Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Pokročilé polymorfní aplikace na platformě Android

Plzeň 2015 Milan Nikl

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 25. června 2015

Milan Nikl

Abstract

This thesis presents the concept of running a polymorphic application on the Android platform. The polymorphic application represents an alternative to the traditional approach of using many single purpose application on an Android device. Instead the device user can change the design of the application interface by selecting a XML pattern file appropriate to the function he wants to use.

The polymorphic application also enables running simple programmes on the Android device without need to create a standalone application, which makes the development process much faster. This thesis presents two examples of such programmes - the RSS reader and contact information finder. Both of these programmes present a way of extracting data from an internet source and displaying it to the user in a human readable form meant for the mobile device.

Abstrakt

Tato práce představuje koncept použití polymorfní aplikace na platformě Android. Polymorfní aplikace umožňuje uživateli vybrat a spustit takovou funkci aplikace, jakou zrovna vyžaduje, a k ní vybrat odpovídající XML soubor s definicí uživatelského rozhraní.

Polymorfní aplikace rovněž představuje způsob, jak pod operačním systémem Android spustit jednoduché programy bez nutnosti vytváření samostané aplikace, což významně urychluje proces vývoje. Tato práce předkládá dvě ukázky takových programů - RSS čtečku a vyhledávač kontaktních informací. Oba programy představují způsob, jak získat data z internetového zdroje a zobrazit je v podobě vhodné jak pro uživatele, tak samotné zařízení.

Poděkování

Rád bych poděkoval Ing. Ladislavu Pešičkovi za obrovskou trpělivost a cenné rady při vedení této práce.

Obsah

1 Úvod

Cílem této diplomové práce je prozkoumání současných možností vývoje polymorfních aplikací na platformě Android a jejich praktického použití.

Operační systém Android je v současné době nejrozšířenější mobilní platformou. Z původního operačního systému určeného pouze pro mobilní telefony se v současnosti stává systém pohánějící rovněž tablety, televizory, nositelnou elektroniku, ale rozšiřuje se rovněž do automobilů a dalších typů zařízení. Aplikace pro tento systém pak bereme jako programy poskytující přidanou hodnotu daného zařízení.

Klasické aplikace pro OS Android jsou všeobecně vnímány jako programy sloužící k jednomu účelu. A at' už vykonávají jakoukoli funkci, jedná se většinou o specializovaný software, který je svojí funkcí rovněž svázán a omezen. Uživatel mobilního zařízení je pak zpravidla nucen používat až desítky takových jednoúčelových programů.

Oproti tomu představuje polymorfní aplikace řešení, které umožňuje změnu funkcionality programu bez nutnosti instalace dalších aplikací. Jediné, co musí uživatel udělat, je zvolit šablonu, podle které se dynamicky vytvoří funkční program pro daný účel. Tento postup rovněž umožňuje spouštění programového kódu na mobilním zařízení bez nutnosti vyvíjet novou, samostatnou, aplikaci.

Tato práce si neklade za cíl nahradit všechny existující aplikace jedním univerzálním programem, ale popisuje koncept, který je možné využít pro vytvoření jediné aplikace použitelné pro vykonávání různých funkcí.

2 Polymorfní aplikace

Problematikou použití polymorfních aplikací se v minulých letech zabývalo již několik studentských prací, na jejichž výsledky tato diplomová práce částečně navazuje.

Zatímco [Hul12] popisuje možnost dynamicky nahrávat a případně měnit grafické rozhraní aplikace bez toho, aby se zároveň měnila její funkce, [Sta13] se zabývá především možností dynamické změny funkčního kódu aplikace. Využívá k tomu jak soubory knihoven obsahujících různé sady funkcí, ale také možnost spuštění vzdálených procedur prostřednictvím mobilní aplikace.

Tato diplomová práce pak koncept představený ve výše uvedených pracích aktualizuje pomocí technologií používaných v roce 2015. Polymorfní aplikace rozšiřuje zejména o možnosti použití cloudových úložišť a možnost použití dynamičtějších grafických rozhraní, která je možné ovlivňovat i za běhu aplikace, nejen při nahrání příslušného předpisu. Nicméně práce rovněž zachovává některé principy představené mými předchůdci.

V této kapitole (2) jsou popsána především podstatná specifika, omezení a výhody, kterými se polymorfní aplikace odlišuje od běžně používaných programů. Následující kapitola 3 pak nabízí srovnání s několika dalšími současnými technologiemi, které se dají použít k podobným účelům jako polymorfní aplikace.

Kapitola 5 pak popisuje vylepšení z hlediska funkcionality a uživatelského rozhraní, které pokročilé polymorfní aplikace přináší. Zatímco kapitola 7 předkládá návrhy možných použití polymorfní aplikace v praxi.

2.1 Oprávnění aplikací

Každá aplikace běžící na platformě Android je svázána se sadou oprávnění, která může při své funkci využívat. Mezi oprávnění patří například zjišťování stavu připojení, možnost zápisu do paměti zařízení, práce s identitou uživatele a další. Práci s oprávněními popisuje podrobně [Asp15].

U klasické aplikace je výčet jejích jednotlivých oprávnění obsažen v tzv. *Manifestu aplikace*, jednotlivá oprávnění a jejich použití shrnuje [Amp15]. Při

instalaci aplikace pak uživatel musí potvrdit, zda souhlasí s tím, že aplikace bude využívat daných oprávnění při svém běhu.

Polymorfní aplikace mají nevýhodu v tom, že jejich činnost není při instalaci předem známa. Není tak jednoduché určit, jaká oprávnění bude aplikace potřebovat. Nelze automaticky povolit všechna oprávnění, protože by takové plošné povolení mohlo být zdrojem bezpečnostních rizik. Proto je vhodné omezit oprávnění aplikace pouze na nejnutnější nutné funkce. Ale pak nutně narážíme na fakt, že polymorfní aplikace bude omezovat oprávnění v ní běžících aplikací příliš.

Seznam oprávnění polymorfní aplikace vytvořené v rámci této diplomové práce je uvedený v příloze A i s patřičnými vysvětleními, k čemu je daná funkce nutná. Ve stručnosti uvedu jen to, že veškerá oprávnění související přímo s placenými službami (např. Uskutečňování hovorů, odesílání textových zpráv a provádění finančních transakcí).

Bezpečnostní riziko může rovněž představovat i zásah do soukromí uživatele zařízení. Pokud si aplikace žádá taková oprávnění, která očividně nesouvisí s její funkcí, je to nanejvýš podezřelé. Společnost Google[1], tvůrce platformy Android, si je tohoto problému vědoma, a proto v následující verzii systému, v preview verzi označené jako *Android M*, bude systém práce s oprávněními přepracován.

Uživatel bude moci jednotlivá oprávnění potvrdit jak při instalaci aplikace, tak je později povolit, či zakázat dle potřeby v nastavení zařízení, jak uvádí [Gio15]. Z hlediska polymorfních aplikací se jedná o významnou změnu, která nastavení oprávnění aplikace značně zjednodušuje.

2.2 Práce se zdroji

Aplikace vytvořené pro OS Android pracují s několika typy zdrojů (*Resources*), které podrobněji popisuje [Art15]. Nejvýznamnější používané zdroje jsou:

- **Drawable** - Grafické zdroje, např. obrázky nebo definice geometrických tvarů použitých v rozhraní aplikace;

[1] Více o společnosti na: `https://www.google.com/about/company/`

- **Layout** - Rozložení elementů grafického rozhraní, umožňuje staticky definovat vzhled vybrané obrazovky v aplikaci nebo jejích součástí;
- **String** - Textové prvky sloužící jako popisky, návody, upozornění atd.

Při vývoji klasické aplikace pro OS Android je možné zdroje oddělit od samotného kódu aplikace a následně použít relativní reference na daný zdroj v projektu aplikace, jak přibližuje [Aar15]. Kromě toho, že slouží zdroje k vytváření kvalitnějšího návrhu aplikace a zjednodušují práci programátorovi, mají využití zejména při přizpůsobení aplikace aktuálním potřebám zařízení uživatele.

Textové zdroje *(String)* umožňují lokalizaci aplikace do různých jazyků. V samotném kódu aplikace je použita pouze jedna reference, která je společná libovolnému počtu jazykových mutací souboru obsahujícímu samotné texty. Dle potřeby je pak možné mezi jazyky libovolně přepínat za chodu aplikace.

Grafické zdroje *(Drawable)* pak zajišťují uzpůsobení grafického vzhledu aplikace podle parametrů displeje zařízení, na kterém aplikace běží. Obdobně jako u textových zdrojů je použit jeden společný odkaz a při načtení dané obrazovky aplikace se vybere nejvhodnější varianta dle natočení zařízení, poměru stran obrazovky atd.

Při vytváření polymorfní aplikace lze ale pracovat se zdroji pouze v rámci samotné mateřské aplikace. Jednotlivé generované aplikace totiž nemají přehled o tom, jaké zdroje jsou v aplikaci definovány. A i když by, čistě technicky vzato, bylo možné referencovat zdroje definované v polymorfní aplikaci, jednalo by se o zcela nevhodný přístup ke tvorbě aplikace. Princip polymorfismu stojí na faktu, že jednotlivé generované aplikace mají logiku omezenou pouze na jejich funkci a nemohou tedy počítat, nebo dokonce zacházet, s prvky entity vyšší úrovně.

Bohužel není možné soubor zdrojů rozšiřovat za běhu aplikace, neboť provázání aplikace se zdroji probíhá při kompilaci aplikace. Za běhu aplikace je tedy možné vybrat nejvhodnější zdroj z předem stanovené množiny, nicméně nelze rozsah této množiny měnit.

2.3 Kompatibilita

Dle [Gio15] používá platfromu Android přibližně 4000 typů zařízení. Pomineme-li systémy typu Android TV, Android Wear, Android Auto a budeme-li brát v potaz pouze mobilní zařízení, zjistíme, že i tak je platforma Android rozdrobena do mnoha verzí operačního systému. V tabulce 2.1 jsou vypsány nejpoužívanější z nich:

Verze	**Označení**	**API**	**Podíl**
2.2	Froyo	8	0.3%
2.3.x	Gingerbread	10	5.6%
4.0.x	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

Tabulka 2.1: Verze platformy Android, zdroj: [Apf15]

V tabulce 2.1 je názorně vidět, že každá nová verze operačního systému Android přichází s novým API[2]. Dle rozsahu změn v daném API se pak mění odpovídajícím způsobem i číslo verze systému. Verze operačního systému Android M nebyla do tabulky zanesena, neboť zatím není k dispozici oficiální vydání tohoto systému, ani není známo jeho označení.

Většinou nové API přináší nové prvky uživatelského rozhraní, zjednodušený přístup k některým funkcím telefonu a podobné funkce. Při tvorbě aplikace pro platformu Android je třeba v *manifestu* aplikace specifikovat nejnižší možnou verzi API, na které bude možné aplikaci provozovat. Zvolené verzi systému pak musí být podřízeny všechny funkce tak, aby byla zachována kompatibilita aplikace.

U polymorfní aplikace má volba další logický důsledek - kromě samotné polymorfní aplikace musí být vybranému API podřízeny i jednotlivé generované aplikace, které budou polymorfní aplikaci využívat pro svůj chod.

[2]Application Programming Interface - rozhraní pro programování aplikací

Pro platformu Android je k dispozici tzv. *Support Library* - knihovna, která zajišťuje zpětnou kompatibilitu aplikací. V praxi je tak například umožněno použití prvků grafického rozhraní na starší verzi API, než pro kterou byly tyto prvky vytvořeny. Tím je ve výsledku dosaženo jednotného vizuálního stylu aplikace napříč různými verzemi systému Android. Použití a obsah Support Library popisuje [Asl15].

Při použití Support Library se ukazuje další pozitivum polymorfních aplikací. U klasických aplikací, které využívají jakékoli funkce či prvky z kolekce Support Library, musí být tako knihovna součástí projektu a tedy i výsledné aplikace, což má následně vliv na výslednou velikost aplikace. Pokud bychom měli takových aplikací více (např. 10) a každá z nich by používala Support Library (soubor o velikosti 1,5 MB), znamenalo by to mnohem více paměti zařízení zabrané instalačními soubory (10 * 1.5 MB = 15 MB místa v paměti).

V dnešní době se sice výkonové parametry mobilních zařízení neustále zvyšují, nicméně velikost paměti zařízení je vždy o něco menší, než by bylo potřeba, ať je jakákoli. Polymorfní aplikace, která implementuje maximum knihoven pouze jednou, tak přináší vítanou úsporu místa v zařízení.

3 Alternativní přístupy

V následující kapitole jsou popsána řešení, která umožňují oproti klasickým mobilním aplikacím snadnější změnu obsahu prezentovaného uživateli mobilního zařízení, nebo používají jiné technologie, kterými se podobají polymorfním aplikacím.

Některá řešení by přitom mohla při vhodném použití polymorfní aplikaci zcela nahradit. Jednotlivé části této kapitoly pak popisují oblasti, ve kterých daná řešení oproti polymorfní aplikaci ztrácí, ale rovněž i atributy, kterými polymorfní aplikaci předčí.

3.1 Mobilní web

Přístup k internetu je jednou z hlavních vlastností tzv. chytrých telefonů. Množství uživatelů, které používají pro prohlížení webu[1] primárně své mobilní zařízení, se samozřejmě liší dle ekonomických, technických i sociálních faktorů vybrané země. Například [Ceb14] uvádí, že zemí s nejvyšší mírou přístupu k internetu pomocí mobilních zařízení je s 99% Keňa, zatímco zemí s největším zastoupením chytrých telefonů (oproti jiným zařízením) je Nigérie s 66%.

Obecně je pak možné říci, že podíl webového obsahu konzumovaného uživateli na mobilních zařízeních oproti používání klasických osobních počítačů se každoročně zvyšuje, a oblast mobilního webu bude i nadále růst na významu. Zatímco čísla z předchozího odstavce představují bezesporu zajímavé srovnání, následující odstavce jsou zaměřeny na největší specifika využití mobilního webu.

3.1.1 Dostupnost připojení

Aby bylo možné využívat připojení k internetu na mobilním zařízení, je třeba pokrytí mobilním signálem v dostatečné míře a kvalitě. Připojení pomocí lokální bezdrátové sítě lze samozřejmě považovat za výhodu, nicméně je to

[1] zkr. z World Wide Web - angl. pro celosvětová síť

právě připojení k internetu *kdykoli a kdekoli*, co odlišuje mobilní zařízení od osobních počítačů. V současnosti je pak synonymem pro rychlé připojení tzv. standard *4G* představovaný technologií *LTE*[2].

Tabulka 3.1 pak znázorňuje aktuální úroveň pokrytí LTE signálem v České republice. Pro úplnost je však nutné zmínit, že 4G zařízení nejsou v současné době zdaleka tak rozšířená, aby mohl výhod LTE připojení využívat každý. Mnohem častěji dochází k tomu, že je uživatel omezen rychlostí připojení starší verze 3G, a v některých lokalitách ještě pomalejším 2G. Možnost stahovat do aplikace pouze skutečně potřebný obsah tak může být pro mnoho uživatelů významným přínosem polymorfní aplikace.

Operátor	**Pokrytí území**	**Pokrytí obyvatel**
O2	82,2%	95,3%
T-Mobile	82,2%	95,2%
Vodafone	83,0%	96,4%

Tabulka 3.1: Pokrytí ČR kombinovaným signálem LTE (4G) a HSPA+ (3G), zdroj: [Ctu15]

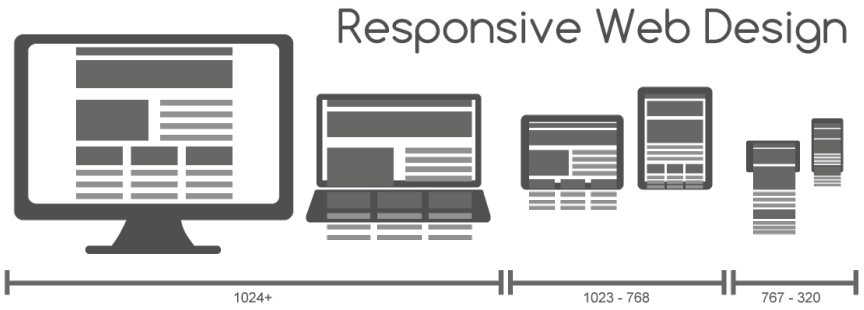
Zatímco mobilní aplikace fungující bez připojení k internetu (offline) je ve většině případů výhodou, protože může například znamenat i vyšší úsporu baterie mobilního zařízení, velká většina aplikací používá připojení k internetu pro svůj chod stejně jako mobilní web. Připojení je nutností zejména v případech, kdy je nutné komunikovat se vzdálenou databází, kontrolovat a pracovat s nejaktuálnějším obsahem, či provádět ověření totožnosti uživatele. Pokud aplikace takové funkce využívá, přibližuje se webovému řešení, a to zejména ve svém návrhu.

Subjekt, který uvažuje o vytvoření aplikace pro svoji propagaci, či pro vytvoření kontaktu a zprostředkování služeb uživateli, zpravidla mívá pro podobný účel vytvořené stávající řešení na bázi webových stránek. Dle funkcí, které jsou od aplikace vyžadovány, je pak rozhodováno o jejím případném vytvoření. Pokud není vyžadována vysoká míra interakce s uživatelem (viz část 3.1.3) nebo speciální funkcionalita, lze aplikaci nahradit mobilním webem, jak uvádí [Sum15]. Takové řešení může znamenat významnou finanční úsporu zejména pro malé subjekty.

[2]3GPP Long Term Evolution - Dlouhodobě rozvíjený standard skupiny 3GPP

3.1.2 Responsivní návrh

Responzivní, neboli přizpůsobitelný návrh webových stránek zajišťuje, že se daná stránka správně zobrazí na displejích různé velikosti. V praxi to pak znamená, že uživatel se dostane k vybranému obsahu bez ohledu na to, zda k prohlížení využívá počítač, mobilní telefon či televizor. Názorný příklad je k vidění na obrázku 3.1.



Obrázek 3.1: Responsive web design, zdroj: [Srr14]

Jeden z prvních návrhů responzivního webu představil [Mar10] a ještě před několika lety byl responsivní design výsadou několika vybraných stránek, dnes je považován za samozřejmost u všech nově vytvořených webů. Pro tvorbu responsivní webové stránky je možné použít více návrhů, které podrobně popisuje [W3r15]. V dnešní době je použití mobilního webu usnadněno i tím, že webový obsah je více standardizován a rozdíly mezi jednotlivými prohlížeči na úrovni zobrazení se do velké míry eliminovaly.

Výhodou použití mobilního webu je i velká rozšířenost nástrojů pro tvorbu webu. Společnost Wordpress.com je největším poskytovatelem nástrojů pro správu webu. Podle údajů [W3t15] je právě nástroj wordpress v 60,4% případů vybraným nástrojem pro správu webu. Což ve výsledku znamená, že 24% všech webových stránek používá zmíněné řešení. Dle [Wpc15] tedy stačí pouze vybrat jednu ze šablon pro návrh webu a vytvořit celé řešení takzvaně na klíč.

Zatímco vytváření klasické mobilní aplikace představuje poměrně náročný proces, vytváření dílčích aplikací pro polymorfní aplikaci tento proces poněkud zjednodušuje. Nicméně se ani tak nemůže vyrovnat jednoduchému návrhu webu pomocí již připravených šablon a komponent. Navíc má takový

web významnou výhodu v tom, že není vázán na žádnou platformu a vytvořením jednoho řešení je možné požadovaný obsah rozšířit mezi větší základnu uživatelů.

3.1.3 Interakce s uživatelem

Hlavním důvodem pro použití mobilního návrhu je fakt, že nabízí jednotnou prezentaci obsahu uživatelům za použití jednoduchých programovacích technik a komponent. Pomineme-li nutnost být neustále online pro zpřístupnění tohoto obsahu (viz část 3.1.1), má webové řešení i jednu poměrně negativní stránku - možnosti a kvalita interakce s uživatelem jsou u webu poněkud omezené.

Mobilní web ze své podstaty využívá architekturu typu *Request - Response*[3]. Jedná se tedy o komunikaci dvou zařízení, na které se projevuje komunikační zpoždění. Aby mohlo mobilní zařízení zobrazit webový obsah, musí ho nejprve stáhnout a načíst. Rozšíření rychlého mobilního připojení tuto nevýhodu do velké části odstraňuje, nelze ji však eliminovat úplně.

Mobilní zařízení nemají běžně k dispozici nástroj v podobě kurzoru, takže na mobilních zařízeních nelze použít CSS[4] selektor *:hover*[5]. Je proto brát ohled na toto omezení a vytvořit takový návrh stránek, aby se v něm uživatel dokázal snadno zorientovat i bez pomoci polohovacího zařízení.

Další nevýhodou je fakt, že aplikace prohlížeče, ve které se obsah zobrazuje, nemůže spolupracovat s ostatními aplikacemi či funkcemi na daném mobilním zařízení tak snadno, jako klasická mobilní aplikace. Synchronizace s uživatelskými účty vybraných služeb lze provést většinou i pomocí prohlížeče. Ale oproti spolupráci na úrovni dvou aplikací jsou možnosti webového přihlášení zpravidla omezenější a obvykle i méně uživatelsky přívětivé. Zcela znemožněná je pak například práce s fotoaparátem zařízení, kontakty, kalendářem, neboť webová stránka postrádá potřebná oprávnění pro přístup k osobním informacím uživatele na zařízení.

Problematikou uživatelského pohodlí mobilního webu se rovněž zabývá společnost Google, která v budoucí verzi operačního systému Android M

[3] z angl. pro Žádost - Odezva

[4] Cascading Style Sheets - angl. pro Kaskádové styly, `http://www.w3.org/Style/CSS/`

[5] položka nacházející se pod kurzorem - viz `http://www.w3schools.com/cssref/sel_hover.asp`

hodlá představit významná vylepšení. Z [Iwa15] vyplývá, že prohlížeč na platformě Android se bude chovat při vhodném nastavení jako mobilní aplikace. Budou tedy rozšířeny možnosti interakce s dalšími aplikacemi instalovanými na mobilním zařízení a rozdíl mezi mobilní aplikací a mobilním webem se opět o něco zmenší.

Nicméně takové řešení je vázáno nejen na specifickou platformu, ale dokonce na konkrétní prohlížeč a rovněž mu bude muset být přizpůsobena i zobrazovaná webová stránka, takže nelze hovořit o plošném řešení.

3.1.4 HTML5 Aplikace

Pro úplnost uvádím, že kromě pojmu mobilní web se lze setkat rovněž s označením *HTML5*[6] *aplikace*. V takovém případě se jedná o mobilní webovou stránku upravenou do takové podoby, že je takřka k nerozeznání od klasické aplikace. Nicméně jde stále o kód vykonávaný v prohlížeči. Takové označení se v některých případech používá i pro mobilní web.

Jistým propojením obou technologií mohou být tzv. *instalovatelné webové aplikace*, které popisuje [Iwa15]. Propojení webové aplikace a samotného zařízení je zvýšeno do takové míry, že uživatel vůbec nepozná, že aplikace běží v prohlížeči. Navíc je možné takové aplikace „instalovat“ do zařízení tak, že se v zařízení vytvoří odkaz na danou aplikaci a rovněž ikona pro spuštění jako u klasických apikací.

Nicméně se v současnosti jedná o koncept představený v rámci budoucího vývoje platformy Android. A jakékoli hodnocení tohoto návrhu by probíhalo zcela v rovině spekulací.

