé parametry virtuálního stroje.

Knihovny Java Interoperability jsou vlastně open source implementací části knihovny standardních Java utilit. Ty byly pro potřeby aplikací psaných převážně v programovacím jazyku Java a běžících na DVM vhodným způsobem upraveny. Slouží vývojářům především pro usnadnění práce s textovými řetězci (tzv. stringy), síťovou komunikací, manipulaci se soubory a mnoho dalšího. Alespoň dílčí části této knihovny jsou při vývoji aplikací masově využívány prakticky všemi programátory bez ohledu na účel a strukturu vyvíjené aplikace.

Jako knihovny Androidu jsou označovány ty knihovny psané v Javě, které jsou specifické pro vývoj na platformě Android - zejména tedy knihovny aplikačního frameworku, dále pak knihovny sloužící pro tvorbu uživatelského rozhraní, přístup k databázi a vykreslování grafiky. Mezi nejdůležitější patří zejména:

android.app – Poskytuje přístup k aplikačnímu modelu a je základním kamenem všech Android aplikací.

android.content – Usnadňuje přístup k aplikačnímu obsahu (content), zasílání zpráv mezi aplikacemi a aplikačními komponentami.

android.database – Slouží ke zpřístupnění dat poskytovatelů obsahu a obsahuje třídy pro práci s SQLite databází.

android.graphics – Jedná se o nízkoúrovňové 2D grafické API určené pro kreslení, zahrnující barvy, body, plátna a obdélníky.

android.hardware – Je to API umožňující přístup k hardware (akcelerometry, světelný senzor a další).

android.opengl – Java wrapper pro OpenGL knihovnu (volanou přes C/C++ třídy), což je knihovna pro rendrování 3D grafiky.

android.os – Umožňuje přístup ke standardním servisám operačního systému, jako jsou zprávy nebo meziprocesová komunikace.

android.media – Obsahuje třídy umožňující přehrávání audia a videa.

android.net – Sada API, které umožňují přistupovat k síti (např. android.net.wifi umožňuje přístup k bezdrátovým sítím zařízení)

android.text – Používá se k vykreslování textu a manipulaci s ním na displeji zařízení.

android.util – Nápomocné třídy (utility) sloužící k operacím se řetězci, číselné konverzi, práci s datumy, nebo třeba zpracování XML

android.view – Jde o sadu základních stavebních bloků, ze kterých se skládá uživatelské rozhraní aplikací.

android.widget - Sada předpřipravených prvků uživatelského rozhraní jako tlačítka, popisky, layout manažery atd.

C/C++ knihovny

Na této vrstvě jsou společně s Android Runtime umístěny rovněž nativní knihovny psané v programovacím jazyce C/C++. Ač by se mohlo na první pohled zdát, že přítomnost C/C++ je nadbytečná (vzhledem k tomu, že primárním programovacím jazykem aplikací na Androidu je Java) a programátorovi postačí Core Libraries z Java Runtime, skutečnost je poněkud jiná.

Java knihovny z Core Libraries jsou ve skutečnosti v podstatě wrappery nad nativními knihovnami psanými v C/C++. Pokud je kupříkladu zavolána knihovna android.opengl pro

vykreslení 3D grafiky v zařízení, Java knihovna provede volání do OpenGL ES knihovny psané v C++ a tato následně kooperuje s linuxovým jádrem pro dosažení požadovaného výsledku. Kromě grafických operací obstarávají C/C++ knihovny ještě šifrovanou komunikaci pomocí SSL, práci s databází SQLite, přehrávání audia i videa a spousta dalších, včetně funkcí standardní systémové knihovny jazyka C (libc).

Standardně se tedy knihovny C/C++ nevolají na přímo, nýbrž v podstatě prostřednictvím Java Core Libraries a dalších. Existuje však i způsob, kterým lze nativní "nejavovské" knihovny zavolat přímo z Java kódu. K tomuto účelu byl vytvořen NDK (Native Development Kit), který umožňuje volat metody nativních knihoven prostřednictvím JNI (Java Native Interface).

3.3.4 Aplikační Framework

Aplikační Framework operačního systému Android je tvořen širokou sadou vysokoúrovňových služeb zprostředkovaných Java třídami, které mohou vývojáři používat ve svých aplikacích a které jako celek vytváření operační prostředí pro běh aplikací. Jejich přehled je znázorněn na obrázku 3.4.

Jedná se o služby jako například zpřístupňování dat v jiných aplikacích, interakce s prvky uživatelského rozhraní, správa aplikací běžících na pozadí, notifikace v systémovém stavovém řádku, toastové zprávy zobrazované uživateli a řada jiných funkcí či služeb.

Snahou Aplikačního Frameworku je implementace konceptu, kdy aplikace jsou vytvářeny ze znovupoužitelných, zaměnitelných a nahraditelných komponent. To přináší celou řadu výhod počínaje lepší upgradovatelností, přes zajištění korespondujícího vzhledu jednotlivých komponent, až po garantovanou kompatibilitu při vzájemné interakci.

Aplikační Framewok obsahuje mimo jiné následující klíčové služby:

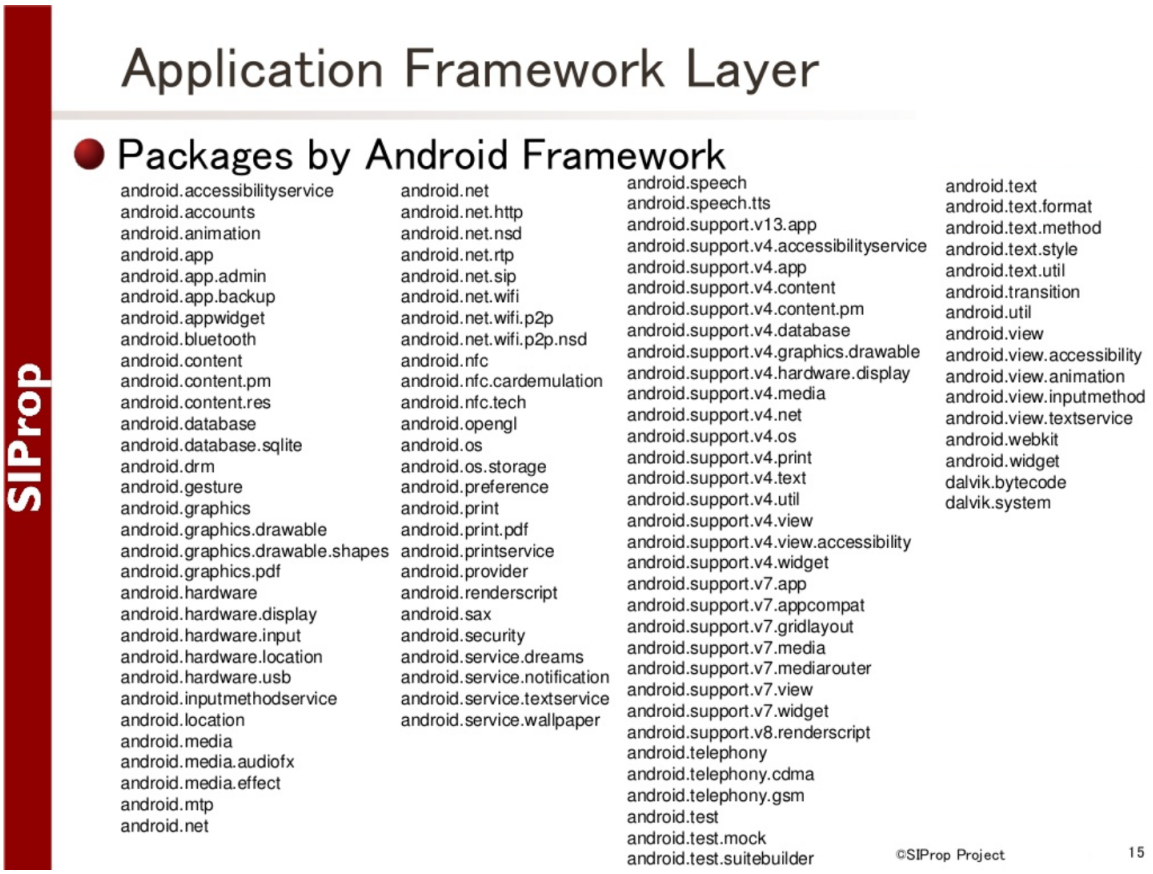
Activity Manager - Spravuje všechny části životního cyklu aplikací a operací zásobníku aktivit.

Content Provider - Umožňuje aplikacím publikovat a sdílet data s dalšími aplikacemi.

Location Manager - Poskytuje přístup k servisu umožňujícímu aplikaci získávání informací o poloze zařízení a jejích změnách.

Notifications Manager - Dovoluje aplikacím zobrazovat uživateli upozornění a informace ve stavovém řádku.

Package Manager - Nabízí aplikacím přehled o ostatních aplikacích, které jsou aktuálně nainstalované v zařízení.



Obrázek 3.4: Přehled Java tříd k dispozici na vrstvě Aplikačního Frameworku [2]

Resource Manager - Poskytuje přístup k řetězcům, nastavení barev a rozhraní uživatelských layoutů (poziční rozvržení jednotlivých prvků UI).

Telephony Manager - Umožňuje aplikacím získat informace o telefonické službě, jako je stav hovoru a informace o účastníkovi hovoru.

View System - Rozšiřitelná sada prvků sloužících k sestavení uživatelského rozhraní (tlačítka, textová pole, checkboxy, seznamy a další).

3.3.5 Vrstva Aplikací

Na samém vrcholu prezentované pětivrstvé architektury operačního systému Android se nachází vrstva Aplikací, které se dají rozlišit na dva druhy. Prvním z nich je vybraná sada předinstalovaných aplikací jako je webový prohlížeč, emailový klient, kalendář, aplikace pro sms komunikaci, správa kontaktů, obrázková galerie, kalkulačka a další. Druhým druhem jsou pak aplikace třetích stran. Ty mohou buď zajišťovat tutéž funkcionalitu, jako aplikace již předinstalované, anebo zcela odlišný druh aktivit. Jsou však doinstalovávány uživatelem

samostatně až následně po zakoupení zařízení s OS Android.

Aplikace jsou, jak již bylo zmíněno, psány v programovacím jazyce Java. O překlad se starají nástroje zabudované do IDE[9], které je volně k dispozici všem vývojářům aplikací pro platfomu Android. Nalézt jej lze na oficiálních stránkách operačního systému. [] Nástroj Android SDK zkompiluje kód spolu se všemi daty a zdrojovými soubory do Android balíčku, což je archivní soubor s příponou *.apk*. Tento soubor pak slouží k instalaci aplikace v zařízeních.

Vzhledem k tomu, že Android je tzv. multi-user linuxový systém, každá aplikace je vlastně z pohledu systému jeden uživatel. Má přiděleno své vlastní unikátní uživatelské ID, se kterým však pracuje pouze operační systém a aplikaci tak tento identifikátor zůstává skryt. Systém využívá tento identifikátor k ochraně aplikace a řízení přístupu následujícím způsobem: Veškerým souborům aplikace jsou nastavena přístupová oprávnění odpovídající uživateli, jehož ID bylo přiděleno právě této aplikaci. Tím je zajištěno, že nikdo jiný kromě nainstalované aplikace nebude mít přístup k jejím nesdíleným zdrojům.

V části 3.3.3 bylo popsáno, jak každý proces běží ve své vlastním DVM. To zajišťuje, že každá aplikace běží ve svém vlastním paměťovém prostoru, který je od ostatních aplikací izolován. Každá aplikace také standardně běží ve svém vlastním linuxovém procesu, který je Androidem spuštěn v okamžiku, kdy nějaká komponenta aplikace potřebuje vykonat určitou akci. Pokud není již více proces třeba, je systémem ukončen. K ukončení procesu může rovněž dojít, pokud je třeba obnovit paměť pro jiné běžící aplikace při nedostatku systémových zdrojů.

Tímto je docíleno, aby každá aplikace měla přístup pouze ke komponentám a systémovým součástem, které skutečně potřebuje a ke kterým má přidělena patřičná oprávnění. Aplikace může požádat o oprávnění přístupu ke kontaktům v zařízení, manažeru sms zpráv, připojeným paměťovým zařízením jako SD karta, kameře, internetovému připojení, bluetooth a řadě dalších. Požadavky na přístup k těmto zdrojům musí aplikace explicitně definovat v manifestu a právo na přístup k nim musí uživatel potvrdit při instalaci aplikace[10].

Aby mohly dvě či více aplikací sdílet stejné soubory, existuje možnost aby více aplikací sdílelo stejné uživatelské ID, běželo v jednom linuxovém procesu a na jednom a tomtéž virtuálním stroji. Je k tomu však zapotřebí, aby všechny takovéto aplikace byly podepsány stejným certifikátem.

[9] Integrated Development Environment, česky nazývané jako vývojové prostředí. Jedná se o software, jehož účelem je usnadnění práce programátorům při vývoji aplikací, často na určitou konkrétní platformu. Z pravidla bývá vybaven editorem zdrojového kódu, kompilátorem či interpret a většinou také debuggerem.

[10] Do verze Android 6.0 (s výjimkou verze 4.3) bylo možné buď všechna aplikací požadovaná práva najednou přidělit, nebo aplikaci vůbec neinstalovat. To se s novou verzí 6.0 mění a je možné o každém konkrétním zdroji pro každou aplikaci rozhodnout samostatně po instalaci aplikace.

Kapitola 4

Existující aplikace

Na úvod této kapitoly je nutno podotknout, že od jara roku 2014, kdy vznikl impuls k vytvoření této práce, se na poli použitelných konkurenčních aplikací odehrál mimořádný převrat. Nebýt něj, byl by obtížný nejen vznik této kapitoly, ale především by tím byl ovlivněn vznik výsledné klávesnice, neboť některé nyní existující aplikace poskytly cennou inspiraci ve fázích návrhu a vývoje vlastního řešení.

Tento beze sporu pozitivní trend lze přisuzovat tomu, že se přístupnost stává stále významnějším tématem při návrhu jednotlivých aplikací i celých operačních systémů. Podkapitola 3.2 demonstrovala vývoj systémových vylepšení i úpravy API, které poskytly programátorům přístup k širším a komplexnějším skupinám funkcí. Tím umožnily vznik nových aplikací uplatňujících inovativní přístupy k zadávání vstupního textu i celé manipulaci s prostředím daného zařízení.

Stávající aplikace, které jsou v tuto chvíli na trhu k dispozici, lze z jistého úhlu pohledu rozdělit na dva druhy. Prvním z nich jsou aplikace, které se instalují a slouží výhradně jako samostatné ekvivalenty nativní systémové klávesnice operačního systému Android. Může zde být pochopitelně konfigurační část, která se instaluje jako standardní aplikace a lze ji následně nalézt v seznamu přítomných aplikací, nicméně tato skupina aplikací si neklade ambice vyšší, než je zastávání postu vstupního komunikačního kanálu.

Druhým typem jsou oproti tomu aplikace typu Launcher. Launcherem se nazývá aplikace, která obsluhuje domovskou stránku systému, stará se o správu ploch, ikon a widgetů, zpravidla také poskytuje přístup do seznamu nainstalovaných aplikací, může zapouzdřovat přístup do zpráv, systémových nastavení a mnohé další. Úroveň ovlivnění vzhledu základního systému se může u různých Launcherů lišit od drobného předesignování tvaru ikonek plochy, až po téměř kompletní "převlečení" vanilla Androidu [1], kdy Launcher může přeski-

[1] Za vanilla Android jsou označovány takové verze systému, které obsahují pouze čistý build systému Android od společnosti Google. Obvykle totiž výrobci zařízení základní systém upravují, ať už přidáváním specializovaných ovladačů poskytujících lepší vlastnosti při běhu systému na konkrétním hardware, nebo

novat většinu prvků UI a dokonce i způsob interakce s nimi. Tím lze dosáhnout kompletní změny původního vzhledu systému a způsobu jeho ovládání.

