Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Dynamická tvorba aplikací v systému Android

Plzeň, 2012 Josef Hula

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných zdrojů.

V Plzni dne 9. května 2012 Josef Hula

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Ladislavu Pešičkovi za odborné vedení a cenné rady při vypracování bakalářské práce.

Abstract

Dynamic application development for Android

The subject of my bachelor thesis is to explore possibilities of remote code execution on Android platform. The thesis examines how to create dynamic user interface according the information downloaded from the server and how to add program actions to the user interface. The result is an application which downloads XML file from server and according this file create user interface. As an example is realized calculator and application for temperature transfer.

Obsah

1. Úvod

Cílem této práce je prozkoumat možnosti, jak lze v systému Android provozovat aplikace včetně příslušného grafického prostředí, kde jejich uživatelské rozhraní a funkcionalita budou definovány prostřednictvím externích popisů. Stejná aplikace může tedy vykazovat různou funkcionalitu na základě předpisu staženého ze serveru. Při návrhu budou zváženy související bezpečnostní aspekty. Funkcionalita bude ověřena na ukázkové aplikaci.

Práce v první části obsahuje stručný přehled verzí platformy Android, možnosti a způsoby vývoje aplikací pro tento systém. Zkoumá mechanismy spouštění vzdáleného kódu, a to jak spouštění na lokálním uzlu, tak na externím uzlu. Dále zjišťuje, zdali a prostřednictvím jakých knihoven jsou tyto mechanismy dostupné pro Android.

Na základě analýzy je navržen systém pro dynamické vytvoření jednoduchého uživatelského rozhraní podle informací stažených ze serveru. Následně je navržen mechanismus pro přiřazení různých programových akcí uživatelskému rozhraní dle informací ze serveru.

V poslední části je navržený systém realizován a funkcionalita ověřena na uvedených testovacích aplikacích.

2. Android

Android [1] je open source softwarová platforma společnosti Google založena na linuxovém jádře, která vznikla v roce 2007. Používá se na mobilních zařízeních, zejména na smartphonech a tabletech, ale rozšiřuje se i na další zařízení např. televize.

2.1. Verze systému

Android 1.0 byl oficiálně představen v roce 2008. Vývoj pokračoval verzemi 1.5 Cupcake a 1.6 Donut. V roce 2009 přišla aktualizace na verzi 2.0/2.1 Enclair následována verzemi 2.2 Froyo a 2.3 Gingerbread. Výhradně pro tablety byl vytvořen Android 3.0 Honeycomb. S příchodem Androidu 4.0 Ice Cream Sandwich došlo ke sloučení těchto dvou vývojových větví do jedné a tento systém je používán jak pro smartphony, tak pro tablety.

2.2. Vývoj aplikací

Vývoj pro Android probíhá primárně v jazyce Java. Pro vývoj aplikací jsou dostupné Android SDK [2] a ADT plugin pro Eclipse [3]. Je možné vyvíjet i v jazyce C/C++ a to použitím Android NDK [4]. Vývoj probíhá ve Windows, MacOS a Linuxu.

3. Spouštění vzdáleného kódu

Vzdáleným kódem rozumíme kód, který se nachází na nějakém zařízení v síti. Na spouštění vzdáleného kódu se dá nahlížet z dvou pohledů. První pohled je, že aplikace vykoná proceduru, která je umístěna na jiném místě v síti, než na kterém se nachází volající program. Může se například jednat o výpočet funkce na jiném počítači v síti. Druhou možností je, že aplikace ze serveru získá kód, který následně sama zpracuje a vykoná. Dochází tedy ke spuštění kódu buď na lokálním uzlu anebo na externím uzlu.

V této práci se budeme zabývat především spouštěním externího kódu na lokálním uzlu, ale prozkoumáme i možnosti spouštění kódu na externím uzlu (viz tab. 3.1). Nejprve zjistíme, jaké jsou možnosti pro spouštění vzdáleného kódu na externím uzlu.

Metoda	Implementace pro Android
Externí uzel	
RPC	ano
Java RMI	ne
CORBA	ano
DCOM	nezjištěno
XML-RPC	ano
JSON-RPC	ano
SOAP	ano
REST	ano
Lokální uzel	
Externí Java třída	ano
Externí XML soubor	ano
Externí JSON soubor	ano

Tabulka 3.1 Shrnutí možností pro spouštění vzdáleného kódu

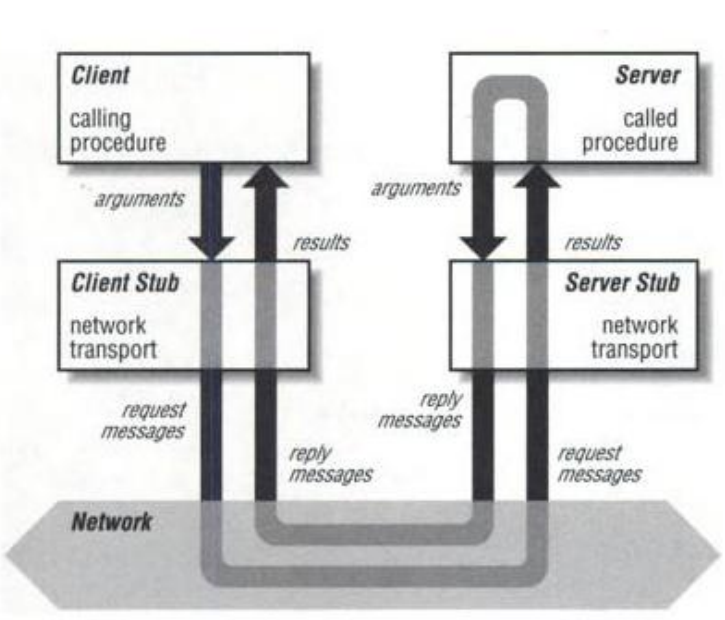
3.1. Remote Procedure Call

Remote Procedure Call (RPC) [5] je jednou z metod pro komunikaci programů na dálku a jedná se o obdobu lokálního volání procedury nebo funkce. Metoda je založena na modelu klient/server, kde klient odesílá serveru požadavky a od serveru očekává odpověď. Volání služeb je zabaleno do volání vzdálené procedury s předáváním parametrů. RPC tedy umožňuje programu volat proceduru umístěnou na vzdáleném počítači v síti nebo na serveru. Příkladem využití této technologie může být výpočet funkce na jiném počítači v síti.

Komunikace mezi klientem a serverem probíhá tak, že klientská procedura odešle požadavek vzdálené proceduře na serveru a ta po vykonání požadavku zpět odesílá odpověď. Protože komunikace probíhá po síti, je potřeba zajistit překlad volání

(tj. předání identifikátorů procedury a jejích parametrů) do formy vhodné pro přenos. Překlad volání zajišťuje spojka programu tzv. *stub*. Překlad volání se nazývá *marshalling* a zpětný překlad *unmarshalling*. Spojka je komunikační rozhraní, které implementuje protokol RPC. Jedna je umístěna na straně klienta a druhá na straně serveru. Spojka také zajišťuje „nahrazení" procedury, takže v programu je zápis volání vzdálené procedury velmi podobný zápisu volání lokální procedury. Spojka bývá vytvořena automaticky při generování programu.

Volání vzdálené procedury tedy probíhá tak, že klientská procedura zavolá proceduru, která není umístěna lokálně. Spojka zajistí překlad volání do formy vhodné pro přenos a odešle zprávu serveru. Server zprávu přijme a jeho spojka provede překlad. Pomocí identifikátoru procedury je zavolána konkrétní procedura, která se následně vykoná. Po vykonání procedury je výsledek předán zpět spojce, která ho opět přeloží a odešle klientovi. Spojka na straně klienta odpověď dekóduje a vrátí výsledek klientské proceduře. Ukázka komunikace pomocí RPC je zobrazena na obrázku 3.1.



Obr. 3.1 Komunikace RPC(převzato [5])

Výhodou vzdáleného volání procedur je zjednodušení klientské aplikace, kdy většinou složitější procedury jsou umístěny na serveru. Naopak nevýhodou je, že RPC podporuje pouze základní datové typy, přenos strukturovaných dat jako jsou např. fragmenty XML, není přímo podporován. Na platformě Android lze RPC implementovat za použití Android Interface Definition Language (AIDL) [6].

