VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

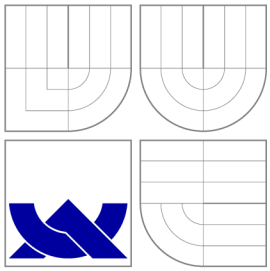
INTERAKTIVNÍ IKONY PRO MOBILNÍ APLIKACI INTELIGENTNÍ DOMÁCNOST

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

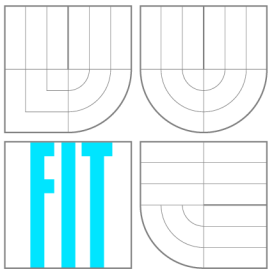
AUTOR PRÁCE
AUTHOR

TOMÁŠ MLYNARIČ

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

INTERAKTIVNÍ IKONY PRO MOBILNÍ APLIKACI INTELIGENTNÍ DOMÁCNOST

WIDGETS FOR A MOBILE APPLICATION OF INTELLIGENT HOME CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ MLYNARIČ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN ŽÁDNÍK, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá vývojem interaktivních ikon (widgetů) pro platformu inteligentní domácnost vyvíjenou na FIT VUT v Brně a operační systém Android. V práci jsou představeny platformy, pro které je vývoj určen, dále jsou vysvětleny principy vývoje widgetů a v praktické části pak návrh a implementace widgetů. Na konci práce jsou zhodnoceny výsledky a projednány možnosti rozšíření.

Abstract

This thesis deals with development of widgets designed for intelligent home control in the environment of Android operation system. The intelligent home control is a framework developed at FIT BUT. The thesis introduces the framework and explains principals of developing widgets. The practical part discusses design and implementation of the specific widgets. The implementation is evaluated and the results are discussed. The concluding section provides summary and suggests future improvements.

Klíčová slova

inteligentní, chytrá, domácnost, Android, widgety, interaktivní ikony, služba, broadcast

Keywords

intelligent, smart, home, Android, widgets, service, broadcast

Citace

Tomáš Mlynarič: Interaktivní ikony pro mobilní aplikaci
inteligentní domácnost, bakalářská práce, Brno, FIT VUT v Brně, 2015

Interaktivní ikony pro mobilní aplikaci inteligentní domácnost

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Žádníka, PhD.

.....
Tomáš Mlynarič
17. května 2015

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu práce, panu Ing. Martinu Žádníkovi, Ph.D., za odborné vedení práce. Také bych chtěl poděkovat své rodině a přítelkyni za morální podporu a trpělivost, kterou mně při řešení této práce věnovali.

Obsah

Kapitola 1

Úvod

Po dlouhá staletí byly domácnosti brány pouze jako místo, kde člověk má pouze své zázemí, ve kterém přetrvává noci a nevlídná počasí. Elektřina dokázala změnit tento způsob života, kde díky různým přístrojům začaly domácnosti přebírat činnosti, které do té doby bylo možné provádět pouze venku. Jak šel čas dál, z elektřiny se stal společník, díky kterému se v domácnostech dokážeme také bavit. Dalším a zřejmě zatím posledním milníkem se stal příchod Internetu. Jeho zásluhou se domácnosti začínaly proměňovat v interaktivní místa, která umožňují spojení se zbytkem světa a dokonce jistým způsobem dokážou domácnosti s člověkem komunikovat. Jelikož ale i tato doba je již za námi a nastupuje doba Internetu věcí (z angl. *Internet of Things*), kde spousta přístrojů v domácnosti je připojena k Internetu, člověk může začít odpovídat na komunikaci domácnosti. A co víc, díky chytrým telefonům, které mají schopnost být připojeny k Internetu, můžeme domácnostem rozkazovat, i když nejsme doma. Člověk se tedy nemusí obávat o svou domácnost, když je někde na výletě či v práci a je schopen kontrolovat si své zázemí i na dálku.

Cílem této práce je umožnit ovládat inteligentní domácnost skrze telefon s operačním systémem Android a to díky použití interaktivních ikon (widgetů) na domovské obrazovce telefonu. Interaktivní ikony mohou rozšiřovat již existující aplikaci pro ovládání inteligentní domácnosti a díky své existenci umožnit částečně obejít danou aplikaci a tím zefektivnit práci s celým systémem. Aby byla práce s widgety pro uživatele co nejvíce intuitivní a nepůsobila problémy v kombinaci s widgety jiných aplikací, je potřeba zajistit, aby widgety byly flexibilní, přehledné a zároveň poskytovaly správnou míru informací.

Tato práce popisuje co je to inteligentní domácnost a představuje systém vyvíjený na FIT VUT v Brně (kapitola 2). V následující kapitole, 3, je popsána platforma Android, vrstvy, ze kterých se celý systém skládá a způsoby vývoje pro tuto platformu. Kapitola 4 představuje interaktivní ikony. Projednává, jaké existují typy ikon, jakým způsobem probíhá vývoj, v čem se liší od klasické aplikace a jaké jsou problémy spojené s jejich vytvářením. Následující kapitoly 5 a 6 jsou věnovány praktické části této práce. První ze zmíněných kapitol se zabývá postupem návrhu této práce. Jsou rozebrány možnosti manipulace s widgety, představeny návrhy grafických rozhraní a jakým způsobem lze widgety řídit. Druhá z kapitol je určena samotné implementaci práce. Poslední kapitola, 7, je určena přezkoumání správné funkčnosti práce a dosažených výsledků při testování.

Kapitola 2

Inteligentní domácnost

Tato kapitola má za cíl seznámit čtenáře s pojmem inteligentní domácnost a dále představit platformu Inteligentní domácnosti vyvíjenou na FIT VUT v Brně. Obsah této kapitoly je inspirován textem z [2].

2.1 Pojem inteligentní domácnost

Dle článku [1] byly inteligentní domácnosti po dlouhou dobu brány jako způsob, jakým dokážeme žít uvnitř pohodlněji. Automatická vrata či rozsvěcování pomocí tlesknutí jsou sice technologie zajímavé, ale nepřinášejí žádnou revoluci. V současnosti se ale společnosti snaží udělat domácnosti chytřejší tím, že je např. propojí s mobilními telefony. Tento krok umožní, že člověk dokáže mít kontrolu nad svou domácností ze své kapsy a to odkudkoliv, kde je internetové připojení. Takto je nejen možné ovládat technologie na dálku, ale také, aby domácnost sama komunikovala s telefonem (např. kávovar, který automaticky uvaří kávu podle budíku v telefonu nebo vyhřátí automobilu v zimě podle plánovaného odjezdu).

2.2 Architektura platformy na FIT VUT

Architektura platformy (viz obrázek 2.1) je složená z několika vrstev komunikujících mezi sebou. Nejblíže reálným datům je vrstva senzorů, kterou zapouzdřuje vrstva domácí sítě, v níž se nachází brána. Cloudové řešení představuje vrstva Internetu a nejblíže k uživateli je vrstva koncových zařízení, díky kterým je celý systém možné ovládat.

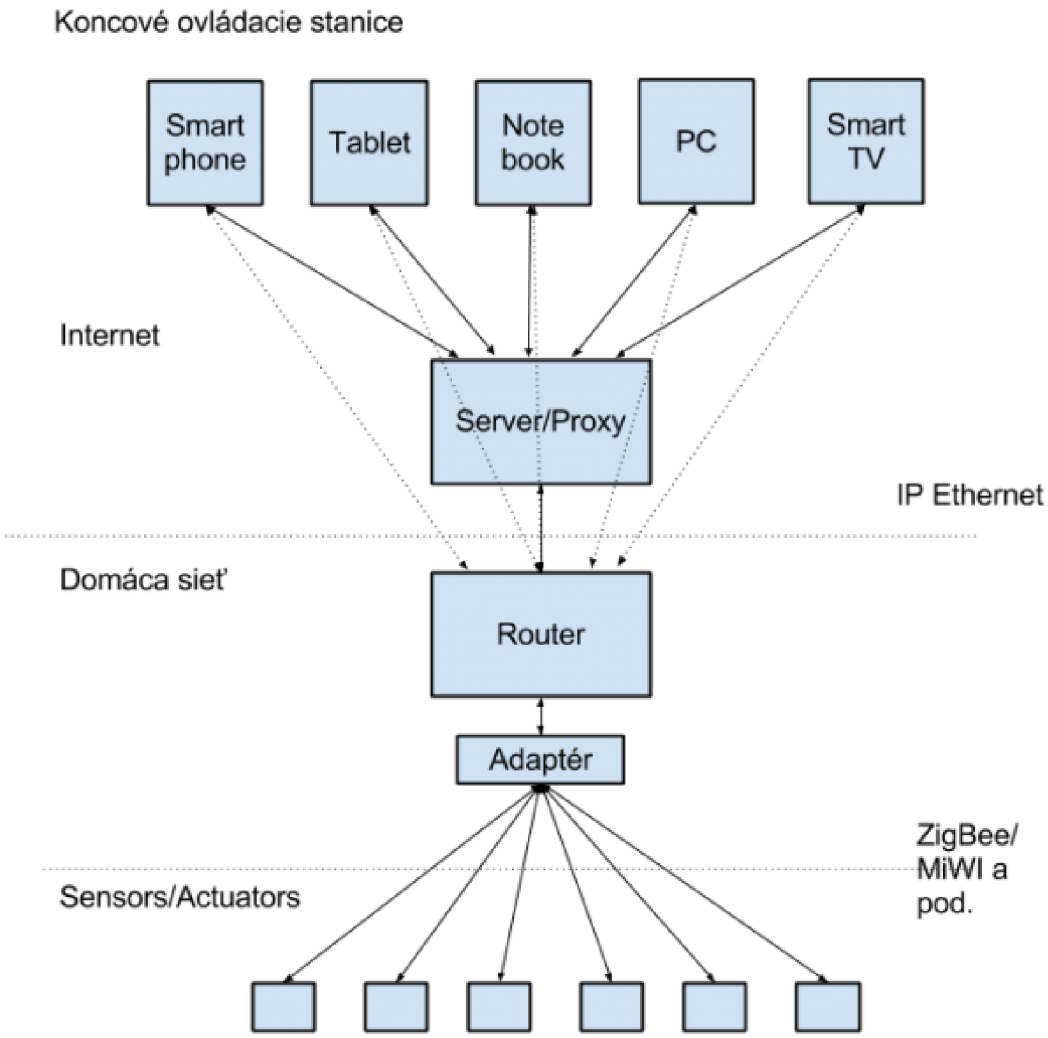
Koncová zařízení

Ovládání domácnosti je možné při použití dostupné spotřební elektroniky, která dokáže komunikovat přes Internet (tj. chytré televize, počítače, tablety a hlavně mobilní zařízení[1]). Takto se všechna komunikace připojuje na vzdálený cloud (server).

