VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A IMPLEMENTACE KNIHOVNY PRO ČÍSLICOVÉ ZPRACOVÁNÍ SIGNÁLŮ NA PLATFORMĚ ANDROID

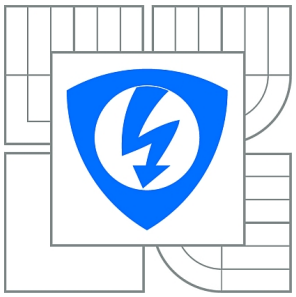
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MARTIN JANOVSKÝ
AUTHOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A IMPLEMENTACE KNIHOVNY PRO ČÍSLICOVÉ ZPRACOVÁNÍ SIGNÁLŮ NA PLATFORMĚ ANDROID

DESIGN AND IMPLEMENTATION OF SIGNAL PROCESSING LIBRARY USING ANDROID

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

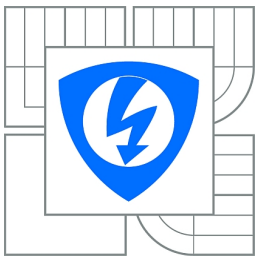
AUTOR PRÁCE — Bc. MARTIN JANOVSKÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL TRZOS

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Martin Janovský	***ID:***	78265
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Návrh a implementace knihovny pro číslicové zpracování signálů na platformě Android

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte možnosti vývojového balíčku pro mobilní zařízení založené na platformě Android k číslicovému zpracování signálů. Realizujte základní algoritmy číslicového zpracování signálů jako je konvoluce a filtrace signálu filtrem typu FIR a IIR. Tyto algoritmy realizujte v jazyce Java jako aplikaci s grafickým uživatelským rozhraním. Při realizaci používejte nástroje pro verzování zdrojového kódu a nástroje pro automatické generování dokumentace ze zdrojového kódu.

DOPORUČENÁ LITERATURA:

[1] Android Developers [online]. Google, 2010. Dokument dostupný na http://developer.android.com/index.html

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Michal Trzos

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

Tato diplomová práce řeší problém návrhu a implementace knihovny pro zpracování signálů pod platformou Android. Rozebírá problematiku vlastního vývoje aplikací pod tento systém. Částečně diskutuje, zda je pod touto platformou vhodné aplikace vyvíjet. Popisuje přitom problematiku uplatnění mobilních aplikací a jejich náročnost na vývoj. Vysvětluje technologie a postupy, kterých bylo při tvorbě aplikace využito. Přitom i osvětluje programovací postupy ze strany efektivity a v některých případech nabízí náhradní řešení a cesty jeho využití. Efektivitu přitom zkoumá z pohledu výpočetního výkonu i paměťové náročnosti. Pokračuje přes rozbor architektury platformy Android a některé funkce vlastního systému až k popisu jeho některých částí. Dostává se až k popisu práce s daty nad zvukovými formáty či k práci se soubory a různými kolekcemi. Další popis pokračuje teorií použitých funkcí a samotnou aplikací. Vlastní knihovna je řízena pomocí uživatelského rozhraní a dalšími pomocnými funkcemi. Intuitivní uživatelské rozhraní umožňuje uživateli jednoduchou volbu všech parametrů. Uživatel si navolí jednotlivé parametry signálu, který se bude zpracovávat. Signálem může být vedle generovaných signálů i nahraný zvuk. Po zvolení parametrů jednotlivých filtrů si může uživatel nechat přehrát výstupní signál a pozorovat jeho změnu. V další části jsou porovnávány různé přístroje a diskutována některá vylepšení, která zrychlují a zkvalitňují celou aplikaci. V závěru jsou shrnuty výsledky a přínosy celé práce.

KLÍČOVÁ SLOVA

Android, zpracování signálu, knihovna, aplikace, chytrý telefon

ABSTRACT

The diploma thesis solves the problem of design and implementation of library for digital signal processing on Android platform. It describes problems with development software on this platform. There is also a debate about portion of Android on market. It describes technologies and ways, that were used for developing an application. It also explains the efficient programming techniques, in some cases, offering alternative solutions and ways. Effectiveness examines from the perspective of processing power and memory consumption. Continues through analysis Android platform architecture, and some features of its system to the description of some of its parts. Gets up to the description of work with the audio data formats, or to work with files and various collections of data. Then description continues with functions and the application itself. The actual library is controlled through the user interface and some other functions. User select parameters of each signal to be processed. The signal can be generated or the recorded sound signal. After selecting the parameters of each filter, the user can play the output signal and observe the change. In other part different instruments are compared and discussed some improvements, which accelerate and improve the entire application. The conclusion summarizes the results and benefits of the entire diploma thesis.

KEYWORDS

Android, signal processing, library, aplication, smartphone

JANOVSKÝ, Martin *Návrh a implementace knihovny pro číslicové zpracování signálů na platformě Android*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2010. 54 s. Vedoucí práce byl Ing. Michal Trzos

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Návrh a implementace knihovny pro číslicové zpracování signálů na platformě Android“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Michalu Trzosovi za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Cílem práce bylo prostudovat možnosti dostupných knihoven a balíčků pro číslicové zpracování signálů v prostředí platformy Android. Z tohoto bodu měl vyjít jakýsi návrh samotné aplikace. Dalším bodem byla samotná implementace algoritmů knihovny pro zpracování signálů. Postupem času se objevily i další možnosti rozšíření aplikace. Zejména pak možnost nahrávání zvukového záznamu a jeho další zpracování. Hlavním požadavkem tedy byla knihovna s uživatelským prostředím, které bude schopno využít jednotlivých funkcí knihovny. Důležitým požadavkem byla také jednoduchá použitelnost a rozšiřitelnost knihovny pro další využití. Knihovna by tedy měla být využitelná samostatně pro výpočty jiných aplikací.

V celé práci měl být kladen důraz na jednotlivé možnosti využití různých programovacích technik. Nešlo o to, dosáhnout co nejpřesnějších výsledků vlastních filtrovacích algoritmů, ale především o to, jak správně převést náročné výpočetní operace na méně náročné a jak tyto správně zpracovávat v prostředí mobilní platformy. Čtenář by tedy, po dočtení této práce, měl být obeznámen o výpočetních možnostech a hlavních výhodách či nevýhodách systému Android.

V textu je řešen problém pro mobilní zařízení s platformou Android 2.2 a 2.1. V jiných verzích se mohou jednotlivé funkce či použité třídy lišit. Celá aplikace byla vyvíjena a testována na těchto verzích softwaru a zařízeních:

- prostředí – Eclipse SDK Verze: 3.6.1
- emulátor a manažer – Android SDK Manager Revize 7
- platforma – Android 2.1–update1
- nástroj pro verzování kódu – Eclipse Subversive(SVN Team Provider)
- generátor dokumentace – Javadoc plugin do Eclipse
- uložiště verzovaného kódu – Assembla(http://www.assembla.com/)
- zařízení – LG–P500 Optimus One, Android 2.2(hlavní zařízení)
- zařízení – Samsung Galaxy Spica, Android 2.2(použito pro porovnání výpočetního výkonu a testování funkčnosti aplikace)

1 ANDROID

V první kapitole je probírána platforma Android. Od všeobecných informací a motivace až k popisu podrobnějšímu. V dalším textu se ze znalostí, které zde čtenář získá vychází, a tak tato kapitola tvoří jakousi prerekvizitu dalšího textu. Celá platforma je popisována pouze s ohledem na její zapojení do celkové koncepce diplomové práce. Je tedy popsána pouze problematika, která s prací souvisí. Text vychází z mnoha informačních zdrojů uvedených v části Literatura. Citace konkrétního pramene je vždy vyznačena přímo v textu.

1.1 Motivace

Čím dál více lidí přistupuje pravidelně k internetovým službám pomocí „netradičních“ technologií, jako jsou mobilní telefony, čtečky, aj. Po mobilních telefonech je již dávno, vedle odesílání SMS a volání, požadováno připojení k internetu. S rozvojem technologií a stoupajícím počtem nejrůznějších služeb, je kladen vyšší důraz na výpočetní kapacitu koncového zařízení. Zároveň jsou zde požadavky na miniaturizaci a energetickou nenáročnost zařízení.

Toto vše, v roce 2005, přimělo firmu Google Inc. k akvizici firmy Android Inc. S ní převzal i projekt Android jako platformu pro chytré telefony, navigace a PDA. Oficiální vznik platformy Android byl ohlášen na konci roku 2007, kdy Google přenechává tuto platformu sdružení Open Handset Alliance, ve které je sám členem.

Aby platforma Android nezůstala jednou z mnoha, dochází k masivní podpoře a motivaci vývojářů aplikací. Nejvýznamnějšími kroky v podpoře vývoje jsou hlavně vytvoření balíčku Android SDK[1] pro vývoj v prostředí Eclipse[2], pořádání celé řady konferencí, obsáhlá dokumentace a v neposlední řadě i soutěž Android Developer Challenge[3].

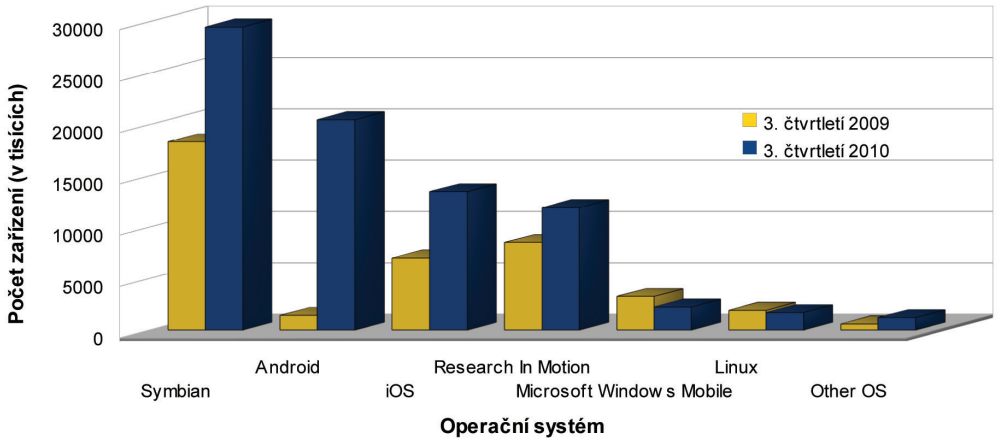
Po několika vývojových verzích je v dubnu roku 2009 uvedena na trh platforma Android 1.5(Cupcake). Díky politice podpory vývojářů aplikací jsou v září a posléze v říjnu téhož roku představeny verze 1.6(Donut)a 2.1(Eclair). V květnu 2010 Android pokračuje s verzí 2.2(Froyo) a v říjnu představuje zatím poslední verzi 2.3(Gingerbread). Za rok tak Android dodává na trh platformu srovnatelnou s konkurenčními produkty. Nárůst jeho podílu na trhu je také patrný na grafu v obrázku 1.1. Je vidět,

[1] Android Software Development Kit – balíček do prostředí Eclipse pro vývoj aplikací pod platformu Android

[2] Eclipse – vývojové prostředí pro celou řadu programovacích jazyků

[3] Android Developer Challenge – je soutěž vývojářů pořádaná společností Google a dotovaná až 10 milióny dolarů(2008)

že za jediný rok získává 20 procent trhu. Tak se Android stává velice významnou platformou těchto zařízení.