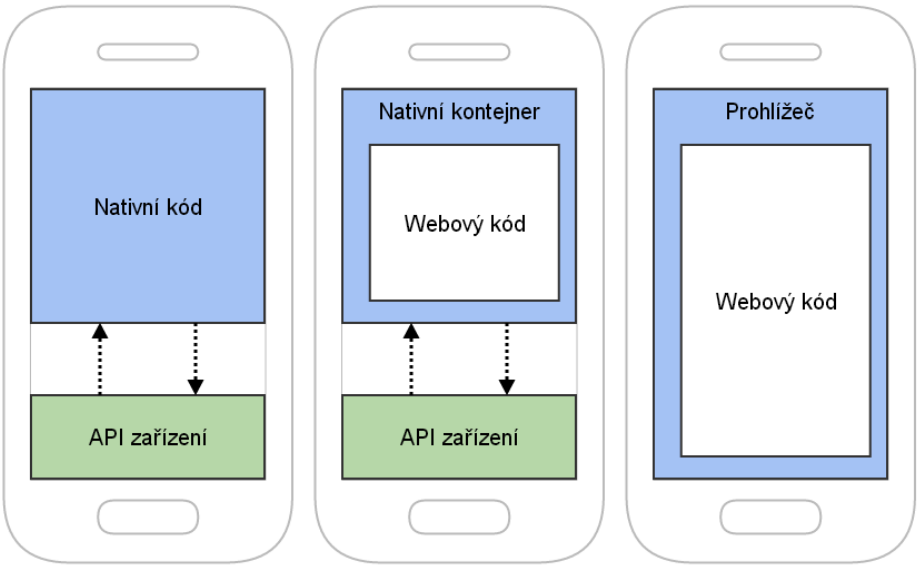
3.2 Hybridní aplikace

Hybridní aplikace představují zejména z vizuálního hlediska mezistupeň mezi klasickou aplikací a mobilním webem. Z hlediska návrhu je lze rozdělit do dvou skupin:

[6]HyperText Markup Language - angl. pro Hypertextový značkovací jazyk, specifikace dostupná na `http://www.w3.org/TR/html5/`

1. **WebView aplikace** - tzn. takové aplikace, které využívají interního prohlížeče k zobrazení obsahu. Takovými aplikacemi se zabývají následující odstavce.
2. **Kompilované aplikace** - aplikace, které jsou vytvořeny v jiném než nativním[7] programovacím jazyce vybrané platformy a jsou následně zkompilovány do nativního kódu dané platformy. Příklad takového řešení je popsán v části 3.3.

Schematické porovnání nativních, hybridních a webových aplikací je znázorněno na obrázku 3.2.



Obrázek 3.2: Nativní aplikace, WebView aplikace a mobilní web

3.2.1 WebView aplikace

Webview aplikace jsou postaveny na technologiích *HTML5* a *CSS3*. Takové aplikace obvykle používají jazyk JavaScript[8] pro obsluhu událostí a jako rozhraní s API platformy. WebView obsah je pak obalen pomocí rozhraní klasické aplikace, takže z hlediska operačního systému se WebView aplikace nijak významně neliší.

[7] přirozeném, z lat. nativus

[8] specifikace dostupná na: `http://www.ecma-international.org/publications/standards/Ecma-262.htm`

Pro využití WebView aplikace existuje řada důvodů. Hlavní z nich představuje znovupoužitelnost kódu, který může být společný napříč platformami. Pro každý operační systém se přizpůsobí logika aplikace, nicméně vizuální stránka zůstane zachována. To umožňuje jednak zpřístupnění totožného obsahu a služeb větší základně uživatelů, ale také určitou úsporu při vytváření aplikace.

Oproti mobilnímu webu je maximum obsahu aplikace uloženo přímo v zařízení uživatele. Při běhu aplikace pak dochází pouze k vykreslování samotného obsahu, ale eliminuje se většina komunikace se vzdáleným serverem či databází. Většina funkcí aplikace přitom využívá výpočetní výkon samotného mobilního zařízení, což umožňuje vytváření komplexnějších rozhraní i programů. Odstraněním nadbytečné komunikace se také zrychluje odezva na akce uživatele, neboť není nutné čekat na odpověď serveru.

3.2.2 Vývoj hybridních aplikací

Výhodou hybridních aplikací bývá využití některého z existujících frameworků[9] pro tvorbu rozhraní. To umožňuje jednak rychlejší tvorbu aplikací danou tím, že se pracuje hlavně s grafickou stránkou využívající technologie běžné pro webové stránky. Oproti klasickému postupu při vytváření aplikací, popsanému například v [Lac15].

Tento přístup také do velké míry snižuje požadavky na vývojáře aplikace, co se týká znalostí konkrétní platformy. Pro některé společnosti může být rovněž motivující použití levnější pracovní síly. Avšak hlavním ziskem oproti vývoji nativních aplikací zůstává časová úspora při vývoji, jak uvádí [Bmp13] společně s porovnáním několika vybraných frameworků.

Vybrané frameworky umožňují kromě vývoje samotných WebView aplikací i vývoj mobilního webu, což může dále usnadnit tvorbu aplikace, neboť je možné použít shodné grafické prvky pro obě řešení.

Jedním z důvodů pro volbu hybridní aplikace může být například jednotné vizuální propojení s webem společnosti. Zachování korporátní identity v rámci různých platforem může být pro některé subjekty velmi podstatné.

[9] aplikační rámec - sada vývojářských nástrojů

3.2.3 Technologická omezení

Oproti klasické aplikaci nabízí webové rozhraní hybridních aplikací poměrně odlišnou uživatelskou zkušenost, která nemusí být nutně výhodou. Sice je u nich eliminována odezva běžná pro mobilní web, nicméně ne všechny grafické prvky se chovají shodně jako odpovídající prvky nativní aplikace.

Zejména v případě, že je webové rozhraní shodné napříč několika platformami, případně shodné s mobilním webem, může být pro uživatele konkrétní platformy matoucí z hlediska odezvy. I proto v současnosti panuje trend používat takové grafické styly, aby aplikace co nejvíce připomínala nativní řešení vzhledem i odezvou, jak uvádí [Rud14]. Spokojenost uživatelů tak vítězí nad původní ideou sjednotit design pro všechny platformy.

Nevýhodou pro většinu společností je pak nemožnost indexace aplikací podobně, jako je tomu u internetových stránek. Obsah distribuovaný prostřednictvím mobilní aplikace nelze zahrnout do výsledků vyhledávače, ani pro něj použít SEO[10].

I v případě, že má aplikace strukturu shodnou s mobilním webem, tak není možné zlepšit pozici tohoto obsahu v porovnání s nekvalitním webem konkurence apod. Zatímco na jednu stranu může hybridní aplikace znamenat přísun nových uživatelů, musí využívat jiných forem zviditelnění a marketingu než mobilní web.

Hybridní aplikace využívající webového rozhraní mohou rovněž narazit na problémy při publikaci na klasických distribučních kanálech, např. *Google Play*[11] pro platformu Android. Všechny aplikace jsou před zveřejněním v obchodu analyzovány. A zatímco u klasických aplikací nebývá se strukturou aplikace problém, u hybridních aplikací nemusí být proces bez komplikací.

Z hlediska výkonnosti mohou mít hybridní aplikace problém zejména při zobrazování náročných scén, 3D[12] efektů, velmi rychlých animací a dalších přechodů, jak upozorňuje [Rud14].

Ve většině případů jsou možnosti hybridních aplikací omezeny možnostmi jazyka JavaScript, který je limitován více, než by tomu bylo u nativního řešení. Nicméně je třeba mít na paměti, že výkon mobilních zařízení je oproti

[10]Search Engine Optimization - angl. pro Optimalizace pro vyhledávače
[11]dostupné na `https://play.google.com/store`
[12]zkr. pro trojrozměrný, trojdimenzionální

osobním počítačům značně omezen, s čímž je třeba počítat již při návrhu aplikace.

3.3 Xamarin

Společnost Xamarin[13] je správcem platformy Mono[14], která slouží pro vytváření multiplatformních aplikací vytvořených v programovacím jazyku *C#*[15].

Xamarin nabízí k dispozici sadu nástrojů pro vývoj *nativních* aplikací pro všechny hlavní mobilní platformy. Vývoj programu probíhá buď ve firemním vývojovém prostředí Xamarin Studio nebo v programu Visual Studio se sadou rozšiřujících doplňků. Při kompilaci výsledného programu se pak provádí převod z kódu napsaného v jazyku C# do nativní formy pro danou platformu. Takový přístup umožňuje zachování maximálního pohodlí programátora i uživatele aplikace.

3.3.1 Xamarin Forms

Skutečnou, a reálně využitelnou výhodu oproti vývoji nativních aplikací představuje řešení zvané Xamarin Forms. Jedná se o způsob vytváření *formulářových* aplikací pomocí jazyka XAML[16].

Výhoda řešení Xamarin Forms tkví v tom, že využívá návrhu *Model-View-ViewModel*, kdy grafické rozhraní vytvořené pomocí XAMLu striktně odděluje od aplikační logiky funkcí napsaných v jazyku C#. Výsledné řešení lze pak zkompilovat pro platformy Android, iOS i Windows Phone bez nutnosti větších zásahů do kódu pro specifické platformy, jak uvádí [Xaf15].

Takový přístup maximalizuje znovupoužitelnost kódu a významně zkracuje dobu potřebnou na vývoj multiplatformní aplikace. Jednotný vývojářský nástroj pak významně snižuje požadavky na znalosti programátora, co se týká specifických řešení pro jednotlivé platformy. Jazyk XAML je používán

[13] viz `http://xamarin.com/`

[14] viz `http://www.mono-project.com/`

[15] specifikace dostupná na: `http://www.ecma-international.org/publications/standards/Ecma-334.htm`

[16] eXtensible Application Markup Language - značkovací jazyk pro tvorbu grafických rozhraní aplikací

například i při vývoji mobilních aplikací pro platformu Windows Phone, pro programy postavené na Windows Presentation Foundation a další řešení.

Oddělením výkonného kódu aplikace od předpisu pro rozhraní aplikace ve formě XML[17] souboru pak tento přístup velmi připomíná polymorfní aplikace. Nicméně se liší v tom, že u platformy Xamarin jsou obě části propojeny při kompilaci. Není tedy možné měnit chování či vzhled aplikace za jejího běhu. Nicméně použití jazyka XAML by mohlo možnosti polymorfních aplikací významně rozšířit.

3.3.2 Technologické výhody

Cílem této části práce není porovnání programovacích jazyků C# (používán platformou Xamarin) a Java (nativní řešení pro platformu Android). Následující odstavce popisují několik zásadních vylepšení, která při tvorbě aplikací platforma Xamarin nabízí.

Jednou z technologických výhod, které přináší vývoj pomocí jazyka C#, je použití knihoven a doplňků vytvořených přímo pro tento jazyk. I když se použitá platforma *Mono* zcela neshoduje s platformou *.NET* používanou pro vývoj aplikací pro osobní počítače, mají obě platformy mnoho společných prvků. Při programování aplikace v prostředí Xamarin je tedy možné využít například *Entity Framework*[18], nástoj *LINQ*[19] či další knihovny a balíčky funkcí.

Nedílnou součástí jazyka C# jsou pak tzv. *lambda funkce* a použití *delegátů*. Jedná se o prostředky, které mohou mnohdy vést k programátorsky čistému a zároveň efektivnímu řešení a platforma Xamarin jejich použití umožňuje.

Oproti tomu platforma Android postavená na jazyku Java použití takových nástrojů neumožňuje. Ač byly lambda funkce do jazyka Java zavedeny ve verzi 8, platforma Android podporuje pouze jazyk Java verze 7 (vybrané funkce už v API 19, oficiálně od API 21), a to ještě ne kompletně. I když řešení pro použití lambda funkcí pod verzí Java 5 a novější existuje, jak uvádí

[17] eXtensible Markup Language - značkovací jazyk

[18] Objektově-relační rozhraní, více viz `https://msdn.microsoft.com/en-us/data/ef.aspx`

[19] Language-Integrated Query - Sada dotazovacích funkcí, více viz `https://msdn.microsoft.com/cs-cz/library/bb397926.aspx`

[Luo15].

Rovněž je možné podporu lambda funkcí zprovoznit na platformě Android, čímž se zabývá [Tat15]. Nicméně se v obou případech jedná o neoficiální řešení, které nahrazuje inkriminované úseky kódu při kompilaci. Ale i toto řešení má mnoho nedostatků a sami autoři ho dle [Zai14] označují jako *Hack*, neboť obchází oficiální kompilátor jazyka Java.

Hlavní výhodou použití lambda funkcí je možnost nahradit systémová rozhraní obsahující pouze jednu metodu. Přesně takový případ je ukázán na následujícím úseku kódu, který představuje obsluhu stisknutí tlačítka v uživatelském rozhraní. Ukázka byla převzata z [Zai14].

```
// Rozhraní definované Android SDK
interface OnClickListener {
    public void onClick(View v);
}

// implementace provedená programátorem
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // provedení akce
    }
});
```

Ukázka kódu 3.1: Obsluha tlačítka - klasicky

Při použití lambda funkcí může být kód zjednudušen na následující výraz:

```
mButton.setOnClickListener((View v) -> {
    // provedení akce
});
```

Ukázka kódu 3.2: Obsluha tlačítka - lambda výraz

V budoucnu může být jedním z důvodů pro volbu jazyka C# i fakt, že základní součásti platformy .NET jsou od roku 2014 uvolněny jako *open-source*[20] (viz [Lan14]), což může tomuto řešení nahrávat, neboť vývojáři tak nebudou omezeni pouze na jazyk Java.

[20]angl. pro Otevřený software

4 Návrh polymorfní aplikace

Jak bylo uvedeno již v kapitole 2, navazuje tato diplomová práce na projekt představený Ing. Milanem Staffou v práci [Sta13]. V této kapitole jsou popsány jak části návrhu, které zůstaly zachovány, tak části polymorfní aplikace, které byly pozměněny nebo přidány oproti původnímu řešení.

Z hlediska funkcionality celé aplikace zůstal zachován návrh rozdělení aplikace na dvě klíčové části, kdy jedna má za úkol výběr samotných vizuálních a funkčních podkladů, které se použijí pro vytvoření dceřinné aplikace, druhá část pak slouží k obsluze samotné generované aplikace.

Zatímco původní verze provádí načtení programového kódu až při běhu generované aplikace, nový návrh řešení cílí na to, aby všechny důležité části generované aplikace byly připraveny již před spuštěním generované aplikace. Jak je toho docíleno, popisuje část 4.1.

M. Staffa se ve své práci zaměřil z velké části na komunikaci aplikace se vzdáleným serverem, vykonávání vzdálených procedur a zabezpečení komunikace. Aplikace popsaná v této práci cílí především na vykonávání kódu v samotném zařízení. Tato změna byla umožněna hlavně faktem, že v posledních několika letech zaznamenala mobilní zařízení znatelný nárůst z hlediska výpočetní síly a některá zařízení svým výkonem leckdy i předčí starší osobní počítače.

Smysl jednoduchých generovaných aplikací formulářového typu však zůstal zachován a vykonávané dílčí aplikace se stále skládají především z kódu s jednoduchou funkcionalitou. Ve své pokročilejší verzi však polymorfní aplikace nabízí rozmanitější uživatelské rozhraní, které je možné vytvářet jak pro samotnou generovanou aplikaci, tak jako výsledek její činnosti. Rozšíření palety grafických prvků pro tvorbu generovaných aplikací umožňuje tvorbu rozmanitějších rozhraní. Provedené změny jsou popsány v části 4.2 a samotnou realizaci grafického návrhu popisuje část 5.1.

Rozšíření původní práce rovněž představuje nové typy aplikací, které lze využívat. Stejně jako v původním návrhu cílí ukázkové aplikace na spuštění jednoduché funkce z připojené knihovny, nicméně programy popsané v kapitole 7 slouží pro vyhledání obsahu, který by jinak byl pro uživatele hůře dostupný či zobrazitelný.

Ze zdroje zpráv či vybraného webu získávají ukázkové aplikace data, která pak uživateli zobrazují ve formě vhodnější pro mobilní zařízení. Samotné načítání těchto dílčích programů do polymorfní aplikace probíhá stejným způsobem jako v práci [Sta13], zejména však proto, že platforma Android stále nenabízí efektivnější dynamické načítání spustitelného kódu za běhu aplikace.

Následující tabulka 4.1 přehledným způsobem ukazuje nejvýznamnější změny pokročilého návrhu polymorfních aplikací představeného v této práci.

Původní verze	**Aktuální verze**
• XML šablona + knihovna dex	• XML šablona + knihovna dex
• Aktivita pro načtení + aktivita pro generovanou aplikaci	• Aktivita pro načtení + aktivita pro generovanou aplikaci
• Soubory pouze z URL[1]	• Stahování z URL, Google Drive, Dropbox, lokální úložiště, historie vybraných zdrojů
• Jedna obrazovka	• Dynamická práce s fragmenty
• Neměnné rozhraní	• Změna vzhledu za běhu aplikace
• Vykonávání vzdáleného kódu	• Vykonávání lokálního kódu
• Volání jednoduchých funkcí	• Samostatně použitelné moduly
• Cílem spuštění funkce	• Cílem prezentace dat
• Základní podoba prvků	• Přizpůsobení významu a obsahu
• Žádný přesah aplikace	• Spolupráce s telefonem, prohlížečem, e-mailovým klientem

Tabulka 4.1: Porovnání původního a současného řešení

4.1 Princip fungování

Aby mohla polymorfní aplikace správně zobrazit a vykonávat generovanou aplikaci, potřebuje dva základní prvky:

1. Předpis definující zobrazení prvků, jejich vlastnosti a tím pádem i způsob, jakým budou vykresleny na displeji zařízení - tj. určitou *Šablonu*.

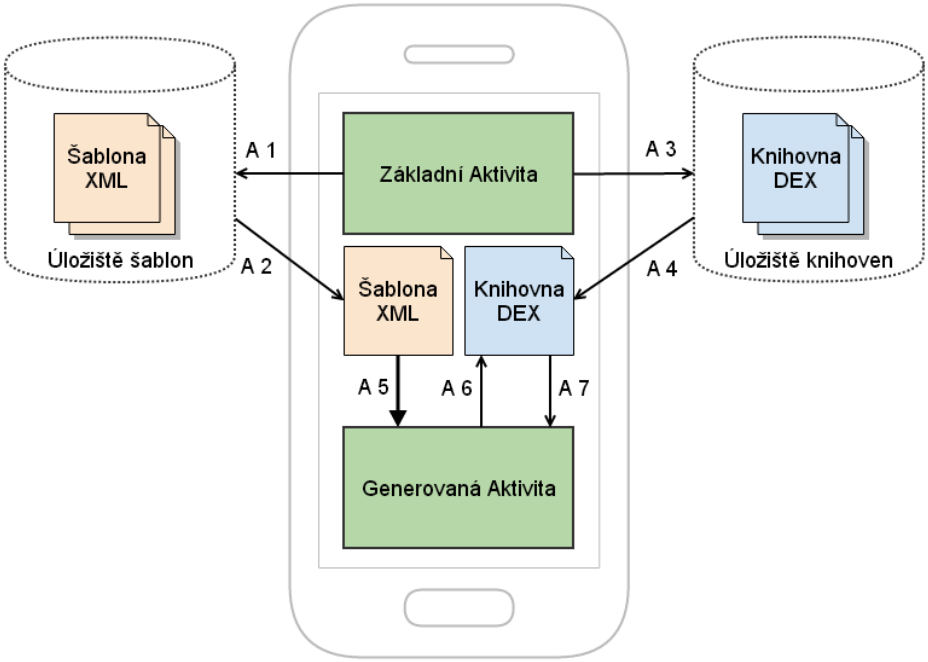
2. Programový kód v přeložené a spustitelné podobě, který bude moci za běhu aplikace načíst a vykonat - tedy spustitelnou *Knihovnu*.

Celou polymorfní aplikaci lze tedy z logického hlediska rozdělit na dvě části. První část bude opatřovat potřebné soubory a zajistí jejich přesun do zařízení. Druhá část pak na základě poskytnutých souborů vygeneruje uživatelské rozhraní a spustí programový kód, který má k dispozici.

Vytváření uživatelského rozhraní dle šablon popisuje část 5.1.3. Práce s programovým kódem je rozebrána v části 5.2.2.

Zmíněné dvě části aplikace jsou představovány dvěma samostatnými programovými prvky typu *Activity* (též aktivita). Následující obrázek 4.1 znázorňuje rozsah činností těchto aktivit a jednotlivé kroky nutné pro úspěšný běh polymorfní aplikace, v obrázku označené jako akce *A1 - A7*.

Pro jasnější odlišení jsou zelenou barvou zvýrazněny aktivity vykonávané polymorfní aplikací, oranžová barva značí šablonu grafického rozhraní dceřinné aplikace, modrá barva představuje programovou část, která se vykonává v generované aplikaci.



Obrázek 4.1: Běh aplikace

- **A 1** - Uživatel pomocí první aktivity vybere soubor šablony z libovolného zdroje. Je možné načíst soubor z paměti zařízení, zadaného URL, z úložišť Dropbox či Google Drive (práci s úložišti popisuje část 5.3).
- **A 2** - Soubor šablony se musí z vybraného umístění přesunout do paměti zařízení, aby s ním bylo možné pracovat.
- **A 3** - Analogicky probíhá výběr souboru spustitelné knihovny.
- **A 4** - Stejně tak i její stažení do zařízení pro budoucí použití.
- **A 5** - Na základě stažené šablony se vytvoří uživatelské rozhraní jedné či více obrazovek generované aplikace.
- **A 6** - Z generované aplikace je možné spustit připravený programový kód.
- **A 7** - Výsledkem běhu programového kódu je další šablona pro vzhled generované aplikace, kterou se nahradí či rozšíří původní vzhled.

4.2 Změny v uživatelském rozhraní

Původní návrhy polymorfní aplikace představené v pracích [Hul12] a [Sta13] pracují pouze se základními grafickými prvky typu *Button* (tlačítko), *TextView* (zobrazení popisku či textu) a *EditText* (kolonka pro zadání textu). Aby však bylo možné programy generované v polymorfní aplikaci efektivě používat, bylo nutné nabídku těchto prvků rozšířit. Nejedná se o rozšíření, co se do počtu použitých objektů týče, ale spíše o využití povahy jednotlivých prvků, jejich rozlišení z hlediska důležitosti a funkce.

Současná verze polymorfní aplikace využívá následující prvky uživatelského rozhraní:

- **Tlačítko** - oproti původnímu návrhu nezměněno, slouží ke spuštění přiřazené funkce.
- **Text** - nově umožňuje rozlišit důležitost informace pomocí zvýraznění typu kurzíva a/nebo tučné písmo. Tím je umožněno jasné strukturování obsahu na displeji zařízení. Dále je zdůrazněna významová stránka obsahu - v případě, že je v textu uvedeno telefonní číslo, je možné kliknutím toto číslo vytočit. Kliknutí na odkaz zase slouží pro přechod

do internetového prohlížeče. Tyto úpravy výrazně zlepšují uživatelskou zkušenost s používáním generované aplikace.

- **Kolonky** - je možné dle požadovaného obsahu přizpůsobit rozložení klávesnice zařízení pro několik vybraných typů (datum, text, telefonní číslo, atd.).
- **CheckBox** - prvek sloužící pro zapnutí či vypnutí přiřazené volby. Dále rozšiřuje možnosti volané funkce.
- **Obrázky** - mnohdy zprostředkují potřebnou informaci efektivněji než slovní popis, mohou rovněž sloužit pro doplnění textu o další rozměr informace.

Povaha polymorfní aplikace omezuje možnosti a typ použitých prvků oproti klasickým aplikacím. Hlavními požadavky na rozhraní generované aplikace jsou přehlednost, jednoduchost, ale rovněž schopnost vygenerovat rozhraní pro předem neznámé rozložení a počet prvků.

Vzhledem k tomu, že jakýkoli dynamický prvek rozhraní (např. *Slider* či *Picker*) je z podstaty svázán s určitou proměnnou obsahující jeho hodnotu a obslužnou funkcí, která obsluhuje manipulaci s takovým prvkem, není možno tyto prvky jednoduchým způsobem vytvořit dynamicky. Efektivní práce s takovými *aktivními* prvky vyžaduje předchozí znalost rozložení uživatelského rozhraní, proto byly z polymorfní aplikace vynechány.

Podrobný popis všech komponent použitelných v rozhraní polymorfní aplikace je uveden v příloze C.