Server

Server je hlavní výpočetní jednotkou celého systému. Je dosažitelný přes Internet z koncových zařízení a obsahuje jak logiku, tak i data uživatelských domácností (data jsou na serveru zabezpečena a komunikace probíhá výhradně zašifrovaným přenosem).

[1] V současnosti dostupné na mobilních zařízeních a tabletech s OS Android, a zařízení s OS Windows Phone.



Obrázek 2.1: Koncept architektury platformy Inteligentní domácnost (zdroj: [2])

Nevýhodou tohoto způsobu je, že pokud uživatel není připojen k internetu, nedokáže ovládat svoji domácnost. Výhodou naopak je, že je možné nad daty uživatelů provádět složitější algoritmy, jako např. počítání rosného bodu v interiéru domácnosti či předpověď počasí. Tyto výpočty často potřebují velké množství dat a nemalý výpočetní výkon, co znamená, že aby mohly být použity přímo v domácnosti, musely by brány domácnosti mít dostatečný výpočetní výkon a to by se ve výsledku prodražilo.

Brána

Brána je zařízení podobné domácímu routeru. Jeho úkolem je sbírat data z jednotlivých senzorů a pomocí internetového připojení komunikovat se serverem. Stará se o adresování a komunikaci se senzory. Obecně by se tedy dalo říci, že brána je pouze mostem mezi bezdrátovými protokoly senzorů a IP protokolem. Aby nedocházelo k odposlechům, např. v panelových domech, brána komunikuje pouze se senzory, které jsou s ním spárovány. Proces párování probíhá podobně jako u technologie Bluetooth, kde si brána vyžádá párování a senzor příslušným způsobem odpoví na dotaz.

Senzory a aktory

Nejnižší části celé architektury jsou jednotlivé prvky, které snímají danou hodnotu z okolí či ovlivňují svým působením nějaké zařízení. Senzory by měly být velmi jednoduchá zařízení, která ideálně snímají pouze jednu veličinu. Tuto hodnou přeposílají pomocí bezdrátového protokolu[2] ZigBee nebo MiWi na domácí bránu. Aktor taktéž splňuje jeden účel, a to např. spínání elektrického kontaktu a tudíž zapínání nějakého zařízení (např. termostatická hlavice nebo zásuvka). Jak již bylo zmíněno, obě tato zařízení by měla být velmi jednoduchá, resp. levná na výrobu.

2.3 Uživatelské role

Výsledný systém má být komplexní a počítá s používáním v různých podmínkách a ne jenom v rámci jedné rodiny a jedné domácnosti. Z tohoto důvodu je potřeba zamezit některým uživatelům jisté funkce, aby nemohlo dojít k použití systému nesprávným způsobem (např. otevření vrat garáže neoprávněnou osobou). Systém tedy disponuje uživatelskými rolemi, kde každá role má přístup k podmnožině operací role nadřazené. Existují 4 role mající tato práva[3]:

- **Host** – Má možnost pouze zobrazovat údaje ze senzorů.
- **Uživatel** – Dokáže zobrazovat údaje ze senzorů a přepínat aktory.
- **Správce** – Může měnit umístění senzorů a spravovat lokace.
- **Vlastník** – Existuje pouze jeden v rámci brány a k operacím nižší úrovně může také spravovat uživatele brány.

2.4 Použití

Celý systém je možné zařadit do fenoménu Internet věcí, kde je možné ovládat domácnost rychle, efektivně a hlavně vzdáleně. Systém by do budoucna měl analyzovat stav jednotlivých meteorologických jevů uvnitř domu a dle toho činit tak, aby byla spotřeba energií co nejnižší a efektivnost využití co nejvyšší. Také z bezpečnostního hlediska je možné principy inteligentní domácnosti využít – buďto jako senzory snímající pohyb, kamery či jako automatické zapínání světel v případě nepřítomnosti. V neposlední řadě by domácnost mohla přemýšlet samostatně a zautomatizovat opakující se činnosti či je vykonávat v požadovaných případech. Typickým příkladem může být automatické zalévání květin či otevírání oken při velké vlhkosti. Nasazení by mohlo být jak v domácnostech, které by tyto služby měly již zabudovány, tak i tam, kde by byla použita externí zařízení.

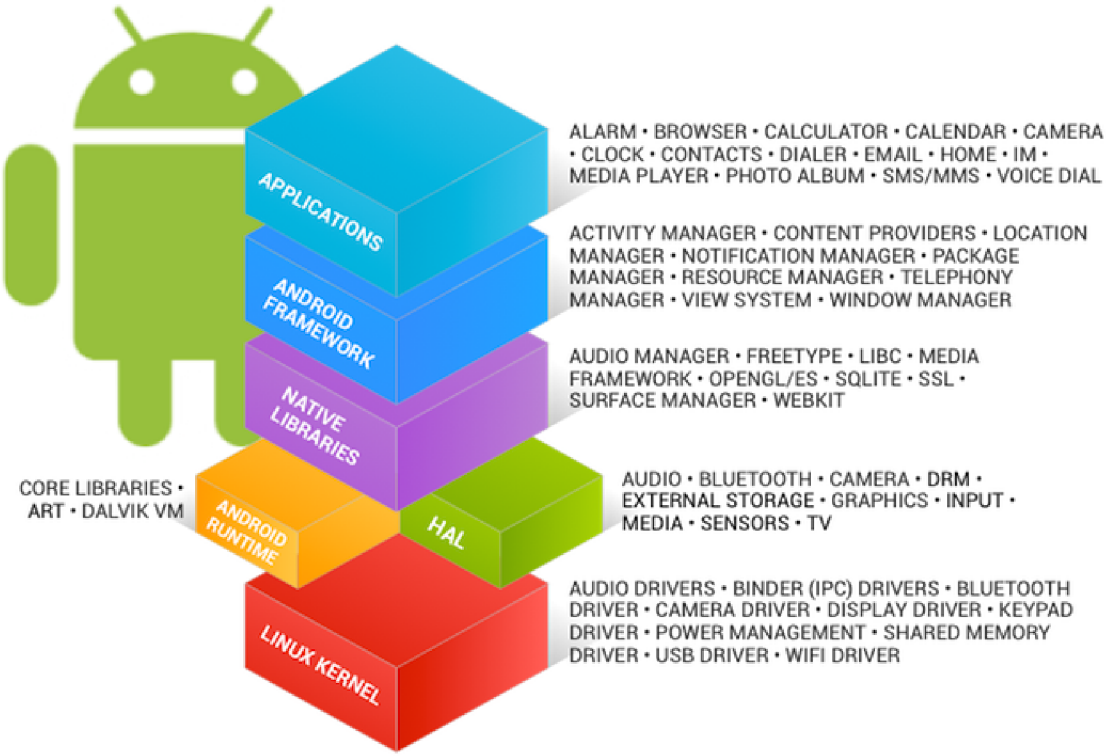
[2] Tento protokol bude od roku 2015 vyměněn za vlastní protokol vyvinut pro potřeby inteligentní domácnosti na FIT VUT (viz https://ant-2.fit.vutbr.cz/projects/senzory/wiki/protokol/)

[3] Celá tabulka operací viz https://ant-2.fit.vutbr.cz/projects/android-app/wiki/User_roles_android

Kapitola 3

Android

Android je systém vyvíjený společností Google Inc. Zahrnuje operační systém založený na Linuxu, middleware a klíčové mobilní aplikace spolu se sbírkou API knihoven pro psaní aplikací, které formují vzhled, použití a funkčnost zařízení, na kterých jsou spuštěny. Přestože vývoj pro operační systém linux bývá často upřednostněn v nativních programovacích jazycích, největší počet aplikací je vyvíjeno v jazyce Java, pro který je vytvořen speciální framework (viz sekce 3.1). Tato kapitola čerpá z [12].



Obrázek 3.1: Architektura systému Android (zdroj: [11])

3.1 Architektura systému

Architektura je složena zjednodušeně řečeno z Linuxového jádra a kolekcí C/C++ knihoven, které jsou zpřístupněny skrz aplikační framework. Ten poskytuje služby pro běhové prostředí a aplikace. Tak jako každý operační systém, i Android je složen z několika vrstev komunikujících mezi sebou (viz obrázek 3.1):

- **Linuxové jádro** (Kernel) – Nejspodnější vrstva zahrnují základní služby systému (včetně ovladačů hardwaru, správy procesů a paměti, zabezpečení, sítě a řízení napájení). Kernel také poskytuje abstrakci mezi hardwarem a zbytkem systému.
- **Nativní knihovny** – Vrstva nad jádrem zpřístupňující různé základní C/C++ knihovny pro správu videa a audia, řízení displeje, grafiky (*OpenGL*), databáze (*SQLite*) či *WebKit* a *SSL* pro integrovaný webový prohlížeč a zabezpečení.
- **Běhové prostředí Android** (Android Runtime) – Je to, co tvoří Android a ne pouze Linuxový systém. Je to engine, který pohání aplikace a spolu s knihovnami formuje základy aplikačního rámce. Jelikož aplikace jsou programovány v jazyce Java, je potřeba virtuálního stroje Javy. Android používá *Dalvik Virtual Machine*, který je optimalizovaný tak, aby zajistil co nejlepší využití systémových prostředků (obsahuje optimalizované vlastnosti řízení vláken a správy paměti).
- **Aplikační rámec** (Application Framework) – Vrstva, kterou používá nejvíce vývojářů Androida. Poskytuje třídy pro vytváření aplikací, obecnou abstrakci pro přístup k hardwaru a spravuje uživatelské rozhraní a prostředky aplikací (z angl. *application resources*).
- **Aplikace** – Do této vrstvy spadají jednotlivé aplikace, které uživatelé mohou používat. Obvykle je lze získat skrze repositáře typu Google Play či Amazon Appstore. Druhým způsobem je nainstalovat aplikaci manuálně skrze instalační balíček, tzv. APK. Všechny aplikace používají stejné API knihovny a jsou spouštěny uvnitř běhového prostředí Androida.