Následující text si klade za cíl seznámení s některými významnými aplikacemi v oblasti přístupnosti dnešních dní. Aplikace jsou představeny z uživatelského pohledu a popis je zaměřen především na klíčové vlastnosti či podstatné slabiny konkrétních dílčích částí. Rovněž je zde umístěno případné srovnání či odůvodnění, proč nevzít danou aplikaci jako ideální prostředek, ale proč se naopak snažit současný stav ještě o kousek posunout směrem k lepšímu uživatelskému zážitku (user experience). Technické informace čerpají ze závěrů vlastního testování a dále z oficiálních webových stránek produktů [6], [7] a [8].

4.1 BlindShell

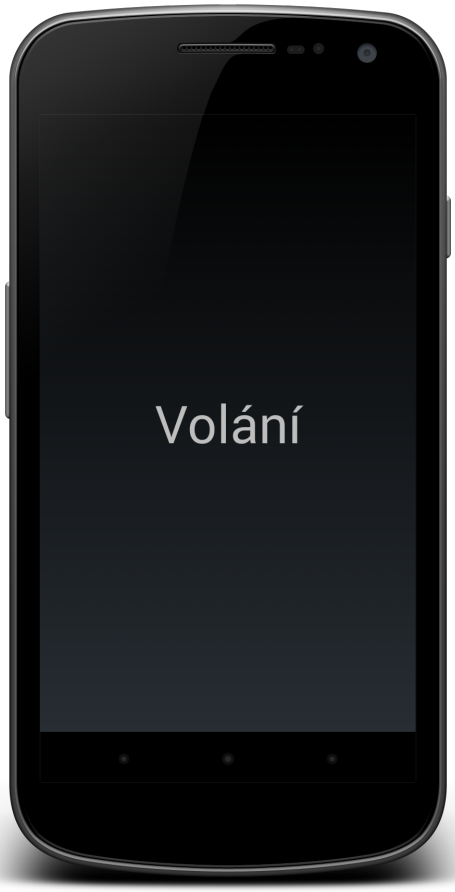
BlindShell je aplikace fungující na principu Launcheru, kdy tato po své instalaci žádá o přidělení práv "Správce zařízení". Následně je zobrazována místo standardních ploch s pozadím a ikonami nainstalovaných aplikací či widgetů. Jejím cílem je nabídnout uživateli komplexní správu celého zařízení takovou formou, jaká je pro nevidomé a slabozraké v rámci možností nejpříjemnější.

Jednotlivé obrazovky BlindSchellu obsahují každá vždy jen jeden název vykreslený velkým písmem přes značnou část displeje, viz obrázek 4.1. Jedná se například o Zprávy, Volání, Nastavení, Kontakty a další. Menu je děleno do podsekcí a velmi tak svou stavbou připomíná uživatelské rozhraní mobilních telefonů druhé poloviny devadesátých let minulého století. Celá aplikace BlindShell pochopitelně plně spolupracuje s odečítačem TalkBack a co více, je na něm do značné míry závislá. Sama totiž uživateli neposkytuje hlasovou odezvu, ale využívá již integrovanou systémovou funkcionalitu.

Kromě změny uživatelského rozhraní zahrnuje BlindShell ještě jednu podstatnou věc a tou je implementace vlastní klávesnice pro zadávání textového vstupu uživatelem. Rozvržení aplikace je znázorněno na obrázku 4.2, včetně textového pole v horní části obrazovky, kam je zobrazován uživatelem zadaný text. Z rozložení tlačítek je možno vidět, že se jedná o rozložení odpovídající tlačítkům na hardwarových klávesnicích zejména starších mobilních zařízení 4*3, doplněnou o spodní řadu 3 tlačítek sloužících pro odeslání zprávy, posun kurzoru v textu a smazání posledního napsaného znaku.

Každé z tlačítek pak obsluhuje více znaků klávesnice. Znaky, zejména písmena, jsou řazeny sekvenčně v abecedním pořadí a pro výběr druhého až n-tého znaku je na většině standardních klávesnic s rozložením 4*3 nutné vícekrát stisknout danou klávesu. Zde je princip mírně odlišný z důvodu způsobu orientace nevidomých na displeji telefonu. Nevidomý uživatel "hledá" každé tlačítko přejížděním prstem po displeji a naslouchání odečítači, jež

úpravou části UI za účelem většího odlišení zařízení od konkurence.



Obrázek 4.1: UI BlindShellu [6]

Obrázek 4.2: Klávesnice BlindShellu [6]

díky funkci *explorebytouch* sděluje, jaký prvek se právě nachází pod uživatelovým prstem. Z tohoto důvodu by mohla nutnost vícenásobného kliknutí na stejné místo vést k náhodnému kliknutí na vedlejší tlačítko vlivem nechtěného nepatrného posunutí prstu.

Kvůli výše zmíněné specifické potřebě nevidomých byl zde zvolen přístup, kdy prst, kterým uživatel hledal požadované tlačítko, zůstane po nalezení tohoto tlačítka pevně přitisknut k displeji telefonu. Tím je do značné míry omezeno nebezpečí nechtěného posunutí. Vícenásobné kliknutí a tím výběr dalších znaků v pořadí je pak realizován klepnutím jiného prstu na libovolné místo displeje - tedy pomocí tzv. dvojdoteku.

BlindShell klávesnice je dále vybavena funkcí hlasové kontroly napsaného textu. Ta probíhá tak, že v okamžiku, kdy je napsán znak mezera, přečte aplikace celé poslední slovo a tím uživateli umožní zkontrolovat, zda v průběhu zadávání neudělal chybu. Aplikace prozatím nenabízí žádné možnosti autokorekce pomocí předdefinovaného slovníku ani našeptávače pro dokončení textu.

Z počátku roku 2014 byla aplikace zdarma stažena z oficiálního Obchod Play uložiště a otestována na zařízení Samsung Young s OS Android 4.1.2, přičemž zůstala po celou dobu

funkční až do současnosti. Dnes již však z oficiálního zdroje aplikaci stáhnout nelze. Autor se rozhodl nabízet BlindShell předinstalovaný do mobilního zařízení stejného názvu společně se syntetizátorem od společnosti Acapela. Celý komplet je dle oficiálních stránek společnosti nabízen za 6690 Korun českých. Některá nezisková centra pak jako součást zařízení nabízejí i několikahodinový kurz ovládání zařízení.

BlindShell telefon je vybaven ještě rozšířeními, jako je čtečka elektronických knih, rozpoznávač barev a identifikátor bankovek. Protože se jedná o zařízení postavené na OS Android 4.4.2, mohou vidící uživatelé používat i nativní systémové prostředí a spolu s ním i veškeré standardní funkce. Zejména pak přístup k internetu pomocí webového prohlížeče, který v prostředí BlindShell Launcheru k dispozici není. Zda však při použití nativního uživatelského rozhraní Androidu zůstane k dispozici upravená klávesnice, nebo zda bude primárně k dispozici pouze standardní integrovaná QWERTY klávesnice, není bez reálného otestování zařízení zřejmé.

BlindShell Launcher je primárně určen k základnímu ovládání telefonu bez prohlížení webu, používání chatovacích aplikací a podobných vymožeností. V současné době je navíc poměrně draze placený vlivem nemožnosti získat samotnou aplikaci bez celého zařízení, které je v podstatě mobilní telefon s Androidem jako každý jiný. Přesto je na BlindShellu řada zajímavých věcí, jako je rozložení klávesnice a samotný postup při psaní textu. Rovněž automatická kontrola slov formou jejich vyslovení po napsání mezery může být z uživatelského hlediska příjemným zvýšením použitelnosti.

4.2 SwiftKey

Zatím co BlindShell (kapitola 4.1) je aplikace stvořená českým vývojářem a do celosvětového povědomí se dostává jen velmi pozvolna, navíc je zaměřena na zrakové hendikepované spoluobčany, v případě SwiftKey je situace diametrálně odlišná. SwiftKey je celosvětově značně rozšířená aplikace, která je určená pro všechny uživatele, jež touží po rychlejších způsobech zadávání textu a alternativních rozvrženích klávesnice.

SwiftKey je k nalezení na oficiálním uložišti Obchod Play a od června 2014 je navíc zdarma. Testovaná verze je označena jako 6.3.9.65*Beta*. Klávesnice se instaluje jako běžná aplikace, která kromě několika oprávnění nepotřebuje žádná speciální privilegia pro svůj běh, na rozdíl například od výše zmíněného BlindShellu s nutností učinit jej správcem zařízení. To je ovšem zapříčiněno tím, že zde se jedná "pouze"o systémovou klávesnici a nikoliv o komplexní launcher.

Součástí instalace je i konfigurační aplikace, kterou je možno spustit přes ikonu klávesnice v nainstalovaných aplikacích. Ta ihned po svém startu nabídne uživateli průvodce třemi základními kroky, které je nutné učinit, aby mohla být klávesnice v Androidu sku-

tečně používána jako systémová. Ukázku z konfiguračního menu je možné vidět na obrázku 4.3.

Prvním krokem je povolení této klávesnice v systému Android. Přes první položku konfiguračního menu je uživatel přesměrován přímo do systémového nastavení, konkrétně do sekce "Jazyk a zadávání", kde vidí seznam dostupných metod zadávání vstupu a může každou z nich individuálně povolit nebo zakázat. Po aktivování je uživatel upozorněn, že tato vstupní metoda může shromažďovat psané texty včetně hesel a čísel platebních karet. Druhým nutným krokem je pak vybrat SwiftKey jako klávesnici, která se bude skutečně používat. Výběr probíhá ze seznamu klávesnic povolených v předchozím kroku. V tuto chvíli je již klávesnice aktivována a zobrazí se po každé, kdy je vyžadováno zadání vstupního textu uživatelem.

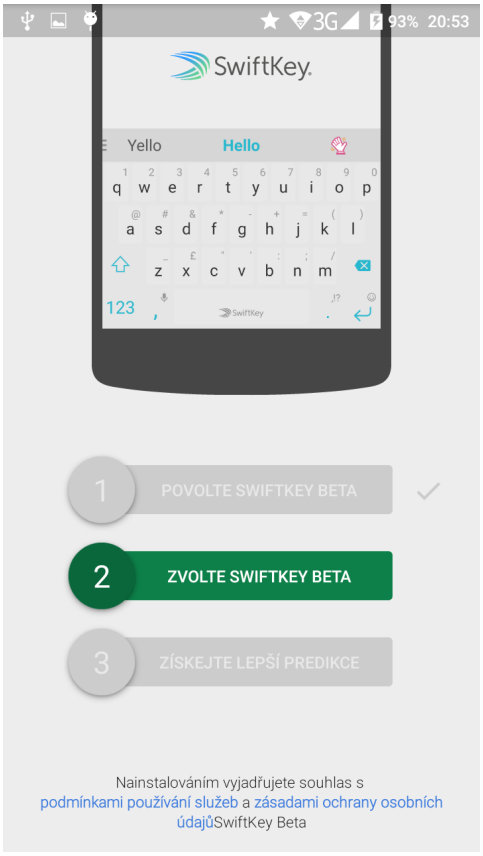
Konfigurační menu navíc ještě nabízí volbu "Získejte lepší predikce". Za ní se skrývá propojení aplikace s emailovým účtem od společnosti Google. Tím získá uživatel přístup k funkci, která proskenuje jeho emailovou komunikaci a z jím často používaných slov vytvoří slovník pro našeptávač šitý lidově řečeno "na míru"každému uživateli zvlášť. Dále pak sadu nových motivů či možnost zálohování a synchronizace.

SwiftKey nabízí velice širokou škálu dodatečných nastavení tak, aby umožnila uživateli maximální možnou customizaci[2]. Například pro český jazyk je možné vybrat hned ze šesti různých rozložení klávesnice (Dvořák, Coleman, QWERTZ, ...). Pro klávesnici lze vybrat z pěti velikostí kláves a tudíž i celé klávesnice na displeji. Volitelně je ke standardním čtyřem fixním řádkům (dynamicky se přidává jeden se slovy našeptávače/autokorektoru) dá přidat ještě jeden nahoru s čísly a jeden dolů, ve kterém jsou navigační šipky. Zajímavá je i možnost umístit znak "."jako druhé stisknutí tlačítka mezera za účelem zrychlení zadávání textu.

K největším počinům v oblasti rozložení klávesnice pak beze sporu patří dvě možnosti pokročilého nastavení. První z nich je změna klávesnice z jednoho uceleného bloku v bloky dva, kdy každý z nich je určen k ovládání palcem ruky. Druhou je pak možnost "odpoutat"klávesnici (nejen při dvoublokovém rozložení) od svého standardního spodního umístění a oba bloky umístit kdekoliv na displeji v libovolné výšce i vzdálenosti od sebe. Obě dvě vymoženosti jsou znázorněny společně na obrázku 4.4 na zařízení Oukitel K4000.

Se zadáváním textu souvisí i další za zmínku stojící vlastnost klávesnice, kterou je možnost zadávání textu tzv. přejížděním prstem přes displej. To funguje tak, že uživatel místo neustálého zvedání prstu a klepání na jednotlivé virtuální klávesy pouze položí prst na jednu klávesu a k dalším se přesouvá tažením položeného prstu po displeji. Tak, kde se v přejíždění na okamžik zastaví, nebo zde změní směr pohybu, aplikace detekuje stisk klávesy. U tohoto zadávání textu je díky relativně vyšší chybovosti detekce zadávaných písmen vhodná autokorekce, která však ve SwiftKey funguje velmi dobře (viz obrázek 4.5).

[2]Uzpůsobení aplikace tak, aby vyhovovala potřebám a způsobu práce konkrétního uživatele.



Obrázek 4.3: Konfigurační menu pro po-instalační aktivaci klávesnice [18]

Obrázek 4.4: Alternativní rozložení klávesnice ve vertikální poloze zařízení [18]

Pokud je tento způsob zadávání deaktivován, lze využít gesta jako změnu písma z velkého na malé nebo zpět (přejetí prstem přes klávesnici z vrchu dolů nebo obráceně), smazání posledního slova přejetím velmi rychle prstem z pravé části klávesnice do levé, případně změna rozložení klávesnice.