5 Realizace polymorfní aplikace

Zatímco předchozí kapitola 4 se zabývala především strukturou polymorfní aplikace v porovnání s řešeními známými z předchozích let, cílem této kapitoly je popsat samotný princip, na kterém polymorfní aplikace reálně běží. Nezabývá se sice dopodrobna všemi použitými třídami, metodami a dalšími komponentami programu, nicméně by po jejím přečtení měl být jasný způsob jejího fungování.

5.1 Uživatelské rozhraní

Jak bylo již uvedeno v části 4.1, je základním stavebním prvkem klasické aplikace na platformě Android objekt typu aktivita - programová část obsluhující události, vykonávající kód hlavního vlákna programu, tzv. UI Thread[1]. U jednoduchých aplikací je každé aktivitě přiřazen soubor typu *Layout*, který pevně definuje rozložení prvků rozhraní dané aktivity a přiřazuje jim funkce, které mají vykonat. Aktivita společně s layoutem pak tvoří z pohledu uživatele jednu obrazovku aplikace.

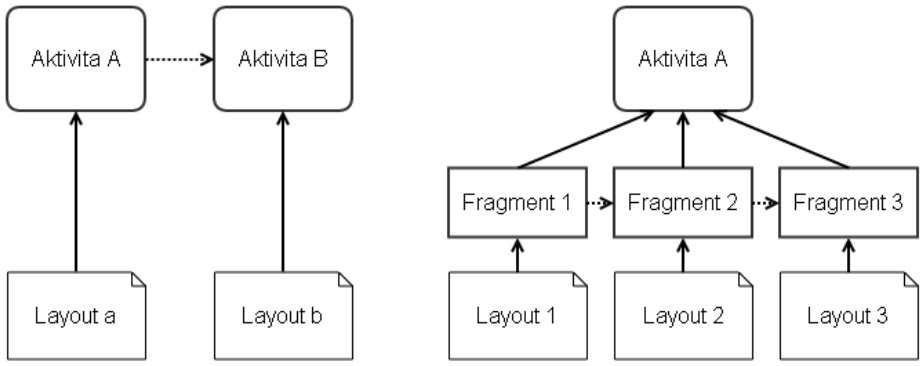
Aby bylo možné využívat různá rozložení prvků a jednoduše mezi nimi přepínat, lze použít prvky typu *Fragment*. Fragment představuje část aktivity, kterou lze dynamicky vytvářet či ukončovat, čímž je možné docílit jednodušší navigace v obsahu, či vytvořit vícepanelové rozložení obrazovky (původní účel fragmentů).

Ve většině případů mají fragmenty, podobně jako aktivity přiřazený předem vytvořený soubor layoutu. Narozdíl od aktivit, se ale jedná především o prvky uživatelského rozhraní a samotný kód aplikace je stále obsluhován nadřazenou aktivitou fragmentů. Toto schéma znázorňuje obrázek 5.1.

5.1.1 Struktura prvků

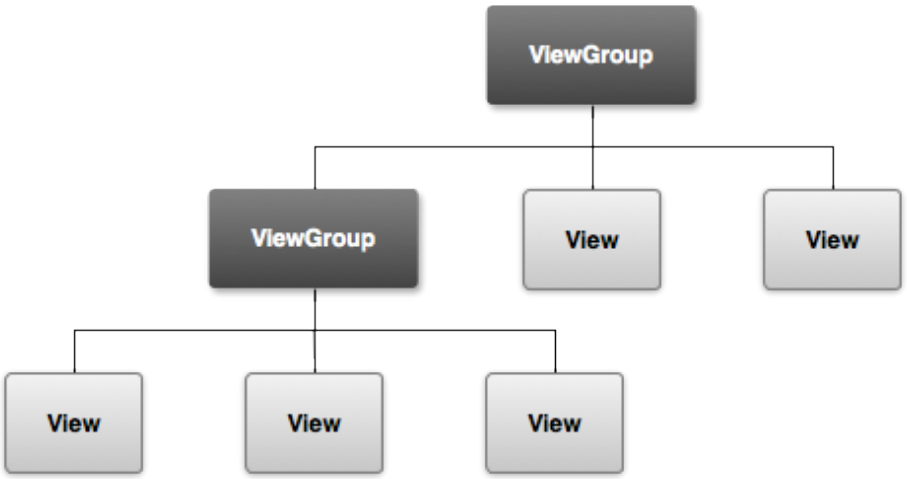
Platforma Android využívá v uživatelských rozhraních objekty typu *View*. Jedním z podtypů tohoto objektu je *ViewGroup*, který může obsahovat další

[1]User Interface Thread - angl. pro Vlákno uživatelského rozhraní



Obrázek 5.1: Struktura aplikace

prvky typu View či ViewGroup. Kombinací těchto prvků dochází k vytvoření stromové struktury rozhraní, která je znázorněna na obrázku 5.2.



Obrázek 5.2: Struktura prvků View, zdroj: [Aui15]

Soubor layoutu typicky definuje rozložení uživatelského rozhraní pomocí jednoho nebo více prvků typu ViewGroup, který obsahuje další specializované prvky typu View. Nejpoužívanějšími prvky typu ViewGroup jsou:

- *LinearLayout* - seznam prvků, kde každé View představuje jeden řádek (resp. sloupec, dle orientace).
- *RelativeLayout* - rozložení, které umožňuje definovat vzdálenosti a umístění prvku dle jeho sousedních prvků.

- *ScrollView* - rozložení, které přizpůsobí obsah velikosti displeje tak, aby bylo možné zobrazit i prvky, které by se jinak vykreslily mimo obrazovku.

5.1.2 Vytváření prvků

V klasické aplikaci jsou všechny důležité prvky View definovány v příslušném souboru layout, nicméně je možné dle potřeby vytvářet prvky i dynamicky (za běhu aplikace, po vykreslení daného fragmentu či aktivity). Klasický přístup, ale i dynamické vytváření programových součástí aplikace platformy Android se detailněji zmiňuje například [Gra13].

Možnost vytvářet prvky uživatelského rozhraní programově je základem tvorby grafického rozhraní polymorfní aplikace. Ta namísto klasických souborů typu layout používá podobné definice v podobě XML souborů, které se použijí při načtení daného fragmentu (viz. následující část 5.1.3. V klasické aplikaci je možné všem předem známým prvkům typu View přiřadit unikátní identifikátor v rámci aplikace, který slouží k vyhledávání daného prvku a změně jeho parametrů.

U polymorfní aplikace však není možné předem definovat jedinečné identifikátory tak, aby bylo možné zaručit, že se v rámci aplikace nebude vyskytovat prvek se stejným ID[2]. V aplikaci jsou identifikátory reprezentovány jako unikátní zdroje typu integer[3], ale programátor s nimi zachází pomocí jejich slovního názvu, aniž by znal samotnou hodnotu identifikátoru vygenerovanou aplikací.

Nemožnost používat jednoduše identifikátory prvků do velké míry znemožňuje úpravy rozhraní na úrovni konkrétních prvků za běhu polymorfní aplikace. Nicméně je stále možné pro prvek vybraný jiným způsobem (procházení seznamu, prvek jako původce události) měnit jeho parametry či obsah. Pokud se jedná o prvek typu ViewGroup, lze rovněž měnit jeho potomky.

Pro zjednodušení a možnost jednoznačného pozicování prvků byla množina použitých prvků omezena na několik základních typů: *LinearLayout, TextView, EditText, Button, ImageView* a *CheckBox*.

I tento poměrně limitovaný počet prvků nicméně umožňuje tvorbu rela-

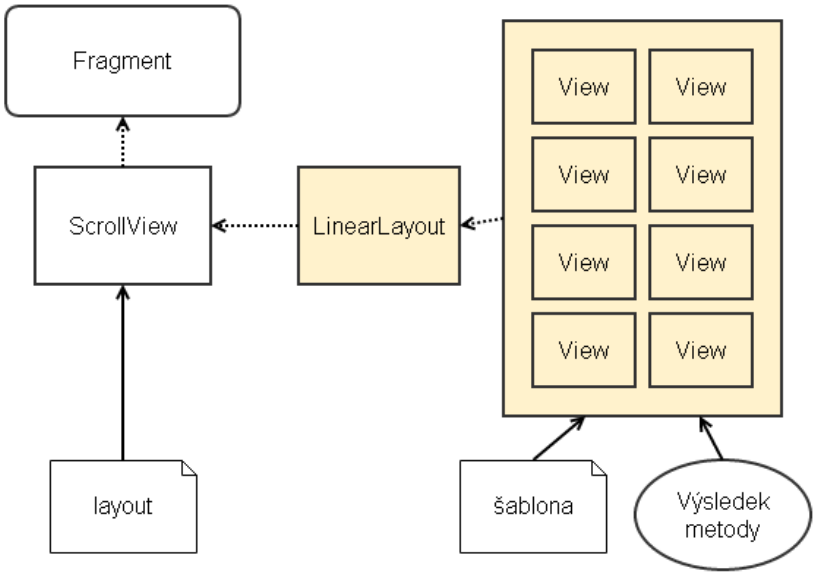
[2]zkratka pro identifikátor
[3]celé číslo desítkové soustavy

tivně rozmanitých uživatelských rozhraní při zachování přehlednosti šablon. Jednotlivé prvky a atributy, které jim lze přiřadit, jsou popsány v příloze C.

5.1.3 Šablony rozhraní

Šablony rozhraní představují alternativu k souborům typu layout. Jedná se vždy o validní XML, buď v podobě souboru, nebo jako řetězec (String), který je výsledkem funkce běžící v programu. Jak ukazuje obrázek 5.3, používá každý fragment generované aplikace klasický soubor layout. Nicméně tento soubor definuje pouze obalující scrollView pro další obsah. Jedná se tedy o definici, jak vykreslit fragment bez ohledu na jeho obsah.

Jednotlivé prvky View ve fragmentu jsou navíc obaleny pomocí LinearLayout, který zajišťuje rozmístění prvků na jednotlivé řádky rozhraní v pořadí, ve kterém jsou do něj vloženy. Prvky generované dynamicky při vytvoření fragmentu jsou v obrázku 5.3 zvýrazněny žlutou barvou.



Obrázek 5.3: Načítání obsahu Fragmentu

Každá šablona využívá toho, že všechny prvky typu View jsou v aplikaci reprezentovány pomocí stromu, jak znázorňuje obrázek 5.2. Proto jsou i v šablonách definovány odpovídajícím způsobem jako strom elementů.

Šablony jsou zpracovávány pomocí DOM[4] parseru, jehož definici popisuje [Dom04]. Každá šablona může obsahovat definici více prvků typu fragment (použito zejména u souborů), u provádění kódu z načtených knihoven se však používá jedna šablona pro jeden fragment, jak podrobněji popisuje 5.2. Příklady použitých šablon je možné najít v příloze D.

5.1.4 Načítání fragmentů

Pro generovanou aplikaci je vytvořena jediná aktivita, v rámci které se generují potřebné fragmenty dle použité šablony. Tato aktivita používá objektu *SectionsPagerAdapter*, který umožňuje dynamické vytváření fragmentů při výběru jedné ze záložek v navigačním prvku *ToolBar*[5].

Protože počet vygenerovaných panelů není předem znám, jsou vytvářeny vždy dle aktuální potřeby, což zajišťuje třída *FragmentStatePagerAdapter*, od které *SectionsPagerAdapter* dědí. Mezi panely lze rovněž přecházet pomocí posouvání aktuálního fragmentu do stran (tzv. *Swipe Views*).

Obsah jednotlivých fragmentů je uložen v seznamu v aktivitě zastřešující práci s nimi. Při pokusu o přístup do jiného než aktuálního fragmentu se vybere View na odpovídající pozici seznamu a je předán nové instanci fragmentu ke zobrazení. Tím, že jsou všechny prvky View v seznamu vytvořeny předem, dochází k rychlému načtení obsahu fragmentu, aniž by bylo třeba samotný obsah vytvářet vždy znovu. Po odchodu z aktuálního fragmentu dochází k zrušení jeho instance, ale obsah ve formě View je stále dostupný pro opětovné využití.

V rámci aktivity je možné rovněž aktuální fragment uzavřít, není-li tento fragment jediným vygenerovaným. Fragmenty, které byly načteny ze souborové šablony, lze rovněž libovolně duplikovat. Cílem této akce je vytvořit nový fragment, který umožňuje vykonávání kódu, který mu byl přiřazen v definici v původní šabloně. Fragmenty, které jsou produktem vykonaného kódu, duplikovat nelze.

Namísto takového fragmentu se vytvoří kopie původního fragmentu v takové podobě, v jaké byl před vykonáním přiřazené funkce (resp. v jaké byl načten ze souboru šablony). Kombinací zavírání a duplikace fragmentů je

[4]Document Object Model - angl. pro Objektový model dokumentu
[5]Nástrojový nebo též navigační panel

mimo jiné umožněno zavření nepotřebných fragmentů výsledku a opětovné spuštění funkce.

Pokud fragment obsahuje objekty ImageView, které mají přiřazenu adresu (URL) zdroje obrázku, proběhne po prvním zobrazení daného fragmentu načtení těchto obrázků. Pro každý prvek ImageView se vytvoří objekt typu bitmap, který představuje samotnou obrazovou informaci, a je do ImageView přiřazen. Soubory obrázků se vytvářejí v cache[6] zařízení, takže mohou být odstraněny dle potřeb systému, nebo při spuštění čisticího programu.

5.2 Použití knihoven

Programy vytvořené v jazyku Java pro osobní počítače vznikají kompilací zdrojových kódů (soubory .java) do bytecode - strojově čitelného kódu (soubory .class). Soubory bytecode lze dále ještě komprimovat zabalením do archivu (soubory .jar) spolu s dalšími součástmi programu a následně je kód vykonáván pomocí JVM[7].

Oproti tomu zařízení běžící na platformě Android 4.4 a starší používají vlastní virtuální stroj DVM[8], který funguje na principu architektury registrů, na rozdíl od JVM, který používá zásobníkovou architekturu. Prostředí Dalvik[9] využívá just-in-time (JIT) kompilaci, při které je spuštěný program kompilován přímo v době provádění.

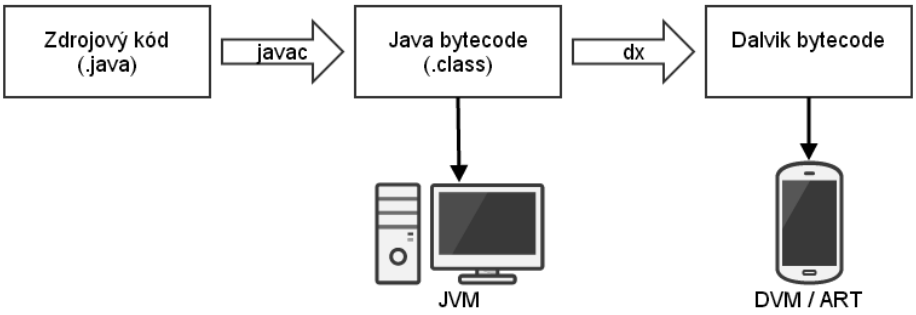
Od verze Android 5.0 je Dalvik nahrazen prostředím ART[10], které kód programu kompiluje předem, při jeho instalaci, pomocí ahead-of-time (AOT) kompilátoru. Stejně tak kompiluje programové části systému při startu zařízení. Předem kompilovaný kód by měl přinést zrychlení běhu aplikací a ve výsledku vyšší výkon zařízení. Prostředí ART je kompatibilní s aplikacemi vytvořenými pro Dalvik, porovnání obou uvedených systémů popisuje [Asa15].

[6] V tomto kontextu dočasná paměť'
[7] Java Virtual Machine - angl. pro Virtuální stroj Javy
[8] Dalvik Virtual Machine - angl. pro Virtuální stroj Dalviku
[9] více informací na `http://www.dalvikurbyggd.is/EN/About-Dalvik/`
[10] Android Runtime - angl. pro aplikační (běhové) prostředí

5.2.1 Dalvik executable archiv

Důsledkem použití odlišných virtuálních strojů na mobilních zařízeních je nutnost dodatečné kompilace kódu aplikace pro mobilní prostředí. Mobilní aplikace používají namísto klasických jar archivů tzv. dex, neboli *Dalvik Executable*, soubory popsané v [Asd15]. Rozdíl mezi obsahem jar a dex archivu je ukázán v příloze na obrázku E.1. Z hlediska bezpečnosti může být důležitý i fakt, že oproti jar archivu se nedá obsah dex archivu prohlédnout bez pomoci specializovaných dekompilačních nástrojů.

Následující obrázek 5.4 ilustruje proces kompilace dex souborů do podoby spustitelné na platformě Android.



Obrázek 5.4: Kompilace dex souborů

Při vytváření aplikace pro platformu Android dochází k dodatečné kompilaci automaticky. Výsledný soubor *apk* se sestává z manifestu aplikace, zdrojů aplikace, dodatečných knihoven a souboru *classes.dex*, který obsahuje všechny překompilované zdrojové kódy potřebné k běhu aplikace.

Samostatně stojící archivy dex je ale třeba vytvořit manuální kompilací jar archivu pomocí nástroje *dx*, který je základní součástí nástrojů Adroid SDK[11]. Nástroj dx umí do výsledného archivu zabalit soubory tříd, které jsou mu dány k dispozici, nicméně problém nastává u knihoven třetích stran, kdy není možné manuálně ovlivnit obsah výsledného archivu.

Může tak dojít k tomu, že z původní knihovny budou vynechány například soubory typu *.property*, se kterými prostředí Dalvik standarně nepracuje, což zapříčiní nefunkčnost aplikace. Knihovny třetích stran je tak třeba přidat přímo do projektu polymorfní aplikace, který je přibalí do výsledného souboru

[11]Software Development Kit - angl. pro Sada vývojových nástrojů

apk. Tím je bohužel vyloučena úplná nezávislost generovaných aplikací na jejich mateřské aplikaci. Na druhou stranu je možné tutéž knihovnu použít ve více dílčích aplikacích.

Příkaz pro kompilování archivů pomocí nástroje dx vypadá následovně:

```
dx --dex --keep-classes --output="./vysledek.dex"
"./puvodni.jar"
```

Ukázka kódu 5.1: Kompilace dex archivu

5.2.2 Načítání archivů

Standardně jsou při běhu aplikace načítány pouze soubory obsažené v jejím vlastním archivu *classes.dex*, který je součástí instalačního balíku aplikace *apk*. K načítání kompilovaného kódu aplikace slouží třída *DexClassLoader*, která zpřístupňuje třídy obsažené v archivech jar a apk. Tuto třídu lze využít rovněž k manuálnímu načtení dex souborů z paměti zařízení.

Následující ukázka kódu představuje vytvoření instance třídy DexClassLoader, která jako parametry bere:

- Vstupní soubor dex;
- Adresář v cache aplikace, do kterého se rozbalí potřebné soubory;
- Případný soubor nativních knihoven;
- ClassLoader používaný obalující Aktivitou.

```
File dexFile = Utilities.getFileInWorkingDirectory(context,
    dexFilename);
File dexOutput = context.getDir("outdex",
    Context.MODE_PRIVATE);

DexClassLoader dcl = new
    DexClassLoader(dexFile.getAbsolutePath(),
    dexOutput.getAbsolutePath(), null,
    context.getClassLoader());

Class libProviderClass = null;
libProviderClass = dcl.loadClass(targetClass);
```

Ukázka kódu 5.2: Použití DexClassLoaderu

Po načtení potřebné třídy je možné vytvořit její instanci a spustit některou z jejích metod. V realizované aplikaci je konzistence načítání zajištěna tím, že všechny použité podprogramy musí implementovat společné rozhraní *ILibraryFunction*, které obsahuje metodu *doLibraryFunction*.

Zmíněná metoda je pak definována v jednotlivých knihovnách, takže ji lze vždy použít. Jako vstupní parametr bere seznam řetězců a výstupem je rovněž seznam řetězců. Tím je dosaženo velkého zobecnění při volání metody bez předchozí znalosti jejího obsahu či funkce.

```
interface ILibraryFunction
{
    public ArrayList<String>
        doLibraryFunction(ArrayList<String> parameters);
}
```

Ukázka kódu 5.3: Rozhraní dex archivu

Samotného volání metody je dosaženo pomocí reflexe. Při instancování metody je podstatná pouze samotná metoda *doLibraryFunction*, která je volána. Další metody a jiný obsah třídy jsou ignorovány. Získání výsledků volané metody vypadá následovně:

```
final Object myInstance  = libProviderClass.newInstance();
final Method doSomething =
   libProviderClass.getMethod("doLibraryFunction",
   ArrayList.class);
Object output = doSomething.invoke(myInstance, input);
```

Ukázka kódu 5.4: Volání knihovní funkce

Obecně je možné volat libovolnou metodu vybrané třídy, pro její použití stačí pouze zadat jméno metody a typ jejích vstupních parametrů. Použitím co nejobecnějšího návrhu však máme zajištěno, že právě metoda *doLibraryFunction* bude v cílené třídě implementována, což je podstatné zejména z hlediska jednoduchého návrhu polymorfní aplikace. V případě potřeby by bylo možné návrh rozšířit o volání libovolné funkce, u které by bylo pomocí reflexe nutné zjistit počet a typy všech jejích parametrů, například způsobem popsaným v [Omt15].

5.2.3 Obsluha generované aplikace

Jak popisuje předchozí část 5.2.2, je komunikace mezi generovanou aplikací a jejím programovým kódem realizována jednotně pomocí rozhraní *ILibrary-Function* a jeho metody *doLibraryFunction*.

Původcem volání externí metody je tlačítko, které má přiřazen objekt typu *Function*. Tento objekt se vytvoří na základě šablony, ze které bylo uživatelské rozhraní vytvořeno. Zápis v souboru šablony může vypadat například takto:

```
<Button text="Najít">
  <function type="replace"
     target="com.example.control.ContactFinder"/>
</Button>
```

Ukázka kódu 5.5: Přiřazení funkce tlačítku

Po stisku tlačítka dojde v obalující aktivitě k obsluze události, dle typu funkce se provede patřičná akce. Pokud má dané tlačítko typ funkce *replace*, znamená to, že se obsah aktuálního fragmentu bude nahrazovat obsahem vygenerovaným specifikovanou třídou odpovídající parametru *target*. Tato vybraná třída bude instancována pomocí nástroje *DexClassLoader* a bude v ní spuštěna metoda *doLibraryFunction*.

Parametrem této metody je seznam prvků typu String, který je generován ze všech editovatelných prvků typu View obsažených v aktuálním fragmentu. Mezi editovatelné prvky patří prvky typu *EditText* a *CheckBox*. Pro každý uvedený typ se vygeneruje jeden společný řetězec s obsahem prvků pomocí metody *itemsContentToString* obalující aktivity *ChildActivity*. z prvků typu EditText je extrahován jejich textový obsah, prvky CheckBox generují řetězec *true* či *false* dle aktuální hodnoty. Ve výsledku může vypadat vstup volané metody následovně:

```
"class android.widget.EditText=[text1;text2;...]",
"class android.widget.CheckBox=[true;false;...]"
```

Ukázka kódu 5.6: Struktura vstupních parametrů funkce

Následně se vytvoří instance třídy *ProcessFunctionTask*, která na pozadí provede načtení přiřazeného dex archivu, instancuje specifikovanou třídu a zavolá její metodu *doLibraryFunction* s předanými parametry. Volaná třída musí sama znát počet a typ parametrů, které potřebuje k vykonání metody.

Tyto parametry vybere ze vstupu, provede svůj kód a následně vrací obsah, který se má zobrazit ve Fragmentu, z něhož byla metoda zavolána.

Výstupním objektem volané metody je opět seznam typu *ArrayList* objektů typu *String*. Prvním prvkem seznamu je vždy návratový kód metody. V případě úspěšného vykonávání je daným kódem *OK*, v případě neúspěchu je to řetězec *ER*. Jednoduchým porovnáním pak volající aktivita zjistí, zda má dojít k překreslení obsahu (a v takovém případě zpracuje další prvky seznamu), či nikoli.

Všechny další prvky výstupního seznamu jsou tvořeny šablonami určenými pro překreslení jednotlivých fragmentů volající aktivity. Po doběhnutí úlohy na pozadí je výsledek předán právě volající aktivitě a zpracování výsledku probíhá již v rámci hlavního UI vlákna. V případě, že doběhla volaná metoda bez chyby, vyhodnocuje se nejprve počet navrácených šablon.

