VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

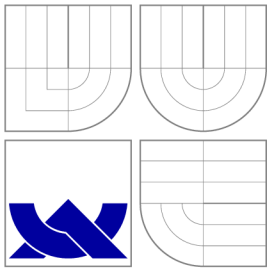
PROSTŘEDÍ PRO TESTOVÁNÍ MOBILNÍ APLIKACE INTELIGENTNÍ DOMÁCNOST

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

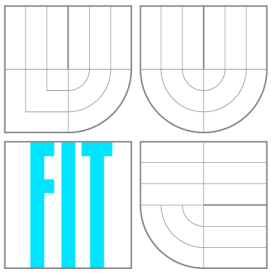
AUTOR PRÁCE
AUTHOR

MARTINA KŮROVÁ

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

PROSTŘEDÍ PRO TESTOVÁNÍ MOBILNÍ APLIKACE INTELIGENTNÍ DOMÁCNOST

FRAMEWORK FOR TESTING OF MOBILE APPLICATION IMPLEMENTING INTELLIGENT HOME CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTINA KŮROVÁ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN ŽÁDNÍK, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Bakalářská práce se zabývá návrhem a implementací vhodného způsobu testování mobilní aplikace sloužící pro vzdálené ovládání inteligentní domácnosti vyvíjené v rámci projektu Internet of Things na FIT VUT. Hlavním cílem práce je vytvoření zautomatizovaných testovacích scénářů pro ověření správné funkčnosti aplikace a ve výsledku tak přispět ke zvýšení její stability. V první části práce se nachází představení systému inteligentní domácnosti a testované aplikace, dále je uveden teoretický přehled operačního systému Android a testování mobilních aplikací na této platformě. Další část je věnována popisu implementaci navrženého způsobu testování aplikace a na závěr je uvedeno zhodnocení výsledků vytvořených testů a jejich srovnání.

Abstract

This thesis deals with testing of mobile application implementing inteligent home remote control developed under the project Internet of Things at FIT BUT. The main goal is to create sets of automated test scenarios to verify correct functionality of the application with the intention to increase its stability. The first part presents inteligent home architecture, further continues a theoretical overview of operating system Android and testing of mobile applications with the Android operating system. Other parts are devoted to describing an implementation of the proposed method for testing of the application. Conclusion stated assess the quality of developed tests and evaluation of their results.

Klíčová slova

testování, automatizace, UiAutomator, Android OS, Apache Ant, inteligetní domácnost

Keywords

testing, automation, UiAutomator, Android OS, Apache Ant, intelligent home

Citace

Martina Kůrová: Prostředí pro testování mobilní aplikace inteligentní domácnost, bakalářská práce, Brno, FIT VUT v Brně, 2015

Prostředí pro testování mobilní aplikace inteligentní domácnost

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením Ing. Martina Žádníka, Ph.D. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Martina Kůrová
19. května 2015

Poděkování

Ráda bych poděkovala Ing. Martinu Žádníkovi, Ph.D. za odborné vedení, užitečné rady a především za trpělivost a ochotu věnovanou v průběhu vypracování mé bakalářské práce.

Dále děkuji svým rodičům za podporu po celou dobu mého studia na vysoké škole.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Testování mobilních aplikací i testování v širším slova smyslu je v dnešní době stále vnímáno jako méně důležitá a nezajímavá část vývojového cyklu softwaru. Často bývá zkracováno nebo úplně opomíjeno kvůli dodržení termínů. Vývoj mobilních aplikací dosahuje za poslední roky nevídaného růstu a stále patří mezi ty nejvíce rostoucí odvětví informačních technologií. Takto rozvinutý trh mobilních aplikací má za následek stav, kdy je aplikací dostatek a konkurence se předhánějí o zájem zákazníků. Začíná tedy více rozhodovat kvalita a stabilita těchto aplikací. I naproti tomuto faktu, jak vyplývá z nedávné studie podpořené společností IBM [5], většina vyvíjených mobilních aplikací není v současné době, dokonce ani na úrovni průměrně velké společnosti, testována.

Každá aplikace má chyby a pokud má být spolehlivá, musí být důkladně otestována a nalezené chyby opraveny. Tímto procesem se aplikace stávají stabilnějšími, a to činí z testování důležitou součást vývojového cyklu softwaru. Hlavně z důvodu, že neodradí budoucí uživatele v jejím používání. Testování je součástí životního cyklu aplikace a nesmí se podcenit. Platí to hlavně v případech, jedná-li se o aplikaci, na kterou má uživatel plně spoléhat nebo která nějakým způsobem ovlivňuje nebo řídí věci kolem nebo i místo něho. V této práci je navrhován vhodný způsob testování mobilní aplikace pro vzdálené ovládání inteligentní domácnosti, pomocí které lze získávat informace o domácnosti a patřičným způsobem ji i ovládat.

V první části práce se nachází teoretický rozbor, který se vzhledem k zadaní práce skládá ze tří částí, ke kterým je vhodné nastudovat detailnější informace. V kapitole 2 je přestaven systém inteligentní domácnosti, jehož je součástí aplikace, pro kterou je v této práci navrhováno testování. Následuje kapitola 3 s informacemi o operačním systému, na kterém aplikace běží a pro který budou přizpůsobeny testy a v poslední kapitole této části 4 je rozebírána hlavní teoretická oblast této práce, a to je testování z pohledu mobilních aplikací i jako součást vývojového cyklu softwaru. Následuje průzkum současných přístupů k testování mobilních aplikací na platformě Android a přehled k tomu určených nástrojů v kapitola 5. Dále je v kapitole návrhu 6 je více představena testovaná aplikace, způsoby jejího použití a jsou shrnuty požadavky na její testování. Nejpodstatnějšími částmi práce jsou kapitoly s popisem návrhu a implementace 7 navrženého způsobu testování mobilní aplikace, které se skládají z výběru vhodných nástrojů pro testování, tvorbou testovacích scénářů a způsobem jejich automatizace. V závěrečné kapitole 8 je uvedeno zhodnocení výsledků implementovaných testů a jsou nastíněny možnosti další rozšíření práce.

Kapitola 2

Inteligentní domácnost

Inteligentní domácností nazýváme budovu vybavenou moderní počítačovou a komunikační technikou, která předvídá a reaguje na potřeby uživatelů s cílem zvýšit jejich komfort a pohodlí. Prostřednictvím řízení všech technologií v domě a jejich interakcí s vnějším světem se lze zajistit i snížení spotřeby energií, zabezpečení domu, a nebo poskytnutí zábavy. Více o inteligentních domech se lze dočíst zde [12].

Instalace takových technologií do nových nebo již stávajících budov je perspektivně rostoucím oborem již několik let a je předmětem i projektu inteligentní domácnosti vyvíjené v rámci projektu IoT (Internet of Things) na FIT VUT.

Pro představení celého systému a jednotlivých jeho částí, které budou zmiňovány i v následujících kapitolách této práce, následuje popis architektury celého sytému inteligentní domácnosti a podrobnější rozbor aplikace, která slouží pro vzdálené ovládání teto domácnosti a která bude předmětem testování.

2.1 Popis architektury

Architektura celého systému inteligentní domácnosti, jak je zřejmé z obrázku 2.1, se skládá z několika vrstev. Patří mezi ně:

- koncové bezdrátové prvky
- adaptér
- server
- ovládací stanice

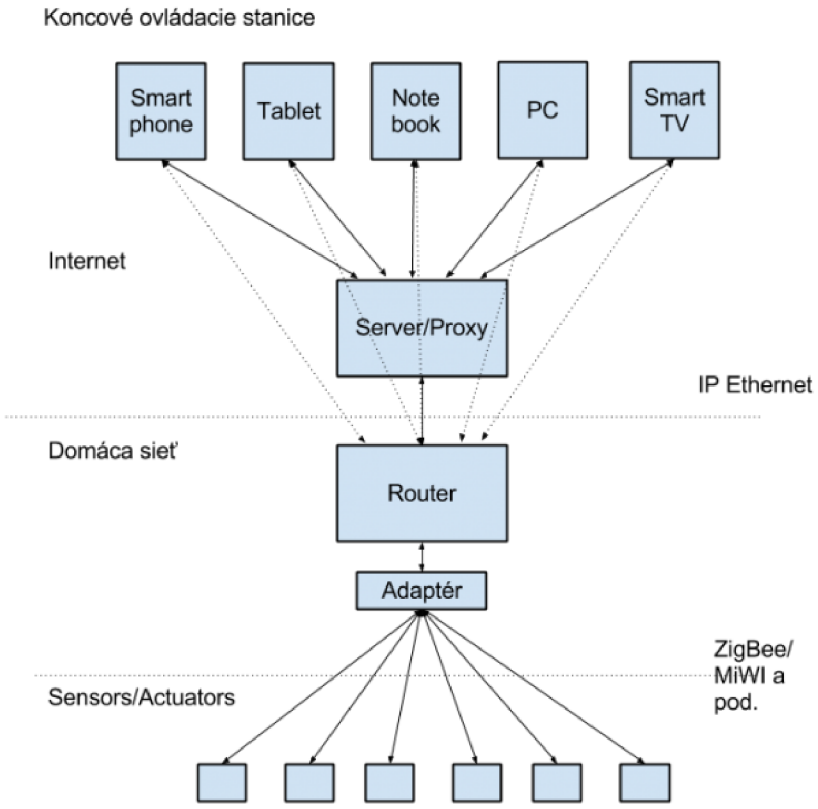
Na nejnižší úrovni se nacházejí jednotlivé bezdrátové koncové prvky. Podle funkce je rozdělujeme na senzory a aktory. Senzory slouží ke snímání hodnot nějaké veličiny z okolního prostředí. Naopak aktory přímo do okolního prostředí zasahují, tedy slouží k vykonání nějaké akce definované vyššími vrstvy systému. Například může jít o sepnutí zásuvky. Oba tyto prvky bezdrátově komunikují s představitelem další vrstvy - adaptérem.

Adaptér dokáže jednoznačně adresovat a následně i komunikovat s každým bezdrátovým prvkem přes jeho komunikační protokol, přičemž komunikace s prvky je podmíněna předchozím úspěšný spárováním prvku s adaptérem. Prostřednictvím této komunikace se posílají nasnímané hodnoty od senzoru směrem k vyšším vrstvám a naopak od vyšších vrstev se posílá pokyn pro aktor. Adaptér je jednoznačně identifikován pomocí celého čísla

nebo QR kódu. Díky této záležitosti se dokáže zaregistrovat do ještě vyšších vrstev - tedy na server.

Vrstva serveru obsahuje hlavní řídící systémy a databázi pro ukládání všech informací a dat o domácnosti a řídí další záležitosti v inteligentní domácnosti, jako je přihlašování uživatelů, registrování adaptérů nebo třeba řízení aktualizací snímaných hodnot senzory, které se následně ukládají do databáze.

Poslední vrstvou jsou pak koncové ovládací stanice mezi které mimo jiné patří i chytré mobilní telefony, které mohou disponovat různými aplikacemi. Právě aplikace pro vzdálené ovládání této inteligentní domácnosti, konkrétně pro platformu Android, bude předmětem testování této bakalářské práce.