Funkce autokorekce či našeptávače, které spolu implementačně velice úzce souvisejí, již byla zmíněna stejně jako možnost jejího posílení zpřístupněním své emailové schránky u Googlu pro analýzu aplikací SwiftKey, která si ze slov psaných v emailové komunikaci vytváří svou vlastní databázi používaných slov. Aplikace si však nevytváří pouze slovník používaných slov, ale analyzuje kompletní stavbu textu. Je tak schopna s určitou pravděpodobností předvídat další slovo, které bude chtít uživatel napsat, na základě již napsané části věty. Aplikace tedy dokáže pracovat i s kontextem a provádí jistou obdobu syntaktické analýzy prováděné kompilátory a interprety formálních programovacích jazyků[3]. Našeptávač rovněž umí pracovat až se třemi jazyky zároveň, čímž odpadá častý problém, kdy při nastavení například automatické korekce v češtině prakticky není možné napsat část textu anglicky,

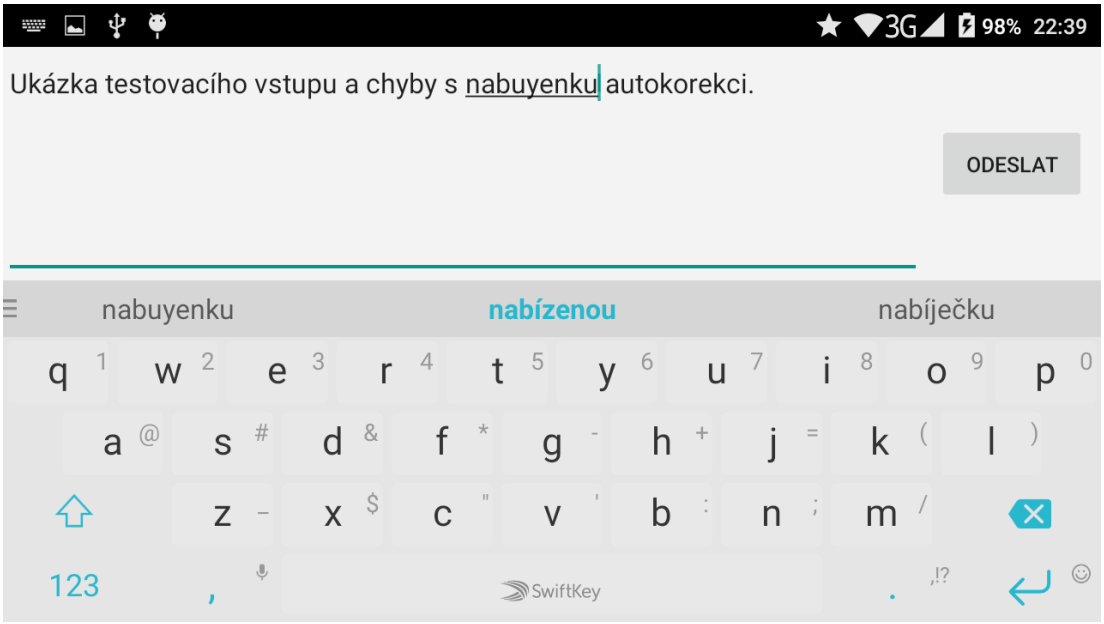
[3]Formálním jazykem zde rozumíme množinu slov (řetězců) konečné délky nad určitou abecedou.

aniž by byla autokorekce dočasně deaktivována.

Všechny výše zmíněné funkce jsou k dispozici všem uživatelům, v režimu TalkBack a zejména díky Explore by Touch (více viz kapitola 3.2.1) je použitelnost některých vylepšení omezena. Nelze využít ukotvení klávesnice do prostoru ani gesta klávesnice SwiftKey. Rovněž návrhy na automatické doplnění slova, které se dynamicky zobrazují v horním řádku klávesnice, bohužel nejsou dostupné jinak, než že by uživatel dokázal odhadnout jejich polohu a následně si je nechal přečíst přejetím prstem. Oproti tomu vyslovování slov po napsání slova funguje spolehlivě a stejně tak samotný režim Explore by Touch. Psaní probíhá tak, že pokud je hledaný znak klávesnice přejížděním prstem nalezen, zvednutím tohoto prstu z displeje zařízení je znak vložen to textového vstupního pole.

Klávesnice SwiftKey poskytuje velkou inspiraci z pohledu funkcí implementovaných pro uživatele bez zrakového hendikepu. Pro nevidomé je však velká část z nich nepoužitelná, což je zase dáno tím, že klávesnice primárně necílí na tuto skupinu uživatelů. Navíc je zde přítomno několik dalších problémů. Kupříkladu pro uživatele mobilních telefonů Samsung, kteří jako zdroj aplikací pro stahování využívají aplikaci Galaxy Apps, byla na konci července 2016 dostupná verze SwiftKey 5.2.2.124, která ještě podporu pro režim Explore by Touch na svých klávesách vůbec implementovánu nemá a je tak pro nevidomé zcela nepoužitelná. Dalším problémem je spolupráce klávesnice s některými aplikacemi.

Díky tomu, že SwiftKey je na rozdíl od BlindShellu pouze klávesnice, například pro psaní sms je využívána v systému nainstalovaná aplikace pro textové zprávy. Při testech se v některých aplikacích vyskytoval problém, že místo aby odečítač informoval uživatele o znaku, který byl momentálně napsán, informoval jej o změně počtu zbývajících znaků, které lze ještě do sms napsat. To samozřejmě vina SwiftKey není, nicméně technicky méně zdatného uživatele může tento fakt spíše odradit, než aby se pokusil nainstalovat jinou aplikaci pro psaní zpráv (např. aplikace Handcent SMS v6.9.3.3 byla úspěšně otestována).



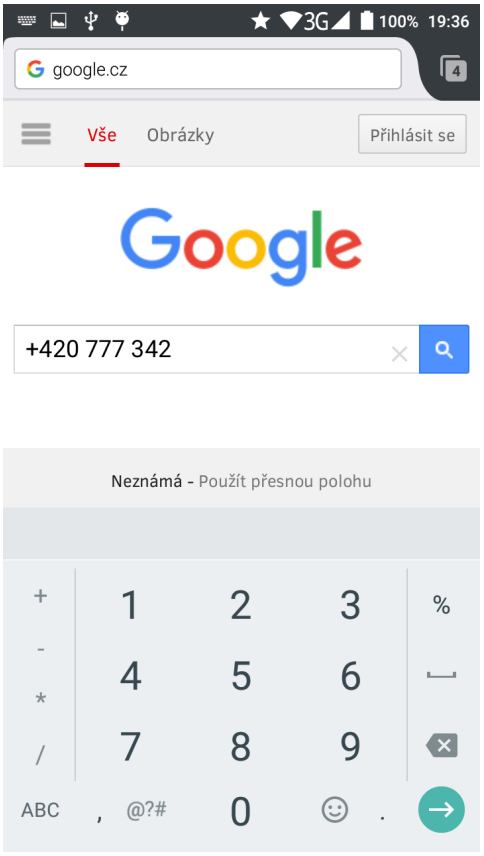
Obrázek 4.5: Autokorekční mechanismus a horizontální rozložení klávesnice SwiftKey [18]

4.3 Google Klávesnice

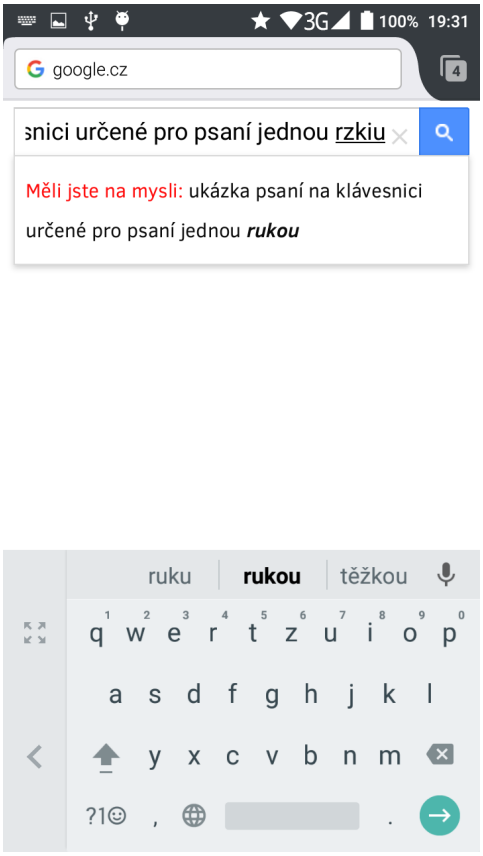
Systémová klávesnice pro operační systém Android od toho samého tvůrce, tedy společnosti Google, je od uvolnění verze s majoritním číslem 5 další významnou aplikací umožňující nevidomým zadávání vstupu v podobě textu. Stejně jako v případě klávesnice SwiftKey, i zde se jedná o samostatnou klávesnici a nikoliv o launcher. A podobností s je zde celá řada, proto bude věnována pozornost převážně pouze odlišnostem.

Testovaná verze 5. je volně ke stažení na oficiálním uložišti Obchod Play a její součástí je rovněž konfigurační rozhraní. Průvodce při prvním spuštění je v podstatě totožný s průvodcem aplikace SwiftKey, je zde tedy jak přímý přístup do nastavení pro aktivací klávesnice v systému, tak její následné nastavení jako primární klávesnice pro používání ve všech aplikacích. Jako třetí krok se spustí mini průvodce demonstrující klíčové funkce klávesnice. Jsou jimi opět psaní přejížděním prstem (zde pojmenováno jako Psaní gesty), našeptávač z naučeného slovníku a zde trošku odlišné umístění tlačítek pro zobrazení celé numerické klávesnice v rozložení 4*3 (viz obrázek 4.6) a zobrazení emotikon. V nastavení vzhledu je pak kromě možnosti volby barevného skinu ještě nastavitelná volba zobrazení okrajů každého tlačítka, bez nichž se klávesnice na pohled jeví jako jednolitá plocha s písmeny a znaky rovnoměrně rozprostřenými po celé šíři.

Defaultně jsou po instalaci v rozšířených nastaveních skryty aktivní volby odesílání statistických informací a úryvků uživatelských vstupů zadávaných do aplikací, dle popisu údajně pouze těch od společnosti Google. Oproti tomu volba synchronizace naučených slov



Obrázek 4.6: Numerický mód psaní klávesnice Google 5.x [18]



Obrázek 4.7: Mód psaní jednou rukou klávesnice Google 5.x [18]

mezi více zařízeními přihlášenými pod stejným google účtem, je v sekci *Slovnk* primárně deaktivována, ač se jedná o zajímavou funkcionalitu. Další možnosti jako psaní tečky pomocí mezerníku, nastavení velikosti klávesnice a pokročilá konfigurace chování našeptávače zůstává, stejně jako u SwiftKey, přítomna i zde.

Zajímavým počinem a zároveň lákadlem zde je opět funkcionalita z oblasti rozložení klávesnice, konkrétně režim psaní jednou rukou. Rozložení klávesnice v tomto režimu je možné vidět na obrázku 4.7, zde v nastavení pro pravou ruku. U zařízení o velikosti do 5" (pochopitelně v závislosti na velikosti ruky uživatele) není tento režim až tak podstatný, ovšem vzhledem k stále se zvětšujícím rozměrům displejů některých zařízení, či používání přímo menších (7") tabletů, jeví se tato funkce jako velmi užitečná. Očekává se ovládání klávesnice palcem, celá funkcionalita původní klávesnice je kompletně zachována, včetně našeptávače.

Pokud jde o samotný našeptávač, ten se z uživatelského hlediska chová stejně, jako u klávesnice SwiftKey, nicméně při reálných testech se jevil jako méně efektivní. Co oproti tomu fungovalo u obou klávesnic velmi dobře je rozpoznání hlasu, byť probíhalo za ideálních

podmínek bez okolního hluku a zařízení bylo připojeno k wifi. Obě klávesnice mají na jednom z tlačítek jako zrychlenou volbu při dlouhém stisknutí právě převod řeči na text, který je prováděn online. Více viz kapitola 2.3.

Klávesnice Google měla mít dle oficiálních release informací k testované verzi ještě jedno vylepšení a tím měl být upravený způsob odezvy aplikace v případě automatické korekce chybně zadaného vstupního textu. Jelikož však funkcionalita automatických oprav na žádném dostupném zařízení nepracovala korektně, nebylo možné ověřit chování našeptávače v tomto okamžiku. Dle informací by se každopádně mělo jednat o vyslovení slova, které má nahradit chybný řetězec a mělo by k tomuto vyslovení dojít dříve, než bude záměna provedena. To z toho důvodu, aby měl uživatel možnost takovou změnu vůbec nedovolit a ponechat původní text.

Za závěr porovnání se dá považovat fakt, že z pohledu přístupnosti jsou obě klávesnice velice podobné. Ze způsobu zpracování poinstalačního průvodce shodného s průvodcem implementovaným ve SwiftKey vyplývá, že se jedná o standard, který by měly seriózní aplikace poskytovat svým uživatelům tak, aby bylo možné co nejsnazší nastavení a tudíž co možná nejpohodlnější a nejrychlejší cesta k okamžitému používání.

Kapitola 5

Analýza a návrh aplikace

Ve spolupráci se střediskem pro pomoc studentům se specifickými nároky – Teiresiás, zřízeném Masarykovou univerzitou v Brně, vznikla nejprve myšlenka a následně začala vznikat i implementace staro-nového přístupu k zadávání vstupního textu – Sense Keyboard. Jejím cílem je nabídnout uživateli rozložení kláves a návaznou podporu zadávání zejména z pohledu přístupnosti na takové úrovni, aby i nevidomý uživatel byl schopen zadávat text bez nutnosti zdlouhavého hledání požadovaných kláves na displeji za pomoci vestavěného systémového odečítače.

Cílem je dosažení stavu, kdy bude uživatel zadávat text intuitivně, tedy bez potřeby zvýšené zrakové nebo zvukové kontroly. Právě toto intuitivní rozmístění bude v konečném důsledku přínosem nejen pro zrakově hendikepované, ale pro všechny uživatele. Ti již nebudou muset udržovat neustálý vizuální kontakt s displejem zařízení. Výsledné řešení má být spustitelné na co možná největším počtu zařízení ve smyslu nejnižší podporované verze operačního systému.

5.1 Analýza požadovaných vlastností

V rámci analýzy je třeba se zaměřit zejména na vzhled rozhraní pro zadávání vstupních dat a podpůrné technologie, jejichž cílem bude maximalizovat efektivitu při zadávání vstupů. V neposlední řadě je rovněž nutný rozbor vhodného vzhledu uživatelského rozhraní aplikace pro konfiguraci jejího chování.

Závěry uvedené v této podkapitole vycházejí zejména ze tří zdrojů. Prvním z nich jsou závěry stanovené z rozboru stávajících řešení, která jsou v současné době dostupná. Věnována je jim kapitola 4. Druhým zdrojem dat jsou vlastní zkušenosti autora, které zahrnují více než 16 let uživatelské práce, z čehož bylo přibližně 8 let na zařízeních s OS Android[1]. Třetím je pak pozorování zejména nevidomých uživatelů při interakci s mobil-

[1] Verze 4.0.4, 4.1, 4.2.2, 4.4, 5.1 a 6.0.1.

ními zařízeními. Pozorování probíhala jak osobně v Brně v prostorách střediska Teiresiás, tak zprostředkovaně formou instruktážních videí od nevidomých uživatelů.

5.1.1 Zadávání uživatelského vstupu

Z pozorování způsobů uživatelské interakce jednoznačně vyplývá, že nejvíce limitujícím bodem procesu zadávání uživatelského vstupu pomocí klávesnice je okamžik hledání správné klávesy pomocí režimu explore by touch. Čím více tlačítek je nutno tímto způsobem prozkoumat, tím větší je časová prodleva mezi zadáváním jednotlivých znaků. Při uživatelských testech se tato rychlost pohybovala v rozmezí 12-20 úhozů za minutu. Z toho jednoznačně vyplývá, že zadání jednoho znaku zabralo v průměru 3-5 vteřin. Klíčovou vlastností uživatelského rozhraní klávesnice Sense Keyboard by tedy měl být počet kláves, které bude uživatel aktivně obsluhovat, přičemž jejich množství by mělo být co možná nejmenší. S tím nepřímou úměrou souvisí i další podstatná vlastnost, neboť čím menší počet tlačítek bude vyplňovat prostor obrazovky, jejich velikost bude moci být vyšší. To umožní uživateli snazší zasažení požadované klávesy dotykem prstů při zadávání vstupu pouze na základě znalosti pozice klávesy.

Druhá jmenovaná vlastnost je pak ještě zásadnější, protože ta zásadně ovlivní užitné vlastnosti nejen pro uživatele se zrakovou indispozicí, ale rovněž pro uživatele bez vady zraku. Těm tak může kompenzovat komplikovanější zadávání oproti klasické QWERTY klávesnici způsobené nutností opakovaného stisku stejné klávesy v situaci, kdy je na jedné klávese umístěno více znaků.

