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

Diplomová práce

Polymorfní aplikace pro systém Android

Plzeň 2013 Bc. Milan Staffa

Prohlášení

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

V Plzni dne 12. května 2013

Bc. Milan Staffa

Abstract

This thesis deals with the creation of polymorphic application for Android OS, which will change its behavior and graphical user interface according to set rules. The goal of this work is to explore the possibility of local and remote code execution on the Android platform and its use in the polymorphic application. The next task is to create XML rules for this application. The polymorphic application must be made with regard to security. The applicability of the final polymorphic application is tested on a set of generated applications.

Obsah

Poděkování

Rád bych poděkoval Ing. Ladislavu Pešičkovi za vedení a cenné rady při vytváření této diplomové práce.

1 Úvod

Diplomová práce se věnuje problematice tvorby polymorfní aplikace pro mobilní zařízení s operačním systémem Android.

Operační systém Android se v dnešní době stává jedním z nejpopulárnějších a nejrozšířenějších operačních systémů na trhu s mobilními zařízeními. Díky tomu se objevují nové možnosti jeho použití pro tvorbu aplikací s možností jejich rozšíření do praktického využití. Mobilní zařízení jsou dnes i přes rychlý vývoj stále často poměrně nevýkonné (vůči např. notebookům, stolním počítačům, serverům a dalším zařízením) a je proto nutné u vytvářených aplikací brát ohled na výkonnostní nároky aplikací.

Některá zařízení jsou limitována např. velikostí operační paměti (která je u mobilních zařízení často poměrně omezena) a výkonem procesoru. Dalším omezením je vysoká spotřeba energie při vykonávání složitějších operací s nežádoucím vlivem na vybíjení baterie.

Standardní mobilní aplikace má přesně udáno své chování a grafickou podobu. Tato podoba a chování aplikace byla vytvořena vývojářem. Jedna aplikace tak plní svou předem udanou úlohu. V případě, že uživatel potřebuje provést jiný úkon či provádět jiné úlohy, pro které daná aplikace nebyla naprogramována, musí k tomu využít jinou aplikaci. Takovou aplikaci musí někdo naprogramovat a publikovat. Uživatel si ji pak musí stáhnout a nainstalovat. To je důvod, proč je vhodné využít jedné aplikace, která získá po předání nějakého předpisu jinou podobu a chování. Aplikace, která má tyto vlastnosti se nazývá „polymorfní aplikací“. Polymorfní aplikace má oproti standardní aplikaci výhodu. Ta spočívá v tom, že aplikaci stačí jednou nainstalovat a pak lze měnit podobu a chování polymorfní aplikace pouze tím, že jí předáme adresu předpisu definující tyto vlastnosti.

Další výhoda, která lze využít u polymorfní aplikace, je pak zpracovávání operací na vzdáleném zařízení. Náročné operace lze zpracovávat na jiném zařízení a tím snížit čas jejich provádění. S tím jsou také spojené menší nároky na výkonnost mobilního zařízení, protože se zátěž na provedení úlohy přenese na vzdálené zařízení.

Obecně má polymorfní aplikace dáno své chování určitým předpisem. Dle tohoto předpisu se může změnit jak uživatelské rozhraní dané aplikace, tak i její funkčnost. Proto vyvstává otázka, zda za určitých okolností není vhodnější pro vývojáře aplikací vytvářet polymorfní aplikace, které se mohou chovat dle zadaného předpisu, náročnější výpočty se budou provádět vzdáleně a z důvodu úspory paměti si

stahovat pouze části kódu, které potřebují pro svou činnost.

Cílem této diplomové práce je prozkoumat možnosti tvorby takovéto polymorfní aplikace a navrhnout specifikaci předpisu, dle které se bude určovat podoba a chování této aplikace. Pomocí toho také dosáhnout popisu pro vývojáře, jak pro tuto aplikaci vytvářet širokou škálu různých „generovaných aplikací“, fungujících s podporou této polymorfní aplikace.

Dalším úkolem práce je ukázat na vytvořených vzorových aplikací správnost a použitelnost tohoto návrhu a jeho široké možnosti.

2 Polymorfní aplikace

Pro představu toho, co je úkolem této práce je potřeba vysvětlit pojem „polymorfní aplikace“. Většina aplikací, které se pro mobilní operační systém Android používají, se chová dle toho, jak je původní programátor naprogramoval. Jde tedy o to, že aplikace má již od svého vzniku přesně dáno, jak se má chovat a jakou podobu má mít její grafické uživatelské rozhraní.

Polymorfní aplikace jsou založeny na principu, že fungují jako jeden program, který dle získaného předpisu může napodobovat nebo převzít chování s různou funkčností. Jedna aplikace tak může simulovat více aplikací, které by jinak bylo nutné vytvořit samostatně.

Tento princip poskytuje řadu výhod. Jednou z výhod je to, že pro mobilní zařízení je často limitujícím faktorem velikost vnitřní paměti a není tak možné mít vždy všechny potřebné aplikace na jednom mobilním zařízení. Proto se dá využít toho, že veškerá potřebná funkčnost je získávána až dle potřeby.

Další výhoda vyplývá např. z použití v různých organizacích a firmách. Jde o to, že firmy mohou na mobilní zařízení svým zaměstnancům nahrát jednu aplikaci a dle potřeby či funkce zaměstnanců jim dát k dispozici pouze funkčnosti, které pro svou práci potřebují.

Takováto aplikace by tedy mohla mít velké využití jak pro vývojáře, tak i pro zjednodušení práce v mnoha odvětvích.

V rámci určitých projektů dnes vznikají obdobné aplikace, které však myšlenku polymorfismu zpracovávají různým způsobem. Mezi nejrozšířenější typ patří aplikace, jejichž cílem není mít různá chování jedné aplikace, ale možnost, aby jedna aplikace mohla být spuštěna na různých platformách. Tato problematika je založena na myšlence, že některé aplikace, které vzniknou pro např. mobilní operační systém iOS, nejsou dostupné pro platformy Android, Windows Phone a další. Proto se hledá způsob, kterým by bylo možné vytvořit v rámci vývoje jednu danou aplikaci, kterou lze provozovat na různých mobilních zařízeních (respektive různých mobilních operačních systémech) bez úprav nebo celého „nového“ vývoje aplikace.

Uvedená problematika byla například probírána v rámci prezentace „*One CODE to rule them all*“ na konferenci *Apps World* roku 2011 v Cape Town (Jihoafrická republika). Více informací o této prezentaci a jejích výsledcích v [Lin11].

Tímto principem se například zabývají projekty *PhoneGap* (viz [Pho13]), *Rhomobile* (viz [Mot13]), *RAMP* (viz [Vir10]), *Appcelerator Titanium* (viz [App13]) a *J2ME Polish* (viz [Eno13]), které jsou v [Lin11] též zmíněny.

Dalším možným přístupem se zabývalo jedno ze setkání v rámci sady konferencí „*La Degustation Online*“. Na tomto setkání se probírala možnost, zda není místo použití nativních aplikací (tedy takových, které jsou vytvořeny pouze pro jednu danou platformu) či hybridních aplikací (ty dokáží využít hardwarových schopností telefonu a zároveň jsou použitelné na více platformách, musí však být nicméně online) lepší použít mobilní web. Mobilní web je podle výsledků tohoto setkání nejuniverzálnější formou, která je funkční napříč platformami. Jedná se o podobu stránek, jež by měla být přizpůsobena omezením mobilního prohlížeče i malých obrazovek telefonů. Uživatelská přívětivost je pochopitelně závislá na rychlosti připojení, která se liší dle lokality, poskytovatele, druhu datového tarifu atd. Zásadní nevýhoda však vyplývá z nutnosti stálého online připojení k správné funkčnosti a velmi omezené funkčnosti, kterou lze využít pro práci s mobilním telefonem. Více o tomto setkání v [Mic11].

V minulých letech vznikla na Západočeské univerzitě v Plzni bakalářské práce na téma „Dynamická tvorba aplikací v systému Android“ (viz [Hul12]). V této práci se její autor zabýval problematikou toho, zda je možno vytvořit za běhu aplikace její uživatelské rozhraní. Myšlenka o vytvoření aplikace pro mobilní operační systém Android, chovající se dle nějakého předpisu, byla základní pro tuto diplomovou práci zabývající se tvorbou polymorfní aplikace.

Ve zmíněné bakalářské práci však byla navržená a vytvořená mobilní aplikace svou funkčností velmi omezená a z hlediska polymorfismu patřila spíše mezi standardní aplikace, u nichž je chování přesně udáno dopředu.

3 Předpis chování a podoby polymorfní aplikace

Pro vytvoření polymorfní aplikace je nejprve nutno určit, jakým způsobem aplikaci sdělit, jak se má aplikace chovat (respektive jakou činnost má aplikace provádět) a jak má vypadat (jinak řečeno, určit rozložení a podobu ovládacích prvků grafického uživatelského rozhraní aplikace).

K tomuto účelu se dá například využít externího kódu, který bude chování a podobu aplikace předepisovat. Tento kód může být uložen na síti (respektive na serveru) a z ní ho může polymorfní aplikace získat pro dynamické vygenerování svého grafického uživatelského rozhraní.

Pro vytvoření grafického uživatelského rozhraní mobilní aplikace (tedy definování podoby uživatelského rozhraní mobilní aplikace) je třeba nadefinovat, jaké objekty se mají v tomto rozhraní vyskytovat, kde se mají vyskytovat a předepsat jim jejich chování. K takovému popisu se nejvíce hodí „Extensible Markup Language“ (neboli XML), jehož základní vlastností je jeho přesně udaná specifikace. Ta se obecně využívá například pro výměnu dat mezi aplikacemi. Další výhodou XML je jednoduchá čitelnost a přehlednost práce s tímto typem popisu (respektive typem souboru obsahující formát dat daný touto specifikací).

Pomocí XML lze tedy snadno popsat, jak má aplikace vypadat. Vývojářům se nabízí možnost XML dokumenty doplňovat tak, aby si sami dotvořili podobu aplikace dle svých představ a požadavků.

V případě tvorby „form-based“ (respektive formulářové) aplikace stačí z kolekce možných objektů vývojového prostředí Android využít ty nejzákladnější, ze kterých se dá jakákoli formulářová aplikace následně složit. K těmto objektům patří „textview“ (prvek pro popisek), „edittext“ (prvek pro vkládání textu), „button“ (tlačítko) a prvek, do kterého je možno vystupovat grafický výstup (třeba graf).

K „aktivním“ objektům (například tlačítku) je možno v jejich popisu také přiřadit jejich chování při jejich aktivaci (například stisknutí tlačítka). K tomu aby bylo možno provést nějakou akci (respektive operaci), která není součástí původního kódu polymorfní aplikace, je nutno tento zdrojový kód operace někde získat. K tomu lze použít například spuštění vzdáleného kódu ze serveru nebo stáhnutí kódu ze vzdáleného úložiště a jeho následné spuštění. Jsou možné i další varianty. Ty budou zmíněny v dalších kapitolách.

Před definováním objektů je nutno nejprve popsat, jak budou XML soubory zpracovávány. Způsob jejich zpracování je popsán v následující kapitole.

4 Zpracovávání XML

Pro přečtení souboru ve formátu XML, který by mohl sloužit pro popis vzhledu grafického uživatelského rozhraní či chování generované aplikace pro polymorfní aplikaci, je potřeba tzv. *XML parser*. Tedy prostředek pro rozpoznání dat uložených v XML formátu. Z obecného hlediska existují dva základní druhy těchto XML parserů. Každý z nich má své specifika a využívá se k jinému účelu.

Tyto dva druhy jsou tzv. `SAX` a `DOM` parsery.

4.1 DOM rozhraní

DOM rozhraní (z anglické zkratky pojmu „Document Object Model“) je standardem W3C (neboli mezinárodního konsorcia *World Wide Web Consortium*).

Tzv. DOM parser při zpracování dat přistupuje k XML dokumentu jako k stromové datové struktuře (tato technologie se nazývá „grove“ - neboli „Graph Representation Of property ValuEs“). Její princip spočívá v uložení celého parsovaného XML dokumentu do vnitřní paměti, ve které ho lze dále zpracovávat, upravovat a například ukládat do nového XML souboru.

Hlavní výhodou, kromě toho, že lze dokument upravovat, je i náhodný a opakovaný přístup k datům načteným do stromové struktury bez nutnosti opakovaného čtení původního XML souboru.

Nevýhodou tohoto přístupu je však velká paměťová náročnost (z důvodu uloženého XML dokumentu ve vnitřní paměti po celou dobu zpracování).

Tuto nevýhodu řeší tzv. sekvenční model SAX.

Více je tato problematika probrána ve článku viz [Kos01].

4.2 SAX rozhraní

SAX rozhraní neboli „Simple API for XML“ bylo vytvořeno pro sériový přístup k souboru ve formátu XML.

Principem zpracování XML souboru pomocí SAX je tzv. proudové zpracování. Při tomto zpracování dojde k rozdělení XML souboru na jednotlivé části (např. dle počátečních a koncových značek, XML elementů atd.). Následně pak dojde k sekvenčnímu čtení XML souboru po těchto částech a programátor může zpracovávat data dle pořadí, ve kterém se v daném souboru nacházela.

Oproti DOM je SAX varianta často rychlejší na zpracování a paměťově méně náročnější. Paměťová náročnost je nejvíce znatelná u zpracovávání velkých souborů, kde u DOM je ukládáno celé XML do vnitřní paměti. Avšak nevýhodou je to, že celý XML soubor je přečten jednou a to dle pořadí, které je dáno obsahem zpracovávaného souboru. Nelze tedy přistupovat k souboru v náhodném pořadí, jako je tomu u DOM.

Z hlediska mobilních aplikací je pro pouhé získání dat z XML dokumentu lepší použít SAX parser, který tolik nezatěžuje zařízení, u kterých se předpokládá velmi omezená vnitřní paměť a nízký výkon procesoru.

Více je tato problematika probrána ve článku viz [Kos01].

5 Vzdálený kód

Jak bylo řečeno v předchozích kapitolách, lze pro zpracování úloh využít i kódu, který není umístěn přímo v dané aplikaci. Pro použití vzdáleného kódu, je potřeba vyřešit, jak dát tento kód aplikaci k dispozici. Vzdálený kód lze například uchovávat v podobě knihoven, které se stáhnou ze vzdáleného úložiště a za běhu aplikace provedou svůj kód. Další možností je kód poskytovat na vzdálených zařízeních, která kód provedou a vrátí zpět výsledek.

Pro návrh popsaného principu připadají v úvahu tři základní varianty řešení, které budou popsány v následujících podkapitolách.

5.1 Varianta volání vzdálených procedur

RPC neboli „Remote procedure call“ znamená v češtině „Vzdálené volání procedur“. Pro mobilní zařízení jsou některé vykonávané funkce velmi náročné na výpočetní složitost a to se může projevit jak na poklesu výkonu zařízení tak i například na spotřebě energie v případě práce zařízení přes baterii. Proto může být výhodnější zpracovat požadovaný kód (respektive výpočet) na vzdáleném zařízení (např. na specializovaném výpočetním serveru) a zpět poslat výsledek operace.

Jde o obdobu volání lokální funkce. Volání těchto funkcí je založeno na modelu klient-server, kde server může být umístěn na vzdáleném zařízení v síti. Klient-server model v tomto případě je založen na tom, že klient iniciuje komunikaci (respektive posílá zprávu serveru) a po odeslání očekává od serveru zprávu s odpovědí. Pro volání služeb je potřeba zprávu s předávanými parametry pro přijímající stranu zabalit.

Princip RPC je tento:

1. Na straně klienta nejprve dojde k zabalení odesílaných dat (parametrů). Tento krok se nazývá jako tzv. „marshalling“.
2. Klient provede odeslání zabalených dat serveru.
3. Na straně zařízení zpracovávajícího data dojde k příjmu dat odeslaných klientem.
4. Server provede rozbalení příchozích dat. Tato operace se označuje jako tzv. „unmarshalling“.

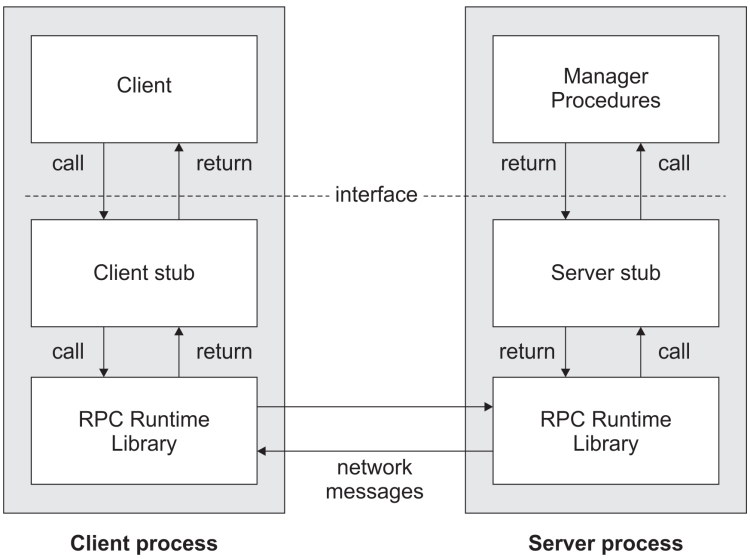
5. Server podle přijaté zprávy zajistí volání správné procedury (jejíž identifikace se přenáší v hlavičce přenášené zprávy).
6. Server data zpracuje a připraví odpověď pro klienta.
7. Server data s výsledkem operace zabalí a připraví k odeslání.
8. Server data s odpovědí/výsledkem odešle klientovi.
9. Klient, který operaci původně požadoval přijme data od serveru.
10. Klient rozbalí data a zpracuje je.

Celá komunikace probíhá přes spojky programu. Ty se označují jako tzv. „stub“. „Stub“ je komunikačním rozhraním, které RPC protokol implementuje.

Každý z prvků server-klient obsahuje jednu spojku. Na straně serveru dochází k přiřazení požadované procedury tak, že z vnějšího pohledu vypadá toto chování podobně jako při volání lokálních procedur.

Nevýhoda RPC však spočívá v tom, že pro přenos klient-server a server-klient je možno použít pouze základních datových typů jazyka. Přenos složitějších struktur prostřednictvím RPC lze realizovat vytvořením vlastních struktur.

Princip činnosti RPC je blíže ukázán na obrázku 5.1.



Obrázek 5.1: Schéma činnosti RPC. Obrázek vytvořen dle obrázku z [Jav12].

Varianty RPC jsou popsány v následujících podkapitolách.

5.1.1 XML-RPC

Protokol „XML-RPC“ slouží především pro vzdálené komunikace s webovými službami. Je založen na modelu klient-server, kde klient iniciuje komunikaci.

Jak název protokolu napovídá, jde o protokol, ve kterém jsou data zapouzdřena do XML. Ta jsou pak posílána síťovým protokolem HTTP (resp. pomocí tzv. HTTP-POST) na daný server.