Obrázek 2.1: Model architektury inteligentní domácnosti (převzato z [3])

2.2 Vzdálené ovládání

Jednou z možností vzdáleného ovládání inteligentní domácnosti je využití chytrého mobilního telefonu. V rámci celého systému se nyní pohybujeme na nejvyšší vrstvě, a to mezi koncovými ovládacími stanicemi. K ovládání inteligentní domácnosti je nutné mít v mobilním zařízení nainstalovanou aplikaci, která je vyvíjena pro operační systém Android, Windows Phone a Blackberry OS. Všechny aplikace jsou stále je vývoji a slouží pro dohled, kontrolu a ovládání v rámci domácí sítě i v případě, že se nacházíme mimo dům nebo teprve na cestě domů.

Aplikace komunikuje se serverem, ke kterému se připojuje kvůli registraci uživatele do aplikace, registraci adaptérů nebo zařízení, pro ukládání dat poskytnutých uživatelem a pro

načítaní informací o domácnosti, které se mají v aplikaci zobrazit.

Aplikace pro Android

Aplikace, která je předmětem testování této bakalářské práce je vyvíjena pro platformu Android. Momentálně je implementovaná registrace uživatele, přihlašování prostřednictvím Google účtu nebo sociální sítě Facebook. Lze přidat adaptér, kterého vlastníme. Aplikace disponuje čtyřmi uživatelskými rolemi, na jejichž správu má právo pouze vlastník adaptéru. Lze přidávat senzory i aktory a měnit jejich nastavení. K dispozici je také demo mód aplikace, který slouží pro demonstraci základních funkcí aplikace uživatelům, kteří adaptér nevlastní.

Kapitola 3

Android OS

Operační systém Android[1] pro mobilní telefony byl vytvořen před několika lety společností Google s hlavním cílem udržet tento systém otevřený a kompatibilní pro běh na různém hardwaru. I když se jedná o poměrně mladý operační systém, jeho oblíbenost rapidně vzrůstá. V listopadu roku 2014 se jeho podíl na trhu s chytrými telefony dotkl téměř 84%.[6] Vzhledem k rozšířenosti a obrovskému růstu popularity toho operačního systému roste poptávka po kvalitním programovém vybavení. Dostáváme se do doby, kdy aplikací je dostatek a rozhoduje jejich kvalita a stabilita.

V následujících kapitolách je stručně popsána architektura OS Android, dále jsou zmíněny nástroje pro vývoj a testování aplikací, které jsou využívány i v implementaci testování aplikace v rámci této práce. A v poslední řadě je vysvětlen princip grafiko-uživatelského rozhraní OS Android, jehož znalost je potřebná pro implementaci testů, a to zejména z toho důvodu, že je potřeba přistupovat ke konkrétním prvkům grafického uživatelského rozhraní a znát možné vlastnosti těchto prvků.

3.1 Architektura systému

Operační systém Android je založen na linuxovém jádře, jeho hlavní funkcí je implementace abstrakce hardwaru pro vyšší vrstvy systému a zabezpečení, správu paměti, správu procesů, přístup k síti a ovladačům vnitřních senzorů a komponent. Jednotlivé aplikace nepřistupují k funkcím jádra přímo, ale prostřednictvím Android API.

Android API řadíme do vrstvy knihoven obsahující řadu rozhraní pro vývoj aplikací. Funkce těchto knihoven jsou dostupné vývojářům prostřednictvím Android Application Framework - aplikačního rámce, který například zprostředkovává přístup k prvkům grafické-uživatelského rozhraní, hardwaru zařízení a nebo umožňuje spouštět další aplikace. Základní aplikace pro používání mobilního zařízení představují nejvyšší vrstvu systému.

Aby aplikace mohly v systému běžet, je k tomu potřeba přítomnost vrstvy Android runtime, která obsahuje virtuální stroj zvaný Dalvik, speciálně vytvořený pro OS Android se záměrem optimalizace pro mobilní zařízení, a to zejména v oblasti úspory energie a výkonu. Součástí této vrstvy jsou i základní knihovny jazyka Java, v němž jsou také aplikace pro OS Android vyvíjeny. Více o architektuře OS Android se lze dočíst v [11].

[1] Android - domovská stránka: http://www.android.com/

3.2 Vývoj a testování aplikací

Podobně jak je to u vývoje mobilních aplikací, tak i vývoj testovacích scénářů pro tyto aplikace se neobejde bez přítomnosti vývojového prostředí a dalších poskytovaných nástrojů.

Java

Vzhledem k tomu, že jsou aplikace pro OS Android vyvíjeny v programovacím jazyce Java, je výhodné v témže jazyce vyvíjet i testy. Pro překlad zdrojového kódu testovacích scénářů do spustitelné podoby je tedy nutné mít nainstalované *JDK*[2] *(Java Develepoment Kit)*, respektive *JRE*[2] *(Java Runtime Environment)*.

Nástroje Android SDK

Pro vývoj a ladění aplikací je dostupný balíček vývojových nástrojů Android SDK (Software Developement Kit), jehož značná část bude využívána i pro testování aplikace v této práci. Za zmínku stojí především *SDK Tools*[3], které obsahuje základní nástroje pro testování mobilních aplikací, správu virtuálních zařízení, Android emulátor a nástroj pro analýzu grafického layoutu. Další důležitou součástí je *SDK Platform-tools*[4], která obsahuje nástroje, které jsou závislé na verzi platformy. Klíčovým nástrojem je *ADB*[5] *(Android Debug Bridge)*, který slouží k pokročilému vývoji a ladění zařízení. Lze pomocí něj propojit zařízení, ať už reálné nebo emulované, s PC a tak ovládat zařízení z venku. Lze tak například do telefonu nainstalovat vyvíjenou nebo testovanou aplikaci a nebo volat Linuxové příkazy přímo na telefonu. Poslední významnou součástí je *Android SDK Platforms*[6] obsahující knihovny, systémový obraz, skiny emulátoru pro jednotlivé platformy.

Virtuální mobilní zařízení

Aplikace mohou být testovány na skutečném přístroji opatřeném operačním systémem Android nebo v jeho emulátoru. Pro vytváření virtuálních zařízení, které je možné v emulátoru spouštět, je k dispozici plugin *AVD*[7] *(Android Virtual Device)*. Emulátor je implementace virtuálního stroje Dalvik, vytvářející z něj platnou platformu pro spouštění Android aplikací jako pro jakékoliv jiné reálné fyzické zařízení s OS Android.

3.3 Grafické uživatelské rozhraní

Uživatelské rozhraní je v OS Android definováno pomocí jednotlivých aktivit. Každá jedna aktivita představuje jednu obrazovku vyskytující se v aplikaci. Pro jednotlivé obrazovky aplikace i pro jejich různé orientace lze tvořit layouty definované pomocí XML souboru ve speciálním formátu. Základní stavební kámen pro komponenty uživatelského rozhraní představuje třída *View*, která je rodičem všech zobrazovaných grafických prvků. Jednotlivé interaktivní prvky uživatelského rozhraní, jako jsou například textová pole, tlačítka, seznamy, checkboxy a další, spadají pod balík tříd `android.wigdet`.

[2]Zdroj: http://www.oracle.com/technetwork/java/javase/downloads/index.html
[3]Zdroj: https://developer.android.com/sdk/index.html
[4]Zdroj: http://developer.android.com/tools/sdk/tools-notes.html
[5]Zdroj: http://developer.android.com/tools/help/adb.html
[6]Zdroj: http://developer.android.com/tools/revisions/platforms.html
[7]Zdroj: http://developer.android.com/tools/help/avd-manager.html

Následuje výčet nejčastěji používaných tříd tohoto balíku, jejich popis a definice XML souboru při použití prvku v layoutu aplikace.

- *TextView* - nejjednodušší komponenta - textová položka, která nemůže být upravována uživatelem

```
<TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:clickable="true"
      android:text="Hello,␣I␣am␣a␣TextView" />
```

- *EditText* - editovatelná vrstva uživatelem umístěna nad TextView

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Place␣a␣hint␣here"
    android:inputType="Hello,␣I␣am␣a␣EditText" />
```

- *Button* - interaktivní tlačítko