Průměrná velikost displeje mobilních telefonů se dle http://screensiz.es/ pohybuje úhlopříčně okolo 5", tedy 127 mm. Při nejčastějším poměru stran 9:16 tak jednoduchým výpočtem získáme rozměry modelového prostoru:

$$V^2 + S^2 = U^2$$

$$(16x)^2 + (9x)^2 = 127^2$$

$$256x^2 + 81x^2 = 16129$$

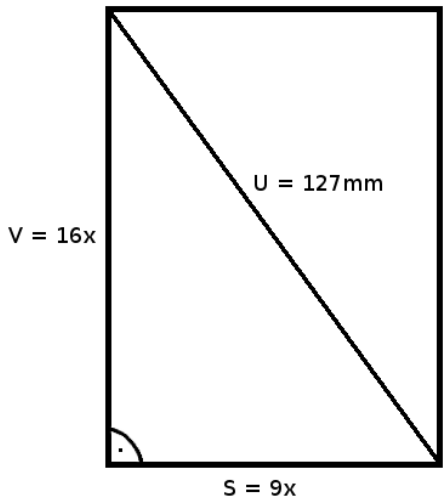
$$337x^2 = 16129$$

$$x \doteq \sqrt{47.86}$$

$$x \doteq 6.92mm$$

$$V = 16x \doteq 110.7mm$$

$$S = 9x \doteq 62.3mm$$



Při orientaci mobilního telefonu na výšku je pak u QWERTY klávesnice pouze přibližně 6,2mm na jednu klávesu[2], což fakticky zcela znemožňuje intuitivní použití takovéto klávesnice. Klávesnice typu 4*3 dosahuje výrazně lepších výsledků, neboť v tomto případě se o kratší, 62 milimetrovou šířku dělí pouze 3 klávesy a tak na každou zbývá v přepočtu více než 2cm. Díky tomu, že displeje zabírají drtivou většinu plochy dnešních mobilních telefonů, nabízí se uživateli držícímu zařízení na výšku možnost snadné orientace i bez vizuálního kontaktu - levá část displeje, střed displeje a pravá část displeje. Obdobná situace je pak u klávesnice 4*3 ve vertikálním směru, kde je při využití celé plochy a započítání minimální výšky pro zobrazení editovaného pole (cca 3cm) možno dosáhnout rovněž 2cm pro každý řádek kláves[3]. Z výše uvedeného tedy vyplývá závěr, že pro potřeby klávesnice Sense Keyboard se zejména z důvodu potřeby intuitivního zadávání textu jeví jako nejvhodnější klávesnice s rozložením 4*3.

5.1.2 Technologie podporující efektivní zadávání textu

Pod slovním spojením "technologie podporující efektivní zadávání textu"jsou myšlena funkční rozšíření aplikace, která mají za následek zrychlení, zpřesnění či zpříjemnění zadávání vstupního textu uživateli. Naprostým standardem u většiny obdobných aplikací je našeptávač, který umí za uživatele nejen dokončit některá často zadávaná slova či celá slovní spojení, ale slouží zároveň jako možná korekce v případě překlepů. Tato vlastnost je o to důležitější, pokud slouží jako podpora nevidomému uživateli, neboť jeho orientace v již napsaném textu a pohybování se v něm zpětně je poměrně komplikované. Našeptávač je v tomto případě schopen opravit chybu v právě napsaném slově, nahrazením celého zadaného slova nejvíce podobným slovem ze slovníku. Tím odpadá uživateli nutnost vrátit se ve slově na chybně zadaný znak, jeho oprava a následné opětovné přesunutí na konec vstupního pole pro pokračování v zadávání textu.

Velmi často se vyskytující je chování, kdy klávesnice dokáže sama bez uživatelské akce změnit své rozložení tak, aby co nejvíce vyhovovala obsahu právě zadávaného pole. Pokud je tedy například klávesnice vybavena standardním rozložením s písmeny a ještě speciálním, kdy jsou místo písmen na tlačítkách umístěna přímo čísla, může být uživateli ve vhodných případech automaticky nabídnuta klávesnice numerická místo té standardní. To může být žádoucí v situacích, kdy je zadáváno telefonní číslo nebo číslo popisné. Pokud je takovéto vstupní pole správně označeno, dokáže klávesnice nabídnout uživateli nejvhodnější rozložení.

Další, neméně rozšířenou, ačkoliv ne až tak kritickou vlastností klávesnic je, že dokáží detekovat konec věty a automaticky tak vložit velké písmeno na začátek věty nové. Někteří

[2]Při standardním rozložení QWERTY klávesnice s 10-ti klávesami v řádku.

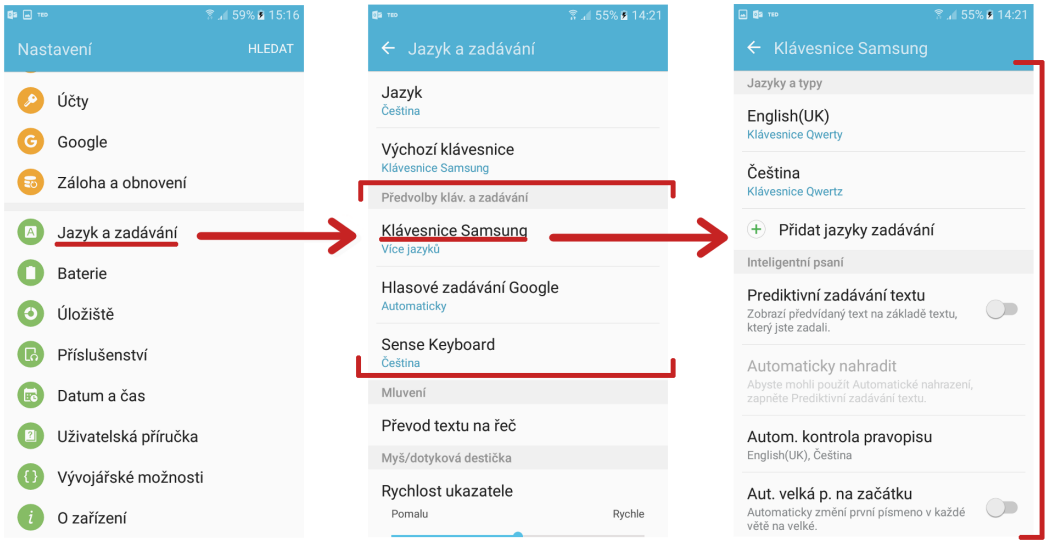
[3]11cm celkem - 3cm na textové pole = 8cm rozdělených do 4 řad, tedy 2cm na jednu řadu.

uživatelé by se pravděpodobně bez tohoto chování obešli a lze jej také v některých aplikacích deaktivovat v konfiguraci. Pro ty, kteří dbají na úpravu svého písemného projevu, je to však vítané vylepšení, které šetří čas.

Pro zrakově hendikepovaného uživatele se pak nabízejí další důležité vlastnosti, jako například možnost přečtení poslední napsané věty nebo průchod celým textem po slovech či větách.

5.1.3 Uživatelské rozhraní aplikace

Nedílnou součástí každé aplikace, která umožňuje customizaci, nebo vyžaduje prvotní nastavení, je její konfigurační rozhraní. Standardní aplikace mají svá nastavení součástí uživatelského rozhraní. Systémové aplikace, kterými jsou i systémové klávesnice, mají své konfigurace umístěny v systémovém nastavení v části "Jazyk a zadávání[4]". Zde je k dispozici seznam všech v systému dostupných metod zadávání textu. U každé, která to podporuje, je možné měnit její defaultní chování, viz obrázek 5.1.



Obrázek 5.1: Ukázka konfigurace systémové metody zadávání vstupu v OS Android 6.0.1 na zařízení Samsung J5(2016) [18]

Jelikož aplikace Sense Keyboard v sobě obsahuje systémovou servisu pro zadávání vstupu, před vlastním použitím bude nutné provést nejméně dva kroky, kterými se tato metoda pro zadávání vstupu v systému Android aktivuje a následně nastaví jako výchozí klávesnice. Tyto kroky je možné provést ve výše popsané části systémového nastavení. Z hlediska uživatelské přívětivosti se však nejedná o nejefektivnější způsob, neboť uživatel po

[4]Organizace hierarchie a pojmenování položek systémového nastavení se může lišit v závislosti na grafické nadstavbě jednotlivých výrobců, popis odpovídá stavu ve vanilla androidu.

instalaci samotné systémové servisy neobdrží žádnou informaci. Řešením, které se nabízí, je vybavit servisu také aktivitou (podrobnosti v kapitolách 3.3.4 a 6), která by uživateli poskytla informaci, kde požadovaná iniciální nastavení provézt.

Postup spočívající v notifikaci uživatele v sobě však skrývá rizika, zejména pak fakt, že na různých zařízeních mohou být vlivem UI nadstavby výrobce pozměněny názvy a struktura položek systémového nastavení. Aplikace by pak musela obsahovat postup pro všechna aktuálně podporovaná zařízení, ze kterých by si uživatel vybral to své, nebo v lepším případě by aplikace detekovala model sama a nabídla ihned správný návod. U tohoto přístupu však hrozí riziko neaktuálnosti aplikace při uvedení nových modelů na trh, nebo při aplikaci systémových aktualizací.

Za optimální byl zvolen přístup blízký tomu popsanému již dříve u aplikace SwiftKey a znázorněnému na obrázku 4.3. Hlavní myšlenkou je zde nabídnutí obdoby wizardu, který by s uživatelem bezprostředně po instalaci provedl konfiguraci krok po kroku. Součástí tohoto průvodce by měla být i obě výše popsaná nastavení nutná pro aktivaci klávesnice. Celý průvodce by měl být samozřejmě upraven pro odečítač obrazovky tak, aby měl uživatel s vadou zraku potřebné informace o dění na obrazovce svého mobilního zařízení.

5.2 Návrh aplikace Sense Keyboard

Koncept Sense Keyboard vychází částečně z analýzy slabých míst stávajících přístupů k zadávání textu (viz kapitola 2.3). Dále a především pak staví na výsledcích testování vybraných komerčních řešení, jejichž podrobnému popisu jak z pohledu klíčových funkcionalit, tak z hlediska jejich hlavních nedostatků a limitací byla věnována kapitola 4. Podrobnější popis postupu analýzy některých klíčových vlastností je možné nalézt v kapitole 5.1.

5.2.1 Rozložení klávesnice Sense Keyboard

V kapitole 5.1.1 bylo jako nejvhodnější rozložení pro účely aplikace Sense Keyboard popsáno rozložení kláves 4*3. Šablonou pro návrh se tak stávají staré tlačítkové telefony. Zatím co v pozicích čísel a písmen na klávesnicích běžných výrobců v podstatě neexistuje rozdíl, jak demonstruje obrázek 5.2, jiná situace nastává u tlačítek určených pro zadávání interpunkce či mezer.

Zařízení od společnosti Nokia měla mezeru umístěnu na tlačítku s pomyslným pořadovým číslem 11, tedy na tlačítku s numerickým znakem "0". U telefonů Sony Ericsson byla mezera umístěna až na úplně posledním tlačítku v pravém dolním rohu. Siemens, zejména prvotních modelech[5], typicky umisťoval mezeru na první tlačítko nesoucí číslo "1". Analo-

[5]Například u zařízení s modelovým označením C35i či M65

Obrázek 5.2: Hardwarové klávesnice používané před rozmachem dotykových displejů [18]

gicky k tomu byla pak odlišná i pozice ostatních interpunkčních a jiných speciálních znaků, stejně jako přepínání mezi režimy zadávání a velikostí písma.

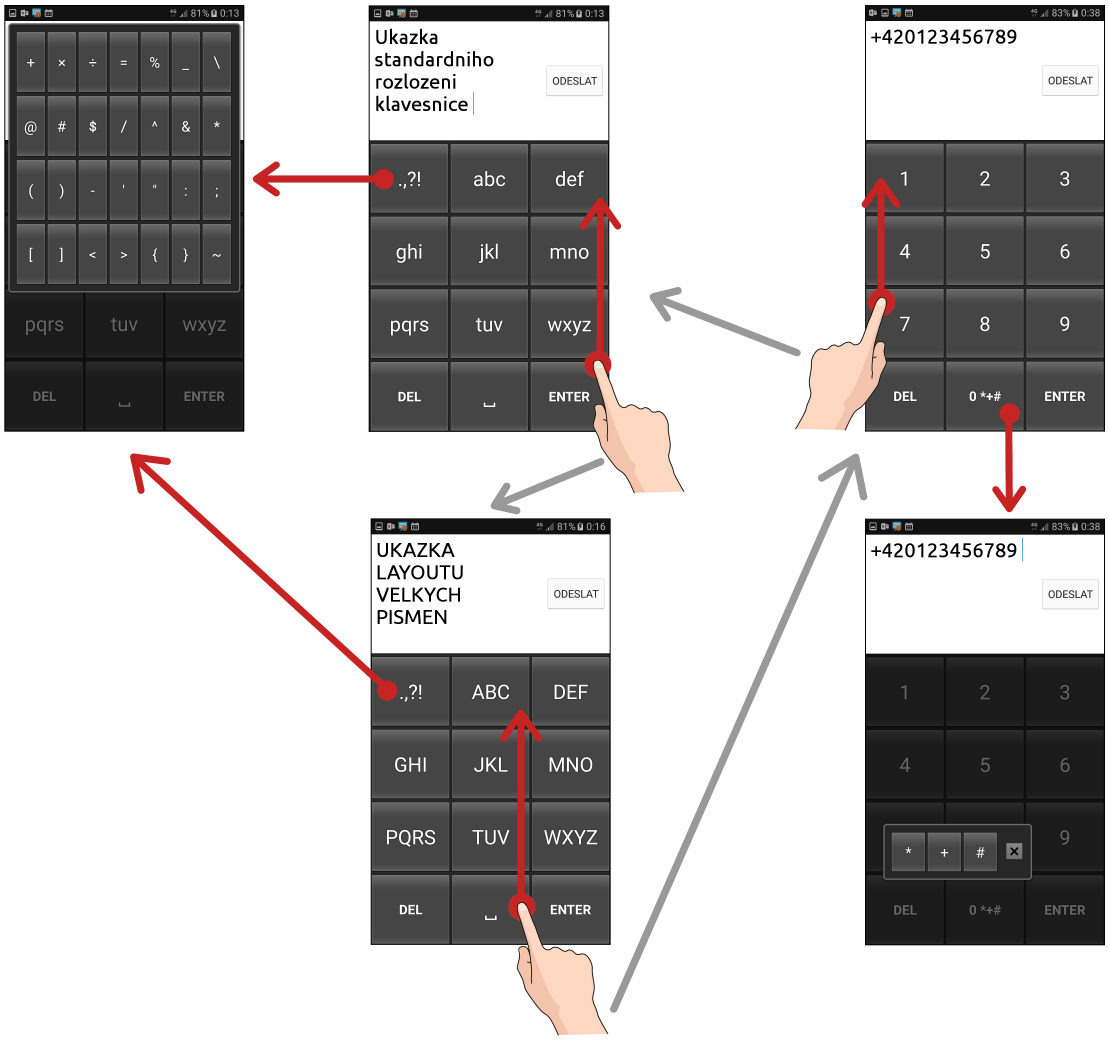
Z obrázku 5.2 také jasně vyplývá, že ačkoliv měla zařízení klávesnici rozložení 4*3, byla kromě ní vybavena dalšími pomocnými a navigačními tlačítky, dedikované klávesy pro mazání a potvrzování nevyjímaje. Aby mohla zůstat klávesnice Sense Keyboard skutečně klávesnicí pouze 4*3 a nabídnout tak co největší plochu každé jednotlivé klávesy, musí být logika z původně samostatných hardwarových kláves vhodně zaintegrována do nového návrhu.

Jedno z tlačítek, u telefonů Nokia typicky to se znakem "#", bylo určeno k přepínání mezi jednotlivými režimy zadávání vstupu[6]. Při dlouhém stisku pak skrývalo několik speciálních znaků. Pro změnu režimu zadávání je na dotykových zařízeních možno využít gesta, speciální znaky ze všech tlačítek integrovat do jednoho samostatně se otevírajícího okna (popup) a tím si uvolnit až 2 tlačítka pro nutné akce, kterými jsou mazání a potvrzování zadaného textu.