Server na základě volání provede požadovanou proceduru a zpět pomocí stejného mechanismu posílá klientovi data s odpovědí (nebo informace o chybě).

Ukázka kódu zprávy v XML-RPC (převzato z [Sci11]), kde se přenáší parametr typu `integer` (tedy celé číslo) pro zpracování procedurou (v tomto případě `jmenoMetody`), která se nachází ve třídě (v tomto případě `trida`) na straně serveru:

```
POST /server HTTP/1.0
User-Agent: Mozilla/5.0
Host: xmlrpc.example.com
Content-Type: text/xml
Content-length: 314
<?xml version="1.0"?>
<methodCall>
  <methodName>trida.jmenoMetody</methodName>
  <params>
    <param>
      <value>
         <int>
           250
         </int>
       </value>
    </param>
  </params>
</methodCall>
```

Odpověď pak může vypadat například takto:

```
HTTP/1.0 200 OK
Connection: close
Content-Length: 144
Content-Type: text/xml
Date: Wed, 13 Apr 2013 12:00:00 GMT
Server: ZCU client/1.0
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>
          Retezec s vysledkem operace
        </string>
      </value>
    </param>
  </params>
</methodResponse>
```

5.1.2 JSON-RPC

Protokol „JSON-RPC“ (neboli „JavaScript Object Notation - Remote procedure call“) je založen na zakódování zprávy pomocí „JSON“. Jde o textový formát (nezávislý na použitém jazyce) určený pro výměnu dat, který je založen na podmnožině jazyka *JavaScript* dle standardu „ECMA-262“ (více o tomto standardu v [Ecm11]).

JSON využívá dvou typů struktur. Prvním typem je kolekce párů název-hodnota. Ta bývá v jazycích často realizována *objektem*, záznamem (*record*) nebo stukturou (*struct*).

Druhým typem struktur pro JSON jsou tříděné seznamy hodnot. V jazycích je často realizován jako pole (*array*), vektor (*vector*), seznam (*list*) a nebo posloupnost (*sequence*). Citace viz [Jso13].

Komunikace v rámci JSON-RPC probíhá obdobně jako je tomu u XML-RPC. Tedy voláním procedury dojde k odeslání zprávy serveru (který implementuje JSON-RPC protokol) přes TCP/IP socket. V případě nedostupnosti TCP/IP se může

použít HTTP protokol. Klient svým voláním v jednu chvíli volá jednu zvolenou metodu. V případě, že má dané volání více vstupních parametrů, dojde k jejich předání dané metodě formou *objektu* nebo *pole* (viz výše).

Obdobným způsobem se předává zpět klientovi výsledek operace provedený metodou na straně serveru (lze tedy navracet více výstupních parametrů výsledku zpracované metody).

V případě posílání zprávy s požadavkem na metodu (respektive odeslání požadavku na metodu serveru) jde o zasílání jednoho (serializovaného) JSON objektu. Objekt obsahuje následující parametry:

- `methods` - řetězec (*string*) jednoznačně identifikující metodu, která je požadována pro provedené operace.
- `params` - pole objektů, které se mají použít při zpracování metody jako její vstupní parametry.
- `id` - hodnota jakéhokoli typu, která slouží ke spárování dotazu na server a odpovědi klientovi (respektive jde o to, aby klient správně rozpoznal odpověď na jeho volání vzdálené procedury).

Odesílaná zpráva by pak mohla pro příklad vypadat následovně:

```
{"method": "echo", "params": ["Hello JSON-RPC"], "id": 1}
```

Po příjmu požadavku a jeho zpracování je zpět poslána odpověď s výsledkem operace. Odpověď nese následující parametry:

- `result` - data s výsledkem operace. V případě chyby zpracování se v tomto parametru vrací zpět hodnota *null*.
- `error` - specifikace pomocí chybového kódu o jakou šlo chybu. V případě, že byla činnost vzdálené procedury úspěšná, je v tomto parametru navrácena hodnota *null*.
- `id` - hodnota, která pomáhá klientovi spárovat dotaz-odpověď ze serveru.

Odpověď navazující na předchozí ukázkový příklad by pak tedy mohla vypadat následovně:

```
{"result": "Hello JSON-RPC", "error": null, "id": 1}
```

V JSON-RPC existuje ještě jeden speciální druh odesílané zprávy pro případ, že se na server odesílá zpráva (žádost), k níž se neočekává odpověď ze strany serveru. Tato zpráva je specifická tím, že v parametru `id` obsahuje hodnotu `null` (jelikož v tomto případě se neočekává odpověď a tím pádem není s čím zprávu spárovat).

Ukázkové příklady byly vytvořeny na základě tutoriálu z [W3j99].

Pro mobilní operační systém Android existuje open-source knihovna „android-json-rpc“. Tato knihovna je k dispozici pod *MIT* licencí. Další informace o této knihovně viz [Goo09].

5.1.3 SOAP

Protokol „SOAP“ (z anglického pojmu „Simple Object Access Protocol“) vychází z principu výměny zpráv přes síť. K tomu je často využíváno internetového protokolu „HTTP“ (neboli „Hypertext Transfer Protocol“ - viz [Int99]).

Data jsou u protokolu SOAP zapouzdřena v jazyce XML. Pro komunikaci se nejčastěji používá RPC s modelem klient-server.

Princip činnosti je následující:

1. Klientská aplikace odešle zprávu s požadavky na vykonávanou proceduru ve formátu XML na server.
2. Server zprávu přijme.
3. Na serveru dojde k vykonání volané vzdálené procedury.
4. Na straně serveru je připravena odpověď v XML formátu.
5. Server pošle klientovi zprávu s odpovědí.
6. Klientská aplikace zprávu od severu přijme a zpracuje.

SOAP zprávy mají strukturu XML dokumentu. Kořenovým elementem je element `Envelope`. V tomto kořenovém elementu jsou další dva elementy:

- `Header` - tzv. hlavička je nepovinným elementem. Slouží pouze jako doplňková informace, která lze použít pro zpracování zprávy. Může být použita například pro identifikaci uživatele apod.
- `Body` - tzv. tělo je povinným parametrem, který identifikuje volané služby a parametry, které mají být volané proceduře předány.

Ukázka zprávy s požadavky posílaných klientskou aplikací na server:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/
  2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/
  2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

Odpověď od serveru klientské aplikaci by pak mohla mít následující podobu:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/
  2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/
  2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>\textbf{34.5}</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

SOAP je pro Android dostupný například v podobě knihovny `ksoap2-android` (viz [Goo10]). Ta je k dispozici pod MIT licencí.

Citace této kapitoly a příkladů viz [W3s99].

5.1.4 REST

„REST“ (neboli z anglického „Representational State Transfer“) je architektura rozhraní, navržená pro distribuované prostředí. Pojem REST byl poprvé představen v roce 2000 v disertační práci Roye Fieldinga, jednoho ze spoluautorů protokolu HTTP ([Int99]).

REST je oproti již zmíněným XML-RPC (viz kapitola 5.1.1) či SOAP (viz kapitola 5.1.3) orientován datově nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům.

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (tzv. „resources“). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze

popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI (Uniform Resource Identifier neboli „jednotný identifikátor zdroje“) a REST definuje čtyři základní metody pro přístup k nim.

Tyto metody se označují jako „CRUD“. Označení vzniklo spojením prvních písmen názvů jednotlivých metod:

- ***C**reate* - metoda pro vytvoření dat. Tato metoda je implementována HTTP protokolem pomocí metody `POST`.
- ***R**etrieve* - metoda pro získání požadovaných dat. Tato metoda je implementována HTTP protokolem pomocí metody `GET`.
- ***U**pdate* - metoda pro změnu dat. Tato metoda je implementována HTTP protokolem pomocí metody `PUT`.
- ***D**elete* - metoda pro mazání dat. Tato metoda je implementována HTTP protokolem pomocí metody `DELETE`.

Zdroji mohou být například formáty XML, JSON, RSS a ATOM. Citace viz [Mal09].

Řešení REST je k dispozici i pro vývoj aplikací v platformě Android, které přistupují pomocí rozhraní REST k datům webových služeb. Více o tomto řešení v [Dob10].

5.2 Varianta distribuovaných objektů

Tato varianta využívá distribuovaných objektů. Pomocí mechanismu zástupných objektů (tzv. *proxy*) lze přistupovat ke vzdáleným objektům stejným způsobem jako k lokálním. Více o distribuovaných objektech viz [Ocs10].

5.2.1 Java RMI

Java RMI (neboli „Java Remote Method Invocation“) je první technologií programovacího jazyka Java umožňující volat metody objektů nacházející se na jiném JVM (neboli „Java Virtual Machine“).

Jde o architekturu klient-server, kde úkolem serveru je vytvářet objekty a poskytovat reference na ně, tak aby byly vzdáleně přístupné. Server po vytvoření těchto objektů čeká, až budou tyto objekty volány klientem (respektive klientskou aplikací).

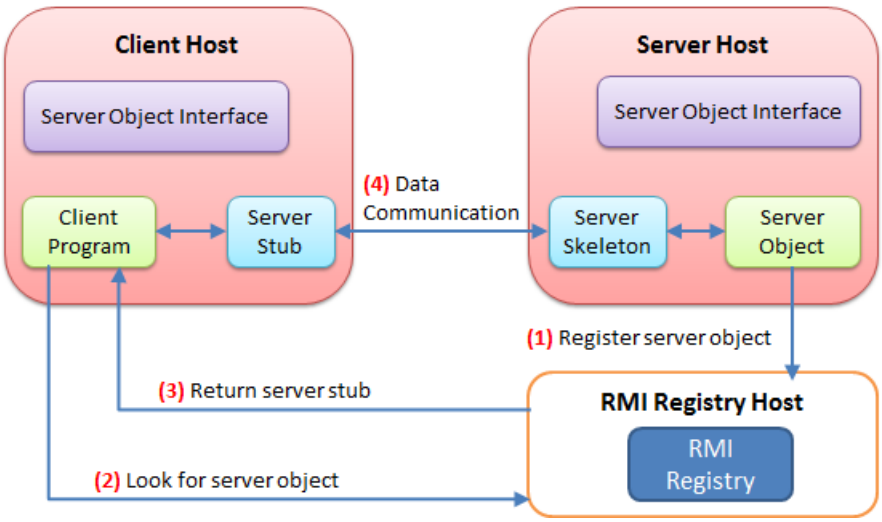
Klient (respektive klientská aplikace) vlastní reference na objekty vytvořené serverem, aby mohl vzdáleně volat jejich metody. Reference jsou v RMI umístěny v registru, přičemž server pomocí tohoto registru zajišťuje, aby byla asociována jména se vzdálenými objekty.

V případě, že klient chce volat metodu vzdáleného objektu, tak si ji vyhledá dle jména objektu v registru a vykoná volání této metody.

Volání je zprostředkováno obdobně jako tomu je u RPC pomocí lokálních zástupných objektů, neboli spojek (*stub*). Princip Java RMI je ukázán na obrázku 5.2, kde čísla v obrázku udávají pořadí prováděných operací. Více o tomto tématu viz [Lyc11].

Java RMI je dostupné i pro mobilní operační systém Android. K tomu lze využít knihovnu *android-xmlrpc*. Ta je šířena v rámci licence *Apache License 2.0*.

Zdroj informací pro tuto kapitolu byl využit viz [Lyc11, Sch13].



Obrázek 5.2: Ukázka principu Java RMI. Obrázek převzat z [Lyc11].

5.2.2 CORBA

Standard „CORBA“ (z anglického „Common Object Request Broker Architecture“) byla definována skupinou OMG (neboli „Object Management Group“) v říjnu roku 1991. Tento standard slouží pro podporu tvorby distribuovaných objektově orientovaných aplikací. Je navržen tak, aby komponenty napsané v různých programovacích jazycích a běžících na různých počítačích měli možnost spolu komunikovat.

Komunikace v CORBA je navržena na modelu klient-server. Klientská aplikace pomocí tzv. ORB (neboli „Object Request Broker“) přistupuje k službám objektů zasíláním požadavků. ORB obsahuje mechanismy pro vyhledání požadovaného objektu (na straně serveru), generování požadavků, jejich přenos z klientské aplikace na server a přenos výsledku zpět do klientské aplikace (citace viz [Hul12]).

Každý objekt CORBA, který je poskytován serverem má definované rozhraní v tzv. IDL (neboli „Interface Definition Language“), které specifikuje, jaké objekty jsou na straně serveru přístupné klientským aplikacím.

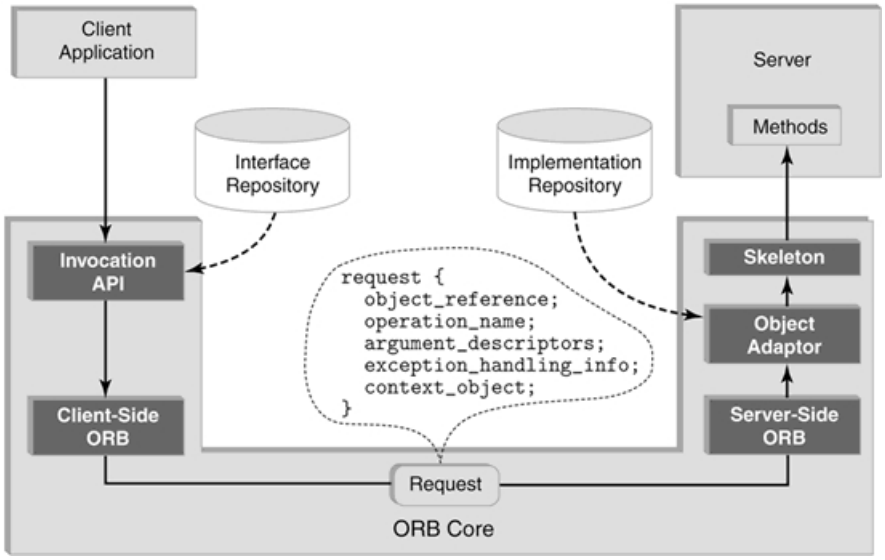
Při vývoji s CORBA je potřeba nejprve nadefinovat používané rozhraní pomocí IDL. Po kompilaci rozhraní pak dojde k vytvoření Java třídy, klientských spojek (tzv. „stub“), které slouží k propojení klientského kódu s ORB agentem. Dále je také vytvořena kostra objektu (tzv. „skeleton“). Ta slouží jako bázová třída odpovídající definici objektu v IDL.

Při volání funkce objektu je volání přesunuto přes klientskou spojku do ORB. Ten pošle požadavek kostře volaného vzdáleného objektu. Popsaný princip je vidět na obrázku 5.3.

Hlavní nevýhodou tohoto řešení je to, že CORBA využívá komunikace na síťovém portu, který často bývá na firewallech zakázán a je nutné ho manuálně na daném firewallu povolit.

Pro platformu Android existuje řešení ve formě například knihovny `TIDorb` (viz [Gar11]) šířený pod GPL a Affero-GPL licencemi pro nekomerční použití.

Zdroje použité pro tvorbu této kapitoly viz [Rho09, Cof13]. Specifikace a další informace o aktuální verzi `CORBA 3.3` v [Cos13].



Obrázek 5.3: Ukázka principu funkce CORBA. Obrázek převzat z [Rho09].

5.2.3 DCOM

DCOM (z anglického „Distributed Component Object Model“) je proprietární technologie vyvíjená společností *Microsoft*. Tato technologie umožňuje komunikovat různým počítačům (primárně s operačním systémem *Microsoft Windows*) spolu po síti. Vyskytuje se i pro jiné platformy v podobě implementace v komerčních produktech.

Pro programovací jazyk Java existuje například knihovna `j-Interop` implementující DCOM protokol pod LGPL licencí. Nevýhodou je však, že DCOM nekomunikuje prostřednictvím internetu. Více o knihovně `j-Interop` viz [Jin13].

Citace této kapitoly použity z [Hul12].

Pro platformu Android bohužel stále neexistuje DCOM podpora. Více o této technologii v [Msd07].

5.3 Varianta kódu získaného ze vzdáleného úložiště

Další variantou je možnost stažení funkčního kódu ze vzdáleného úložiště. Využívat lze tak pouze kód, který je zrovna potřeba k vykonávání daného požadavku. Toho

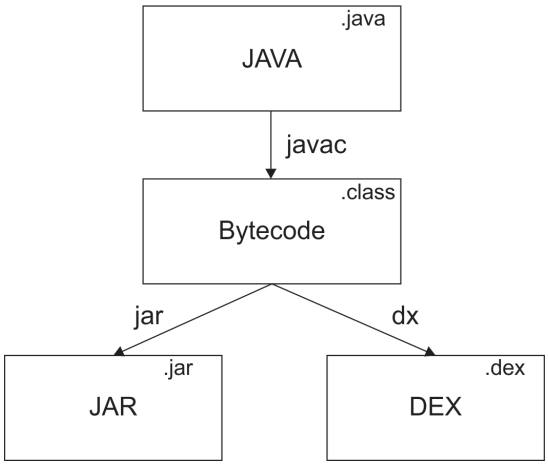
je možno dosáhnout vytvořením knihovny zabalené v `jar` souboru (více o formátu `jar` v [Ibm13]).

Program, který by měl mít všeobecnou funkčnost (a tedy by jeho velikost byla teoreticky nekonečná), je tak možno rozdělit na velké množství relativně malých knihoven s požadovanými funkcemi.

Ve vývojovém prostředí *Java* pro *Android* lze využít tzv. DEX neboli „Dalvik EXecutable“. Jde o platformu, kde java třídy `.class` lze zkompilovat do `dex` formátu. Ten není tolik paměťově náročný jako java kód pro JVM (*Java Virtual Machine*). Rozdíl mezi klasickým `jar` a `dex` ve fázi překladu je vidět na obrázku 5.4.

Se standardním `jar` má `dex` ve fázi překladu souborů ve formátu `.java` společný překlad do `bytecode` (spustitelná mezikód ve formát `.class`). Překlad zajišťuje Java kompilátor `javac`. Získaný soubor s `bytecode` je pak archivován pro normální Javu do formátu `.jar`. Pro `dex` je však nutno z `Java bytecode` provést ještě kompilaci na `Dalvik bytecode`. K provedení kompilace na `Dalvik bytecode` existuje nástroj nazvaný `dx`.

Formát `dex` využívá operační systém místo použití standardního javovského `.class` pro úsporu paměti a optimalizaci na mobilní zařízení. Obecně uváděná komprese kódu je `dex` : `jar` přibližně 0.44 : 1 (viz [Sra08]), což je poměrně velká úspora uloženého kódu v paměti.



Obrázek 5.4: Schéma rozdílu mezi překladem `jar` a `dex`. Obrázek vytvořen na základě obrázku z [Scr11].

Pro Android existuje možnost, jak načíst Java třídy `.class` (uložené v archivu `.jar`) za běhu aplikace. Realizace tohoto načítání je detailně popsána v [Chu11].

Při použití je však nutno dbát na to, aby nebyly použity žádné knihovny, které nejsou podporovány systémem Android.