3.2 Verze operačního systému

Operační systém Android běží na velkém množství zařízení různých firem. Každá firma často implementuje svá rozšíření systému a kvůli toho se v praxi jednotlivé verze Androidu liší. Liší se jak velikostí a jemností displeje (*dp*[1]), tak různými verzemi systému. Kvůli této fragmentaci je nutno buď aplikace přizpůsobovat verzím systému, nebo umožnit instalovat aplikaci pouze na vybrané verze. Jak lze z tabulky 3.1 vyčíst, nejvíce zařízení běží na systémech verze Jelly Bean, KitKat a Lollipop. Pro tyto verze by ve většině případů měl být uzpůsoben vývoj aplikací. V dnešní době je ale časté, že jsou aplikace přizpůsobovány starším verzím systému (např. Gingerbread), na kterém většina nové funkčnosti systému není podporována. Jak ale tabulka napovídá, množství používaných zařízení s tímto systémem není rozhodující, takže je možné, že za krátkou dobu již nebude nutné tyto verze podporovat.

[1] Density-independent pixel je virtuální pixelová jednotka, která se přepočítává na základě rozlišení displeje.

Verze	Název	API	Podíl
2.2	Froyo	8	0.3%
2.3.x	Gingerbread	10	5.7%
3.0	Honeycomb	11	$<0.1\%$
4.0.x	Ice Cream Sandwich	15	5.3%
4.1.x	Jelly Bean	16	15.6%
4.2.x		17	18.1%
4.3		18	5.5%
4.4	KitKat	19	39.8%
5.0	Lollipop	21	9.0%
5.1		22	0.7%

Tabulka 3.1: Rozložení používaných verzí systému Android (zdroje: [8] a [10])

3.3 Vývoj aplikací

Pro vývoj aplikací je možné použít oficiální nástroje Android Developer Tools (ADT) s vývojovým studiem Eclipse[2], nebo od roku 2014 je také oficiálně podporované vývojové prostředí Android Studio založené na platformě IntelliJ Idea[3].

K vývoji je také nutné používat Android SDK[4], což je kolekce vývojových nástrojů obsahující zdrojové kódy, příklady, dokumentaci atd. Při použití Android Studia je SDK již součástí instalace. Následuje výčet nejčastějších způsobů vývoje pro Android:

- **Nativní aplikace** – Je možné je psát v jazycích C/C++ a jsou často využívány pro programování komplexních programů (např. herní enginy, fyzické simulace, zpracování signálů apod.) nebo nižších úrovní systému. Pro vývoj takových aplikací je možné použít oficiální vývojový kit NDK[5] od Androidu.
- **HTML5 aplikace** – Jsou aplikace vyvíjené pomocí webových technologií, které jsou spouštěny klasickými aplikacemi uvnitř kontrolního prvku WebView[6].
- **Java aplikace** – Nejčastější způsob vývoje aplikací pro Android jsou aplikace psané v jazyce Java, které jsou oficiálně podporovány systémem s plnou funkčností.

[2] https://eclipse.org/
[3] https://www.jetbrains.com/idea/
[4] Software Development Kit
[5] https://developer.android.com/tools/sdk/ndk/index.html
[6] Kontrolní prvek dovolující zobrazit webový obsah uvnitř Java aplikace

3.4 Komponenty aplikace

Android aplikace obsahují nesporné množství komponent, ze kterých je možné danou aplikaci vytvářet. Zde je uveden výčet nejdůležitějších, které byly použity v této práci.

AndroidManifest

Je to nejdůležitější část aplikace, bez které by žádná aplikace nemohla fungovat. Definuje seznam aktivit, služeb a broadcast receiverů, které daná aplikace obsahuje. Dále také určuje, která aktivita bude vstupním bodem do aplikace a jak bude vypadat spouštěcí ikonka. V neposlední řadě je zde definován seznam oprávnění[7], která aplikace využívá. Bez formulace těchto oprávnění lze sice aplikaci zkompilovat, když ale bude v telefonu spuštěna, okamžitě je systémem ukončena.

Aktivita a fragment

Podle informací dostupných z [7] jsou aktivity a fragmenty nejzákladnějšími komponentami aplikace, ve kterých se odehrává veškerá manipulace uživatele s aplikací. Aktivity se mohou otevírat navzájem a přecházet tak na jiné „stránky“ aplikace. Nejčastěji vystupují v podobě okna přes celou obrazovku, mohou ale existovat i ve formě plovoucího okna nebo uvnitř jiné aktivity. Od systému Honeycomb (viz tabulka 3.1) Android podporuje také odlehčenou formu aktivit v podobě fragmentu. Ten může být vytvořen jedině v aktivitě, ale aktivita jich může obsahovat více a je možné spravovat je za běhu aplikace. Fragment je také schopen znovupoužitelnosti, co znamená, že je možné jej používat na více místech aplikace – v různých aktivitách.

Broadcast receiver

Zdroj [3] uvádí, že přijímač broadcast zpráv (z angl. *broadcast receiver*) je komponenta, která je volána vždy, když je zaslána ním definovaná broadcast zpráva[8]. Zprávy mohou být systémové a ten jimi informuje všechny běžící aplikace o dané události, druhou možností je, že jsou v aplikaci nadefinované vlastní zprávy. Broadcast receiver obecně neobsahuje žádný kontext aplikace a tudíž by neměl uchovávat nestatické proměnné. Systém sám přenáší kontext dané aplikace do broadcast receiveru pouze v případě, že je zaslána nějaká zpráva a tudíž volána metoda `onReceive()`.

Service

Na základě informací z [4] je zřejmé, že služba (z angl. *service*) je komponenta, která provádí dlouhodobé operace na pozadí a neposkytuje uživatelské prostředí. Když uživatel přepne aplikaci na jinou, služba přetrvává na pozadí a je ukončena pouze tehdy, pokud se sama ukončí (např. po skončení práce), ukončí ji samotná aplikace nebo systém v případě, že nemá dostatek systémových prostředků. Služba obecně může být spuštěna pouze jednou, a to i přesto, že se jí pokusí aplikace spustit vícekrát. Naopak ukončena je vždy po prvním zavolání konce služby.

[7] Typickým příkladem oprávnění mohou být přístup k internetu (`INTERNET`), používání vibrací telefonu (`VIBRATE`) či zobrazování uživatelských účtů v telefonu (`MANAGE_USER_ACCOUNTS`).

[8] Broadcast je podobně jako v komunikačních sítích zpráva, která se pošle napříč celým systémem.

Alarm manager

Když je potřeba provádět operace založené na čase mimo životní cyklus aplikace, je možné použít `AlarmManager`. Dle [9] se často používá ke spouštění dlouho běžících operací (např. nastavení budíku na určitou hodinu každý den). Je možné jej používat ve dvou módech a variantách s probouzením a neprobouzením procesoru ze spánku:

- `ELAPSED_REALTIME` – Spouští se na základě času uplynulého od zapnutí telefonu.
- `RTC` – Spouští se ve specifikovaný reálný čas.

Spánek procesoru znamená, že procesor nevykonává žádné důležité operace. Nejčastěji je procesor v režimu spánku, když má telefon vypnutý displej delší dobu. Pokud je `AlarmManager` použit ve variantě probouzení procesoru, je potřeba zajistit, že se spouští pouze v nutně potřebnou dobu, protože jinak může docházet k rychlému vybití baterie telefonu.

Kapitola 4

Interaktivní ikony – widgety

Interaktivní ikony (dále jen widgety) jsou samostatné mini aplikace, které je možné umístit na domovskou stránku (tzv. launcher). Pojem samostatná miniaplikace znamená, že widget je sice v instalačním balíčku dané aplikace, ale není přímo součástí aplikace.

4.1 Typy widgetů

Podle zdroje [6] je widgety možné zařadit do několika kategorií na základě jejich funkčnosti a vzhledu:

- **Informační** – Zobrazují pouze pár důležitých informací, které se mění v čase. Typickým příkladem je widget hodin nebo různé widgety pro sledování informací (např. výsledky sportu, akcie apod.).
- **Kolekce** – Specializují se na zobrazování několika informací stejného typu s možností přechodu do aplikace, kde je možné zobrazit podrobnější informace. Widget kolekce je možné posouvat vertikálně. Příkladem je emailový widget nebo záložky prohlížeče.
- **Kontrolní** – Slouží jako dálkový ovladač často používaných funkcí dané aplikace. Příkladem mohou být různé přepínače funkcí telefonu (internetové připojení, bluetooth, rotace displeje atd.).
- **Hybridní** – Spojuje všechny kategorie v jednu. Dá se říci, že nejčastěji používanou kategorií widgetů je právě hybridní, kvůli možnosti ovládat aplikaci a také zobrazení důležitých informací. Nejčastěji se do této kategorie řadí widget ovládání hudby nebo pokročilé widgety různých aplikací.