Návrh řešení je zachycen na obrázku 5.3, kde jsou znázorněna všechna 3 v současné době podporovaná rozložení kláves. Dále jsou zde vyobrazena okna pro výběr speciálních znaků včetně tlačítek, kterými je lze vyvolat. Schéma rovněž zobrazuje postup a pořadí při přepínání mezi jednotlivými rozloženími pomocí gest, viz podrobnější rozbor jednotlivých layoutů:

Základní layout – Defaultní rozložení, ze kterého vycházejí i všechna následující. Obsahuje matici 4 kláves vertikálně a 3 horizontálně. Obsahuje základní sadu písmen bez diakritiky, která jsou rozdělena do skupin, vždy 3-4 písmena na jedno tlačítko. Počátek abecedy je umístěn do středu horní řady a její konec v pravém sloupci třetí

[6]Obvykle malá písmena, velká písmena a číslice, případně režim T9



Obrázek 5.3: Návrh rozložení jednotlivých layoutů klávesnice Sense Keyboard [18]

řady, zachovává tedy stejné rozložení písmenných znaků, jako standardní klávesnice demonstrované na obrázku 5.2. Specifika jsou následující:

- Oddělovač slov (mezerník) se nachází v prostředku spodní řady. To umístění je zvoleno záměrně vzhledem k tomu, že ze zbylé skupiny (speciální znaky, enter, delete, mezerník) je právě poslední jmenovaný při psaní používaný nejčastěji a pozice ve středu dole je při základním svislém držení zařízení jednou rukou nejpohodlnější pro stisk.
- Funkční klávesy pro mazání (DEL) a potvrzení[7] (ENTER) zadaného textu jsou

[7] V závislosti na režimu zadávání může být funkce tohoto tlačítka i vložení nového řádku. Toto chování se vyskytuje zejména v základních textových polích. Ve vyhledávačích a vstupních textových boxech formulářů je očekávané chování této klávesy potvrzení.

lokalizována po obou stranách mezerníku a vytváří tak spodní řadu kláves bez grafického výstupu.

- Posledním tlačítkem je v pořadí první, nacházející se v levé horní části klávesnice. Zde jsou umístěny základní znaky pro větnou stavbu, tedy ".,?!". Kromě nich se po dlouhém stisku této klávesy zobrazí obsáhlý panel speciálních znaků, ze kterého je možné vybrat požadovaný méně častý znak.
- Přejetím po klávesnici prstem směrem odspodu vzhůru (gesto) dojde k přepnutí rozložení na layout velkých písmen.

Layout velkých písmen - V podstatě identický se základním layoutem, pouze písmena v něm obsažená jsou všechna velká.

- Přejetím po klávesnici prstem směrem odspodu vzhůru (gesto) dojde k přepnutí rozložení na layout numerických znaků.
- Pokud nedojde k přepnutí rozložení pomocí gesta, zůstává layout velkých písmen aktivní po celou dobu zadávání vstupu a simuluje tak psaní s aktivním režimem Caps Lock.
- Z důvodu úspory jednoho typu gesta nelze přepnout zpět na základní layout přímo, ale pouze přes numerický layout. Toto chování je nicméně identické se stavem na hardwarových 4*3 klávesnicích.

Numerický layout - Rozložení sloužící pro zadávání číselných znaků

- Přejetím po klávesnici prstem směrem odspodu vzhůru (gesto) dojde k přepnutí rozložení na základní layout (opět nelze přímo na layout velkých písmen).
- Pro co možná nejpodobnější rozložení těm známým z hardwarových 4*3 klávesnic bylo vyskakovací okno speciálních znaků přesunuto ke tlačítku "0"do středu spodní řady, odkud je možné jej vyvolat dlouhým stiskem. V tomto případě se jedná o výrazně méně obsáhlý seznam, než v případě předchozích dvou layoutů. Zde je výčet omezen pouze na znaky nacházející se standardních numerických klávesnicích, konkrétně "* + #".

5.2.2 Uživatelské rozhraní pro konfiguraci

Z rozboru provedeného v kapitole 5.1.3 vyplývá, že bezprostředně po instalaci Sense Keyboard bude muset uživatel provést několik konfiguračních kroků v systémové části Androidu tak, aby bylo možné klávesnici začít používat. Tato nastavení lze provést nezávisle na uživatelském rozhraní aplikace, tedy přímo v systému, ale jedná se o uživatelsky nepřívětivý postup. O to více, že klávesnice je cílena na podporu nevidomých, pro které je navigace

v pro ně dosud neznámých částech systémové konfigurace ještě obtížnější, než pro běžného uživatele.

Aplikace je tedy vybavena vlastním uživatelským rozhraním tvořeným sadou aktivit[8], které se zobrazí jednorázově bezprostředně po instalaci aplikace a které tvoří průvodce konfigurací. V současné době se skládá ze tří samostatných aktivit, které na sebe vzájemně navazují. Po spuštění wizardu se zobrazí první aktivita, která obsahuje úvodní informace o aplikaci. Poslední v řadě se nachází aktivita, která nabízí přímý vstup do systémového nastavení. V něm je po instalaci nutné klávesnici povolit (aktivovat) a následně ji nastavit jako primární vstupní metodu pro zadávání textu. Z drátového modelu znázorněného na obrázku 5.4 je vidět nejen propojení jednotlivých obrazovek (aktivit), ale také jejich struktura.

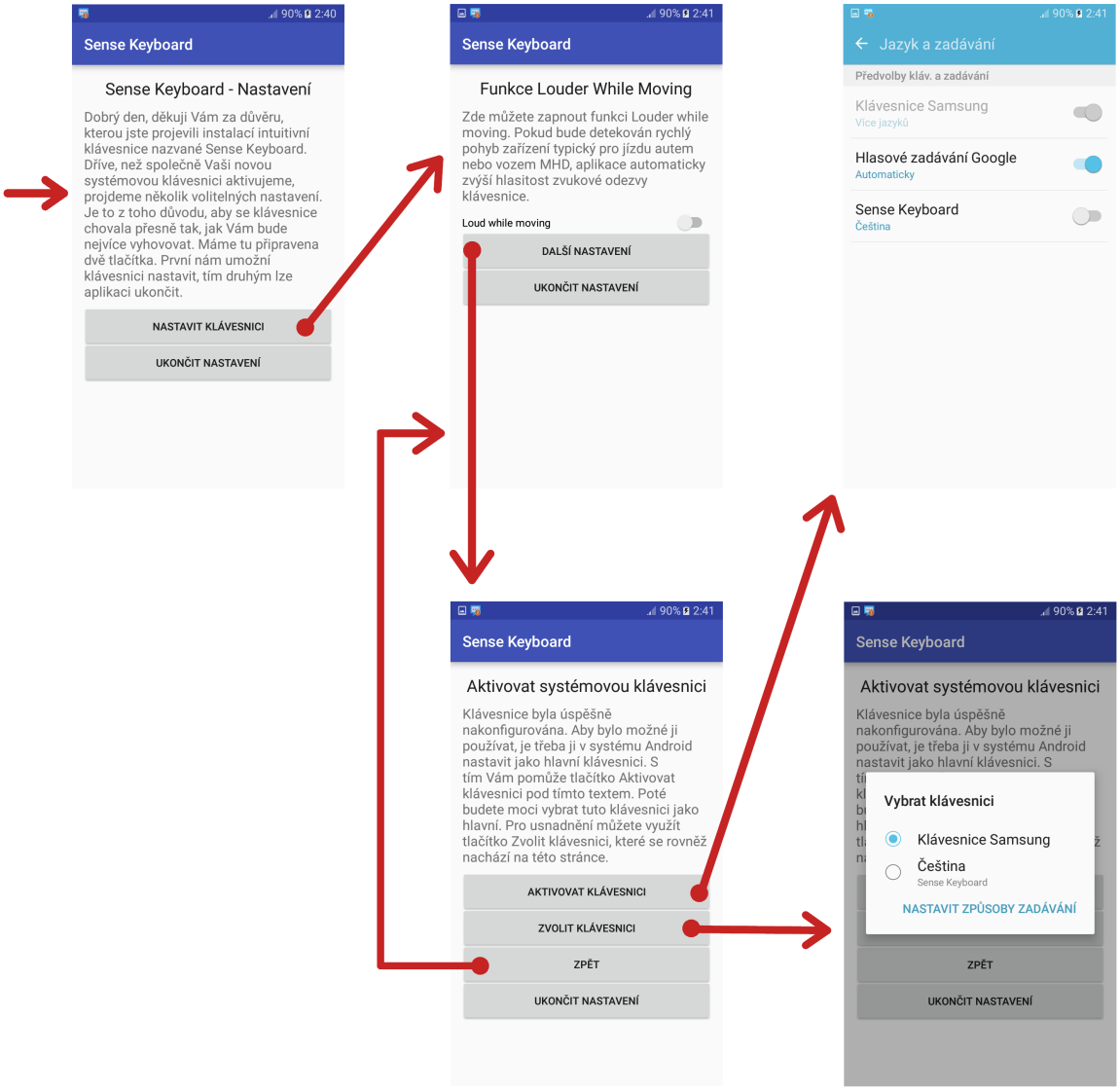
Byl zvolen inovativní přístup, kdy jsou použity delší textové popisky každého z „oken“ aplikace. Je to z toho důvodu, aby bylo uživateli odkázanému na odečítač obrazovky vždy na začátku sděleno, jaké prvky se na dané stránce zobrazují a co je možné s nimi nastavit. V případě potřeby je možno v budoucnu části textu, které jsou relevantní pouze pro Talkback (souhrnné popisy výskytů tlačítek na stránce), snadno skrýt a nechat je k dispozici pouze odečítači. V první fázi byl zvolen přístup, kdy aplikace své zaměření cíleně prezentuje.

Součástí wizardu je i nastavení základních feature. Jelikož většina funkcionalit usnadnění práce, které Sense Keyboard nabízí, spadá do kategorie změny chování bez očekávaného nežádoucího efektu (odpadá případná potřeba jejich deaktivace), je v tuto chvíli volitelná pouze jedna experimentální feature - LWM[9]. Všechna další nastavení a vylepšení, u kterých bude v budoucnu dávat smysl jejich konfigurace, budou součástí tohoto wizardu a očekává se jejich řetězení mezi první a poslední aktivitu.

Jelikož wizard je k dispozici pouze při poinstalační konfiguraci, musí aplikace podporovat i standardní způsob úpravy vlastností vstupní metody a tím je systémové nastavení. Obrázek 5.1 ukazuje zmíněné rozhraní pro základní systémovou klávesnici zařízení Samsung J5 (2016). Touto cestou je uživateli umožněno v případě změny preferencí v chování klávesnice dodatečně upravit parametry zadané pomocí wizardu.

[8] Třída v Android frameworku, více v kapitole 6.1.1

[9] Louder While Moving = hlasitější během pohybu, viz kapitola 6.3.5



Obrázek 5.4: Drátový model konfiguračního wizardu aplikace Sense Keyboard [18]

Kapitola 6

Implementace

V této kapitole jsou diskutována konkrétní implementační řešení wizardu, systémové konfigurace, samotné vstupní metody (klávesnice) i realizovaných rozšíření. Kapitola čerpá teoretický základ z [17] a [16].

6.1 Wizard a systémová konfigurace

Jelikož aplikace plní svůj primární účel systémové vstupní metody zadávání textu, konfigurační a uživatelské rozhraní není přímo součástí jejího UI. Aplikace je vybavena wizardem pro iniciální nastavení bezprostředně po instalaci a dále pak standardním konfiguračním rozhraním v systému Android, jak bylo popsáno v kapitole 5.2.2. Každá část vyžaduje svou samostatnou implementaci a je provedena odlišným způsobem.

6.1.1 Průvodce prvním nastavením - wizard

Wizard je tvořen sérií potomků třídy `AppCompatActivity`, kteří jsou zřetězeni a vzájemně propojeni pomocí tlačítek `Button`. Při stisku tlačítka pro přesun na další aktivitu je vytvořen explicitní `Intent`. Intent je v podstatě zapouzdření objektu, které umožňuje jeho předání například jinému vláknu a spuštění jako samostatnou proceduru.

```
   ...
Button nextButton = (Button)findViewById(R.id.main_settings_next_button);
if (nextButton != null) {
    nextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(SenseKeyboardSettingsInit.this,
                    SenseKeyboardSettingsLwm.class);
            intent.setFlags(intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
```

```
            finish();
        }
    });
}
```

Stejně jako `Button`, i všechny další prvky uživatelského rozhraní jsou potomky třídy `View`. To umožňuje jejich organizaci do skupin (`ViewGroup`), a layoutů. Ukázka popisu layoutu úvodní aktivity wizardu pomocí XML je v příloze B.2. Takto poskládaný grafický obsah je možno zobrazit v aktivitě následujícím příkazem (umístěným v `onCreate()` metodě této aktivity):

```
setContentView(R.layout.activity_main_settings_init);
```

Která aktivita bude spuštěna jako první po instalaci aplikace (případně jejím spuštění) lze ovlivnit nastavením parametrů překladače. Sense Keyboard je kompilována tak, aby použila defaultní aktivitu. Rozhodnutí, která aktivita bude defaultní a tudíž bude spuštěna jako první, se tedy odehrává v souboru `AndroidManifest.xml`. První aktivita definovaná uvnitř tagu aplikace, která má korektně nastaveny parametry intentu pro spuštění, se považuje za defaultní:

```
   ...
<application ...>
    <activity android:name=".SenseKeyboardSettingsInit">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
...
```

Každý prvek uživatelského rozhraní je možno (v případě definování pomocí XML) opatřit tagem `android:contentDescription` či mu (pokud je dynamicky generován v Java kódu) nastavit atribut `setContentDescription(CharSequence)`. Tato akce zajistí potřebnou odezvu odečítače. U některých prvků, jako textová pole nesoucí vlastní popis ve svém obsahu, toto není nezbytně nutné, přesto je to považováno za best practice. U jiných elementů, kterými jsou například `ImageView` nebo `checkbox` se jedná naopak o nutnost. U prvků, které jsou seskupeny do `ViewGroup` je možno definovat jak popis každého zvlášť, tak i souhrnný popis pro celou skupinu.

Pokud UI obsahuje prvky, které jsou pro uživatele z pohledu orientace v obsahu nepodstatné, je možno nastavit jejich ignorování při průchodu obsahem obrazovky, které nevidomí využívají buď gesty (následující/předchozí prvek UI) nebo speciálními hardwarovými zařízeními (externí klávesnice/brailský řádek, trackball, ...). Ignorování prvku je možno docílit

atributem `android:isImportantForAccessibility="no"`. Případně je možno znemožnit prvku zaměření (focus) zcela, k čemuž slouží tag `android:focusable=false` nebo příkaz `setFocusable(NOT_FOCUSABLE)`.

V neposlední řadě je možno (od API 22+) předem definovat logické pořadí, ve kterém se budou prvky UI při sekvenčním průchodu uživateli nabízet. Provádí se dvojicí tagů `android:accessibilityTraversalBefore` a `android:accessibilityTraversalAfter`. Tyto jako parametr přijímají číselný identifikátor předchozího/následujícího prvku uživatelského rozhraní.