6 Android

Mobilní operační systém Android je open source platforma založená na jádru Linuxu vyvíjena společností Google. Primárním účelem je nasazení na chytré mobilní telefony, tablety a obdobná zařízení.

6.1 Verze operačního systému Android

Mobilní operační systém Android byl od počátku svého vývoje rozlišován třemi druhy značení. Prvním je značení tzv. *API level* (z anglického „Application Programming Interface“). Toto označení je dáno číslem, kde vyšší číslo udává novější verzi. Značení je klíčové především pro vývojáře, kterým udává, které funkčnosti a prostředky mohou pro vývoj v dané verzi použít a které jsou již nedoporučené (tzv. „deprecated“).

Druhý typ označení udává verzi systému. Značení je založeno na dvou a více číslech oddělených tečkou. První číslo udává hlavní verzi a další dodatková čísla specifikují přesně verzi daného systému.

Posledním typem značení je kódové značení. Toto značení je pojmenováním hlavních verzí systému. Jména se přiřazují dle anglických kulinářských názvů sladkostí a dezertů.

Přehled verzí seřazený od nejaktuálnějších je vidět v tabulce 6.1. Verze operačního systému, které mají v aktivních zařízeních zastoupení méně než 0.1% v tabulce uvedeny nejsou.

Aktivních zařízení s Android s API level rovnému 7 a méně je k datu 2. dubna 2013 dle zdrojů *Google* v rámci statistik obnovovaných každých 14 dní poměrově k ostatním verzím 1.8%. Z průběžných poznatků je možno pozorovat rapidní úbytek zařízení s nižšími verzemi. Pro ty už je také podporováno čím dál méně nově vyvíjených aplikací.

Pro vývoj polymorfní aplikace byla zvolena minimální verze 4.0 (odpovídající API level 14). Tato verze je nasazována na většinu nových mobilních zařízení. Díky rychlému vývoji nových verzí systému Android bude dostupných čím dál více zařízení s vyššími verzemi tohoto systému.

API level	Verze	Kódové označení
17	4.2.x	Jelly Bean
16	4.1.x	Jelly Bean
15	4.0.x	Ice Cream Sandwich
13	3.2	Honeycomb
12	3.1	Honeycomb
10	2.3.3 až 2.3.7	Gingerbread
9	2.3 až 2.3.2	Gingerbread
8	2.2	Froyo
7	2.0 až 2.1	Eclair
4	1.6	Donut

Tabulka 6.1: Seznam vybraných verzí operačního systému Android.

Více o aktuálních statistikách aktivních zařízení, které byly použity pro tento text, je možno vidět v [Ded13].

6.2 Vývojové prostředí

Pro vývoj pro mobilní operační systém lze využít celou řadu vývojových prostředků, mezi které patří například *Adobe AIR* či *App Inventor*. Velmi často je také využíváno vývojové prostředí *Eclipse*, které je možno stáhnout včetně všech potřebných knihoven a balíků (tedy včetně celého SDK - *Software development kit*, respektive ADT - *Android Development Toolkit*) přímo z oficiálních stránek Android (viz [Sdk13]).

Tento vývojový nástroj poskytuje mnoho možností vývoje, včetně poměrně dobře zpracovaného emulátoru, dovolující vývojáři testovat vytvářené aplikace na emulovaných zařízeních různých druhů s různými parametry. Díky této vlastnosti se dá odchytit většina nedostatků, které by se mohly projevit na různých konfiguracích zařízeních se systémem Android včetně simulování různých nepředvídatelných situací, které mohou běh aplikace narušit.

Jazyk, ve kterém se pro operační systém Android vyvíjí aplikace, je Java, přičemž některé knihovny (např. pro `Swing` a `ADT`) nejsou k dispozici. Pro tvorbu uživatelského rozhraní jsou využity vlastní knihovny (nedostupné v `Java Standard Edition`). Dále jsou pro vývoj aplikací využívajících síťových prostředků k dispozici knihovny `Apache`.

Celá polymorfní aplikace pro tuto práci byla vyvíjena ve vývojovém prostředí *Eclipse Juno 4.2.0* s ADT (verze 20.0.0.v201206242043-391819).

6.3 Zajištění bezpečnosti v systému Android

Bezpečnost v mobilních aplikacích je velmi důležitým tématem. Z hlediska dnešní doby, kdy jsou časté hrozby ze strany vnějších útoků na mobilní zařízení, existuje i např. možnost podvrhnutí aplikace (nebo její určité části), která se tváří jako „bezpečná a užitečná“ tzv. „škodlivým kódem“.

Tento problém se může pro příklad vyskytnou v aplikacích zajišťujících například internetové bankovnictví, kde může dojít k vážné finanční ztrátě uživatele, způsobené zneužitím osobních/přihlašovacích údajů. Další způsob zneužití se může vyskytnout například u posílání prémiových SMS zpráv bez vědomí uživatele atd.

Problémy s bezpečností však mohou nastat i u aplikací, jejichž udávaná funkčnost nemá na první pohled nic společného s možným nebezpečím.

Vývojáři systému Android navrhli způsob zabránění těmto útokům. Pro každou aplikaci, jež chce její vývojář pro systém Android napsat, musí na začátku definovat oprávnění, které tato aplikace smí využívat. Tato oprávnění jsou zařazena do souboru (v XML formátu), který se označuje jako tzv. „Android Manifest“. Oprávnění, která je možno pro aplikaci vyžádat, je velká řada.

V případě, že vývojář aplikace chce použít některou z akcí či pracovat s nějakými prostředky a nevyžádá si pro ně dané oprávnění, reaguje Android na takovýto pokus znepřístupněním daných zdrojů. Pro příklad, chce li vývojář, aby jeho aplikace mohla zapisovat do externího úložiště (kterým může být například v zařízení vložená SD karta) a nezadá-li do manifestu požadované oprávnění, Android nedovolí této aplikaci jakýkoli zápis a vrací zpět výjimku (z anglického *Exception*).

Pro ukázku možných oprávnění, jsou zde ukázány ty nejčastěji používané:

- `android.permission.WRITE_EXTERNAL_STORAGE`
  - pro povolení zápisu do externí paměti.
- `android.permission.READ_EXTERNAL_STORAGE`
  - pro povolení čtení z externí paměti.

- `android.permission.WRITE_SMS`
  - pro povolené napsání SMS zprávy.
- `android.permission.READ_SMS`
  - pro povolení čtení SMS zprávy.
- `android.permission.ACCESS_NETWORK_STATE`
  - pro zjišťování stavu síťového připojení.
- `android.permission.INTERNET`
  - pro povolení přístupu aplikace k síti.
- `android.permission.INSTALL_PACKAGES`
  - pro povolení instalování nových balíčků (aplikací).

Další možná oprávnění je možno vidět v [Dev13] a jejich použití viz například [Mur11].

Princip ochrany uživatele a mobilního zařízení, na kterém má daná aplikace běžet, je založen na tom, že při instalaci dané aplikace musí uživatel potvrdit, že s danými oprávněními souhlasí. Dále pak uživateli nezbývá, než důvěřovat vývojáři aplikace, že vložené důvěry nezneužije.

Jednou z mála možností pasivní ochrany proti zneužití je logická kontrola. Ta funguje tak, že uživatel, který instaluje aplikaci na zařízení, má možnost potvrdit souhlas použití práv pro aplikaci. Uživatel má možnost posoudit, zda je instalovaná aplikace podezřelá. Pro příklad aplikace, která má funkci kalkulačky je podezřelá v případě, když požaduje oprávnění pro odesílání „SMS“ nebo přístup ke kontaktům uživatele.

V rámci aktivní ochrany dnes již existují antivirové programy, které dokáží detekovat nebezpečný kód (či chování) aplikace. Tyto aplikace fungují tak, že čerpají znalosti z aplikací, které již na jiných zařízení způsobily problémy a varují před nimi uživatele již ve fázi pokusu o instalaci aplikace.

V polymorfní aplikaci tato problematika nabírá velkých rozměrů. Problém je v tom, že pokud je požadavkem vytvořit aplikaci, která bude umět širokou škálu operací (funkčností), tak takováto aplikace bude také potřebovat velký rozsah dostupných oprávnění. Pro uživatele tedy bude při instalaci potvrzení oprávnění pro všechny aplikace matoucí a ke všemu takto ztrácí veškerou kontrolu nad tím, co bude moci daná aplikace na jeho zařízení dělat.

Jedním z možných řešení je vytvoření různých verzí polymorfní aplikace. Například by mohly existovat dvě verze polymorfní aplikace, kde jedna by měla například pouze základní práva, která by teoreticky nemohla nějakým způsobem poškodit jak mobilní zařízení tak i jeho uživatele.

Druhá verze polymorfní aplikace by pak měla všechny možná práva, která by se dala teoreticky pro aplikaci smysluplně využít a uživatel (a jeho mobilní zařízení) by pak museli spoléhat na důvěryhodnost poskytovatele částí aplikace (respektive na vývojáře a poskytovatele „podaplikací“).

7 Realizace polymorfní aplikace

V rámci této diplomové práce bylo navrženo a realizováno řešení polymorfní aplikace pro mobilní operační systém Android. V následujícím textu je popsán tento návrh a řešení.

7.1 Rozvržení polymorfní aplikace

Pro vývoj aplikací pro Android se využívá tzv. „aktivit“. Aktivita je programový kód, který definuje „jednu zobrazovanou obrazovku“ včetně veškerého chování v rámci teto obrazovky. Aktivitu lze považovat za programovou entitu, která slouží k interakci s uživatelem.

Jedna aplikace se obecně může skládat z více jednotlivých aktivit. Jedna z těchto aktivit je v „manifestu“ aplikace pak označena jako ta, která se má spustit při spuštění aplikace. Z této jsou pak ostatní vyvolávány dle kódu aktuálně spuštěné aktivity.

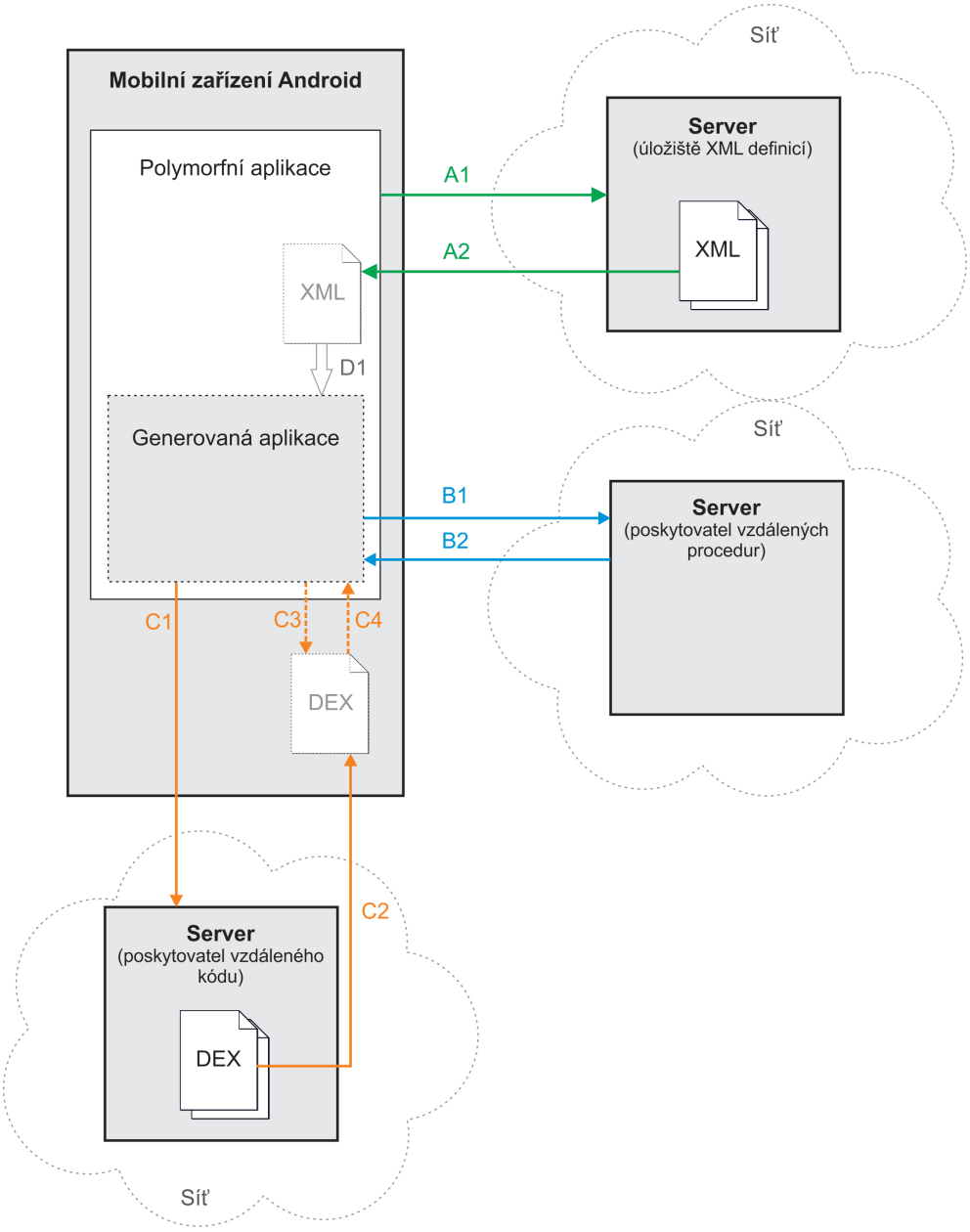
Pro polymorfní aplikaci této práce bylo navrženo prostředí skládající se ze dvou základních aktivit.

První aktivitou je tzv. „hlavní aktivita“, jejímž úkolem je dát uživateli možnost si zvolit požadovanou „generovanou aplikaci“ (respektive aktivitu, jejíž chování a grafické uživatelské rozhraní bude udáno XML souborem s daným popisem), která bude běžet v rámci druhé připravené aktivity.

Úkolem druhého typu aktivity je reprezentovat „generovanou aplikaci“ dle zadaného XML předpisu poskytnutého první („hlavní“) aktivitou.

V rámci dalšího textu bude pro popis vytvořené polymorfní aplikace využit referenční obrázek 7.1. Na zmíněném obrázku je vidět schéma popisující, jak polymorfní aplikace funguje. Šipky v tomto obrázku udávají komunikaci polymorfní aplikace a budou dále v textu vysvětleny.

V následující části jsou popsány jednotlivé aktivity realizované polymorfní aplikace.



Obrázek 7.1: Schéma činnosti polymorfní aplikace.

7.2 Hlavní aktivita

Úkolem hlavní aktivity je dát uživateli možnost zvolit si požadované chování a podobu grafického uživatelského rozhraní aplikace (v jiném slova smyslu jde o výběr „generované aplikace“).

K tomu bylo v rámci hlavní aktivity navrženo okno s jednou nabídkou a dvěmi tlačítky.

Pro dodržení požadavku co největší variability polymorfní aplikace a možnost jejího snadného budoucího rozšíření, nebyla aplikace omezována pouze na přednastavené vzorové „generované aplikace“. Byla zvolena varianta, ve které pomocí uživatelem zadávané *URL* adresy se vzdáleným XML souborem popisujícím podobu a funkčnost aplikace, byla dána možnost uživateli vybrat si, jakou aplikaci chce spustit (ze zadané URL adresy).

Toto řešení bylo zvoleno, aby vývojáři „generovaných aplikací“ nebyly omezeni, jako by tomu bylo v případě, že by veškerá funkčnost byla předem v aplikaci udaná jejím původním vývojářem. Uživateli tedy stačí zadat správnou adresu s XML souborem a může aplikaci začít používat.

Hlavní aktivita vychází z práce [Hul12], kde se však její autor omezil pouze na dynamickou tvorbu rozhraní. Autor dále nepředpokládal využití vytvořené generované aplikace bez nutnosti úprav kódu samotné původní aplikace. Ta byla vytvořena a nastavena pouze na míru dvěma ukázkovým aplikacím („kalkulačka“ s jednoduchými funkcemi a „převodník teplot“). Pro další možné „podoby“ aplikace by bylo nutné změnit velkou část zdrojového kódu samotné aplikace. Tato úprava by obnášela vložení veškeré požadované funkčnosti přímo do kódu aplikace.

Hlavní aktivita vychází ze zmíněné myšlenky, avšak je dovedena k větší obecnosti s tím, že navíc podporuje myšlenku zpracování kódu bez nutnosti jeho umístění v původním zdrojovém kódu aplikace samotné.

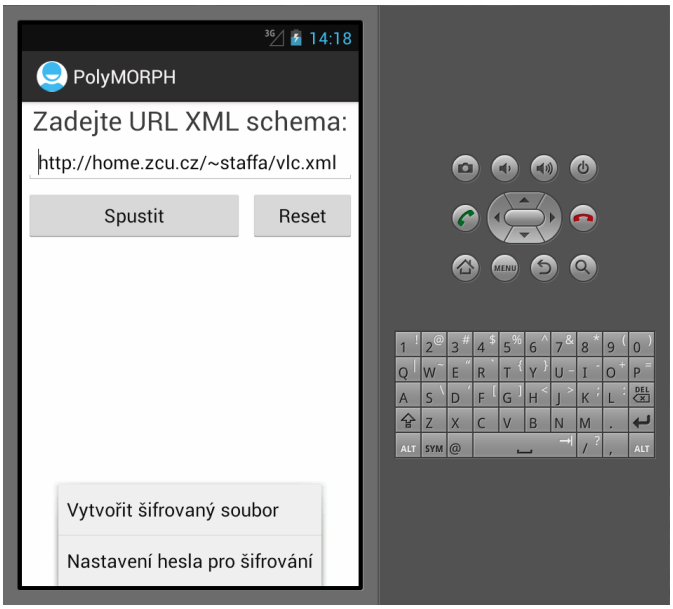
Funkce této aktivity se skládá z několika fází.

V první fázi je kontrolováno připojení k síti. K tomu dochází, aby se zkontrolovala možnost stáhnutí požadovaného XML souboru s popisem generované aplikace a případně dalších potřebných souborů. Po tom, co uživatel zadá *URL* (z anglického „Uniform Resource Locator“) adresu s požadovaným XML souborem, se tato adresa uloží do tzv. „sdílených preferencí“, odkud ji bude možno další aktivitou přečíst.

V další fázi se spustí nová aktivita (reprezentující „generovanou aplikaci“). Spu-

štění této aktivity je vidět v následujícím ukázkovém zdrojovém kódu:

```
Intent intent
  = new Intent(MainActivity.this, WorkClass.class);
startActivity(intent);
```



Obrázek 7.2: Ukázka grafického uživatelského rozhraní hlavní aktivity spuštěné v emulátoru.

Hlavní aktivita je z pohledu grafického uživatelského rozhraní složena z pole pro editovatelný text a dvou tlačítek. Do editovatelného pole se zadává adresa požadovaného XML souboru a tlačítkem *Spustit* se aktivita daná tímto XML spustí. Tlačítko *Reset* nastavuje přednastavené XML do editovatelného pole. Další prvky slouží k zabezpečení a budou popsány v kapitole 8. Jak tato aktivita vypadá při spuštění v emulátoru mobilních zařízení, je vidět na obrázku 7.2.