4.2 Tvorba widgetů

Následující sekce čerpá ze zdroje [5]. Jak již bylo zmíněno, widgety nejsou součástí aplikace a kvůli tomu je potřeba vytvářet je specifickým způsobem. Implementace widgetu se skládá z několika kroků, které je potřeba udělat, aby bylo používání widgetů možné. Nejprve je potřeba definovat widget (v podobě broadcast receiveru) do `AndroidManifestu`. V druhém kroku je třeba tuto třídu implementovat (minimálně tedy oddědit od třídy `AppWidgetProvider`). Následně je zapotřebí poskytnout metadata o widgetu a posledním krokem je vytvoření vzhledu.

AppWidgetProvider

`AppWidgetProvider` je třída zařizující příjem událostí (broadcast zpráv) pro widgety. Dědí ze třídy `BroadcastReceiver` (viz sekce 3.4) a prakticky pouze rozšiřuje jeho činnost o akce spojené s manipulací widgetů. Nicméně pro fungování widgetů je tato třída esenciální, protože systém využívá této třídy k identifikaci daného typu widgetu. Každý widget dokáže přijímat od systému tyto broadcasty:

- `onEnabled` – zpráva volána při vytvoření prvního widgetu stejného typu
- `onUpdate` – když systém chce aktualizovat již přidané widgety
- `onDeleted` – kdykoliv je widget odstraněn z launcheru
- `onDisabled` – když je nakonec odstraněn poslední widget dané aplikace
- `onAppWidgetOptionsChanged` – V API verzi 16 (viz 3.1) byla přidána tato zpráva indikující změnu velikosti daného widgetu. To umožňuje případně překreslit widget a zobrazit tak více informací.

Metadata widgetu

Jsou informace o základních vlastnostech widgetu. Nejdůležitější z těchto vlastnosti jsou:

- **Minimální rozměry** – Slouží k určení, na kolik políček domovské obrazovky bude widget umístěn.
- **Počáteční layout** – Když je widget přidán na domovskou obrazovku, výchozí vzhled je určen layoutem specifikovaným tímto parametrem.
- **Konfigurační aktivita** – Není povinná u každého widgetu, nicméně pokud je nadefinovaná, během přidání widgetu se nejprve zobrazí tato aktivita, která má v režii, jestli se widget nakonec vytvoří. Obvykle tato aktivita slouží k nastavení dat, bez kterých by widget nemohl existovat (nebo jeho nastavení by bylo nejednoznačné).
- **Perioda aktualizace** – Díky ní lze specifikovat, jak často bude widget aktualizován systémem.

V Androidu 3.0 byla přidána možnost zobrazit **náhled widgetu** a v Androidu 4.0 byly přidány tyto vlastnosti:

- **Mód změny velikosti** – Dovoluje změnu velikosti widgetu. Na výběr jsou buď vertikálně, horizontálně nebo obojí.

- **Minimální rozměry pro zmenšení widgetu** – Widget je možné roztáhnout na maximální velikost obrazovky telefonu, ale zmenšit jej lze pouze na rozměry této vlastnosti.

Budování vzhledu

Při vývoji aplikace se jednotlivé aktivity či fragmenty (sekce 3.4) vytvářejí tak, že se vzhled nadefinovaný v XML souboru „nahustí“ (z angl. *inflate*) a poté je možné programátorsky získávat jednotlivé kontrolní prvky a měnit jejich vlastnosti. Tento způsob neplatí u tvorby vzhledu widgetů. Jelikož widget neběží v procesu aplikace, nýbrž v systémovém procesu, je nutno použít vzdálené budování vzhledu pomocí třídy `RemoteViews`. Tato třída nejprve layout načte z XML souboru a poté z něj vytvoří hierarchii kontrolních prvků jako posloupnost akcí. Změna vlastnosti nějakého elementu probíhá pouze nastavením atributu kontrolního prvku identifikovaného pomocí `id` z layoutu (uvnitř se vykoná pouze vytvoření nové akce). Když jsou všechna potřebná nastavení dokončena, je možné překreslit widget pomocí zavolání metody `updateAppWidget`, co na pozadí vykoná v systémovém procesu všechny definované akce.

Omezení

I přes to, že widgety jsou chápány jako mini aplikace, mají jistá omezení, která vyplývají z toho, kde jsou umístěny. Jelikož tím umístěním je domovská stránka, která používá několik gest (např. horizontální posuv či dlouhé kliknutí) jsou jedinými povolenými gesty pro widgety zmáčknutí (`onClick`) a vertikální posuv. Z těchto důvodů je také možné vytvářet vzhled pouze z omezeného výběru kontrolních prvků[1] a to výlučně základní skupinové layouty[2] a základní designové prvky jako např. tlačítko, obrázek, textový blok atd.[3] Jestliže tedy je potřeba použít jiné prvky, jedinou možností je nasimulovat jejich vzhled a funkčnost pomocí dostupných prvků.

Následujícím omezením je, že verze OS Android starší než API 16 (viz tabulka 3.1) nepodporují widgety kolekce. Neexistuje způsob, jakým by šlo tuto kategorii widgetů simulovat na daných zařízeních, a proto jedinou možností pro tato zařízení je widgety kolekce zakázat.

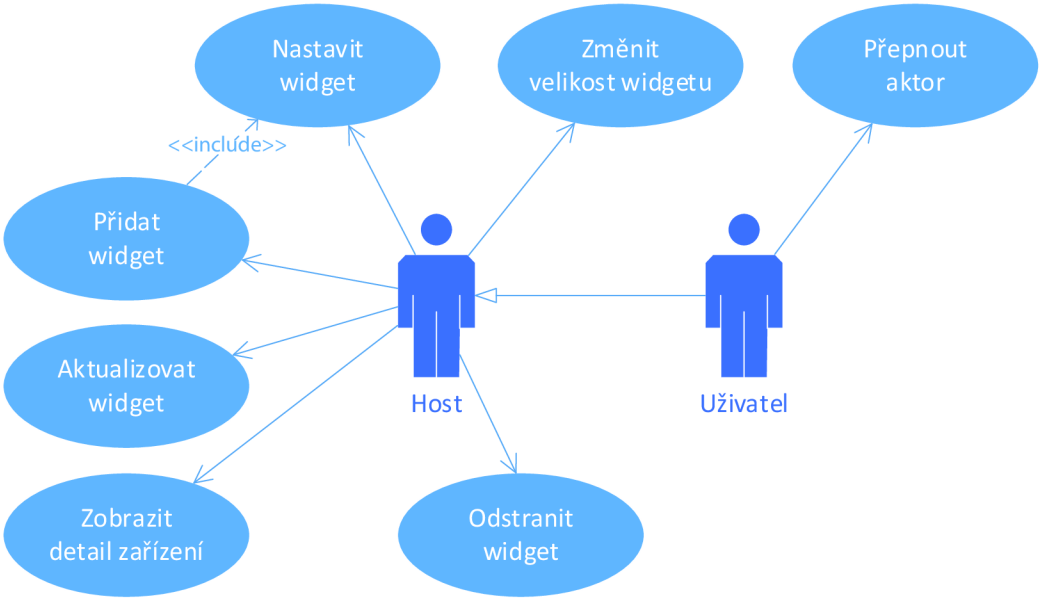
Dále je zde také problém, že systém sice má možnost, jak aktualizovat widgety jednotně (viz sekce 4.2), bohužel tato aktualizace nemůže probíhat častěji než každých 30 minut, co u widgetů zobrazujících často měnící se informace může být problematické.

[1] UI Building Blocks - jsou to elementy, ze kterých je možné tvořit layout android aplikací
[2] ViewGroup je kontrolní prvek, který může obsahovat jiné kontrolní prvky.
[3] Kompletní seznam ve zdroji [5]

Kapitola 5

Návrh

Tato kapitola se zabývá možnostmi práce s widgety, jejich konfigurací, způsobem aktualizace a také grafickými návrhy předpokládaných widgetů. Při návrhu bylo třeba rozhodnout, jakým způsobem zajistit možnost rozšíření widgetů o další typy a jak sjednotit celý projekt do jednoho frameworku, který dokáže všechny typy widgetů řídit jednotným způsobem.



Obrázek 5.1: Schéma případů užití – práce s widgety

5.1 Manipulace s widgety

Na obrázku 5.1 jsou znázorněny jednotlivé operace, které má uživatel aplikace k dispozici. Tyto operace se pro každý widget mohou lišit a zde jsou pouze zobrazeny případy užití, které jsou stejné pro většinu předpokládaných widgetů. Některé z těchto případů vyplývají přímo ze systému a aplikace tedy přejímá část této funkcionality (viz sekce 4.2).

S widgety mohou manipulovat všichni uživatelé dané domácnosti nehledě na jeho roli v systému (viz sekce 2.3), nicméně je možné implementovat ve widgetu kontrolu nad danými prvky dle práv uživatele (pokud by například ve widgetu bylo zařízení aktor, uživatel s nedostatečným oprávněním by widget viděl pouze jako senzor a nemohl jej přepínat).

Přidání widgetu Na domovskou obrazovku lze widget přidat tak, že se buď přidá okamžitě a není tedy potřeba jej nastavovat (případně je použito předpokládané nastavení), a nebo se před přidáním zobrazí konfigurační aktivita. Je vhodné odchytávat broadcast systému vytvoření widgetu.

Odstranění widgetu K odstranění widgetu je možno přejímat funkcionalitu systému a odchytávat broadcast. Při výskytu této události systém odstraní objekt widgetu (potomka třídy `AppWidgetProvider`) a je vhodné, aby byly uvolněny všechny zdroje, které alokoval.