```
<Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello,␣I␣am␣a␣Button" />
```

- *ImageButton* - tlačítko, které umožňuje zobrazit místo textu obrázek
- *ListView* - vertikální list s možností scrollování
- *Spinner* - rozbalovací nabídka
- *SeekBar* - posuvník hodnot
- *Toast* - pohled obsahující krátkou zprávu pro uživatele, která se zobrazí po danou dobu na obrazovce

Popis celé knihovny grafických uživatelských prvků i s příklady použití lze nalézt na stránkách pro vývojáře Androidu [1].

Kapitola 4

Testování mobilních aplikací

Testování znamená spouštění softwaru s účelem ověřování správných výsledků se záměrem zvýšení jeho kvality. V následujících kapitolách zahrnující testování a průzkum nástrojů k tomu určených je čerpáno z knihy *Co programátory ve škole neučí, aneb, Softwarové inženýrství v reálné praxi* [8] doplněné o informace z velmi podrobného průvodce testováním mobilních aplikací *Android Application Testing Guide*[7].

Základní dělení testování

- *Funkční* - Ověření, zda systém vyhovuje všem bodům specifikace a zda dělá to, co dělat má, provádí se většinou hned od začátku vývoje softwaru.
- *Uživatelské* - Testy prováděné budoucími uživateli, provádí se ve chvíli, kdy je aplikace už celkem hotová.
- *Výkonnostní* - Testy sloužící k získávání výkonnostních charakteristik měřeného systému při různých typech zátěže, případně i při různých konfiguracích daného systému.

Testovací scénář *(Test Suite)*
: slouží k testování aplikace, je tvořen několika testovacími případy, které na sebe musí navazovat a tvořit tak skupinu logicky navazujících kroků, jejichž účelem je otestovat vybranou funkčnost aplikace.

Testovací případ *(Test Case)*
: popisuje konkrétní akce prováděné s určitou softwarovou komponentou, je tvořen seznamem prováděných kroků a očekávaných výsledků. testovací případy se vytvářejí jak pro manuální tak i automatizované testy.

Testovací dokumentace *(Test Report)*
: Existují dva hlavní dokumenty k testování: *System Test Specification (STS)*, který obsahuje kompletní dokumentaci k testování (jednotlivá pravidla, definice, na čem se má testovat, jednotlivé testovací scénáře apod.), zjednodušeně říká co a jak testovat a *System Test Report (STR)* který slouží jako závěrečná zpráva po testování, obsahuje tedy informace, co a jak bylo otestováno a především s jakými výsledky.
Navíc existuje ještě jeden dokument, který neřadíme přímo mezi testovací dokumentace, ale je velmi významný při tvorbě požadavků na testy a vytváření testovacích případů a tím je *System Requirements Specification (SRS)*, který se skládá ze specifikace požadavků na vyvíjený systém, tedy jak má konečná aplikace vypadat a které funkce má vykonávat.

4.1 Principy testování

Hlavní rozdělení u testování mobilních aplikací je na testování funkcionálních a nefunkcionálních vlastností dané aplikace. První varianta je založena funkcích a vlastnostech, které popisují chování systému, jeho vstupy a výstupy a vyplývají ze specifikace aplikace. Testování nefunkcionálních vlastností pak zahrnuje všechno ostatní, od testování výkonu, použitelnosti, udržovatelnosti, spolehlivosti, přenositelnosti, přes zátěžové a stres testování až po testování bezpečnosti. Je testováním toho, jak systém pracuje.

- *Funkcionální testování (Functional testing)* - ověřuje obsah aplikace - obrázky, text, ovládací prvky, odkazy - jestli všechno funguje tak, jak má
- *Testování použitelnosti (Usability testing)* - nadstavba funkcionálního programování, boxy a tlačítka, splňující záměr/účel, snadná navigace, snadný přístup k nápovědě
- *Testování kompatibility zařízení (Device Interoperability Testing)* - zařízení mají velmi rozdílný výkon, různé rozlišení a hustoty displejů, a taky mnohdy všelijaká softwarová vylepšení. Z toho vyplývá, že je nutné provést testování na co možná nejvíce zařízeních - typech mobilních telefonů, ale i tabletů, co se týče rozlišeni obrazovky, kdy se testuje především responzivnost i verzí operačního systému, kde se naopak jedná spíše o funkčnost.
- *Integrační testování (Integration tesing)* - v systému se vyskytuje více komponent, je nutné otestovat každý subsystém a hlavně rozhraní mezi nimi
- *Testování výkonu (Pefrormance testing)* - schopnost systému přesně plnit své funkce a plnit je dostatečně rychle, provádí se analýza zdrojů, testuje se například CPU, GPU, operační paměti a paměťového úložiště. Pod tuhle část spadá i zátěžové testování, jehož hlavní testovací metriky zahrnují odezvu, výkon, objem dat, chyby transakce, serveru a využití sítě, a řadu dalších ukazatelů.
- *Regresní testování (Regression testing)* - znovuspouštění testů, zpravidla po vydání nové verze aplikace, účelem je ověření, že novými funkcemi nebo změnami v těch stávajících nebyly zaneseny chyby v již implementovaných a otestovaných částech aplikace

4.2 Automatizace

S postupem času byly vyvíjeny stále rozsáhlejší systémy a to mělo za následek i růst objemu testů. Při testování takových rozsáhlých systémů bylo nutné testování zautomatizovat pomocí různých softwarových nástrojů a technologií, které se testerovi snaží jeho práci jak ulehčit, tak zvýšit její efektivitu. Automatizované testování tedy snižuje čas a náklady spojené s testováním, prostřednictvím různých automatizačních nástrojů a metod. U mobilních aplikací existuje celá řada frameworků pro automatizaci testovacích scénářů, kdy test sám umí rozpoznat, zda se provedla správná akce či nikoliv. Samotné testování tak přebírá framework a tester se může zabývat tvořením dalších testovacích scénářů, úpravě stávajících testů či testováním testovacích případů, které nemohly být zautomatizovány.

Výhody automatických testovacích nástrojů

- *Rychlost* - Nejdůležitější vlastnost automatických nástrojů. Scénáře nejsou zpracovávány manuálně, ale strojově a bez prodlev. Jak uvádí *Ron Patton* v knize Testování Softwaru [9] : *Je prokázáno, že při správně napsaných testech dokáže automatický nástroj urychlit testování až 15x.*
- *Efektivita* - Samotné testování přebírá framework a tester se může zabývat tvořením dalších testovacích scénářů, úpravě stávajících testů či testováním testovacích případů, které nemohly být zautomatizovány.
- *Správnost a přesnost* - Neúnavnost stroje, který testovací scénáře provádí.

Nevýhody

Jako nevýhodu lze považovat nutnost neustálé údržby testů. Pokud je do stávající a otestované části implementace zanesena nějaká změna, automatizované testy musí být také aktualizovány a přizpůsobeny těmto změnám.

Kapitola 5

Nástroje pro testovaní mobilních aplikací

U mobilních aplikací existuje celá řada nástrojů pro automatizaci testovacích scénářů, kdy lze nakonfigurovat test tak, aby sám uměl rozpoznat, zda se provedla správná akce či nikoliv. Vzhledem k faktu, že se pro vývoj mobilních aplikací používá programovací jazyk Java, je drtivá většina frameworků pro psaní testovacích scénářů přizpůsobena tomuto jazyku.

5.1 Android JUnit

Android JUnit[1] je rozšíření klasických JUnit testů, což je testovací framework pro programovací jazyk Java. V současné době je celkem rozšířený, a proto je součástí i mnoha vývojových prostředí. Představuje testování jednotek, tozn. že jde o proces, při kterém testujeme funkčnost co nejmenších částí aplikace - tedy tříd, metod apod. Nedochází zde tedy přímo k volání Android API.

JUnit testy mají podobu obyčejných tříd, mohou se zde nacházet různé metody, jenž pomocí anotací přiřadíme k jednotlivým procesům testování. Základní třídou je *AndroidTestCase*, která poskytuje standardní metody setUp(), tearDown() a assert metody. Dále obsahuje metody pro testování práv (permissions). Rozšiřující třídou je *InstrumentationTestCase*, která se používá pokud potřebujeme využívat instrumentation metody. Specifičtější testovací třídy jako např. *ServiceTestCase*. Dalšími třídami pro testování jsou specifické třídy podle typu komponenty. Existují tedy třídy pro testování aktivit, služeb a poskytovatelů obsahu. Pro testování aktivit se nejčastěji používají dvě třídy, a to *ActivityUnitTestCase* a *ActivityInstrumentationTestCase2*, které obě dědí od základní testovací třídy InstrumentationTestCase.

Pro vyhodnocování průběhu testů se používají tzv. assert statements. Pomocí těchto metod můžete porovnávat reálné výsledky testovacího případu s těmi očekávanými. Android poskytuje i rozšiřující assert třídy *MoreAsserts* a *ViewAsserts*. První poskytuje metody např. pro kontrolu textu pomocí regulárního výrazu. Metody druhé třídy slouží k testování zarovnání objektů v uživatelském rozhraní.

[1] Android JUnit: http://developer.android.com/tools/testing-support-library/index.html#ajur-junit

5.2 Framework Robotium

Robotium[2] je framefork pro automatizované testování mobilních aplikací na platformě Android vhodný pro tvorbu funkčních, systémových a akceptačních testovacích případů i k testování uživatelského rozhraní. Robotium rozšiřuje Android JUnit testy a je navržen tak, aby bylo psaní testů uživatelského rozhraní jednoduché, rychlé a efektivní. Oproti samotným Junit testům tímto nástrojem se většinou testují aktivity k balíku GUI dostupné jako jar knihovna a pro její použití je tedy nutné tuto knihovnu přidat do adresáře `libs` testovacího projektu.

Pomocí Robotia můžeme otestovat aplikaci tak, jakoby jí procházel samotný uživatel. Jedná se o způsob testování, jako se používá u webových aplikací pomocí nástroje Selenium - tedy, že se framework chová jako uživatel, vidí to, co vidí uživatel, může klikat na prvky uživatelského rozhraní a dokonce přepínat mezi zobrazením na výšku a na šířku. Testovací případy napsané v Robotiu lze spouštět jak v emulátoru, tak i v reálném zařízení s operačním systémem Android. Další výhodou Robotium je možnost otestovat samotný apk soubor bez jeho zdrojových kódů. Poskytuje navíc i snadnou integraci s Mavenem či Antem. Více podrobností o frameworku Robotium lze nalézt zde [4].

Objekt Solo

Veškerý přístup k testovacím funkcím frameworku Robotium je přes objekt *Solo*. Tento objekt je potřeba vytvořit v metodě `setUp()` a předat konstruktoru instanci objektu *Instrumentation* a aktivitu pomocí metody `getActivity()`. V samotných testovacích metodách pak můžeme volat metody objektu Solo, mezi které patří například:

`getCurrentActivity()` - získání řetězce současné aktivity
`clickOnButton((String regex)` - kliknutí na tlačítko obsahující text
`clickOnButton(int)` - kliknutí na tlačítko pomocí indexu
`waitForText(text)` - čekání na zobrazení textu
`enterText(int index,String text)` - vložení text to textového políčka

Lokace GUI objektů

Z předcházejícího výčtu metod vyplývá, že jednotlivé prvky uživatelského rozhraní můžeme lokalizovat buď pomocí:

ID	`Solo.clickOnButton(R.id.btnOk);`
Pozice	`Solo.cklickOnButton(1);`
Textu	`Solo.clickOnButton("OK");`

ID elementu lze zjistit v adresáři `/res` dvojklikem na element v XML souboru dané aktivity, ve které se prvek nachází. U pozice rozhoduje pořadí umístění prvku v rámci dané obrazovky a používá se v případě, kdy není ID definované. Uvedený příklad znamená, že se prvek nachází v rámci aktivity jako 2. tlačítko v pořadí. A nakonec u lokace pomocí textu se zadává text který je definován jako atribut daného elementu.

[2]Robotium - domovská stránka: http://code.google.com/p/robotium/

Spuštění testovacích scénářů

```
$ adb shell am instrument -w <Class>/android.test.InstrumentationTestRunner
```

-w ≪***test-name***≫

Název třídy testovacího scénáře.

5.3 UI Automator

Framework UI Automator[3] byl nedávno představen společností Google a pro testování mobilních aplikací je dostupný od Jellybean verze Androidu[4]. Je zde opět možné psát testy v programovacím jazyce Java a testovat v emulátoru či na reálném zařízení.

Dostupné třídy

UI Automator poskytuje pět různých tříd pro implementaci automatizovaných testovacích případů, které umožňují zachytit a manipulovat s prvky uživatelského rozhraní Android aplikace. Patří mezi ně:

- *UiDevice* - poskytuje přístup ke statických informacím o zařízení a umožňuje simulovat uživatelské akce prostřednictvím ovládání hardwarových tlačítek
- *UiObject* - reprezentuje UI element, na který můžeme aplikovat různá gesta, načítat z něj atributy a nebo mu nastavovat text
- *UiScrollable* - poskytuje podporu pro hledání objektů ve scrollovacích pohledech, simuluje swipování
- *UiSelector* - slouží k vyhledání daného elementu podle zadaného kritéria
- *UiCollection* - slouží k přístupu k prvkům v rodičovském elementu a k získání jejich počtu

Celý popis UI Automator API je dostupný na stránkách pro vývojáře androidu.[5]

UI Automator viewer

Velká výhoda při používání tohoto frameworku spočívá v dostupnosti nástroje pro rozpoznávání prvků grafického uživatelského rozhraní s názvem UI Automator Viewer[6]. Oba tyto nástroje jsou poskytovány společností Google jako součást Android SDK.

[3] UI Automator: http://developer.android.com/tools/testing-support-library/index.html#UIAutomator

[4] od API úrovně 16, tedy verze OS Android 4.1 a výš

[5] UI Automator API - http://developer.android.com/reference/android/support/test/uiautomator/package-summary.html

[6] UI Automator Viewer: http://developer.android.com/tools/testing-support-library/index.html#uia-viewer

5.4 UI/Application Exerciser Monkey

UI/Application Exerciser Monkey[7] je taktéž nástroj poskytován Android SDK a slouží pro zátěžové testování uživatelského rozhraní. Je typu command-line, tedy nedisponuje vlastním uživatelským rozhraním, běží přímo na zařízení, ať už v reálném nebo emulátoru a funguje tak, že zasílá do aplikace pseudonáhodné proudy systémových a uživatelských událostí (kliknutí, doteky, gesta). Umožňuje také reportovat chyby, které se během testování objeví.

Spuštění

Protože Monkey nástroj běží přímo v prostředí emulovaného nebo reálného zařízení, je nutné jej spouštět pomocí Shellu přímo v tomto zařízení. Pro tento účel využijeme k tomu určený nástroj ADB.