3.2. Distribuované objektové systémy

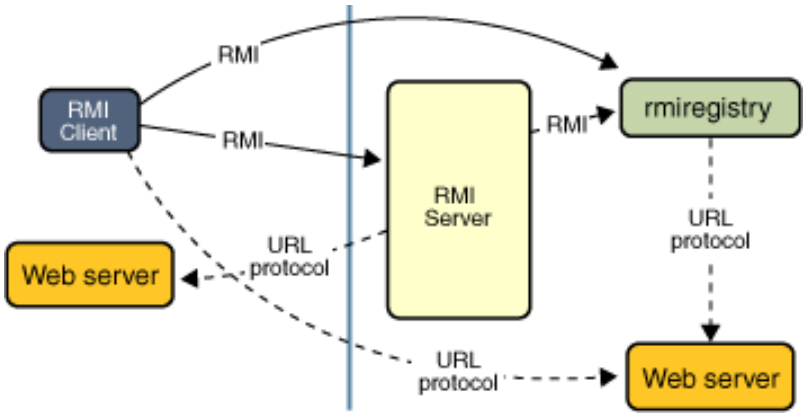
Distribuované objektové systémy umožňují volání vzdálených objektů, tak jako by objekty byly místní. K tomuto volání je použit mechanismus zástupných lokálních objektů (proxy). Mezi nejznámější distribuované systémy patří RMI, DCOM a COBRA.

3.2.1. Java RMI

Pro spouštění kódu na vzdáleném zařízení existuje v Javě technologie Remote Method Invocation (RMI) [7], která umožňuje objektu z jednoho Java Virtual Machine (JVM) volat metody objektů na jiném JVM.

RMI aplikace se obvykle skládá ze serveru a klienta. Server obvykle vytváří objekty a reference na tyto objekty, aby byly přístupné. Následně čeká na volání těchto objektů od klientské aplikace. Klient obsahuje reference na tyto vzdálené objekty, aby mohl vzdáleně vyvolávat jejich metody. Reference na vzdálené objekty se nachází v RMI registru. Server volá registr, aby asocioval jména se vzdálenými objekty. Klient vyhledá vzdálený objekt dle jména v registru serveru a vyvolá vykonání metody. Volání vzdálených objektů probíhá prostřednictvím lokálních zástupných objektů – *stub*. Výhoda tohoto řešení je, že programátor zachází se vzdáleným objektem, jako by byl místní. Ukázka komunikace RMI aplikace (viz obr. 3.2.1).

Rozhraní a třídy, které jsou zodpovědné za funkčnost RMI jsou nadefinovány v balíčku *java.rmi*. Tento způsob nemůže být v systému Android realizován, protože balíček *java.rmi* není v Android API podporován [8].



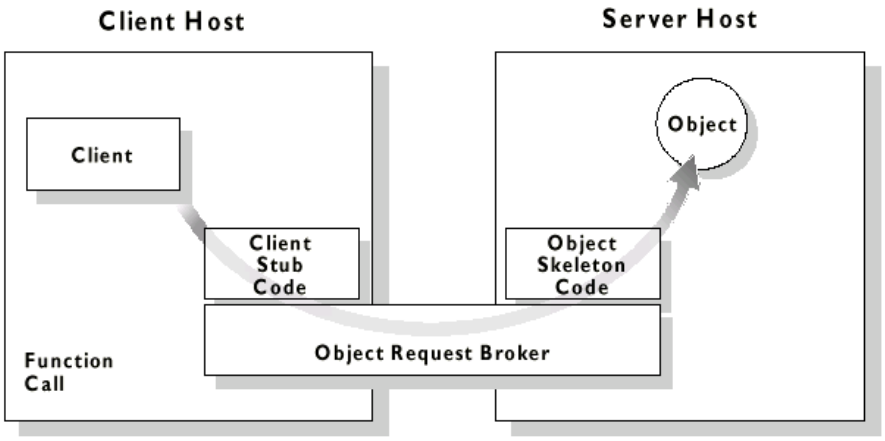
Obr. 3.2.1 Komunikace RMI aplikace (převzato z [7]).

3.2.2. CORBA

Common Object Request Broker Architecture (CORBA) [9] je standard definovaný skupinou Object Management Group (OMG) tvořící architekturu pro podporu tvorby distribuovaných objektově orientovaných aplikací. Softwarové komponenty napsané v jiném programovacím jazyce a běžící na různých počítačích mají možnost společně

komunikovat. Komunikace je založena na modelu klient/server, kde klient přistupuje ke službám objektů zasíláním požadavků. Komunikaci zajišťuje Object Request Broker (ORB), který obsahuje mechanismy pro vyhledání požadovaného objektu, generování a přenos požadavků a výsledků. Každý CORBA objekt má definované rozhraní v jazyce Interface Description Language (IDL), které specifikuje, jaké objekty jsou přístupné klientům.

Při vývoji aplikace je nejprve potřeba definovat rozhraní pomocí IDL. Rozhraní je následně zkompilováno a jsou vytvořeny Java třídy, klientské spojky (*stubs*) propojující klientský kód s ORB agentem a kostra objektu (*skeleton*) sloužící jako bázová třída odpovídající definici objektu v IDL. Při volání funkce objektu je volání přesunuto přes klientskou spojku do ORB, který pošle požadavek kostře volaného vzdáleného objektu. Ukázka volání funkce (viz obr. 3.2.2):



Obr. 3.2.2 Volání funkce (převzato z [10])

Nevýhodou tohoto řešení je, že port, přes který probíhá komunikace pomocí CORBA nebývá na firewallu obvykle povolen.

Pro využití technologie CORBA na platformě Android je možné použít například TIDorb [11], který je šířen pod GPL a Affero-GPL licencemi pro nekomerční použití.

3.2.3. DCOM

Distributed Component Object Model (DCOM) [12] je proprietární technologie společnosti Microsoft umožňující softwarovým komponentám na různých počítačích mezi sebou komunikovat po síti. Vznikla primárně pro Windows a na jiných platformách je implementována většinou v komerčních produktech. Pro Javu např. existuje knihovna *j-Interop* [13] implementující DCOM protokol. Tato knihovna je šíření pod LGPL licencí. Implementaci pro Android jsem během mého výzkumu neobjevil.

3.3. Zasílání zpráv

Komunikace mezi softwarovými komponentami či aplikacemi je zajištěna zasíláním zpráv. Využití tohoto řešení je zejména u webových služeb. K zakódování posílané zprávy se používá značkovací jazyk XML, jelikož poskytuje jednoduchou reprezentaci dat a také standard pro strukturu těchto dat. Další možností pro zakódování zprávy je JSON, což je formát určený pro výměnu dat založený na jazyce JavaScript. Přenos zpráv po sítí je zajištěn pomocí HTTP protokolu. Hlavní výhodou použití XML, JSON a HTTP je jejich široká podpora, tudíž aplikace napsané v různých programovacích jazycích mohou komunikovat mezi různými počítačovými architekturami. Výhoda využití HTTP spočívá v neomezené komunikaci na portu vyhrazeném pro HTTP protokol bez nutnosti konfigurace firewallů. Na rozdíl od technologií CORBA či DCOM, kde je potřeba povolit komunikaci na portech, které jejich přenosové protokoly využívají.

3.3.1. XML-RPC

XML-RPC [14] je protokol pro vzdálené volání procedur využívaný zejména pro komunikaci s webovými službami. Data jsou zapouzdřena pomocí XML a přenášena po sítí pomocí protokolu HTTP.

XML-RPC využívá ke komunikaci model klient/server. Klient odešle serveru XML-RPC zprávu, což je HTTP-POST dotaz a tělo zprávy je zapsáno v XML. Server na základě přijaté zprávy provede volanou proceduru a klientovi vrací zprávu s výsledkem anebo chybu. Každá zpráva se skládá z hlavičky a těla.

Příklad XML-RPC dotazu:

```
POST /xmlrpc HTTP/1.0
User-Agent: myXMLRPCClient/1.0
Host: home.zcu.cz
Content-Type: text/xml
Content-length: 144
<?xml version="1.0"?>
<methodCall>
   <methodName>getCompanyName</methodName>
   <params>
      <param>
         <value><int>25</int></value>
      </param>
   </params>
</methodCall>
```

Příklad XML-RPC odpovědi:

```
HTTP/1.0 200 OK
Connection: close
Content-Length: 144
Content-Type: text/xml
Date: Fri, 11 Apr 2012 19:55:02 GMT
Server: ZCU myXMLRPCClient/1.0
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>HTC</string></value>
    </param>
  </params>
</methodResponse>
```

Realizace tohoto řešení je možná i v systému Android, a to za použití knihovny android-xmlrpc [15], které je šířena pod licencí Apache License 2.0.

3.3.2. JSON-RPC

Další možný způsob k zakódování přenášené zprávy je použití JSON [16], což je také formát určený pro výměnu dat. Zápis v JSON je platným zápisem jazyka JavaScript. JSON je založen na dvou strukturách: objekt a pole. Na rozdíl od XML je menší, rychlejší a jednodušší k parsování [17].

JSON-RPC je protokol velmi podobný XML-RPC sloužící pro vzdálené volání procedur. Při volání procedury protokol odešle zprávu serveru implementující tento protokol. Komunikace probíhá odesíláním dotazu vzdálené proceduře za použití TCP/IP socketu anebo při některých omezeních pomocí HTTP. Klient obvykle volá jednu metodu a pokud volání obsahuje několik parametrů, jsou metodě předány jako pole nebo objekt a samotná vzdálená metoda také může vracet více výstupních dat. Vzdálená procedura je vyvolána přijmutím dotazu od klienta.