Změna velikosti widgetu Je také částečně v režii systému Android. Měnit velikost lze pouze widgetům, které to mají nadefinované ve svých metadatech. Systém v takovém případě mění velikost vždy o jednu buňku mřížky domovské obrazovky a záleží na widgetu, jakým způsobem se bude měnit vzhled. Nejjednodušším (a často jediným) způsobem je, že widget změní svůj layout na jiný zobrazující více resp. méně informací. V případě widgetů kolekce je v takové situaci nejčastější, že se pouze zobrazí více položek kolekce, nicméně např. v informačním widgetu je potřeba lépe naplánovat, jaké doplňující informace a jakým způsobem se budou zobrazovat. Na obrázku 5.2 je zobrazen způsob, jakým je možno překreslovat widget na základě jeho velikosti.

Obrázek 5.2: Příklad zobrazení změny velikosti widgetu (zdroj: [6])

5.2 Způsob aktualizace

Pro aktualizaci widgetů je možné použít systémový způsob (skrze implementování metody `onUpdate()`, která je volána broadcastem), nicméně vzhledem k požadavku vytvořit sadu widgetů, které mohou potřebovat aktualizovat v různých časových intervalech, nelze spoléhat na tento způsob kvůli jeho omezení viz sekce 4.2. Z tohoto důvodu je vhodné vytvořit službu, která bude aktualizovat widgety podle potřeby i přes krátký čas aktualizace. Tato služba by neměla být spouštěna častěji než je potřeba a to i v případě, že widgety budou mít velmi podobné časové intervaly aktualizace. V takovém případě je vhodné požadavky aktualizace shlukovat a zabránit tím nesprávnému využívání systémových prostředků.

5.3 Konfigurace widgetů

Nastavení widgetů může probíhat tak, že bude v aplikaci globálně nastaveno, jaké informace se ve widgetech budou zobrazovat. Druhým a lepším způsobem je, že se použije konfigurační aktivita pro widgety, která zajistí, že každý widget může zobrazovat jiné informace a dokáže mít jiný vzhled. Toto nastavení lze automaticky použít i u přidávání widgetů na domovskou obrazovku a tím dosáhnout toho, že widget nemůže být přidán nenastavený.

5.4 Uživatelská rozhraní

Systém Android disponuje domovskou obrazovkou (launcher), která představuje mřížku (viz obrázek 5.3), do níž je možné vkládat buď zástupce aplikací, nebo widgety. Jelikož rozměry buněk mřížky mohou být na každém zařízení jiné (z důvodů různých verzí systému, jiných launcherů či velikostí displejů), navrhování vzhledů widgetů může být složitější než návrh vzhledu aplikace. Navíc, kvůli možnosti změnit velikost widgetu, je často potřeba pro jeden widget navrhnout více podob vzhledů. Toto neplatí na zařízeních s API menší než 16, kde nelze měnit velikost widgetu, takže je také potřeba rozhodnout, který vzhled widgetu bude výchozí, protože jej nebude možno na některých zařízeních měnit.

Návrhy widgetů byly koncipovány tak, aby byly správně zobrazeny na co nejmenší počet buněk při teoretické velikosti buňky 80dp na šířku a 100dp na výšku.

Obrázek 5.3: Mřížka domovské obrazovky

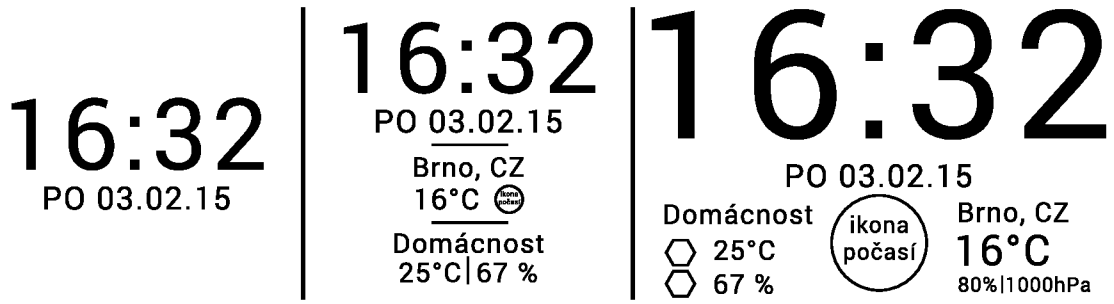
Widget hodin

Jedním z možných widgetů by mohly být hodiny s aktuálními informacemi o počasí. Tento typ widgetu je velmi populární (dokonce je často použit jako výchozí aplikace pro hodiny v daném systému). S kombinací informací z inteligentní domácnosti a venkovních údajů by tento widget dokonce mohl nabývat většího přínosu pro uživatele. Pro zobrazení aktuálního počasí je možné vybrat z několika poskytovatelů:

- **YahooWeather**
- **yr.no**
- **OpenWeatherMap**
- **Vlastní předpověď na serveru**[1]

Takový widget by mohl spadat do kategorie informačních widgetů a frekvence aktualizace by nemusela být velmi vysoká kvůli ne často měnícím se meteorologickým podmínkám (předpokládaný minimální interval aktualizace 2krát za hodinu).

Na obrázku 5.4 jsou znázorněny navržené vzhledy widgetu hodin. Nejvíce vlevo je zobrazen widget bez doplňujících informací, který je možné zobrazit na nejmenším počtu buněk (předpoklad je 2 na šířku a 1 na výšku). Uprostřed je widget zahrnující již informace o počasí a domácnosti (2×2 buňky). Napravo je nejdetailnější vzhled widgetu zahrnující také informace o vlhkosti a tlaku v daném městě.

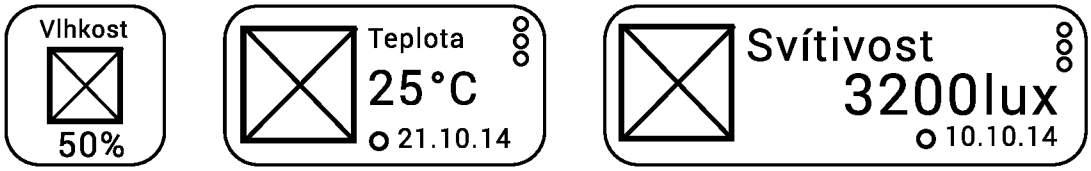


Obrázek 5.4: Grafické návrhy widgetu hodin

Widget zařízení

Druhým navrhovaným widgetem je zařízení z inteligentní domácnosti. U tohoto widgetu je možné, že by uživatel chtěl získávat informace v reálném čase, a proto předpokládaný minimální časový interval aktualizace je 10 vteřin. Na obrázku 5.5 jsou znázorněny návrhy widgetu zařízení. Čtverec s uhlopříčky představuje ikonku daného zařízení s vyobrazením aktuálního stavu (pokud je aktor světla vypnutý, žárovka nesvítí a naopak). Pokud má uživatel domácnosti dostatečné oprávnění (viz sekce 2.3), místo hodnoty senzoru se v případě použití aktoru zobrazí přepínač pro změnu stavu. Widget také obsahuje informaci o poslední aktualizaci hodnot ze serveru. Tuto skutečnost představuje kolečko s datem vedle něj (ve výsledné grafice se předpokládá nějaká ikonka synchronizace místo kolečka).

[1] Při použití vlastních předpovědí počasí by server mohl případně využívat informace z venkovních senzorů uživatelů a vytvořit tak velmi přesnou předpověď v určitém místě.



Obrázek 5.5: Grafické návrhy widgetu zařízení

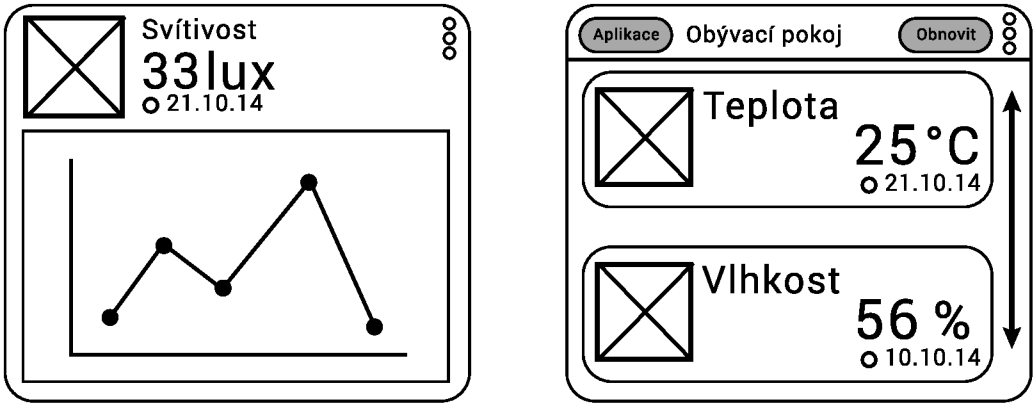
Widget zařízení s grafem

Rozšiřuje widget zařízení o historii hodnot v podobě grafu. Na obrázku 5.6a je představen návrh tohoto widgetu (části obrázku představují stejné informace jako ve widgetu zařízení). Hodnoty grafu bude možné zobrazovat podle tří kategorií:

- **Denní data** – Jsou získávána s půlhodinovým intervalem a obsahují informace každou hodinu po dobu jednoho dne.
- **Týdenní data** – Jsou získávána s 12hodinovým intervalem obsahující informace každou hodinu po dobu jednoho týdne.
- **Měsíční data** – Interval aktualizace je 24 hodin, informace jednou denně po dobu měsíce.

Widget umístění

Posledním navrženým widgetem je widget kolekce. Zobrazuje všechna zařízení ve vybrané místnosti. Jelikož může být žádoucí sledování informací o dané místnosti v reálném čase, je povoleno aktualizovat tento widget velmi často podobně jako widget zařízení – maximálně jednou za 10 vteřin. Tento widget je představen na obrázku 5.6b. Widget znázorňuje jednotlivé senzory se vzhledem widgetu zařízení (kvůli lehčímu pochopení obrázku), šipka představuje možnost posouvání obsahem nahoru a dolů.