```
$ adb -d shell -p <packege-name> -v <event-count>
```

Pomocí přepínačů `-d` a `-e` můžeme specifikovat provádění testů na reálném zařízení/ v emulátoru. Za přepínač `-p` umísťujeme název balíčku testované apliakce. Úroveň podrobností výpisu lze specifikovat pomocí přepínače `-v`, s každým `-v` na příkazovém řádku roste úroveň podrobnosti výpisů. Poslední parametr udává počet vyslaných pseudo-náhodných událostí do zařízení.

5.5 Apache Ant

Apache Ant[8] je nástroj pro sestavování softwarových aplikací. Umožňuje automatizovat řadu činností od kompilace, testování, až po vytvoření balíku pro distribuci. Princip Antu je shodný s unixovým nástrojem *Make*, avšak na rozdíl od něho se skripty píší v jazyce *XML*. Nástroj samotný je napsaný kompletně v jazyce **Java**. Z toho vyplývá jeho největší výhoda – platformní nezávislost.

Ant je open source projektem organizace *Apache Software Foundation*. Podmínky jeho použití upravuje licence *Apache License 2.0*.[9]

Sestavení

Předpokladem je správné nastavení proměnných prostředí `PATH` i `JAVA_HOME` dle dokumentace a přítomnost konfiguračního souboru build.xml v tomto adresáři. Není-li zadán parametr target, použije se výchozí.

```
$ ant [target]
```

[7] UI/Application Exerciser Monkey: http://developer.android.com/tools/help/monkey.html
[8] Apache Ant - domovská stránka: http://ant.apache.org/
[9] Apache Ant - licence: http://ant.apache.org/license.html

V konfiguračním souboru build.xml jsou pomocí tagu `target` definovány cíle, kterých má být v rámci procesu sestavení dosaženo. Každý cíl má svůj název a kromě toho může mít definován v atributu `depends` také seznam závislostí na dalších cílech, které musí být vykonány před ním. V kořenovém tagu project se v atributu default udává výchozí cíl, jenž bude zpracován v případě, že Antu explicitně neurčíme jiný.

Konfigurační soubor

K popsání sestavovacího procesu Antu pro zvolený projekt je třeba vytvořit konfigurační soubor ve formátu XML, zpravidla nese název *build.xml*. Velké projekty se často skládají z dalších podprojektů a proto Ant umožňuje definovat závislosti na dílčích konfiguračních souborech. Nejprve jsou v definované hierarchii sestaveny všechny podprojekty a až poté hlavní projekt.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="sampleproject" default="archive">
    <property name="version" value="1.0" />

     <target name="init">
         <mkdir dir="src/META-INF" />
         <mkdir dir="build/classes" />
         <mkdir dir="dist/${version}" />
     </target>

     <target name="compile" depends="init">
         <javac srcdir="src" destdir="build/classes" includeantruntime="false"/>
     </target>

     <target name="archive" depends="compile">
         <manifest file="build/manifest.mf">
            <attribute name="Main-Class" value="org.antbook.welcome.Main"/>
         </manifest>
         <jar destfile="dist/${version}/project.jar" basedir="build/classes" manifest="
             build/manifest.mf"/>
     </target>

     <target name="clean" depends="init">
         <delete dir="build"/>
         <delete dir="dist/${version}"/>
     </target>
</project>
```

Obrázek 5.1: Příklad konfiguračního souboru `build.xml` (převzato z [2])

Kapitola 6

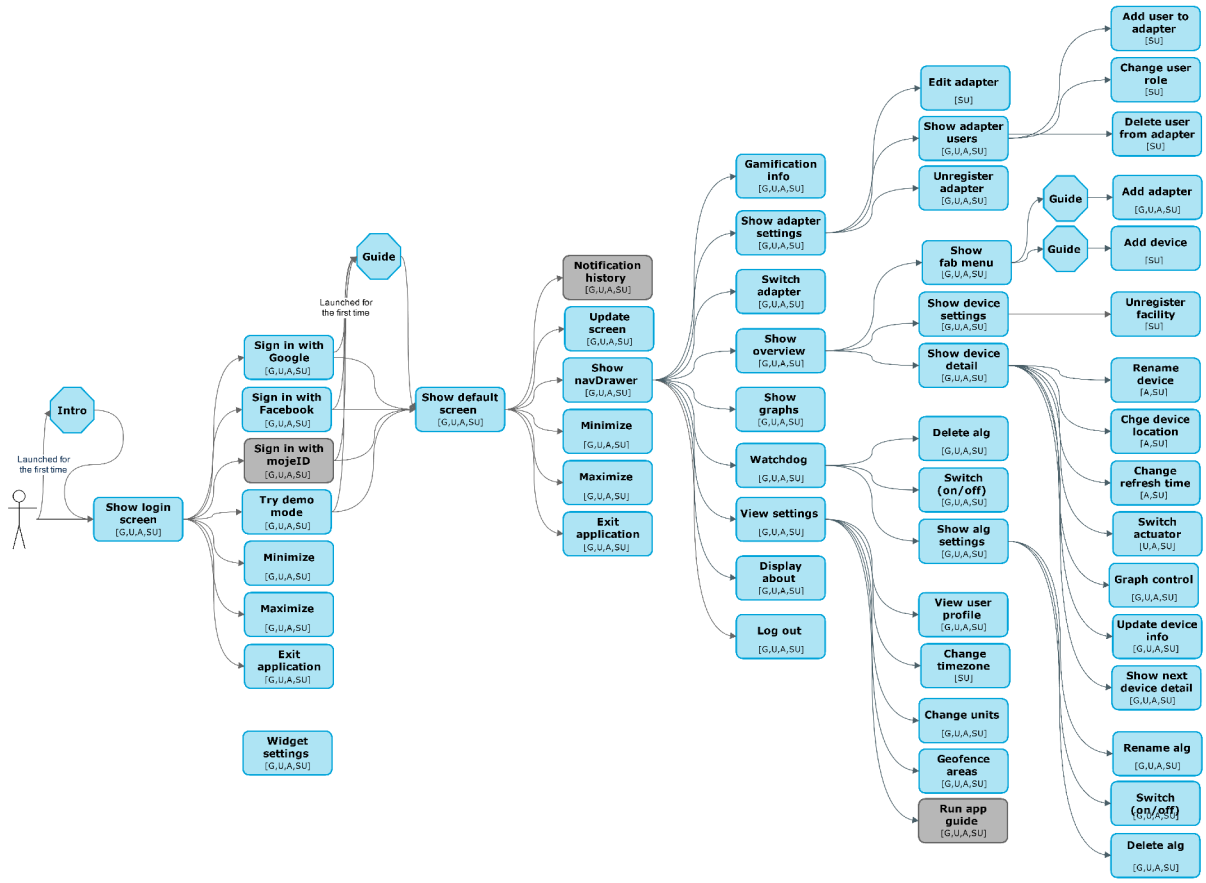
Návrh testování aplikace

Předmětem testování je mobilní aplikace pro platformu Android, která slouží ke vzdálenému ovládání inteligentní domácnosti. Aby bylo možné zaručit spokojenost uživatele při zadávání různých požadavků skrz aplikaci do nižších vrstev systému, je nutné nejdříve ověřit všechny funkce této aplikace. Aby při tvorbě testovacích scénářů nebyla opomenuta nějaká důležitá funkce aplikace, je v první části této kapitoly rozebírána specifikace testované aplikace a způsoby jejího použití, které jsou následně demonstrovány diagramem případů užití.

6.1 Specifikace testované aplikace

Cílem vytvoření této specifikace je seznámení se s testovanou aplikací, zdůraznění nejdůležitějších funkcí, kterými aplikace disponuje, zachycení aktérů, kteří aplikaci budou používat a jejich přístupových práv ve formě uživatelských rolí, ujasnění si míst, kde by se měly objevovat nápovědy, které předvídají akce uživatele a napomáhají mu ke správně konfiguraci domácnosti a případy a místa v aplikaci, kde by se měly tyto nápovědy zobrazovat opakovaně a nakonec odhadnout, kde by mohly být aplikace náchylná k chybám a následnému pádu.

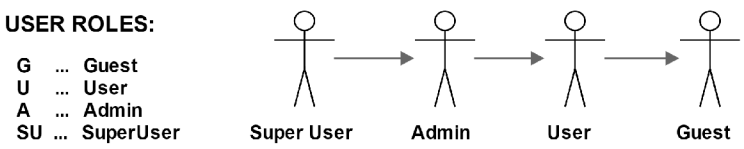
Diagram na Obrázku 6.1 demonstruje všechny funkce aplikace, kterými momentálně disponuje, tedy všechny akce, které je uživatel schopen v aplikaci provést.



Obrázek 6.1: Diagram případů užití testované aplikace

Forma diagramu případů užití je uzpůsobena tomu, aby se z něj lehce a přehledně daly tvořit testovací případy, kdy postupně krok po kroku ověřujeme, zda se provedla určitá akce a přinesla odpovídající výsledky, zda se objevila zpráva pro uživatele, chybová hláška a nebo prvek grafiko-uživatelského rozhraní. Jednotlivé kroky konkrétního testovacího scénáře pak představují průchod tímto diagramem, kdy ve většině případů odpovídá jeden případ užití jednomu testovacímu případu. Přítomnost případů užití s názvem obrazovek aplikace je zejména z toho důvodu, aby bylo jasné, v jakém momentě se má jaká nápověda objevit.

Jak je znázorněno na Obrázku 6.2, v diagramu figurují celkem čtyři různí aktéři, kteří reprezentují uživatelské role aplikace. Od té z nejnižšími právy jsou to Guest, User, Admin a SuperUser. V diagramu jsou pro přehlednost znázorněny v každém případu užití v hranatých závorkách.



Obrázek 6.2: Aktéři reprezentující uživatelské role aplikace

Jelikož je testovaná aplikace stále ve vývoji, nacházejí se zde i některé případy užití, které v současné době nejsou implementované a tím pádem nebudou ani předmětem testování. V diagramu jsou znázorněny šedou barvou.

Obrázek 6.3: Rozeznání ne/implementovaných částí

6.2 Shrnutí požadavků na testování

Následuje výčet několika hlavních funkcí aplikace, které by měly být hlavním podnětem testování a na základě kterých budou vybrány vhodné metody a nástroje pro testování. Za klíčové funkce aplikace, které bude potřeba primárně otestovat, budou považovány zejména:

- přihlašování se do aplikace
- registrace brány
- přidání zařízení do domácnosti
- nastavování uživatelských rolí

Požadavky na testování budou dále zaměřeny na:

- rychlé opakování akcí a random testování
- změna orientace obrazovky (kontrola správného překreslení obrazovky)
- závislost na internetovém připojení
- minimalizace (kontrola zachování stavu aplikace)
- přerušení běhu aplikace (tlačítkem zpět, příchozím hovorem)
- testovaní dialogu
- dynamické změny konfigurace (změna času či jazyka)

Testování těchto funkcí a požadavků bude podrobnější než ostatních částí systému.