7.3 Aktivita pro „generované aplikace“

Při spuštění této aktivity dojde ke zjištění adresy XML souboru uložené hlavní aktivitou. Následně je provedeno zkontrolování dostupnosti XML souboru na adrese jeho přečtením.

V obrázku 7.1 je získání XML souboru znázorněno šipkami označenými `A1` a `A2`.

Dle popisu získaného z tohoto XML souboru dojde k vytvoření uživatelského prostředí v rámci obrazovky dané aktivity (v obrázku 7.1 znázorněno šipkou označenou jako `D1`.

Aktivita dle získaného XML popisu také zařizuje, aby v případě, že se nachází v popisu tlačítko (či více tlačítek), byly k dispozici požadované zdroje. Zdroje jsou pro vytvořenou polymorfní aplikaci dvojího druhu. Prvním jsou tlačítka využívající staženého vzdáleného kódu. Tento druh tlačítka pro svou práci používá kód stažený ze vzdáleného zdroje. Pro jeho spuštění je nutno, aby tento kód byl k dispozici na zařízení před jeho použitím. Vzdálený kód je tedy nutno otestovat na jeho dostupnost a v případě, že je dostupný, tak ho stáhnout na lokální zařízení. Tato varianta je označovaná jako „DEX“.

Obdobou, která je v této aktivitě také dostupná, je spouštění vzdáleného kódu (označovaného jako „RPC“), kde se kontroluje, zda je tento kód dostupný až v momentě jeho vyžádání. K tomu dochází, protože existuje reálná šance, že během doby, kdy je „generovaná aplikace“ spuštěna, může dojít ke změně stavu serveru (například dojde k přerušení jeho činnosti) či stavu připojení k síti. Tedy v případě, že například dojde k ztrátě signálu potřebného k připojení k síti a síťovým službám.

Popis, který byl vytvořen pro popsání vzhledu grafického uživatelského rozhraní „generovaných aplikací“ a jejich chování je vidět v následující kapitole.

7.4 Popis prostředí pomocí XML

V kapitole 3 byly popsány objekty potřebné pro tvorbu aplikací s převážně formulářovým zaměřením. Tyto objekty se popisují v XML souboru, který je jedním z důležitých výstupů této diplomové práce.

XML soubor pro popis generované aplikace je reprezentací jejího grafického uživatelského rozhraní. Obsahuje jeden kořenový element, který v sobě zapouzdřuje všechny objekty generované aplikace. Tento element je označen jako `<application>` pro udání začátku popisu a `</application>` pro konec popisu grafického uživatelského rozhraní generované aplikace. V XML popisu tak lze popsat libovolnou formulářovou aplikaci s neomezeným počtem objektů.

Jednotlivé objekty jsou udány elementem `<object>` (resp. `</object>` pro ukončení popisu objektu).

Popsané objekty jsou v polymorfní aplikaci zobrazovány aktivitou pro generované aplikace. Aktivita pro generované aplikace objekty shlukuje v poli, kde jednotlivé prvky jsou tzv. *LinearLayout*. Úkolem *LinearLayout* je v této aktivitě zobrazovat jednotlivé objekty vedle sebe. Každý *LinearLayout* reprezentuje jednu řádku objektů. Pole obsahující prvky *LinearLayout* je zapouzdřeno do tzv. *ScrollView*. Řešení využívající *ScrollView* řeší problém, který vznikne při zobrazování více řádek, než se vejde na obrazovku zařízení. Poskytuje totiž možnost rolování obrazovky.

Každý z objektů má nějaké vlastnosti. Tyto vlastnosti definují jak má objekt vypadat, tak i to, jak se má chovat.

Návrh popisu objektů vychází z myšlenky použité v [Hul12]. Oproti němu je však v řešení, použitém v této diplomové práci, možno přiřazovat jednotlivým prvkům jejich funkce nezávisle na vytvořené polymorfní aplikaci. Dalším rozdílem je větší schopnost navrženým popisem ovlivňovat jak podobu, tak umístění jednotlivých objektů.

V následujícím textu jsou popsané jednotlivé objekty a jejich vlastnosti.

7.4.1 Objekt pro popisek

Objekt pro zobrazování popisku má být primárně informačním prvkem, který uživatel nemůže přepisovat. Jde pouze o formu informačního textu na obrazovce.

Pro popis XML má tyto povinné elementy:

- `<type>` – Udává, že jde o popisek (nastavením na hodnotu „textview“).
- `<id>` – Jednoznačný identifikátor objektu v celém XML popisujícím uživatelské rozhraní.
- `<nameOfObject>` – Jméno objektu v uživatelském rozhraní.
- `<onRow>` – Na jakém řádku uživatelského rozhraní se má tento objekt objevit (s číslováním od hodnoty 1).

Mezi nepovinné elementy pak patří:

- `<text>` – Jaký text se má v popisku objevit. V případě, že je nevyplněn, nebo se v definičním XML souboru nevyskytuje, je považován za popisek s prázdným textem.

Objekt popisek má možnost přepisovat svůj atribut text v případě, že má zobrazovat nějakou užitečnou dodatečnou informaci (například výsledek operace).

V následující ukázce je vidět, jak lze pomocí XML popsat tento objekt:

```
<object>
  <type>textview</type>
  <id>0</id>
  <nameOfObject>JmenoObjektu0</nameOfObject>
  <text>Popisek</text>
  <onRow>1</onRow>
</object>
```

7.4.2 Objekt pro zadávací pole

Úkolem tohoto objektu je možnost zadávání textu uživatelem. Tento text může být předem omezen na například pouze číslice a podobně.

Pro popis XML má tyto povinné elementy:

- `<type>` – Udává, že jde o zadávací pole (nastavením na hodnotu „edittext“).
- `<id>` – Jednoznačný identifikátor objektu v celém XML popisujícím uživatelské rozhraní.
- `<nameOfObject>` – Jméno objektu v uživatelském rozhraní
- `<onRow>` – Na jakém řádku uživatelského rozhraní se má tento objekt objevit (s číslováním od hodnoty 1).

Mezi nepovinné elementy pak patří:

- `<text>` – Jaký nápovědný text (tzv. „hint“) se má v zadávacím poli objevit. Pokud je nevyplněn, nebo se v definičním XML souboru nevyskytuje, je považován za zadávací pole bez nápovědného textu.
- `<subtype>` – Udává, o jaký typ zadávacího pole se jedná. Těchto typů je několik:

- `decimal` – Formát pro zadávání pouze desetinných čísel.
- `sigdecimal` – Formát pro zadávání pouze desetinných čísel se znaménkem.
- `phone` – Formát pro zadávání pouze telefonních čísel.
- `date` – Formát pro zadávání textu pouze v datumovém formátu.
- `text` – Formát pro zadávání textu (přednastavená hodnota použitá v případě, že nebyl nastaven žádný jiný „podtyp“).

V následující ukázce je vidět, jak lze pomocí XML popsat tento objekt:

```
<object>
  <type>edittext</type>
  <subtype>text</subtype>
  <id>1</id>
  <nameOfObject>JmenoObjektu1</nameOfObject>
  <text>Nazev funkce</text>
  <onRow>2</onRow>
</object>
```

7.4.3 Objekt reprezentující tlačítko

Objekt, který je v XML popsán s funkcí pro tlačítko, má za úkol sesbírat všechny zadané hodnoty z polí pro zadávání (označených jako „edittext“) a ty poslat ke zpracování. Tento objekt je hlavním ovládacím prvkem generované aplikace.

Pro popis XML má tyto povinné elementy:

- `<type>` – Udává, že jde o tlačítko (nastavením na hodnotu „button“).
- `<id>` – Jednoznačný identifikátor objektu v celém XML popisujícím uživatelské rozhraní.
- `<nameOfObject>` – Jméno objektu v uživatelském rozhraní
- `<onRow>` – Na jakém řádku uživatelského rozhraní se má tento objekt objevit (s číslováním od hodnoty 1).

- `<typeOfFunction>` – Typ funkce, která se má použít. Jsou dva možné typy: `dex` nebo `rpc`.
- `<functionAddress>` – Adresa, vzdáleného kódu.
- `<nameOfFunction>` – Název funkce, která se má vykonat. Zadání je nutno provést ve formát `JmenoTridy.Funkce`.
- `<idOfResult>` – Id objektu, kam se mají uložit výsledky provedené akce. Pokud nemá být výsledek uložen do žádného objektu, obsahuje hodnotu `-1`.

Mezi nepovinné elementy pak patří:

- `<text>` – Text popisu tlačítka. Tento parametr je sice nepovinný avšak z důvodu přehlednosti programu vysoce doporučený.

V následující ukázce je vidět, jak lze pomocí XML popsat tento objekt:

```
<object>
  <type>button</type>
  <id>2</id>
  <nameOfObject>JmenoObjektu2</nameOfObject>
  <text>Popisek</text>
  <onRow>3</onRow>
  <functionAddress>
    http://192.168.10.103:8080/xml-rpc-server/xmlrpc
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
  <nameOfFunction>Trida.Procedura</nameOfFunction>
  <idOfResult>1</idOfResult>
</object>
```

7.4.4 Objekt reprezentující graf

Objekt, který je v XML popsán pro graf (označený jako „graph“) je oproti ostatním objektům poměrně specifický. Tento objekt se v normálním vývojovém prostředí Androidu nevyskytuje. Jde o objekt speciální knihovny `Androidplot` (viz [Apl12]).

Úkolem objektu je vykreslovat grafy dle získaných dat.

Pro popis XML má tyto povinné elementy:

- `<type>` – Udává, že jde o tlačítko (nastavením na hodnotu „graph“).
- `<id>` – Jednoznačný identifikátor objektu v celém XML popisujícím uživatelské rozhraní.
- `<nameOfObject>` – Jméno objektu v uživatelském rozhraní
- `<onRow>` – Na jakém řádku uživatelského rozhraní se má tento objekt objevit (s číslováním od hodnoty 1).

Mezi nepovinné elementy pak patří:

- `<title>` – Hlavní název grafu (neboli text zobrazený v hlavičce grafu).
- `<axisX>` – Popisek osy X v grafu.
- `<axisY>` – Popisek osy Y v grafu.

V následující ukázce je vidět, jak lze pomocí XML popsat tento objekt:

```
<object>
  <type>graph</type>
  <id>3</id>
  <nameOfObject>JmenoObjektu3</nameOfObject>
  <title>Popisek grafu</title>
  <onRow>4</onRow>
  <axisX>Osa X</axisX>
  <axisY>Osa Y</axisY>
</object>
```

Tento objekt teoreticky pokrývá velkou množinu případů, kdy je potřeba uživateli dát nějaký grafický výstup (například u grafické kalkulačky to může být průběh funkce atd.).

7.5 Vzdálený kód

Jak již bylo řečeno v 5. kapitole, existují různé možnosti, jak využít vzdálený kód. Pro tuto práci byly vybrány nejvhodnější varianty těchto prostředků RPC a DEX. Obě tyto varianty byly popsány v kapitole 5. Jejich použití ve vytvořené polymorfní aplikaci je ukázáno v následujících podkapitolách.

7.5.1 RPC - vdálené procedury

Prvním typem je RPC, tedy prostředek pro vzdálené volání procedur. K tomuto prostředku existují dvě základní varianty popsané v kapitole 5.

Pro vytvořenou polymorfní aplikaci byla použita varianta *XML-RPC* s využitím knihovny *Apache XML-RPC* (viz [Apa10]). Ta byla použita z důvodu její jednoduché použitelnosti při další tvorbě generovaných aplikací pro vytvořenou polymorfní aplikaci.

Princip této varianty je takový, že některý z ovládacích prvků (tlačítko) má přiřazené volání vzdálené procedury „RPC“ jako způsob reakce na událost. Tou je stisknutí tlačítka, které má nastaveno ve svém XML popisu element následujícím způsobem: `<typeOfFunction>rpc</typeOfFunction>`.

Pokud nastane tento případ, tak se vezme adresa serveru, která je přidružena tomuto ovládacímu prvku (tlačítku) a název vzdálené procedury, která má být na serveru vykonána ke zpracování dat. Ukázka parametrů specifikovaných v XML pro daný ovládací prvek je vidět dále:

```
<functionAddress>
  http://192.168.10.103:8080/xml-rpc-server/xmlrpc
</functionAddress>
<typeOfFunction>
  rpc
</typeOfFunction>
<nameOfFunction>
  NazevTridy.JmenoProcedury
</nameOfFunction>
```

Volání procedury na serveru pak probíhá dle zdrojového kódu, který odpovídá následující ukázce:

```
XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
config.setServerURL(new URL("http://192.168.10.103:8080/
    xml-rpc-server/xmlrpc"));
XmlRpcClient client = new XmlRpcClient();
client.setConfig(config);
Object[] params = new Object[]{new Integer(33), new Integer(9)};
Integer result = (Integer)client.execute("Calculator.add",params);
```

V této ukázce je vidět volání serveru (respektive *XML-RPC serveru*) provozovaného na síťové adrese udané elementy `<functionAddress>` v XML popisu (viz příklad výše). Na serveru je požadováno volání procedury `add` umístěné ve třídě `Calculator`.

Pro předání parametrů, potřebných pro správný běh vzdálené procedury, je použito pole objektů (`Object[]`) nazvané `params`. V ukázce jsou přenášeny dvě hodnoty čísla typu celé číslo (neboli `integer` hodnot `33` a `9`). Pro správnou funkci však musí strana serveru přijímat stejný počet a typy proměnných, jinak dochází k chybě a tím pádem neprovedení požadované procedury serveru. To platí i opačně, tedy při předávání výsledků ze serveru ke klientovi, kdy je nutné přijímat stejný typ a počet dat poslaných od serveru klientské aplikaci.

Výsledek operace provedené na vzdálené proceduře serveru je po jejím provedení uložen do hodnoty `result` typu `integer`. Obecně však lze použít jakýkoli typ objektu či struktury složené ze základních datových typů programovacího jazyka.

K tomu, aby server mohl poskytnout odpověď, je potřeba na serveru mít v danou chvíli spuštěnou odpovídající službu. Tato služba bude pomocí XML-RPC knihovny získávat dotazy od klientů, zpracovávat je a posílat zpět požadované výsledky provedené operace. Server volanou proceduru zapouzdřuje v třídě (viz příklad výše - `Calculator`).

Ukázka funkce volané na straně serveru vypadá následovně:

```
package org.apache.xmlrpc.demo;
public class Calculator {
  public int add(int i1, int i2) {
    return i1 + i2;
  }
  public int subtract(int i1, int i2) {
    return i1 - i2;
  }
}
```

V ukázce jsou vidět dvě metody, které je možné na serveru ze třídy `Calculator` volat. První sčítá dva vstupní parametry a druhá provádí odečítání druhého parametru od prvního.

K tomu aby takovéto volání mohlo fungovat, je potřeba na serveru vytvořit ještě soubor `XmlRpcServlet.properties`, který specifikuje třídy dostupné na serveru. Tento soubor je třeba vytvořit ve specifické složce serveru (pro ukázkový příklad by šlo o `org/apache/xmlrpc/webserver`) a do něj uložit následující příkaz:

```
Calculator=org.apache.xmlrpc.demo.Calculator
```

V případě, že je potřeba přidat více tříd na daný server, tak se do tohoto souboru přidají pouze další řádky dle uvedeného vzoru.

Následně je pak potřeba upravit soubor `web.xml` například dle následující ukázky:

```
<servlet>
  <servlet-name>XmlRpcServlet</servlet-name>
  <servlet-class>
    org.apache.xmlrpc.webserver.XmlRpcServlet
  </servlet-class>
  <init-param>
    <param-name>enabledForExtensions</param-name>
    <param-value>true</param-value>
    <description>
```

```
      Sets, whether the servlet supports
      vendor extensions for XML-RPC.
    </description>
  </init-param>
</servlet>
<servlet-mapping>
<servlet-name>XmlRpcServlet</servlet-name>
  <url-pattern>/xmlrpc</url-pattern>
</servlet-mapping>
```

Ukázky klientské i serverové části jsou převzaty z [Apc10, Aps10] a v modifikované podobě použity ve vytvořené polymorfní aplikaci.

Pro vytvořenou polymorfní aplikaci bylo v základní podobě použito postupu, že před odesláním parametrů na server jsou sesbírána všechna data určená pro odeslání. Jde o procházení všemi editovatelnými textovými poli (objekty typu `edit-text` a jejich obdob) a uložení jejich obsahu do dynamického pole - kolekce `Array-List<String>`.

Následně je ověřena dostupnost serveru (pomocí chybové odpovědi s číslem `405`, kterou v případě volání HTTP požadavku vrací server pokud na něm běží *XML-RPC server*). Pokud server odpoví na tento požadavek správně, jsou mu pomocí pole objektů (`Object[] params`) předány vstupní parametry. Vstupní parametry jsou získány z kolekce obsahů editovatelných polí. Pole objektů s parametry je pak odesláno serveru zavoláním funkce `execute()` nad objektem typu `XmlRpcClient` (v obrázku 7.1 je odeslání požadavku na server označeno jako `B1`).

```
Object[] outObjectArray
   = (Object[]) client.execute(nameOfFunc, params);
```

Výsledek operace je pak navrácen do pole objektů `outObjectArray` (na obrázku 7.1 je získání výsledku operace označeno jako `B2`). V tomto poli se na první pozici nachází kódové číslo stavu operace. V případě, že během operace na straně serveru nemohla být data zpracována (a to buď špatným formátem, pořadím nebo obsahem vstupních dat), je navrácena záporná hodnota na pozici prvního parametru. Pokud k tomu dojde, je druhým parametrem text chybového hlášení, které se má uživateli zobrazit na obrazovce.

Jak již bylo zmíněno, pokud je operace na serveru úspěšně provedena, tak je vráceno kladné číslo v prvním parametru. Druhým parametrem je hlášení, které se má uživateli zobrazit na obrazovce zařízení. Další parametry jsou pak rozlišeny kódovým číslem stavu operace. V případě, že hodnota byla různá od hodnoty čísla `2`, tak je podle tlačítka, které operaci volalo, nalezen objekt, kterému se má jako „popisek“ nastavit poslední (třetí) parametr. Tento objekt je určen hodnotou udanou u objektu (respektive tlačítka), které danou operaci volalo elementem `<idOfResult>` udaným v XML souboru popisujícím podobu prostředí polymorfní aplikace.