(a) Zařízení s grafem

(b) Umístění

Obrázek 5.6: Návrhy widgetů

5.5 Persistence dat

Jelikož widgety musí být alespoň informativně funkční i po restartování zařízení a případném spuštění bez internetového připojení, je potřeba nejdůležitější informace ukládat do persistentní paměti. K těmto účelům lze nejčastěji používat buď `SharedPreferences`, které umožňují ukládat primitivní datové typy jako klíč – hodnotu do vnitřního úložiště. Data jsou uložena do souboru ve složce dané aplikace a dokážou přetrvat ukončení aplikace a dokonce i restart systému. Dalším způsobem je `ContentProvider`, který funguje jako vnitřní databáze aplikace[2]. Často tento způsob umožňuje pokročilejší manipulaci s daty, jako například doplňování textu na základě dat (tzv. *autocomplete*) nebo je možné poskytovat data jiným aplikacím. Také v případě widgetu s grafem by bylo vhodné vygenerovaný obrázek grafu ukládat do vnitřní paměti, aby nebylo potřeba ukládat mnoho grafových hodnot, ze kterých by se graf vygeneroval.

[2]Je pravděpodobné, že v budoucnu bude aplikace ukládat část dat ze serveru lokálně díky `ContentProvider`u a tudíž widgety také přejdou na tento způsob ukládání.

Kapitola 6

Implementace

V této kapitole je popsána problematika implementace s popisem nejdůležitějších částí této práce. Je zde obsaženo, jakým způsobem se widgety aktualizují (sekce 6.3), dále jak data přetrvají přístupné (sekce 6.4) a nakonec z jakých částí se musí widget v tomto frameworku skládat (sekce 6.5).

Widgety jsou implementovány pro systém Android v jazyce Java. Cílová verze je API 22, ale zpětná kompatibilita u většiny widgetů je zaručena až do API 10. Některé widgety musely být na starších zařízeních zakázány kvůli nefunkčnosti jistých prvků viz sekce 4.2.

6.1 Konfigurace widgetů

Všechny widgety v této práci využívající systému inteligentní domácnosti musí být navrženy tak, aby používaly konfigurační aktivitu, která bude stejná pro všechny a zahrnuje ověření, zda je uživatel přihlášen do aplikace, aby bylo možné načíst uživatelská data. Aktivita také provádí nezbytné operace pro načtení dat z domácnosti. Každý widget může potřebovat nastavovat jiné informace, a proto je nutné, aby balíček widgetu obsahoval konfigurační fragment, ve kterém mohou být jiné kontrolní prvky a jiná nastavení pro různé widgety. Tento fragment je vkládán do konfigurační aktivity během vytvoření (metoda `onCreate()`) a určování výběru fragmentu probíhá na základě názvy třídy widgetu, který je přidáván na domovskou obrazovku. Uživatel dále může ovládat fragment konfigurace stejným způsobem, jako by byl v aplikaci. Po dokončení nastavení jsou informace uloženy do widgetu a případně do všech jeho persistentních objektů (sekce 6.4). Když je tedy aktivita ukončena úspěchem, spustí se aktualizační služba, která se pokusí získat aktuální data pro daný widget (v případě, že již v telefonu nejsou aktuální). Pokud je konfigurován již přidaný widget, služba musí po dokončení nastavování widget znovu nainicializovat (kvůli možnosti změnit typ widgetu v nastavení apod.).

6.2 Uživatelská rozhraní

Na obrázcích B.1a, B.1d, B.1e a B.1f jsou znázorněny výsledné vzhledy navržených widgetů. U widgetu hodin byl pro získávání počasí použit poskytovatel **OpenWeatherMap**, ikony pro aktuální počasí[1] jsou implementovány jako zobrazení fontu, který kvůli omezení widgetů musí být převáděn do obrázku a ten je teprve vykreslen.

[1] Získané z projektu http://erikflowers.github.io/weather-icons/

6.3 Aktualizace widgetů

Aktualizace widgetů probíhá, kvůli časovému omezení systémové aktualizace (sekce 4.2), skrze službu `WidgetService`, která je spuštěna, když je widget přidán na domovskou stránku. Tato služba zařizuje veškeré manipulace s widgety. Widget samotný obsahuje pouze vlastní logiku, jak se má vykreslovat, aktualizovat svá data a ukládat je do persistentní paměti. Na obrázku A.1 je znázorněn způsob vyhodnocení aktualizace widgetů.

Každé spuštění služby, pokud není spuštěná se speciálním příkazem (sekce 6.3.1), znamená, že jeden z widgetů potřeboval aktualizovat a v minulosti byl nastaven čas další aktualizace. Při tomto spuštění se také vypočítá čas následující aktualizace na základě nejkratšího aktualizačního intervalu widgetu – služba je tedy spouštěná vždy s frekvencí nejkratšího intervalu. Jelikož ale jiné widgety nepotřebují aktualizaci, jsou v aktualizační metodě přeskočeny a není na ně brán zřetel během aktualizace dat ze serveru.

6.3.1 Speciální případy spuštění služby

Službu je také možné spouštět s definovanými parametry neboli `Intenty` a tím dosáhnout specifického chování. Všechny tyto případy jsou popsány na obrázku A.1.

Přepnutí aktoru Je vždy vyvoláno požadavkem z widgetu při kliknutí na přepínač. Služba přijímá spolu s parametrem obsluhy aktora také jeho `id` a `id` brány, se kterou je aktor spárován. Přepnutí je obslouženo ve dvou fázích kvůli zabránění nekonzistentních stavů mezi aktory a získání aktualizace stavu z aplikace:

- **Požadavek** (`changeWidgetActorRequest`) nejprve vyhledá ve všech použitých widgetech aktor s požadovanými parametry. V druhém kroku jsou zablokovány přepínače ve všech widgetech, aby nemohlo dojít k nekonzistentnímu stavu mezi přepínači, a je odeslán požadavek přepnutí na server. Tento stav přepínače trvá tak dlouho, dokud ze serveru nepřijde nějaká odpověď nebo je požadavek zahozen kvůli vypršenému času. V případě neúspěšného požadavku se aktor přepne zpět do původního stavu.
- **Výsledek** (`changeWidgetActorResult`) je vyvolán po úspěšném ukončení požadavku na server. Aby bylo možné zobrazovat aktuální stav aktora i když uživatel přepne stav přímo v aplikaci, vyvolává se broadcast `BROADCAST_ACTOR_CHANGED`, aby se na všech místech v aplikaci (tj. i widgetech) překreslil stav na aktuální. Služba v této situaci tedy pouze zavolá aktualizaci dat widgetu a ten se poté překreslí.

Změna velikosti widgetu Jak již bylo zmíněno v sekci 4.2, systém Android dokáže měnit velikost widgetů. Při změně rozměrů může widget překreslit svůj vzhled (pokud má tuto možnost naimplementovanou). Jelikož všemi widgety zařizuje služba, `AppWidgetProvider` při zmenšení či zvětšení pouze zjistí nové rozměry widgetu, které jsou předávány spolu s broadcastem změny velikosti[2], a předá je službě spolu s příznakem změny. Ta zavolá v objektu widgetu metodu `handleResize()`, které předá parametry rozměrů a její rolí je buďto změnit aktuálně použitý layout, nebo neudělat nic.

[2]Framework widgetů používá pouze parametry `OPTION_APPWIDGET_MIN_WIDTH` a `OPTION_APPWIDGET_MIN_HEIGHT` (existují také varianty `_MAX_`).

Odstranění widgetu Framework také využívá možnosti, kterou poskytuje operační systém, a to odchycení broadcast zprávy, když je widget odstraněn. Při této zprávě se spustí služba s parametrem odstranění widgetu. Při obsluze se nejprve odstraní persistentní data (`SharedPreferences` daného widgetu) a následně se objekt widgetu vymaže z cache uložených widgetů. Po úspěšném dokončení odstranění se vždy musí přepočítat čas následující aktualizace a pokud se nenajde žádný další používaný widget, jsou všechny prostředky služby označeny k vymazání a ta je ukončena.

6.3.2 Aktualizace dat ze serveru

V rámci ušetření požadavků na server a celkového objemu dat přenášeného mezi telefonem a serverem jsou widgety nepožadující aktualizaci přeskočeny a u zbytku jsou získány referenční objekty[3], které jsou posléze seskupeny a zpracovávají se hromadně. V případě, že se načtení dat z nějakého důvodu nepovede (proběhne vyhození výjimky `AppException`[4]), automaticky se u všech widgetů nastaví příznak nepovedené aktualizace a každý widget má ve své režii, zda nějakým způsobem dá uživateli najevo, že data nejsou aktuální. Při následující povedené aktualizaci se tento příznak vynuluje a do persistentní paměti se uloží aktuální data, která se takto pořád přepisují a přístupná jsou pouze posledně získaná.

6.3.3 Aktualizace času

Aktualizace času musí probíhat asynchronně vůči běžné aktualizaci. Tyto asynchronní zprávy vysílá systém jako broadcast a tudíž služba má vlastní broadcast receiver, který má zaregistrovaný příjem těchto zpráv[5]. Po přijetí této zprávy se překreslí všechny widgety hodin. Kvůli kompatibilitě se staršími API je nutno překreslit celý widget a ne pouze čas, a proto je widget hodin vždy vykreslován s naposledy získanými daty ze serveru.