6.3 Výběr vhodné testovací metody a pokrytí

U mobilních aplikací je důležité především funkční testování, protože se při něm testuje přímo to, co uživatel vidí, a proto. Na základě specifikace testované aplikace a požadavků na testování bylo jako hlavní kritérium pro tvorbu testovacích scénářů zvoleno testování na základě požadavků, kdy budou ověřeny všechny funkce, kterými aplikace disponuje.

"100% pokrytí požadavků znamená cca 70% kódu." (line/node coverage). [10]

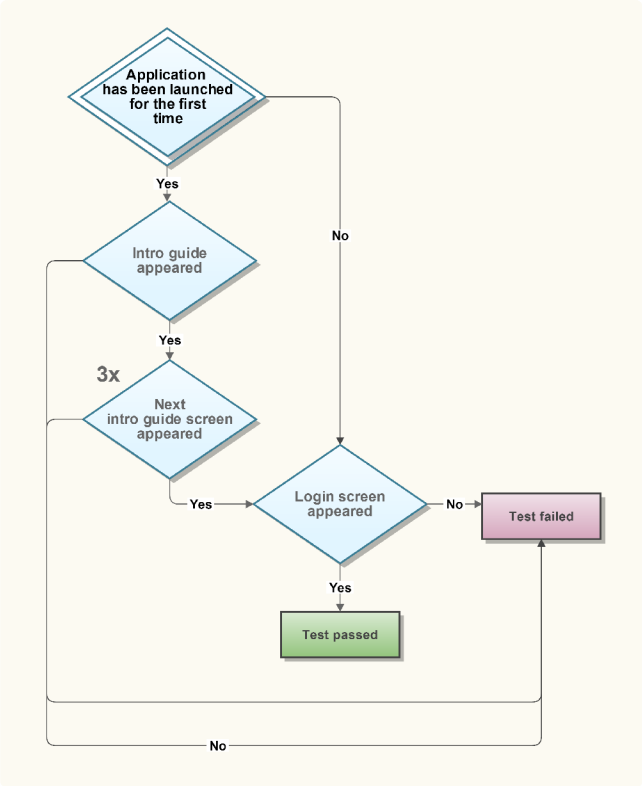
V kap. 4.2 jsme došli k výsledku, že je velmi výhodné testování co nejvíce zautomatizovat a vzhledem k tomu, že je testovaná aplikace stále ve vývoji, tak i uzpůsobit možnosti regresního testování.

6.4 Příprava testovacích scénářů

Při funkčním testování je nejlepší rozdělit aplikaci na uživatelské případy a každý testovat zvlášť. Automatizované testovací případy by měly být schopny, na rozdíl od manuálních, samy rozpoznat, zda uspěly či selhaly. Pro teto fakt byly u složitějších testovacích případů vytvořeny flow diagramy, ze kterých bylo jednodušší testovací případy implementovat a ze kterých jde jednoznačně určit, jaké akce nastaly po jaké události.

Testovací případ úvodní nápovědy

Úvodní nápověda je charakteristická tím, že se objevuje pouze po prvním spuštění aplikace a slouží k prvotnímu představení celého sytému. Testovací scénář k této nápovědě ověřuje, zda se napoprvé opravdu objeví, nezpůsobí pád aplikace a zda je správně předáno řízení další aktivitě, která se stará o přihlašování. Také je ověřováno, že při dalším spuštění aplikace už již tato úvodní nápověda nenaběhne.



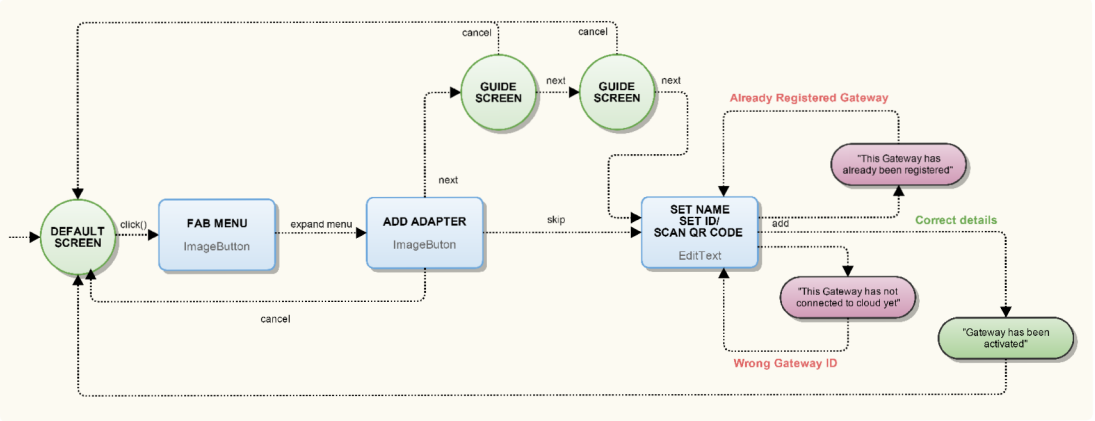
Obrázek 6.4: Diagram k testovacímu případu spuštění úvodní nápovědy

Testovací případ registrování brány

Z výchozí obrazovky lze pomocí menu zvaného `fab`[1] přidat bránu. Následuje nápověda k přidání brány, kterou lze buď celou projít, přeskočit a nebo zrušit. Následuje obrazovka pro

[1]Podle knihovny z níž je menu tvořeno

specifikaci ID a jména brány. Pokud jsou tyto údaje zadány správně, dojde k ověření hlášky oznamující, že brána byla aktivována a zkontroluje se, zda se brána nachází v seznamu domácností. V případě, že byly zadány údaje již registrované brány, dojde k ověření hlášky vztahující se k této záležitosti a ověření, že brána nebyla přidána do seznamu domácností. Podobný scénář je v případě již neregistrované brány v systému.



Obrázek 6.5: Diagram k testovacímu případu registrování brány

Kapitola 7

Implementace testů pro regresní testování

Regresní testování slouží ke znovu spouštění testovacích případů, které nezpůsobily selhání při posledním spuštění, abychom se ujistili, že neselže ani při ověřování dalších implementovaných části v nové verzi aplikace. V následující kapitole budou nejdříve popsány implementované testovací sady sloužící pro regresní testování aplikace, dále bude popsán proces automatizace překladu a spouštění těchto testovacích scénářů a nakonec bude uvedeno, jaké získáme vyhodnocení po skončení provádění testu.

7.1 Testovací sady

Podle návrhu bylo implementováno několik testovacích sad.

Regresní testy reálného a demo módu

Je o sady kompletních testů na všechny funkcionality systému, které ověřují, zda lze vykonat všechny funkce aplikace podle její specifikace a zda funkce mají odpovídající výsledky. Spouští se zpravidla po vydání každé nové verze aplikace za účelem ověření, že nová nebo změněná implementace nezpůsobila chyby v ostatních částech aplikace

Vzhledem k tomu, ze lze aplikaci používat ve dvou různých módech, a to buď v reálném nebo demo módu, byla pro tento fakt přizpůsobena i tvorba hlavních testovacích sad. Jedná se o sady, které budou sloužit především pro regresní testovaní, tudíž budou ověřovat kompletně správnost všech funkcí, kterými aplikace disponuje.

Pro tyto sady byla zvlášť pro každou vytvořena testovací třída, s názvy `TestDemoMode` a `TestRealMode`, obsahující kompletní sadu na sebe navazujících testovacích případů.

Testy práv uživatelských rolí

Jak již bylo zmíněno ve specifikaci v kap. 6.1, aplikace disponuje čtyřmi uživatelskými rolemi, které mají různá práva. Testy pro ověření práv uživatelských rolí byly implementovány ve třídě `TestUserRoles`.

Testy vytvořené na základě typických chyb v aplikaci

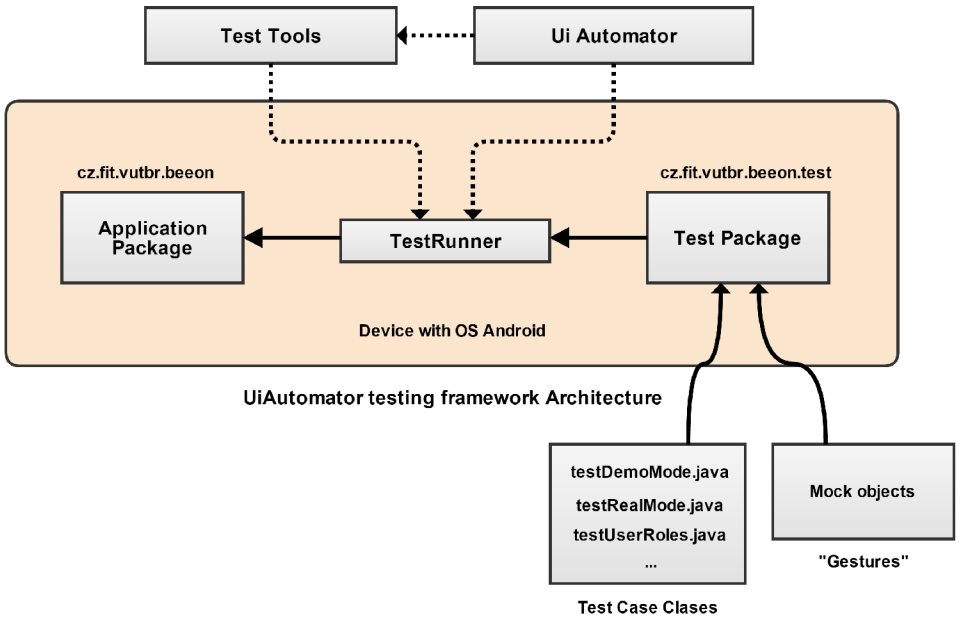
Vývojáři své chyby často opakují, a proto se občas stane, že se ta samá chyba objeví v nějaké další verzi aplikace znovu. Pro tyto případy je nutné se zajistit, aby v takovém případě byla podruhé chyba hned zachycena a následně opravena.

Na základě průzkumu typických chyb v android aplikacích a dříve reportovaných chyb testované aplikace byl vytvořen seznam nejdůležitějších oblastí, ve kterých se nejčastěji vyskytují chyby a seznam akcí, po kterých dochází k pádu aplikace.

Tyto testy, které byly vytvořeny na základě výskytu chyby v předchozí verzi, budou sloužit pro opakované spouštění a ověření, zda byla příčina selhání správně opravena. Jednotlivé testovací případy jsou zahrnuté ve třídě `TestFailures`.

7.2 Využití frameworku UiAutomator pro psaní testovacích scénářů

Testovací scénáře byly psány za pomocí frameworku pro automatizované testování UiAutomator.