V případě, že je navrácena pouze jedna šablona, je překreslen obsah aktuálního fragmentu. V případě, že je šablon více, je pro každou vytvořen nový fragment, kterému je následně přiřazen obsah vzniklý načtením odpovídající šablony. Načítání obsahu přitom probíhá stejně tak, jak je popsáno v části 5.1.3.

5.3 Práce s úložišti

Jak bylo uvedeno v části 4.1, umožňuje základní aktivita polymorfní aplikace stažení šablon a dex archivů z několika zdrojů. Kromě samotného zařízení a síťového URL jsou takovým zdrojem i tzv. *cloudová úložiště*. Cloud v tomto případě znamená veřejně dostupnou službu pro ukládání dat na internetové servery, přičemž koncový uživatel je oddělen od samotné technologické implementace a k úložišti přistupuje pomocí specializované aplikace či internetového prohlížeče.

Cílem těchto služeb je záloha dat, sdílení dat mezi uživateli a umožnění dostupnosti dat uživatele z několika různých zařízení. V posledních letech se z cloudových úložišť rovněž stávají nástroje pro správu dokumentů. Mezi nejpoužívanější služby tohoto typu patří:

- **Dropbox** - odhadovaný počet uživatelů převyšující 300 milionů, dostupné na `https://www.dropbox.com/`.

- **Google Drive** - odhadovaný počet uživatelů 240 milionů, dostupné na `https://www.google.com/intl/cs/drive/`.
- **OneDrive** - úložiště společnosti Microsoft s více než 250 miliony uživateli, dostupné na `https://onedrive.live.com/about/cs-cz/`.
- **ownDrive** - úložiště sdružení CESNET pro akademickou sféru, dostupné na `https://du.cesnet.cz/cs/navody/owncloud/start`.

Zdroj dat: [For14]. Vybraná úložiště jsou mnohdy přímo svázána s uživatelským účtem mobilního zařízení či operačního systému, proto nelze považovat číselné údaje za reálné počty aktivních uživatelů.

Využitím cloudových úložišť a stahování souboru přímo z internetu odpadá nutnost nahrávat soubory do zařízení fyzickým připojením k počítači. Soubory je možné nahrávat takřka kdekoli. A rovněž je možné použít úložiště ke správě a verzování souborů dílčí aplikace. Celkově se tak zlepšuje pohodlí uživatele a přístupnost aplikace jako celku.

5.3.1 Google Drive

Platforma Android je jakožto produkt společnosti Google přímo svázána s uživatelským účtem zmíněné firmy. Google Drive je tak přirozeně podporován Androidovými zařízeními, neboť ta ke své funkci vyžadují přihlášení právě k onomu uživatelskému účtu společnému pro všechny služby společnosti, tedy i Google Drive. Podmínkou pro využívání služby je právě úspěšné spárování s účtem služeb Google Play, které jsou nativní součástí operačního systému Android.

Tohoto faktu bylo využito i při realizaci polymorfní aplikace. Ta při prvním spuštění uživatele požádá o přístup k uživatelskému účtu a následně umožní stahování souborů z cloudového úložiště. I když je služba Google Drive automaticky synchronizována na pozadí a umožňuje přímý přístup k souborům přímo ze sítě či z lokální cache zařízení, je u všech vybraných souborů uložena kopie souboru přímo do domovské složky aplikace.

To umožňuje jednotné zacházení se soubory, které jsou nezávisle na zdroji ukládány do společné složky pomocí asynchronní úlohy. Tím je zajištěno, že se ve složce nachází vždy nejaktuálnější verze souboru, kterou lze později využít při výběru položky z historie. Navíc nemůže dojít k problémům způsobeným

například výpadkem připojení při načítání dex archivu přímo ze síťového proudu dat.

5.3.2 Dropbox

Ač není služba Dropbox přímo svázána s uživatelským účtem zařízení tak, jako Google Drive, bývá často součástí zařízení vybraných výrobců hardware. Mimo to se jedná o cloudové úložiště s nejvyšším počtem uživatelů, a tak bylo logicky dalším cílem podpory v polymorfní aplikaci.

Služba Dropbox využívá odlišný systém přihlášení uživatele, kdy je do aplikace začleněn plug-in[12] *dropboxChooser*, který využívá aplikace již přítomné v zařízení k přihlášení a přenosu dat. Aplikace společnosti Dropbox se tak stává podmínkou pro využívání této služby, ale na druhou stranu řeší technickou stránku spojení s úložištěm, aniž by bylo nutné programovat jakékoli rozsáhlejší rozhraní v polymorfní aplikaci.

Při stahování souborů z úložiště Dropbox je v polymorfní aplikaci využito toho, že služba Dropbox poskytuje pro vybraný soubor URL odkaz platný několik hodin. Samotné stažení souboru pak probíhá zcela stejně jako stahování z jakéhokoli URL, ale odpadají problémy se zadáváním zdrojové adresy souboru.

[12]angl. pro zásuvný modul

6 Významné programové části

Tato kapitola si klade za cíl představit nedůležitější části polymorfní aplikace a jejich funkci. Pro bližší seznámení s kódem aplikace však doporučuji přímo prohlídku zdrojových kódů, které jsou součástí CD přiloženého k této práci. Následující odstavce mají sloužit spíše pro lepší přehled ve funkci aplikace, nikoli nahrazovat programovou dokumentaci, i když obsahují několik ukázek kódu převzatého přímo z polymorfní aplikace.

6.1 Balík activity

V tomto balíčku se nacházejí základní stavební prvky aplikace - dvě aktivity, které zajišťují její chod, správnou reakci prvků uživatelského rozhraní a práci s dílčími komponentami aplikace.

6.1.1 MainActivity

Jedná se o hlavní spouštěcí třídu aplikace, jejímž hlavním účelem je opatření souborů potřebných pro spuštění generované aplikace. Tato aktivita řeší přihlášení uživatele a napojení na účet Google Play. A z této třídy jsou spouštěny asynchronní úlohy pro stažení souborů.

Aktivita jako taková se stará o zpracování událostí vzniklých použitím jednotlivých prvků uživatelského rozhraní. S každým zdrojem souborů je asociováno tlačítko uživatelského rozhraní, které obsluhuje danou volbu. Příkladem je výběr souboru z Google drive na ukázce 6.1. Po získání potřebných souborů je možné spustit aktivitu ChildActivity.

```
/**
 * Otevře nabídku Google Drive.
 * @param v tlačítko Google Drive
 */
public void driveButtonAction(View v)
{
  IntentSender intentSender = Drive.DriveApi
      .newOpenFileActivityBuilder()
      .build(mGoogleApiClient);
  try {
```

```
    startIntentSenderForResult(
        intentSender, REQUEST_CODE_OPENER, null, 0, 0, 0);
  } catch (IntentSender.SendIntentException e) {
    Log.w("Google API", "Unable to send intent", e);
  }
}
```

Ukázka kódu 6.1: Výběr souboru z Google Drive

6.1.2 ChildActivity

Tato aktivita obaluje samotnou generovanou aplikaci. Důležitým atributem této aktivity je *ArrayList<View> content* - seznam s obsahy jednotlivých fragmentů. Po spuštění této aktivity se vytvoří instance třídy *XmlParser parser*, která provádí převedení prvků z textové podoby na objekty uživatelského rozhraní. Názvy souborů vybraných uživatelem v první aktivitě jsou předány pomocí sdílených preferencí programu, odkud jsou po vytvoření aktivity načteny.

Tato aktivita se rovněž stará o obsluhu událostí generovaných aktivními prvky uživatelského rozhraní. Po stisku tlačítka dochází k obsloužení jeho přidružené funkce metodou *processFunction* a případně k vytvoření asynchronní úlohy *ProcessFunctionTask* pro obsluhu této funkce, pokud je typ funkce *replace*. U takových funkcí se provedou následující kroky:

1. Vybrání všech objektů *View* v aktuálním fragmentu prohledáním odpovídající položky seznamu *content* metodou *getAllChildrenBFS*.
2. Zpracování obsahu všech položek typu *EditText* a *CheckBox* a vytvoření seznamu parametrů pro volanou funkci.
3. Vytvoření a spuštění asynchronní úlohy pro obsluhu dané funkce s odpovídajícími parametry.

U prvků typu *TextView* s aktivním obsahem (url, e-mailová adresa, telefonní číslo) pak zajišťuje spuštění správného *Intent* s odpovídajícím parametrem. Například předá jako parametr telefonní číslo a vytvoří metodou *startDialerIntent* odpovídající *Intent.ACTION_DIAL*. Pro ostatní typy obsahu funguje spouštění odpovídajících aplikací analogicky. Ukázka 6.2 představuje vyvolání e-mailového klienta.

```
/**
 * Spustí intent a otevře aplikaci asociovanou s
    elektronickou poštou.
 * @param address e-mailová adresa z TextView.
 */
private void startMailIntent(String address)
{
  String[] adresses = new String[]{address};
  Intent intent = new Intent(Intent.ACTION_SENDTO);
  // omezuje výběr na e-mail
  intent.setData(Uri.parse("mailto:"));
  intent.putExtra(Intent.EXTRA_EMAIL, adresses);
  if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
  }
}
```

Ukázka kódu 6.2: Intent mailové aplikace

Aktivita rovněž umožňuje dynamickou práci s fragmenty pomocí metod *duplicateFragment* a *removeFragment* dostupnými pomocí tlačítek v nástrojové liště. Po dokončení asynchronní úlohy vyhodnocuje navrácené výsledky a na základě jejich počtu buď nahradí obsah aktuálního fragmentu metodou *updateFragment* (viz ukázka 6.3), nebo vytvoří a přidá nové fragmenty pomocí metody *addFragment*.

```
/**
 * Použije předané View jako obsah aktuálního fragmentu.
 * @param fragmentContent budoucí obsah fragmentu
 */
private void updateFragment(View fragmentContent)
{
  if (fragmentContent == null) {
    return;
  }
  // aktuální položka
  int currentIndex = mViewPager.getCurrentItem();
  View v = content.get(currentIndex);

  String title =
     fragmentContent.getTag(R.id.title_key).toString();

  // asociace s původní šablonou
  fragmentContent.setTag(R.id.resource_key,
     v.getTag(R.id.resource_key));

  // aktualizace záložek
```

```
  ActionBar actionBar = getSupportActionBar();
  actionBar.getTabAt(currentIndex).setText(title);

  // aktualizace obsahu
  content.set(currentIndex, fragmentContent);
  updateImages();
  mSectionsPagerAdapter.notifyDataSetChanged();
}
```

Ukázka kódu 6.3: Nahrazení obsahu fragmentu

Při prvním načtení fragmentu, který obsahuje obrázky typu *ImageView* s přiřazeným URL, spouští tato aktivita asynchronní úlohu *DownloadImageTask*, která provede jejich stažení a přiřazení do odpovídajících prvků.

6.2 Balík fragment

Tento balík obsahuje dvě třídy zajišťující správnou práci s fragmenty. Jedná se o třídu *PlaceholderFragment*, která představuje samotný fragment, a třídu *SectionsPagerAdapter*, která se stará o správné vytváření záložek v nástrojovém panelu pomocí metody z ukázky 6.4.

```
/**
 * Načte správný titulek pro fragment.
 * @param position pozice fragmentu.
 * @return TITULEK
 */
@Override
public CharSequence getPageTitle(int position)
{
  Locale l = Locale.getDefault();
  String title = "";

  View v  = ChildActivity.content.get(position);
  title = v.getTag(R.id.title_key).toString();

  if(title.isEmpty())
  {
    title = "Fragment " + position;
  }

  return title.toUpperCase(l);
}
```

Ukázka kódu 6.4: Načtení titulku fragmentu

Vytvoření nového fragmentu probíhá tak, že si nově vytvořená instance vybere ze seznamu prvků *content* třídy *ChildActivity* položku na odpovídajícím indexu a převezme její obsah. Při načtení tohoto fragmentu dojde k vložení obsahu do layoutu fragmentu. Při přechodu na jiný fragment dojde k odstranění tohoto obsahu. Tím je zajištěno, že se v paměti neuchovává zbytečně mnoho načtených fragmentů, které nejsou potřeba. Veškerá práce s fragmenty je přitom obsluhována pomocí instance třídy *SectionsPagerAdapter* v aktivitě *ChildActivity*.

6.3 Balík task

Balík task obsahuje definice všech asynchronních úloh používaných při běhu aplikace. Všechny obsažené třídy dědí od třídy *AsyncTask* a liší se pouze funkcí, kterou vykonávají, a parametry, které používají.

6.3.1 IOnTaskCompleted

Jedná se o rozhraní, které musí implementovat aktivity, aby bylo možné spustit zpracování výsledku okamžitě po doběhnutí asynchronní úlohy. Doba vykonávání úlohy nikdy není předem známa. Použitím tohoto rozhraní lze ale vždy spolehlivě vyvolat obslužnou událost aktivity, jakmile je úloha dokončena.

6.3.2 CopyFileTask

Asynchronní úloha pro kopírování souboru v rámci paměti zařízení. Všechny soubory používané aplikací jsou uchovávány ve společné domovské složce v interní paměti zařízení, neboť načítání archivů dex z externí paměti není možné. Proto je nutné soubory z paměťové karty přesunout právě do této domovské složky.

Kopírovaný soubor je pomocí tříd *FileInputStream* a *FileOutputStream* rozdělen na bloky o velikosti 1024 byte a přesunut do odpovídajícího umístění. Po přečtení každého bloku je aktualizován *ProgressDialog* běžící na UI vláknu, zatímco kopírování souboru je vykonáváno na pozadí.

6.3.3 DownloadFileTask

Analogicky funguje asynchronní úloha pro stažení souboru ze zadaného URL. Rozdílná je v tom, že nejprve musí vytvořit spojení se vzdáleným umístěním pomocí *HttpURLConnection*. V případě úspěšného připojení je možné otevřít proud pro čtení souboru metodou *url.openStream()* a postupovat stejně jako při kopírování lokálního souboru.

6.3.4 DownloadImageTask

Úloha stahující obrázky pro vybrané prvky *ImageView* funguje obdobně jako úloha pro stažení jakéhokoli jiného souboru. Jediný rozdíl spočívá v tom, že načtení souboru je zde ponecháno přímo na systému, který požadovaný objekt typu *Bitmap* vytváří pomocí metody *BitmapFactory.decodeStream*. V tomto případě není možné určit aktuální fázi dokončení při přenosu souboru, takže jsou možnosti zobrazeného dialogu omezené. V ukázce 6.5 je uvedena část těla metody *doInBackground*, která definuje všechny asynchronní úlohy.

```
// převod adresy na URL
url = new URL(urlString);
// vytvoření spojení
connection = (HttpURLConnection) url.openConnection();
connection.connect();

// očekáváme HTTP 200 OK, v opačném případě by mohlo dojít
// ke stažení chybové zprávy místo souboru
if (connection.getResponseCode() !=
    HttpURLConnection.HTTP_OK)
{
  errorString = "Server returned HTTP " +
     connection.getResponseCode()
      + " " + connection.getResponseMessage();
  return null;
}

// vytvoření bitmapy
InputStream is = connection.getInputStream();
bm = BitmapFactory.decodeStream(is);
```

Ukázka kódu 6.5: Stažení obrázku z URL

Objekt *Bitmap* je uložen v cache zařízení jako dočasný soubor. Protože není možné z vlákna běžícího na pozadí upravovat prvky uživatelského rozhraní vytvořené UI vláknem, je získaná instance objektu *Bitmap* předána ke zpracování aktivitě *ChildActivity*, která se stará o správné přiřazení do prvku *ImageView*.

6.3.5 DriveFileTask

Tato úloha má za úkol opatřit lokální kopii souboru umístěného v úložišti Google Drive. Z dialogu služby Google Drive je výběrem souboru získáno jeho *DriveId*. Podle tohoto údaje je možné vytvořit instanci objektu *DriveFile* a pomocí několika metod služeb Google Drive získat i jeho základní metadata - název a velikost. Pokud je spojení se službou Google Drive úspěšně otevřené, lze vytvořit instanci objektu *DriveContents*, ze které je dále možné získat proudový vstup a zkopírovat soubor jako při normálním stahování.

6.3.6 ProcessFunctionTask

ProcessFunctionTask je úloha sloužící ke spuštění funkce z archivu typu dex. Pomocí nástroje *DexClassLoader* je možné získat instanci cílové třídy z připraveného archivu. Následně je pomocí reflexe zavolána metoda *doLibraryFunction* instance této třídy. Tento postup je popsán již v kapitole 5.2.2.

Nevýhodou takového volání metody je fakt, že není možné určit, kolik zbývá času k dokončení metody. Podle výkonu zařízení a rychlosti připojení k internetu se tak může doba potřebná k provedení operace velmi lišit.

6.4 Balík type

Obsahem tohoto balíku je třída *Function*, která slouží pouze jako objekt pro uložení informací o funkci, která má být volána po stisknutí tlačítka, kterému je přiřazena. Oddělení funkce jako samostatného objektu umožňuje použití vyšší míry abstrakce a jednodušší předávání parametrů v průběhu zpracování funkce.

Dalším prvkem balíku je rozhraní *ILibraryFunction*, které sice není v po-

lymorfní aplikaci přímo využíváno, jeho implementace je však vyžadována u dílčích funkcí generované aplikace, aby bylo možné je spouštět jednotným přístupem popsaným v části 5.2.2.

6.5 Balík util

Součástí tohoto balíčku jsou třídy, které přímo nezasahují do chodu aplikace, nicméně poskytují nástroje a metody nutně potřebné pro její správnou funkci.

6.5.1 Utilities

Tato třída se stará zejména o práci se soubory na úrovni kontroly jejich existence, vytváření potřebných adresářů a jejich případné odstranění. Další důležitou funkcí třídy *Utilities* je práce se *shared preferences*, které představují způsob, jak ukládat data aplikace pro pozdější použití - například historii načtených souborů a další údaje. Ukázka kódu 6.6 představuje metodu, která provádí uložení jedné hodnoty do seznamu řetězců v *shared preferences*.

```
/**
 * Přidá uvedenou hodnotu do odpovídajícího seznamu v shared
    Preferences.
 * @param context Kontext volající aplikace
 * @param key Klíč pro výběr seznamu
 * @param value Hodnota pro přidání
 */
public static void writeToSet(Context context, String key,
   String value)
{
  Set<String> values = new HashSet<String>();
  Set<String> data =
     getPreferences(context).getStringSet(key, values);
  values.addAll(data);
  values.add(value);

  getEditor(context).putStringSet(key, values).commit();
}
```

Ukázka kódu 6.6: Zápis do shared preferences

Použitím této třídy je zajištěno, že ani jedna aktivita nemusí znát přesné umístění domovské složky v souborovém systému zařízení, aby mohla pra-

covat se soubory. Významnou metodou je pak například metoda *getPath*, která převádí obecný identifikátor umístění souboru na zařízení Android na skutečnou cestu k tomuto souboru.

V této třídě je rovněž umístěna metoda pro zjištění aktuálního stavu připojení k síti *isOnline* uvedená na ukázce 6.7.

```
/**
 * Zjistí, zda je zařízení připojeno k síti.
 * @param context Kontext volající aktivity.
 * @return true pro aktivní připojení, jinak false
 */
private boolean isOnline(Context context)
{
  try
  {
    ConnectivityManager cm =
        (ConnectivityManager)context.getSystemService(
        Context.CONNECTIVITY_SERVICE);
    return
        cm.getActiveNetworkInfo().isConnectedOrConnecting();
  }
  catch (Exception e)
  {
    return false;
  }
}
```

Ukázka kódu 6.7: Ověření připojení k síti

6.5.2 XmlParser

XmlParser je třída klíčová pro zpracování všech XML šablon a jejich převedení na prvky grafického rozhraní. Samotné prvky je nutné vytvářet v UI vlákně, které je posléze ovládá. Proto jsou tvořeny v rámci kontextu *Child Activity*, která parser instancuje. Průběh vytváření grafického rozhraní generované aplikace je popsán již v části 5.1. Z hlediska třídy *XmlParser* se skládá z následujících kroků:

1. Parsování vstupního souboru či stringového vstupu pomocí instance třídy *DocumentBuilder*.

2. Rozdělení na jednotlivé fragmenty, pokud je jich obsaženo více.

3. Vytvoření obalujícího *LinearLayout*, do kterého se všechny další prvky vybraného fragmentu přidají, metodou *parseFragmentContent*.
4. Procházení struktury prvků do hloubky pomocí rekurzivní metody *parseNode*.
5. Převedení vybraného elementu šablony na odpovídající prvek uživatelského rozhraní a přiřazení všech atributů.

Následující ukázka 6.8 představuje metodu pro převedení jednoho uzlu šablony na prvek uživatelského rozhraní (viz bod 4 v předcházejícím seznamu).

```
/**
 * Převede vybraný uzel šablony na prvek rozhraní.
 * @param node prvek šablony
 * @return prvek uživatelského rozhraní
 */
private View parseNode(Node node)
{
  View v = null;

  Element ele = (Element) node;
  String nodename = ele.getTagName();

  switch (nodename)
  {
    case ("linearlayout"):
      v = parseLinearLayout(ele);
      break;
    case ("button"):
      v = parseButton(ele);
      break;
    case ("textview"):
      v = parseTextView(ele);
      break;
    case ("edittext"):
      v = parseEditText(ele);
      break;
    case ("checkbox"):
      v = parseCheckBox(ele);
      break;
    case ("imageview"):
      v = parseImageView(ele);
      break;
    default:
      v = parseTextView(ele);
  }
```

```
  String weightValue = ele.getAttribute("weight");
  LinearLayout.LayoutParams defaultLP = new
     LinearLayout.LayoutParams(
     LinearLayout.LayoutParams.MATCH_PARENT,
      LinearLayout.LayoutParams.MATCH_PARENT,
         parseWeight(weightValue));

  v.setLayoutParams(defaultLP);

  return v;
}
```

Ukázka kódu 6.8: Parsování uzlů šablony

Poslední ukázka této kapitoly 6.9 představuje vytvoření prvku uživatelského rozhraní z elementu šablony. Jedná se o metodu volanou z kódu v ukázce 6.8. Vytváření ostatních prvků uživatelského rozhraní probíhá analogicky, liší se jen použité atributy elementů, které popisuje příloha C.

```
/**
 * Vytvoří z elementu šablony prvek uživatelského rozhraní
    EditText
 * @param e vybraný element šablony
 * @return Vytvořená instance objektu EditText
 */
private EditText parseEditText(Element e)
{
  EditText et = new EditText(appContext);

  et.setTextSize(TypedValue.COMPLEX_UNIT_PX,
      appContext.getResources().getDimension(
         R.dimen.abc_text_size_medium_material));

  et.setHint(e.getAttribute("hint").trim());
  et.setTag(e.getAttribute("tag"));
  String inputTypeValue = e.getAttribute("type");
  switch (inputTypeValue)
  {
    case ("datetime"):
      et.setInputType(InputType.TYPE_CLASS_DATETIME);
      break;
    case ("number"):
      et.setInputType(InputType.TYPE_CLASS_NUMBER);
      break;
    case ("phone"):
      et.setInputType(InputType.TYPE_CLASS_PHONE);
      break;
    case ("multiline"):
```

```
        et.setSingleLine(false);
        et.setHorizontalScrollBarEnabled(false);
        break;
      case ("text"):
      default:
        et.setInputType(InputType.TYPE_CLASS_TEXT);
    }

    return et;
}
```

Ukázka kódu 6.9: Vytvoření prvku EditText

Při načítání šablony ze souboru se každý načtený fragment ukládá do struktury *Map<String, ArrayList<Node> > fragmentSources*, ve které je asociován svým titulkem. V případě, že je třeba duplikovat vybraný fragment uživatelského rozhraní, dojde k výběru předpřipraveného obsahu přímo z této zdrojové mapy.