Pokud však je hodnota kódového stavu operace nastavena na hodnotu čísla `2`, jde o objekt vykreslení do objektu typu `graph` (neboli grafu). Pro graf je možno, aby za prvními dvěma parametry (popsanými výše) bylo větší množství parametrů typu `String` (neboli textový řetězec). Každý z těchto řetězců má pak tvar:

```
NázevGrafu@x1;y1;x2;y2;x3;y3; ...
```

Symbolické proměnné (`x1,y1` atd.) jsou ve skutečném řetězci oproti této ukázce nahrazeny hodnotami udávajícími souřadnice bodů v osách x a y křivky, která se má v grafu vykreslit. Znakový symbol „@“ slouží v řetězci k oddělení názvu grafu a souřadnic křivky grafu. Pokud se oddělující symbol v řetězci nenachází, je pro textový popisek grafu použito přednastavené pojmenování „Graph“.

Požadovaný graf se určuje stejným způsobem jako se v předchozím popisu udával objekt, do kterého se měl vypsat popisek z návratového řetězce (tedy přes element `idOfResult`, který je zadaný u objektu typu tlačítko v XML popisujícím podobu polymorfní aplikace).

7.5.2 DEX - kód ze vzdáleného úložiště

Druhým typem prostředku pro zpracování operací je tzv. „DEX“. Obecný princip činnosti této varianty byl popsán v kapitole 5.3.

Varianta je založena na stažení vzdáleného kódu ve formátu `dex` na lokální zařízení (na obrázku 7.1 je kontrola existence DEX souboru na úložišti označena jako `C1` a jeho stažení samotné do zařízení jako `C2`). Tato varianta používá pro XML popis daného ovládacího prvku sloužícího k operacím s DEX podobný syntaktický zápis, jako tomu bylo ve variantě RPC popsané v předchozí kapitole 7.5.1.

Pro ukázku lze uvést následující příklad:

```
<functionAddress>
  http://home.zcu.cz/~staffa/DexLibrary.jar
</functionAddress>
<typeOfFunction>
  dex
</typeOfFunction>
<nameOfFunction>
  Calculator.add
</nameOfFunction>
```

V rámci těchto elementů uvedených u ovládacího prvku (respektive tlačítka) je možno vidět, jaké DEX se má použít pro vykonání požadované operace. Element `<functionAddress>` udává adresu, kde se nachází daný DEX v URL formátu. Ten, jak je vidět z této ukázky, však není v očekávaném formátu `.dex`, ale je zapouzdřen ve formátu souboru `.jar`, zde bude však označován jako DEX.

V programovacím jazyku Java existuje komponenta (tzv. `ClassLoader`), která se stará o nahrávání tříd do aplikace. V Javě i Androidu však existuje možnost, jak za běhu nahrát do aplikace „externí“ třídu. Tyto třídy pak mohou být umístěny na vzdáleném úložišti či na lokálním souborovém systému. Pro samotné načítání tříd se využívá tzv. `DexClassLoader`, který umožňuje načítat DEX soubory za běhu aplikace.

Polymorfní aplikace je vytvořena tak, že v případě nedostupnosti síťového připojení lze využívat i DEX soubory, které byly použity jako poslední. K tomu je však nutné, aby při úplně prvním spuštění generované aplikace, která bude využívat daný DEX, byl tento DEX stažen. V případě dostupnosti připojení je při spuštění generované aplikace požadovaný DEX vždy stažen. Důvodem tohoto řešení je, aby v případě aktualizace DEX souborů poskytnutých vývojářem/distributorem na úložišti byly vždy použity aktuální verze (respektive byly vždy znovu stáhnuty za předpokladu síťového připojení).

V polymorfní aplikaci byl pro stažení DEX knihoven zvolen následující způsob.

Po přečtení a parsování XML souboru, který popisuje danou generovanou aplikaci, dojde k procházení všech ovládacích prvků (tlačítek) a zjištění adresy DEX v XML souboru, pokud mají nastaven DEX jako obslužný typ operace. Adresy se uloží do `ArrayList<String> arrayOfAddresses`. Potom je `ArrayList` procházen po jednotlivých uložených záznamech a jsou stahovány potřebné DEX soubory.

V tomto kroku existují dvě varianty. První je ta, že pokud je k dispozici připojená a pro čtení/zápis dostupná externí paměť (například tzv. SD karta), tak je DEX uložen na ni. To je z důvodu úspory interní paměti na mobilních zařízeních. V případě, že externí úložiště k dispozici není, je použita interní paměťová oblast určená pro data aplikací. Ta je pro vytvořenou polymorfní aplikaci udána cestou `data/data/dpapp1`.

Samotné stahování souborů DEX pak vypadá přibližně dle následující ukázky:

```
InputStream input
  = new BufferedInputStream(url.openStream());
File myDir;

if (isSDCardPresent) {
  String root
    = Environment.getExternalStorageDirectory().toString();
  myDir
    = new File(root + "/Android/data/" + PACKAGE_NAME + "/");
      } else {
  myDir
    = new File("/data/data/" + PACKAGE_NAME + "/dex/");
}

OutputStream output
  = new FileOutputStream(myDir + "/" + nameOfFile);
byte data[] = new byte[1024];
long total = 0;
int count;

//read data
while ((count = input.read(data)) != -1) {
  total += count;
  // publishing the progress....
  publishProgress((int) (total * 100 / fileLength));
  output.write(data, 0, count);
}

output.flush();
output.close();
input.close();
```

Následně je pak nutné při volání akcí nad těmito DEX soubory dané soubory zpracovat. K tomu je potřeba znát rozhraní (respektive interface) pro přístup k procedurám, které jsou poskytovány tímto DEX.

Aby bylo vytváření a používání DEX souborů co nejpřehlednější, byla zvoleno následující rozhraní:

```
public interface LibraryInterface {
        public ArrayList<String>
    doLibraryFunction(String nameOfFunction,
      ArrayList<String> params);
}
```

Z ukázkového příkladu je vidět, že způsob volání procedury je hodně podobný volání procedury u RPC uvedeného v příkladu v předchozí kapitole 7.5.1.

K samotnému načtení třídy s požadovanou procedurou potřebujeme již zmíněný `DexClassLoader`. V příloze B je vidět jeho použití.

Jak je z tohoto ukázkového zdrojového kódu vidět, podobně jako v kapitole 7.5.1 o RPC je navrácen kód stavu operace. Dalšími parametrem je stejně jako u zmíněného RPC text, který má být vypsán ve formě informačního textového pole na obrazovku. Posledními parametry jsou zpětná data pro vykreslení grafu či vypsání textu do jednoho z textových polí/objektů generované aplikace.

Vytvořený DEX soubor je pak založen například na následující ukázce kódu:

```
public class DexLibrary implements LibraryInterface{
  public ArrayList<String> doLibraryFunction
    (String nameOfFunction, ArrayList<String> params) {

    int first = 0;
    int second = 0;
    int result = 0;
    ArrayList<String> outputArray = new ArrayList<String>();

    if (nameOfFunction.equals("add"))
    {
      try
```

```
      {
        first = Double.parseDouble(inputArray[1]);
        second = Double.parseDouble(inputArray[2]);
      } catch (Exception e) {
        System.err.println(e);
        outputArray.add("-1");
        outputArray.add("Spatne zadana cisla!");
        return outputArray;
      }
    }
    else { ... }

    outputArray.add("0");
    outputArray.add("Vysledek je souctu je: " + result);
    outputArray.add(result.toString());
    return outputArray;
  }
}
```

Podmínka této varianty spočívá v dodržení vstupního a výstupního formátu dat. Další důležitou podmínkou je to, že daná DEX knihovna (v tomto případě s názvem `DexLibary`) musí mít stejné jméno jako `.jar` soubor, ve kterém je distribuována. V opačném případě operace provedená nad touto knihovnou končí chybou.

Vývojáři pro navrhování vlastních aplikací následně stačí pouze vytvořit popisové XML (pro popis chování a uživatelského rozhraní generované aplikace) a dle toho, zda chce využívat RPC či DEX, vytvořit danou knihovnu či server poskytující požadované procedury.

Navržené řešení bylo realizováno na základě výzkumu z [Chu11, Hul12].

8 Bezpečnost polymorfní aplikace

V kapitole 6.3 byla z obecného hlediska popsána bezpečnost aplikací v mobilních zařízeních se systémem Android. Následující kapitola se zabývá popisem problémů s bezpečností vzhledem k vytvořené polymorfní aplikaci a jejich řešením.

8.1 Nově vzniklé problémy bezpečnosti

Návrhem polymorfní aplikace pro tuto práci vyvstaly nové problémy s bezpečností. Za prvé jde o již zmíněné velké množství potřebných práv nutných při použití DEX varianty. Důvodem tohoto požadavku je, aby polymorfní aplikace mohla být co nejvíce všestranná, k čemuž potřebuje velký rozsah práv. Řešením je, že vytvořená polymorfní aplikace bude distribuována v různým verzích rozlišených poskytnutými právy.

Nebezpečím je i možnost podvrhnutí (respektive zaměnění) DEX souboru. K tomu by mohlo dojít v momentě stahování DEX souboru ze serveru. Soubor by následně mohl zneužít práv poskytnutých aplikaci a tak způsobit škody. Škody by mohly být způsobeny tím, že v této knihovně může být umístěn škodlivý či jinak nebezpečný kód. Tento kód by mohl například poškodit zařízení či data na něm uložená. V jiném případě by mohl data ze zařízení poskytovat třetím osobám.

Ideálním řešením problematiky bezpečnosti v polymorfní aplikaci zmíněného problému by bylo zavedení tzv. elektronického podpisu souborů (někdy též označovaného jako „digitální podpis“). Ten by zaručoval jednoznačnost a pravost souborů získávaných z „vnějších zdrojů“. Ze stejného důvodu by bylo dobré zabezpečit XML soubor sloužící pro popis generované aplikace, který by byl uložen na serveru (respektive jeho úložišti). Mohlo by totiž dojít k tomu, že uživatel by požadoval určité XML popisující generovanou aplikaci, avšak útočník by mohl namísto odpovědi s původním originálním XML souborem poslat zpět do polymorfní aplikace podvrhnutý soubor. Ten by se mohl z pohledu uživatele tvářit jako původní aplikace, avšak mohl by v popisech ovládacích prvků odkazovat na nebezpečné zdroje RPC i DEX.

Více o digitálních podpisech souborů viz [Ber03].

Další možností zabezpečení je použití šifrování. To by zabezpečilo, že XML a DEX soubory nebude možno zaměnit za jejich obdoby obsahující škodlivý kód nebo jinou formou nebezpečný obsah. Druhou kladnou vlastností této varianty je

to, že zašifrovaný soubor není možno jednoduchým způsobem bez znalosti dešifrovacího algoritmu a klíče pro dešifrování a šifrování zneužít či přečíst. Ochrana proti přečtení tohoto souboru by byla vhodná pro využití polymorfní aplikace například pro bankovnictví a další podobné účely. U těchto aplikací totiž může být nutné distribuovat kód v šifrované podobě, aby byl ochráněn jeho obsah, ve kterém se mohou nacházet důvěrné informace, či zdrojový kód, který je nutno nějakým způsobem chránit.

Pro realizaci zabezpečení polymorfní aplikace byla zvolena varianta s použitím šifrování.

8.2 Možnosti šifrování

Šifrování, které lze použít pro zabezpečení dat se dá obecně rozdělit na dva základní druhy:

- *Symetrické*
- *Asymetrické*

8.2.1 Symetrické šifrování

Symetrické šifrování je založeno na principu jednoho stejného klíče pro šifrování a dešifrování. Slabé místo symetrického šifrování spočívá v tom, že je třeba pro dekódování získat klíč, kterým byly data kódovány. K tomu je třeba použít nějaký bezpečný způsob přenosu kódovacího/dekódovacího klíče. Pokud však existuje, je často jednodušší poslat data přímo za použití tohoto přenosu.

Symetrické šifrování je mnohem jednodušší než asymetrické, pro jeho zpracování obecně stačí menší výkon než na asymetrické šifrování.

Symetrické šifrování se dá dobře použít v případě, že odesílatel a příjemce dat si předem dohodli, jaký klíč pro šifrování použijí.

Představiteli symetrického šifrování jsou *DES* („Data Encryption Standard“), *Triple DES* a *AES* („Advanced Encryption Standard“ neboli také *Rijndael*). Šifrování *AES* je bezpečnější než *DES* a *Triple DES*. Více o historii, vlastnostech a principu *AES* v [Dol01].

Symetrické šifrování lze dále dělit na tyto typy:

- *Blokové šifry* - Blokové šifry pracují s pevným počtem bitů. Tyto bloky jsou pomocí klíče zašifrovány a výstupem je příslušná část šifrovaného textu. Některé algoritmy provádějí několikanásobné šifrování bloku dokola, aby se zvýšila bezpečnost. Blokové šifry jsou na šifrování dat v praxi využívány častěji než proudové (ty budou popsány dále). Mezi blokové šifry patří například algoritmy *DES*, *AES*, *IDEA*, *Blowfish* a jiné.
- *Proudové šifry* - Proudové šifry jsou rychlejší než blokové. Vhodnější jsou tam, kde není možné využívat buffer. Typickým příkladem je telekomunikace a jiné real-time spojení, kde není vhodné čekat na naplnění bufferu, ale je třeba data šifrovat ihned. Tyto šifry jsou založeny na šifrování každého znaku otevřeného textu zvlášť. Mezi zástupce proudových šifer patří algoritmy *RC4*, *FISH*, *Helix*, *SEAL*, *WAKE* a další.

8.2.2 Asymetrické šifrování

Asymetrické šifrování využívá toho, že klíč použitý pro šifrování dat je jiný než klíč pro dešifrování.

Prvním typem klíče je tzv. „veřejný“ klíč. Tento klíč dokáže dešifrovat zprávy zašifrované druhým typem klíče (tzv. „soukromým“, „privátním“ nebo-li také „tajným“ klíčem).

Odesílatel zprávy provede zašifrování zprávy veřejným klíčem adresáta. Ten po přijetí této zašifrované zprávy provede dešifrování svým privátním klíčem.

Nejrozšířenějším typem tohoto šifrování je RSA. Tento typ používá modulární aritmetiku (neboli matematikou zabývající se zbytky po dělení celých čísel).

Citace pro tuto část textu byly použity z [Mun07], kde se lze dozvědět více o principech a vlastnostech šifrování.

8.2.3 AES pro šifrování v rámci polymorfní aplikace

Tento šifrovací algoritmus byl pro polymorfní aplikaci navrhnut a použit z důvodu své poměrně velké bezpečnosti a rychlosti. Hlavním cílem této šifry bylo řešit nedostatky `DES` šifrování. Oproti `Triple DES` a `DES` je rychlejší a bezpečnější. AES je symetrickou blokovou šifrou, která využívá délku klíče 128, 192 nebo 256 bitů.

O vzniku a použití algoritmu AES se lze dozvědět v [Dol01].

Pro polymorfní aplikaci je šifrování AES vhodné na zabezpečení souborů získávaných aplikací ze zdrojů, které nemusí být bezpečné. Tím je myšleno, že přenos dat z nich a na ně je možno narušit (záměnou či poškozením přenášených dat). U šifrování AES použitého v polymorfní aplikaci byla experimentováním nalezena výhoda. Ta spočívá v tom, že pokud při přenášení zašifrovaných dat dojde ke změně byť jediného bytu, tak je daný soubor nedešifrovatelný. K tomuto problému by mohlo dojít, pokud by „útočník“ chtěl změnit obsah dat. Aby bylo možné efektivně prolomit toto zabezpečení, musel by útočník znát heslo, kterým byla data šifrována, a použitý algoritmus šifrování.

Využití šifrování AES ve vytvořené polymorfní aplikaci je takové, že vývojář má možnost zašifrovat pomocí AES XML soubor pro popis generované aplikace. Stejným způsobem je možno šifrovat i DEX knihovny použité pro polymorfní aplikaci. K provedení této akce stačí zadat v hlavní aktivitě přes tlačítko *Menu* položku *Nastavení hesla pro šifrování* a zde nastavit heslo pro šifrování. Pro zvýšení bezpečnosti je nutné, aby toto heslo mělo 16 znaků (jinak bude použito přednastavené heslo „*fakultafavzcuplz*“). Následně pak zadat adresu souboru, který chce vývojář zašifrovat do textového pole, a přes *Menu* zvolit položku *Vytvořit šifrovaný soubor*.

Provedením této akce se daný soubor přečte ze zadané adresy a zašifruje do souboru ve vnitřní paměti. V případě přítomnosti SD karty je zašifrovaný soubor přednostně uložen na ni. Odtud si ho vývojář může stáhnout a uložit na požadovaný server, odkud ho budou moci využívat i další zařízení prostřednictvím polymorfní aplikace pro svou činnost. V této části bylo implementováno zjednodušení v podobě možnosti nechat si zašifrovaný soubor zaslat na email (v případě, že je na daném zařízení odpovídající aplikace pro odesílání elektronické pošty).

Pro šifrování souborů je potřeba nejdříve tento soubor z dané adresy získat. To je v polymorfní aplikaci řešeno následujícím způsobem:

```
DefaultHttpClient hc
  = new DefaultHttpClient();
ResponseHandler <String> res
  = new BasicResponseHandler();
HttpPost postMethod
  = new HttpPost(sUrl[0].toString());
String response
  = hc.execute(postMethod,res);
String pass
  = readString(MainActivity.this, "password",
    "fakultafavzcuplz");
byte[] crypted
  = encrypt(response, pass);
```

Metoda `readString` v této ukázce reprezentuje zjištění hesla, uloženého v tzv. „sdílených preferencích“. Po zjištění tohoto hesla dojde k zašifrování textu souboru (v ukázce získán do proměnné `response`). Výsledek je uložen jako `byte[]` s názvem `encrypted`.

Ukázka kódu použitého pro šifrování textu je vidět v následujícím kódu:

```
private static byte[] encrypt(String text, String pass)
  throws Exception {
  byte[] keyStart = pass.getBytes();
  KeyGenerator kgen = KeyGenerator.getInstance("AES");
  SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
  sr.setSeed(keyStart);
  kgen.init(128, sr); // 192 and 256 bits may not be available
  SecretKey skey = kgen.generateKey();
  byte[] key = skey.getEncoded();
  SecretKeySpec skeySpec = new SecretKeySpec(key, "AES");
  Cipher cipher = Cipher.getInstance("AES");
  cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
  byte[] encrypted = cipher.doFinal(text.getBytes());
  return encrypted;
}
```

Dešifrování je prováděno obdobně jako šifrování. Vstupem je pole `byte[]`, obsahující data šifrovaného souboru. Výsledkem je řetězec (`decrypted`) obsahující původní text. Dešifrování je vidět v následující ukázce zdrojového kódu polymorfní aplikace:

```
private static byte[] decrypt(byte[] encrypted, String pass)
  throws Exception {
  byte[] keyStart = pass.getBytes("UTF-8");
  KeyGenerator kgen = KeyGenerator.getInstance("AES");
  SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
  sr.setSeed(keyStart);
  kgen.init(128, sr); // 192 and 256 bits may not be available
  SecretKey skey = kgen.generateKey();
  byte[] key = skey.getEncoded();
  SecretKeySpec skeySpec = new SecretKeySpec(key, "AES");
  Cipher cipher = Cipher.getInstance("AES");
  cipher.init(Cipher.DECRYPT_MODE, skeySpec);
  byte[] decrypted = cipher.doFinal(encrypted);
  return decrypted;
}
```

Tento způsob lze použít pouze v případě, že šifrované soubory, které byly zašifrovány pod systémem Android, budou dešifrovány znovu na tomto systému.