Jelikož účelem wizardu je nastavení některých parametrů (a tím změna chování) dle přání uživatele, hodnoty těchto parametrů je nutné centralizovaně uchovávat. K tomu lze v Androidu využít `SharedPreferences`. Do nich může každá aplikace ukládat svá sdílená nastavení. Klíčem je identifikátor typu `String` a jako hodnoty mohou být uchovány základní datové typy (`int, char, boolean,` ...), které mohou být následně za použití zmíněného identifikátoru kdykoliv opět vyčteny. Základní práce se sdíleným nastavením je následující:

```
   ...
// get default preferences connection
SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences
        (getApplicationContext());
// get stored value from preferences
Boolean lwmFeatureActive = sharedPref.getBoolean(getString
        (R.string.LWMFeatureId), defaultLVMValue);
...
// edit preference and store it again
SharedPreferences.Editor editor = sharedPref.edit();
editor.putBoolean(getString(R.string.LWMFeatureId), isChecked);
editor.commit();
...
```

Wizard také usnadňuje uživateli aktivaci a zvolení metody zadávání vstupu. Zatím co prvního je možné dosáhnout pomocí dříve popsané třídy `Intent`, pro volbu metody zadávání má Android vestavěný widget, který je možno zobrazit v libovolné aplikaci.

```
   ...
// show input system settings screen
startActivityForResult(new Intent(android.provider.Settings
        .ACTION_INPUT_METHOD_SETTINGS), 0);
...
// show widget for input method selection
Context context = getApplicationContext();
InputMethodManager imeManager = (InputMethodManager) context
        .getSystemService(Context.INPUT_METHOD_SERVICE);
imeManager.showInputMethodPicker();
...
```

6.1.2 Systémová konfigurace parametrů

Jelikož je Sense Keyboard systémovou servisou pro zadávání vstupu, náleží jí neoddělitelně místo i v systémovém nastavení metod zadávání vstupu (více viz kapitola 5.1.3). Aby však bylo možné z tohoto místa konfigurovat volitelné parametry aplikace stejně, jako je tomu ve wizardu, je nutná implementace další logiky.

Stejně jako všechny aktivity, poskytovatelé (provider) a příjemci (receiver), i servisy musí být deklarovány v souboru `AndroidManifest.xml`. Kromě toho, že je zde hlavní servisa definována s oprávněním `BIND_INPUT_METHOD` a intent filtrem `InputMethod`, byla přidána ještě část `<meta-data android:name="android.view.im" ...>`, která odkazuje na XML soubor s popisem servisy. XML soubor musí obsahovat tag `<input-method>` s atributem `android:settingsActivity="..."`, jež odkazuje na potomka třídy `PreferenceActivity`, v implementaci Sense Keyboard je to třída `ImeSystemPreferencesActivity`. Zde je možno v přepsané metodě rodiče `public Intent getIntent()` přidat do intentu další části, nazývané `Fragment`:

```
   ...
public class ImeSystemPreferencesActivity extends PreferenceActivity {
    @Override
    public Intent getIntent() {
        final Intent modIntent = new Intent(super.getIntent());
        modIntent.putExtra(EXTRA_SHOW_FRAGMENT, Settings.class.getName());
        modIntent.putExtra(EXTRA_NO_HEADERS, true);
        return modIntent;
    }

    public static class Settings extends InputMethodSettingsFragment {
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setInputMethodSettingsCategoryTitle(R.string
                    .languageSelectionTitle);
            setSubtypeEnablerTitle(R.string.selectLanguage);

            // Load the preferences from an XML resource
            addPreferencesFromResource(R.xml.ime_preferences);
        }
    }
...
```

Odkazovaný XML soubor `ime_preferences.xml` pak obsahuje vlastní vzhled obrazovky nastavení. V případě Sense Keyboard se jedná i zde, stejně jako ve wizardu, o featuru LWM

(viz kapitola 6.3.5). Je zde tedy umístěn XML tag pro preference checkbox:

```
<CheckBoxPreference android:key="@string/LWMFeatureId" ...>
```

Atribut `android:key` je velice důležitý, neboť s jeho pomocí je dosaženo provázání mezi hodnotou tohoto checkboxu a hodnotou nastavenou pomocí chechboxu wizardu.

6.2 Vstupní metoda - systémová klávesnice

`SenseKeyboardService` je dědičnou implementací třídy `InputMethodService` a zároveň implementuje rozhraní `KeyboardView.OnKeyboardActionListener`. To společně s náležitou deklarací v `AndroidManifest.xml` zajišťuje, že tato servisa je hlavní servisou způsobilou k příjmu eventů o stisku fyzických tlačítek a doteku displeje, které registrovaným vstupním servisám zasílá systém.

6.2.1 Layout klávesnice, rozložení tlačítek

Každá systémová klávesnice je tvořena sadou tlačítek. V případě Sense Keyboard tvoří layout klávesnice matice 4x3 tlačítek (více v kapitole 5.2.1). Layout neboli rozložení, v našem kontextu rozložení klávesnice zobrazované uživateli na obrazovce mobilního zařízení a sloužící k zadávání vstupních dat, je definováno pomocí XML souboru `keys_positions.xml`.

V souboru jsou specifikovány řádky displeje. Každý řádek v sobě může ukrývat 1 až N kláves označených tagem `<Key>`. Pomocí atributů je pak pro každou klávesu možno definovat číselný kód, který bude předán systému po stisku této klávesy, dále její popisek zobrazený jako text (obvykle znaky písmen, které klávesa reprezentuje), pozice/ukotvení na obrazovce (levý dolní roh, pravý dolní roh, ...) či vlastnost opakovatelnosti. Opakovatelnost znamená, že při dlouhém stisku klávesy dochází k jejímu opakovanému volání do systému. Využívá se nejčastěji u kláves pro mazání textu či vkládání mezer. Zkrácená ukázka XML souboru `keys_positions.xml` pro definování rozložení kláves:

```
   ...
<Keyboard xmlns:android="http://schemas.android.com/apk/res/android"
    android:keyWidth="33.33333%p"
    android:keyHeight="20%p">
    <Row>
        <Key android:codes="46,44,63,33"
             android:keyLabel=".,\?!"
             android:keyEdgeFlags="left"/>
        <Key .../>
        <Key android:codes="100,101,102"
             android:keyLabel="def"
             android:keyEdgeFlags="right"/>
```

```
    </Row>
</Keyboard>
```

Z analýzy provedené v kapitole 5.1.1 a následného návrhu popsaného v kapitole 5.2.1 vyplývá potřeba co největší plochy klávesnice a jednotlivých tlačítek, ze kterých je složena. Jak bylo demonstrováno výše, pomocí XML souboru `keys_positions` je možno dosáhnout v podstatě jakéhokoliv rozložení z hlediska velikosti i umístění kláves, například i přes celý display zařízení.

Zde ovšem nastává zásadní problém, a to se vstupním polem, tedy prvkem do kterého zadáváme vstupní text. Při zobrazení klávesnice přes celý displej, nebo přes jeho dominantní část, může dojít k celému nebo částečnému zakrytí tohoto vstupního pole a to i přesto že systém provede při zobrazení klávesnice přepočet pozice prvků na obrazovce a pokusí se zobrazení vstupního pole optimalizovat do prostoru, kde klávesnice zobrazena není. V praxi by to mohlo vést k situaci, kdy bude pomocí klávesnice zadáván text, který nebude po dobu zobrazení klávesnice k dispozici pro vizuální kontrolu. Toto by pravděpodobně nebyl problém pro nevidomé uživatele, ovšem Sense Keyboard by měla být vhodná i pro lidi se zbytky zraku, s méně závažným zrakovým hendikepem a v neposlední řadě i pro uživatele bez zrakového omezení. Pro ně by již toto omezení bylo zásadně limitující.

K řešení problému zakrytých vstupních polí bylo nakonec zvoleno systémem podporované řešení. V případě, že je telefon orientován "na šířku" dochází obvykle k automatickému přepnutí vstupního pole do tzv. extrahovaného pohledu. To znamená, že vstupní pole je spolu se svým obsahem vyextrahováno z původního `View` (je zreplikován jeho obsah) do nového pole, které je umístěno zpravidla bezprostředně nad klávesnicí a utváří spolu s ní zobrazení v režimu tzv. celé obrazovky. Tím je eliminována možnost nežádoucího zakrytí původního textového vstupního pole.

Již bylo zmíněno, že standardně je režim celé obrazovky dostupný pouze v situaci, kdy je zařízení orientováno na šířku. Jelikož je třída `SenseKeyboardService` potomkem třídy `InputMethodService`, může přepsat metodu `onUpdateExtractingVisibility()`. Tato metoda je volána frameworkem ve chvíli, kdy má být po změně vyhodnoceno, zda se klávesnice a textové pole zobrazí ve standardním režimu, nebo v režimu celé obrazovky. Následující příkaz zavolaný v těle této funkce způsobí, že se aplikace zobrazí vždy v režimu celé obrazovky:

```
setExtractViewShown(true);
```

Přepsáním těla metody `public View onCreateExtractTextView()` je poté možno upravit parametry zobrazeného extrahovaného pole, například velikost textu. Ta je důležitá zejména pro osoby se zhoršeným zrakem, proto Sense Keyboard provádí její umělé navýšení.

Extrakci obsahu vstupního pole tedy zajistí systém sám, ale pokud původní vstupní pole obsahovalo například našeptávač, k přenosu jeho hodnot již nedojde. Je to z důvodu,

že extrahováno je skutečně pouze samotné vstupní pole, zatím co našeptávač je obvykle realizován formou popupu zobrazeného pod tímto extrahovaným textovým polem. Sense Keyboard řeší tento problém pomocí tzv. `CandidatesView`, které je podrobněji popsáno v kapitole 6.3.4.

6.2.2 Zachytávání a zpracování stisku tlačítek

Ještě před zahájením vlastního odchytávání kláves se v metodě `onStartInput()` na základě obdrženého parametru vyhodnotí typ vstupního pole. Vstupních polí totiž existuje několik druhů:

- `EditorInfo.TYPE_CLASS_NUMBER`
- `EditorInfo.TYPE_CLASS_DATETIME`
- `EditorInfo.TYPE_CLASS_PHONE`
- `EditorInfo.TYPE_CLASS_TEXT`
  - `EditorInfo.TYPE_TEXT_VARIATION_PASSWORD`
  - `EditorInfo.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD`
  - `EditorInfo.TYPE_TEXT_VARIATION_EMAIL_ADDRESS`
  - `EditorInfo.TYPE_TEXT_VARIATION_URI`
  - `EditorInfo.TYPE_TEXT_FLAG_AUTO_COMPLETE`
  - ...

V závislosti na typu vstupního pole jsou nastaveny vnitřní proměnné aplikace, kterými jsou `mDisplayComposingText` (vysvětlena dále v této kapitole), `mIsBeginningOfNewSentence` (viz kapitola 6.3.3) a `mActiveKeyboardInt`, která má za následek volbu jiného rozložení (obrázek 5.3). Pro `TYPE_CLASS_PHONE` tak bude k dispozici rozložení klávesnice s čísly, zatím co pro `TYPE_CLASS_TEXT` rozložení s písmeny.

K zachytávání stisku tlačítek, respektive události (event) tuto akci provázející, jsou v třídě `InputMethodService` k dispozici tři metody:

```
   ...
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {}
```

```
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {}

@Override
public void onKey(int primaryCode, int[] keyCodes) {}
```

Zatím co první dvě výše uvedené metody jsou určeny k zachytávání událostí stisku hardwarových tlačítek, metoda `onKey()` je vstupním bodem zpracování stisku virtuální klávesy. Z deklarace funkce vyplývá, že prvním ze dvou vstupních parametrů je kód stisknutého tlačítka, konkrétně v ASCII kódování. Kódy jednotlivých tlačítek byly definovány pomocí atributu `android:codes` v souboru `keys_positions.xml`, viz ukázkový výřez v kapitole 6.2.1. Jelikož má Sense Keyboard pouze 12 tlačítek, využívá vždy jedno tlačítko pro více hodnot. Obsluha této skutečnosti[1] je řízena systémem.

Nepříznivým chováním systémové implementace je, že pokud uživatel stiskne dvakrát klávesu "abc", aby napsal znak "b", metoda `onKey()` je zavolána hned třikrát. Po prve pro znak "a", podruhé pro znak "delete" (odmazání jednoho písmena ze vstupu) a potřetí pro znak "b". Toto chování má za následek nutnost implementace workaroundů pro specifické aktivity. Například přehrání zvuku při stisku klávesy. Zde uživatel stisk dvakrát klávesu a ve zvukové odezvě by se nacházely tři signály. Ignorovat znak "delete"možné rovněž není, protože je to také regulérní znak na klávesnici a uživatel jej může kdykoliv stisknout cíleně. Jako řešení bylo v tomto případě zvoleno odložení zvukové odezvy pomocí časovače a očekávání příchodu dalšího tlačítka.

Zpracování přijatého znaku dále probíhá vnitřní logikou aplikace, kdy je znak zařazen do jedné ze 4 skupin:

Alfanumerický znak – Pokud byl nastaven příznak `mDisplayComposingText`, nevkládá se text do vstupního pole přímo, nýbrž pomocí metody `setComposingText()`. To má za důsledek, že se vytvářené slovo postupně skládá jako celek a je možné jej například celé nahradit slovem jiným (využívá se při autokorekcích a návrhu kandidátních slov apod.)

Mezera – Znak mezery se nejen že ihned vkládá přímo do textového pole, ale ještě před vlastním vložením je proveden commit všech znaků (obvykle posledního napsaného slova), které se momentálně nacházejí v composing části vstupu. Jedná se tak vlastně o potvrzení napsaného slova.

Delete – Při zpracování znaku "delete" je nastaven a spuštěn časovač pro přehrání zvuku signalizujícího stisknutí tlačítka (kvůli zasílaným systémovým událostem při výskytu

[1] Včetně definice velikosti prodlevy určující rozdíl mezi dvěma stisky klávesy "abc", kdy jednou se jedná o "aa" a podruhé o "b"

více kódů na jednom tlačítku, jak bylo popsáno výše). Dále je prověřen obsah composing části vstupu. Následně je buď odebrán jeden znak z composing (nacházelo se zde 2 a více znaků), vynulován celý composing (nacházel se zde poslední znak), nebo odmazán znak přímo ze vstupního pole (composing byl prázdný, nebo se nepoužívá).

Enter – Podrobněji viz kapitola 6.3.2.

6.2.3 Explore by touch odezva

Nezbytnou vlastností při aktivovaném odečítači obrazovky je, aby při najetí prstem na tlačítko v režimu explore by touch došlo k notifikaci uživatele, která klávesa se na dané pozici nachází. V kapitole 6.1.1 bylo popsáno, jak lze tohoto chování jednoduše docílit na všechny prvky, které jsou potomky třídy `View`. Použitá třída `Keyboard.Key` však není potomkem třídy `View`, nýbrž třídy `Object`. Automaticky tedy režim explore by touch na takto postavené klávesnici fungovat nebude.

Sense Keyboard implementuje třídu `MyKeyboardView`, která rozšiřuje původní třídu `KeyboardView` mimo jiné právě o rozpoznání klávesy, která byla uživatelem zaměřena, a o následnou zpětnou notifikaci pomocí Accessibility manageru získaného ze systémové servisy `ACCESSIBILITY_SERVICE`.