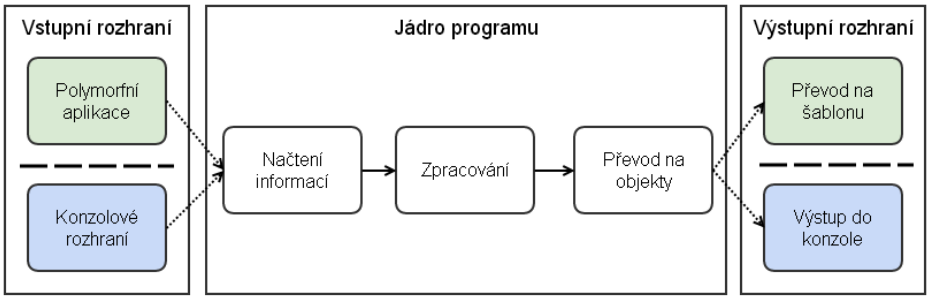
Toto řešení je zvoleno z důvodu, že samotné prvky uživatelského rozhraní nejde jednoduše duplikovat, pokud jsou již vytvořeny v kontextu aktivity *ChildActivity*. Načtením prvků z mapy tak lze efektivně a rychle provést duplikaci fragmentu se zcela novými prvky, což je důležité hlavně pro fragmenty, u kterých již byl původní obsah nahrazen výstupem z volané funkce generované aplikace.

7 Ukázkové aplikace

Funkcionalitu polymorfní aplikace je možné nejlépe prezentovat na sadě generovaných aplikací, které pod polymorfní aplikací běží. Jedná se o aplikace, které byly vytvořeny nezávisle, jako samostatné projekty v jazyce Java, ve vývojovém prostředí Eclipse[1].

Aplikace byly záměrně vytvořeny odděleně od mateřské aplikace, bez použití jakýchkoli funkcí či prvků platformy Android, aby bylo ukázáno, že se jedná o samostatně běžící řešení, které lze do polymorfní aplikace jednoduše integrovat.

Vzhledem ke způsobu, kterým aplikace vznikly, je lze spustit použít dvěma způsoby znázorněnými na obrázku 7.1.



Obrázek 7.1: Spuštění ukázkových aplikací

1. Jako standardní desktopové aplikace s rozhraním vývojového prostředí, případně konzolovým rozhraním. Aplikace v takovém případě načítá jen nezbytně nutné parametry a výstupem je formátovaný text do konzolového rozhraní.

2. Jako jádro generované mobilní aplikace, které je spuštěno z uživatelského rozhraní odpovídající šablony. Program se provede v rámci asynchronní úlohy polymorfní aplikace, vstupem je seznam textů editovatelných prvků rozhraní a seznam hodnot nastavených pomocí CheckBoxů v rozhraní. Výstupem je pak jedna nebo více šablon vygenerovaných z nalezených výsledků.

[1]Použitá verze: Eclipse Standard, Kepler Service Release 1, Build id: 20130919-0819

7.1 Omezení a požadavky

Pokud nemá uživatel k dispozici již hotový dex archiv s výkonným kódem aplikace, je třeba před použitím aplikace v mobilním zařízení aplikaci nejprve překompilovat do požadovaného formátu. Důvody pro tento krok popisuje kapitola 5.2. Přesný postup pak popisuje příloha B.3.

Kvůli omezením daným kompilací kódu do dex archivu není možné používat v rámci dílčí aplikace knihovny třetích stran. Respektive není možné zaručit, že ve výsledném dex archivu budou obsaženy veškeré komponenty nutné pro správný chod programu. V případě, že dílčí aplikace knihovnu třetích stran využívá, je lepší přiložit soubor knihovny do projektu polymorfní aplikace dle návodu v příloze B.4, a to i v případě, že samotná polymorfní aplikace žádnou funkci dané knihovny nevyužívá.

Uvedené aplikace pracují na bázi hledání odpovídajících informací na internetu, takže je pro jejich běh vyžadováno aktivní připojení k síti v mobilní i desktopové podobě. Je rovněž třeba dbát na to, aby vytvořená aplikace nepřekračovala systémová povolení polymorfní aplikace popsaná v příloze A.

7.2 RSS Čtečka

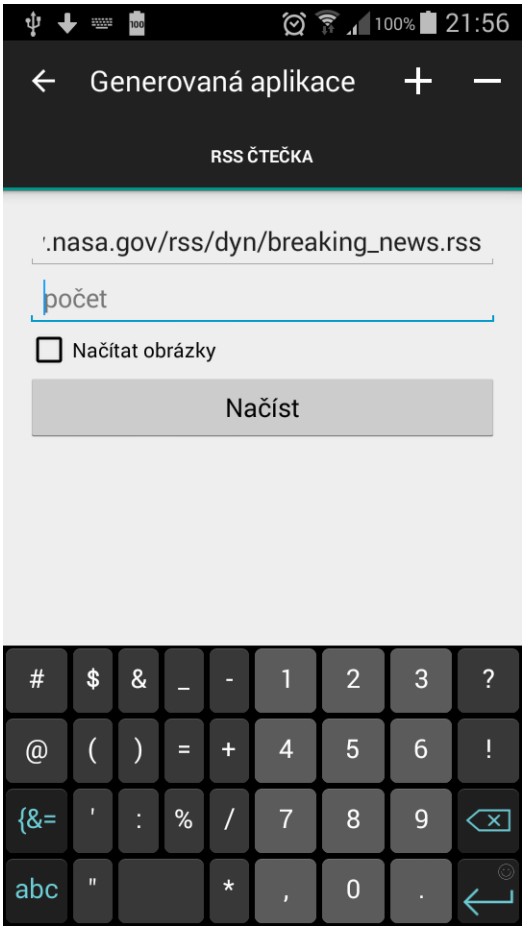
RSS[2] je podskupinou jazyka XML. Původně bylo vytvořeno jako formát pro výměnu informací mezi počítačovými servery, ale v současné době je všeobecně využíváno jako kanál pro zasílání novinek - zpráv, zveřejněných článků atd.

Základem čtení RSS zpráv je připojení se k vybranému kanálu, identifikovanému odpovídajícím URL. V rámci tohoto kanálu (*channel*) jsou pak postupně zveřejňovány jednotlivé příspěvky (*item*). V ukázkové aplikaci byla použita specifikace RSS 2.0.1 popsaná v [Rss09] jako nejuniverzálnější struktura zdroje zpráv, i když v současnosti nabývají na významu novější řešení, například Atom definovaný v [Not05]. Řešení RSS 2 je však nejrozšířenější zejména z historických důvodů.

Cílem aplikace je pak zpřístupnění novinek z vybraných kanálů ve formě čitelné na displeji mobilního zařízení. Uživatel zadává do aplikace pouze dva

[2]Really Simple Syndication - angl. pro Velmi jednoduchá syndikace obsahu

údaje - adresu zdroje a počet příspěvků, které se mají načíst. Dále je možné specifikovat, zda se má pro načtené příspěvky stahovat i obrazový materiál z přílohy příspěvku. Úvodní obrazovku generované aplikace představuje obrázek 7.2.



Obrázek 7.2: Úvodní obrazovka RSS čtečky

Na obrázku 7.2 je také patrné, že u položek typu EditText lze vybrat jeden z typů vstupu, kterému se pak přizpůsobí rozložení klávesnice na zařízení. Více o definici prvků uživatelského rozhraní se lze dočíst v příloze C.

Po načtení vybraného zdroje proběhne jeho parsování pomocí DOM parseru. Původně byl použit StAX[3] parser inspirovaný řešením uvedeným ve [Vog13]. Bohužel zde dochází k tomu problému, že třídy balíku *javax.xml stream.events* nejsou implementovány v základních prostředcích platformy

[3]Streaming API for XML - angl. pro Rozhraní pro čtení XML proudů.

Android, ač se jedná o zcela běžně používaný prostředek standardní verze jazyka Java.

Namísto integrace potřebných knihoven do polymorfní aplikace tak bylo zvoleno náhradní řešení a parser RSS zdroje byl kompletně předělán. Pro úplnost ale uvádím, že platforma Android nemusí podporovat ani některé funkce jazyka Java verze 7, pokud jsou ve třídách načítaných pomocí DexClassLoaderu. A je tak lepší se novým prvkům jazyka Java raději vyhnout, například příkaz *switch* s parametrem typu *String* nemusí být vždy bezpečný, ačkoli ho využívá samotná polymorfní aplikace.

```
@Override
public String toString() {
  return "Feed [copyright=" + copyright + ", description=" +
      description
       + ", language=" + language + ", link=" + link + ",
          pubDate="
       + pubDate + ", title=" + title + ", lastBuildDate=" +
          lastBuildDate + "]";
}
```

Ukázka kódu 7.1: Atributy objektu Feed

Po zpracování zdrojového souboru parserem probíhá převedení na odpovídající objekty - *Feed* a *FeedMessage*. Jednotlivé atributy objektů jsou ukládány v podobě řetězce a jsou pojmenovány stejnými názvy jako v definici RSS zdrojů. Proto namísto výčtu uvádím pouze metody výpisu objektů na konzoli, které jsou dostatečně názorné.

```
@Override
public String toString() {
  return "FeedMessage [title=" + title + ", description=" +
      description
       + ", link=" + link + ", author=" + author + ", guid="
          + guid  + ", pubDate=" + pubDate
       + ", enclosureType=" + enclosureType + ",
          enclosureURL=" + enclosureURL + "]";
}
```

Ukázka kódu 7.2: Atributy objektu FeedMessage

Jednotlivé položky FeedMessage mohou ve zdrojovém souboru obsahovat prvek *enclosure*, který značí přítomnost přidaného obsahu ve formě souboru.

Například:

```
<enclosure url="http://www.scripting.com/mp3s/\\
weatherReportSuite.mp3" length="12216320" type="audio/mpeg"
   />
```

Ukázka kódu 7.3: Enclosure - zvukový soubor

Pokud uživatel vybere možnost stahování obrázků v RSS zprávách a pokud zpráva obsahuje přílohu typu *image*, uloží se zdrojové URL obrázku do objektu FeedMessage. Jelikož ale modul aplikace neřeší práci se souborovým systémem, není možné obrázek rovnou uložit. Nahrávání obrázků probíhá až při prvním zobrazení daného prvku v cílovém fragmentu aplikace.

Aby bylo možné stažení provést, je třeba předat aplikaci URL v bezpečné podobě, která nenaruší strukturu výsledné XML šablony. K převedení obvykle slouží vybraná metoda balíku *java.net*, nicméně ta byla nahrazena vlastním řešením, aby bylo možné se vyhnout jakékoli závislosti na externích knihovnách.

```
/*
 * Převede URI na bezpečný řetězec
 * @param s původní řetězec
 * @return převedený řetězec
 */
private String encodeURIcomponent(String s) {
  StringBuilder o = new StringBuilder();
  for (char ch : s.toCharArray()) {
    if (isUnsafe(ch)) {
      o.append('\%');
      o.append(toHex(ch / 16));
      o.append(toHex(ch \% 16));
    } else
      o.append(ch);
  }
  return o.toString();
}

/**
 * Vrací hexa decimální hodnotu znaku.
 * @param ch znak k převedení
 * @return hexadecimální hodnota
 */
private char toHex(int ch) {
  return (char) (ch < 10 ? '0' + ch : 'A' + ch - 10);
}

/**
```

```
 * Kontroluje bezpečnost znaků
 * @param ch zkoumaný znak
 * @return true pro speciální znaky, false pro ostatní
 */
private boolean isUnsafe(char ch) {
  if (ch > 128 || ch < 0)
    return true;
  return " %$&+,/:;=?@<>#'`\"\\".indexOf(ch) >= 0;
}
```

Ukázka kódu 7.4: Převedení URL na bezpečný řetězec

Stejný postup je použit pro samotný odkaz na plnou verzi zprávy, který se též vkládá do výsledné šablony. Pomocí použití různých typů textu pro jednotlivé prvky TextView je dosaženo toho, že po kliknutí na uvedený odkaz bude otevřen daný článek přímo v prohlížeči zařízení. Pokud se tedy uživatel aplikace chce dozvědět více informací či zobrazit obsah, který byl v rámci aplikace vynechán (videa, zvukové soubory, HTML elementy v popisu zprávy).

Výsledkem činnosti dílčí aplikace RSS Čtečka je pak šablona se zadaným počtem prvků, kde je každý prvek zobrazen ve formátu: obrázek, **nadpis**, *datum publikování*, popis zprávy, *zvýrazněný odkaz na celý text*. Snímky výsledků čtečky přímo ze zařízení jsou přiložené v příloze E.3.

7.3 Kontakty ZČU

Další ukázková aplikace slouží k rychlému nalezení kontaktních informací o pracovnících Západočeské univerzity v Plzni. Využívá k tomu infromací získaných z profilů zaměstnanců na stránkách `http://www.zcu.cz/media/about/people/`, které upraví do podoby vhodné pro mobilní zařízení.

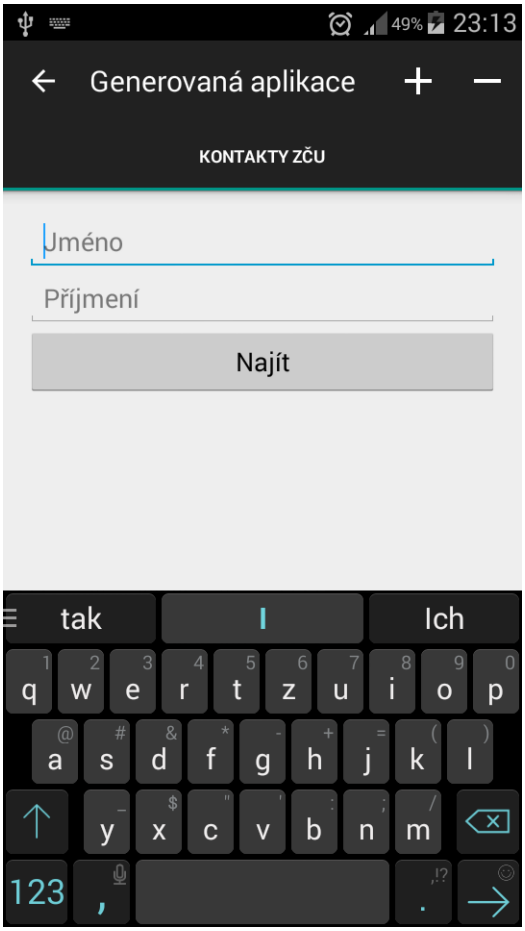
Jednotlivé informace jsou získávány přímo z webových stránek pomocí nástroje jsoup[4]. Ten umožňuje výběr elementů jazyka HTML pomocí jejich unikátních CSS selektorů. V případě jednotné struktury „vizitek“ jednotlivých zaměstnanců je tedy možné vytvořit parser na míru všem stránkám obsahujícím informace o zaměstnancích ZČU.

Jelikož se jedná o knihovnu třetích stran používanou v dílčí aplikaci, bylo

[4]více informací na `http://jsoup.org/`

nutné ji zavést přímo do mateřské aplikace jako knihovnu, protože nebylo možné ji přidat do archivu pomocí nástroje dx. Po přidání této knihovny je však aplikace připravena na parsování jakékoli webové stránky, stačí pouze přizpůsobit pravidla parseru dané stránce.

Vyhledávání kontaktů probíhá dle zadaného jména a příjmení hledané osoby. Ani jeden z parametrů není povinný, nicméně nepřesný dotaz povede v absolutní většině případů k nepřesným výsledkům. Nedoporučuje se tedy zadávání příliš obecných dotazů typu: <Petr, "_"> nebo <"_", Novák>. Ukázka vstupu aplikace je na obrázku 7.3. Prvky typu CheckBox nejsou v této aplikaci použity, a proto je může výkonný modul ignorovat.



Obrázek 7.3: Úvodní obrazovka aplikace Kontakty ZČU

Po zpracování vstupních údajů program přistoupí pomocí knihovny jsoup na základní stránku `http://www.zcu.cz/media/about/people/`, odkud vyextrahuje všechny odkazy na jednotlivé stránky pracovníků ZČU. Z těchto

odkazů vybere takové stránky, u kterých se shoduje příjmení osoby se zadaným údajem, a dále je vyfiltruje podle zadaného jména.

Dalším krokem je postupný přístup na všechny odpovídající stránky zaměstnanců, ze kterých jsou získány potřebné údaje. Ty se použijí pro vytvoření instance objektu *Person*, která má stejně jako v u aplikace RSS Čtečky všechny atributy typu String, a tak bude popsána pouze pomocí metody *toString* v ukázce 7.5.

```
@Override
public String toString() {
  StringBuilder builder = new StringBuilder();

  builder.append("Jméno: " + name + "\n");
  builder.append("Telefon: " + phone + "\n");
  builder.append("Mail: " + mail + "\n");
  builder.append("Místnost: " + room + "\n");
  builder.append("Pracoviště: " + department + "\n");
  builder.append("Fotka: " + photo + "\n");

  return builder.toString();
}
```

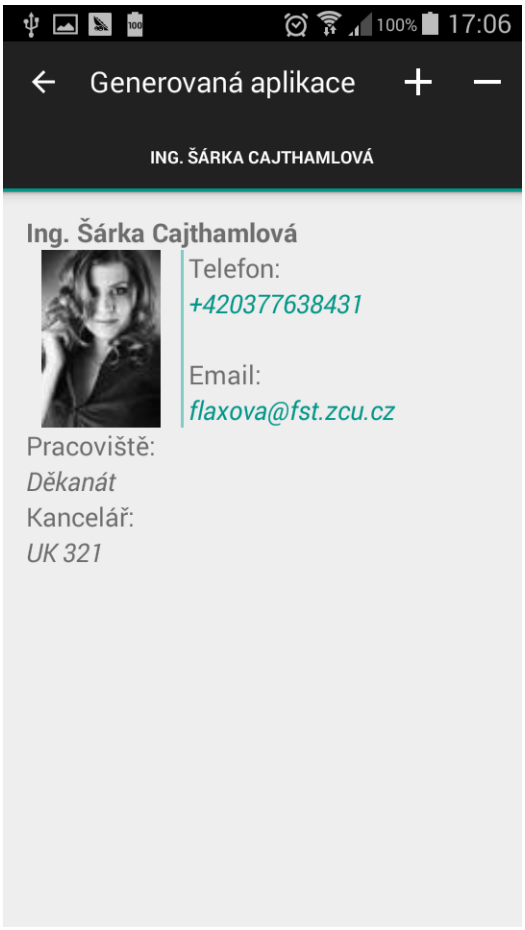
Ukázka kódu 7.5: Atributy objektu Person

Pro každou instanci objektu *Person* je následně vytvořena vlastní šablona pro polymorfní aplikaci. Aby nedošlo k zahlcení polymorfní aplikace nově vytvořenými fragmenty, je počet vrácených výsledků omezen na pět. I tak je kladen důraz na přesnost dotazu zadaného uživatelem. Struktura šablon pro vstup i výstup aplikace je popsána v příloze D.2.

Na obrázku 7.4 je snímek obrazovky výstupu aplikace pro jeden nalezený výsledek. Zatímco údaje o pracovním zařazení zaměstnance a jeho pracovišti slouží jen pro čtení, údaje s telefonním číslem a e-mailovou adresou jsou interaktivní. Po kliknutí na telefonní číslo dojde k otevření aplikace mobilního zařízení určené pro hovory - tzv. *dialer*u[5].

Následná akce je pak už jen na uživateli. Je možné dané číslo vytočit, uložit do kontaktů atd. Tím, že aplikace neprovádí přímo volání na dané číslo, je zajištěna také větší bezpečnost, neboť je zaručeno, že hovor může uskutečnit pouze uživatel sám, nikoli aplikace bez jeho vědomí. I proto není potřeba povolení pro uskutečňování hovorů, viz. příloha A.

[5]angl. pro vytáčení



Obrázek 7.4: Vizitka zaměstnance ZČU

Po kliknutí na e-mailovou adresu zaměstnance dojde analogicky k otevření nové zprávy ve výchozí aplikaci pro vyřizování elektronické pošty. Předmět a text zprávy je opět ponechán pouze na uživateli, stejně jako její finální (ne)odeslání.

8 Testování aplikace

Aplikace byla vyvíjena v prostředí Android Studio a testována na zařízení **Samsung Galaxy S4 mini** na verzi systému **4.4.2** (API 19), které je ve vlastnictví autora. Cílovou skupinou použitých zařízení byly mobilní telefony, takže rozložení obrazovky na tabletech či jiných zařízeních nemusí odpovídat snímkům obrazovky uvedeným v této práci a v příloze E.

Obecně by mělo být možné aplikaci provozovat na všech zařízeních s verzí systému 4.0.3 (API 15) a novější, což znamená funkcinalitu na 89% zařízení na trhu, jak uvádí [Apf15]. Aplikace byla kompilována cílovým API 22, tedy nejaktuálnějším v době vývoje.

8.1 Použitá zařízení

Protože jsou některé funkce aplikace, zejména práce s uživatelskými účty a spolupráce s aplikacemi třetích stran, v rámci emulátoru Android nedostupné, bylo možné v rámci emulátoru otestovat pouze základní funkce aplikace. Na verzích emulátoru se systémem verze 4.0.3. (API 15) a 5.1.1 (API 22) byla aplikace nainstalována i spuštěna úspěšně. Pro praktické testy ukázkových aplikací byla použita následující zařízení[1]:

- Samsung Galaxy Fame, verze systému 4.1.2 (API 16), úhlopříčka displeje 3,5", rozlišení $320 * 480$ bodů;
- Samsung Galaxy S3 mini, verze systému 4.2.2 (API 17), úhlopříčka displeje 4,0", rozlišení $480 * 800$ bodů;
- Samsung Galaxy S4 mini, verze systému 4.4.2 (API 19), úhlopříčka displeje 4,3", rozlišení $480 * 960$ bodů;
- Samsung GalaxyTab 4, verze systému 4.4.2 (API 19), úhlopříčka displeje 10,1", rozlišení $1280 * 800$ bodů.

Na všech uvedených zařízeních proběhla úspěšně instalace aplikace, spuštění aplikace, provedení ukázkových aplikací a odinstalace aplikace. Bohužel

[1] Vývoj aplikace nebyl sponzorován či jinak podporován společností Samsung.

nebylo možné do odevzdání této práce úspěšně ověřit plnou funkcionalitu na zařízení s verzí systému Andoid 5.0 (API 21) či Android 5.1 (API 22).

8.2 Aplikace třetích stran

Aby bylo možné používat některé funkce pro výběr souborů generovaných aplikací, jsou nutné následující předpoklady:

- **Připojení k internetu** - pro stahování souborů a úspěšný provoz ukázkových aplikací. Testování proběhlo jak na Wi-fi připojení, tak na mobilním internetu 4G úspěšně. Při použití mobilního 3G připojení byla citelným nedostatkem rychlost připojení, přesto proběhlo testování úspěšně.
- **Spárování s účtem Google** - bez použití účtu nejde využívat služeb Google Drive. Testování proběhlo úspěšně na třech různých uživatelských účtech.
- **Aplikace Dropbox** - klientská aplikace služby Dropbox obsluhuje výběr souborů z tohoto úložiště. Testování proběhlo úspěšně na dvou různých uživatelských účtech. V případě, že není aplikace nainstalována, je uživatel vyzván k její instalaci, která je dobrovolná, avšak podmiňující pro použití služeb Dropbox.
- **Správce souborů** - aplikace nutná pro výběr souborů z lokální paměti zařízení. Testování proběhlo úspěšně s aplikacemi *ES File Explorer, Total Commander a Správce souborů společnosti Cheetah Mobile*. V případě, že není nainstalována žádná aplikace pro práci se soubory, může nabídka pro výběr lokálně uloženého souboru obsahovat i služby *Drop box*, *Google Drive (Disk)* či jiné aplikace. Obecně by mělo být možné použít libovolnou aplikaci pro správu souborů systému Android.

8.3 Publikace aplikace

V době vydání této práce není aplikace veřejně dostupná na oficiálním obchodu `https://play.google.com/store` či jiných aplikačních kanálech. Je-

diný způsob, jak aplikaci nainstalovat, je použití souboru z CD přiloženého k této práci. Jak instalaci provést, popisuje příloha B.1.