Obr. 1.1: Podíl jednotlivých OS na celosvětovém prodeji chytrých telefonů.[7]

V průběhu psaní této práce je také zveřejněna další studie [8], která ukazuje data za celý rok 2010 a odhaduje další vývoj. Jak je vidět z grafu na obrázku 1.2, Android už během tohoto roku(2011) překoná všechny své konkurenty. Navíc v roce 2015 odhad říká, že Android bude ve více než polovině všech prodaných zařízení. Za zmínku také stojí předpokládané druhé místo společnosti Microsoft na trhu v roce 2015 s jejich platformou Windows Mobile.

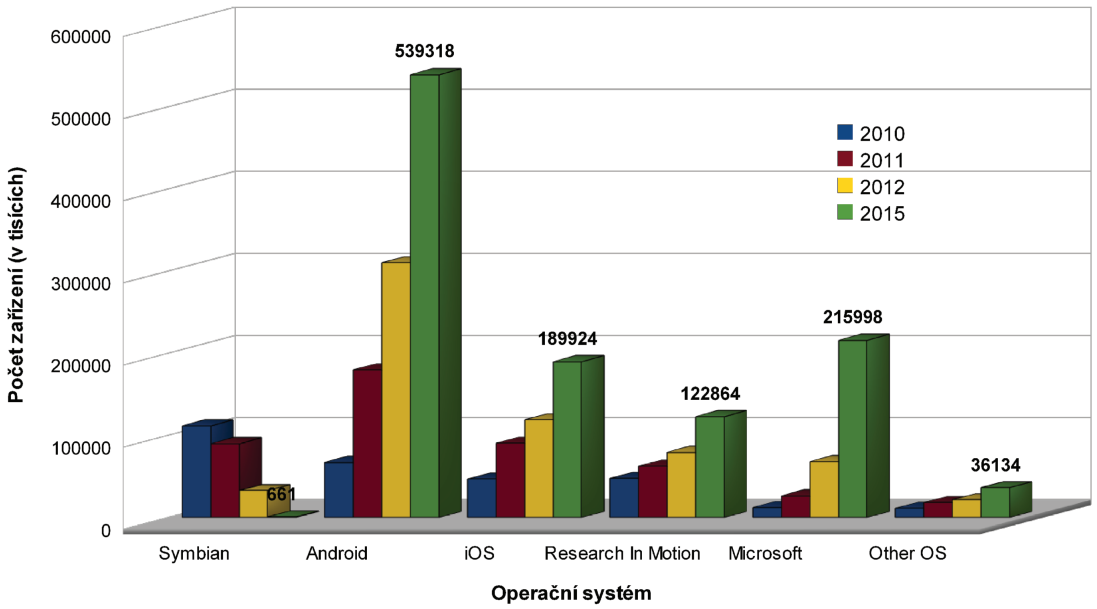
1.2 Licence

Celá platforma je spojena především s licencí Apache License 2.0[4]. Ta byla použita, protože celá platforma je složená z více softwarových produktů, které spadají pod různé licence. Tato open source[5] licence je dále výhodná pro vývoj aplikací. Jednotlivé aplikace či části kódů se mohou využívat a měnit bez větších problémů. Nejedná se však o absolutně volnou licenci, kde si můžeme dělat, co chceme. Musíme, stejně jako u GPL[6] licence, zachovávat určitou formou referenci, na původního autora a licenci. Hlavní důvodem, proč autoři dali přednost licenci Apache 2.0 před GPL je, že GPL vyžaduje při použití částí s touto licencí, aby výsledek byl šířen pod stejnou licencí. Přesné vymezení pravidel je možno vyčíst ze zdroje [3].

[4] Apache License, Version 2.0 – zdrojové kódy softwaru pod touto licencí jsou volně šiřitelné, upravitelné a se zachováním určitých pravidel dále šiřitelné i pod jinou licencí

[5] Open Source Software – software s veřejnými zdrojovými kódy

[6] General Public License – zdrojové kódy softwaru pod touto licencí jsou volně šiřitelné, upravitelné, ale dále distribuovatelné jen pod touto licencí



Obr. 1.2: Podíl jednotlivých OS na celosvětovém prodeji chytrých telefonů s výhledem do roku 2015.[8]

1.3 Architektura systému

Oficiálně je názvem Android nazýván balíček softwaru, který obsahuje operační systém, klíčové aplikace a software zajišťující komunikaci mezi nimi. Toto je nejvíce patrné a srozumitelné na schématu vnitřní architektury na obrázku 1.3.

Za základní blok bylo zvoleno linuxové jádro. Další vrstvy modelu využívají především jeho stability, ohledu na energetickou náročnost a pokročilého přístupu k víceprocesovému zpracování. Je využíváno verze jádra 2.6 a jeho různých podverzí. Jádro však muselo být ořezáno o některé funkce, které by v mobilních zařízeních hledaly těžko uplatnění. Dalším důvodem pro zmenšení jádra byla i výpočetní kapacita těchto zařízení. Základním úkolem jádra je přístup k samotnému hardware a správa paměti. Dále musí efektivně zpracovávat jednotlivé procesy. K tomu musí dohlížet na to, aby tyto funkce byly vykonávány s ohledem na výdej elektrické energie.

Nad jádrem je připravena sada knihoven napsaných v jazycích C/C++. Tyto knihovny obsahují téměř veškeré funkce, které přímo využívají, nebo jsou jinak spojené s jádrem.

Každá aplikace v prostředí Android běží ve svém vlastním procesu, uvnitř vlastní instance Dalvik VM[7]. Tento virtuální stroj byl navržen tak, aby efektivně mohlo být

[7] Dalvik Virtual Machine – obdoba Java VM v prostředí Android, přizpůsobuje kód vyšších vrstev spodním vrstvám

Obr. 1.3: Popis architektury platformy Android.

spuštěno více jeho instancí. Celý jeho koncept vychází z Java VM. Při psaní aplikací pro Android se využívá také jazyk Java. Kód napsaný v jazyce Java se ale musí ještě upravit pomocí tzv. dx[8] nástroje do výstupního formátu (`*.dex`). Takto upravený kód je pak možné spustit v Dalvik VM. Ten navazuje na funkcionality jádra v oblasti správy vláken a správy paměti.

Další je vrstva, která obsahuje předpřipravené šablony. Jde o balíček funkcí, které jsou využívány pro vlastní tvorbu aplikací v poslední vrstvě modelu. Tyto šablony tedy tvoří jakési vývojářské rozhraní. Nechá se totiž zjednodušeně říci, že vývojáře moc nemusí zajímat, co je pod touto vrstvou. O tom, že to do jisté míry není pravda, se přesvědčíme například u zmínky o nativním kódu. Pro nepříliš náročné aplikace však opravdu nemusí vývojář o spodních vrstvách vědět mnoho.

[8] dx – nástroj upravující soubory jazyka Java na soubory s příponou (*.dex) spustitelné pomocí Dalvik VM

1.4 Aplikace a jejich komponenty

Jak už jsem zmínil, aplikace jsou zde psány v programovacím jazyku Java. Přeložený kód je spolu s dalšími daty zabalen do balíku s koncovkou `*.apk`. Toto je pak nazýváno aplikací, kterou si může uživatel uložit do svého zařízení. O zkompletování finální aplikace se stará nástroj `aapt`[9]. Je začleněn v celé řadě nástrojů, které jsou pro vývojáře volně k dispozici pod jednotným názvem Android SDK.

Při spuštění vzniká, do jisté úrovně abstrakce, oddělený svět. Je to dáno oddělením paměti a vznikem další instance Dalvik VM. Na vrstvě jádra vzniká samostatný linuxový proces s unikátním identifikátorem uživatele. Tím se zaručí, že jednotlivé aplikace nemají přístup k paměti a souborům jiných aplikací. Každý uživatel(aplikace) má totiž implicitně přístup pouze ke svému vlastnictví. Pro sdílení dat mezi aplikacemi jsou zde připraveny speciální metody. Mohou však existovat i aplikace se stejným uživatelským identifikátorem. Jedná se o jednu z více možností sdílení paměťových zdrojů. Takto Android využívá silných stránek v oblasti bezpečnosti, a přesto neztrácí na funkčnosti.

Existuje zde celá řada úrovní pro spouštění aplikací. Od procesů běžících na pozadí až po zobrazovací metody, na které jsou kladeny nejvyšší nároky. Tyto procesy jsou pak spouštěny s ohledem na jejich prioritu. Celý systém se stará o maximální využití systémových zdrojů. Když tedy například dochází paměť, jsou nejprve vypínány procesy s nízkou prioritou. Je však možné vypínat i jednotlivé části aplikací. Aplikace v Androidu se totiž skládají ze čtyř základních komponent. Tyto se navzájem kombinují a vzniká tak výsledná aplikace. Systém dokáže odlišit jejich prioritu a efektivně pak hospodařit s výpočetním výkonem a pamětí. Prioritu jednotlivých komponent však neřídí pouze systém. Může být změněna i při samotném vývoji aplikace.

Základní komponenty pro vývoj aplikací:

- Aktivity(Activities)
- Služby(Services)
- Přijímače oběžníků(Broadcast receivers)
- Poskytovatelé obsahu(Content providers)

Aktivity

Každá aktivita představuje grafické uživatelské rozhraní. Například když uživatel posílá zprávu. Komponenta, která se stará o vykreslení této informace na displej, je

[9] aapt – Android Asset Packaging Tool – nástroj pro tvorbu aplikace s koncovkou (*.apt) spustitelné na platformě Android

právě aktivita. Další aktivita se může starat o zobrazení kontaktů, nastavení nebo jiné služby aplikace. Dohromady tedy pracují jako uživatelské rozhraní. Jsou na sobě nezávislé a každá je podtřídou základní třídy `Activity`.

Aplikace tedy může obsahovat pouze jednu nebo jako v předešlém příkladu hned několik aktivit. Kolik jich je, a jakým způsobem na sobě budou záviset, je otázkou konkrétní aplikace. Jedna aktivita, která je určena jako první, je spuštěna po startu vlastní aplikace. Přechod z jedná aktivity na druhou je uskutečněn tak, že aktuální aktivita spustí následující.

Každá aktivita má implicitní okno, do kterého může vykreslovat. Nejčastěji okno vyplňuje celou obrazovku. Velikost a pozice okna se však dají měnit. Na jedné obrazovce může být také více oken, která se mohou různě překrývat. Například jako dialog informující uživatele o vybité baterii, který se objeví pouze na omezenou dobu, ale zakryje všechno ostatní pod ním.