6.4 Ukládání dat

Aktivní widgety představují objekty uchované ve službě frameworku. Pokud je ale telefon restartován nebo systém službu ukončí (např. pro nedostatek volné paměti), všechny objekty widgetů jsou zdestruovány a tudíž s nimi nelze manipulovat. Aby bylo možné objekt widgetu obnovit, tzn. znovu instanciovat, jsou ukládány nezbytné informace o objektu do persistentní paměti. K uchovávání důležitých informací widgetů byly použity `SharedPreferences`, které ke každému widgetu ukládají tyto informace:

- **Id widgetu** – slouží k identifikaci v rámci systému
- **Název třídy widgetu** – díky ní služba ví, který typ widgetu má znovu vytvořit.
- **Aktuálně použitý layout** – může se změnit, když uživatel změní velikost widgetu
- **Interval aktualizace** – čas v milisekundách
- **Čas poslední aktualizace** – ve formátu Unixového času
- **Id uživatele, který widget vytvořil**

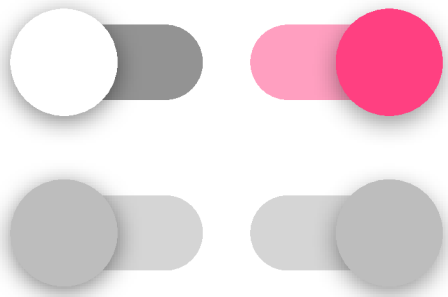
Widgety se ale mezi sebou liší, a tudíž je nutné ukládat části widgetů jako samostatný persistentní objekt, který může být znovupoužit pro jiné widgety.

[3] Pro každý widget může být jiný – `Facility` pro widget zařízení, `Location` pro umístění atd.
[4] `AppException` je vnitřně používaný typ výjimek, které slouží k identifikaci známých problémů v aplikaci.
[5] Jsou použity broadcasty `ACTION_TIME_TICK`, `ACTION_TIMEZONE_CHANGED` a `ACTION_TIME_CHANGED`.

Persistentní objekty Jsou objekty mající vlastní logiku a schopnost vykreslování do widgetu. Objekty nejsou nijak propojeny a každý widget jich může používat kolik potřebuje ke své funkčnosti. Jsou součástí souboru `SharedPreferences` widgetu a ukládají se s definovanou předponou u klíče požadované hodnoty. Všechny tyto objekty by neměly spoléhat na bytí v instanciovaném stavu, protože widget může být kdykoliv mezi aktualizacemi odstraněn, a všechny potřebné neuložené informace by měly být schopny obnovení z persistentních hodnot. Existují dva typy objektů – **s grafickým rozhraním** a **pouze data**.

Mezi objekty s grafickým rozhraním patří např. zobrazení lokace či hodnoty zařízení. Ta se může zobrazovat podle typu hodnoty zařízení jako text s jednotkou (např. 25°C). Druhou možností je zobrazit přepínač hodnoty aktoru, který musí být simulován obrázky kvůli 4.2. Přepínač (viz obrázek 6.1) je vzhledem podobný do klasického kontrolního prvku, ale postrádá animaci. Do budoucna, kdy aplikace bude používat více typů přepínacích hodnot, bude potřeba rozšířit možnosti zobrazení.

Druhá kategorie, pouze datové objekty, mezi něž patří objekt nastavení widgetu či počasí, nevykreslují žádná data přímo do widgetu, ale pouze poskytují informace, které jsou uložené persistentně.



Obrázek 6.1: Simulovaný kontrolní prvek – přepínač
(vlevo vypnutý stav, vpravo zapnutý, dole zakázané stavy)

6.5 Struktura balíčku widgetu

Jelikož widgety této práce používají vlastní způsob aktualizace, je vyžadováno, aby se skládaly z daných částí pro zajištění správné funkčnosti. Nově implementovaný widget je tedy vytvářen ve složce (balíčku) `java/widget` a jeho jednotlivé části pak v podsložkách:

- `widget/configuration` – Musí obsahovat konfigurační fragment.
- `widget/data` – Třída s logikou widgetu.
- `widget/receivers` – Zde musí být implementován potomek třídy `AppWidgetProvider`[6]
- `widget/persistence` – Nepovinně může widget rozšiřovat možnosti ukládání persistentních objektů.

[6]Ikdyž potomek třídy `AppWidgetProvider` může být bez implementace, systém Android vyžaduje, aby existoval, jinak widget nebude funkční.

Kapitola 7

Testování a optimalizace

Tato kapitola se zabývá testováním chování a vzhledů widgetů na různých zařízeních a různých domovských obrazovkách a také způsoby optimalizace služby aktualizace.

7.1 Testování vzhledu

Jelikož jsou widgety součástí jiného procesu – domovské obrazovky, různá zařízení nehrají příliš velkou roli v testování. Důležitým faktorem ale jsou různé aplikace domovské obrazovky (launcherů).

Testování zařízení

Widgety byly otestovány na zařízeních uvedených v tabulce 7.1. Některá zařízení byla simulována oficiálním emulátorem Androida. Toto testování nezjistilo žádné problémy, jelikož, jak již bylo zmíněno, funkčnost se zásadně neliší mezi verzemi systému.

Zařízení	**Emulátor**	**Verze systému**	**Displej**	
			Velikost [″]	**Rozlišení**
OnePlus One		5.0.2	5.5	1080 × 1920
HTC One		5.0.2	4.7	1080 × 1920
LG Nexus 4	✓	4.4	4.7	768 × 1280
Nexus S	✓	4.0.3	4.0	480 × 800
Nexus One	✓	4.0	3.7	480 × 800

Tabulka 7.1: Tabulka použitých zařízení pro testování

Testování domovských obrazovek

Pro testování aplikací domovské obrazovky bylo použito zařízení OnePlus One se systémem Lollipop. Přehled testovaných launcherů je v tabulce 7.2. Na základě tohoto měření bylo vyvozeno, že většina testovaných aplikací domovské stránky nedokázaly správně zařizovat změny velikosti widgetů. Bohužel po prostudování chování widgetů jiných aplikací se tento problém vyskytoval u všech a dá se tedy předpokládat, že chyba je na straně launcheru.

Řešením tohoto problému by bylo vytvoření velmi malého layoutu pro každý widget, což by bylo velmi obtížné, protože některé widgety by zkrátka nemohly poskytnout žádné informace. Výsledkem je tedy předpoklad, že uživatelům těchto launcherů tento problém nevadí a zřejmě nakonec přizpůsobí velikost widgetu tak, aby se zobrazoval správně.

Název aplikace	**Dodržení rozměrů**	**Jiný problém**
výchozí launcher	✓	
dodol launcher		
Google Launcher	✓	
GO Launcher EX		
Stock Launcher		Nefunkční změna velikosti
Nova Launcher		
EverythingMe Launcher	✓	Divně chovající se změna velikosti

Tabulka 7.2: Přehled testovaných domovských stránek – launcherů

7.2 Optimalizace služby

Díky implementaci služby aktualizace widgetů je možné některé požadavky widgetů shlukovat a tím optimalizovat potřebné zdroje k aktualizaci.

Vytížení paměti

Systém Android spravuje prostředky aplikací vlastním způsobem – monitoruje aktuální stav paměti, a pokud začíná být paměť příliš obsazená, nejprve zkouší upozorňovat spuštěné aplikace na tento stav voláním metody `onTrimMemory()`. Metoda se volá vždy s parametrem, jak velké množství prostředků je potřeba uvolnit (tzn. jak špatně na tom systém s pamětí je). Pokud aplikace je na pozadí (platí i pro služby) a zabírá nějaké systémové prostředky, měla by v této metodě uvolnit nejméně důležité objekty a data. Pokud se takto nestane a systém pořád nebude mít dostatečné množství volné paměti, začne ukončovat aplikace, které paměť zabírají. Kvůli zpětné kompatibilitě s nižšími verzemi systému není v tomto projektu u služby použita metoda `onTrimMemory()`, ale starší verze `onLowMemory()`, která je volána pouze při větším nedostatku paměti. V případě, že by tato metoda byla ignorována, je možné, že by systém službu aktualizace po nějakém čase ukončil a to by znamenalo, že na nějakou dobu nebude možné widgety aktualizovat.

Vytížení procesoru

Aktualizace widgetů má smysl, pouze pokud uživatel aktuálně používá telefon, tzn. že má zapnutý displej telefonu. Ve službě je proto implementovaný broadcast receiver, který má zaregistrovanou zprávu zapnutí/vypnutí displeje a pokud je displej vypnutý, zastaví se provádění aktualizací. K obnovení aktualizací dojde při zapnutí displeje a aktualizace pak pokračují běžným způsobem.

Množství přenesených dat

Problémem použití některých widgetů je (konkrétně těch s možností časté frekvence aktualizace), že by kvůli aktualizaci mohly přenášet velké množství dat a tím překročit limit u operátora. Proběhlo tedy měření (viz tabulka 7.3), kolik se přenese dat při jednom požadavku během aktualizace jednoho až čtyřech widgetů zařízení a následně teoretické spočítání kolik by bylo možné přenést za jisté časové úseky. Podle měření 1 widget při aktualizaci přenese přibližně 700 bytů dat. Díky optimalizaci služby, která shlukuje požadavky aktualizace, při aktualizaci 2 widgetů data nevzrostla dvojnásobně, ale pouze o 200 bytů. To stejné se projevuje při aktualizaci více widgetů, což znamená, že data jednoho zařízení obsahují přibližně 200 bytů informací.

	Přenesená data [B]		
Widgety	**RX**	**TX**	**Suma**
1	394	314	708
2	554	346	900
3	714	394	1108
4	874	426	1300

Tabulka 7.3: Přenesená data v jednom požadavku widgetů zařízení

V tabulce 7.4 je ukázáno teoretické množství přenesených dat v časových úsecích minuta, hodina, 4 hodiny, 12 hodin a den, widgety zařízení byly aktualizovány s největší možnou frekvencí, tj. co 10 vteřin. Tato data by odpovídala, pokud by telefon byl aktivní po celou dobu časového intervalu, nicméně reálná data by byla ještě menší a to na základě doby používání telefonu uživatelem. Z toho vyplývá, že widget zařízení i přes častou aktualizaci nepřenáší příliš mnoho dat, nicméně bylo by možné přidat do nastavení widgetu možnost aktualizace pouze, když je telefon připojen k Wi-Fi síti, což bylo do výsledné aplikace doimplementováno.