Aplikace je přiložena v debug verzi, neboť nástroje Google Developers Console neumožňují současné využívání služeb Google Drive více verzemi téže aplikace. Po vydání release verze aplikace by bylo nutné změnit ověřovací SHA1[2] kód aplikace, což by znemožnilo využití uživatelského účtu vývojovou verzí aplikace.

Obdobně služba Dropbox je omezena počtem 100 uživatelů, kteří se smějí pomocí aplikace připojit. Po překročení tohoto počtu je nutné změnit typ spolupráce se službou Dropbox na produkční typ.

Před publikováním aplikace by bylo nutné provést rozsáhlé testování a další kroky potřebné pro uveřejnění aplikace. Také by bylo třeba vzít v potaz samotnou funkcionalitu aplikace, která zjednodušeně umožňuje spouštění libovolného kódu na zařízení platformy Android. V případě uveřejnění by bylo nutné například vytvořit právně platné zřeknutí se zodpovědnosti za akce uživatele, který si například může pomocí generované aplikace odstranit ze zařízení důležité soubory. A obecně by mohla nejasná funkcionalita aplikace být problémem při schvalovacím procesu v rámci obchodu Google Play.

[2]druh 160 bitového kódování

9 Možnosti dalšího rozšíření

Na budoucí rozšíření polymorfní aplikace lze nahlížet z několika úhlů pohledu: přidání nových grafických prvků, rozšíření funkcionality stávajících prvků, začlenění nových typů funkcí, přidání možností nových vstupů a výstupů generované aplikace atp. Z hlediska praktické použitelnosti bych ale rád rozebral dva typy změn, které by mohly významně vylepšit používání polymorfní aplikace.

Následující možnosti zdokonalení polymorfní aplikace vycházejí mimo jiného i z nerealizovaných projektů generovaných aplikací, které nebylo možné vytvořit z časových důvodů či přílišné odlišnosti návrhu od aktuálního řešení, případně projektů, které byly zavrženy již ve stádiu návrhu.

9.1 Návaznost funkcí

Navázáním výstupu jedné funkce generované aplikace na vstup následující funkce by bylo možné dosáhnout komplexního řešení pro vybrané specializované úlohy jako dotazníky, kvízy či jednoduché logické hry, u které by se v každé následující úrovni přizpůsobila obtížnost problému uživateli podle toho, jak rychle a úspěšně vyřešil předchozí problém.

V současné podobě umožňuje aplikace pouze dva typy akcí:

1. Zadání parametrů a zavolání přiřazené funkce.
2. Zobrazení získaných výsledků.

Pro komplexní funkci generované aplikace by se však dal vytvořit takový návrh, kde je výše uvedený proces pouze jedním z několika kroků při používání aplikace. Generovaná aplikace by jako výsledek vracela rozhraní obsahující odlišná data a volající rozdílné funkce, například na základě voleb uživatele v předchozím zobrazení. Takovou aplikaci by bylo možné využít například jako formu dotazníku. Uživateli by se na základě jeho odpovědí zobrazovaly další dotazy a po krátké sérii otázek by dospěl k závěrečnému vyhodnocení. Případně by takový proces mohl vést k personalizaci jiné, rozsáhlejší aplikace, jejíž součástí by byla polymorfní aplikace.

9.2 Zpracování více fragmentů

Aktuální řešení polymorfní aplikace používá jako vstupní parametry funkce pouze data z jednoho, aktuálně vybraného fragmentu. Kombinace dat z několika otevřených fragmentů by umožnila použití komplexnějších funkcí pro agregaci dat.

Využití takové aplikace by mohlo být velmi různorodé. Jedním z příkladů by mohlo být zaslání hromadného e-mailu všem kontaktům právě otevřeným v záložkách jakožto rozšíření současné ukázkové aplikace popsané v části 7.3.

Jedním z nerealizovaných projektů byla i jednoduchá aplikace pro vytváření nákupního seznamu. Aplikace měla načítat seznam surovin potřebných pro uvaření pokrmu dle vybraného receptu na některém z několika webů. V případě jednoho vybraného receptu by došlo pomocí knihovny jsoup k extrakci releventních dat ve formě seznamu potřebného množství všech surovin nutných pro přípravu daného jídla. V případě více vybraných receptů by došlo ke kombinaci jednotlivých receptů za účelem vytvoření jednoho nákupního seznamu. V ideálním případě by pak bylo možné takový seznam exportovat jako poznámku externí aplikace.

Obdobným způsobem by bylo možné kombinovat například rozvrhy několika žáků za účelem nalezení průniku volných časů, či k sloučení jiného typu dat. Samozřejmě by se daly uvedené postupy použít k vytvoření mnohem sofistikovanějších aplikací bez nutnosti vytvářet samostatně běžící jednoúčelovou aplikaci.

10 Závěr

Úkolem této práce bylo porovnat možnosti tvorby polymorfních aplikací pro platformu Android. V rámci tohoto porovnání představila práce největší výhody a slabiny webových a multiplatformních řešení v porovnání s polymorfní aplikací vyvinutou výhradně pro operační systém Android.

Tato práce rozšiřuje již známý koncept polymorfní aplikace používající soubory XML pro definici uživatelských rozhraní o nové prvky, které umožňují zdůraznit význam a obsah zobrazovaných informací. Dále přináší možnost dynamického vytváření fragmentů v rámci generované aplikace, jejich duplikaci, přidávání nebo nahrazení původního obsahu nově získanou informací. Přesto je hlavním cílem aplikace vytvořené v této práci nabídnout jednoduchý nástroj jak pro uživatele mobilního zařízení, tak pro vývojáře mobilních aplikací. Jedním z účelů polymorfní aplikace je i snadné převedení klasického programu jazyka Java do mobilní podoby. Tento postup je rovněž v práci popsán.

Polymorfní aplikace umožňuje výběr souborů pro definici uživatelského rozhraní a knihoven s výkonným kódem programu generované aplikace z několika různých zdrojů - z paměti zařízení, webového URL, ale i z cloudových úložišť Dropbox a Google Drive. Uživatel tak má možnost výběru zdroje dle aktuální potřeby a rovněž je tak usnadněn přenos potřebných souborů do zařízení, jejich spravování a zálohování a v neposlední řadě i sdílení. Uvedené cloudové služby mohou v budoucnu sloužit i jako jeden z distribučních kanálů pro dílčí programy spouštěné v polymorfní aplikaci.

Možnosti vytvořeného řešení jsou prezentovány na dvou ukázkových aplikacích - čtečce RSS a vyhledávači kontaktních informací pracovníků Západočeské univerzity v Plzni. Tyto aplikace mají za cíl jednak předvést způsob práce s polymorfní aplikací, a pak mají sloužit jako vzor pro programátory, kteří by se rozhodli koncept polymorfní aplikace využít ve svých projektech nebo jej dále rozvíjet.

Literatura

[Aar15] Android Developers. *App Resources* [online]. 2015, [cit. 2015-06-01]. Dostupné z: `http://developer.android.com/guide/topics/resources/index.html`

[Amp15] Android Developers. *Manifest.permission* [online]. 2015, [cit. 2015-05-29]. Dostupné z: `http://developer.android.com/reference/android/Manifest.permission.html`

[Apf15] Android Developers. *Platform Versions* [online]. Akt. 2015-06-01, [cit. 2015-06-01]. Dostupné z: `https://developer.android.com/about/dashboards/index.html`

[Art15] Android Developers. *Resource Types* [online]. 2015, [cit. 2015-06-01]. Dostupné z: `http://developer.android.com/guide/topics/resources/available-resources.html`

[Asa15] Android Source. *ART and Dalvik* [online]. 2015, [cit. 2015-06-18]. Dostupné z: `https://source.android.com/devices/tech/dalvik/index.html`

[Asd15] Android Source. *Dalvik Executable format* [online]. 2015, [cit. 2015-06-18]. Dostupné z: `https://source.android.com/devices/tech/dalvik/`

[Asl15] Android Developers. *Support Library* [online]. Akt. 2015-05, ver. 22.2.0, [cit. 2015-06-01]. Dostupné z: `http://developer.android.com/tools/support-library/index.html`

[Asp15] Android Developers. *System Permissions* [online]. 2015, [cit. 2015-05-29]. Dostupné z: `http://developer.android.com/guide/topics/security/permissions.html`

[Aui15] Android Developers. *UI Overview* [online]. 2015, [cit. 2015-06-18]. Dostupné z: https://developer.android.com/guide/topics/ui/overview.html

[Bmp13] Mob Partner. *Top 5 HTML5 Frameworks' for Mobile Web Apps* [online]. Pub. 2013-06-24, [cit. 2015-06-01]. Dostupné z: http://blog.mobpartner.com/2013/06/24/top5-html5-frameworks-web-apps/

[Ceb14] GNEST, Anja. *Mobile Trends 2014* [online]. Pub. 2014-02-21, [cit. 2015-05-28]. Dostupné z: http://blog.cebit.de/2014/02/21/infographic-mobile-trends-2014/

[Ctu15] Český telekomunikační úřad. *Veřejné širokopásmové mobilní sítě* [online]. Akt. 2015-05-19, [cit. 2015-05-28]. Dostupné z: http://lte.ctu.cz/pokryti/

[Dom04] W3C. *Document Object Model (DOM) Level 3 Core Specification* [online]. Pub. 2004-04-07, ver. 1.0, [cit. 2015-06-18]. Dostupné z: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

[For14] GRIFFITH, Erin pro Fortune.com *Who's winning the consumer cloud storage wars?* [online]. Pub. 2014-11-06, [cit. 2015-06-18]. Dostupné z: http://fortune.com/2014/11/06/dropbox-google-drive-microsoft-onedrive/

[Gio15] Google Developers. *Google I/O Keynote* [video]. Pub. 2015-05-28, [cit. 2015-05-28]. Dostupné z: https://www.youtube.com/watch?v=7V-fIGMDsmE

[Gra13] GRANT, Allan. *Android 4 : průvodce programováním mobilních aplikací*, 1. vyd. Brno: Computer Press, 2013. 656s. ISBN 978-80-251-3782-6

[Hul12] HULA, Josef. *Dynamická tvorba aplikací v systému Android* [online]. 2012, [cit. 2015-05-28]. Bakalářská práce. ZÁPADOČESKÁ UNIVERZITA V PLZNI, Fakulta aplikovaných věd. Vedoucí práce Ladislav Pešička. Dostupné z: http://theses.cz/id/vf9jn8/.

[Iwa15] Google Developers. *Installable Web Apps* [video]. 2015, [cit. 2015-05-28]. Dostupné z: https://www.youtube.com/watch?v=N1Bdu7ukN40

[Lac15] LACKO, Ľuboslav. *Vývoj aplikací pro Android*, 1. vyd. Brno: Computer Press, 2015. 472s. ISBN 978-80-251-4347-6

[Lan14] LANDWERTH, Immo. *.NET Core is Open Source* [online]. Pub. 2014-11-12, [cit. 2015-06-01]. Dostupné z: `http://blogs.msdn.com/b/dotnet/archive/2014/11/12/net-core-is-open-source.aspx`

[Luo15] LUONTOLA, Esko. *Retrolambda* [online]. Akt. 2015-04-14, ver. 2.0.2, [cit. 2015-06-01]. Dostupné z: `https://github.com/orfjackal/retrolambda`

[Mar10] MARCOTTE, Ethan. *Responsive Web Design* [online]. Pub. 2015-05-25, [cit. 2015-06-01]. Dostupné z: `http://alistapart.com/article/responsive-web-design`

[Nam13] NAM, Changwon. *Decompiling Android* [online]. Pub. 2013-09-28, [cit. 2015-06-18]. Dostupné z: `http://nckwon.blogspot.cz/2013/09/decompiling-android.html`

[Not05] NOTTINGHAM, M. - SAYRE, R. *The Atom Syndication Format* [online]. Pub. 2005-12, [cit. 2015-06-18]. Dostupné z: `https://tools.ietf.org/html/rfc4287`

[Omt15] Oracle Java documentation. *Obtaining Method Type Information* [online]. 2015, [cit. 2015-06-18]. Dostupné z: `https://docs.oracle.com/javase/tutorial/reflect/member/methodType.html`

[Rss09] RSS Advisory board. *RSS 2.0 Specification* [online]. Pub. 2009-04-30, [cit. 2015-06-18]. Dostupné z: `http://www.rssboard.org/rss-specification`

[Rud14] RUDOPLPH, Patrick. *Hybrid Mobile Apps: Providing A Native Experience With Web Technologies* [online]. Pub. 2014-10-21, [cit. 2015-06-01]. Dostupné z: `http://www.smashingmagazine.com/2014/10/21/providing-a-native-experience-with-web-technologies/`

[Srr14] Staffing Robot. *Responsive Web Design is Not a Fad, It's Great for SEO* [online]. Pub. 2014-12-05, [cit. 2015-06-01]. Dostupné z: `http://www.staffingrobot.com/2014/12/responsive-web-site-design-just-fad-great-seo.html`

[Sta13] STAFFA, Milan. *Polymorfní aplikace pro systém Android* [online]. 2013, [cit. 2015-05-28]. Diplomová práce. ZÁPADOČESKÁ UNIVERZITA V PLZNI, Fakulta aplikovaných věd. Vedoucí práce Ladislav Pešička. Dostupné z: `http://theses.cz/id/fiunqh/`.

[Sum15] SUMMERFIELD, Jason. *Mobile Website vs. Mobile App* [online]. 2015, [cit. 2015-05-28]. Dostupné z: http://www.hswsolutions.com/services/mobile-web-development/mobile-website-vs-apps/

[Tat15] TATARKA, Evan. *Gradle-Retrolambda* [online]. Akt. 2015-05-02, ver. 3.1.0, [cit. 2015-06-01]. Dostupné z: https://github.com/evant/gradle-retrolambda

[Vog13] VOGEL, Lars. *RSS feeds with Java - Tutorial* [online]. Akt. 2013-03-06, [cit. 2015-06-18]. Dostupné z: http://www.vogella.com/tutorials/RSSFeed/article.html

[W3r15] W3Schools. *HTML Responsive Web Design* [online]. 2015, [cit. 2015-06-01]. Dostupné z: http://www.w3schools.com/html/html_responsive.asp

[W3t15] W3Techs. *Usage of content management systems for websites* [online]. Akt. 2015-06-06, [cit. 2015-06-06]. Dostupné z: http://w3techs.com/technologies/overview/content_management/all

[Wpc15] Wordpress.com. *Responsive Layout WordPress Themes* [online]. 2015, [cit. 2015-06-01]. Dostupné z: https://theme.wordpress.com/themes/features/responsive-layout/

[Xaf15] Xamarin. *Introduction to Xamarin.Forms* [online]. 2015, [cit. 2015-05-28]. Dostupné z: http://developer.xamarin.com/guides/cross-platform/xamarin-forms/introduction-to-xamarin-forms/

[Zai14] ZAITSEV, Serge. *Lambda support for Android* [online]. Pub. 2014-01-09, [cit. 2015-06-01]. Dostupné z: http://zserge.com/blog/android-lambda.html

Seznam zkratek

V textu práce byly použity následující výrazy a zkratky:

- 3D - Three-dimensional, trojrozměrný
- Activity - aktivita, základní část aplikace pro OS Android
- Android - operační systém mobilních zařízení
- AOT - Ahead Of Time, předběžná kompilace kódu
- API - Application Programming Interface, aplikační rozhraní
- APK - Android Application Package, balíček aplikace
- ART - prostředí platformy Android
- Button - tlačítko, prvek uživatelského rozhraní aplikace
- C# - programovací jazyk
- CSS - Cascading Style Sheets, kaskádové styly
- Dalvik - prostředí platformy Android
- DEX - Dalvik EXecutable, spustitelný kód Dalviku
- DOM - Document Object Model, objektový model dokumentu
- Drawable - grafický zdroj aplikace
- DVM - Dalvik Virtual Machine, virtuální stroj Dalviku
- DX - nástroj pro kompilaci balíčků dex
- Eclipse - vývojové prostředí

- EditText - kolonka, prvek uživatelského rozhraní aplikace
- Fragment - část aktivity, základní část aplikace pro OS Android
- Framework - sada vývojářských nástrojů
- HSPA+, 3G - předchůdce mobilního připojení LTE
- HTML - HyperText Markup Language, značkovací jazyk pro tvorbu webu
- CheckBox - prvek uživatelského rozhraní aplikace
- ID - identifikátor
- ImageView - obrázek, prvek uživatelského rozhraní aplikace
- Integer - typ celého čísla
- iOS - operační systém společnosti Apple
- JAR - Java ARchive, balíček kódu jazyka Java
- Java - programovací jazyk
- JavaScript - programovací jazyk
- JIT - Just In Time, kompilace kódu za běhu
- JVM - Java Virtual Machine, virtuální stroj Javy
- Layout - definice rozložení uživatelského rozhraní
- LinearLayout - řádkové rozložení prvků View v uživatelském rozhraní
- LINQ - Language-Integrated Query, dotazovací funkce jazyka C#
- LTE, 4G - standard rychlého mobilního připojení
- Manifest - soubor popisující aplikaci a její povolení
- Mono - platforma pro aplikace v jazyku C#
- .NET - platforma pro aplikace v jazyku C#
- OS - operační systém
- Permission - oprávnění aplikace

- RelativeLayout - relativní rozložení prvků View v uživatelském rozhraní
- RSS - Really Simple Syndication, systém zpřístupnění zpráv v síti
- ScrollView - posuvné rozložení prvků View v uživatelském rozhraní
- SDK - Software Development Kit, sada vývojových nástrojů
- SEO - Search Engine Optimization, optimalizace webu pro vyhledávač
- SHA1 - 160 bitové kódování pro tvorbu klíčů aplikací
- StAX - Streaming API for XML, nástroj pro čtení XML proudů
- String - typ řetězce
- TextView - text, prvek uživatelského rozhraní aplikace
- ToolBar - nástrojový panel, prvek uživatelského rozhraní aplikace
- UI - User Interface, uživatelské rozhraní
- UI Thread - hlavní vlákno aplikace
- URL - Uniform Resource Locator, jednotná adresa zdroje
- View - prvek uživatelského rozhraní aplikace
- ViewGroup - prvek uživatelského rozhraní aplikace obsahující prvky View či ViewGroup
- WebView - zobrazení HTML kódu v aplikaci
- Wordpress - platforma pro vývoj webových stránek
- XAML - eXtensible Application Markup Language, značkovací jazyk pro grafická rozhraní
- XML - eXtensible Application Markup Language, značkovací jazyk

A Oprávnění aplikace

První sada povolení obsahuje položky potřebné pro přístup k síti. Ten je potřebný jednak pro stahování požadovaných souborů šablon a archivů dex, jednak je vyžadován ukázkovými aplikacemi. Bez připojení k síti je možné stále využívat poslední použité soubory z paměti zařízení a případně pouštět dílčí aplikace, které pro svoji funkci připojení k síti nevyžadují.

Položka *ACCESS_NETWORK_STATE* slouží pro ověření, zda je připojení aktivní, či nikoli, zatímco povolení *INTERNET* je vyžadováno pro použití připojení k síti.

```
<uses-permission android:name=
"android.permission.INTERNET" />
<uses-permission android:name=
"android.permission.ACCESS_NETWORK_STATE" />
```

Ukázka kódu A.1: Oprávnění - Internet

Povolení *WRITE_EXTERNAL_STORAGE* zajišťuje možnost zápisu souborů na paměťovou kartu (obecně do externí paměti zařízení). Toto povolení zahrnuje rovněž i čtení souborů z externí paměti.

```
<uses-permission android:name=
"android.permission.WRITE_EXTERNAL_STORAGE" />
```

Ukázka kódu A.2: Oprávnění - Externí paměť

Poslední povolení slouží k přiřazení polymorfní aplikace k uživatelskému účtu zařízení a k využití údajů z tohoto uživatelského účtu pro přihlášení do služby Google Drive.

```
<uses-permission android:name=
"android.permission.GET_ACCOUNTS" />
<uses-permission android:name=
"android.permission.USE_CREDENTIALS" />
```

Ukázka kódu A.3: Oprávnění - Uživatelské účty

B Uživatelská příručka

V této příloze jsou popsány základní kroky pro úspěšnou instalaci a používání vytvořené polymorfní aplikace a pro práci s jednotlivými generovanými aplikacemi.

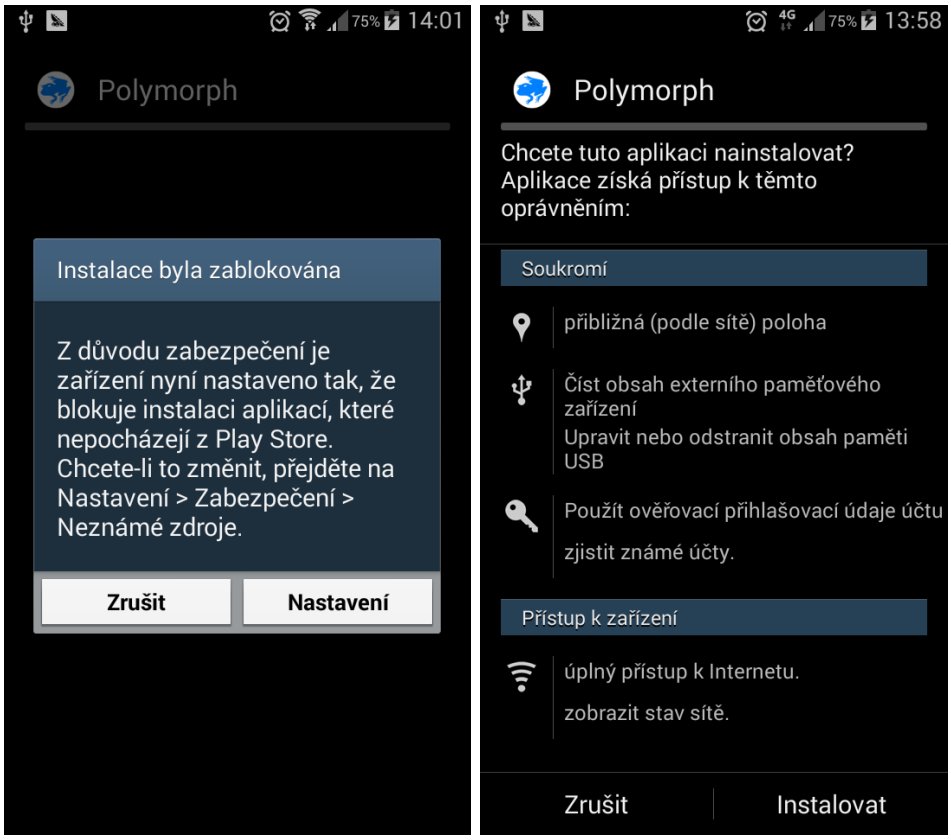
B.1 Instalace aplikace

Jedním způsobem, jak nainstalovat aplikaci do zařízení, je import projektu aplikace z přiloženého CD do vývojového prostředí Android Studio a následné spuštění projektu. Takový přístup umožňuje současnou kontrolu zdrojových kódů a logu zařízení. Rovněž je možné použít nástroje pro ladění aplikace a její běh na vybraném místě pozastavit či krokovat.

Klasickým přístupem je však instalace pomocí aplikačního balíčku **polymorph.apk**. Ten se nachází na přiloženém CD ve složce apk. Jedná se o aplikaci vydanou neoficiální cestou, proto bude na většině zařízení nutné povolit instalaci aplikací z neznámých zdrojů.

Tento krok je možné podniknout samostatně pomocí odpovídající možnosti v *Nastavení* > *Zabezpečení* > *Správa zařízení* (může se lišit dle modelu a verze systému). Případně k němu bude uživatel vyzván při pokusu o instalaci aplikace, jak ukazuje obrázek B.1a.

V prvním kroku instalace je nutné provést souhlas se všemi povoleními, které aplikace pro svůj běh potřebuje. Seznam těchto povolení při instalaci zobrazuje snímek B.1b, jejich seznam a odůvodnění pak popisuje kapitola A. Po úspěšném dokončení aplikace je možné ji používat.