Zobrazený obsah okna je dostupný díky hierarchickým pohledům. Jde o objekty základní funkce `View`. Každý kontroluje svůj vlastní prostor uvnitř okna. Základní objekt `View` pak obsahuje sjednocení těchto prostorů. Každý prostor také obsahuje funkce pro interakci s uživatelem. Například uvnitř pole pro vkládání textu po kliknutí začne blikat kurzor. Android má celou řadu těchto předpřipravených rozšíření třídy `View` zahrnující tlačítka, textová pole, posuvníky, položky menu, atd. Celý obsah okna je pak začleněn do aktivity pomocí funkce

```
Activity.setContentView().
```

Takto nastavíme, co se v okně bude zobrazovat. Jak budou jednotlivé prostory uspořádány v okně, je určeno v XML[10] šabloně. Je možné nadefinovat celou řadu těchto šablon a snadno tak měnit i zobrazení jednotlivých aktivit.

Služby

Naproti tomu jsou služby, které žádné grafické rozhraní nemají. Pouze běží na pozadí jakkoli dlouho a uživatel si jich ani nemusí všimnout. Každá služba vznikne rozšířením základní třídy `Service`. Aktivity i služby běží ve stejném vláknu(vrstva jádra) i stejném procesu(vrstva procesů).

Jako příklad mějme přehrávač hudby se seznamem skladeb. Bude zde nejspíše několik aktivit, které budou obsluhovat uživatelská nastavení jako vybírání skladeb k poslechu či nastavení hlasitosti. Hraní hudby samotné však bude zajištěno službou, která poběží na pozadí. To zajistí, že jakmile uživatel opustí aplikaci, hudba bude i

[10] eXtensible Markup Language – rozšiřitelný značkovací jazyk, uživatel si může vytvářet vlastní tagy

nadále hrát. Systém totiž nechá běžet službu na pozadí, i když aktivita, která službu spustila, už není na obrazovce. V případě, že uživatel opět otevře aplikaci, se pouze naváže aktivita k již běžící službě. Službu pouze musíme upravit, aby disponovala rozhraním pro ovládání funkcí jako start, stop, pauza, ad. Tyto funkce pak pomocí aktivity řídí běh celé služby.

Přijímače oběžníků

Tyto přijímače nedělají nic jiného, než že naslouchají oznámením(oběžníkům). Tato oznámení jsou generovány systémem nebo samotnými aplikacemi. Jedná se například o oznámení změny souřadnic GPS nebo jen indikace nízkého stavu baterie. Samotná aplikace pak může například generovat oběžník o tom, že data jsou stažena a připravena dalším aplikacím.

Aplikace může mít neomezeně přijímačů oznámení, které jsou pro danou aplikaci relevantní. Všechny přijímače vzniknou rozšířením základní třídy přijímačů `BroadcastReceiver`.

Stejně jako služby nemají přijímače žádné uživatelské rozhraní. Jejich výstupem ale může být spuštění aktivity. Na informaci o příchozím oznámení mohou také uživatele upozornit prostřednictvím třídy `NotificationManager`. Vibrace, blikání osvětlení a hraní hudby jsou jen některé z jejích schopností. Často užívaným je zobrazení ikony v hlavní liště. Po aktivaci ikony se zobrazí výpis zprávy, kterou přijímač vygeneroval.

Poskytovatelé obsahu

Poskytují určitá data dalším aplikacím. Tato data mohou být uložena jednoduše v souborovém systému nebo třeba v SQLite[11] databázi. Pro aplikaci je pak výhodné pouze požádat poskytovatele o data. Stará se tak pouze o komunikaci s ním. Nemusí ji zajímat, odkud data jsou nebo kdo k nim má přístup. Jednotliví poskytovatelé mohou také komunikovat mezi sebou. Každý poskytovatel vznikne rozšířením základní třídy `ContentProvider`.

1.5 Přístup k datům

Kvůli názornosti ukázek funkcí knihovny budeme využívat zvukové signály. Přehrávání zvukových dat je na platformě Android umožněno pomocí třídy `AudioTrack`. Tato třída není sice vhodná pro přehrávání se seznamem skladeb, ale to nám nemusí

[11] SQLite – databázový dotazovací jazyk vycházející z SQL, nepotřebuje k běhu server

vůbec vadit. Tato třída umožňuje odesílání toku dat ve formátu PCM[12] přímo na zvukový hardware zařízení. Jde o tok jednotlivých vzorků původního analogového signálu, které byly s určitou frekvencí vzorkovány a kvantovány. Za pomoci funkce

```
write (byte[]/short[] audioData, int offsetInBytes, int sizeInBytes)
 parametry:
  audioData - pole s uloženými daty k přehrání
  offsetInBytes - posunutí v datech odkud budou data přehrávána
  sizeInBytes - počet bytů v poli audioData, které se mají přehrát
```

se jednoduše přidávají do toku další data. Jde vlastně o jakýsi zásobník. Na jeho začátkek pomocí funkce `write` zapisujeme nová data. Z jeho konce jsou pak data odesílána na výstup.

Instance třídy `AudioTrack` může fungovat ve dvou základních módech: statický a proudový. V proudovém módu jde vlastně o souvislý tok dat, která jsou přidávána funkcí `write`. To je výhodné, pokud jsou zvuková data příliš objemná. Vždy tedy pracuji pouze s částí celých dat. Naproti tomu statický mód je vhodný pro paměťově méně náročné zvuky. Vyznačuje se také nižším zpožděním a menším rozdělením jednotlivých toků.

Po vytvoření instance třídy `AudioTrack` se k ní asociuje zvukový zásobník. Velikost tohoto zásobníku se specifikuje během vytváření a určuje, jak dlouho můžeme přehrávat zvuk, než nám dojdou data. U statického módu je jeho velikost určena maximální velikostí zvuku, který můžeme přehrát. U proudového módu jsou pak data zapisována v blocích menších, než je velikost zásobníku.

1.6 Eclipse

Eclipse je open source vývojové prostředí. Původně bylo vytvořeno pro jazyk Java. V aktuální verzi však podporuje celou řadu programovacích jazyků. Velkým přínosem je snadná rozšiřitelnost pomocí zásuvných modulů(plugin). Tak je Eclipse schopno, pouhým přidáním nového modulu, naučit se úplně nové funkce. Od jazyku Java přes C++ až třeba k XML.

Pomocí modulu Android SDK si rozšíříme Eclipse o kompletní balík nástrojů, které jsou určeny pro vývoj. Po provedení instalace dle podrobného návodu [1] můžeme začít s vývojem aplikací pro platformu Android. Pokud změníme perspektivu

[12] Pulse Code Modulation – formát digitální reprezentace analogových signálů

v Eclipse na DDMS, můžeme využít celé řady nástrojů vhodných pro vývoj a ladění. Můžete připojit paměťovou kartu, odeslat zprávu, uskutečnit hovor nebo třeba změnit pozici GPS[13] souřadnic.

Součástí celého tohoto balíčku je i emulátor viz obrázek 1.4. Ten má téměř totožné vlastnosti jako skutečné zařízení. Některé funkce jako třeba akcelerometry nebo vibrace tu však nenajdeme. I tak se jedná o dobrý produkt a zvláště v začátcích vývoje je velice užitečný. Finální aplikace však podle výrobce musí být testována přímo na konkrétním zařízení.

Obr. 1.4: Emulátor platformy Android v prostředí Eclipse.

1.7 Nativní kód

Android umožňuje psát a využívat také nativní kód. Za pomoci Android NDK[14], nám to vývojové prostředí Eclipse umožní. Modul Android NDK je sada nástrojů, které dokáží zapustit do aplikace komponenty používající nativní kód. Původní aplikace běží v Dalvik VM. Její nativní část pak běží o úroveň níže.

[13] Global Positioning System – globální družicový polohový systém s jehož pomocí je možno určit polohu a čas kdekoliv na Zemi

[14] Android Native Development Kit – balíček do prostředí Eclipse pro vývoj aplikací v nativním kódu

Můžeme tak využít funkce, které jsou v jazycích C/C++ již napsány. V některých případech tak dokážeme i výslednou aplikaci zrychlit. Nevýhodou však je nekompatibilita různých zařízení. Daná aplikace musí být poté psána s ohledem na konkrétní zařízení. NDK modul je podporován od verze Android 1.5. Podporovány jsou hlavně procesory rodiny ARM[15]. I jejich jednotlivé verze instrukčních sad se však mohou dosti lišit. Proto by měl být vývojář při používání nativního kódu velice opatrný. Profit z použití nativního kódu musí převyšovat nad ztrátou kompatibility mezi různými zařízeními. Těžko se na trhu uplatní výborná aplikace, když půjde spustit pouze na dvou telefonech na trhu.

Pro ilustraci uveďme několik knihoven, které jsou u verze(2.2) deklarovány jako stabilní:

- libc (standardní C knihovna)
- libm (matematická knihovna)
- OpenGL ES (3D grafická knihovna)
- liblog (Android knihovna pro vytváření logů)
- libjnigraphics (knihovna pro práci na úrovni pixelů)

1.8 Teoretický rozbor

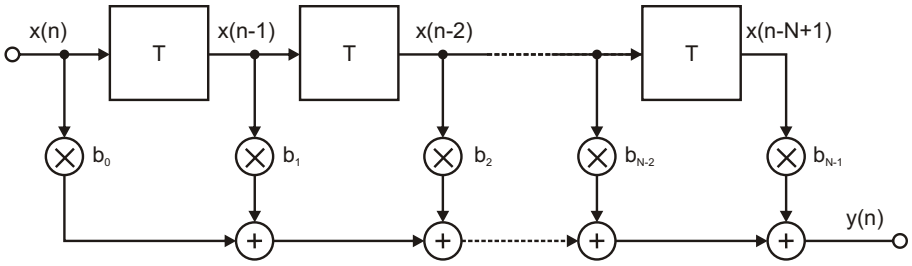
Definic slova filtr muže být celá řada. Všeobecně vyjadřuje způsob výběru určité entity, která má specifické vlastnosti z celé řady dalších entit. My se budeme věnovat filtrům, které pracují nad číslicovými audio signály. Signály si zde můžeme představit jako sadu vzorku s jinou frekvencí a amplitudou. Náš filtr zde provede selekci vzorku dle jejich frekvence. Podle toho, kde ve spektru působí, se filtry rozdělují na tři základní druhy:

- Dolní propust(Lowpass) – filtr propouští frekvence nižší než je jeho mezní kmitočet a naproti tomu tlumí frekvence nad tímto kmitočtem
- Horní propust(Highpass) – filtr propouští frekvence vyšší než je jeho mezní kmitočet a naproti tomu tlumí frekvence pod tímto kmitočtem
- Pásmová propust(Bandpass) – filtr propouští pouze frekvence, které jsou v rozmezí dolního a horního mezního kmitočtu

Podle toho jakou mají odezvu na signál existují dva druhy filtrů.