Protože při prozkoumávání klávesnice pomocí režimu explore by touch není generována událost stisku klávesy, musí být implementován jiný algoritmus zpracování. Třídá `MyKeyboardView` obdrží v metodě `onTouchEvent()` ze systému `MotionEvent`, který v sobě neobsahuje informaci o tom, která klávesa byla zaměřena. Obdrží však souřadnice, na kterých byl posun prstu detekován. S jejich pomocí je možno následujícím algoritmem získat konkrétní klávesu:

```
   ...
private String extractKeyCharFromMotionEvent(MotionEvent event) {
    String pressedKeyChar = "";
    float fx = event.getX();
    float fy = event.getY();
    int x = Math.round(fx);
    int y = Math.round(fy);

    Keyboard keyboard = super.getKeyboard();
    List<Keyboard.Key> keysList = keyboard.getKeys();
    for(Keyboard.Key key : keysList) {
        if(key.isInside(x, y)) {
            char code = (char)key.codes[0];
            pressedKeyChar = String.valueOf(code);
            break;
        }
```

```
    }
    return pressedKeyChar;
}
...
```

`onTouchEvent()` se však volá kontinuálně po celou bodu, kdy je aktivní režim explore by touch, zatím co uživatel nemusel během zkoumání vůbec opustit oblast jedné klávesy. Proto je algoritmus dále rozšířen tak, aby notifikoval vždy pouze při přechodu na novou klávesu. Samotná zvuková notifikace s názvem právě zaměřeného znaku je pak odečítači předána následujícím způsobem:

```
   ...
mAccessibilityManager = (AccessibilityManager) context.
        getSystemService(Context.ACCESSIBILITY_SERVICE);
AccessibilityEvent e = AccessibilityEvent.obtain();
e.setEventType(AccessibilityEvent.TYPE_ANNOUNCEMENT);
e.setClassName(getClass().getName());
e.setPackageName(this.getContext().getPackageName());
e.getText().add(accessibilityText);
mAccessibilityManager.sendAccessibilityEvent(e);
...
```

6.3 Realizovaná rozšíření

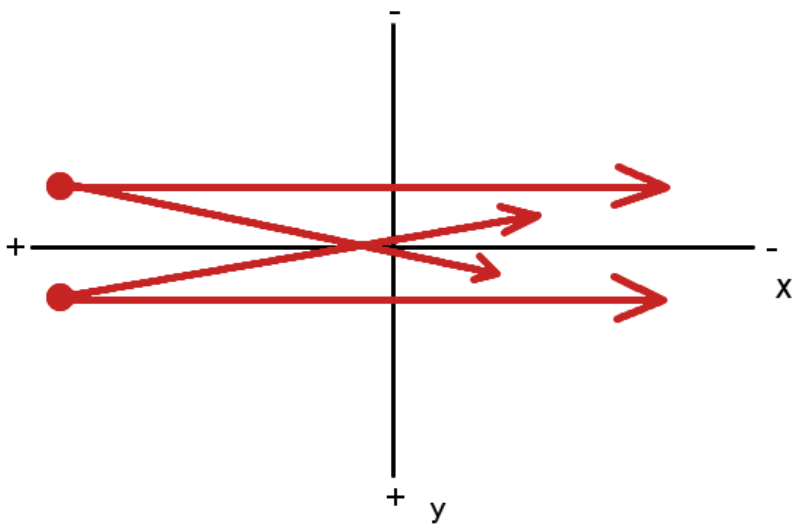
Vzhledem k účelu použití aplikace a faktu, že každému uživateli vyhovuje poněkud odlišný způsob interakce, se nabízí velice široká škála možných rozšíření a parametrizovatelných chování klávesnice. V tuto chvíli jsou implementovány ty popsané v kapitole 5.1.2 *Technologie podporující efektivní zadávání textu* + některá další.

6.3.1 Detekce a použití gest

Pro automatické získávání systémových událostí spojených s gesty stačí servisní třídě implementovat rozhraní `KeyboardView.OnKeyboardActionListener`. Pak je tato notifikovaná o akcích jako `swipeDown()`, `swipeLeft()`, `onPress()` apod. Během testování se však tato notifikace ukázala být nepříliš přesná. Některá gesta byla z neznámého důvodu ignorována a tak se přesnost detekce pohybovala i u nejzákladnějších gest pouze okolo 60%. Paleta gest je navíc poměrně omezená a tak Sense Keyboard obsahuje vlastní třídu `MyGestureListener`, jež implementuje rozhraní `GestureDetector.OnGestureListener` a `GestureDetector.OnDoubleTapListener`.

Zpracování gest v hlavní servisní třídě však v jednom případě stále použité zůstává. Je jím gesto rychlého pohybu prstem po displeji směrem odspodu nahoru. To vyvolává událost `swipeUp()`. Na "švihnutí vzhůru", jak je gesto také někdy označováno, je vázána akce

přepnutí rozložení klávesnice z normálního na velká písmena, dále pak z velkých písmen na čísla a z čísel opět na normální rozložení, jak znázorňuje obrázek 5.3. Toto gesto bylo zachováno z důvodu, že pro změnu `View` (layout klávesnice) musí být akce vyvolána ve stejném threadu, ve kterém bylo `View` vytvořeno. V případě využití třídy `MyGestureListener` tomu tak není, a to díky systémovému callbacku `GestureDetector.onTouchEvent(event)`, přes který jsou ve třídě `MyGestureListener` volány metody odpovídající detekovaným gestům.



Obrázek 6.1: Všechny přípustné stavy pro gesto přejetí vpravo [18]

Další použitá gesta:

Zleva doprava – Přesun na další slovo v listu navrhovaných slov (viz kapitola 6.3.4).

Zprava doleva – Přesun na předchozí slovo v listu navrhovaných slov.

Z vrchu dolů – Potvrzení zvoleného slova z listu navrhovaných slov a jeho nahrazení za text obsažený v composing (viz kapitola 6.2.2).

K detekci výše zmíněných gest ve třídě `MyGestureListener` je využita callback metoda `onFling()`. Ta je volána po skončení pohybu prstem do displeji a obsahuje mimo jiné rychlosti pohybu prstu po osách x a y měřeno v pixelech za sekundu. Za zmínku stojí, že polarita je na osách obrácena oproti běžně uváděného souřadnému systému. Algoritmus rozpoznání gesta vychází z obrázku 6.1 a je implementován následujícím způsobem:

```
   ...
float absVelocityX = Math.abs(velocityX);
float absVelocityY = Math.abs(velocityY);

if(velocityX > 0 && absVelocityX > absVelocityY) {
```

```
    mMyKeyboardView.onSwipeRightGesture();
} else if(velocityX < 0 && absVelocityX > absVelocityY) {
    mMyKeyboardView.onSwipeLeftGesture();
} else if(velocityY > 0 && absVelocityX < absVelocityY) {
    mMyKeyboardView.onSwipeDownGesture();
} else if(velocityY < 0 && absVelocityX < absVelocityY) {
    //mMyKeyboardView.onSwipeUpGesture();
}
```

Pokud se jedná například o pohyb zleva doprava, pak hodnota y může nabývat kladných i záporných hodnot, ale vždy musí být v absolutní hodnotě menší, než hodnota x. Zatím co hodnota x musí být v tomto případě vždy větší než 0.

6.3.2 Změna chování klávesy enter

Stejně, jako je možno měnit rozložení klávesnice podle toho, jaký typ vstupního pole je momentálně editován (bylo popsáno v kapitole 6.2.2), je možno na základě stejné logiky měnit i akce klávesy "Enter".

Při zadávání textu v aplikaci pro psaní SMS je očekávaná akce odřádkování. Pokud však tlačítko "Enter" stiskneme během manipulace s webovým prohlížečem v okamžiku změny URL v adresovém řádku, případně zadání názvu aplikace ve vyhledávači aplikace Google Play, je odřádkování akcí zcela nevhodnou. Místo ní se jako smysluplná jeví akce potvrzující ukončení zadávání vstupu a zahájení vyhledávání. Chování tlačítka "Enter" by tedy mělo být kontextově závislé.

Kontext vstupního pole, pro které je klávesnice otevírána, můžeme detekovat v metodě `onStartInputView()` třídy `SenseKeyboardService`. Akce, jež má klávesnice provést, se však vyhodnocuje v metodě `onKey()`. Při implementaci tedy bylo nutné zavést nový atribut, do kterého je po zobrazení klávesnice zaznamenán typ pole `IME_ACTION_`, ve kterém bylo zadávání vstupu zahájeno. V závislosti na hodnotě tohoto atributu je následně při detekci stisknutí klávesy "Enter" v metodě `onKey()` použita odpovídající akce podle následující logiky:

```
   ...
switch (mEditorInfoImeAction) {
    case EditorInfo.IME_ACTION_GO:
        inputConnection.performEditorAction(EditorInfo.IME_ACTION_GO);
        break;
    case EditorInfo.IME_ACTION_NEXT:
        inputConnection.performEditorAction(EditorInfo.IME_ACTION_NEXT);
        break;
    case EditorInfo.IME_ACTION_SEARCH:
        inputConnection.performEditorAction(EditorInfo.IME_ACTION_SEARCH);
        break;
```

```
    case EditorInfo.IME_ACTION_SEND:
        inputConnection.performEditorAction(EditorInfo.IME_ACTION_SEND);
        break;
    case IME_ACTION_DEFAULT_LOCAL:
        inputConnection.sendKeyEvent(new KeyEvent(KeyEvent.ACTION_DOWN,
                KeyEvent.KEYCODE_ENTER));
        break;
    default:
        Log.i(CLASS_NAME_STRING,"onKey-DEFAULT-UNKNOWN ACTION!!");
        break;
}

...
```

6.3.3 Automatická změna velikosti písma

Hlavním smyslem této funkcionality je usnadnit zadávání textu uživateli. Ten již nemusí měnit rozložení klávesnice na velká písmena na začátku každé nové věty. Aplikace sama detekuje, že posledním (nebo předposledním v případě mezery) znakem ve vstupním poli před kurzorem je tečka, otazník či vykřičník. Případně, že je vstupní pole doposud prázdné či obsahuje pouze bílý znak (mezeru, enter). Algoritmus sám o sobě není nijak překvapivý. O něco zajímavější je výčet situací, které musí být pro správnou funkčnost této featury ošetřeny:

- Při zahájení vstupní metody podle typu vstupního pole nastavit příznak velkého písmene na `true` pro typ vstupního pole `TYPE_CLASS_TEXT`, který není `PASSWORD`, `EMAIL_ADDRESS`, `URI` ani `AUTO_COMPLETE`. Pro všechny ostatní typy pole nastavit příznak velkého písmene na `false`.
- Po zadání znaku ”.”,”?” nebo ”!” nastavit příznak velkého písmene na `true`, pro jiné znaky nastavit na `false`.
- Po odmazání znaku zkontrolovat, zda předcházející znak není ”.”,”?” nebo ”!”. Pokud ano, nastavit příznak velkého písmene na `true`, jinak na `false`.
- Po změně pozice kurzoru zkontrolovat, zda předcházející znak není ”.”,”?” nebo ”!”. Pokud ano, nastavit příznak velkého písmene na `true`, jinak na `false`.
- ...

Při každé změně příznaku velkého písmene je třeba změnit rovněž rozložení klávesnice na velká písmena a postoupit tuto skutečnost zároveň odečítači. Tím je zajištěna notifikace uživatele.

6.3.4 Našeptávač

Nezbytnou podmínkou praktické použitelnosti klávesnice, která vyplynula z analýzy v kapitole 5.1.2, je implementace našeptávače, který by zároveň zastával úlohu kontroly překlepů, což je v tomto případě standardní způsob využití.

Aby mohla hlavní servisní třída přijímat událostí od našeptávače, musí implementovat rozhraní `SpellCheckerSession.SpellCheckerSessionListener`. Následně je pak prostřednictvím callbacku přes implementovanou metodu rozhraní `onGetSentenceSuggestion` (API 14+) notifikována o odpovědích našeptávače. To ale pouze v případě, že byla našeptávači předána vstupní data od uživatele. K tomuto účelu se typicky použije obsah composing (viz kapitola 6.2.2), který je našeptávači předán takto:

```
   ...
TextServicesManager textServicesManager = (TextServicesManager)
        getSystemService(Context.TEXT_SERVICES_MANAGER_SERVICE);
SpellCheckerSession mSpellCheckerSession = textServicesManager
        .newSpellCheckerSession(null, null, this, true);
mSpellCheckerSession.getSentenceSuggestions(new TextInfo[] {
        new TextInfo(mComposing.toString())}, 3);
...
```

Zde je nutno zmínit jednu ne příliš dobře známou skutečnost, že zařízení Samsung používají nestandardní třídu spellcheckeru, která je frameworkem vrácena jako odpověď na volání `newSpellCheckerSession()`. Tato třída je podle názvu balíku implementací přímo od společnosti Samsung. Do publikování této práce se nepodařilo zjistit, jak s touto třídou pracovat, aby poskytovala požadovaný výstup. Výše popsaný postup, který byl úspěšně otestován na několika různých zařízeních, sice pro Samsung spellchecker také callback vyvolá, ale tělo odpovědi obsahuje vždy pouze dvě prázdné pozice.

Poté, co je získána odpověď našeptávače, vzniká potřeba jí uživateli vhodným způsobem prezentovat. Sense Keyboard k tomuto účelu využívá třídu `MyCandidatesView`, která je pouze rozšířením standardní třídy `View`. Její implementace byla částečně převzata z [15]. Třída je předána systému jako návratová hodnota přepsané callbackové metody z třídy `InputMethodService` a tou metodou je `public View onCreateCandidatesView()`.

`MyCandidatesView` zobrazuje uživateli navrhovaná slova stejným způsobem, který je možno vidět na obrázku 4.5. Z přijatého vstupu, kterým je seznam vhodných slov vrácený našeptávačem, provede tato třída vykreslení do řádku za sebe.

Úprava implementace spočívala zejména v možnosti snadné změny označení jednoho z nabízených slov tučným fontem. Tím je uživateli znázorněno, že je toto slovo zvoleno pro výběr. Uživatel pak může potvrzovacím gestem (přejetí prstem směrem odspodu nahoru, viz kapitola 6.3.1) nahradit tímto slovem aktuálně psaný text (obsah composing).

6.3.5 LWM Feature

LWM (Louder while moving“) znamená v doslovném překladu „hlasitější když se hýbe“. Jedná se o experimentální feature. Pokud je tato funkce aktivována, pak se pokouší detekovat pohyb zařízení. Jelikož pro LWM funkci je důležitý pouze určitý specifický druh pohybu (jízda dopravním prostředkem), jsou aplikovány filtry, jejichž cílem je eliminovat nahodilé otřesy při běžné manipulaci s mobilním zařízením. Pokud je zařízení označeno jako „v pohybu“, aplikace zvýší hlasitost médií pro lepší slyšitelnost našeptávače a zvukové signalizace stisku klávesy. Detekce pohybu je implementována servisní třídou:

```
public class SignificantMotionSensor implements SensorEventListener
```

Pokud je aplikace spuštěna na OS Android verze 4.3 a nebo vyšší (API >= 18), servisní třída se nejprve pokusí najít senzor typu `Sensor.TYPE_SIGNIFICANT_MOTION` (vistuální softwarový). Testovací zařízení Oukitel K4000 (v5.1), Samsung A3(v5.01), Lenovo A10-70L (v5.0.1) ani Samsung J5 2016 (v6.0.1) jím bohužel vybaveny nebyly, takže jeho funkcionalita není otestována. V tomto případě by se však mělo jednat o systémem podporovanou detekci rozpoznání významného pohybu zařízení, takže následná akce již spočívá pouze v triviálním nastavení příznaku pohybu. Pro něj používá Sense Keyboard member proměnnou `private boolean mDeviceInMove`. Tato je následně vyhodnocována v hlavní servisní třídě.

Když však aplikace běží na zařízení, které `Sensor.TYPE_SIGNIFICANT_MOTION` nepodporuje, přichází na řadu vlastní experimentální implementace. V ní je využit akcelerometr a aplikována vlastní logika detekce významného pohybu. Akcelerometr není k tomuto účelu zcela vhodný, protože jeho tři složky (trojrozměrný souřadný systém) dávají dohromady vektor zrychlení zařízení určitým směrem. Jejich hodnota je zatížena nejen gravitačním zrychlením, ale na testovaných zařízeních vykazoval senzor i v klidovém stavu nepřesné hodnoty (např. $x = -0.427365, y = 0.2645593, z = 9.680955$). Hodnoty byly ve své nepřesnosti konstantní. Dalším negativním faktem je, že pokud bychom se pohybovali (například ve voze MHD) neměnnou rychlostí $50km$, vektor zrychlení ve směru pohybu by byl v té chvíli nulový. Aplikace se tedy pokouší detekovat alespoň zvýšené množství otřesů z okolí. Podobu algoritmu demonstruje následující odstavec, kde je pomocí pseudo kódu popsána implementace filtrování senzorových dat (celý originální kód je pak k nalezení v příloze B.1):

1. Ze senzoru se získají hodnoty zrychlení v jednotlivých osách x, y, z.
2. Hodnota celkového vektoru zrychlení z minulého cyklu se přepíše hodnotou celkového vektoru zrychlení v kroku současném.