Dotaz odeslaný vzdálené proceduře je jeden serializovaný JSON objekt a musí obsahovat 3 základní vlastnosti:

- *method*: string obsahující název metody, která má být vykonána
- *params*: pole objektů předávaných jako parametry metodě
- *id*: id žádosti, může být různého typu a používá se ke spojení správných žádostí a odpovědí k sobě.

Po skončeném vyvolání metody je potřeba vrátit výsledek opět jako jeden serializovaný JSON objekt a tento objekt má opět 3 povinné vlastnosti:

- *result*: data vrácená od vyvolané metody. Pokud došlo k chybě, musí to být *null*
- *error*: je zde uveden kód chyby. Pokud k žádné chybě nedošlo je *null*
- *id*: identifikační číslo, které musí být shodné s id žádosti.

Notifikace je speciálním druhem dotazu, který nevyžaduje žádnou odpověď. Také se jedná o jeden serializovaný JSON objekt pouze s tím rozdílem, že oproti normálnímu dotazu musí být *id null*.

Ukázka komunikace pomocí JSON-RPC, kde → značí data odeslané službě a ← značí data přijatá od služby:

```
→{ "method": "echo", "params": ["Hello World"], "id": 1}

←{ "result": " Hello World ", "error": null, "id": 1}
```

Pro Android existuje knihovna android-json-rpc k usnadnění implementace JSON-RPC klienta [18] šířená pod licencí MIT License.

3.3.3. SOAP

Simple Object Access Protocol (SOAP) [19] je protokol pro přenášení zpráv vycházející ze svého předchůdce XML-RPC. Data jsou zapouzdřena v jazyce XML a přenos zpráv po síti probíhá přes protokol HTTP. Pro komunikaci pomocí protokolu SOAP existuje několik různých druhů šablon. Nejčastější použití je pro RPC. SOAP využívá model klient/server. Klient vysílá serveru zprávu s požadavky zapouzdřenou v jazyce XML, na serveru je vykonána volaná vzdálená procedura a zpráva s výsledkem je poslána zpět.

Struktura SOAP zprávy je následující. Jedná se o XML dokument, který obsahuje kořenový element `Envelope`. Uvnitř tohoto kořenového elementu jsou dva elementy `Header` (hlavička) a `Body` (tělo). Hlavička je nepovinný element používaný pro přenos pomocných informací pro zpracování zprávy, které můžou být např. identifikace uživatele. V těle se přenáší informace sloužící k identifikaci volané služby a parametry předávané vzdálené metodě.

Ukázka dotazu od klienta, který se ptá na informaci o telefonu:

```
POST /MySOAP HTTP/1.1
Host: www.mysoap.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 198
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getPhoneInfo xmlns="http://www.mysoap.com/data">
       <phoneID>143</phoneID>
     </getPhoneInfo>
   </soap:Body>
 </soap:Envelope>
```

Ukázka odpovědi od serveru:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 336
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getPhoneInfoRespons xmlns="http://www.mysoap.com/data">
       <getPhoneInfoResult>
         <companyName>HTC</companyName>
         <phoneName> Desire S</phoneName>
         <phoneID>143</phoneID>
         <platform>Android</platform>
       </getPhoneInfoResult>
     </getPhoneInfoRespons>
   </soap:Body>
</soap:Envelope>
```

Pro použití SOAP v systému Android existuje knihovna ksoap2-android [20] šířená pod licencí MIT License.

3.3.4. REST

Representational State Transfer (REST) [21] je architektura rozhraní, která je navržená pro distribuované prostředí. Na rozdíl od XML-RPC a SOAP je REST orientován procedurálně nikoliv datově. Webové služby definují vzdálené procedury a jejich volání, REST určuje, jak se přistupuje k datům. Rozhraní REST je použitelné pro jednotný přístup ke zdrojům. Zdroji mohou být data či stavy aplikace, pokud tyto stavy jdou popsat konkrétními daty. Všechny zdroje mají vlastní identifikátor URI a jsou definovány čtyři základní funkce pro přístup k nim.

Metody pro přístup ke zdrojům jsou označovány CRUD, pro vytvoření dat (*Create*), získání požadovaných dat (*Retrive*), změnu (*Update*) a smazání (*Delete*). Metody jsou implementovány pomocí odpovídajících metod HTTP protokolu.

Zdroje mohou být reprezentovány v různých formátech (XML, JSON, RSS a další). Pro platformu Android je možné vyvíjet aplikace, které přistupují pomocí rozhraní REST k datům webových služeb [22]. Android podporuje reprezentaci zdrojů v XML, JSON nebo binárně. Nejčastěji využívané jsou formáty XML a JSON.

3.4. Využití v systému Android

Z předchozích možností v systému Android nelze využít Java RMI, kvůli nepřítomnosti java.rmi balíčku v Android API. Řešení pro DCOM jsem nezjistil, ale je možné že existuje. Pro další metody existují implementace pro Android, takže je na uživateli, která metoda se mu bude jevit jako nejvýhodnější.

3.5. Webové aplikace

Za zmínku stojí webové aplikace, které běží na serveru a přistupuje se k nim pomocí webového prohlížeče. Výhodou webových aplikací je multiplatformnost, která umožňuje jejich spouštění na jakémkoliv zařízení s webovým prohlížečem. Snadno se u webových aplikací ladí chyby. Naopak nevýhodou je, že tyto aplikace nemají přístup ke všem hardwarovým možnostem zařízení. Hlavní technologie, využívané pro tvorbu webových aplikací, jsou HTML5, CSS a JavaScript.

4. Spouštění kódu na lokálním uzlu

V této části se zaměříme na to, jaké jsou možnosti pro spuštění vzdáleného kódu na lokálním uzlu v systému Android. Je tedy potřeba prozkoumat v jakém formátu bude muset být externí kód, aby bylo možné ho v systému Android nějakým způsobem zpracovat a na základě tohoto kódu vykonat určitou činnost.

4.1. Externí Java třída

Pro aplikaci, která má spouštět vzdálený kód, by se jako ideální řešení zdálo, pokud by existovala možnost stáhnout kód v Javě a ten následně zpracovat a spustit. Prozkoumáme tedy, jestli nějaké takové řešení existuje a zdali je aplikovatelné.

V Javě existuje komponenta `ClassLoader`, která je zodpovědná za nahrávání tříd do aplikace. V Javě a i v systému Android existuje možnost pro dynamické nahrávání tříd za běhu aplikace či načítání externích tříd do aplikace. Třídy mohou být umístěny buď mimo naší aplikaci na lokálním uzlu anebo je možné je získávat ze sítě. Pro načítání tříd z lokálního úložiště slouží v systému Android `DexClassLoader`, který umožňuje načítat dex (Dalvik executable) soubory z jiného než defaultního umístění a `PathClassLoader`, který operuje nad určitým seznamem souborů a složek na lokálním souborovém systému. Pro spuštění vzdáleného kódu slouží `UrlClassLoader`, který načítá třídy ze sítě. Tento *class loader* je zodpovědný za nahrávání tříd a zdrojů ze seznamu URL adres, kde tyto adresy odkazují na adresáře obsahující třídy anebo přímo na JAR soubory.

Pokud dochází k načítání tříd ze sítě, může docházet k bezpečnostním rizikům, protože třídy mohou obsahovat potenciálně nebezpečný kód. Řešením pro ošetření bezpečnostního rizika je vytvořit *class loader*, který bude před spuštěním vzdáleného kódu kontrolovat digitální podpis. Vzdálenému kódu také mohou být nastavena oprávnění, jaké akce může při spuštění provádět, např. zákaz otevírání souborů.

Toto řešení se pro naši práci jeví jako velmi zajímavé. V následujícím příkladu (převzato z [23]) je ukázka implementace tohoto řešení v systému Android. Ukázka se týká načtení třídy, která se nenachází v defaultním dex souboru. Aplikace obsahuje aktivitu, která vyvolá komponentu z knihovny a zobrazí text na displeji. Aktivita se

nachází v defaultním dex souboru a kód knihovny je získán z druhého dex souboru, který je také obsažen v APK.

Aplikace se skládá za tří tříd. Hlavní třída je `MainActivity` a vyvolává komponentu z knihovny. Druhou třídou je `LibraryInterface`, což je definice rozhraní pro knihovnu a poslední třídou je `LibraryProvider`, což je třída umístěna mimo defaultní dex soubor a obsahující komponentu pro zobrazení textu na displej.

Nejprve je potřeba získat externí třídu. V následující ukázce části kódu z `MainActivity` je třída `LibraryProvider` zkopírována do umístění, kde cesta k tomuto umístění může být poskytnuta pro *class loader*. Jako úložiště je zde použita interní privátní paměť aplikace.