- filtr s konečnou odezvou(Finite Impulse Response filter – FIR)
- filtr s nekonečnou odezvou(Infinite Impulse Response filter – IIR)

[15] Advanced RISC Machine – je architektura procesorů vhodných pro mobilní zařízení, vyvinutá v Británii firmou ARM Limited



Obr. 1.5: Filtr s konečnou odezvou – FIR.

Rozdíl mezi nimi je možná více patrný ze schématu na obrázku 1.5. U FIR filtru jsou jednotlivé vazby vedeny přímo a sčítány do výsledného signálu. Naproti tomu u IIR filtru se objevuje zpětná vazba, která způsobuje nekonečnou odezvu systému.Vztah mezi vstupem a výstupem filtru z uvedeného schématu je podle [5](2.34) popsán rovnicí

$$y(n) = \sum_{i=0}^{N-1} b_i \cdot x(n-i). \tag{1.1}$$

$x(n)$ . . . vstupní signál

$y(n)$ . . . výstupní signál

b_i . . . koeficient úpravy v dané větvi

N . . . počet vzorků

Jedná se tedy o váženou sumu zpožděných vstupních vzorků. Často velice dlouhé posloupnosti vzorků musejí být počítány zvláštními postupy.

Rovnice pro jednotlivé typy filtrů byly převzaty ze zdroje [14]. Jejich přesný význam zde tedy nebude vysvětlován.

1.9 Nástroje pro verzování kódu

Zastřešujícím názvem pro všechny nástroje a postupy s tímto spojené je SCM[16]. Jedná se o nástroje umožňující ukládání jednotlivých verzí digitálních dat. Změna oproti běžnému uložení dat je v tom, že běžně jsou data uložena celá. Naproti tomu tyto nástroje umožňují ukládat pouze změny v datech. Dochází tak k veliké úspoře paměťového prostoru. Nejvíce využívány jsou právě při psaní zdrojových kódů. Dokáží totiž přehledně spravovat jednotlivé verze kódu. Jedním pohledem pak díky nim lze zjistit zvýrazněné odlišnosti verzí a snadno se dopátrat chyby.

Pomocí vývojového prostředí programátor může spravovat a upravovat jednotlivé verze na svém počítači. Reálně se jedná o systém složený z úložiště dat a z rozhraní pro jeho správu zabudovaném ve vývojovém prostředí. Nyní jsou nejvíce používány tyto dvě platformy.

- CVS – Concurrent Version System
- SVN – Subversion

Díky tomu, že se ve většině případů jedná o open source projekty, dostane zákazník produkt bezplatně. Nejčastěji jsou pak zpoplatněny samotná úložiště dat. Uživatel tedy platí pravidelné měsíční poplatky za služby, které na úložišti využívá. Některé projekty však mají základní službu bezplatně a účtují si až za nadstandardní služby. Může se jednat o větší datový prostor, objem dat za časovou jednotku, více uživatelů pracujících nad jedním projektem a mnoho dalších rozšíření. Celý tento model má jisté rysy Cloud computingu[17].

Největší rozvoj těchto technologií přišel zejména s rozvojem týmové práce. Právě u velkých týmových projektů je při použití těchto technik vidět velká úspora času a peněz. Jednotlivá vývojová centra jsou nyní na různých místech země. I přesto je díky těmto postupům dosahováno veliké přehlednosti a efektivity.

Nevýhodou těchto systémů je, že všechny operace nad daty provádějí servery. Při nedostupnosti těchto serverů je tak práce na softwaru do jisté míry omezena. Tento problém řeší některé decentralizované systémy jako například Git[18]. Ty umožňují vývojáři mít uloženu historii verzí na svém počítači.

[16] Software Configuration Management, je zastřešující název pro všechny techniky spojené s verzováním digitálních dat

[17] Cloud computing – model vývoje a využívání počítačových technologií založený na internetu, uživatel často platí za službu a né aplikaci jako takovou, politika plať jen co užíváš s anglického pay as you go

[18] Git – distribuovaný systém pro správu verzí, původně pro vývoj jádra Linuxu

1.10 Nástroje pro generování dokumentace

Jedná se o nástroje, které automaticky vytvářejí dokumentaci k psanému kódu. Jejich výstupem jsou často webové stránky, které přehledně reflektují a komentují kód. Vedle webové stránky mohou mít celou řadu výstupů jako nápověda v systému Windows či Linux a mnoho dalších. Problematika generování dokumentace souvisí i s typem programovacího jazyka či jeho syntaxí. U nejrozšířenějších jazyků se syntaxe příliš neliší, ale k různým odlišnostem určitě dochází. My se nyní budeme zabývat pouze otázkou generování dokumentace v prostředí jazyka Java.

Vzhledem k tomu, že celou dokumentaci je jen těžko možné vygenerovat počítačem, je nutné klasické komentáře trochu obohatit o některé instrukce. Překladač funkčního kódu se jim musí vyhnout, aby neovlivnily funkci samotného programu. Existuje více technik, jak toto zajistit. Asi nejrozšířenější z nich je, že dokumentace se zapisuje do rozšířeného tagu než je u komentáře.

```
/**
* toto je text, který bude zobrazen v dokumentaci
*/
```

Naproti tomu dva typy uvození u normálního komentáře, z kterého předchozí syntaxe vychází.

```
/*
* toto je komentář, který se v dokumentaci nezobrazí
*/

// toto je komentář, který se v dokumentaci nezobrazí
```

Jediným rozdílem jsou tak pouze dvě hvězdičky `/**` v prvním řádku, namísto jedné `/*` u komentáře. Existuje celá řada vnitřních příkazů samotné dokumentace. Pro ilustraci jen představme některé z nich.

```
    syntaxe formátu Javadoc
/**
* Dokumentace kódu. První věta je jako hlavní popisek entity.
* @author Martin Janovsky
* odkaz na jiné místo v dokumentaci {@link getValues()}
* @param parametr text k proměnné/parametru
* @return text k návratové hodnotě funkce
* @version verze kódu
*/
```

To bylo několik základních příkazů. Každý generátor se trochu liší, a tak čím méně užívaný příkaz se užije, tím je větší pravděpodobnost, že daný generátor příkaz nebude podporovat. V praxi si vývojář zvolí nástroj, který mu vyhovuje a toho se drží. Zakrátko si tak zvykne na jeho specifika, která přijme za svá.

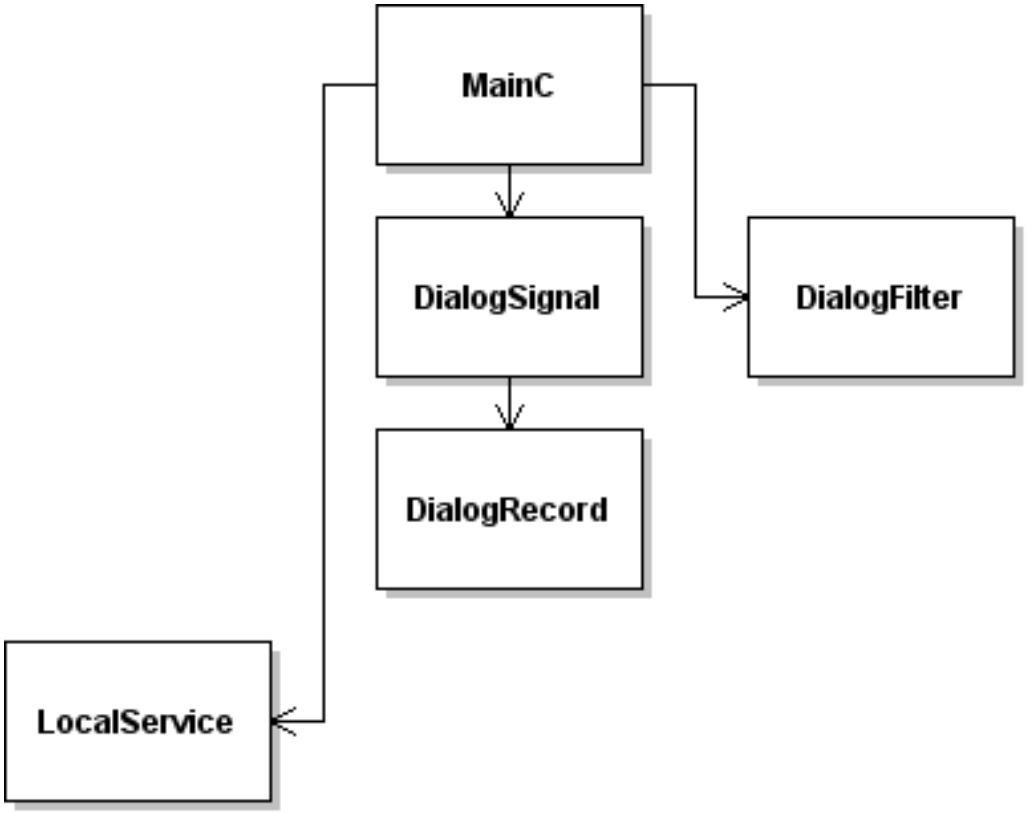
Často se ve spojení s dokumentací zapomíná na to, že dokumentace musí být věcná. Není totiž výjimkou, že se setkáte s komentářem tohoto typu.

```
/**
* Methoda getTypeOfValue() vrací typ proměnné.
* @param parametr, s kterým je funkce volána
* @return navratová proměnná
*/
```

Z tohoto komentáře nezúčastněná osoba mnoho nezjistí. Vývojář by si měl být vědom, že čím více smysluplné dokumentace napíše, tím přehlednější je kód pro ostatní, což urychluje další vývoj.

2 VÝSLEDKY STUDENTSKÉ PRÁCE

Celá aplikace měla být navržena tak, aby nabízela uživatelsky jednoduché a efektivní rozhraní pro práci s algoritmy pro zpracování signálů. Šlo též o to, najít vhodnou metodu práce se signálem. To jak na úrovni jeho zpracování, tak na úrovni uživatelské, pomocí níž se data interpretují uživateli. Současně se mělo přihlédnout k tomu, že výpočetní schopnosti mobilních zařízení jsou omezené. Při tom byl kladen důraz spíše na problematiku vlastní aplikace a její užití na platformě Android než na vlastní algoritmy zpracování signálů.



Obr. 2.1: Pohled na zjednodušenou architekturu aplikace FilterOid.