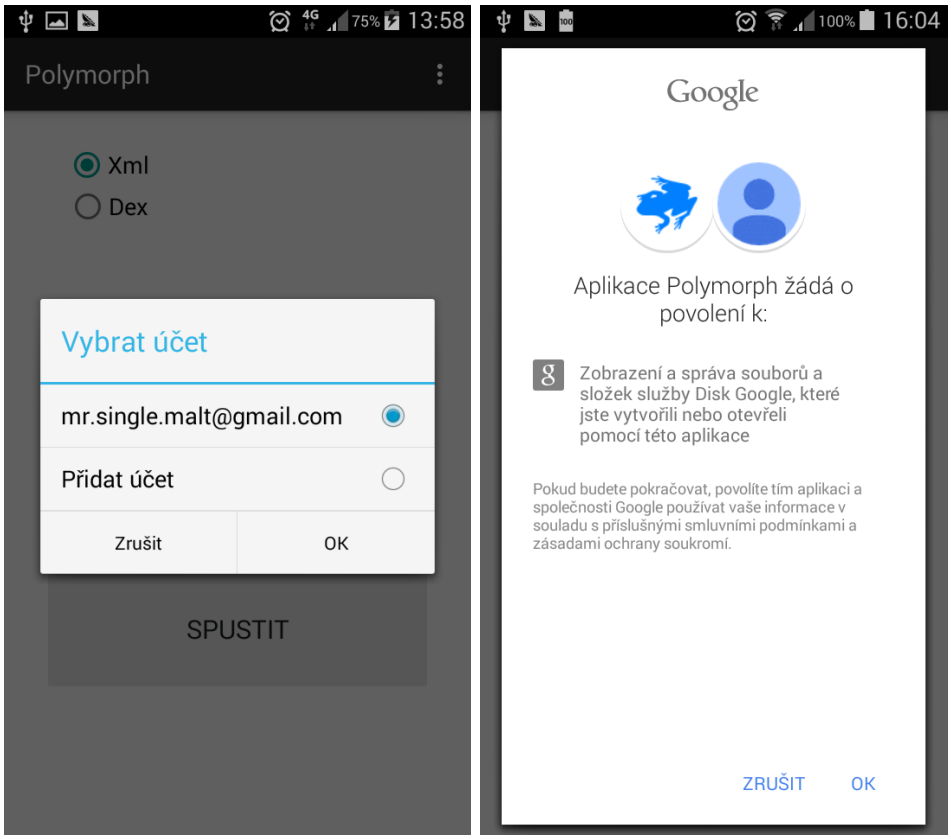
(a) Neznámé zdroje (b) Povolení aplikace

Obrázek B.1: Instalace polymorfní aplikace

B.2 Používání aplikace

Při prvním spuštění aplikace je uživatel vyzván, aby provedl spárování aplikace se svým uživatelským účtem (obrázek B.2a). Po přidání tohoto účtu je provedeno spárování a vyžadován souhlas uživatele s přístupem aplikace k jeho datům ve službě Google Drive, viz obrázek B.2b. Pro zrušení párování je třeba odebrat odpovídající možnost v nastavení účtu Google v kategorii *Přihlášení a zabezpečení* > *Přidružené aplikace a weby* > *Spravovat aplikace* > *Polymorph*.

Po provedení autorizace je možné aplikaci používat, přičemž připojení k uživatelskému účtu už je zajišťováno automaticky a aplikaci lze používat bez omezení.

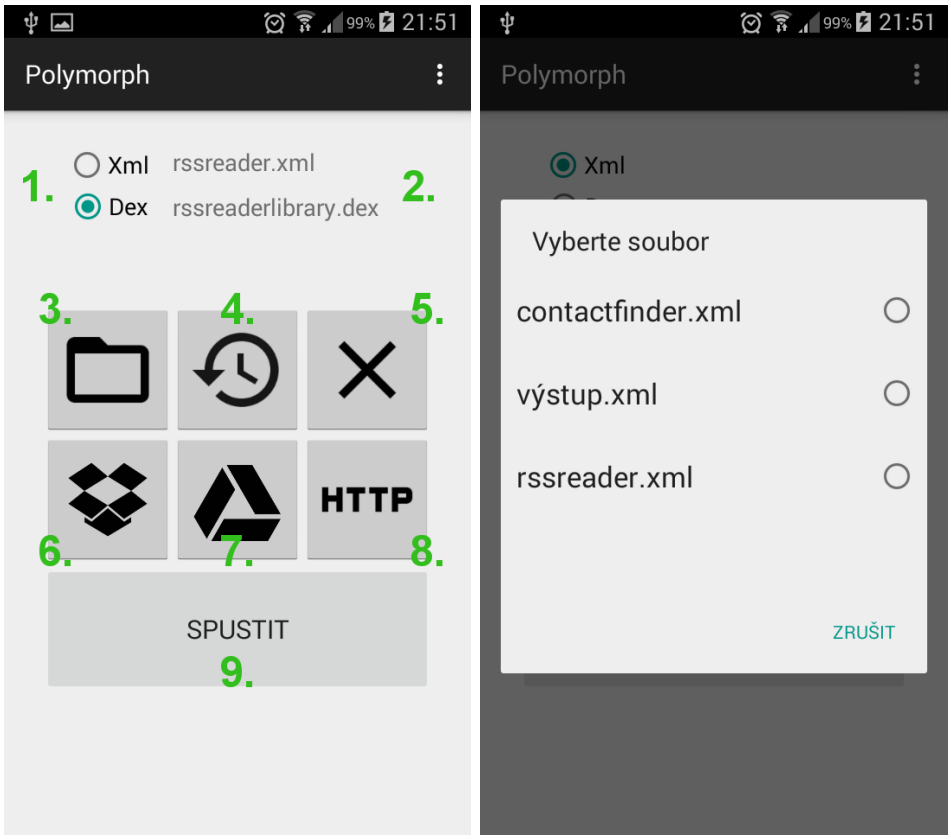


(a) Připojení účtu (b) Přístup ke Google Drive

Obrázek B.2: Párování s účtem Google Play

Obrázek B.3a ukazuje všechny prvky uživatelského rozhraní úvodní aktivity aplikace. Pomocí prvku **1** je možné přepínat mezi výběrem souboru šablony a souboru knihovny dex. Pokud je prvek vybrán, uložen a přiřazen, zobrazí se název daného souboru v prvku **2**, který je jinak prázdný.

Prvky **3, 4, 6, 7, 8** slouží pro výběr souboru z paměti zařízení, z historie, ze služby Dropbox, z Úložiště Google Drive a z vybraného URL. Dialogy těchto služeb zobrazuje příloha E.2. Tlačítko **5** slouží k odstratnění dané položky z výběru. Prvek **9** pak slouží k přechodu na generovanou aplikaci v další aktivitě. Na obrázku B.3b je zobrazen výběr souboru z historie posledních použitých šablon.



(a) Prvky rozhraní (b) Historie výběrů

Obrázek B.3: Používání aplikace

B.3 Vytvoření dex archivu

Vytvoření dex knihovny pro generovanou aplikaci je relativně jednoduchou záležitostí. Nejprve je nutné založit nový Java projekt v libovolném IDE. Struktura balíků, obsah tříd a funkce kódu je na programátorovi. Je však nutné zajistit, aby základní třída projektu obsahovala metodu *doLibrary-Function*, nejlépe pomocí implementace rozhraní *ILibraryFunction*, které je součástí projektu aplikace a nachází se v balíku *cz.zcu.kiv.polymorph.type*. Načítání knihovny je popsáno v kapitole 5.2.2, se kterou je nutné se předem seznámit.

Pro volanou funkci je potřeba vytvořit odpovídající šablonu, ze které bude možné funkci spustit. Struktura prvků šablon je popsaná v příloze C a je nutné, aby programátor zachoval požadovanou strukturu šablony.

Funkcionalitu programu je vhodné nejprve otestovat přímo ve vývojovém prostředí a odladit případné chyby. Poté je možné projekt přeložit libovolným způsobem pomocí kompilátoru *javac*. Dalším krokem je zabalení vygenerovaných souborů *.class* do společného archivu. To lze provést například příkazem uvedeným na ukázce B.1 z příkazové řádky. *lib.jar* představuje název výsledného archivu, druhým parametrem je umístění souborů k zabalení, v tomto případě *./com*.

```
jar cf lib.jar ./com
```

Ukázka kódu B.1: Vytvoření archivu jar

Po zabalení souborů je vhodné překontrolovat, jestli cesta k cílovému souboru opravdu sedí s cestou uvedenou v šabloně pro spuštění aplikace, jinak nebude možné spustit požadovanou funkci. Jakmile bude archiv překódován do formátu dex, nebude již možné zobrazit a procházet jeho obsah.

Kompilaci nástrojem *dx* je možné spustit příkazem ve tvaru uvedeném na ukázce B.2. Umístění nástroje dx se liší na každém stroji, proto není možné vytvořit jednoduchý kompilační skript. Standardní umístění nástroje dx je *<Android_SDK_Path>/build-tools/<verze>/dx.bat*. Cestu k Android SDK má nastavenu uživatel individuálně a je možné ji zjistit například pomocí nástroje *Android SDK Manager*, který slouží ke správě vývojářských nástrojů platformy Android.

```
dx --dex --verbose --keep-classes --output="./lib.dex"
   ./lib.jar
```

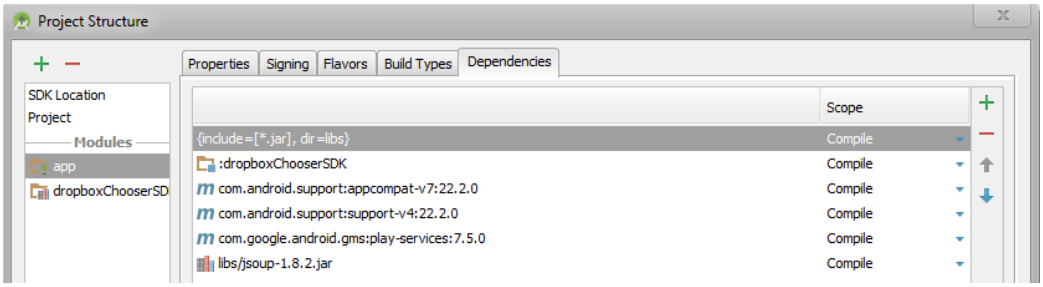
Ukázka kódu B.2: Vytvoření archivu dex

V ukázce B.2 představuje *./lib.jar* archiv vytvořený v ukázce B.1, jméno výstupního archivu je pak nutné uvést do parametru *--output*. Parametr *--verbose* dává programátorovi přehled o tom, jaké třídy již byly zpracovány. Tímto krokem je knihovna dex vytvořena a lze ji použít v zařízení.

B.4 Použití externí knihovny

Nástroj *dx* neumí spolehlivě zabalit do výsledného archivu knihovny třetích stran. Proto je nutné při jejich použití přidat knihovny přímo do projektu polymorfní aplikace pomocí následujících kroků:

1. V nástroji Android Studio přepněte v postranním panelu perspektivu z *Android* na *Project*.
2. V modulu aplikace *app* vyberte adresář *libs* a do něj nakopírujte vybranou knihovnu.
3. V nástrojovém panelu vyberte možnost *File* > *Project Structure*.
4. Ve zobrazeném okně vyberte modul *app* a kartu *Dependencies*, viz obrázek B.4.
5. Pomocí nástroje přidat (+) přidejte požadovanou knihovnu *(File Dependency)*.
6. Potvrďte tlačítkem *OK*. Po sestavení projektu by měla být knihovna jeho součástí.



Obrázek B.4: Závislosti projektu

Pro knihovny využívající systému závislostí Gradle nebo Maven je proces jednodušší a stačí pouze zanést knihovnu do *dependencies* souboru *build.gradle* modulu *app*.

C Prvky uživatelského rozhraní

V rámci této kapitoly jsou popsány takové prvky, které lze v generované aplikaci vykreslit pomocí šablon uživatelského rozhraní, a tak se do velké míry prolíná s přílohou D. Každému z prvků uživatelského rozhraní lze přiřadit určitý počet vybraných argumentů. Při zpracování šablony rozhraní pomocí XML parseru pak dochází k výběru těchto prvků a jejich atributů, na jejichž základě se pak vytvářejí samotné objekty uživatelského rozhraní.

C.1 Obecné vlastnosti

Jak popisuje část 5.1.3, jsou všechny vytvořené prvky nejprve obaleny rozložením LinearLayout a následovně vloženy do prvku ScrollView. Pro tento rodičovský LinearLayout je použita výchozí direktiva, která zajišťuje, že všechny prvky vložené do seznamu budou mít stejnou váhu. A rovněž se budou snažit vyplnit maximum přiděleného prostoru.

```
LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
   LinearLayout.LayoutParams.MATCH_PARENT, defaultWeight);
parent.setLayoutParams(lp);
```

Ukázka kódu C.1: Výchozí nastavení LinearLayout

Tím je zajištěno, že pokud bude jediným definovaným prvkem daného fragmentu pouze LinearLayout, dostane k dispozici prostor celého displeje, nikoli pouze jednu řádku z obalujícího prvku.

Váhu jednotlivých prvků obalujícího layoutu je možné měnit pomocí parametru *weight*. Jeho hodnota udává, jaký podíl prostoru by měl v rámci obalujícího LinearLayout zabrat. Například LinearLayout obsahující tři prvky s vahami $1, 1, 2$ by měl přidělit polovinu $(2/4)$ prostoru poslednímu, největšímu, prvku a zbylý prostor rozdělit mezi dva menší prvky. Nicméně je nutné vzít v potaz to, že na finální zobrazení mají vliv například i rozestupy mezi prvky, okraje a další faktory.

Při použití atributu **weight** je však třeba dát pozor na to, že určuje rozměr prvku pouze v jednom rozměru. Pokud tedy prvek obsahuje text či popisek, bude jeho obsah rovněž přizpůsoben rozměrům celého prvku. Může

se tak stát, že obsah prvku přeteče na další řádky a tím dojde k protažení a vizuální deformaci objektu. Váhu prvkům je tedy potřeba přiřazovat s rozmyslem.

Všechny prvky s textovým obsahem mohou zpracovávat jak obsah atributu **text**, tak přímo textový obsah elementu. To umožňuje použití atributu pro text tlačítek, zatímco bloky textu je možné umístit dovnitř elementu v XML souboru šablony.

C.2 Fragment

Fragment slouží jako prvek rozhraní, ve kterém se všechen obsah nahraný ze šablony zobrazí. Pro jednoduchost je možné v rámci jednoho souboru šablony definovat více fragmentů, kde každý bude mít svůj titulek daný atributem **title** a vlastní prvky rozhraní, tím pádem i vlastní funkci. Pokud šablona neobsahuje přímo prvek typu <*fragment*>, považuje se veškerý její obsah za obsah jednoho fragmentu a je s ním dále i tak nakládáno.

Pokud obsahuje titulek fragmentu znaky _, jsou tyto znaky nahrazeny ve výsledném názvu znakem „mezera“. Analogický postup je použit při vytváření atributu XML šablony z řetězce názvu.

Atribut	**Hodnoty**
• title	libovolný řetězec validní v souboru šablony

Tabulka C.1: Atributy prvku *fragment*

C.3 LinearLayout

Pomocí kombinace prvků <*linearlayout*> lze rozložení displeje rozdělit na řádky a sloupce. Orientace layoutu může být dvojí - *horizontal* (prvky jsou použité jako sloupečky) a *vertical* (prvky jsou použité jako řádky). V souboru šablony je orientace specifikována atributem **orientation**.

Pro lepší vizuální orientaci v prvcích grafického rozhraní lze použít též prvek oddělovače. Jeho vzhled je pevně nastaven v samotné aplikaci, uživatel pouze může specifikovat, zda se prvek zobrazí, či nikoli.

Atribut	**Hodnoty**
• orientation	*horizontal* nebo *vertical* (výchozí)
• divider	*true* nebo *false* (výchozí)
• weight	číslo formátu double (výchozí hodnota 1.0)

Tabulka C.2: Atributy prvku *linearlayout*

C.4 Button

Tlačítka používají kromě základního popisku i několik dalších důležitých atributů, které sice nebyly v ukázkových aplikacích využity, nicméně mohou mít zásadní vliv na výsledný layout. Jedná se o atributy **visible**, který umí tlačítko zneviditelnit ve výsledném layoutu při zachování jeho pozice. A dále byl použit atribut **enabled** sloužící k deaktivaci tlačítka.

Atribut	**Hodnoty**
• enabled	*false* nebo *true* (výchozí)
• visible	*false* nebo *true* (výchozí)
• weight	číslo formátu double (výchozí hodnota 1.0)

Tabulka C.3: Atributy prvku *button*

Součástí každého tlačítka je i funkce, kterou má tlačítko vykonávat. Obecně je možné specifikovat různé typy funkcí atributem **type**, nicméně v současné verzi polymorfní aplikace je implementována pouze funkce typu *replace*, která provádí spuštění metody *doLibraryFunction* ve třídě specifikované atributem **target**. Ve výsledné aplikaci nebyla zavedena možnost spouštění různých funkcí v cílové třídě, ale atribut **name** byl u objektu funkce ponechán pro možná budoucí rozšíření.

Atribut	Hodnoty
• type	libovolný řetězec validní v souboru šablony
• target	libovolný řetězec validní v souboru šablony
• name	libovolný řetězec validní v souboru šablony

Tabulka C.4: Atributy prvku *function*

C.5 EditText

Editovatelné položky mají společný atribut **hint**, který představuje nápovědu v dané kolonce předtím, než začne uživatel vyplňovat její obsah. Dalším významným atributem je **type**, který specifikuje použití daného prvku pro vybraný typ obsahu. Tím je možné provést obdobu validace vstupu na úrovni samotné kolonky.

Skutečná validace by však vyžadovala zpracování události při zadání údajů do daného políčka. Takovou funkci však není možné jednoduše provádět u dynamicky vytvořeného rozhraní. Proto je nutné vystačit si při tvorbě generovaných aplikací s několika vybranými typy vstupu. Případně použít obecný vstup *text* a validaci provádět až v rámci spouštěné funkce *doLibraryFunction*.

Atribut	Hodnoty
• hint	libovolný řetězec validní v souboru šablony
• type	*number* pro zadání celého čísla *phone* pro zadání telefonního čísla *datetime* pro zadání data *multiline* pro použití víceřádkového vstupu *text* (výchozí) pro obecný vstup
• weight	číslo formátu double (výchozí hodnota 1.0)

Tabulka C.5: Atributy prvku *edittext*

C.6 TextView

Nejdůležitějším atributem prvku pro zobrazení textu bývá zpravidla atribut **text**, kterému je přiřazen obsah prvku. Pro souvislé texty je však lepší využít možnosti vložení textu do těla elementu, což jednak zpřehledňuje zápis šablony, a pak zajišťuje, že validita dokumentu šablony nebude ohrožena nevhodným znakem v hodnotě atributu.

Textové prvky je možné stylovat pomocí atributu **style**, který umožňuje rozlišení nadpisů, běžného textu a textu psaného kurzívou. Rovněž je možné nastavit zarovnání textu k okraji či na střed pomocí atributu **gravity**.

Význam textu je možné zdůraznit atributem **type**, který umí rozlišit telefonní čísla, e-mailové adresy a webová URL. V případě použití URL je třeba jeho text předem překódovat tak, aby neobsahoval nebezpečné znaky. Při načtení tohoto textu pak proběhne dekódování, takže je zaručeno korektní zobrazení vybrané adresy.

Atribut	**Hodnoty**
• text	libovolný řetězec validní v souboru šablony (Lze nahradit obsahem elementu)
• style	*italics* pro použití kurzívy *bold* pro použití tučného písma *bolditalics* pro kombinaci obou *normal* obyčejný text (výchozí)
• gravity	*right* zarovnání vpravo *center* zarovnání na střed *left*zarovnání vlevo (výchozí)
• type	*url* pro asociaci s prohlížečem *mail* pro asociaci s poštovním klientem *phone* pro asociaci s telefonem
• weight	číslo formátu double (výchozí hodnota 1.0)

Tabulka C.6: Atributy prvku *textview*

C.7 CheckBox

CheckBox představuje jednoduchý prvek se dvěma atributy - **text** pro popisek a **checked** pro výchozí hodnotu prvku.

Atribut	**Hodnoty**
• text	libovolný řetězec validní v souboru šablony
• checked	*true* nebo *false* (výchozí)
• weight	číslo formátu double (výchozí hodnota 1.0)

Tabulka C.7: Atributy prvku *checkbox*

C.8 ImageView

Prvek pro zobrazení grafického obsahu je definován jedním atributem - **src** - udávajícím zdrojovou adresu obrázku, který se zobrazí po načtení prvku v uživatelském rozhraní. Pokud z nějakého důvodu ke zobrazení nedojde, neprobíhá ani stahování samotného obrázku do zařízení.

Všechny prvky ImageView mají standardně obsah přizpůsobený velikosti prvku uživatelského rozhraní a vycentrovaný v rámci prvku. Je třeba tedy počítat s tím, že zobrazení dvou různých obrázků může na stejném prostoru působit rozlišně, a pokud je to nutné, použít oddělovač prvku LinearLayout k oddělení obrázku od ostatního obsahu.