V případě dešifrování dat na jiném systému než byla data šifrována, je možnost, že vznikne problém s tím, že data nepůjdou správně dešifrovat. K zmíněnému problému může dojít z důvodu špatného nastavení šifrovacího schématu či z důvodu generování jiného dešifrovacího klíče (to může být způsobeno použitím jiné platformy).

K tomu by mohlo dojít, pokud by DEX knihovna zařizovala komunikaci se serverem, který běží například na systému Windows.

V popsaném případě lze použít přímo nastavení 16 bytů (128 bitů) pro nastavení klíče. To by vypadalo například následovně:

```
private static byte[]
  encrypt(String text, String pass)
  throws Exception {

  byte[] keyBytes
    = new byte[] {
    0x00, 0x01, 0x02, 0x03, 0x04,
    0x05, 0x06, 0x07, 0x08, 0x09,
    0x0a, 0x0b, 0x0c, 0x0d, 0x0e,
    0x0f};

  SecretKeySpec skeySpec
    = new SecretKeySpec(keyBytes, "AES");
  Cipher cipher = Cipher.getInstance("AES");
  cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
  byte[] encrypted = cipher.doFinal(text.getBytes());
  return encrypted;
}
```

Obdobně by se tak dešifrovala data i na straně serveru.

Soubory zašifrované pomocí zmíněného kódu dostávají novou příponu končící písmenem „c“. XML soubor, který bude takto zašifrován, dostane místo své původní přípony novou příponu „.xmlc“. To samé platí i pro zašifrování DEX souboru reprezentovaného jako „.jar“, který po zašifrování dostane příponu „.jarc“.

Polymorfní aplikace pak vždy, když při své práci narazí na typ souboru „.xmlc“ nebo „.jarc“, využije šifrování s heslem nastaveným v již zmíněné položce v *Menu*. Bez správně nastaveného hesla nebude možno tyto soubory dešifrovat.

Pro případ, že by útočník měl přístup k paměti daného zařízení, jsou soubory typu „.jarc“ v paměti uchovávány v šifrované podobě a v nešifrované podobě se využívají pouze pro interní činnost, jinak se v nešifrované podobě do paměti neukládají. Tím je útočníkovi zabráněno podvržení daného souboru v případě jeho zaměnění v paměti za jiný (respektive nebezpečný) soubor.

Pro šifrované komunikace se serverem lze využít vlastní DEX knihovny, ve které bude šifrování vytvořeno vývojářem. Zde však při vytváření polymorfní aplikace

nastává problém s tím, že DEX knihovna sama o sobě nemá možnost komunikovat s uživatelským rozhraním a využívat jeho možností. Jedinou možností, kterou má DEX pro komunikaci s uživatelským rozhraním k dispozici, je přes návratové hodnoty procedur (dle návrhu aplikace). Tím se sice zabezpečí úpravy uživatelského rozhraní, které by mohl útočník provést, avšak problém spočívá v síťové práci uvnitř samotné DEX knihovny.

Podle pravidel vývoje aplikací pro Android je nutné vytvářet pro síťovou komunikaci nové vlákno a nebo využívat prostředku pro asynchronní činnosti (tzv. `AsyncTask`). Toto řešení je nutné z důvodu, aby hlavní vlákno, které by mělo provádět síťové operace, nebylo těmito operacemi zdržováno na úkor plynulosti běhu aplikace (respektive hlavní vlákno nesmí obsahovat síťové operace). Pokud je však v DEX knihovně vytvořena procedura, která má komunikovat síťovými prostředky (tedy komunikační část je umístěna v `AsyncTask`), tak operace DEX samotného skončí dříve, než stačí `AsyncTask` operace zajišťující šifrovanou komunikaci předat návratovou hodnotu. Jinak řečeno takováto komunikace se pak tváří pro klienta jako provedená, avšak výsledky komunikace se uživatel již nedozví.

K tomu byla v aktivitě pro generované aplikace zavedena metoda označená jako `makeText(String text)`, která zařizuje vypisování informačních textů na obrazovku (v rámci hlavního vlákna).

9 Vytvořené ukázkové „generované aplikace“

Pro ověření funkcionality polymorfní aplikace byly vytvořeny různé ukázkové generované aplikace. Tyto generované aplikace mají za úkol potvrdit správnost myšlenky polymorfní aplikace a ukázat cestu, kterou by mohlo být pokračováno v rozvoji realizované myšlenky této polymorfní apliace.

Pro spuštění požadnované generované aplikace prostřednictvím polymorfní aplikace je potřeba následující:

- Polymorfní aplikace musí být nasazena na zařízení se systémem Android (viz příloha C).
- XML popis generované aplikace musí být dán k dispozici serverem (resp. tento soubor musí být síťově dostupný).
- V případě, že generovaná aplikace využívá RPC, tak požadovaný RPC server musí být spuštěný a síťově ze zařízení dostupný (viz příloha D).
- V případě, že generovaná aplikace využívá DEX, tak všechny požadované knihovny musí být síťově ze zařízení dostupné.

V dalších podkapitolách jsou popsány jednotlivě všechny vytvořené generované aplikace.

9.1 Grafická kalkulačka

9.1.1 Základní myšlenka

Aplikace byla navržena na základě myšlenky vytvoření generované aplikace (pro polymorfní aplikaci), která bude umět vykreslovat dle uživatelem zadaných parametrů grafové průběhy.

9.1.2 Realizace

Aplikace se skládá ze zadávací a zobrazovací části. V zadávací části může uživatel nastavit *Typ funkce*, kterou chce vykreslit a k ní počátek a konec vykreslovaného intervalu. Pro ukázkovou generovanou aplikaci „Grafická kalkulačka“ byly jako vzorové implementovány funkce *sinus* a *cosinus*.

Generovaná aplikace „Grafická kalkulačka“ byla vytvořena ve dvou variantách.

První varianta byla vytvořena za použití RPC. Využívá volání vzdálených procedur ve formě, ve které byla tato varianta popisována v předchozích kapitolách.

Samotné zpracování kódu na serveru je vidět v následující ukázce.

Na serveru je připravena metoda, která zabezpečuje vytvoření dat grafu:

```
public Object[] grahpsSinCos (Object[] input) {
  inputArray = new String[input.length];
  outputArray = new String[3];
  String outputText = "";
  double granularity = 0.1;
  double zacatekIntervalu = 0;
  double konecIntervalu = 0;
```

Po inicializaci při spuštění metody je třeba extrahovat vstupní parametry, které byly této metodě předány při jejím volání:

```
  for (int i = 0; i < inputArray.length; i++)
  {
    inputArray[i] = (String) input[i];
  }

  try {
    System.out.println(inputArray[1]);
    System.out.println(inputArray[2]);
    inputArray[1].replace(',', '.');
    inputArray[2].replace(',', '.');
    zacatekIntervalu = Double.parseDouble(inputArray[1]);
    konecIntervalu = Double.parseDouble(inputArray[2]);
```

```
} catch (Exception e) {
  System.err.println(e);
  outputArray = new String[2];
  outputArray[0] = "-1";
  outputArray[1] = "Spatne zadana cisla intervalu!";
  return outputArray;
}
```

Dalším krokem je samotný výpočet dat pro graf. V tomto případě jde o vytváření přímek po tak jemném intervalu, že se uživateli bude výsledný průběh jevit jako plynulá sinusoida či kosinusoida.

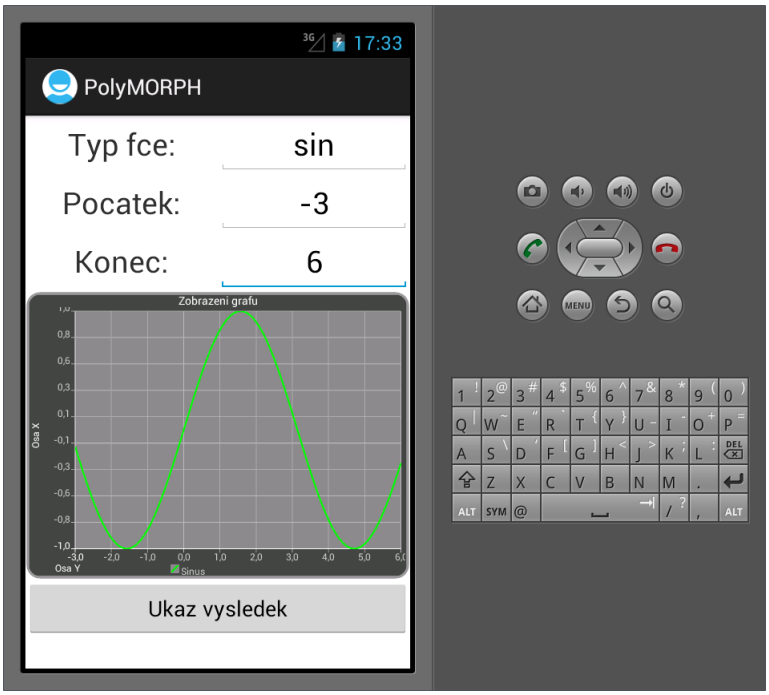
```
double pocet
  = ((konecIntervalu - zacatekIntervalu) / jemnost) + 1;
int pocetZaokr = (int)Math.round(pocet);
outputArray[0] = "0";
outputArray[1] = "Graf vykreslen";
if (inputArray[0].trim().equals("sin")) {
  System.out.println("Zvolen sinus");
  for (int i = 0; i < (pocetZaokr * 2); i += 2) {
    outputText
      += (i / 2) * jemnost
      + zacatekIntervalu + ";";
    outputText
      += Math.sin((i / 2) * jemnost
      + zacatekIntervalu) + ";";
  }
  outputText.substring(0, outputText.length() - 2);
  outputArray[2] = "Sinus@" + outputText;
} else if (inputArray[0].trim().equals("cos")) {
  System.out.println("Zvolen cosinus");
  for (int i = 0; i < (pocetZaokr * 2); i += 2) {
    outputText += (i / 2) * jemnost + ";";
    outputText += Math.cos((i / 2) * jemnost) + ";";
  }
  outputText.substring(0, outputText.length() - 2);
  outputArray[2] = "Cosinus@" + outputText;
} else {
  System.out.println("Nezvoleno nic!");
```

```
      outputArray = new String[2];
      outputArray[0] = "-1";
      outputArray[1] = "Nezvoleno nic!";
    }
    Object[] outputObject = (Object[])outputArray;
    return outputObject;
  }
```

Zdrojový kód je implementací návrhu popsaného v kapitole 7.5.1 (části o popisu grafů polymorfní aplikace).

Obdobným způsobem funguje i varianta pro DEX, která obsahuje v DEX knihovně prakticky totožný zdrojový kód.

Na ukázce je vidět, že pro polymorfní aplikaci lze vytvořit generované aplikace, které zpracovávají například výpočtové úlohy. Graf generované aplikace může být využit při tvorbě aplikací zobrazujících statistické hodnoty či různá měření (například vytížení serveru, aplikace pro zobrazování meteorologických průběhů atd.).



Obrázek 9.1: Ukázka generované aktivity „Grafická kalkulačka“ spuštěné v emulátoru.

Ukázka, jak vypadá generovaná aplikace je vidět na obrázku 9.1.

9.1.3 Varianty generované aplikace

Varianta RPC se dá využít v případech, kde data pro tuto aplikaci nelze získat jinak než ze vzdáleného serveru (či jiného místa v síti). Další možností jsou výpočetně náročné úlohy, které by nebylo možné zpracovávat na mobilním zařízení. Důvodem by mohla být zvýšená spotřeba energie, nízký výpočetní výkon či hardwarové omezení zařízení.

Varianta DEX by se mohla využít v případech, kdy by bylo nutné (např. výpočetní) operace provádět bez dostupnosti síťového připojení. Jedním z dalších faktorů pro použití DEX je také to, že by přenos dat v případě RPC byl časově (či objemově) náročný a nebo v případě, že by data mohla být nějakým způsobem zneužita.

9.2 Dálkový multimediální ovladač

9.2.1 Základní myšlenka

Tato generovaná aplikace vytvořená pro polymorfní aplikaci je založena na myšlence ovládání jiných zařízení z polymorfní aplikace a tím rozšíření jejích možností.

Pro ukázku vzdáleného ovládání byla použita aplikace *VLC media player 2.0.3 „Twoflower“* (více viz [Vlc12]). Zmíněná aplikace slouží jako multimediální audio/video přehrávač.

9.2.2 Realizace

Generovaná aplikace „Dálkový multimediální ovladač“ byla navržena tak, aby její pomocí bylo možno využívat co největší množství funkčností ovládaného programu *VLC media player*.

Z pohledu grafického uživatelského rozhraní se dá aplikace rozdělit na dva bloky.

Prvním blokem jsou standardní ovládací prvky, které lze najít u jakéhokoli ovladače podobných programů (i zařízení). Zde se nacházejí ovládací prvky pro spuštění, pozastavení a úplné zastavení ovládaného přehrávače. Dále lze přepínat skladby uložené v seznamu skladeb (tzv. „playlistu“). Posledním ovládacím prvkem této části je vypnutí přehrávače.

Druhým blokem uživatelského rozhraní je ovládání pomocí konzolových příkazů. Jde tedy o to, že uživatel nemusí mít v uživatelském rozhraní všechny možné ovládací prvky, ale pouze ty, které využívá nejčastěji.

V případě, že by uživatel chtěl při přehrávání nějakého audio/video souboru ztlumit zvuk, tak místo hledání ovládacího prvku stačí do konzolového okna na spodní části uživatelského rozhraní napsat příkaz „mute“ (teda v překladu do českého jazyka „ztlumit“). Po zadání řídícího příkazu dojde k okamžitému odeslání tohoto příkazu na server, kde je dále zpracován.

Pokud na serveru neběží požadovaná obslužná procedura, je o tom uživatel informován pomocí (dočasného) textového pole na obrazovce.

Za předpokladu, že na server příkaz dorazí a server tento příkaz rozpozná, dojde k odeslání tohoto příkazu pomocí tzv. „telnet“ (tedy protokolu pro vzdálený přístup) na port, se kterým byl program *VLC media player* spuštěn. Více o telnetu použitého v této práci viz [Apa13]

Pro spuštění programu *VLC media player* připraveného k dálkovému ovládání přes zadaný port je potřeba přehrávač spustit následujícím příkazem:

```
vlc --rc-host=localhost:1112
```

Uvedený příkaz spustí *VLC media player* s otevřeným síťovým portem `1112`. Na zmíněný port lze následně ze serveru zasílat příkazy. Port je pro ukázkovou generovanou aplikaci napevno nastaven na hodnotu `1112`.

Odesílání příkazů ze serveru na zadaný port *VLC media playeru* pak vypadá například dle následujícího ukázkového zdrojového kódu pro příkaz „pause“:

```
public Object[] pause (Object[] input) {
  inputArray = new String[input.length];
  outputArray = new String[2];
  String command = "pause";
  try {
    tl = new TelnetClient();
    tl.connect("localhost", 1112);
    if(tl.isConnected()) {
      outputArray = new String[2];
    } else {
```

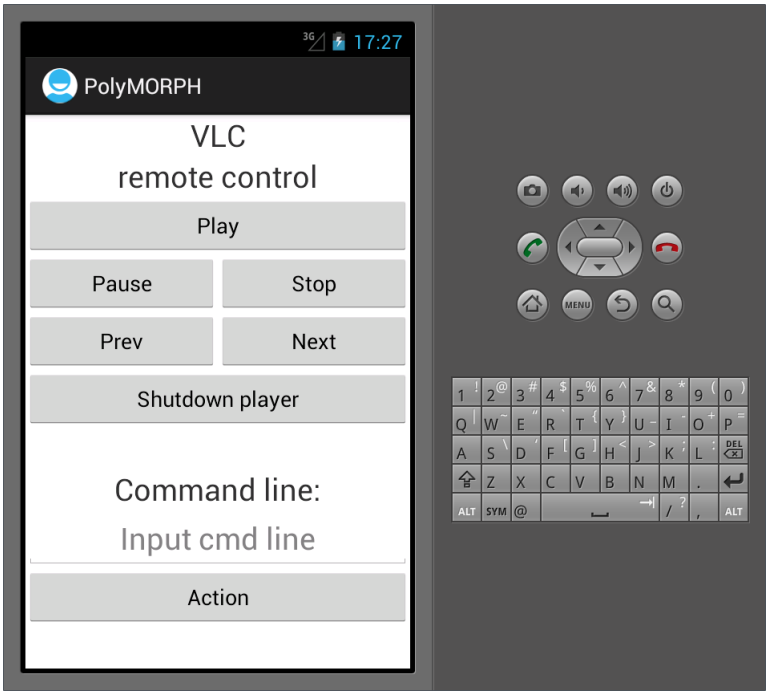
```
        outputArray[0] = "-1";
        outputArray[1] = "Problém s připojením!";
        tl.disconnect();
        return outputArray;
      }
    } catch (Exception e) {
      outputArray = new String[2];
      outputArray[0] = "-1";
      outputArray[1] = "Problém s připojením!";
      return outputArray;
    }
    outputArray[0] = "0";
    outputArray[1] = "Action: " + command;
    try {
      BufferedWriter bw
        = new BufferedWriter(
        new OutputStreamWriter(tl.getOutputStream()));
      bw.write(command);
      bw.flush();
      tl.disconnect();
    } catch (Exception e) {
      outputArray[0] = "-1";
      outputArray[1] = "Action error!";
      return outputArray;
    }
    Object[] outputObject
      = (Object[])outputArray;
    return outputObject;
  }
```

Vytvořená generovaná aplikace ukazuje jednu z možností, jak lze využít zpracování vzdáleného kódu. Po úpravě XML popisu aplikace a změně serverové části by mohla být například použita ve variantách vzdáleného systémového příkazového řádku či jiným obdobným způsobem.

V původním návrhu této generované aplikace byla uvažována varianta vracet ze serveru zpět zprávu o aktuálně přehrávaných souborech. Dle neoficiálních návodů tato možnost „teoreticky“ existuje, avšak po sérii testů bylo zjištěno, že zkoušená verze programu *VLC media player* nevrací zpět prostřednictvím telnet požadované zprávy (o aktuálním stavu a přehrávání). To však nemá vliv na ověření funkcionality

realizované polymorfní aplikace.

Pro tvorbu serverové části byl použit zdroj [Vid10].



Obrázek 9.2: Ukázka generované aktivity „Dálkový multimediální ovladač“ spuštěné v emulátoru.

Ukázka, jak vypadá generovaná aplikace je vidět na obrázku 9.2.

9.3 Zabezpečené hlasovací zařízení

9.3.1 Základní myšlenka

Generovaná aplikace „Zabezpečené hlasovací zařízení“ byla navržena na prezentaci zabezpečené komunikace s využitím šifrovacího algoritmu AES popsaného v kapitole 8.2.3.