Aplikace je po nahrání do emulátoru nebo stažení do mobilního zařízení připravena k použití. V menu je dostupná pod názvem FilterOid. Při startu aplikace se spustí hlavní aktivita `MainC`, která řídí další události celé aplikace. Tato aktivita tedy tvoří základní uživatelské rozhraní mezi knihovnou a uživatelem. Pod ní jsou aktivity, které jsou spojeny s výběrem parametrů filtru a signálu. U nastavení signálu je možnost nahrání vlastního záznamu pomocí mikrofonu. Pomocí služby `LocalService`, běžící na pozadí, dochází k vlastnímu zpracování signálu. Jedná se o jakési rozhraní mezi knihovnou a uživatelským rozhraním. Tato služba využívá

více třídu, která se stará o chod vlastní knihovny. Celá knihovna je pak spouštěna v novém vlákně, aby nezatěžovala vlákno starající se o vlastní grafické prostředí. Pomocí prvotní aktivity tak uživatel může služby knihovny spouštět, případně měnit některé její parametry. Celé prostředí zároveň používá metody pro dorozumění mezi jednotlivými částmi. Tak je možné vzájemně sdílet data potřebná k chodu aplikace. Pro ilustraci je výše popsaná architektura zobrazena na obrázku 2.1. Zjednodušeně se nechá říci, že všechny třídy napravo se starají o uživatelské prostředí. Zatímco třída `LocalService` nalevo od nich zajišťuje spojení s knihovnou a její ovládání.

2.1 Vlastní funkce

Konfigurační soubor

Velice důležitým stavebním prvkem každého projektu na platformě Android je soubor `AndroidManifest.xml`. Pomocí něho je deklarováno základní nastavení chování celé aplikace. V kódu

```
<activity android:name=".MainC"
          android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
<activity android:name=".DialogRecord"
          android:label="@string/dialog_record"
          android:configChanges="orientation">
 ...
 zbytek deklarace aktivity DialogRecord a deklarace dalších aktivity
 ...
```

je postupně deklarováno, že aplikace bude obsahovat aktivitu `MainC`, její popisek, kdy má být spuštěna a to, že je hlavní aktivitou celé aplikace. Stejně jako tato aktivita, musejí být v tomto souboru uvedeny všechny další, které za běhu aplikace budeme chtít používat. Snadno se na tuto konstrukci zapomene a chyba se pak hledá v samotném kódu. Z ladících výpisů se o této chybě také nedozvíme. Vůbec celý ladící subsystém vývojového prostředí je velice nepřehledný a výpisy chyb jsou často ohlašovány nesouvisejícími popisky. Jedná se však často o velice známé problémy, ke kterým se nechá snadno najít řešení.

Po deklaraci aktivit je nutné uvést služby, které aplikace obsahuje. Zároveň, abychom mohli v budoucnu využít získávání informací, je nutné deklarovat povolení přístupu k této službě.

```
<service android:enabled="true" android:name=".LocalService" />
```

Služba `LocalService` je tak deklarována jako služba naší aplikace.

Poslední důležitou částí deklarace souboru `AndroidManifest.xml` je výpis oprávnění. Ke každé hardwarové či softwarové službě, ke které chceme přistupovat, se musí uvést oprávnění.

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

Takto nastavíme přístup k paměťové kartě a nahrávání audio souborů, tedy mikrofonu a hardwaru, s tímto spojeným. Díky tomuto má uživatel přehled, ještě předtím než samotnou aplikaci nainstaluje, o tom, k jakým médiím bude přistupovat. Tak, pokud uživatel tyto varovaní a popisy čte, se nestane situace, že aplikace začne volat mezistátní hovor či se připojovat k internetu bez jeho vědomí.

Velice důležité při psaní kódu v `AndroidManifest.xml` je dávat pozor na chyby. Malá chyba v tomto souboru totiž může způsobit kolaps celé aplikace. Nemusí však jít pouze o chybu způsobující úplné zhroucení aplikace. Může dojít i k situaci, kdy se celá aplikace chová navenek standardně, ale zároveň nám v ní něco nefunguje. To byl příklad neuvedení aktivity ve výpisů a následné pokusy o její spuštění v aplikaci. Kompilátor vás na tyto chyby často neupozorní, a tak se velice špatně nacházejí. Často pak dochází ke zdlouhavému hledání nesmyslných chyb.

Spojení vlastního kódu s grafickým podkladem a řetězci

Jak už bylo uvedeno, zobrazení je dáno pomocí šablon `XML`. Pomocí tagů se nastaví jednotlivé parametry celkového zobrazení a následně vlastnosti zobrazení jednotlivých entit. Pomocí přiřazovací metody `setContentView` jsou pak jednotlivé šablony přiřazeny dané aktivitě. To jak vypadá grafická část však může být dáno i samotným kódem. Grafická část tak může být naprosto změněna i při běhu programu.

V souboru `strings.xml` jsou uloženy optimálně všechny řetězce, které aplikace používá. V ideálním případě je veškerý text, který se zobrazuje uživateli, uveden v tomto souboru. Editací tohoto jediného souboru tak mohu snadno měnit jazyk programu. Například má aplikace tento soubor v dvojím vyhotovení pro anglickou a německou verzi. Podle konfigurace přístroje zjistím jazyk uživatele a rovnou mu

předložím verzi v jeho jazyce. Tím, že jsou všechny texty na jednom místě, se vyhneme problému s přepisem pouze části řetězců.

Hlavní aktivita MainC

Hlavní aktivita `MainC` rozšiřuje třídu `Activity`, po které dědí základní sadu funkcí, které tato třída obsahuje. Dále implementuje třídu `OnClickListener`, která obstarává obsluhu tlačítek na dotykovém displeji.

Po deklaraci proměnných na začátku třídy následuje metoda `onCreate`, kterou musí obsahovat každá aktivita. Ta je volána po vytvoření instance této třídy a provádí operace nutné pro zobrazení obsahu na displeji. Do jisté míry se tato funkce nechá považovat za konstruktor celé třídy. V našem případě nejprve nastavíme obsah na naši základní šablonu, která je v souboru `main.xml` v adresáři `../res/layout/`. Dále musíme nastavit obsluhu tlačítek a registrovat přijímač dat od služby. Vzhledem k tomu, že některé úkony spojené s inicializací grafického prostředí se musí provést nejen při vytvoření samotné aktivity, musí být některé tyto úkony zařazeny v pomocné funkci `InitUI`. Tato funkce je pak volána na konci vlastní funkce `onCreate`, aby nedošlo k duplicitě kódu. Prvotní inicializace aktivity hlavně obstarává zobrazení předvolené posloupnosti obrázků. Ta nám na displeji vytvoří blokové schéma funkcí, které vykonává. Jednoduchá posloupnost zdroje signálu, zvolených filtrů a výstupu nabízí intuitivní ovládání pouhým dotykem na uvedený blok. Protože po překrytí aktivity a při opětovném zobrazení je volána funkce `onCreate`, je nutné pamatovat si posloupnost jednotlivých bloků. Jejich opětovné načtení pak na začátku zajistí metoda `getLastNonConfigurationInstance`. Jednoduchou podmínkou je pak zjištěno, jestli se má použít základní posloupnost a nebo je k dispozici jiná.

Spoluúčast na počáteční inicializaci prostředí má tedy i funkce `InitUI`. Ta deklaruje tlačítka a spojí je s metodou, která je bude obsluhovat. Po deklaraci entity `GridView`, která zajišťuje zobrazení obrázků v mřížce, je nutno ji spojit s adaptérem obrázků. K jeho deklaraci se dostaneme na konci třídy`MainC`. Poslední hlavní funkcí metody `InitUI` je registrace přijímače zpráv od služby `LocalService`. Na konci je ještě funkce `Log.d` pro záznam běhu aplikace, která slouží hlavně pro snadnější orientaci při odhalování chyb.

Z výčtu metod je další `onActivityResult`. Ta se stará o zpracování příchozích informací od ostatních aktivit. Jedná se o návratovou metodu k volání funkce `startActivityForResult`, která aktivitu spustí. Spouštění aktivit je dvojího typu, a to s výsledkem a bez výsledku. V našem případě chceme převzít data od spouštěných aktivit, tudíž používáme metodu pro dosažení výsledku. Hned na začátku

se rozhodne, jestli výsledek spuštěné aktivity je v pořádku. Pokud není, vytvoříme o tom záznam. To platí například, když ve spuštěné aktivitě bylo pouze stisknuto tlačítko zpět. Pokud výsledek je v pořádku, musí se zjistit, z které aktivity data pocházejí. Pokud jsou z nastavení signálu, převedou se a uloží ve vhodném formátu. Navíc je o tom zobrazena informace uživateli pomocí statického volání třídy `Toast`. V případě, že data pocházejí od nastavení filtru, je situace jen o trochu složitější. Musí se zkontrolovat, zdali byl požadavek na smazání filtru či změnu jeho parametrů. Navíc musí být změněn obrázek, který představuje filtr v blokovém schématu. Při nastavení pásmové propusti se tak zobrazí blok znázorňující pásmovou propust. Všechny tyto operace samozřejmě probíhají jen nad filtrem, který uživatel editoval. O všech provedených změnách je opět uživatel informován.

Jednou z metod, které obsluhují dotyky displeje, je `onItemClick`. Ta se v našem případě stará o obsluhu dotyků jednotlivých obrázků blokového schématu. Pomocí pozice obrázku, který byl stisknut, se určí, jaká operace se provede. V případě prvního obrázku se volá nastavení signálu. Aby volaná aktivita mohla reflektovat aktuální nastavení signálu, musí ji být tento signál spolu s jejím spuštěním odeslán. V případě posledního obrázku se sbalí data do vhodné podoby a spustí se s nimi služba `LocalService`. Data přitom nesou informace o počtu filtrů a jejich parametrech o signálu a několik dalších spíše pomocných proměnných. Jestliže byl stisknut displej nad obrázky mezi prvním a posledním, určí se jeho pozice a spustí aktivita s nastavením tohoto filtru. Aktuální nastavení filtru je odesíláno společně se spuštěním aktivity.

Druhou metodou starající se o dotyky displeje je `onClick`. V aktuální verzi aplikace ale obsluhuje už jen jedno tlačítko. Jde o obsluhu přidávání filtru. Po kontrole maximálního počtu filtrů je automaticky přidán na konec posloupnosti výchozí filtr. O uvedené změně je informován adaptér mřížky, aby byl ihned zobrazen na displeji. Pokud bylo dosaženo maximálního počtu filtrů, je na to uživatel upozorněn a tlačítko pro přidávání dalších se stane neaktivním.

O obsluhu stisknutí klávesy menu na přístroji se starají celkem čtyři metody. O vlastní zobrazení menu po stisku klávesy se stará metoda `onCreateOptionsMenu`. Ve vlastním menu máme pouze položky nápověda a ukončení aplikace. O obsluhu dotyku jednotlivých položek se stará metoda `onOptionsItemSelected`. Ta rozhodne, které položky se uživatel dotkl a podle toho zavolá jednu ze dvou metod. První z nich `openOptionsDialog` spustí dialog s nápovědou k aplikaci. Druhá metoda `exitOptionsDialog` aktivuje dialog vypnutí celé aplikace.