3. Hodnota současného celkového vektoru zrychlení se přepíše hodnotou odmocniny ze součtu druhých mocnin proměnných x, y, z.

4. Ze staré a nové hodnoty vektoru zrychlení se určí rozdíl ve zrychlení mezi předchozím a tímto krokem, značíme `delta`!.

5. Hodnotu `delta` přičteme k 0.9 násobku hodnoty proměnné `mAccel` (reprezentace kvantifikovaného pohybu).

6. Pokud hodnota `mAccel` nepřekročí hranici thresholdu (zvolenou jako 4), vyhodnocování algoritmu končí.

7. Je vyhodnoceno, zda od posledního platného přerušení uplynulo více než 2 sekundy. Pokud nikoliv, stávající přerušení je považováno za neplatné a průchod algoritmem je ukončen.

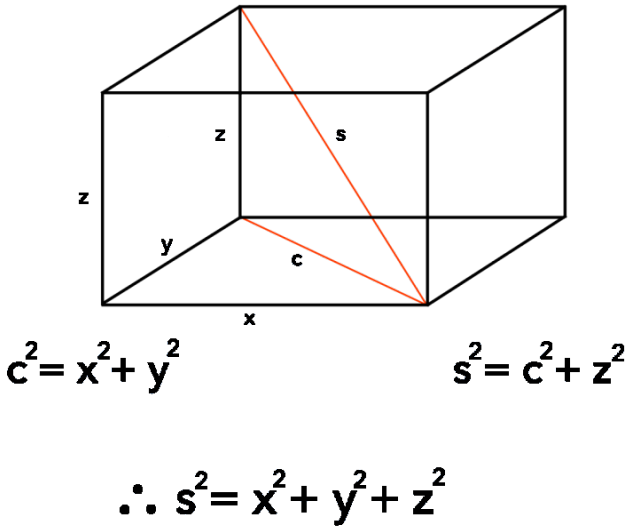
8. Pokud mezi posledním přerušením klidového stavu a současným uplynulo více než 15 sekund, je vynulován čítač počtu přerušení klidu (z kroku 6).

9. Je inkrementován čítač počtu přerušení klidu.

10. Pokud je počet přerušení klidového stavu větší nebo roven 4, vstoupí aplikace do stavu „Detekován pohyb“. Čítač počtu přerušení klidu je vynulován a je zastaven a znovu spuštěn časovač na 2 minuty.

11. Po vypršení časovače dojde ke zrušení stavu „Detekován pohyb“.

Aplikovaný algoritmus využívá výpočet velikosti trojrozměrného směrového vektoru, viz obrázek 6.2. Další proměnné, jako hranice thresholdů, hodnoty časovačů a počty opakování, vychází spíše z intuitivního přístupu doplněného sérií praktických testů.



Obrázek 6.2: Znázornění postupu výpočtu velikosti celkového směrového vektoru [9]

Algoritmus vychází z několika předpokladů:

- Pokud není možno se spolehnout na přesné hodnoty akcelerometru, ale vykazované nepřesnost je konstantní, je možné počítat pouze s rozdílem mezi jednotlivými měřeními `delta`, čímž bude odchylka senzoru eliminována.
- Když uvažujeme velikost akcelerace, ta bude z hlediska delšího časového úseku přibližně (vlivem nízké vzorkovací frekvence a zaokrouhlovacích chyb) rovna nule, protože rychlení ve směru pohybu vpřed bude následováno přibližně stejným zrychlením záporné hodnoty po ustálení rychlosti pohybu.
- Nepřesnost způsobenou nízkou vzorkovací frekvencí a zaokrouhlovací chybou je možno částečně redukovat použitím konstanty, která bude postupně utlumovat velikost zrychlení a v nekonečnu jí limitně přiblížit nule. Konstanta 0.9 se zde vyskytuje právě za tímto účelem.

Kapitola 7

Výsledky, závěr

Aplikace Sense Keyboard se nyní nachází ve fázi alfa testování. Některé dílčí části kódu tedy mohou být refaktorovat, pokud se objeví efektivnější řešení konkrétních implementačních problémů. Případně v důsledku opravy chyb průběžně odhalovaných v testovacích fázích každého softwaru. V celku se však již jezdá o ucelenou a relativně komplexní aplikaci, která uživateli poskytuje vše potřebné pro zadávání vstupu.

Během implementace se objevilo několik větších či menších technických limitací, které bylo nutno více či méně rozsáhlými workaroundy obejít. Největšími překážkami byly jednak nefunkčnost našeptávače na zařízeních Samsung, ale hlavně pak režim talkbacku. Po jeho aktivaci se totiž zásadně mění chování systému a je nutné reagovat zvukovou odezvou na režim explore by touch. Ten však koliduje s detekcí gest. Další nežádoucí chování pak vykazuje talkback při opakovaném stisku též klávesy pro zadání druhého/třetího znaku. Talkback totiž reaguje až na zvednutí prstu uživatele (do té doby je v režimu explore by touch) a tím se zvyšuje riziko, že druhé stisknutí klávesy bude interpretováno jako zadání dalšího prvního znaku této klávesy.

Většina technických problémů byla v průběhu vývoje aplikace úspěšně odstraněna. Její zdrojový kód a text této práce nyní poskytují ucelené a podrobně zdokumentované řešení problematiky, která je diskutována pouze omezeným množstvím dostupných zdrojů. To bylo také jedním z klíčových aspektů zvýšené složitosti řešení.

Aplikace již nyní implementuje řadu funkcionalit, která byla podrobně rozebrána v kapitole 6.3. Jejich cílem je zvýšení uživatelského komfortu při zadávání vstupu. V budoucnu se nabízí několik dalších rozšíření, například:

Zavedení prediktivního zadávání textu (známého též jako T9), které by jednak zrychlilo zadávání textu běžným uživatelům, ale navíc by mělo pozitivní dopad na negativní chování talkbacku při opakovaném stisku téhož tlačítka. Tato potřeba by totiž byla nasazením T9 značně redukována. Z analýzy dostupných T9 knihoven, která proběhla

v průběhu vývoje Sense Keyboard, však vyplynula potřeba implementace vlastní T9 knihovny včetně slovníkové podpory, což dělá z rozšíření netriviální úpravu.

Parametrizované nastavení počtu slov nabízených našeptávačem, které by na zařízeních s větším displejem umožňovalo uživateli zobrazení více doporučených slov.

Přidání dalších podporovaných gest, která umožní snadnější přístup k jiným možným rozšířením, viz například následující bod]

Bohatší, gesty podporovaná zvuková odezva obsahu vstupního pole, která by zahrnovala například přečtení posledního napsaného slova, poslední věty či celého obsahu textového pole.

Konfigurovatelná výška klávesnice umožňující měnit její plochu. To by umožnilo použití klávesnice standardním způsobem bez režimu celé obrazovky.

Možnost volby vzhledu z několika barevných schémat, kdy se nejedná pouze o estetický prvek, ale někteří uživatelé s vadou zraku mohou být citlivější na určité barevné kombinace.

Tato diplomová práce byla výzvou a snahou o tvorbu užitečné aplikace, která by alespoň částečně pomohla zrakově hendikepovaným v jejich každodenních nesnadných aktivitách digitálního světa. Poskytla velké množství zkušeností a také cenné ponaučení, že hlubší analýzou platformy je možno se vyvarovat následným negativním překvapením v průběhu realizace.

Literatura

[1] Announcing the Android 1.0 SDK, release 1 [online]. <http://android-developers.blogspot.in/2008/09/announcing-android-10-sdk-release-1.html>, 2008-09-23 [cit. 2015-01-27].

[2] How to Customize Android Framework System [online]. <http://www.slideshare.net/noritsuna/lecture03-web>, 2014-1-1 [cit. 2016-07-21].

[3] API Differences between 20 and 21 [online]. <http://developer.android.com/sdk/api_diff/21/changes.html>, 2014-10-15 [cit. 2015-02-03].

[4] Turn on TalkBack – Android Accessibility Help [online]. <https://support.google.com/accessibility/android/answer/6007100?hl=en>, [cit. 2016-06-28].

[5] Accessibility Scanner — Aplikace pro Android ve službě Google Play [online]. <https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor>, [cit. 2016-07-12].

[6] O BlindShell [online]. <http://www.blindshell.com/cs/aplikace>, [cit. 2016-07-22].

[7] SwiftKey Support [online]. <https://support.swiftkey.com/hc/en-us>, [cit. 2016-07-23].

[8] Google Keyboard – Android Apps on Google Play [online]. <https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=cs>, [cit. 2016-07-24].

[9] 3D Pythagoras [online]. <http://www.pythagorasandthat.co.uk/3d-pythagoras>, [cit. 2017-8-04].

[10] Agrawal, A.: Android Application Security Part 2-Understanding Android Operating System [online].
<https://manifestsecurity.com/android-application-security-part-2/>, [cit. 2015-03-17].

[11] Android Developers: Dashboards [online].
<http://developer.android.com/about/dashboards/index.html>, 2017-07-6 [cit. 2017-07-20].

[12] Android Developers: Uses-sdk [online]. <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html>, [cit. 2015-02-02].

[13] Android Developers: Behavior Changes [online].
<https://developer.android.com/preview/behavior-changes.html>, [cit. 2016-07-17].

[14] Android Developers: Platform Architecture [online].
<https://developer.android.com/guide/platform/index.html>, [cit. 2017-07-18].

[15] Black, C.: AndroidCustomKeyboard.
<https://github.com/blackcj/AndroidCustomKeyboard>, 2016.

[16] Griffiths, D.; Griffiths, D.: *Head First Android Development*. O'Reilly Media, Inc., první vydání, 2015, ISBN 978-1-449-36218-8, 698 s.

[17] Horton, J.: *Android Programming for Beginners*. Packt Publishing Ltd., první vydání, 2015, ISBN 978-1-78588-326-2, 660 s.

[18] Klapal, T.: 2017.

[19] Meddaugh, J.: An Evaluation of Android 4.1 Jelly Bean Using the Nexus 7 [online]. *AccessWorld*, ročník 13, 2012-10-1 [cit. 2015-02-05].

[20] Meier, R.: *Professional Android 4 Application Development*. John Wiley & Sons, Inc., třetí vydání, 2012, ISBN 978-1-118-10227-5, 817 s.

[21] SONS Brno: 14. prezentace SONS ČR na 16. ročníku veletrhu Invex 2006 [online].
<http://www.brno.braillnet.cz/invex2006/img/16.jpg>, [cit. 2015-01-30].

Příloha A

Obsah CD

Technická zpráva:

technical_report.pdf

Zdrojové kódy aplikace Sense Keyboard:

sense_keyboard.zip

+ složka /Sense_Keyboard/...

Příloha B

Algoritmy

B.1 Implementace Significant motion detection algoritmu

```
...
// delays between events from sensor informing us about changes
private static final Integer SENSOR_DELAY = Integer.MAX_VALUE;
/* 120s - timer between last movement detection and deviceInMove
flag change to false */
private static final Integer MOVEMENT_TURNOFF_TIMER = 120000;
// how intensive movement is classified as silence break
private static final Integer MOVEMENT_CALCULATION_THRESHOLD = 4;
// how many silence breaks needed to classify device as 'in move'
private static final Integer SILENCE_BREAKS_THRESHOLD = 4;
/* 2s - all silence breaks in less than THIS millis from last one
will be ignored */
private static final long MIN_PERIOD_BETWEEN_SILENCE_BREAKS = 2000;
/* 15s - if no silence break during THIS millis,
silenceBreaksInRowCounter will be cleared */
private static final long MAX_PERIOD_BETWEEN_SILENCE_BREAKS = 15000;
// time in millis of last silence break
private long timeOfLastSilenceBreak = 1;

@Override
public void onSensorChanged(SensorEvent event) {
    //Log.i("SenseKeyboard", "onSensorChanged");

    if (event.sensor.getType() == Sensor.TYPE_SIGNIFICANT_MOTION) {
        // TODO: This part not tested yet since no phone had SMSensor

        /* here we change move flag, because we received event
        that significant motion state changed */
        setDeviceInMoveFlag(!getDeviceInMoveFlag()); // flag negation
```

```
}
else if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

    float[] mGravity = event.values.clone();
    float x = mGravity[0];
    float y = mGravity[1];
    float z = mGravity[2];
    float mAccelLast = mAccelCurrent;
    mAccelCurrent = (float) Math.sqrt(x*x + y*y + z*z);
    float delta = mAccelCurrent - mAccelLast;
    mAccel = mAccel * 0.9f + delta;

    if(mAccel > MOVEMENT_CALCULATION_THRESHOLD) {

        /* first of all, check when last threshold break happened
        (if not at least MIN_PERIOD_..., then ignore it) */
        long currentTime = System.currentTimeMillis();
        if(currentTime < (timeOfLastSilenceBreak
                + MIN_PERIOD_BETWEEN_SILENCE_BREAKS)) {
            return;
        }
        else if(currentTime > (timeOfLastSilenceBreak
                + MAX_PERIOD_BETWEEN_SILENCE_BREAKS)) {
            /* reset silence breaks counter, because more than
            MAX_PERIOD... since last break */
            silenceBreaksInRowCounter = 0;
        }
        timeOfLastSilenceBreak = currentTime;

        silenceBreaksInRowCounter++;

        if(silenceBreaksInRowCounter >= SILENCE_BREAKS_THRESHOLD) {
            silenceBreaksInRowCounter = 0; // reset counter

            // device considered to be in move
            setDeviceInMoveFlag(true);

            DropDeviceInMoveFlag dropDeviceInMoveFlag
                    = new DropDeviceInMoveFlag(this);

            if(timer != null) {
                timer.cancel();
            }
            timer = new Timer();
            /* device in move flag set, lets create timer for
```

```
                its dropdown after some period of silence time */
                timer.schedule(dropDeviceInMoveFlag,
                        MOVEMENT_TURNOFF_TIMER);
            }
        }
    }
}
```

B.2 Popis layoutu aktivity pomocí XML

Soubor activity_main_settings_init.xml

```
   ...
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".SenseKeyboardSettingsInit">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/SettingsTitleFirst"
        android:id="@+id/settingsTitle"
        android:layout_gravity="top|center_horizontal" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="@string/SettingsBaseFirst"
        android:id="@+id/settingsTextBase"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp" />
        <!--android:textSize="20sp" -->

    <RelativeLayout
```

```
        android:layout_width = "fill_parent"
        android:layout_height = "fill_parent" >

        <LinearLayout
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:paddingLeft="0dp"
            android:paddingRight="0dp"
            android:orientation="vertical"
            android:layout_gravity="center_horizontal"
            android:baselineAligned="false" >

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Nastavit klávesnici"
                android:id="@+id/main_settings_next_button" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Ukončit nastavení"
                android:id="@+id/main_settings_exit_button" />

        </LinearLayout>
    </RelativeLayout>
</LinearLayout>
```

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Klapal Tomáš	ČSA 1404, Hlinsko	I1300379

TÉMA ČESKY:

Systémová klávesnice pro OS Android se zaměřením na přístupnost

TÉMA ANGLICKY:

Software keyboard for OS Android focusing on accessibility

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem je navrhnout, vytvořit a prezentovat softwarovou klávesnici pro platformu Android, která bude přístupná i nevidomým a slabozrakým uživatelům.

Osnova:
1. Úvod
2. Zraková vada a moderní informační technologie
3. Platforma Android
4. Existující aplikace
5. Analýza a návrh aplikace
6. Implementace
7. Výsledky, závěr
8. Literatura

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: ..... **Datum:** 12.10.2015

Podpis vedoucího práce: ..... **Datum:** .....