Atribut	**Hodnoty**
• src	kódovaný řetězec s adresou obrázku
• weight	číslo formátu double (výchozí hodnota 1.0)

Tabulka C.8: Atributy prvku *imageview*

D Soubory šablon

Z předchozí kapitoly C vyplývá, že funkce vykonávaná v rámci generované aplikace musí vracet výstup splňující definici popsanou dokumentem na ukázce D.1.

```
<!DOCTYPE FRAGMENT [
<!ELEMENT fragment (linearlayout|button|edittext|textview|
   checkbox|imageview)*>
<!ATTLIST fragment title CDATA #IMPLIED>

<!ELEMENT linearlayout
   (linearlayout|button|edittext|textview|
   checkbox|imageview)*>
<!ATTLIST linearlayout orientation (horizontal|vertical)
   "horizontal">
<!ATTLIST linearlayout divider (true|false) "true">
<!ATTLIST linearlayout weight CDATA "1">

<!ELEMENT button (function)>
<!ATTLIST button text CDATA #IMPLIED>
<!ATTLIST button visible (true|false) "true">
<!ATTLIST button enabled (true|false) "true">
<!ATTLIST button weight CDATA "1">

<!ELEMENT function EMPTY>
<!ATTLIST function type CDATA #REQUIRED>
<!ATTLIST function target CDATA #REQUIRED>
<!ATTLIST function name CDATA #IMPLIED>

<!ELEMENT edittext EMPTY>
<!ATTLIST edittext hint CDATA #IMPLIED>
<!ATTLIST edittext type
   (number|phone|datetime|multiline|text) "text">
<!ATTLIST edittext weight CDATA "1">

<!ELEMENT textview CDATA>
<!ATTLIST textview style (italics|bold|italicsbold|normal)
   "normal">
<!ATTLIST textview gravity (right|center|left) "left">
<!ATTLIST textview type (url|mail|phone|text) "text">
<!ATTLIST textview weight CDATA "1">

<!ELEMENT checkbox EMPTY>
<!ATTLIST checkbox text CDATA #IMPLIED>
<!ATTLIST checkbox checked (true|false) "true">
<!ATTLIST checkbox weight CDATA "1">
```

```
<!ELEMENT imageview EMPTY>
<!ATTLIST imageview src CDATA #IMPLIED>
<!ATTLIST checkbox weight CDATA "1">

]>
```

Ukázka kódu D.1: Definice šablony fragmentu

D.1 RSS čtečka

Následující ukázky obsahují šablony použité pro spouštěcí část ukázkové aplikace RSS Čtečka (D.2) a výstup samotné čtečky se třemi položkami a bez obrázků (D.3).

```
<?xml version="1.0" encoding="utf-8"?>
<application title="RSS Čtečka">
  <edittext hint="url"/>
  <edittext hint="počet" type="number" />
  <checkbox text="checkbox" text="Načítat obrázky"/>
  <button text="Načíst">
    <function type="replace"
       target="com.example.control.RSSReader"
       name="readUrl"/>
  </button>
</application>
```

Ukázka kódu D.2: Šablona vstupní části

```
<?xml version="1.0" encoding="utf-8"?>
<fragment title="Sport_iDNES.cz_-_Nejlepší_sport">
<linearlayout orientation="vertical" divider="true">
  <linearlayout orientation="vertical">
    <textview style="bold">Čtyři duely, čtyři výhry.
       Basketbalovým výběrům druhý den v Baku
       vyšel</textview>
    <textview style="italics">24.06.2015, 21:14</textview>
    <textview>Po úterních chmurných premiérách na Evropských
       hrách vysvitlo pro české basketbalové týmy v Baku
       sluníčko. Muži i ženy získali po dvou důležitých
       vítězstvích a do play-off jdou z druhých míst svých
       základních skupin.</textview>
    <textview style="italics"
       type="url">http%3A%2F%2Foh.idnes.cz%2Fbaku-evropske-hry
-basketbal-3x3-deh-%2Fbaku-2015.aspx%3Fc%3DA150624_232404
```

```
_olympiada-baku-2015_ten%23utm_source%3Drss%26utm_medium%3D
feed%26utm_campaign%3Dsport%26utm_content%3Dmain</textview>
  </linearlayout>
  <linearlayout orientation="vertical">
    <textview style="bold">Portugalsko - Švédsko 1:1, drama
       v závěru, po remíze slavily oba týmy</textview>
    <textview style="italics">24.06.2015, 20:59</textview>
    <textview>Dlouho to vypadalo na nudnou bezgólovou
       remízu, až v závěru se zápas Portugalska se Švédskem
       na fotbalovém Euru do 21 let změnil v drama. Po
       vedoucím gólu Portugalců Švédové na chvíli ztratili
       postupovou jistotu, vrátil jim ji ale gól v poslední
       minutě.</textview>
    <textview style="italics"
       type="url">http%3A%2F%2Ffotbal.idnes.cz%2Fportugalsko
-svedsko-mistrovstvi-evropy-do-21-let-fbt-%2Fme-fotbal-2015
.aspx%3Fc%3DA150624_182654_me-fotbal-2015_min%23utm_source%3D
rss%26utm_medium%3Dfeed%26utm_campaign%3Dsport%26
utm_content%3Dmain</textview>
  </linearlayout>
  <linearlayout orientation="vertical">
    <textview style="bold">Hokejbalisté po bezbrankové
       remíze s Kanadou vyhráli na MS skupinu</textview>
    <textview style="italics">24.06.2015, 20:52</textview>
    <textview>Čeští hokejbalisté remizovali v závěrečném
       vystoupení v základní skupině na mistrovství světa v
       Zugu s Kanadou 0:0 a přes první bodovou ztrátu na
       turnaji ovládli tabulku. V pátek od 17:45 čeká
       svěřence trenéra Drahomíra Kadlece čtvrtfinálový
       souboj s vítězem kvalifikace mezi Finskem a
       Indií.</textview>
    <textview style="italics"
       type="url">http%3A%2F%2Fsport.idnes.cz%2Fmistrovstvi
-sveta-hokejbalistu-2015-dty-%2Fsporty.aspx
%3Fc%3DA150624_225412_sporty_ald%23utm_source%3D
rss%26utm_medium%3Dfeed%26utm_campaign%3Dsport%26
utm_content%3Dmain</textview>
  </linearlayout>
</linearlayout>
</fragment>
```

Ukázka kódu D.3: Ukázka výstupní části

D.2 Kontakty ZČU

Šablony pocházející z ukázkové aplikace Kontakty ZČU představují vstupní část generované aplikace (D.4) a výstup s vizitkou jednoho zaměstnance (D.5).

```
<?xml version="1.0" encoding="utf-8"?>
<application title="Kontakty ZČU">
  <edittext hint="Jméno"/>
  <edittext hint="Příjmení"/>
  <button text="Najít">
    <function type="replace"
       target="com.example.control.ContactFinder"/>
  </button>
</application>
```

Ukázka kódu D.4: Šablona vstupní části

```
<?xml version="1.0" encoding="utf-8"?>
<fragment title="Ing._Ladislav_Pešička">
<linearlayout orientation="vertical">
  <textview style="bold">Ing. Ladislav Pešička</textview>
  <linearlayout orientation="horizontal" divider="true">
    <imageview
       src="http%3A%2F%2Fwww.zcu.cz%2Ffoto.jsp%3Fid%3D16847"/>
    <linearlayout orientation="vertical" weight="2">
      <textview text="Telefon:"/>
      <textview style="italics"
         type="phone">+420377632469</textview>
      <textview/>
      <textview text="Email:"/>
      <textview style="italics"
         type="mail">pesicka@kiv.zcu.cz</textview>
    </linearlayout>
  </linearlayout>
  <textview text="Pracoviště:"/>
  <textview style="italics">Fakulta aplikovaných
     věd</textview>
  <textview text="Kancelář:"/>
  <textview style="italics">UN 358</textview>
</linearlayout>
</fragment>
```

Ukázka kódu D.5: Výstup aplikace - jeden kontakt

D.3 Další ukázky

Následující ukázka D.6 není v samotné aplikaci využita, jedná se pouze o příklad, jak lze jednotlivé prvky rozhraní používat. Vizuální podobu fragmentů popsaných touto šablonou si lze prohlédnout na obrázku E.5.

```
<?xml version="1.0" encoding="utf-8"?>
<application>
  <fragment title="texty">
    <linearlayout orientation="vertical">
        <textview text="Nadpis Vlevo" gravity="left"
           style="bold"/>
        <textview> Lorem ipsum dolor sit amet, consectetur
           adipiscing elit, sed do eiusmod tempor incididunt
           ut labore et dolore magna aliqua. Ut enim ad
           minim veniam, quis nostrud exercitation ullamco
           laboris nisi ut aliquip ex ea commodo consequat.
        </textview>
        <textview text="Nadpis Střed"
           gravity="centerhorizontal" style="bold"/>
        <textview style="italics"> Duis aute irure dolor in
           reprehenderit in voluptate velit esse cillum
           dolore eu fugiat nulla pariatur. Excepteur sint
           occaecat cupidatat non proident, sunt in culpa
           qui officia deserunt mollit anim id est laborum.
        </textview>
        <textview text="Nadpis Vpravo" gravity="right"
           style="bold"/>
    </linearlayout>
  </fragment>
  <fragment title="tlačítka">
    <linearlayout orientation="horizontal" divider="true">
      <button text="button1"/>
      <button text="button2"/>
      <button text="button3"/>
    </linearlayout>
    <linearlayout orientation="horizontal">
      <linearlayout orientation="vertical" divider="true"
         weight="1">
        <button text="button4"/>
        <button text="button5"/>
        <button text="button6"/>
      </linearlayout>
      <linearlayout orientation="vertical" weight="2">
        <imageview src="https://www.seznam.cz/favicons/
           26/bd3739b901aa5af7ba36d1d74fb5a042.jpg"/>
      </linearlayout>
    </linearlayout>
```

```
    <linearlayout orientation="horizontal">
      <button text="button8" weight="2"/>
      <button text="button9" weight="1"/>
    </linearlayout>
    <linearlayout orientation="horizontal">
      <button text="button10" weight="2"/>
      <linearlayout orientation="horizontal" weight="1">
        <button text="button11"/>
        <button text="button12"/>
      </linearlayout>
    </linearlayout>
    <linearlayout orientation="horizontal">
      <button text="button13"/>
      <button text="button14"/>
    </linearlayout>
  </fragment>
  <fragment title="editace">
    <edittext hint="datum" type="datetime"/>
    <edittext hint="telefon" type="phone"/>
    <edittext hint="cislo" type="number"/>
    <edittext hint="multiline text" type="multiline"/>
    <checkbox text="checkbox"/>
  </fragment>
</application>
```

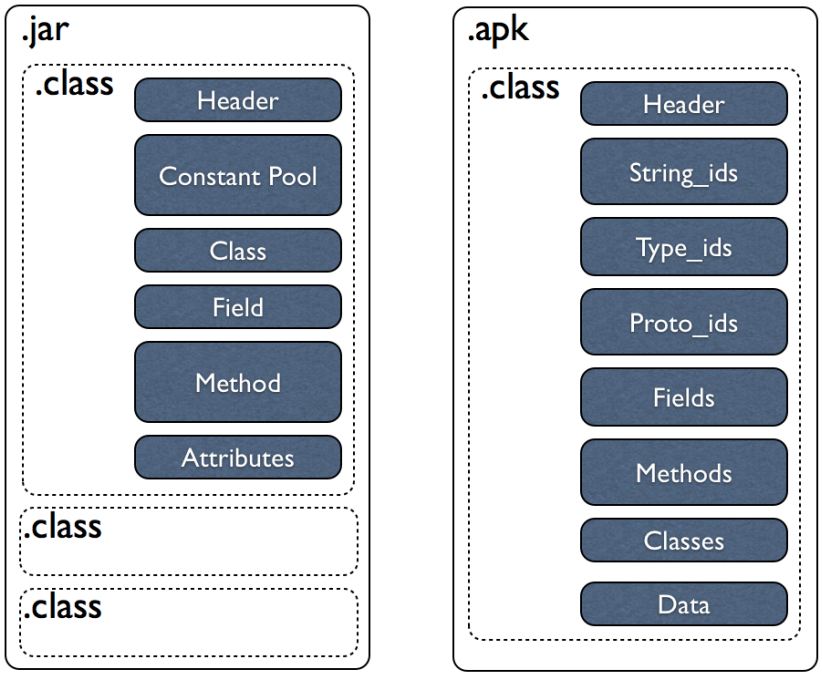
Ukázka kódu D.6: Ukázka textů, tlačítek a editovatelných prvků

E Další grafické přílohy

Na následující přílohy je odkazováno v textu práce, ačkoli do něj nebyly přímo zařazeny. Jedná se o obrázky informativního charakteru, které nejsou nutné pro pochopení principu fungování polymorfní aplikace, nicméně mohou být zajímavé zejména pro uživatele, který nebude aplikaci zkoušet přímo na fyzickém zařízení.

E.1 Archivy dex a jar

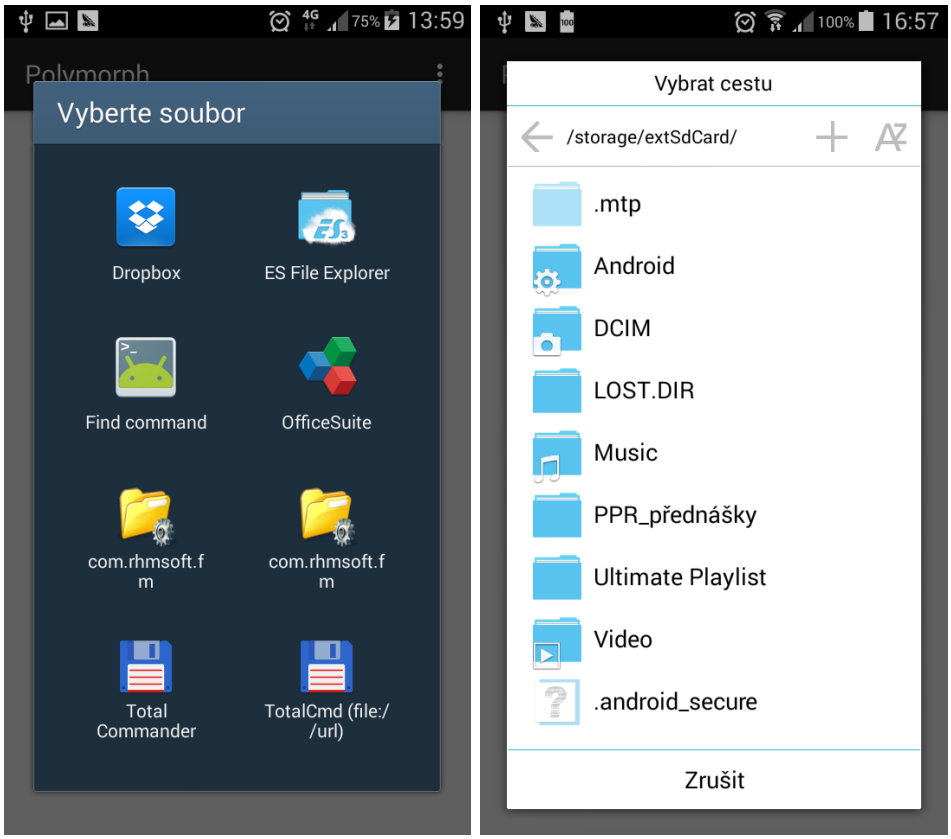
Obrázek E.1 znázorňuje rozdíl v systému kompilace obou archivů. Jedná se též o důvod, proč nejde oba typy libovolně zaměňovat.



Obrázek E.1: Struktura jar a dex archivů, zdroj: [Nam13]

E.2 Polymorfní aplikace

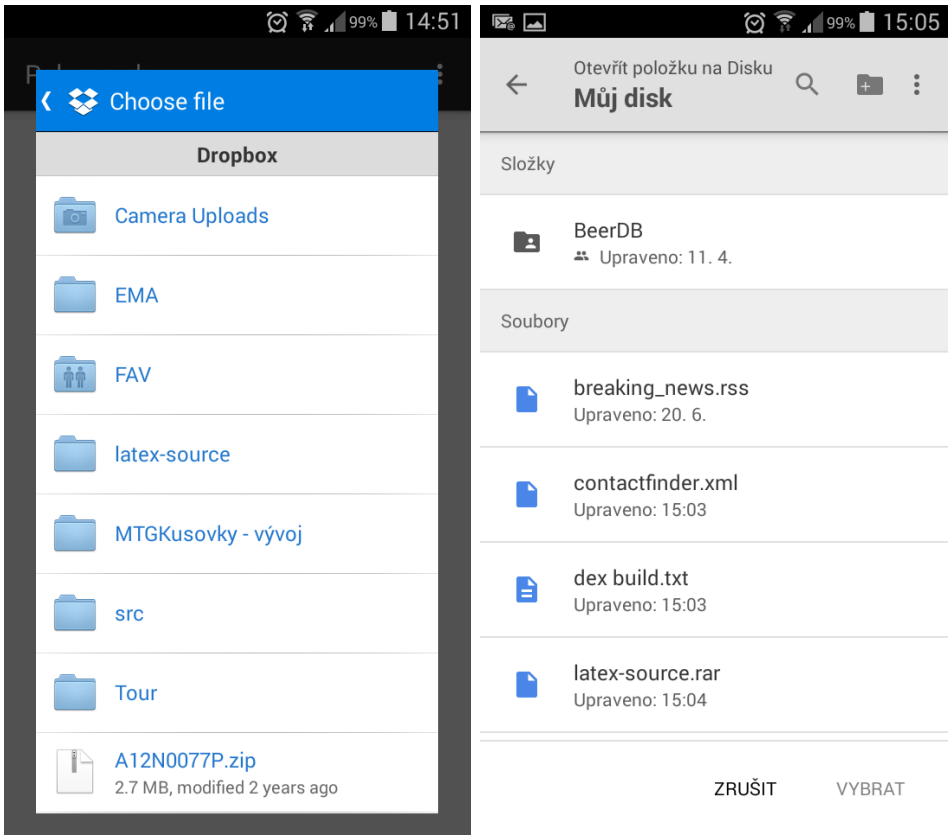
Následující snímky obrazovek E.2 ukazují postup při výběru souboru z paměti zařízení. Prvním krokem je vybrání aplikace správce souborů (E.2a), následuje výběr souboru v samotné aplikaci *ES File Explorer* (E.2b).



(a) Výběr aplikace (b) Výběr souboru

Obrázek E.2: Výběr souboru ze zařízení

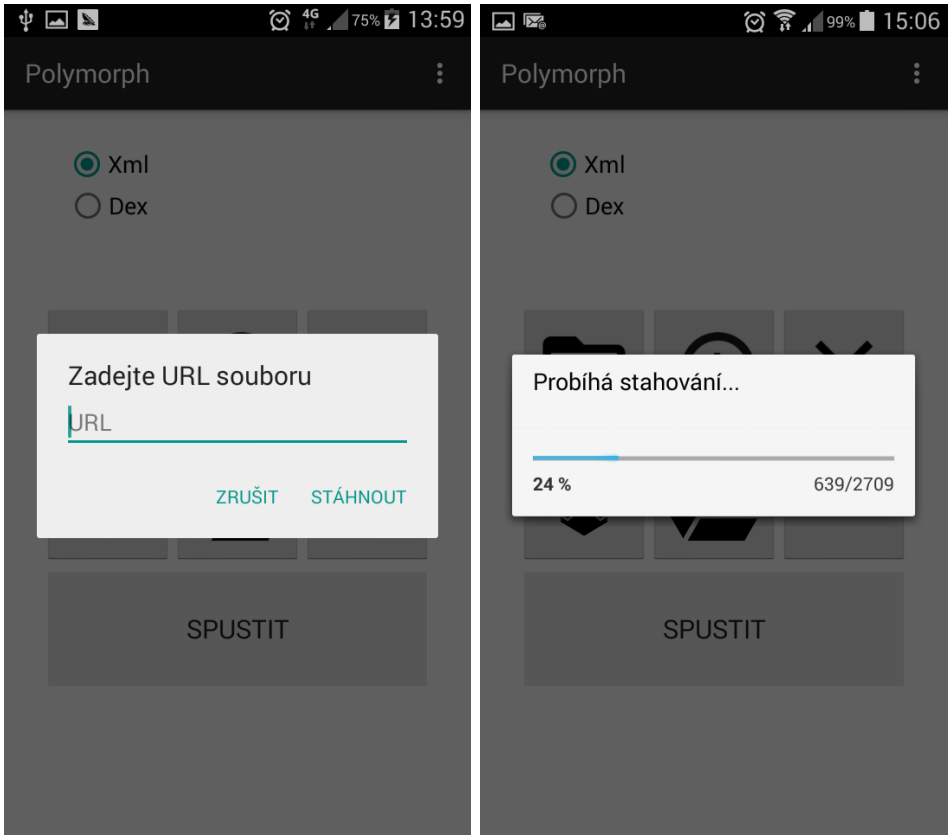
Obdobný postup je na snímcích E.4, jedná se o výběr souboru z úložiště *Dropbox* (E.3a) a výběr souboru z úložiště *Google Drive* (E.3b).



(a) Dropbox (b) Google Drive

Obrázek E.3: Výběr souboru z cloudového úložiště

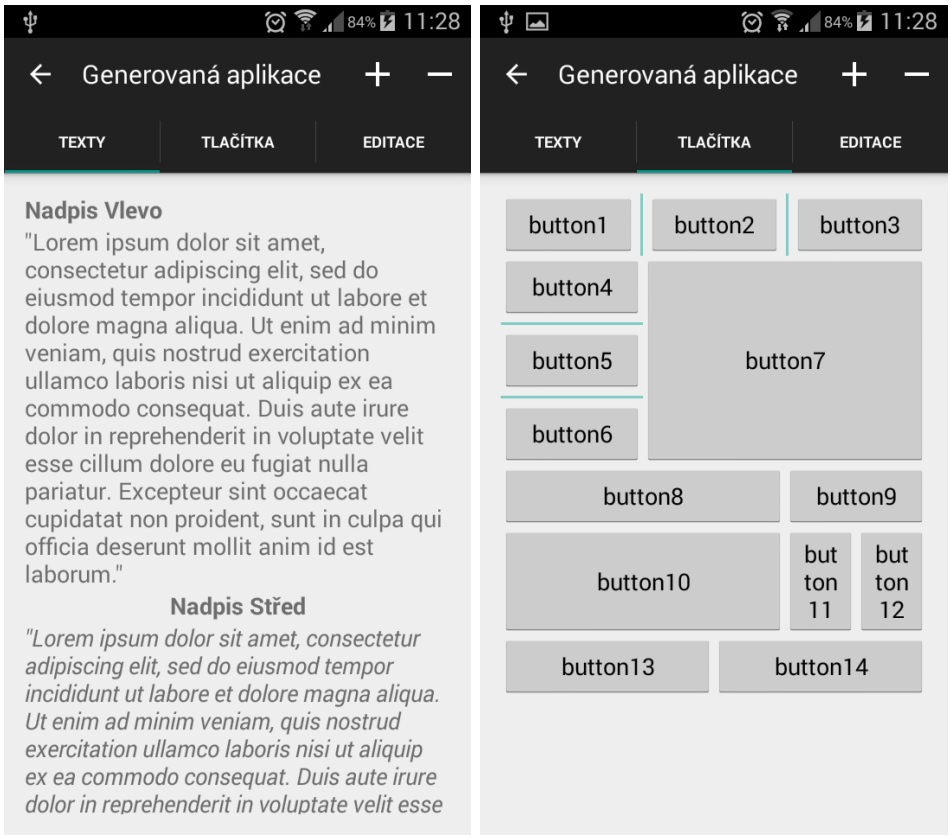
Při stažení souboru z URL je nejprve třeba zadat jeho adresu, a to ručně, či vložením zkopírovaného odkazu (E.4a). Následně je třeba vyčkat, než bude soubor stažen do zařízení (E.4b).



(a) Zadání URL (b) Progress dialog

Obrázek E.4: Stažení souboru z URL

Poslední ukázka E.5 představuje různé kombinace prvků uživatelského rozhraní. První část představuje kominaci různých typů prvku *TextView* (E.5a). Druhý snímek pak předvádí různé kombinace použití prvků *LinearLayout* a *Button* (E.5b).

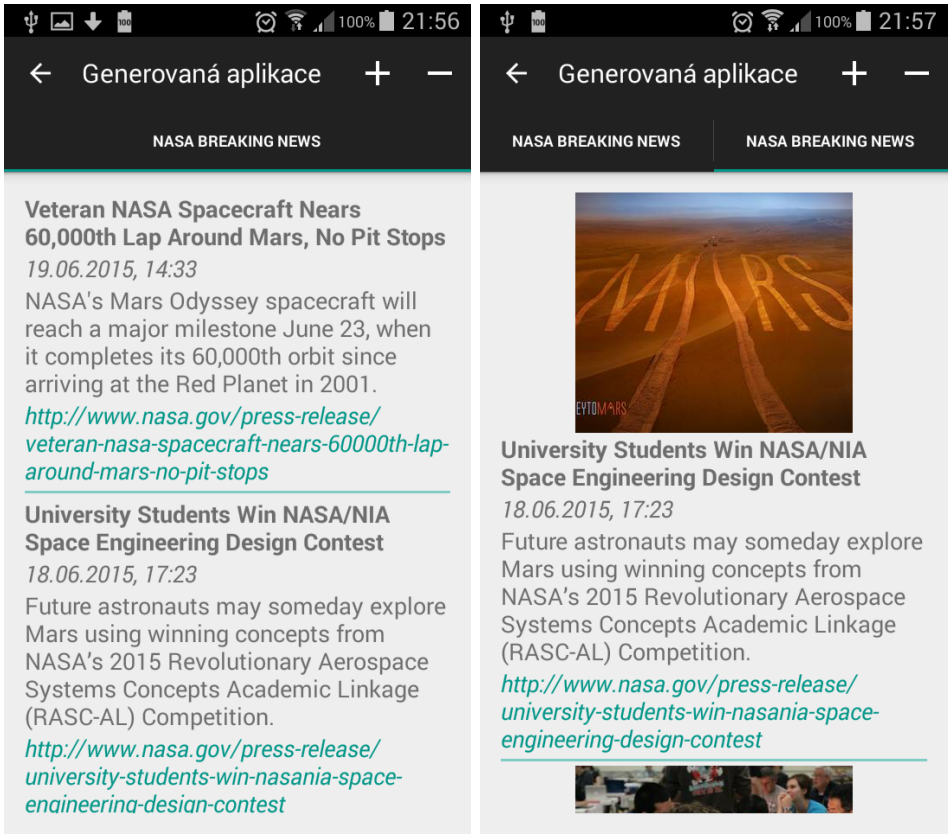


(a) Textové prvky (b) Tlačítka

Obrázek E.5: Příklady rozložení prvků

E.3 RSS Čtečka

Následující snímky představují porovnání použití RSS čtečky bez stahování obrázků (E.6a) a použití se stahováním obrázků (E.6b)



(a) Text (b) S obrázky

Obrázek E.6: Výsledek práce RSS čtečky