Obrázek 7.1: Architektura testovacího frameworku

Application package - balík aplikace, která bude testována

TestRunner - spouší testovací případy, zahrnuje

- *Test Tools* - nástroje Android SDK pro sestavování testů
- *UI Automator* - knihovna nástroje UI Automator, která poskytuje API pro psaní programu, pomocí kterého lze ovládat zařízení z vnějšího prostředí Android kódu

Test Package - testovací projekt, obsahuje

- *Test Case Clases* - testovací třídy s testovacími případy, které mají být spuštěny v zařízení
- *Mock objects* - falešné objekty, které budou použity testovacími případy

Využití tříd UiAutomator API

Všechny implementované testovací třídy dědí od třídy *UiAutomatorTestCase*, která umožňuje zasílat klávesové a klikací události do zařízení a ověřovat výsledky událostí pomocí `assert` metod. Pro přístup a provádění operací na zařízení s testovanou aplikaci je využívána třída *UiDevice*, pomocí které získáváme screenshot obrazovky a simulujeme stisk hardwarových tlačítek. K zachycení a manipulaci s komponentami uživatelského rozhraní bylo využito tříd *UiObject*, *UiScrollable* a *UiSelector* popisovaných podrobněji v kap. 5.3.

Specifikace selektoru UI komponenty

Pro přístup ke konkrétní UI komponentě v aplikaci je využita třída *UiSelector*. Pro specifikaci komponenty je třeba znát konkrétní knihovnu UI objektů Androidu. Za pomocí nástroje UI Automator Viewer a nebo ze zdrojového kódu testované aplikace zjistíme, o jaký interaktivní prvek UI se jedná a jaké má atributy.

```
UiObject nextButton = new UiObject(new UiSelector()
        .className(android.widget.Button.class.getName())
        .resourceId("cz.vutbr.fit.beeeon:id/add_adapter_next")
        .text("NEXT")
        .index(1));
```

V případě komponenty `nextButton` definované v testované aplikaci ve formě tlačítka lze zjistit, že se jedná o třídu *Button* z balíku `android.widget`, je umístěna na indexu `1`, její atribut ID nese název `cz.vutbr.fit.beeeon:id/add_adapter_next` a je specifikována textem `NEXT`.

Vzhledem k faktu, že je testovaná aplikace stále ve vývoji, je výhodné specifikovat komponentu podle ID, protože se tento atribut nejméně často mění. Pokud není ID k dispozici, použijeme specifikaci pomocí testu a až poté popřípadě pomocí indexu.

Implementace hlídačů

Pro implementaci hlídačů bylo použito rozhraní *UiWatcher*. Hlídače běží na pozadí testu a hlídají výskyt specifikované komponenty. Podstatou implementovaného hlídače `anrWatcher` je hlídání komponenty oznamující pád testované aplikace. V případě, že nastane pád aplikace, ukončí se provádění testu, zaznamená se ze zařízení logcat a další soubory užitečné pro reportování chyby, uživateli je oznámeno, že došlo k pádu testované aplikace a na standardním výstupu je odkázán na adresář s výsledky testu. Další implementovaný hlídač `guideWatcher` slouží pro kontrolu správného výskytu nápověd k aplikaci.

Ověřování výsledků

Ověřování výsledků událostí je implementováno především pomocí metod `assertTrue()` a `assertFalse()`. Pokud dojde k selhání jednoho testovacího případu, pokračuje se v prová-

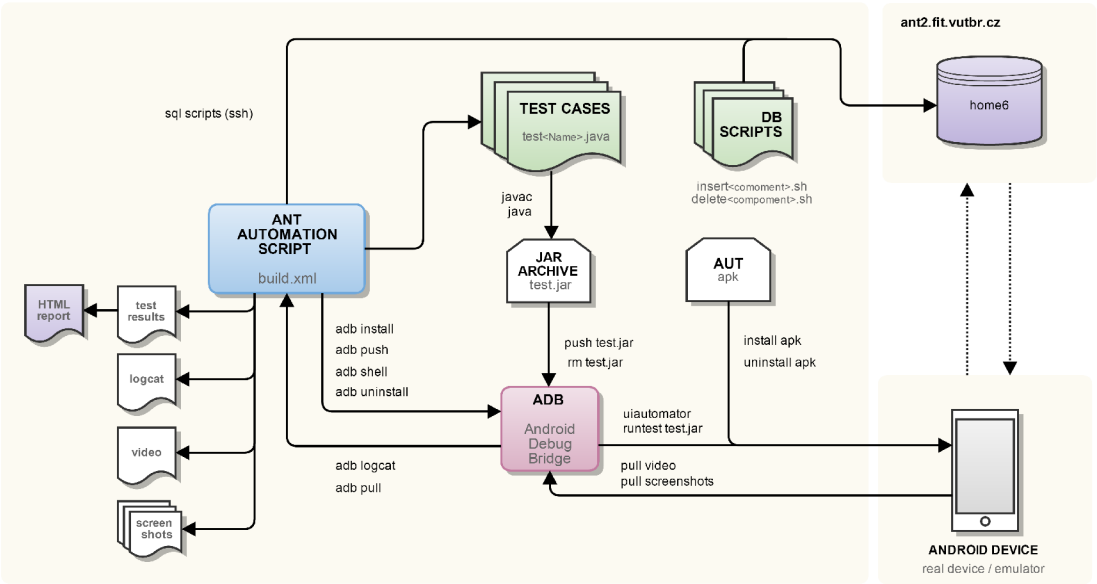
dění testu dalšími testovacími případy a selhání i s oznamovací hláškou je zaznamenáno do logovacího souboru provádění testu a objeví se i ve výsledném HTML reportu.

Implementace dalších metod

`trace();` - vkládání informací o průběhu testu do logu
`setTestCaseName();` - přiřazení jména testu k testovacímu případu
`takeScreenShot(String filename):` - vytvoření screenshotu s časovým razítkem

7.3 Automatizace testování

K testovaní aplikace nestačí pouhé spouštění testovacích scénářů, ale je nutné před jejich spuštěním nachystat všechny potřebné soubory do zařízení a po skončení testů získat report o výsledku testování, lokalizovat případnou chybu a uvést stav zařízení do původního stavu. K tomu bylo využito nástroje pro sestavovaní aplikací a automatické testovaní - Apache Ant, popsaného více v kap. 5.5. Pomoci tohoto nastroje je řízen celý proces testování.



Obrázek 7.2: Schéma procesu automatizace

Funkce skriptu:

- překlad testovacích scénářů do spustitelného jar archivu
- vkládání/mazání dat do/z databáze
- od/instalace testované aplikace ze/do zařízení
- vložení/smazání spustitelného souboru s testy do/ze zařízení
- spuštění/zastavení nahrávání videa
- spuštění testů
- získání záznamu/screenshotů testování
- získání logcatu zařízení

Konfigurační soubor build.xml

Pro správné fungovaní nástroje Ant je nutné správně nakonfigurovat soubor build.xml, kde jsou definovány `targets` *(cíle)*, pomoci kterých jsou řízeny jednotlivé kroky testování.

Definované cíle:

`help`	zobrazení nápovědy k použití skriptu
`clean`	smazání cílových adresářů pro překlad
`mkdir`	vytvoření adresářů pro překlad
`compile`	překlad testovacích scénářů
`jar`	vytvoření spustitelného jar archivu
`push`	vložení spustitelného souboru s testovacími scénáři do zařízení
`uninstall`	odinstalace testované aplikace
`install`	instalace testované aplikace
`db-insert`	vložení testovacích dat do databáze
`screenshotdir`	vytvoření adresáře pro screenshoty v zařízení
`testDemoMode`	spuštění testovací sady pro demo mód aplikace
`testRealMode`	spuštění testovací sady pro reálný mód aplikace
`pull`	stažení souborů ze zařízení
`jar-clean`	úklid v zařízení
`db-delete`	vymazání testovacích dat z databáze
`report`	vytvoření reportu
`finish`	konec skriptu, informace uživatele o umístění výsledků testu

Spuštění testování

Spuštění testování pak provedeme na základě testovací sady, kterou chceme použít. Název testovací sady použijeme jako název cíle.

Testování demo módu aplikace spustíme příkazem:

```
$ ant testDemoMode
```

Po zadání příkazu se provedou cíle: `clean`, `mkdir`, `compile`, `jar`, `push`, `uninstall`, `install`, `testDemoMode`, `jar-clean`, `report` a `finish`.

Pro testování reálného módu aplikace použijeme příkaz:

```
$ ant testRealMode
```

V tomto případě se provedou cíle: `clean`, `mkdir`, `compile`, `jar`, `push`, `uninstall`, `install`, `db-insert`, `testRealMode`, `pull`, `jar-clean`, `db-delete`, `report` a `finish`.

Skript je implementován tak, aby zanechal komponenty, se kterými pracuje, zejména stav databáze a mobilního zařízení, přesně v takovém stavu, v jakým se nacházely přes jeho spuštěním. Po skončení testu se skript postará i o úklid nepotřebných souborů v adresáři projektu.

Výstup skriptu

Obsah adresáře report:

```
/report/<test-suite>/html/index.html ... HTML report
              <test-suite>.<time-stamp>.txt ... output from testing
              <test-suite>.<time-stamp>.log ... device logcat
              <screeenshotName>.<timestamp>.png ... screenshot
```

«**screeenshotName**»

Název screenshotu, který byl vytvořen během provádění testu.

«**test-suite**»

Název testovací sady, která byla provedena.

«**time-stamp**»

Časové razítko ve formátu `yyyy-MM-dd_HHmmss`.

Report o testovaní

K získání reportu o testování je použita knihovna `uiautomator2junit-0.3.jar`, která z textového souboru, který obsahuje výstup z testováni pomocí frameworku *IU Automator*, vytvoří XML soubor typu *jUnit* a posléze z tohoto souboru čitelnější *HTML soubor*.

V případě neúspěšně provedeného testu se dozvíme, při kterém testovacím případu i konkrétně u jaké komponenty test selhal. Pokud dojde k pádu aplikace, můžeme dohledat příčinu pádu v logcatu stáhnutého ze zařízení, kde lze nalézt stack trace konkretní chyby.

7.3.1 Simulace zbytku systému prostřednictvím databáze

Pro správný průběh testovacích scénářů reálného módu aplikace je využíváno databázového rozhraní pro simulaci zbytku systému. Pro provádění testů pro demo mód aplikace tohle není nutné, vzhledem k tomu, že jsou zde napevno dané přednastavené hodnoty a nedochází k připojování k serveru.

Byly implementované jednoduché skripty obsahující SQL příkazy pro přidání prvků inteligentní domácnosti do databáze, které jsou volány před spuštěním testu a skripty pro vymazání těchto prvků z databáze následně po skončení testu.

Příklad použití skriptů:

```
$ sh insertGateway.sh <gateway-ID> <gateway-name>
$ sh insertFacility.sh <facility-mac> <gateway-ID>
$ sh insertDevice.sh <facility-mac> <device-name> <device-type> <value>
$ sh deleteDevice.sh <facility-mac> <device-type>
$ sh deleteFacity.sh <facility-mac>
$ sh deleteGateway.sh <gateway-ID>
```

≪**gateway-ID**≫

Jednoznačná identifikace brány.

≪**gateway-name**≫

Název brány.

≪**facility-mac**≫

Mac adresa zařízení.

≪**device-name**≫

Název zařízení - senzoru nebo aktuátoru.