Další tři metody obsahují maximálně dva řádky kódu, ale jejich funkce je velice důležitá. Při vypínání aktivity je volána metoda `onDestroy`, v které musíme od-

registrovat přijímač. Pro uložení potřebných dat při zastavení běhu aktivity slouží metoda `onRetainNonConfigurationInstance`. V ní uložíme seznam jednotlivých bloků našeho schématu. Při změně konfigurace potřebujeme povést opětnou inicializaci. K tomu slouží metoda `onConfigurationChanged`, v které je volána metoda `InitUI`. V našem případě se za změnu konfigurace bude nejčastěji považovat otočení displeje.

Na konci deklarace celé třídy `MainC` jsou ještě uvedeny dvě pomocné třídy. Třída `ImageAdapter`, která se stará o zobrazení jednotlivých obrázků v mřížce. Druhou třídou je `LocalServiceReceiver` starající se o příjem zpráv od služby `LocalService`. Obě třídy jsou rozšířením tříd `IBaseAdapter`, respektive `BroadcastReceiver`.

Nastavení signálu

Nastavení signálu je možné pomocí aktivity `DialogSignal`. I zde je nutné za několika pomocnými proměnnými deklarovat metodu `onCreate`. Na jejím začátku přečtu parametry, s kterými byla aktivita spuštěna. Tyto parametry jsou potřeba na počáteční nastavení formulářových prvků grafického rozhraní. V případě okamžitého uložení by se totiž tyto parametry přepsaly výchozími hodnotami. Další částí metody je inicializace jednotlivých tlačítek a posuvníků a spojení s jejich obsluhou. O něco složitější je inicializace entity pro výběr typu signálu. Nestačí ji totiž vytvořit a spojit s obsluhou, ale je také nutné propojit ji s polem obsahujícím položky, z nichž se bude vybírat. Je také nutné znemožnit využití tlačítka pro použití nahraného zvuku, pokud soubor s tímto zvukem není dostupný. Na konci je ještě potřeba nastavit všechny entity formuláře podle uložených parametrů.

Celá třída implementuje nejen `OnClickListener` k zabezpečení dotyků tlačítek, ale také `OnSeekBarChangeListener`. Jedná se o třídu sloužící k obsluze posuvníků. Je nezbytné u ní implementovat metodu `onProgressChanged`, která nastaví popisky při změně hodnot a `onStartTrackingTouch` s `onStopTrackingTouch`, které jsou volány po začátku posunování, respektive na konci posunování. Implementace dotyku tlačítek zajišťuje opět metoda `onClick`. Pomocí direktivy `switch` se rozhoduje, kterého tlačítka se uživatel dotkl. V případě požadavku na uložení generovaného signálu se jeho parametry sbalí do instance třídy `InputSignal` a pomocí metody `setResult` se odešlou aktivitě `MainC`. V případě, že uživatel zvolí možnost použití nahraného signálu, se navíc v odeslaných datech indikuje, že signál byl nahrán. Poslední možností je spuštění aktivity pro nahrávání zvuku.

Předposlední metodou v třídě `DialogSignal` je `onActivityResult`, která opět zajišťuje obsluhu návratu z aktivity `DialoRecord` pro nahrávání zvuku. Poslední implementovaná metoda `onPause` zabezpečuje stav v případě zastavení aktivity.

Odešle aktivitě `MainC` vyrozumění o tom, že výsledek aktivity není platný, aby ho nebrala v úvahu.

Nahrávání zvuku

Třída `DialogRecord` se stará o nahrávání zvukového záznamu, jeho přehrávání a uložení. Po deklaraci pomocných proměnných následuje zápis metody `onCreate`. V té se vedle inicializace tlačítek a posuvníku provádí opět spojení s jejich obsluhou. Na konci metody je ještě kontrola existence zvukového záznamu. V případě jeho nepřítomnosti musíme deaktivovat tlačítka přehrávání a uložení.

V dalších metodách je stejně jako v jiných třídách implementována obsluha posuvníku. Naproti tomu obsluha metody `onPause` dostála jistých změn. Díky tomu, že nahrávání nebo přehrávání běží v jiném vlákně, není v případě zastavení aktivity toto nahrávaní přerušeno. Naproti tomu vlákno pro přehrávání zůstává aktivní a dokončuje svou úlohu.

Ani v této třídě nechybí metoda `onClick` obsluhující dotyky tlačítek. V případě, že chceme nahrávat, se spustí nové vlákno a v něm metoda `record`. Je také nutné indikovat nahrávání a deaktivovat pak tlačítka pro přehrávání a uložení. Jestliže uživatel zvolil přehrávání, je spuštěna opět v novém vlákně metoda `play`. Při požadavku na uložení zvuku se zabalí potřebné proměnné a odešlou se nadřazené aktivitě. Po odeslání dat se aktuální aktivita ukončí direktivou `finish`.

Hlavní metodou starající se o vlastní nahrávání je `record`. Ta na svém začátku zkontroluje existenci souboru. Pokud soubor existuje, tak ho smaže a vytvoří nový. V další části je zahrnuta do direktivy `try-catch`. Nejprve je otevřen proud dat do souboru. Po určení velikosti bufferu se vytvoří instance třídy `AudioRecord` s potřebnými parametry a pomocné pole pro přechodné ukládání zvukových dat. Poté následuje samotné nahrávání. Data se ukládají v cyklu, kdy za jeden běh se uloží data délky jednoho bufferu. V tom samém cyklu se ještě tyto data odešlou na otevřený proud, který je zapisuje do souboru. Na konci metody se už jen uzavře soubor a vypíše záznam o souboru a jeho velikosti.

Velice podobná jako metoda `record` je metoda `play`. Odlišuje se od předchozí jen v několika bodech. Místo třídy `AudioRecord` se použije třída `AudioTrack` a místo přesunu dat od mikrofonu do souboru se data přesouvají ze souboru na výstup. Samotný přesun těchto dat se odehrává ve dvou cyklech. V prvním se čtou data a odesílají na výstup až do té doby, než zůstanou v souboru už jen data kratší než buffer. V druhém cyklu je pak zbytek dat vybírán ne po kusech velikosti bufferu, ale po jednotlivých vzorcích. Tímto rozdělením se dosáhne toho, že se nemusí čekat na přečtení celého souboru.

Nastavení filtru

Třída `DialogFilter` je v podstatě pouze upravenou verzí `DialogSignal`. Místo instance třídy `InputSignal` se pracuje s třídou `Biquad`, která uchovává jednotlivé parametry každého filtru. Navíc je zde tlačítko pro smazání filtru, které odešle tento požadavek nadřazené aktivitě a tuto aktivitu ukončí. Naopak tlačítka zajišťující obsluhu nahrávaného zvuku tady nejsou. Není zde ani metoda `onActivityResult`, protože zde není ani žádná podřízená aktivita.

Obsluha knihovny

Při spuštění služby `LocalService` dochází, stejně jako u aktivity, k volání funkce `onCreate`. Uvedená funkce zde však není využívána jako u aktivit. Jsou zde provedeny pouze operace nadřazené třídy `super`[1]. Poté je program směrován do funkce `onStartCommand`. Toto přesměrování má jednoduchý důvod. Předpokládejme případ, že za běhu této služby chce uživatel opustit aplikaci s tím, že se hodlá vrátit. Při opětovném vrácení se ke službě již není volána metoda `onCreate`, ale je zde připravena metoda `onBind`, která je volána právě při opětovném navázání relace mezi aktivitou a službou. Při vyšším vytížení hardwaru může systém uvolnit nepoužívanou aktivitu z paměti a nezatěžovat tak zbytečně hardware. Když pak uživatel znovu spustí hlavní aktivitu, systém pouze obnoví relaci s již běžící službou. Ta přitom vůbec svůj běh nemusela přerušit. Služba je přitom postavena tak, aby se při jejím startu či opětovném navázání relace s aktivitou prováděli téměř totožné operace.

V metodě `onStartCommand` se kontroluje, jestli přišly při spuštění všechny požadované informace. Následuje uložení těchto parametrů. Toto uložení je o něco složitější díky potřebě uložení všech filtrů posloupnosti. Na konci je ještě volána metoda `onStartservice`. Ta založí nové vlákno. V novém vlákně vytvoří instanci třídy `AudioTrackSample` s potřebnými parametry a zavolá její metodu `run`. Celá knihovna je tak pomocí tohoto rozhraní oddělena od uživatele a nechá se tak využívat separovaně.

Také zde nalezneme funkci pro ukončení celé služby. Funkce `onDestroy` opět vedle standardních operací volá funkci `onStopservice`. Ta před ukončením služby odešle oběžník s daty pro aktivitu. A zkontroluje, zdali běží vlákno knihovny. Pokud ano, vyšle mu požadavek o přerušení.

[1] super – identifikátor třídy rodiče v prostředí jazyka Java

Knihovna

V celé službě `LocalService` ovládáme instanci třídy `AudioTrackSample`, která dále přistupuje k vlastnímu signálu. Třída `AudioTrackSample` se tedy stará o přehrávání, generování i úpravu zvuku najednou. V konstruktoru této třídy je pomocí

```
mTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
    input.samplingFrequency, adConf, adEncoding, bufferLength,
    AudioTrack.MODE_STREAM)
```

vytvořena instance třídy `AudioTrack`. Pomocí parametrů určujeme typ dat, frekvenci či velikost bufferu. Data jsou pak přehrávána pomocí metody

```
mTrack.write(mBuffer, 0, mBufferSize).
```

Instance třídy `AudioTrack` se pak stará o vlastní odeslání signálu na hardware zařízení. V konstruktoru celé třídy `AudioTrackSample` je hlavní nastavení všech požadovaných parametrů. Nejpodstatnější z nich je velikost bufferu.

V dalším kódu je několik pomocných metod pro dodatečné úpravy formátu. Metoda `writeSamples` převádí pole proměnných z typu `float` na `short`. Všechny signály jsou generovány ve formátu `float`, který je vhodnější i pro početní úlohy. Na vstup třídy `AudioTrack` však lze odesílat pouze data ve formátu `short` nebo `byte`. Převodu je dosaženo zápisem.

```
short_value = float_value * Short.MAX_VALUE * gain_value
```

kde `gain` označuje poměrnou hodnotu amplitudy signálu. Popisek k vlastnostem signálu generuje metoda `makeLabelOfSignal`. Pokud uživatel chce tisknout výsledné signály do souborů, je zde připravena přetížená metoda `printIntoFile`. Přetížení spočívá v tom, že je možné zapisovat do souboru formát `float` i `short`. Záznam informací o výpočtech má na starosti metoda `printToLogFile`. Ta převezme řetězec a zapíše ho do daného souboru na konec.