Aplikace by se dala využít na odesílání názorů/dotazů studentů na určitou tématiku během konání seminářů a přednášek či k obdobným účelům.

Činnost této aplikace je založena na myšlence použití DEX varianty. DEX knihovna obsahuje proceduru, která komunikuje se serverem jako v případě XML-RPC varianty. Hlavní rozdíl spočívá v tom, že zatímco ve variantě XML-RPC se data posílají v nešifrované podobě, tato DEX knihovna před samotným odesláním dat data zašifruje pomocí AES.

9.3.2 Realizace

Grafické uživatelské rozhraní této generované aplikace je založeno na editovatelném textovém poli a jednom ovládacím prvku. Text, který je zadán do zmíněného editovatelného pole je po aktivování ovládacího prvku (respektive tlačítka) standardně předán DEX knihovně. V této DEX knihovně je na zašifrování textu použito připravené AES šifrování.

Dále je pak zkontrolováno připojení a zpráva pomocí XML-RPC odeslána na server. Zde dochází k problému zmíněnému v kapitole 8.2.3. Ten spočívá v tom, že je nutné veškerou síťovou komunikaci řešit mimo hlavní vlákno aplikace.

K řešení lze použít buď vláken a nebo varianty, která provede asynchronní operace sama (jak již bylo zmíněno v předešlých kapitolách). Označuje se jako tzv. `AsyncTask`, neboli prostředek pro asynchronní zpracování kódu. Je založen na rozdělení kódu, který se má provádět paralelně (v tomto případě síťové operace). Ten je dále rozdělen na dvě části do třídy, která rozšiřuje tzv. `AsyncTask`. Zmíněná třída vytváří jako generický typ, kde prvním typem generika jsou vstupní parametry, dalším je stav pokroku činnosti a posledním výstupní parametry (citace viz [Anw11]). Hlavička této třídy je vidět v následující ukázce:

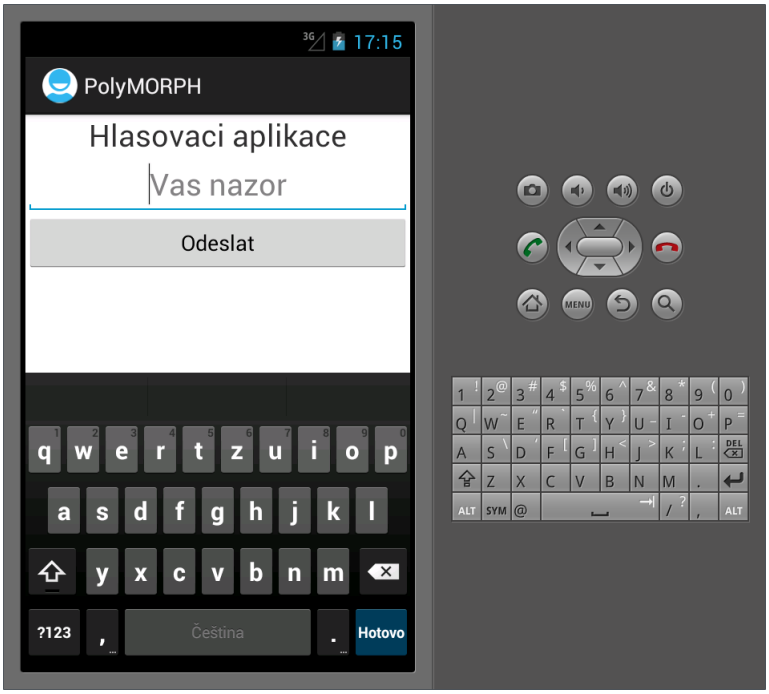
```
class TryResponse extends AsyncTask<String[], Integer,
  Integer>
```

Tato třída v sobě obsahuje více metod. První je část kódu, která je určena pro síťovou komunikaci a je uložena v metodě `doInBackground`. Metoda je provedena asynchronně s hlavním vláknem. Druhá část programového kódu se nachází v metodě `onPostExecute`, která může komunikovat s uživatelským rozhraním spravovaným hlavním vláknem.

Zde dochází ke zmíněnému problému. Volaná DEX knihovna totiž po zašifrování dat vytvoří objekt a předá mu informace pro síťový přenos. Pak předá výsledky zpět do generované aplikace a skončí. V době, kdy již zmíněný přenos je ukončen a je předána návratová hodnota, generovaná aplikace má již odpověď od DEX knihovny a zpracovává další úkony.

K řešení těchto problémů je v aktivitě pro generované aplikace vytvořena metoda `makeText(String text)`, která má za úkol zobrazit v hlavním vlákně aktivity vykreslit informační okno na obrazovce aplikace.

Ukázka, jak vypadá generovaná aplikace je vidět na obrázku 9.3.



Obrázek 9.3: Ukázka generované aktivity „Zabezpečené hlasovací zařízení“ spuštěné v emulátoru.

10 Testování aplikace

Aplikace byla zkoušena na již zmíněném emulátoru vývojového prostředí Eclipse. Polymorfní aplikace byla testována na reálném zařízení *HTC One V* se systémem Android verze 4.0.3.

Pro emulátor byla polymorfní aplikace testována ve verzích Android 4.0 (API 14), 4.0.3 (API 15), 4.1 (API 16) a 4.2 (API 17). Na všech těchto verzích bylo chování polymorfní aplikace stejné a shodné s popisy uvedenými v této diplomové práci.

Na testovaném zařízení fungovala polymorfní aplikace stejně jako na emulátoru. Tím se potvrdila její funkčnost i na „nejnižší“ verzi systému Android, která byla pro tuto práci zvolena.

11 Možnosti budoucího rozšíření

Vytvořenou polymorfní aplikaci lze dále rozšiřovat o funkčnosti a o generované aplikace. Pro rozšíření možných druhů generovaných aplikací je možno do aplikace dodat další prvky grafického uživatelského rozhraní (například obrázkové pole, posuvníky či jiné ovládací prvky).

Polymorfní aplikace je připravena na rozšíření pro přidávání více aktivit do jedné generované aplikace. K tomu lze využít adresy dalších XML popisů generovaných aplikací, které lze ukládat pod proměnou `URL2` připravenou ve sdílených preferencích.

Dalším možným rozšířením aplikace by bylo přidání více různých druhů šifrování včetně možnosti práce s digitálně podepsanými soubory.

12 Závěr

Úkolem diplomové práce bylo navrhnout a realizovat princip polymorfní aplikace pro mobilní operační systém Android. V rámci návrhu byly probrány různé možnosti, jak by šlo tuto polymorfní aplikaci vytvořit.

Jako nejvhodnější pro realizaci polymorfní aplikace byly zvoleny dva druhy řešení. První je zpracování kódu prostřednictvím DEX knihovny získané ze vzdáleného úložiště. Druhou realizovanou možností je využívání vzdálených procedur. Obě tyto varianty slouží k tomu, aby aplikace pokryla svou funkčností co nejvíce možných způsobů chování.

K tomu, aby bylo možné definovat chování a grafické uživatelské rozhraní polymorfní aplikace, byl navržen předpis ve formátu XML. Princip použití tohoto předpisu je takový, že po předání zmíněného předpisu dojde k dynamickému vytvoření generované aplikace. Pomocí navrženého XML předpisu lze pro grafické uživatelské rozhraní určit, jaké objekty a na jakých místech se mají vyskytovat. Objekty, které lze použít pro grafické uživatelské rozhraní generované aplikace, jsou informační popisek, editovatelné textové pole, tlačítko a graf. Zmíněnými objekty lze navrhnout prakticky jakoukoli formulářovou aplikaci. Výhoda spočívá v tom, že po úpravě XML předpisu může generovaná aplikace plnit úplně jiný úkol (respektive mít odlišnou funkčnost).

Část práce byla také věnována zabezpečení navrhnuté polymorfní aplikace. Nejdříve byly probrány druhy porušení bezpečnosti a bezpečnostních rizik. Rizika bylo potřeba nějakým způsobem eliminovat tak, aby bylo možné i další používání polymorfní aplikace. Jedním z realizovaných řešení bylo zabezpečení založené na základě šifrování dat a XML popisů generovaných aplikací. Druhá varianta zabezpečení spočívá v omezení práv pro polymorfní aplikaci uvedených v manifestu aplikace. Obě zmíněná řešení zabezpečení pokrývají nově vzniklé bezpečnostní problémy.

Pro ověření praktické použitelnosti a velkého rozsahu různých funkčností byla vytvořena sada generovaných aplikací. Sada generovaných aplikací obsahuje grafickou kalkulačku, dálkové ovládání pro aplikaci *VLC media player* a zabezpečené hlasovací zařízení. Vytvořené aplikace mají odlišné funkcionality a využívají jak zmíněných knihoven získaných ze vzdálených úložišť, tak i vzdálených procedur poskytovaných servery. Generované aplikace jsou pro polymorfní aplikaci popsány XML soubory. Ty jsou vidět v příloze A této práce. Pro budoucí vývojáře slouží jako ukázka a vzor možného principu, jak lze tyto generované aplikace vyvíjet.

Literatura

[Anu13] Using Hardware Devices. *Android Developers* [online]. 2013 [cit. 2013-04-16].
Dostupné z: http://developer.android.com/tools/device.html.

[Anw11] Vícevláknové zpracování. *AndroidWiki* [online]. 6. 5. 2011, 22. 9. 2011 [cit. 2013-04-09].
Dostupné z: http://androidwiki.cz/Vícevláknové_zpracování.

[Apa10] THE APACHE SOFTWARE FOUNDATION. *Ws-xmlrpc: Apache XML-RPC* [online]. 2001-2010 [cit. 2013-03-30].
Dostupné z: http://ws.apache.org/xmlrpc/.

[Apa13] *Apache Commons Net: Overview* [online]. 2001-2013 [cit. 2013-04-09].
Dostupné z: http://commons.apache.org/proper/commons-net/.

[Apc10] The Apache XML-RPC Client. THE APACHE SOFTWARE FOUNDATION. *Ws-xmlrpc: Apache XML-RPC* [online]. 2001-2010, 16.2.2010 [cit. 2013-03-30].
Dostupné z: http://ws.apache.org/xmlrpc/client.html.

[Apl12] *Androidplot.com: An Android API for creating charts and plots.* [online]. 2012 [cit. 2013-03-13].
Dostupné z: http://androidplot.com/.

[App13] Titanium Mobile Application Development. *Appcelerator Inc.* [online]. 2008-2013 [cit. 2013-04-29].
Dostupné z: http://www.appcelerator.com/platform/titanium-platform/.

[Aps10] The Apache XML-RPC Server. THE APACHE SOFTWARE FOUNDATION. *Ws-xmlrpc: Apache XML-RPC* [online]. 2001-2010, 16.2.2010 [cit. 2013-03-30].
Dostupné z: http://ws.apache.org/xmlrpc/server.html.

[Ber03] BERÁNEK, Marek, Tomáš LÍPA a Ondřej PODZIMEK. Kryptologie: Elektronický podpis. In: *Univerzita Hradec Králové* [online]. 2003, 9.6.2003 [cit. 2013-04-10].
Dostupné z: http://kryptologie.uhk.cz/54.htm.

[Cof13] CORBA FAQ. *CORBA* [online]. 02/14/2013 [cit. 2013-04-02].
Dostupné z: http://www.omg.org/gettingstarted/corbafaq.htm.

[Cos13] CORBA 3.3. *CORBA* [online]. 2012 [cit. 2013-04-02].
Dostupné z: http://www.omg.org/spec/CORBA/3.3/.

[Ded13] Dashboard. *Android Developers* [online]. 2013, 2 April 2013 [cit. 2013-04-06].
Dostupné z: http://developer.android.com/about/dashboards/index.html.

[Dev13] Manifest.permission. *Android Developers* [online]. 2013 [cit. 2013-03-12].
Dostupné z: http://developer.android.com/reference/android/Manifest.permission.html.

[Dob10] DOBJANSCHI, Virgil. Developing Android REST client applications. *Google I/O 2010* [online]. 2010 [cit. 2013-04-02].
Dostupné z: http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html.

[Dol01] DOLEŽAL, Petr. AES: šifra pro třetí tisíciletí. MATEMATICKO-FYZIKÁLNÍ FAKULTA UNIVERZITY KARLOVY. *Gumysh.matfyz.cz* [online]. 6. července 2001 [cit. 2013-04-07].
Dostupné z: http://www.jikos.cz/ gumysh/docs/AES/index.html.

[Ecm11] ECMA-262. *Standard ECMA-262: ECMAScript Language Specification.* Geneva: Ecma International, 2011.
Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

[Eno13] J2ME Polish. *Enough Software: App Development for iOS, Android, BlackBerry, Windows Phone, J2ME and HTML5* [online]. 2005 [cit. 2013-04-29].
Dostupné z: http://www.enough.de/products/j2me-polish/.

[Gar11] GARCÍA, Álvaro Vega. TIDorb ported to Android. *Morfeo CORBA Platform* [online]. 10 October 2011 [cit. 2013-04-02].
Dostupné z: http://corba.morfeo-project.org/archives/tidorb-ported-to-android.

[Goo09] Android-json-rpc: A json-rpc client library for android. *Google Code* [online]. 2009 [cit. 2013-03-12].
Dostupné z: https://code.google.com/p/android-json-rpc/.

[Goo10] Ksoap2-android: A lightweight and efficient SOAP library for the Android platform. *Google Code* [online]. 2010 [cit. 2013-04-02].
Dostupné z: https://code.google.com/p/ksoap2-android/.

[Hul12] HULA, Josef. *Dynamická tvorba aplikací v systému Android* [online]. 2012 [cit. 2013-02-26]. Bakalářská práce. ZÁPADOČESKÁ UNIVERZITA V PLZNI, Fakulta aplikovaných věd. Vedoucí práce Ladislav Pešička.
Dostupné z: http://theses.cz/id/vf9jn8/.

[Chu11] CHUNG, Fred. Custom Class Loading in Dalvik. In: *Android Developers Blog* [online]. 28 July 2011 [cit. 2013-04-07].
Dostupné z: http://android-developers.blogspot.cz/2011/07/custom-class-loading-in-dalvik.html.

[Ibm13] JAR files revealed: Explore the power of the JAR file format. *IBM: developerWorks* [online]. [cit. 2013-03-10].
Dostupné z: http://www.ibm.com/developerworks/library/j-jar/.

[Int99] THE INTERNET SOCIETY. *RFC 2616 - Hypertext Transfer Protocol: HTTP/1.1* [online]. 1999, s. 176 [cit. 2013-04-02].
Dostupné z: http://tools.ietf.org/html/rfc2616.

[Jav12] RPC: Remote Procedure Call protocol. *Javvin: Network Protocols Guide, Network Monitoring & Analysis Tools* [online]. 2012 [cit. 2013-03-10].
Dostupné z: http://www.javvin.com/protocolRPC.html.

[Jin13] *J-Interop: Pure Java-DCOM Bridge* [online]. [cit. 2013-04-02].
Dostupné z: http://j-interop.org.

[Jso13] Úvod do JSON. *JSON* [online]. [cit. 2013-03-11].
Dostupné z: http://www.json.org/json-cz.html.

[Kos01] KOSEK, Jiří. Parsery: XML & aplikace. *XML pro každého* [online]. 2000-2001 [cit. 2013-03-30].
Dostupné z: http://www.kosek.cz/clanky/swn-xml/ar02s30.html.

[Lin11] LINTVELT, Herman. One CODE to rule them all. In: *RichClientGUI* [online]. 8 Jun 2011 [cit. 2013-04-06].
Dostupné z: http://blog.richclientgui.com/?p=449.

[Lyc11] Java RMI Overview. TAING, Nguonly. *LYCOG* [online]. 8.3.2011 [cit. 2013-03-13].
Dostupné z: http://lycog.com/distributed-systems/java-rmi-overview/.

[Mal09] MALÝ, Martin. REST: architektura pro webové API. *Zdroják: o tvorbě webových stránek a aplikací* [online]. 3.8.2009 [cit. 2013-04-02].
Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/.

[Mic11] MICHL, Petr. Je lepší nativní aplikace nebo mobilní web?. *Marketing journal: marketing, public relations, reklama, internet* [online]. 25. 6. 2012 [cit. 2013-04-06].
Dostupné z: http://www.m-journal.cz/cs/internet/Je-lepsi-nativni-aplikace-nebo-mobilni-web_s281x9241.html.

[Mot13] RhoMobile Suite. MOTOROLA SOLUTIONS, Inc. *Motorola Solutions Homepage: Motorola Solutions USA* [online]. 2013 [cit. 2013-04-29].
Dostupné z: http://www.motorola.com/Business/US-EN/Business+Product+and+Services/Software+and+Applications/RhoMobile+Suite.

[Msd07] Distributed Component Object Model (DCOM) Remote Protocol Specification. *MSDN: the Microsoft Developer Network* [online]. 2007 [cit. 2013-04-02].
Dostupné z: http://msdn.microsoft.com/library/cc201989.aspx.

[Mun07] Šifrování. In: FAKULTA INFORMATIKY MASARYKOVY UNIVERZITY. *FI WIKI* [online]. 2007, 25. 5. 2007 [cit. 2013-04-13].
Dostupné z: http://kore.fi.muni.cz/wiki/index.php/Šifrování.

[Mur11] MURPHY, Mark L. *Android 2: průvodce programováním mobilních aplikací*. Vyd. 1. Brno: Computer Press, 2011, 375 s. ISBN 978-80-251-3194-7.

[Ocs10] DistributedObjects. OCSOFTWARE. *OCSite* [online]. 2010 [cit. 2013-04-29].
Dostupné z: http://www.ocs.cz/text/NeXTStep/DistributedObjects.html.

[Pho13] *PhoneGap* [online]. 2013 [cit. 2013-04-29].
Dostupné z: http://phonegap.com/.

[Rho09] RHODES, Loren K. ODMG and CORBA. *Juniata College* [online]. 2009, 20.10.2009 [cit. 2013-04-02].
Dostupné z: http://jcsites.juniata.edu/faculty/rhodes/dbms/odmg.htm.

[Sci11] *XML-RPC.Com: Simple cross-platform distributed computing, based on the standards of the Internet.* [online]. Scripting News, Inc., 2004-2011 [cit. 2013-03-30].
Dostupné z: http://xmlrpc.scripting.com/.

[Scr11] Dalvik, virtual machine of Android. *Scriptol.com: Programming with web standards* [online]. 2011 [cit. 2013-03-10].
Dostupné z: http://www.scriptol.com/programming/dalvik.php.

[Sdk13] Android SDK. *Android Developers* [online]. 2013 [cit. 2013-03-12].
Dostupné z: http://developer.android.com/sdk/index.html.

[Sch13] SCHREINER, Pavel a Petr HRUŠKA. ČVUT. *Java RMI* [online]. [cit. 2013-05-01].
Dostupné z: http://kmdec.fjfi.cvut.cz/ virius/prednes/prezen/rmi.pdf.