≪**device-type**≫

Typ zařízení - např. pro teplotní senzor je typ roven 10.

≪**value**≫

Hodnota zařízení - v případě teplotního senzoru hodnota teploty.

7.3.2 Komunikace se zařízením přes ADB

Jak už bylo popsáno v kap 3.2, ADB (Android Debug Bridge) je ovladač, který slouží pro pokročilý vývoj a testování mobilních aplikací a umožňuje spouštět Linuxové příkazy přímo v zařízení připojené k PC. V rámci této práce jej využijeme k instalování aplikace do zařízení, kopírování souborů ze zařízení do pc a naopak a ke spouštění testovacích scénářů.

Příkazy směrem k zařízení využívané pro správný průběh testu:

Vytvoření adresáře pro	[1]
Nainstalování testované aplikace	[2]
Odinstalování testované aplikace	[3]
Vložení spustitelného souboru s testy	[4]
Spuštění testu	[5]
Získání logcatu	[6]
Stažení screenshotu	[7]
Úklid po testu - smazání adresáře se screenshoty	[8]
Úklid po testu - smazání jar archivu s testy	[9]

```
1  $ adb shell 'mkdir /data/local/tmp/screenshots'
2  $ adb install <app-apk>
3  $ adb uninstall <app-package>
```

```
4    $ adb push bin/test.jar /data/local/tmp
5    $ adb shell uiautomator runtest test.jar -c <testClassName>
6    $ adb logcat -d > <report-dir>
7    $ adb pull /data/loca/tmp/screenshots <report-dir>
8    $ adb shell 'rm -r /data/loca/tmp/screenshots'
9    $ adb shell 'rm /data/local/tmp/test.jar'
```

«**app-apk**»

Název apk souboru testované aplikace.

«**app-package**»

Název balíčku testované aplikace.

«**testClassName**»

Název třídy testovacího scénáře, který má být v zařízení spuštěn.

«**reportDir**»

Adresář sloužící k ukládání výsledků testů ve formě logcatu, videa nebo screencshotu.

7.4 Konfigurace stress testů

Tester by se vždy neměl držet pouze testovacího scénáře, ale měl by vyzkoušet i jiné způsoby. Jelikož bude aplikaci ovládat přímo konečný uživatel, je vhodné zapřemýšlet, jak bude běžný uživatel postupovat a pokusit se dosáhnout stejného výsledku i během testování.

Představme si scénář, kdy uživatel přijde poprvé k aplikaci a protože chce zjistit, co všechno aplikace umí, začne klikat na všechny možné ovládací prvky, které mu jsou k dispozici, i mimo ně. S tímto scénářem ovšem cílené testování založené na specifických testovacích scénářích nepočítá a tak může lehce dojít k pádu aplikace.

Vzhledem k tomuto nepředvídatelnému lidskému faktoru byly do testování zařazeny i stress testy založené na náhodných akcích.

Provádění Monkey testů

Stress testy jsou prováděny pomocí nástroje Ui Exerciser Monkey popisováného v kap. 5.4. Testy jsou kompatibilní pouze se zařízeními s OS Android 5.0+ a to zejména z toho důvodu, že Monkey test otvírá status bar telefonu a zasahuje skrz něj do nastavení telefonu. Ve verzích OS Android 5.0+ lze tento problém vyřešit novou funkcí připnutí aplikace, a tudíž Monkey test nemůže zasahovat do jiných aplikací ani otvírat status bar.

Vytvoření skriptů pro spouštění Monkey testů

Funkce:

- od/instalace testované aplikace ze/do zařízení
- vložení spustitelného souboru pro nahrávání videa do zařízení
- spuštění/zastavení nahrávání videa

- spuštění stress testů
- získání videozáznamu testování ze zařízení
- získání logcatu zařízení

Spuštění:

```
$ sh monkeyTest.sh [seed]
```

Parametr seed

Monkey nástroj při každém provedení testu poskytne informaci o hodnotě parametru `seed`, která je vyjádřena cca deseti až dvacetimístným celým číslem a jednoznačně identifikuje posloupnost událostí, které byly během testu do aplikace vyslány. Slouží tedy pro případnou reprodukci chyby, kdy je nutné tuto hodnotu zadat jako jeden z parametrů Monkey testu. O zařazení parametru `seed` na správné místo se také postará vytvořený skript. Stačí tuto hodnotu uvést jako jeho volitelný parametr.

Záznam videa

Nahrávání videa slouží pro záznam nereprodukovatelných chyb a pro zaznamenání jednotlivých kroků, které byly provedeny před pádem aplikace, a tedy k tomuto pádu vedou. Přímo pro nahrávání videa je využíváno aplikace *Screen Recorder*[1], která je k dispozici volně ke stažení na *Google Play*[2]. Ke zautomatizování ovládání aplikace pro nahrávání videa byly vytvořeny jednoduché třídy `RecordStart` a `RecordStop`, sestavené do spustitelného jar souboru a importované do zařízení. Třídy využívají pro ovládání aplikace frameworku *UI Automator*.

Výstup skriptu

Obsah adresáře report:

```
/report/logcat.<time-stamp>.log ............... logcat zarizeni
        screeenrecorder.<timestamp>.mp4 ....... videozaznam provadeni testu
```

≪**time-stamp**≫

Časové razítko ve formátu `yyyyMMddHHmmss`.

[1]Screen Recorder - domovská stránka: http://nllapps.com/apps/scr/
[2]Odkaz ke stažení: https://play.google.com/store/apps/details?id=com.nll.screenrecorder

Kapitola 8

Testování funkčnosti a parametry navrženého řešení

Následující kapitola popisuje použití hardwaru a softwaru v implementační části této práce. Dále se věnuje vyhodnocení úspěšnosti implementovaných testů, jejich pokrytí a způsobu reportování chyb nalezených během vývoje testovacích scénářů.

Využité technologie pro vývoj testů

Pro vývoj mobilních aplikací pro operační systém Android se využívá programovací jazyk Java, a proto bylo vhodné použít tento jazyk i pro implementaci testů, jelikož je mu uzpůsobena většina nástrojů pro vývoj a ladění aplikací. Mimo standardní knihovny jazyka Java byla využita knihovna *Android Testing Support Library*[1] poskytující prostředí pro testování mobilních aplikací s operačním systémem Android. Pod touto knihovnou se nachází i *UI Automator API*, kterého bylo využito pro jednodušší a efektivnější vytváření testovacích scénářů. Pro implementaci testovacích scénářů bylo využito vývojového prostředí *Eclipse*[2] určené pro programování v jazyce Java. Pro komunikaci se zařízením s OS Android připojeného přes USB rozhraní k pc nebo s emulátorem, na kterém byla aplikace testována, bylo využito nástroje Android SDK *ADB (Android Debug Bridge)*[3]. Pro vytvoření grafického virtuálního prostředí, které simuluje fyzický mobilní telefon bylo využito nástroje AVD (Android Virtual Device)[4]. Pro analýzu a zpracování výsledků testů ukládaných do textového souboru bylo využito knihovny *uiautomator2junit-0.3.jar*[5], která nejprve převede textový soubor do XML formátu jUnit reportu a následně pak do čitelnějšího HTML souboru.

Použitý hardware pro testování

Testování funkčnosti navrženého řešení, tedy spouštění testů za účelem ověření jejich správného chování, bylo prováděno primárně na fyzickém mobilním zařízení Nexus 5 od společnosti LG. Zařízení disponuje momentálně nejnovější verzí operačního systému Android 5.1. V zařízení byly aktivovány vývojářské možnosti, avšak root telefonu proveden nebyl.

[1] Knihovna: http://developer.android.com/reference/android/support/test/package-summary.html
[2] Eclipse - domovská stránka: http://eclipse.org/
[3] ADB: http://developer.android.com/tools/help/adb.html
[4] AVD Manager: http://developer.android.com/tools/help/avd-manager.html
[5] Knihovna: http://github.com/dpreussler/automator-log-converter/tree/master/snapshots

Test Coverage - Pokrytí požadavků testy

Problém pokrytí testů založených na požadavcích spočívá v tom, že kombinací jednotlivých událostí vznikne nespočet podobných testovacích případů. Existuje tedy celá řada způsobů, jak aplikací projít, doslova se jí proklikat. Proto bylo jako primární pravidlo pro pokrytí všech událostí, které se dají v aplikací provést zvoleno *Event Coverage*, který vyžaduje, aby požadavky na testy obsahovaly vykonání všech událostí minimálně jednou. Je to proto, že aplikace obsahuje z velké části pouze elementární události, jako je zobrazení další aktivity po kliknutí na specifické tlačítko, a proto není nutné tyto události více mezi sebou více kombinovat a testovat, protože je pravděpodobné, že nepovedou k pádu aplikace. Testy tedy zaručují, že při jejich provádění ověří všechny případy užití definované v diagramu 6.1 minimálně jednou.

Naopak byl větší důraz kladen na testování založené na průzkumu typických chyb v aplikaci a na nejdůležitější části aplikace, jako je registrace brány, zařízení a nebo ověřování správné implementace práv uživatelských rolí. Co se týče aktivit vyskytujících se v aplikaci, byl zde kladen větší důraz na pokrytí událostí vykonaných v rámci jedné aktivity než tomu bylo u případů užití. Zejména u nápověd pro přidání brány a zařízení se vyskytuje mnoho způsobů, jakými se dá z dané aktivity přepnout na jiné místo. Nápovědu lze buď celou projít nebo ji přeskočit, ale je také možné ji ukončit a tím ukončit i celou událost přidávání komponenty do domácnosti.

Pro pokrytí nepředvídatelného lidského faktoru zasahujícího do aplikace byly nakonfigurovány Monkey testy.

Statistiky navrženého řešení

Počet nalezených chyb během implementace testů: 23
Počet nalezených chyb během ověřování funkčnosti testů: 31

Počet zaslaných událostí během testu:

TestDemoMode:	393
TestRealMode:	537
StressTest:	10000

Průměrná doba běhu regresních testů:

Spuštění	TestDemoMode	TestRealMode
1	218.508 s	301.554 s
2	226.676 s	280.734 s
3	208.824 s	298.540 s
4	208.994 s	296.425 s
5	218.586 s	296.130 s
6	218.328 s	301.124 s
Průměr	**216.653 s**	**295.751 s**

Srovnání: scénáře pro regresní testování vyšlou za 1 vteřinu přibližně 2 události (uživatelská gesta), za to stress testy vysílají okolo 88 událostí za vteřinu[6].

[6]Stress test trvá přibližně 114s

Kapitola 9

Závěr

Inteligentní domácnost představuje rozsáhlý systém, avšak ovládání jejich komponent provádí konečný uživatel právě prostřednictvím mobilní aplikace, ve které jsou také veškerá data o domácnosti zobrazována. Objeví-li se tedy problém v nějaké části systému inteligentní domácnosti, projeví se to pravděpodobně i v koncové aplikaci. Proto se jejím testováním částečně ověřuje i funkčnost celého systému jako celku. Na základě nastudování potřebných informací o testované aplikaci a současných přístupech k testování byl navržen adekvátní způsob testování aplikace. Výsledkem této bakalářské práce je sada testovacích scénářů, které slouží ke kompletnímu ověření všech funkcí aplikace na základě definovaných požadavků a navržený způsob jejich zautomatizovaného provádění za účelem zvýšení stability testované aplikace a úspory času při jejím dalším testování. Navržený způsob testování má tedy ověřovat, že aplikace správně reaguje na požadavky od konečného uživatele a zobrazuje mu o domácnosti spolehlivá data.