```
File dexInternalStoragePath = new File(getDir("dex",
Context.MODE_PRIVATE), SECONDARY_DEX_NAME);
  ...
  BufferedInputStream bis = null;
  OutputStream dexWriter = null;

  static final int BUF_SIZE = 8 * 1024;
  try {
      bis = new
BufferedInputStream(getAssets().open(SECONDARY_DEX_NAME));
      dexWriter = new BufferedOutputStream(
          new FileOutputStream(dexInternalStoragePath));
      byte[] buf = new byte[BUF_SIZE];
      int len;
      while((len = bis.read(buf, 0, BUF_SIZE)) > 0) {
          dexWriter.write(buf, 0, len);
      }
      dexWriter.close();
      bis.close();
  } catch (. . .) {...}
```

Následně `DexClassLoader` musí načíst knihovnu získanou z externího umístění. K vyvolání metod existuje několik možností. V tomto ukázkovém příkladě je instance třídy zavolána přes rozhraní, které metodu volá přímo.

```
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.example.dex.lib.LibraryProvider");

      LibraryInterface lib = (LibraryInterface)
libProviderClazz.newInstance();
      lib.showAwesomeToast(this, "hello");
  } catch (Exception e) { ... }
```

4.2. XML

Uživatelské prostředí Android aplikací je možné vytvářet deklarováním elementů a jejich vlastností pomocí jazyka XML, programově přímo v kódu aplikace nebo použitím obou řešení dohromady. Protože lze uživatelské prostředí vytvořit programově, prozkoumáme možnosti, zdali jde získat vzdálený kód a na jeho základě vygenerovat uživatelské prostředí pro aplikaci. Pro zapouzdření vzdáleného kódu přichází v úvahu XML jazyk.

Značkovací jazyk XML se jeví jako jedna z možností pro zapouzdření. Výhodou je jednoduchá čitelnost pro člověka a standardizovaný způsob formátu dat, což umožňuje jednodušší zpracování. Pro parsování kódu existují rozhraní SAX a DOM, která je možno implementovat i v systému Android.

4.2.1. Rozhraní DOM

Document Object Model DOM [24] je standardem konsorcia W3C, který definuje rozhraní pro přístup k dokumentu a jeho zpracování. Zpracování dokumentu pomocí DOM probíhá tak, že celý dokument je přečten a následně uložen do paměti, kde je reprezentován jako stromová hierarchická struktura, kde každému uzlu ve stromu odpovídá jeden element. Strom je možné procházet a jednotlivé elementy editovat, přidávat či mazat díky funkcím, které poskytuje rozhraní DOM.

4.2.2. Rozhraní SAX

Simple API for XML (SAX) [25] není standardem, ale byl vytvořen skupinou lidí kolem konference XML-DEV. Rozhraní SAX je založeno na řízení pomocí událostí. Uživatel nadefinuje funkce, které se mají vykonat pokud parser narazí na počáteční element, obsah elementu, koncový element, komentář apod. Oproti DOM parseru se liší tedy v tom, že nenačítá celý dokument do paměti jako strom, ale průběžně ho prochází a volá události. Výhodou tohoto řešení je jeho rychlost a malá paměťová náročnost. Naopak nevýhodou je nemožnost dokument upravovat nebo se vrátit k již zpracovaným datům.

4.3. JSON

Další možností jak zapouzdřit přenášená data, je pomocí JSON. Tento formát pro výměnu dat je také jednoduše čitelný. Jak již bylo zmíněno dříve, JSON je založen na dvou strukturách, a to objekt a pole.

Android přímo obsahuje podporu `JSONObjects` a `JSONArrays`, což umožňuje snadné parsování dokumentu ve formátu JSON.

4.4. Bezpečnost

Při spouštění vzdáleného kódu na lokálním uzlu je potřeba brát v úvahu možná bezpečnostní rizika. Samotná nainstalovaná aplikace obsahuje Android Manifest, což je soubor zapsaný v XML, který obsahuje seznam oprávnění, která má daná aplikace povolena. Zmíníme některá oprávnění: možnost telefonních hovorů, možnost odeslat SMS, zápis na interní úložiště, instalování balíčku a další.

Ukázka oprávnění v manifestu:

```
android.permission.WRITE_EXTERNAL_STORAGE"
android.permission.READ_EXTERNAL_STORAGE"
android.permission.INTERNET"
android.permission.ACCESS_NETWORK_STATE"
android.permission.READ_SMS"
android.permission.INSTALL_PACKAGES"
```

U stahování kódu po síti ze serveru je možné riziko pozměnění přenášeného kódu útočníkem. Toto riziko může nastat u akcí, které pracují se soubory, SMS zprávami, emaily či telefonními čísly. Jako příklad uvedeme aplikaci, která bude odesílat SMS zprávy na čísla nadefinované podle staženého externího kódu. Útočník by mohl tento kód odchytit, upravit a odeslat ho na určený cíl. Uživatel by poté bez svého vědomí mohl odesílat zprávy na prémiová čísla za vyšší cenu. Zabránit tomuto riziku by se dalo použitím digitální podpisu nebo šifrováním. Ukázka napadení na obrázku 4.4.

Obrázek 4.4 Ukázka napadení kódu útočníkem

Dalším bezpečnostním rizikem může být, pokud útočník získá aplikační balíček, přidá do něj škodlivý kód a následně aplikaci dále distribuuje. Tomuto napadení se zabrání podepisováním aplikace bezpečnostním klíčem. Toto riziko není výsadou pouze naší aplikace, ale takto napadena může být každá aplikace.

5. Návrh řešení

Při návrhu řešení se budeme zabývat spouštěním kódu na lokálním uzlu. Tento kód budeme získávat z externího úložiště na síti.

5.1. Cíle návrhu

Cílem je navrhnout řešení, jak na základě informací v souboru získaného ze serveru dynamicky vytvořit uživatelské prostředí aplikace. Prvkům v uživatelském prostředí chceme poté přiřadit akce nadefinované v aplikaci.

5.2. Uživatelské rozhraní

U aplikací v systému Android se uživatelské prostředí definuje v XML souboru nebo programově přímo ve zdrojovém kódu. Pro dynamické vytvoření se definuje uživatelské prostředí programově přímo ve zdrojovém kódu. V následujících příkladech je layout definován v XML souboru a následně nadefinován programově.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

Stejný layout, definován programově přímo v kódu:

```
linearLayout = new LinearLayout(this);
linearLayout.setOrientation(LinearLayout.VERTICAL);
// od API verze 8 se pouziva match_parent misto fill_parent
linearLayout.setLayoutParams(new
LayoutParams(LayoutParams.MATCH_PARENT,LayoutParams.MATCH_PARENT));
textView = new TextView(this);
textView.setText("Hello, World!");
textView.setLayoutParams(new LayoutParams(
        LayoutParams.MATCH_PARENT,
        LayoutParams.WRAP_CONTENT));
linearLayout.addView(textView);
setContentView(linearLayout);
```

Jak je vidět, tak vlastnosti nastavené v XML souboru lze nadefinovat i přímo v kódu aplikace. Atributy nastavitelné v XML souboru mají svůj ekvivalent pro jejich nastavení programově. Je potřeba zvolit, jaké vlastnosti pro nastavení uživatelského prostředí budeme získávat z externího souboru a vybrat formát, v jakém bude tento soubor zapouzdřen.

Na základě námi prozkoumaných možností máme na výběr mezi zapouzdřením kódu v jazyce XML nebo JSON. V příkladech je zobrazen zápis dat pomocí XML a poté JSON.

```
<?xml version="1.0"?>
<object>
    <type>button</type>
    <id>1</id>
    <name>erase</name>
    <text>Erase text</text>
</object>
```