[Sra08] ŠRAJER, Michal. INMITE S.R.O. *Android session: 1. poločas* [online]. 2008, 32 s. [cit. 2013-03-10].
Dostupné z: http://android.inmite.eu/data/android_slajdy_01.pdf.

[Tom13] Tomcat Web Application Deployment. THE APACHE SOFTWARE FOUNDATION. *Apache Tomcat 7.0* [online]. 1999-2013, Mar 22 2013 [cit. 2013-04-16].
Dostupné z: http://tomcat.apache.org/tomcat-7.0-doc/deployer-howto.html.

[Vid10] Controlling VLC via rc from java. In: *The VideoLAN Forums* [online]. 15 December 2010 [cit. 2013-04-09].
Dostupné z: http://forum.videolan.org/viewtopic.php?f=14&t=85347.

[Vir10] VIRTUAL MOBILE TECHNOLOGIES. *RAMP Secure Mobile Enterprise Application Platform (MEAP)* [online]. 2010 [cit. 2013-04-29].
Dostupné z: http://ramp.virtualmobiletech.com/.

[Vlc12] *VideoLAN: Official page for VLC media player, the Open Source video framework!* [online]. 1996-2012 [cit. 2013-04-09].
Dostupné z: http://www.videolan.org/vlc/.

[W3j99] JSON Tutorial. *W3schools* [online]. 1999-2013 [cit. 2013-04-02].
Dostupné z: http://www.w3schools.com/json/.

[W3s99] SOAP Tutorial. *W3schools* [online]. 1999-2013 [cit. 2013-04-02].
Dostupné z: http://www.w3schools.com/soap/.

A Ukázky vytvořených XML

A.1 Grafická kalkulačka - RPC

V příloze je vidět XML popis, který je vytvořen k aplikaci „Grafická kalkulačka“ ve verzi RPC. Generovaná aplikace byla podrobně popsána v kapitole 9.1.

Popis XML obsahuje celkem osm prvků. Prvních šest prvků popisu je uspořádáno ve dvojicích. Tyto dvojice jsou složeny z textového popisku (neboli `textview`) a editovatelného textového pole (`edittext`).

Sedmým prvkem XML popisu je ovládací prvek (`button`), jehož úkolem je definovat, jakým způsobem bude prováděno zpracování aplikace. V uvedeném případě jde o zpracování pomocí RPC na serveru.

Posledním prvkem je objekt typu graf, ve kterém se zobrazují výsledné průběhy získané z provedené RPC procedury na serveru.

```
<application>
  <object>
    <type>textview</type>
    <id>0</id>
    <nameOfObject>testobject0</nameOfObject>
    <text>Typ fce:</text>
    <onRow>1</onRow>
  </object>

  <object>
    <type>edittext</type>
    <id>1</id>
    <nameOfObject>testobject1</nameOfObject>
    <text>sin,cos</text>
    <onRow>1</onRow>
  </object>

  <object>
    <type>textview</type>
    <id>2</id>
```

```
  <nameOfObject>testobject2</nameOfObject>
  <text>Pocatek:</text>
  <onRow>2</onRow>
</object>

<object>
  <type>edittext</type>
  <subtype>sigdecimal</subtype>
  <id>3</id>
  <nameOfObject>testobject3</nameOfObject>
  <text>0</text>
  <onRow>2</onRow>
</object>

<object>
  <type>textview</type>
  <id>4</id>
  <nameOfObject>testobject4</nameOfObject>
  <text>Konec:</text>
  <onRow>3</onRow>
</object>

<object>
  <type>edittext</type>
  <subtype>sigdecimal</subtype>
  <id>5</id>
  <nameOfObject>testobject5</nameOfObject>
  <text>1</text>
  <onRow>3</onRow>
</object>

<object>
  <type>button</type>
  <id>6</id>
  <nameOfObject>testobject6</nameOfObject>
  <text>Ukaz vysledek</text>
  <onRow>5</onRow>
  <functionAddress>
    http://192.168.10.103:8080/xml-rpc-server/xmlrpc
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
```

```
    <nameOfFunction>Calculator.grahpsSinCos</nameOfFunction>
    <idOfResult>7</idOfResult>
  </object>

  <object>
    <type>graph</type>
    <id>7</id>
    <nameOfObject>testobject7</nameOfObject>
    <title>Zobrazeni grafu</title>
    <onRow>4</onRow>
    <axisX>Osa X</axisX>
    <axisY>Osa Y</axisY>
  </object>
</application>
```

A.2 Grafická kalkulačka - DEX

V příloze je vidět XML popis, který je vytvořen k aplikaci „Grafická kalkulačka“ ve verzi DEX. Popis obsahuje celkem osm prvků.

Popis varianty je podobný jako v předchozí příloze A.1. Jediným rozdílem je definice ovládacího prvku. Ten má v XML popisu nastaven způsob zpracování DEX. To definuje, že je potřeba získat určenou DEX knihovnu. Ta je v následujícím popisu definována adresou DEX knihovny, typem a jménem funkce, která má být ve zmíněné DEX knihovně zpracována.

```
<application>
  <object>
    <type>textview</type>
    <id>0</id>
    <nameOfObject>testobject0</nameOfObject>
    <text>Typ fce:</text>
    <onRow>1</onRow>
  </object>

  <object>
    <type>edittext</type>
    <id>1</id>
    <nameOfObject>testobject1</nameOfObject>
    <text>sin,cos</text>
    <onRow>1</onRow>
   </object>

  <object>
    <type>textview</type>
    <id>2</id>
    <nameOfObject>testobject2</nameOfObject>
    <text>Pocatek:</text>
    <onRow>2</onRow>
  </object>

  <object>
    <type>edittext</type>
    <subtype>sigdecimal</subtype>
    <id>3</id>
```

```
  <nameOfObject>testobject3</nameOfObject>
  <text>0</text>
  <onRow>2</onRow>
</object>

<object>
  <type>textview</type>
  <id>4</id>
  <nameOfObject>testobject4</nameOfObject>
  <text>Konec:</text>
  <onRow>3</onRow>
</object>

<object>
  <type>edittext</type>
  <subtype>sigdecimal</subtype>
  <id>5</id>
  <nameOfObject>testobject5</nameOfObject>
  <text>1</text>
  <onRow>3</onRow>
</object>

<object>
  <type>button</type>
  <id>6</id>
  <nameOfObject>testobject6</nameOfObject>
  <text>Ukaz vysledek</text>
  <onRow>5</onRow>
  <functionAddress>
    http://home.zcu.cz/~staffa/DexLibraryGraph.jar
  </functionAddress>
  <typeOfFunction>dex</typeOfFunction>
  <nameOfFunction>Calculator.grahpsSinCos</nameOfFunction>
  <idOfResult>7</idOfResult>
</object>

<object>
  <type>graph</type>
  <id>7</id>
  <nameOfObject>testobject7</nameOfObject>
  <title>Zobrazeni grafu</title>
```

```
    <onRow>4</onRow>
    <axisX>Osa X</axisX>
    <axisY>Osa Y</axisY>
  </object>
</application>
```

A.3 Dálkový multimediální ovladač

Příloha ukazuje XML popis, který je vytvořen k aplikaci „Dálkový multimediální ovladač“ popsané v kapitole 9.2.

Popis aplikace obsahuje dvanáct prvků. První dva prvky obsahují informační popisky. Dalších šest prvků obsahuje tlačítka pro jednotlivé akce, které byly vybrány jako nejpoužívanější.

Následující prvek obsahuje popisek pouze s jednou mezerou pro optické oddělení prvků grafického rozhraní.

Poslední tři prvky patří do sekce pro práci s přehrávačem přes příkazovou řádku. Prvním je popisek, druhý je editovatelné textové pole a posledním je prvek s funkčností RPC.

```
<application>
  <object>
    <type>textview</type>
    <id>0</id>
    <nameOfObject>testobject0</nameOfObject>
    <text>VLC</text>
    <onRow>1</onRow>
  </object>

  <object>
    <type>textview</type>
    <id>1</id>
    <nameOfObject>testobject1</nameOfObject>
    <text>remote control</text>
    <onRow>2</onRow>
  </object>

  <object>
    <type>button</type>
    <id>2</id>
    <nameOfObject>testobject2</nameOfObject>
    <text>Play</text>
    <onRow>3</onRow>
    <functionAddress>
      http://192.168.10.103:8080/xml-rpc-server/xmlrpc
```

```
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
  <nameOfFunction>VLCRemoter.play</nameOfFunction>
  <idOfResult>-1</idOfResult>
</object>

<object>
  <type>button</type>
  <id>3</id>
  <nameOfObject>testobject3</nameOfObject>
  <text>Pause</text>
  <onRow>4</onRow>
  <functionAddress>
    http://192.168.10.103:8080/xml-rpc-server/xmlrpc
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
  <nameOfFunction>VLCRemoter.pause</nameOfFunction>
  <idOfResult>-1</idOfResult>
</object>

<object>
  <type>button</type>
  <id>4</id>
  <nameOfObject>testobject4</nameOfObject>
  <text>Stop</text>
  <onRow>4</onRow>
  <functionAddress>
    http://192.168.10.103:8080/xml-rpc-server/xmlrpc
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
  <nameOfFunction>VLCRemoter.stop</nameOfFunction>
  <idOfResult>-1</idOfResult>
  </object>

<object>
  <type>button</type>
  <id>5</id>
  <nameOfObject>testobject5</nameOfObject>
  <text>Prev</text>
  <onRow>5</onRow>
  <functionAddress>
```

```
    http://192.168.10.103:8080/xml-rpc-server/xmlrpc
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
  <nameOfFunction>VLCRemoter.prev</nameOfFunction>
  <idOfResult>-1</idOfResult>
</object>

<object>
  <type>button</type>
  <id>6</id>
  <nameOfObject>testobject6</nameOfObject>
  <text>Next</text>
  <onRow>5</onRow>
  <functionAddress>
    http://192.168.10.103:8080/xml-rpc-server/xmlrpc
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
  <nameOfFunction>VLCRemoter.next</nameOfFunction>
  <idOfResult>-1</idOfResult>
</object>

<object>
  <type>button</type>
  <id>7</id>
  <nameOfObject>testobject7</nameOfObject>
  <text>Shutdown player</text>
  <onRow>6</onRow>
  <functionAddress>
    http://192.168.10.103:8080/xml-rpc-server/xmlrpc
  </functionAddress>
  <typeOfFunction>rpc</typeOfFunction>
  <nameOfFunction>VLCRemoter.quit</nameOfFunction>
  <idOfResult>-1</idOfResult>
</object>

<object>
  <type>textview</type>
  <id>8</id>
  <nameOfObject>testobject8</nameOfObject>
  <text> </text>
  <onRow>7</onRow>
```

```
  </object>

  <object>
    <type>textview</type>
    <id>9</id>
    <nameOfObject>testobject9</nameOfObject>
    <text>Command line:</text>
    <onRow>8</onRow>
  </object>

  <object>
    <type>edittext</type>
    <id>10</id>
    <nameOfObject>testobject10</nameOfObject>
    <text>Input cmd line</text>
    <onRow>9</onRow>
  </object>

  <object>
    <type>button</type>
    <id>11</id>
    <nameOfObject>testobject11</nameOfObject>
    <text>Action</text>
    <onRow>10</onRow>
    <functionAddress>
      http://192.168.10.103:8080/xml-rpc-server/xmlrpc
    </functionAddress>
    <typeOfFunction>rpc</typeOfFunction>
    <nameOfFunction>VLCRemoter.remote</nameOfFunction>
    <idOfResult>-1</idOfResult>
  </object>
</application>
```

A.4 Zabezpečené hlasovací zařízení

V příloze je vidět ukázka XML popisu generované aplikace nazvané „Zabezpečené hlasovací zařízení“ popsané v kapitole 9.3.

Aplikace se skládá ze tří prvků.

- Prvním prvkem je popisek aplikace.
- Druhým prvkem je editovatelné textové pole. To slouží pro zadání názoru uživatele.
- Poslední prvek je ovládací tlačítko s DEX funkčností.

```
<application>
  <object>
    <type>textview</type>
    <id>0</id>
    <nameOfObject>testobject0</nameOfObject>
    <text>Hlasovaci aplikace</text>
    <onRow>1</onRow>
  </object>

  <object>
    <type>edittext</type>
    <id>1</id>
    <nameOfObject>testobject1</nameOfObject>
    <text>Vas nazor</text>
    <onRow>2</onRow>
  </object>

  <object>
    <type>button</type>
    <id>2</id>
    <nameOfObject>testobject2</nameOfObject>
    <text>Odeslat</text>
    <onRow>3</onRow>
    <functionAddress>
      http://home.zcu.cz/~staffa/DexLibrarySecure.jar
    </functionAddress>
```

```
    <typeOfFunction>dex</typeOfFunction>
    <nameOfFunction>Test.echo</nameOfFunction>
    <idOfResult>-1</idOfResult>
  </object>
</application>
```

B Ukázka kódu pro DexClassLoader

Tato příloha ukazuje zdrojový kód pro `DexClassLoader` popsaný v textu práce.

```
final File optimizedDexOutputPath
  = getDir("outdex", Context.MODE_PRIVATE);

DexClassLoader cl
  = new DexClassLoader(dexInternalStoragePath
    .getAbsolutePath(),
    optimizedDexOutputPath.getAbsolutePath(),
    null,
    getClassLoader());

Class libProviderClazz = null;

try {
  libProviderClazz
    = cl.loadClass("com.android.dpapp1."
    + nameOfFileWithoutExt);
  LibraryInterface lib
    = (LibraryInterface) libProviderClazz.newInstance();

  ArrayList<String> parameters
    = new ArrayList<String>();
  ArrayList<String> outParameters
    = new ArrayList<String>();

  /**
  * Prochazeni vsech objektu a u tech,
  * ktere jsou edittextem zjisteni
  * jejich atributu popisek
  */
  for (int i = 0;
    i < objectsList.getNameOfObject().size();
    i++) {
      if (objectsList
        .getType().get(i).trim()
        .toLowerCase().equals("edittext")) {
```

```
            int idOfCurrentObject
              = objectsList.getID().get(i);
            parameters.add
              (edittexts[idOfCurrentObject]
              .getText().toString());
        }
    }

    /** vyuziti knihovni funkce stazene ze site */
    outParameters
      = lib.doLibraryFunction(nameOfFunction, parameters);

    int returnCode = 0;
    try {
      returnCode
        = Integer.parseInt(outParameters.get(0));
    } catch (Exception e) {
      returnCode = -1;
      ...
      return;
    }
```

C Zprovoznění polymorfní aplikace

V příloze jsou popsány způsoby spuštění polymorfní aplikace. Pro příklad lze použít následující dva způsoby:

- Spuštění přes emulátor programu *Eclipse*.
- Spuštění v reálném zařízení.

Tyto způsoby jsou popsané dále.

C.1 Spuštění přes emulátor

Pro spuštění polymorfní aplikace na emulátoru lze využít připraveného vývojového prostředí [Sdk13]. Zde lze také najít návody na zprovoznění prostředí a jeho konfiguraci. Dále je v tomto zdroji možno najít návody na stažení balíků potřebných pro emulátor a správný běh aplikace.

Po nainstalování a spuštění vývojového prostředí je potřeba naimportovat projekt polymorfní aplikace. To provedeme zvolením následující položky `File` → `Import` → `Existing Android Code Into Workspace`. Potvrdíme tlačítkem `Next`. V dalším okně vybereme `Root Directory` a vložíme adresu existujícího projektu polymorfní aplikace. Před pokračováním zaškrtneme položku `Copy projects into workspace`. Tlačítkem `Finish` dokončíme import projektu.

Následujícím krokem je vytvoření virtuálního zařízení pro Android. Pro vytvoření virtuálního zařízení je potřeba ve složce, kde je umístěn Android, na daném počítači spustit tzv. „Android Virtual Device Manager“ (soubor `AVD Manager.exe`). V něm lze vytvořit pomocí tlačítka `New...` nové virtuální zařízení dle vlastních potřeb.

Po vytvoření potřebného virtuálního zařízení stačí zařízení zvolit v nabídce a stisknout tlačítko `Start`. Tím dojde ke spuštění emulátoru s mobilním systémem Android. Ten bude spuštěn s preferencemi, které byly nastaveny během jeho vytváření.

Pro samotné spuštění pak v Eclipse klepneme pravým tlačítkem na adresář projektu polymorfní aplikace, který jsme na začátku tohoto návodu importovali. Zvolíme položku `Run As` → `Android Appliaction`. Tím dojde ke spuštění polymorfní aplikace v běžícím emulátoru.

C.2 Spuštění na reálném zařízení

Spuštění na reálném zařízení, na kterém je nasazen mobilní systém Android lze dvěma způsoby.

- První možností je využití Eclipse pro spuštění polymorfní aplikace na mobilním zařízení - viz návod [Anu13].
- Druhou možností je nahrání souboru `.apk` ze složky projektu do zařízení a nainstalovat polymorfní aplikaci přímo v zařízení.

D Zprovoznění serveru

Některé aplikace pro svou činnost využívají vytvořených RPC serverů, které je potřeba před spuštěním polymorfní aplikace nasadit a spustit.

Nasazení a spuštění serverů Tomcat, které byly při vývoji používány je popsáno v [Tom13].

Druhým způsobem je spuštění serveru z *Eclipse IDE for Java EE Developers* s nainstalovanou Java 7. Eclipse je potřeba nakonfigurovat:

Nejdříve zvolíme `Window` → `Preferences` → `Server` → `Runtime Enviroments` → `Add`. Následně pak vybereme `Apache` → `Apache Tomcat v7.0` a potvrdíme tlačítkem `Next`. V dalším okně je třeba doplnit cestu k adresáři Tomcat install dir, kde se Tomcat nachází. Potvrdíme tlačítkem `Finish`.

Dalším krokem je přidání serveru v Eclipse pomocí `File` → `New` → `Other` → `Server` a po stisknutí tlačítka `Next` vyplnit následující:

- `host name = localhost`
- `type = Tomcat v7.0 Server`
- `runtime` = viz instalační cesta k adresáři, kde se Tomcat nachází.

Nastavení potvrdíme tlačítkem `Finish`.

Posledním krokem před spuštěním je přiřazení Tomcat serveru projektu. Do Eclipse pak stačí naimportovat projekt obsahující serverovou část práce a klepnutím pravým tlačítkem myši na `Tomcat v7.0 Server at localhost` přes nabídku `Add and Remove...` přiřadit Tomcat server projektu.

Spuštění serveru se pak provede klepnutím pravým tlačítkem na projekt a zvolením položky `Run As` → `Run on Server`. Po spuštění se zobrazí okno interního prohlížeče Eclipse s adresou běžícího serveru. To by mohlo vypadat například takto:

`http://localhost:8080/xml-rpc-server/`

Na konec adresy pak stačí přidat `/xmlrpc` a stránku znovu načíst (například pomocí tlačítka refresh). Pokud bude stránka hlásit chybu číslo `405`, tak server s RPC je připraven přijímat požadavky od klienta.