	[kB]		[MB]		
Widgety	**minuta**	**hodina**	**4 hodiny**	**12 hodin**	**den**
1	4,15	248,91	0,97	2,92	5,83
2	5,27	316,41	1,24	3,71	7,42
3	6,49	389,53	1,52	4,56	9,13
4	7,62	457,03	1,79	5,36	10,71

Tabulka 7.4: Množství přenesených dat widgetů zařízení v časových úsecích

Kapitola 8

Závěr

V rámci zadání jsem se seznámil s platformou inteligentní domácnosti, prostudoval jsem možnosti vytváření widgetů pro systém Android a posléze navrhl sadu widgetů pro aplikaci inteligentní domácnost. Práci jsem pojal spíše obecněji než pouze vytvořit několik widgetů a snažil jsem se sjednotit widgety do jednoho celku, který představuje jakýsi framework. Tento úkol byl úspěšný a framework nyní dokáže řídit několik typů widgetů. Pokusil jsem se tedy otestovat widgety a framework a výsledkem bylo, že díky sjednocení widgetů do frameworku bude podstatně jednodušší implementovat další widgety nebo také rozšiřovat funkčnost stávajících. Druhým pozitivním faktorem je, že je možné nad widgety provádět různé optimalizace, co již bylo částečně aplikováno.

Do budoucna bude potřeba rozšiřovat seznam akceptovatelných typů zařízení ve widgetu (připravovat další simulované kontrolní prvky pro jiné druhy aktorů). Také určitě je možné rozšířit nabídku widgetů například o widget s více senzory a aktory či widget kolekce s uživatelem vybranými zařízeními. Nejvíce nadějně vypadá widget hodin, u kterého by bylo možné implementovat různé heuristiky předpovědí na základě informací senzorů uživatelů. Co se týče frameworku, tak je určitě možnost, jak jej více optimalizovat, aby zatěžoval systém co nejméně.

Dle mého názoru jsou widgety obecně velmi špatně zdokumentované, protože všechny zdroje (knihy i oficiální stránky pro vývojáře) probírají widgety pouze okrajově. V podstatě obvykle zahrnují pouze několik základních principů a pár příkladů, což není dostatečné pro pochopení pokročilých možností vývoje v případě, že je žádáno vytvořit složitější widgety. Tato práce by tedy mohla být přínosem pro všechny budoucí řešitele a vést ke snadnějšímu pochopení vývoje widgetů.

Literatura

[1] Smart house. *Engineering & Technology*, ročník 7, č. 6, 2012: s. 42 – 43, ISSN 17509637.

[2] *Architektura inteligentní domácnosti* [online]. https://ant-2.fit.vutbr.cz/projects/iot/wiki/Architektura, [cit. 2015-05-01].

[3] Android Developers: *BroadcastReceiver* [online]. http://developer.android.com/reference/android/content/BroadcastReceiver.html, [cit. 2015-04-16].

[4] Android Developers: *Services* [online]. http://developer.android.com/guide/components/services.html, [cit. 2015-04-16].

[5] Android Developers: *App Widgets* [online]. https://developer.android.com/guide/topics/appwidgets, [cit. 2015-04-28].

[6] Android Developers: *Widgets* [online]. https://developer.android.com/design/patterns/widgets.html, [cit. 2015-04-28].

[7] Android Developers: *Activity* [online]. http://developer.android.com/reference/android/app/Activity.html, [cit. 2015-05-03].

[8] Android Developers: *Dashboards* [online]. https://developer.android.com/about/dashboards/index.html, [cit. 2015-05-07].

[9] Android Developers: *Scheduling Repeating Alarms* [online]. https://developer.android.com/training/scheduling/alarms.html, [cit. 2015-05-08].

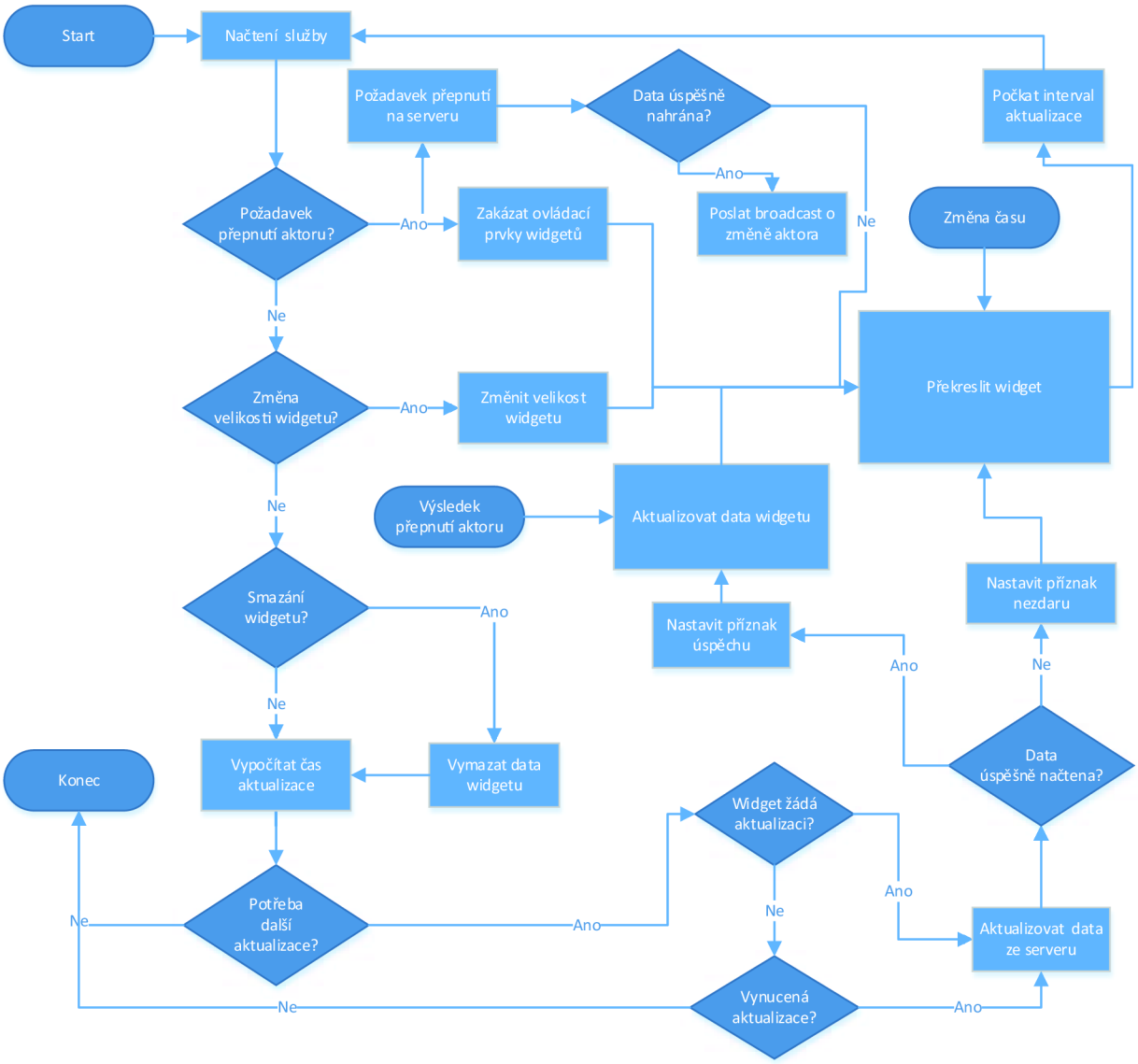
[10] Android Open Source Project: *Codenames, Tags, and Build Numbers* [online]. https://source.android.com/source/build-numbers.html, [cit. 2015-05-07].

[11] Android Open Source Project: *The Android Source Code* [online]. http://source.android.com/source/index.html, [cit. 2015-05-10].

[12] Meier, R.: *Professional Android 4 Application Development*. John Wiley & Sons, Inc., 2012, ISBN 978-1-118-10227-5, 1, 15 s.

Příloha A

Stavový diagram

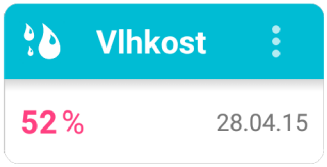


Obrázek A.1: Stavový diagram běhu služby widgetů

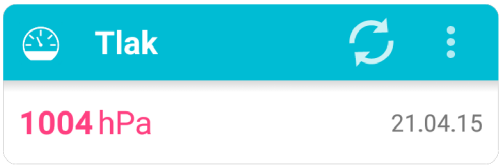
Příloha B

Grafické podoby widgetů

(a) Zařízení malé



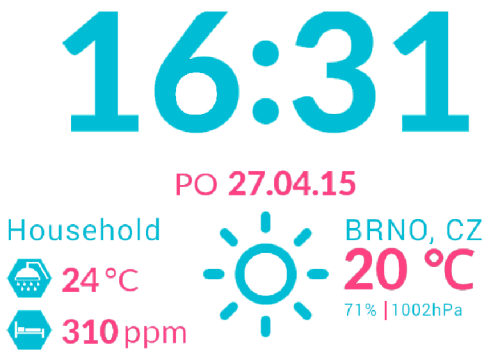
(b) Zařízení střední



(c) Zařízení velké

(d) Zařízení s grafem

(e) Umístění



(f) Hodiny s počasím

Obrázek B.1: Grafické podoby widgetů

Příloha C

Obsah CD

- `/zprava/` – tato zpráva ve formátu PDF
- `/zprava/latex/` – zdrojové kódy technické zprávy ve formátu LaTeX
- `/src/` – zdrojové kódy aplikace
- `/apk/` – instalační soubor aplikace
- `/README`