Pro generování signálů jsou zde funkce `generateToneSin` a z ní odvozené funkce pro pilový a obdélníkový signál. Signál je generován na principu narůstání kruhového vektoru. Pro příslušnou frekvenci je spočítán přírůstek. S každým přírůstkem se pak počítá jeden vzorek signálu. Takto získané vzorky funkce sinus jsou pro další průběhy dále upravovány. V případě obdélníkového signálu se použije funkce `signum`, která v případě kladné hodnoty přepíše hodnotu na 1 a v případě záporné na -1. U pilového průběhu se podle [12](1) vychází z rovnice

$$x(t) = t - floor(t), \tag{2.1}$$

kde `x(t)` je výsledný vzorek, `t` je čas a `floor` je funkce zaokrouhlujíc dolů na celá čísla.

Všechny operace dohromady pak sdružuje metoda `run`. Na jejím začátku se rozhoduje, který signál budeme generovat. Jsou inicializovány všechny filtry. To probíhá tak, že pro každý filtr `Biquad` vznikne instance třídy `BiquadFilter`, která si spočítá koeficienty filtru. Všechny jednotlivé instance jsou ukládány do pole. Následným krokem je samotný výpočet. Každý vzorek vstupní posloupnosti prochází postupně všemi filtry. Následuje převod vzorků z hodnot formátu `float` na `short` a samotné odeslání na výstup. O všem je také vytvořen záznam.

Pomocné třídy

K přehlednosti a kvalitě kódu přispívá několik pomocných tříd a výčtových typů. Dva výčtové typy `filter_types` a `signal_types` uchovávají typy filtru, respektive signálu. Jen o trochu složitější jsou třídy `InputSignal` a `Biquad`. První zmiňovaná uchovává parametry signálu jako jsou jeho délka, frekvence, vzorkovací frekvence, amplituda, typ a informace o tom, jestli se jedná o nahrávaný signál. U filtru se uchovává mezní frekvence, zisk či útlum, šířka pásma, typ filtru a několik dalších proměnných, kterých je využíváno při výpočtech. Za zmínku také stojí fakt, že uvedené třídy implementují třídu `Serializable`, což umožňuje snadné předávání jejich instancí mezi různými procesy.

2.2 Automaticky generovaná dokumentace

Pomocí balíčku Javadoc plugin vývojového prostředí Eclipse byla vygenerována dokumentace k aplikaci. Při jejím generování byla nastavena volba pro zahrnutí pouze kódu s označením `public`. Náhledy hlavní stránky dokumentace na obrázku B.1 a stránky třídy `AudioTrackSample` na obrázku B.2 jsou uvedeny v příloze. Potřebnou formu zápisu, v které se pravidelně opakuje minimum výrazů, si nečinilo problém zapamatovat. Trochu zmatek v dokumentaci může vytvořit záměna dokumentace za komentář. Vzhledem k tomu, že celé generování je otázkou několika vteřin, se jeho využití nesetkává s výraznými zápory.

2.3 Správa verzí kódu

Ve vývojovém prostředí Eclipse bylo využito systému pro správu úložiště SVN Repository. Podoba tohoto systému je naznačena na obrázku B.4. Jako úložiště dat

byl zvolen systém Assembla. Po zaregistrování byl zvolen bezplatný tarif s jedním možným úložištěm. Celý projekt Assembla je postaven tak, že za téměř každé rozšíření se vyžaduje poplatek. Náhled na zobrazení verze aplikace na tomto systému je zobrazen na obrázku B.3. Po zřízení účtu nebyl větší problém spojit úložiště s vývojovým prostředím. V průběhu tvorby aplikace bylo verzování kódu používáno, což je patrné z grafu, který na obrázku B.5 zobrazuje vývoj jednotlivých verzí kódu. Samotný graf není příliš zajímavý, ale pro přehlednost se hodí například s využitím týmu programátorů. Jednotlivé větve se pak vzájemně prolínají a graf tak může představovat výbornou pomůcku.

2.4 Testování na více zařízeních

Kvůli tomu, že platforma Android není příliš stará, se zde objevují problémy spojené s vývojem. Ze začátku šlo hlavně o slabší dokumentaci. V průběhu vývoje celé aplikace se však dokumentace velice zlepšila. Do jisté míry je také patrné, jak jsou postupem času vývojářům předkládány obsáhlejší nástroje. Od psaní samotného kódu k pouhému spojování jednotlivých bloků. Vývoj samotných aplikací se tak zrychluje a zkvalitňuje. Má to však za následek, že jakmile něco balíček nepodporuje, je složitější dosáhnout požadovaného řešení. Pokrok, který za poslední dva roky tato platforma udělala, je však obrovský. Můžeme se tak jen těšit, co nového nám ještě přinese.

Na začátku vývoje byla aplikace testována na emulátoru. Při přechodu od semestrálního projektu k diplomové práci se přešlo k testování na zařízení LG Optimus One. Konečná verze aplikace pak byla testována také na zařízení Samsung Galaxy Spica.

- LG–P500 Optimus One, Android 2.2, ARMv6 600 MHz, RAM 256MB
- Samsung Galaxy Spica, Android 2.2, CPU Samsung S3C6410 800 MHz, RAM 512MB

Pro představu složitosti jednotlivých operací, byl kód rozšířen o výpis časů jednotlivých funkcí. Měřena byla doba filtrace, generování signálu, zápisu signálu do souboru a zápisu signálu na výstup. Průměrné hodnoty posledních tří jsou uvedeny v tabulce 2.1. Pokud jde o dobu zápisu signálu na výstup, dosahovala zařízení téměř totožných výsledků. Těchto hodnot bylo dosaženo až při vhodném zvolení velikosti bufferu. Problém byl v tom, že při příliš malém bufferu byl výstupní zvuk trhaný a při velkém byla data odesílána příliš dlouho. Velikost bufferu je nyní volena jako dvojnásobek minimální velikosti dané funkcí `getMinBufferSize`. Pokud jde o generování signálu, jsou obě doby také téměř totožné. V případě načítání souborů se

však doby již velice liší. To samé platí pro zápis do souboru. Do jisté míry to mohlo být ovlivněno tím, že na zařízení Galaxy Spica bylo spuštěno více vláken přistupujících k paměťové kartě. Dalším faktorem mohl být i fakt, že se zapisovalo do dvou souborů ve stejnou dobu. Faktem ale zůstává, že zápis na paměťovou kartu je příliš pomalý. Při vývoji by na to mělo být hleděno a využíváno oddělených vláken pro obsluhu práce se soubory. Posledním měřeným časem byla samotná filtrace. Nejdříve

Název zařízení	Doba generování signálu [s]	Doba výpisu signálu do souboru [s]	Doba zápisu signálu na výstup [s]
LG Optimus One (Soubor-58kB)	0,51	11,45	3,54
Galaxy Spica (Soubor-44kB)	5,1	-	2,59
LG Optimus One (5s fs=44100)	1,22	154,32	3,49
Galaxy Spica (5s fs=44100)	1,04	600,25	3,83

Tab. 2.1: Vybrané průměrné hodnoty dob práce se signálem.

se používal sériový výpočet jednotlivých filtrů, kdy v cyklu byl vždy spočítán jeden filtr, a až pak se přešlo k počítání dalšího. S požadavkem snížení výpočetního času se přešlo k částečně paralelní verzi, kdy každý vzorek prochází postupně všemi filtry, a až poté se přistupuje k výpočtu dalšího vzorku. Měření a následná analýza ale odhalily, že oba postupy jsou téměř totožné. Doby filtrace pro různé počty filtrů jsou v tabulce 2.2. Rozdíl obou zařízení je více patrný z grafu na obrázku 2.2. Je vidět, že vyšší frekvence procesoru vůbec nepředznamenává vyšší výpočetní výkon. V případě zařízení Samsung Galaxy Spica, s taktem procesoru 800MHz, jde o procesor Samsung. Na druhou stranu u Optimus One se jedná o procesor ARMv6 s frekvencí 600MHz.

2.5 Návod k obsluze aplikace

Klasickou formou přístupu k novým aplikacím je Android Market. Zde si uživatel aplikace vyhledá, uloží a nainstaluje. U nahrávání do zařízení či emulátoru máme dvě možnosti. Pokud v prostředí Eclipse spustíme projekt, aplikace se nainstaluje do připojeného zařízení. Jestliže žádné není připojeno, nahraje se aplikace do emulátoru. Toto vše vyžaduje mít nainstalované prostředí Eclipse s balíčkem Android SDK.

Počet filtrů	Počítání u LG Optimus One [s]	Počítání u Galaxy Spica [s]
1	0,473	0,169
2	1,01	0,691
3	1,378	1,486
4	1,78	4,229
5	2,26	4,276
6	2,554	7,54
7	2,944	9,994
8	3,274	13,064
9	3,639	14,747
10	4,041	20,371

Tab. 2.2: Doby filtrování signálu při změně počtu filtrů.

Samotné nahrání připravené aplikace do zařízení či emulátoru je možné i s pomocí příkazu

```
adb install nazev.apk.
```

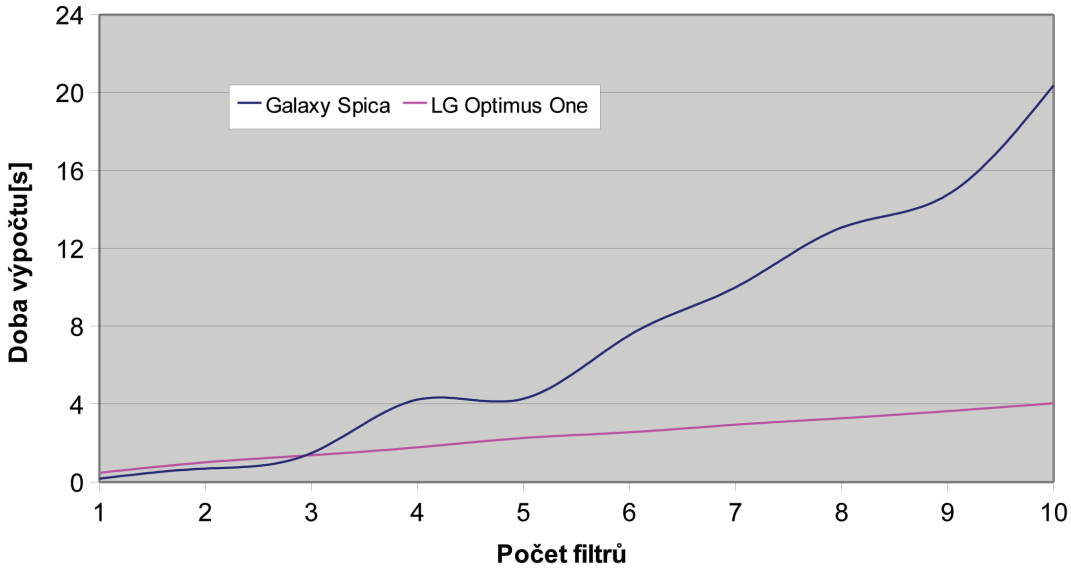
Po nahrání do zřízení je aplikace připravena ke spuštění. Na obrázku 2.3 je znázorněno menu přístroje. Dotykem ikony označené FilterOid bude aplikace spuštěna.