```
{"object": {
  "type": "button",
  "id": "1",
  "name": "erase",
  "text": "Erase text",
}}
```

Při rozhodování mezi těmito dvěma formáty uvážíme jejich výhody a nevýhody. Výhodou XML je snazší čitelnost kódu pro člověka, ale naopak oproti JSON zabírá více místa díky svému formátování. JSON má horší čitelnost kódu, ale tím, že díky svému formátování zabírá méně místa, se více hodí pro přenos po síti. Zpracování v systému Android je pro XML zajištěno pomocí rozhraní SAX a DOM. JSON se pro zpracování kódu hodí více, díky podpoře JSON objektů a polí přímo v Android API. V našem případě bude k zapouzdření kódu použit formát XML, protože chceme využít možností snazšího zpracování kódu pomocí parserů. Větší velikost souboru nehraje podstatnou roli, protože zdrojový kód nebude příliš rozsáhlý.

Když máme formát dat zvolen, zaměříme se na to, jaká data pro uživatelské prostředí budeme získávat z externího souboru. Vytvoření kompletního uživatelského prostředí na základě stažené souboru se zdá být náročné a těžko realizovatelné. Pro naši práci bude postačovat, když ze staženého souboru získáme objekty, které přidáme do připraveného layoutu.

Layout bude nastaven přímo v aplikaci a dle externího souboru budeme definovat objekty, které bude obsahovat. XML soubor se bude skládat ze seznamu objektů a jejich parametrů. Parametry, které tyto objekty mohou mít, jsou např. šířka, výška, identifikační číslo nebo text. Parametry objektů se budou lišit podle toho, k čemu je aplikace určena. Pokud například budeme tvořit aplikaci pro odesílání SMS zpráv na některá konkrétní čísla, tak jako parametr objektu může být telefonní číslo, na které má být zpráva odeslána. Navrhneme parametry, které by měl obsahovat každý objekt bez ohledu na to, o jakou se jedná aplikaci (viz tab. 5.2). Prvky uživatelského prostředí jsou různých typů a mohou mít rozdílné vlastnosti. Abychom tedy rozlišili, jaké prvky získáváme ze souboru, bude jeden z parametrů určovat typ objektu (např. tlačítko, textové pole atd.). Podle toho rozpoznáme odlišné objekty a budeme nastavovat

vlastnosti skupinám objektů. Pro jednoznačnou identifikaci objektů použijeme parametr *id*, který bude sloužit jako unikátní identifikátor. Jelikož budeme chtít prvkům v uživatelském prostředí přiřazovat akce, tak pro určení přidružené akce nastavíme objektu parametr *actionId*. Pro jednodušší identifikaci objektů, než pouze podle identifikačního čísla bude mít každý objekt definované jméno - *name* a zástupný text objektu - *text*. Ostatní parametry se budou lišit podle konkrétního určení aplikace.

Parametry objektů	
type	typ objektu
id	číslo objektu
actionId	číslo přidružené akce
name	jméno objektu
text	zástupný text objektu

Tabulka 5.2 Parametry objektů nadefinované v XML souboru

XML soubor máme nadefinován a musíme ho naší aplikací zpracovat. Pro zpracování kódu v jazyce XML máme na výběr mezi rozhraními SAX a DOM. Při použití rozhraní DOM je XML soubor zpracován a umístěn jako strom do paměti. Poté je možné stromem procházet a prvky editovat, mazat nebo přidávat. V našem případě XML soubor stačí projít pouze jednou pro načtení objektů. Pro naše řešení bude ideální SAX parser, který je rychlejší a zabírá méně paměti než zpracování pomocí DOM parseru.

5.3. Uživatelské akce

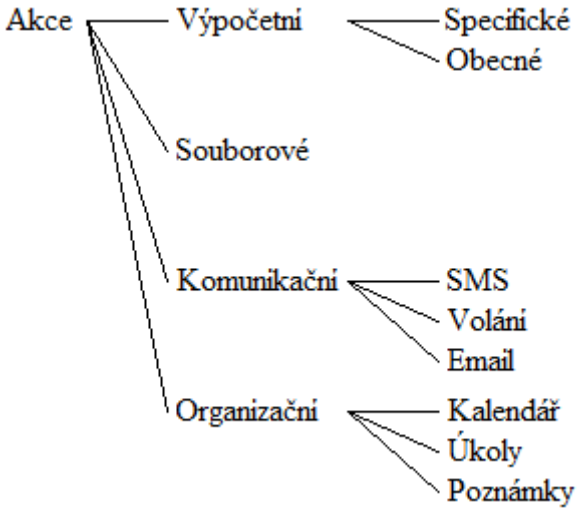
Po vygenerování uživatelského prostředí je potřeba navrhnout řešení, jak přiřadit uživatelské akce ovládacím prvkům aplikace. Chceme, aby aplikace dokázala dynamicky reagovat na změnu zdrojové souboru, který obsahuje nadefinované prvky a jejich parametry. V předchozí části jsme navrhli, že každý prvek bude obsahovat svůj unikátní identifikátor *id* a identifikátor *actionId*, který slouží k označení přidružené akce. K tomuto účelu je nutné, aby aplikace obsahovala předem připravené akce a stavový automat, který podle *actionId* prvku bude vybírat přiřazené akce.

Ukázka navrženého konceptu zapsána v pseudokódu:

```
Function onClick(‘buttonId‘)
   choose ‘actionId‘ where ‘buttonId‘ == ‘id‘
   CASE actionId OF
    1 : runAction1()
    2 : runAction2()
   ENDCASE

Function runAction1()
   open web browser
Function runAction2()
   open calendar
```

Akce, které jsou ve stavovém automatu zahrnuty, se liší podle způsobu použití aplikace (viz obr 5.2). Mohou být například aplikace obsahující výpočetní funkce, kde změnou prvků dojde k výběru jiných funkcí pro výpočet. V následující části navrhneme některé akce, které by mohl stavový automat obsahovat. U všech řešení je potřeba, aby aplikace měla v manifestu nastavené oprávnění pro přístup k internetu, kvůli stahování XML souboru. Další potřebná oprávnění jsou popsána u jednotlivých akcí.



Obrázek 5.2 Kategorie navržených akcí

5.3.1. Návrh akcí

Výpočetní specifické

Stavový automat bude obsahovat specifické výpočetní funkce lišící se určením dané aplikace. Každá aplikace bude mít nadefinované svoje specifické funkce. Podle definice objektů v XML souboru se bude vybírat, které funkce se budou používat. Pro tyto akce není potřeba žádné další oprávnění. Z hlediska bezpečnosti nepředstavují tyto akce žádné riziko. Rizikem můžu být zacyklení a případné vyčerpání paměti či rychlé vybíjení baterie.

Výpočetní obecné

Může se jednat o obecné výpočetní funkce, jako jsou sčítání, násobení, dělení a další. Tyto akce poté mohou využívat aplikace pracující s obecnými výpočty. Žádné další oprávnění není potřeba. U těchto akcí také nehrozí bezpečnostní riziko.

Práce se soubory

Akce, které mohou vytvářet soubory s určitým obsahem, editovat, kopírovat nebo mazat. Soubory se mohou přijímat ze serveru nebo je na server odesílat. V XML souboru například můžeme u odesílacích tlačítek jako parametr předávat adresu

cílového serveru pro odesílání souboru. Při realizaci takovéto aplikace je v manifestu potřeba nastavit oprávnění pro zápis do paměti a čtení z paměti. Rizikem u práce se soubory může být čtení důvěrných dat nebo mazání těchto dat.

Práce s emaily

Definice akcí pro tvorbu emailů, příjem emailů a další. Příkladem může být aplikace pro tvorbu emailů, kde email příjemce bude pevně nadefinován jako parametr odesílacího tlačítka a bude získáván z XML souboru. Riziko u těchto akcí představuje využití emailů za účely spamu.

Práce s SMS

Může se například jednat o aplikaci, ve které bude nadefinováno několik tlačítek pro odesílání SMS zpráv určitým lidem. Z externího XML souboru bude pro tyto tlačítka jako parametry získávat jména osob a jejich telefonní čísla. U realizace takového řešení je potřeba ošetřit situaci, že by ze souboru bylo získáno telefonní číslo, na které je zvýšená cena za odeslání zprávy. Aplikace musí mít nastavené oprávnění pro odesílání SMS zpráv.

Práce s kalendářem

Akce sloužící pro práci s kalendářem, jako je přidávání událostí, výpis událostí z kalendáře. K vytvoření aplikace pracující s kalendářem je nutné mít oprávnění pro zápis a čtení kalendáře. Rizikem u těchto akcí může být přístup k osobním informacím z kalendáře.

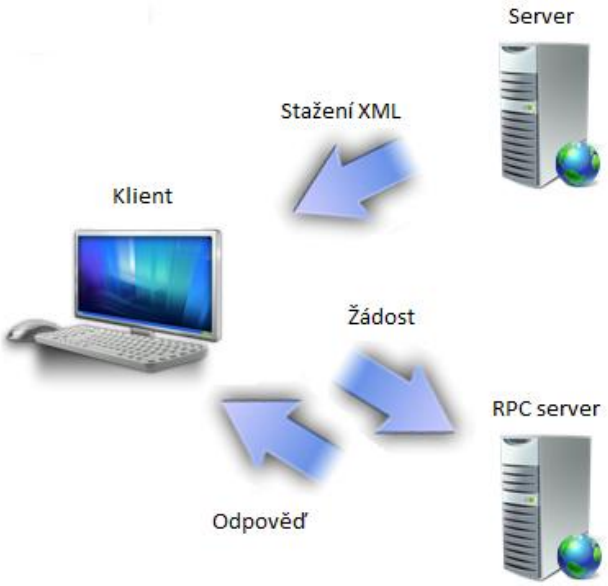
5.4. Omezení návrhu

Zabývali jsme se pouze tím, že do aplikace z externího souboru získáváme prvky uživatelského prostředí. Nezískáváme kompletní definici uživatelského prostředí, takže layout je nastaven přímo v aplikaci a uzpůsoben konkrétnímu XML souboru.

Námi navržené řešení pro vygenerování uživatelského prostředí má využití u aplikací, které nejsou graficky složité. Vytvořit hru tímto mechanismem nepůjde, ale hodí se to spíše pro jednoduché aplikace, obsahující například formuláře.

Pokud vytvoříme pro aplikaci nový předpis v XML souboru a aplikace neobsahuje definice pro akce vyvolávané dle tohoto předpisu, je nutné vytvořit novou verzi aplikace. Když bychom chtěli změnit funkčnost nějaké akce, také je nutné vytvořit novou verzi. V nové verzi aplikace tyto funkce dodefinujeme a můžeme aplikaci obohacenou o novou funkčnost distribuovat. Před realizací je dobré si promyslet užití aplikace a její budoucí rozšiřitelnost.

Řešením toho, abychom nemuseli při změně funkčnosti akcí vytvářet novou verzi aplikace, by mohla být kombinace uvedených možností pro spouštění vzdáleného kódu na lokálním a externím uzlu. Použili bychom nějakou metodu pro RPC. Aplikace by si ze serveru stáhla XML soubor s definicí objektů a vygenerovala by uživatelské prostředí. Volané akce by poté předali zadané parametry vzdálené metodě umístěné na serveru a ten by jako odpověď vrátil výsledek (viz obr. 5.3).



Obrázek 5.3. Aplikace s využitím RPC serveru

Výhodou tohoto řešení je univerzálnost aplikace, protože pro změnu akce stačí změnit její definici na RPC serveru. Není potřeba při každé změně vytvářet a instalovat novou verzi aplikace. Přidáním volání metod na RPC serveru přibyla další síťová komunikace. Rizikem tohoto řešení je, že pokud dojde k výpadku sítě, stane se aplikace nepoužitelnou. Tyto případy je potřeba v aplikaci ošetřit.

6. Realizace aplikace

Zde je popsána realizace aplikace ověřující funkčnost námi navrženého systému pro dynamické spouštění vzdálených kódů na platformě Android.

6.1. Zadání aplikace

Úkolem bylo vytvořit aplikaci, která na základě stažených dat ze serveru dynamicky vytvoří uživatelské prostředí pro kalkulačku nebo pro převod teplot. Ovládacím prvkům v aplikaci poté bylo potřeba přiřadit příslušné akce.

6.2. Návrh aplikace

Dle navrženého řešení, aplikace ze serveru stahuje XML soubor obsahující objekty pro vytvoření uživatelského prostředí. XML soubor je v aplikaci zpracován za použití SAX parseru a následně dochází k vygenerování uživatelského prostředí a přiřazení akcí prvkům aplikace.

Nejprve navrhneme XML soubor, který slouží k definici uživatelského prostředí. Soubor obsahuje seznam objektů a jejich parametrů. Struktura souboru je pro obě aplikace totožná. Liší se pouze objekty, které obsahuje. Parametry nastavené objektům jsou: *type, id, actionId, name* a *text*. Tyto parametry byly popsány v návrhu řešení uživatelského prostředí.

Ukázka struktury XML souboru:

```
<application>
  <object>
    <type>button</type>
    <id>0</id>
    <actionId>0</actionId>
    <name>eraseAll</name>
    <text>Vymazat</text>
  </object>
  ...
</application>
```

Obě aplikace budou mít stejnou kostru a budou se lišit pouze v definici XML souboru (viz přílohy A1 a A2). Aplikace bude obsahovat akce pro kalkulačku i převodník teplot. Pokud budeme chtít vytvořit další aplikaci podle nového předpisu, bude potřeba přidat nové definice akcí a metody pro jejich vykonání. Vzhled aplikace je uzpůsoben našim dvěma XML předpisům. Pro jiný předpis lišící se uspořádáním nebo počtem tlačítek, bude potřeba optimalizovat layout aplikace. Možným rozšířením by bylo použít pozicování prvků v uživatelském prostředí.

Výběr akcí bude realizován pomocí stavového automatu, který bude akce vybírat podle *actionId*. Při stisku tlačítka podle *id* zjistíme, které tlačítko bylo stisknuto a jaké obsahuje *actionId*. Poté je zavolána akce s tímto *actionId*.

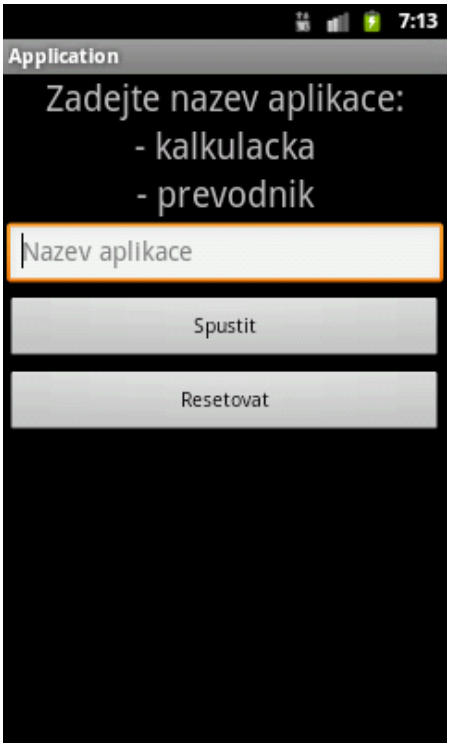
6.3. Implementace

Samotná aplikace obsahuje 4 třídy.

ApplicationBP.java

Obsahuje hlavní aktivitu, která se spustí po startu aplikace. Po spuštění se zobrazí uživatelské prostředí (viz obr. 6.3a), kde se zadává název aplikace, nebo přímá URL adresa ke zdrojovému XML souboru. Předání názvu aktivitě zpracovávající XML soubor je zajištěno pomocí *sharedPreferences*. *SharedPreferences* je společné úložiště, kam je možné ukládat data primitivních datových typů a poté k nim přistupovat. Preference se ukládají jako dvojice klíč-hodnota, kde klíč slouží k přístupu k preferenci. Tyto data zůstávají trvale uchována v úložišti i po skončení aplikace. K zapsání preference slouží metoda *writeString()* a pro získání preference *readString()*. Metoda *writeString()* pro zápis využívá editor pro *sharedPreferences*, který získá díky metodě *getEditor()*. Metoda *readString()* získává preference pomocí metody *getPreferences()*.

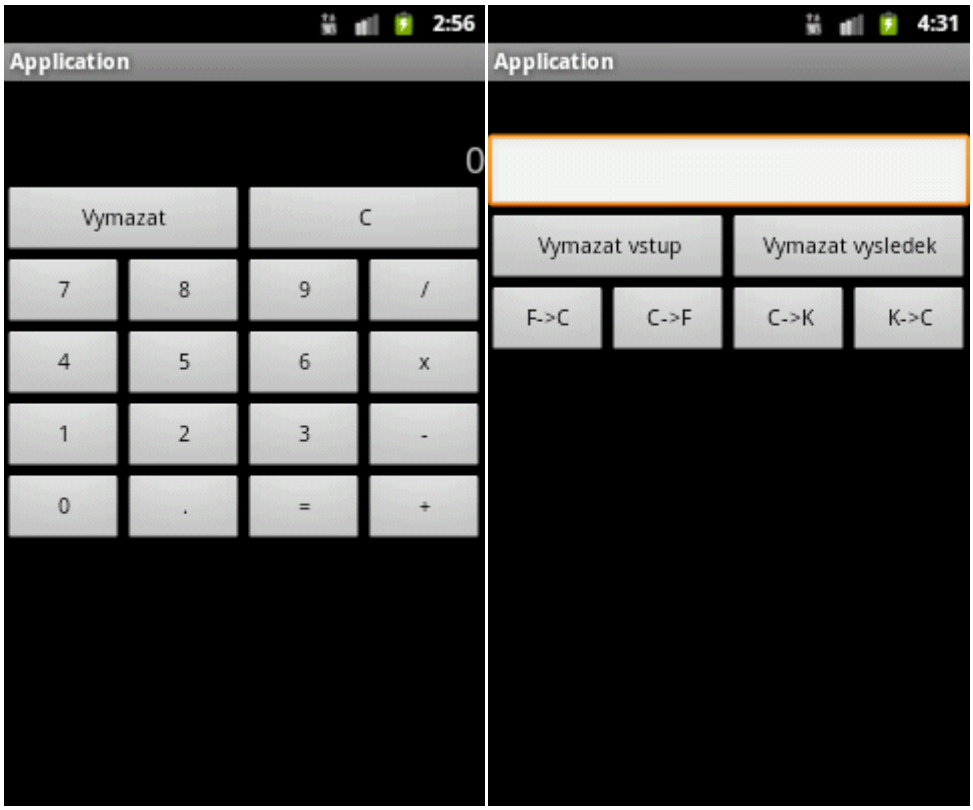
Po zadání názvu aplikace a stisku tlačítka Spustit se spustí metoda *run ()*, která uloží zadaný název jako preferenci a poté dojde ke spuštění aktivity generující uživatelské prostředí aplikace. Pokud není název aplikace vyplněn, vytvoří se defaultně uživatelské prostředí pro kalkulačku. Tlačítko Reset spouští metodu *reset ()*, která slouží k vymazání uložené preference. Při spuštění aplikace se vždy spustí metoda *readUrl()*, která nastaví preferenci jako název aplikace, pokud preference není prázdná.



Obrázek 6.3a Úvodní obrazovka aplikace

MainClass.java

Po spuštění hlavní aktivitou se provede metoda *checkInternetConnection()*, sloužící k otestování internetového připojení. Pokud není připojení dostupné je vypsána chybová hláška, v opačném případě se spustí metoda *run()*. V této metodě je získán název aplikace ze *sharedPreferences* a je použit do zdrojové URL adresy. Z té je stažen XML soubor a zpracován pomocí SAX parseru. Objekty a jejich parametry jsou uloženy do pole *objectsList*. V cyklu se toto pole prochází a podle typu objektu se vytvoří *TextView, Button* nebo *EditText*. Těmto prvkům se nastaví příslušné parametry a uloží do pole *textViews, buttons* nebo *editViews*. Všechny získané prvky jsou přidány do layoutu a ten je zobrazen na obrazovce (viz obr 6.3b).



Obr. 6.3b Ukázka uživatelského prostředí obou aplikací: vlevo kalkulačka, vpravo převodník teplot

Pro vykonání akcí jednotlivých tlačítek je každému tlačítku nastaven posluchač `OnclickListener`, který při stisku tlačítka zavolá metodu `handleKeypadInput()` a předá jí *id* stisknutého tlačítka a *actionId*. Tato metoda obsahuje stavový automat, který vybírá podle *actionId* akci, která se má vykonat. Tyto akce poté vyvolávají svoje metody, které jsou shrnuty v tabulce 6.3. Některé z těchto metod mají ještě svoje další pomocné metody. Metoda kalkulačky *calculate()* volá svojí pomocnou metodu *evaluateResult()*, ve které probíhá samotný výpočet výsledku. Čísla pro výpočet a operátory jsou vkládány do zásobníků, proto pro práci se zásobníky jsou nadefinované metody *clearStacks()* pro vymazání zásobníků a *dumpInputStack()* pro vypsání

zásobníku. Metodou, kterou využívají obě aplikace, je metoda *doubleToString()* sloužící k převodu výsledného čísla ve formátu *double* na řetězec.

Číslo akce	Název akce	Činnost akce
Kalkulačka		
0	eraseAll()	vymaže vstup
1	eraseOne()	vymaže jeden znak na vstupu
15	addPoint()	přidání desetinné čárky
5, 9, 13, 17	mathOperation()	reaguje na stisk tlačítek +, -, *, /
16	calculate()	provede výpočet
2, 3, 4, 6, 7, 8, 10, 11, 12	writeNumbers()	slouží pro zadávání čísel
Převodník teplot		
0	eraseAll()	vymaže vstup
2	eraseResult()	vymaže výsledek
20	fahrToCelsius()	provede převod z Fahrenheita na Ceslia
21	celsiusToFahr()	provede převod z Celsia na Fahrenheita
22	celsiusToKelvin()	provede převod z Celsia na Kelvina
23	kelvinToCelsius()	provede převod z Kelvina na Celsia

Tabulka 6.3 Seznam akcí ve stavovém automatu

Ukázka stavového automatu:

```
switch (objectsList.getActionID().get(id)){
     case 0: {
          eraseAll();
       };
       break;
        case 1: {
          eraseOne(currentInput, currentInputLen);
        };
        break;
        case 15: {
       addPoint(currentInput);
        };
       break;
  ...
```

MyXMLHandler.java

Třída obsahuje metody pro parsování XML souboru. XML soubor je postupně procházen, a pokud se narazí na otevírací tag `<name>` zavolá se metoda `startElement()` a nastaví se hodnota `currentElement` na *true*. Jedná-li se o hlavní tag `<application>`, tak se vytvoří nové pole objektů, do kterého jsou následně objekty přidávány. Pro získání textu mezi tagy se volá metoda `characters()`. Když je `currentElement` nastaven na *true*, uloží se text do `currentValue` a `currentElement` je nastaven na *false*. Pokud se narazí na zavírací tag `currentElement` se nastaví na *false* a podle tagu se přiřadí parametr objektu.

ObjectsList.java

Tato třída definuje pole *objectsList* a parametry objektů *id, actionId, name, text, type*. Obsahuje funkce pro získávání a nastavování těchto parametrů.

Layout main.xml

Zde je definováno grafické prostředí pro úvodní obrazovku. Úvodní obrazovka nezískává definici pro uživatelské prostředí ze vzdáleného kódu.

AndroidManifest.xml

Obsahuje důležité informace o aplikaci. Definuje se zde oprávnění pro přístup k internetu a přístup ke stavu síťového připojení. Dále definuje, která aktivita se spouští jako hlavní při startu aplikace.

6.4. Bezpečnost aplikace

V našem případě získáváme pro naši aplikaci pouze předpis pro prvky uživatelského prostředí, které neprovádí žádné nebezpečné akce. Aplikace má v manifestu nastavena oprávnění jen pro využívání internetového připojení a pro zjišťování stavu internetového připojení. K napadení aplikace může dojít změnou aplikačního balíčku, když by byla změněna funkčnost metod a přidána práva do manifestu.

6.5. Rozšíření aplikace

Aplikace má layout připraven přímo pro náš XML soubor. Layout lze použít i pro další jednoduché aplikace s podobným obsahem objektů. Při větší změně zdrojového souboru, například přidání více tlačítek, by bylo potřeba layout předefinovat. Pokud bychom nadefinovali nový XML soubor a chtěli podle něho realizovat aplikaci, bylo by nutné přidat do stavového automatu další akce. K těmto akcím by poté bylo nutné nadefinovat metody.

6.6. Testování

Testování probíhalo na emulátoru ve verzi 2.3.3, 4.0.3 a také na reálném telefonu s verzí 2.3.3. Aplikace plní svojí činnost správně a během testování nebyly zjištěny žádné chyby.

6.7. Správa a ovládání aplikace

Instalace aplikace na reálné zařízení

- Soubor ApplicationBP.apk umístíme na paměťovou kartu nebo připojíme telefon k počítači a nainstalujeme z počítače pomocí příkazu `adb install <path_to_apk>`

- Pomocí souborového manažera v telefonu aplikaci nainstalujeme.
- Dojde k nainstalování aplikace a v seznamu aplikací se objeví ApplicationBP.
- Spustíme aplikaci

Spuštění aplikace v emulátoru

- Musíme mít nainstalované vývojové prostředí IDE Eclipse.
- Musíme mít nainstalované Android SDK a v Eclipse stažen plugin, který s SDK komunikuje.
- Importujeme aplikaci jako existující projekt a přeložíme jako Android aplikaci. Poté dojde ke spuštění emulátoru, nainstalování a spuštění aplikace.

IDE Eclipse je ke stažení na adrese `http://www.eclipse.org/downloads/`. Android SDK je možné stáhnout z `http://developer.android.com/sdk/index.html` Poté je potřeba nainstalovat a zprovoznit plugin pro Eclipse dle návodu na `http://developer.android.com/sdk/eclipse-adt.html`.

Ovládání aplikace

- Po spuštění aplikace se objeví úvodní obrazovka. Na úvodní obrazovce se nastavuje název zdrojového XML souboru, nebo URL adresa zdrojového souboru.
- Po zadání názvu či adresy a spuštění aktivity dojde vygenerování uživatelského prostředí a aplikace je připravena k používání. Pokud chceme změnit zdrojový soubor, stačí se tlačítkem zpět vrátit na úvodní obrazovku.
- Pro spuštění aplikace je potřeba připojení k internetu, kvůli nutnosti stažení XML souboru.

7. Závěr

V této bakalářské práci jsem se zabýval dynamickou tvorbou aplikací na platformě Android.

Platforma Android je v současnosti nejrozšířenějším mobilním systémem na smartphonech a díky rozšiřování i na další zařízení počet uživatelů stále stoupá. Pro větší flexibilitu aplikací se hledají způsoby jak pozměnit funkcionalitu a uživatelské prostředí, aniž by muselo dojít k instalaci nové verze aplikace.

Výsledkem mé práce je úspěšné prozkoumání možností spouštění vzdáleného kódu a tvorby dynamických aplikací v systému Android. S využitím těchto znalostí jsem navrhl řešení pro vytvoření aplikace, která získává data pro definici uživatelského prostředí po síti ze serveru. Následně jsem tento systém realizoval a otestoval jeho funkčnost na dvou ukázkových aplikacích.

Využití navrženého systému může být například ve firemních aplikacích. Pro malou úpravu funkčnosti není nutné instalovat na všechna zařízení nové verze aplikace, ale stačí pouze změnit zdrojový XML soubor umístěný na serveru.

Přehled zkratek a použitého značení

ADT	Android Development Tools
AIDL	Android Interface Definition Language
API	Application Programming Interface
APK	Android Application Package
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DCOM	Distributed Component Object Model
DEX	Dalvik Executable
DOM	Document Object Model
GPL	General Public License
HTML5	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IDL	Interface Definition Language
JAR	Java Archive
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LGPL	Lesser General Public License
MIT	Massachusetts Institute of Technology
NDK	Native Development Tools
OMG	Object Management Group
ORB	Object Request Broker
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSS	RDF Site Summary, Really Simple Syndication
SAX	Simple API for XML
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Literatura

1. *Android Developers.* [Online] [Citace: 11. 4 2012.] http://developer.android.com/index.html.

2. Android SDK. *Android Developers.* [Online] [Citace: 15. 4 2012.] http://developer.android.com/sdk/index.html.

3. ADT Plugin for Eclipse. *Android Developers.* [Online] [Citace: 15. 4 2012.] http://developer.android.com/sdk/eclipse-adt.html.

4. What is the NDK. *Android Developers.* [Online] [Citace: 11. 4 2012.] http://developer.android.com/sdk/ndk/overview.html.

5. **Bloomer, John.** *Power Programming with RPC.* Sebastopol : O'Reilly & Associates, Inc., 1992. ISBN 0-937175-77-3.

6. **Kratzer, Kevin.** Using the Android Interface Definition Language (AIDL) to make a remote Procedure Call (RPC) IN ANDROID. *App-Solut.* [Online] [Citace: 30. 4 2012.] http://app-solut.com/blog/2011/04/using-the-android-interface-definition-language-aidl-to-make-a-remote-procedure-call-rpc-in-android/.

7. An Overview of RMI Applications. *The Java Tutorials.* [Online] [Citace: 11. 4 2012.] http://docs.oracle.com/javase/tutorial/rmi/overview.html.

8. **Burnette, Ed.** Java vs. Android APIs. *ZDNet.* [Online] [Citace: 4. 4 2012.] http://www.zdnet.com/blog/burnette/java-vs-android-apis/504.

9. CORBA Basics. *CORBA.* [Online] [Citace: 22. 4 2012.] http://www.omg.org/gettingstarted/corbafaq.htm.

10. CORBA. *České Java Centrum.* [Online] [Citace: 22. 4 2012.] http://java.cecrdle.net/vyzkum/kapitola_3.htm.

11. **García, Álvaro.** TIDorb ported to Android. *Morfeo Corba Platform.* [Online] [Citace: 30. 4 2012.] http://corba.morfeo-project.org/archives/tidorb-ported-to-android.

12. **Thai, Thuan.** *Learning DCOM.* Sebastopol : O'Reilly Media, 1999. ISBN 978-1-56592-581-6.

13. *j-Interop.* [Online] [Citace: 23. 4 2012.] http://j-interop.org/.

14. **Winer, Dave.** Specification. *XML-RPC.* [Online] [Citace: 15. 4 2012.] http://xmlrpc.scripting.com/spec.

15. Xml remote procedure calls on android. *Hello Android.* [Online] [Citace: 15. 4 2012.] http://www.helloandroid.com/tutorials/xml-remote-procedure-calls-android.

16. *Úvod do JSON.* [Online] [Citace: 15. 4 2012.] http://www.json.org/json-cz.html.

17. JSON Tutorial. *W3Schools.* [Online] [Citace: 15. 4 2012.] http://www.w3schools.com/json/default.asp.

18. android-json-rpc. *Google code.* [Online] [Citace: 15. 4 2012.] http://code.google.com/p/android-json-rpc/.

19. **Skonnard, Aaron a Gudgin, Martin.** SOAP. *XML - pohotová referenční příručka.* Praha : Grada Publishing a.s., 2006, Kapitola 10, stránky 315-336.

20. ksoap2-android. *Google ode.* [Online] [Citace: 23. 4 2012.] http://code.google.com/p/ksoap2-android/.

21. **Massé, Mark.** *REST API Design Rulebook.* Sebastopol : O'Reilly Media, 2011. ISBN 978-1-4493-1050-9.

22. **Dobjanschi, Virgil.** Developing Android REST client applications. *Google I/O 2010.* [Online] [Citace: 30. 4 2012.] http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html.

23. **Chung, Fred.** Custom Class Loading in Dalvik. *Android Developers.* [Online] [Citace: 1. 5 2012.] http://android-developers.blogspot.com/2011/07/custom-class-loading-in-dalvik.html.

24. XML DOM Tutorial. *w3schools.* [Online] [Citace: 23. 4 2012.] http://www.w3schools.com/dom/.

25. *SAX project.* [Online] [Citace: 23. 4 2012.] http://www.saxproject.org/.

A Přílohy

A1 Definiční XML soubor pro ukázkovou aplikaci Kalkulačka