Během implementace testovacích scénářů se podařilo odhalit v aplikaci řadu chyb, které byly následně reportovány do systému Redmine[1]. Z výsledků testů 8 lze potvrdit splnění účelu a funkčnosti implementovaných testovacích scénářů. Využití implementovaných testů bylo také realizováno ze strany vývojářů mobilní aplikace v případech, kdy si pomocí připraveného testovacího scénáře, vytvořeného na základě výskytu nějaké chyby v aplikaci, chtěli danou chybu reprodukovat a nebo si ověřit její řádné opravení. Ve výsledku lze tedy usuzovat, že navržený způsob testování aplikace šetří čas při testování a zlepšuje stabilitu aplikace. Jediná nevýhoda navrženého způsobu testování spočívá v křehkosti testů, které jsou založeny na uživatelském rozhraní aplikace, a tudíž je při rozsáhlejší změně v implementaci uživatelského rozhraní testované aplikace nutné testovací scénáře novému stylu UI přizpůsobit.

Provádění testů je možné vykonávat na jakémkoliv PC s nainstalovanými prerekvizitami zmiňovanými v *Návodu na přípravu prostředí pro spouštění automatizovaných testů* v příloze B s připojeným fyzickým nebo emulovaným zařízením s operačním systémem Android. Prostředí emulátoru má ale oproti skutečnému mobilnímu zařízení velkou nevýhodu. Simulovat takové prostředí v PC je výpočetně i paměťově náročné, a proto by bylo vhodné mít toto prostředí i s dostupnými testovacími scénáři nakonfigurované na nějakém výkonném serveru, ke kterému by měli přístup všichni členové projektu, a tak by měli možnost si kdykoliv ověřit, že nově implementované řešení negativně neovlivňuje již implementované a otestované části. Další užitečnou věcí k doimplementování by mohlo být zakomponování ce-

[1]Redmine - domovská stránka: http://www.redmine.org/

lého procesu testování do serverově založeného integračního open-source nástroje Jenkins[2], který slouží pro spouštění opakovaných pracovních procesů a který je již využíván pro tuto automatizaci v ostatních částech sytému inteligentní domácnosti. Bylo by postaráno o pravidelnou aktualizaci knihoven a nástrojů Android SDK, překladu testované aplikace a získání jejího aktuálního spustitelného souboru, spuštění testování po vydání nové verze aplikace, reportování výsledků a v případě selhání testu zasílání emailu zainteresovaným osobám. Další možné nástiny rozšíření spočívají v rozšíření testů na další podporované platformy mobilních zařízení aplikací.

[2]Jenkins - domovská stránka: http://jenkins-ci.org/

„Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often. What you eat before you step onto the scale determines how much you will weigh, and the software development techniques you use determine how many errors testing will find. If you want to lose weight, don't buy a new scale; change your diet. If you want to improve your software, don't test more; develop better.“

Jim McCarthy

Literatura

[1] Android - knihovna grafických graficko-uživatelskýh prvků. `http://developer.android.com/reference/android/widget/package-summary.html`.

[2] Wikipedie. `http://cs.wikipedia.org/wiki/Apache_Ant`.

[3] Brychta, T.: *Bezdrátové senzory pro inteligentní domácnost.* FIT VUT v Brně, 2014, bakalářská práce.

[4] Hrushikesh, Z.: *Robotium automated testing for android.* Birmingham: Packt Publishing, 2013, iSBN 978-1-78216-802-7.

[5] Lehr, E.; Rowinski, M.: IBM Sponsored Study Finds Mobile App Developers Not Investing in Security. `http://www-03.ibm.com/press/us/en/pressrelease/46360.wss`, 2015-03-19 [cit. 2015-05-14].

[6] Láska, J.: Android nadále dominuje trhu se smartphony, zabírá 84%. `http://www.mobilmania.cz/bleskovky/android-nadale-dominuje-trhu-se--smartphony-zabira-84-/sc-4-a-1328751`, 2014-11-03 [cit. 2015-05-14].

[7] Milano, D. T.: *Android application testing guide: build intensively tested and bug free Android applications.* U.K.: Packt Pub, 2011, iSBN 978-1-849513-50-0.

[8] Paleta, P.: *Co programátory ve škole neučí, aneb, Softwarové inženýrství v reálné praxi.* Computer Press, 2003, iSBN 80-251-0073-1.

[9] Patton, R.: *Testování softwaru.* Praha: Computer Press, 2002, iSBN 80-7226-636-5.

[10] Smrčka, A.: Přednáška č. 8 Testování na základě požadavků z předmětu Testování a dynamická analýza.

[11] Ujbanyai Miroslav: *Programujeme pro Android.* Academia, 2005, iSBN 978-80-247-3995-3.

[12] Valeš, M.: *Inteligentní dům.* Brno : ERA, 2008, iSBN 978-80-7366-137-3.

Příloha A

Obsah CD

Přiložené CD obsahuje:

- Adresář `src` se zdrojovými soubory testovacích scénářů psaných v programovacím jazyce Java.
- Adresář `lib` obsahující knihovny *android.jar* a *uiautomator.jar* pro překlad testovacích scénářů a knihovna *uiautomator2junit-0.3.jar* pro tvorbu reportu v jUnit formátu.
- Adresář `apk` obsahující spustitelný apk soubor testované aplikace.
- Adresář `DB-scripts` obsahující jednoduché skripty pro ovládání databáze.
- Soubor `build.xml` sloužící k sestavení spustitelného souboru s testy a řízení celého procesu automatizace.
- Soubory `project.properties` a `local.properties` upravující lokální vlastnosti projektu.
- Soubor `README.txt` s návodem k instalaci prostředí pro spuštění testovacích scénářů.
- Soubor `pdf` obsahující technickou zprávu.

Příloha B

Návod na přípravu prostředí pro spouštění automatizovaných testů

Prerekvizity:

- JDK 7.
- Nastavení proměnné prostředí JAVA_HOME.
- Android SDK Tools Revision 21 a vyšší.
- Reálné zařízení nebo emulátor s OS Android s verzí Android API 16 a vyšší[1].
- Apache Ant + knihovna jsch.jar potřebná pro příkaz sshexec.

B.1 Instalace Android SDK a příprava požadovaných platforem

Emulátor je součástí Android SDK, které je možné stáhnout zde[2] z oddílu 'Other download options' jako 'SDK Tools only'. Stažený soubor je nutné rozbalit, umístit kdekoliv v pc a nastavit proměnnou prostředí ANDROID_HOME na tento adresář.

```
$ export ANDROID_HOME="<path-to-sdk>"
$ export PATH=${PATH}:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

Předtím, než bude možné nastavit nějaké virtuální zařízení, je nutné pro něj nejdříve stáhnout odpovídající platformu. V adresáři `<sdk>/tools` spustit příkaz:

```
$ ./android sdk
```

Zaškrtnout:

- Android SDK Platform-tools
- Android X.X.X
- Android SDK Build-tool pro příslušnou platformu[3]

[1] Vyžadováno frameworkem UI Automator.

[2] Android SDK - http://developer.android.com/sdk/index.html.

[3] Verzí Androidu lze vybrat k instalaci více najednou, stejně tak lze tímto způsobem doinstalovat další platformy dodatečně.

- Android Support Repository
- Android Support Library
- Google Play Services
- Google Repository

B.2 Příprava zařízení pro testování

Pro testování je možné využít fyzické zařízení s operačním systémem Android připojené k pc přes rozhraní USB a nebo využít emulátor.

Zprovoznění emulátoru Androidu

Pro nastavení emulátoru v adresáři `<sdk>/tools` spustit příkaz:

```
$ ./android avd
```

Vytvořit nové virtuální zařízení. Vyzkoušet, zda správně funguje pomocí kliknutí na **Start..** nebo pomocí příkazu:

```
$ emulator -avd <avd-name>
```

B.3 Testovaná aplikace

Spustitelný soubor aplikace se nachází v adresáři `apk/` a o její nainstalovaní do zařízení se postará nástroj *Ant* podle připraveného konfiguračního souboru **build.xml**.

B.4 Překlad a spuštění testovacích scénářů

Překlad a spuštění testovacích scénářů je řízeno pomocí nástroje Apache Ant.

Pro správný průběh testu je nutné:

1. upravit cestu k adresáři s nástroji Android SDK v souboru local.properties
2. popřípadě upravit cílovou platformu v souboru project.properies
3. nastavit jazyk telefonu do Angličtiny
4. přidat do zařízení Google účty pro testovací účely[4]

Spuštění testů pro reálný mód aplikace provedeme pomocí příkazu:

```
$ ant testRealMode
```

[4]Více informací je přiloženo v manuálu na CD.

V případě testování demo módu aplikace zadáme příkaz:

```
$ ant testDemoMode
```

V případě potřeby lze volat předdefinované příkazy pro nástroj Ant i samostatně. Zde je výčet implementovaných cílů:

`help`	zobrazení nápovědy k použití skriptu
`clean`	smazání cílových adresářů pro překlad
`mkdir`	vytvoření adresářů pro překlad
`compile`	překlad testovacích scénářů
`jar`	vytvoření spustitelného jar archivu
`push`	vložení spustitelného souboru s testovacími scénáři do zařízení
`uninstall`	odinstalace testované aplikace
`install`	instalace testované aplikace
`db-insert`	vložení testovacích dat do databáze
`screenshotdir`	vytvoření adresáře pro screenshoty v zařízení
`testDemoMode`	spuštění testovací sady pro demo mód aplikace
`testRealMode`	spuštění testovací sady pro reálný mód aplikace
`pull`	stažení souborů ze zařízení
`jar-clean`	úklid v zařízení
`db-delete`	vymazání testovacích dat z databáze
`report`	vytvoření reportu
`report`	konec skriptu, informace uživatele o umístění výsledků testu

Další potřebné informace se objeví na standardním výstupu programu. Zejména informaci o tom, kde máme hledat výsledky testu.

B.5 Spuštění Monkey testů

Prerekvizity:
- Nástroje Android SDK.
- Zařízení s verzí operačního systému Android 5.0 a vyšší[5].

```
$ chmod +x install.sh
$ chmod +x test.sh
```

Nastavené prostředí pro provádění testu:

```
$ sh install.sh
```

Skrip install.sh:
- nainstaluje aplikaci pro nahrávání videa

[5]Důvod ke zvolení verze Android 5.0+ je vysvětlen v kap. 7.4

- nainstaluje BeeeOn aplikaci
- nakopíruje jar archiv pro spuštění a zastavení nahrávání videa

Spuštění testů:

```
$ sh test.sh [seed]
```

Skrip test.sh:
- spustí aplikaci pro nahrávání videa
- zapne nahrávání videa
- provede Monkey test
- po skončení Monkey testu ukončí nahrávání videa
- stáhne video ze zařízení do odpovídající složky v pc
- do stejného adresáře stáhne ze zařízení i logcat