```
<application>
   <object>
      <type>textview</type>
      <id>0</id>
      <actionId>18</actionId>
      <name>result</name>
      <text></text>
   </object>
   <object>
      <type>textview</type>
      <id>1</id>
      <actionId>19</actionId>
      <name>userInput</name>
      <text>0</text>
   </object>
   <object>
      <type>button</type>
      <id>2</id>
      <actionId>0</actionId>
      <name>eraseAll</name>
      <text>Vymazat</text>
   </object>
   <object>
      <type>button</type>
      <id>3</id>
      <actionId>1</actionId>
      <name>eraseOne</name>
      <text>C</text>
   </object>
   <object>
      <type>button</type>
      <id>4</id>
      <actionId>2</actionId>
      <name>button7</name>
      <text>7</text>
   </object>
   <object>
      <type>button</type>
      <id>5</id>
      <actionId>3</actionId>
      <name>button8</name>
      <text>8</text>
   </object>
   <object>
      <type>button</type>
      <id>6</id>
      <actionId>4</actionId>
      <name>button9</name>
      <text>9</text>
   </object>
   <object>
      <type>button</type>
      <id>7</id>
      <actionId>5</actionId>
      <name>buttonDivide</name>
```

```
      <text>/</text>
   </object>
   <object>
      <type>button</type>
      <id>8</id>
      <actionId>6</actionId>
      <name>button4</name>
      <text>4</text>
   </object>
   <object>
      <type>button</type>
      <id>9</id>
      <actionId>7</actionId>
      <name>button5</name>
      <text>5</text>
   </object>
   <object>
      <type>button</type>
      <id>10</id>
      <actionId>8</actionId>
      <name>button6</name>
      <text>6</text>
   </object>
  <object>
      <type>button</type>
      <id>11</id>
      <actionId>9</actionId>
      <name>buttonMultiple</name>
      <text>x</text>
   </object>
   <object>
      <type>button</type>
      <id>12</id>
      <actionId>10</actionId>
      <name>button1</name>
      <text>1</text>
   </object>
   <object>
      <type>button</type>
      <id>13</id>
      <actionId>11</actionId>
      <name>button2</name>
      <text>2</text>
   </object>
   <object>
      <type>button</type>
      <id>14</id>
      <actionId>12</actionId>
      <name>button3</name>
      <text>3</text>
   </object>
   <object>
      <type>button</type>
      <id>15</id>
      <actionId>13</actionId>
      <name>buttonSubtract</name>
      <text>-</text>
   </object>
   <object>
      <type>button</type>
```

```
      <id>16</id>
      <actionId>14</actionId>
      <name>button0</name>
      <text>0</text>
   </object>
   <object>
      <type>button</type>
      <id>17</id>
      <actionId>15</actionId>
      <name>buttonPoint</name>
      <text>.</text>
   </object>
   <object>
      <type>button</type>
      <id>18</id>
      <actionId>16</actionId>
      <name>buttonEquals</name>
      <text>=</text>
   </object>
   <object>
      <type>button</type>
      <id>19</id>
      <actionId>17</actionId>
      <name>buttonAdd</name>
      <text>+</text>
   </object>
</application>
```

A2 Definiční XML soubor pro ukázkovou aplikaci Převodník teplot

```
<application>
   <object>
      <type>textview</type>
      <id>0</id>
      <actionId>18</actionId>
      <name>result</name>
      <text></text>
   </object>
   <object>
      <type>edittext</type>
      <id>1</id>
      <actionId>19</actionId>
      <name>userInput</name>
      <text>0</text>
   </object>
   <object>
      <type>button</type>
      <id>2</id>
      <actionId>0</actionId>
      <name>eraseInput</name>
      <text>Vymazat vstup</text>
   </object>
   <object>
      <type>button</type>
      <id>3</id>
      <actionId>2</actionId>
      <name>eraseResult</name>
      <text>Vymazat vysledek</text>
   </object>
   <object>
      <type>button</type>
      <id>4</id>
      <actionId>20</actionId>
      <name>buttonFtoC</name>
      <text>F->C</text>
   </object>
  <object>
      <type>button</type>
      <id>5</id>
      <actionId>21</actionId>
      <name>buttonCtoF</name>
      <text>C->F</text>
   </object>
   <object>
      <type>button</type>
      <id>6</id>
      <actionId>22</actionId>
      <name>buttonCtoK</name>
      <text>C->K</text>
   </object>
   <object>
      <type>button</type>
      <id>7</id>
      <actionId>23</actionId>
      <name>buttonKtoC</name>
      <text>K->C</text>
   </object>
</application>
```