Na obrázku 2.4 je vidět základní nabídka, kterou aplikace po zapnutí nabízí. Dotykem boxu s označením `IN` budeme přesměrováni do nastavení signálu. Dotykem boxu `OUT` bude přehrán výstup blokového schématu. Ve spodní části obrazovky se nachází tlačítko `AddFilter` pro přidání filtru do blokového schématu. Celkový počet filtrů je omezen, na což je uživatel při přidávání filtru nad limit upozorněn(viz obrázek 2.5). V pravé části je ještě zaškrtávací políčko `Debug`. Při jeho zatržení se současně s přehráním výstupu vzorky tohoto výstupu zapíší i do souboru.

Při stisknutí tlačítka menu na zařízení je vyvoláno menu na obrázku 2.6. Obsahuje položky nápověda aplikace a volbu ukončení. Zobrazení jednoduché nápovědy je na obrázku 2.7.

Na obrázcích 2.8 a 2.9 je vidět dialog pro nastavení signálu. V části pro nahrávaný signál jsou pod dvěma tlačítky volby nahrání zvuku a jeho potvrzení jako vstupního signálu. V části věnované generovanému signálu je volba typu signálu viz obrázek 2.10. Dále pak tři posuvníky pro nastavení frekvence, amplitudy a délky signálu. Celý dialog je ukončený potvrzením parametrů pro generovaný signál.

V případě spuštění aktivity pro nahrávání zvuku na obrázku 2.12 se ověřuje, zdali je přítomen starý zvukový záznam. Pokud není, tlačítka pro jeho přehrání a uložení

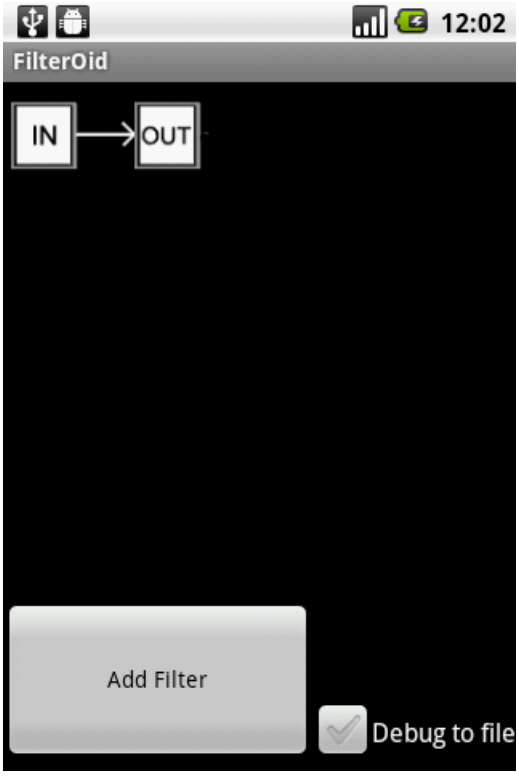


Obr. 2.2: Závislost doby výpočtu na počtu filtrů.

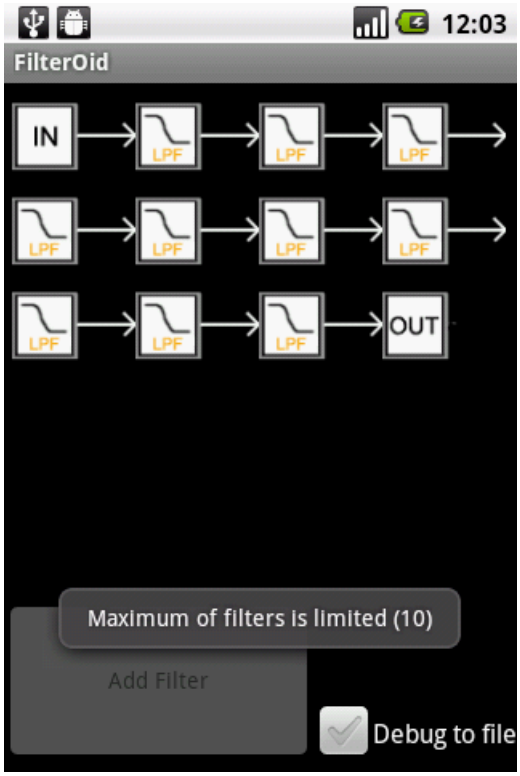
zůstanou neaktivní viz obrázek 2.11. Pomocí posuvníku nastavíme požadovanou vzorkovací frekvenci a poté intuitivně používáme tlačítka `Record`, `Play` a `Save`. Za zmínku jistě stojí možnost nahrání a přehrání zvuku s odlišnou vzorkovací frekvencí. Toho se dá využít pro jakési předzpracování signálu. Poslední obrázek 2.13 ukazuje dialog pro nastavení filtru. Vedle typu filtru, mezní frekvence a koeficientu `Q` je zde možnost smazání filtru.

Uživatel tedy zvolí buď generovaný signál nebo nahraje zvuk. Přidá do blokového schématu filtry. Nastaví jejich parametry a přehraje výstup. Ten si pak také může stáhnout do počítače z uložených souborů.

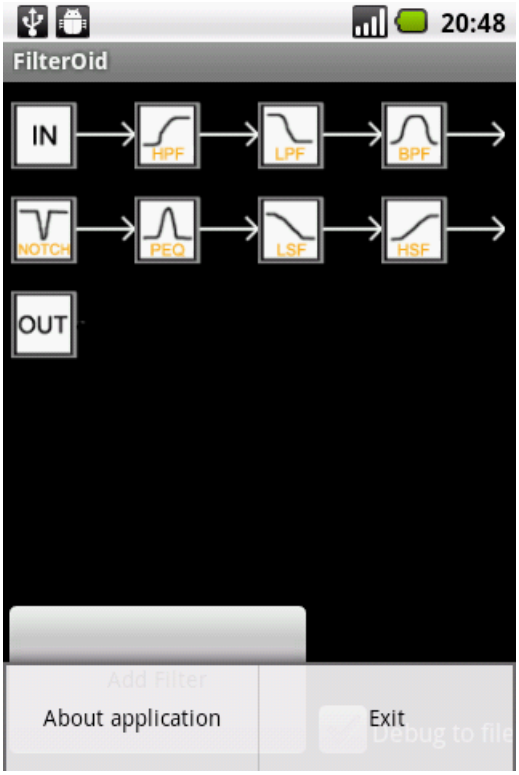
Obr. 2.3: Menu systému Android s ikonou aplikace FilterOid.



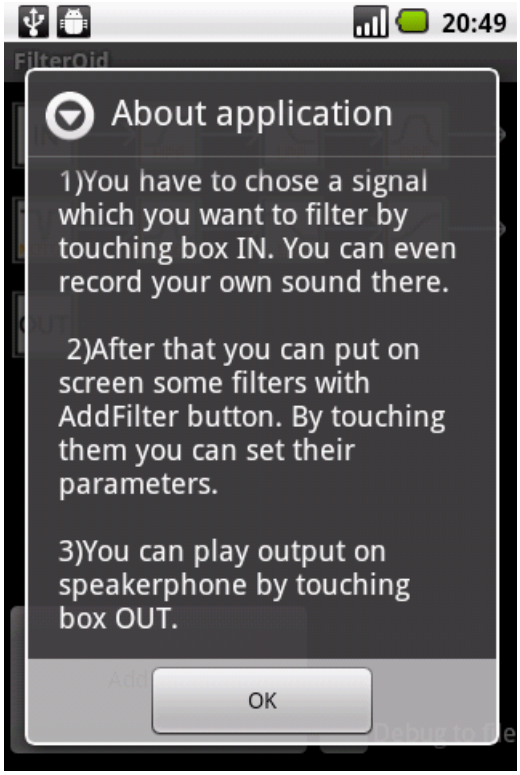
Obr. 2.4: Úvodní stránka aplikace.



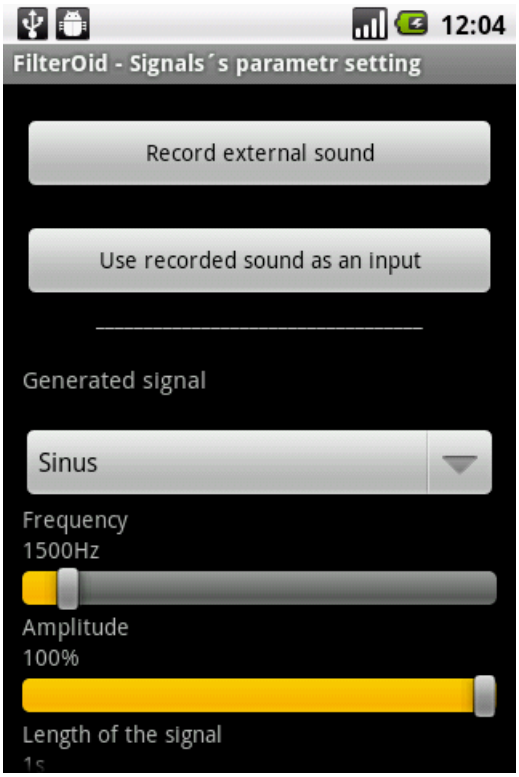
Obr. 2.5: Oznámení omezení filtrů.



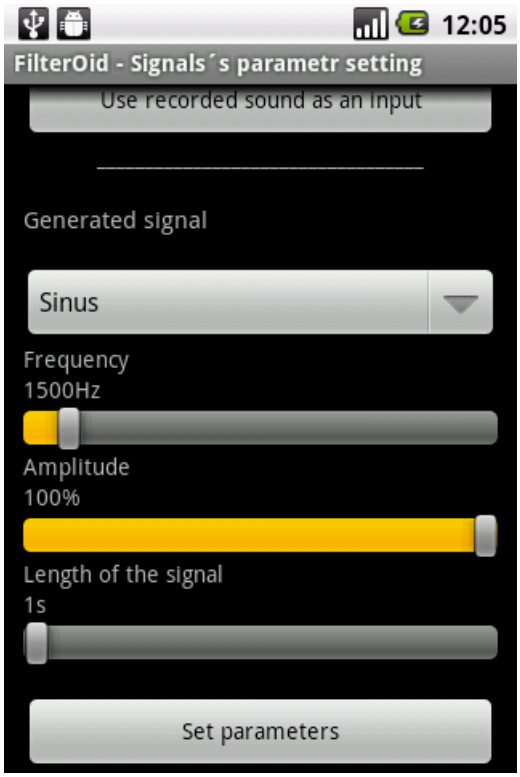
Obr. 2.6: Menu aplikace.



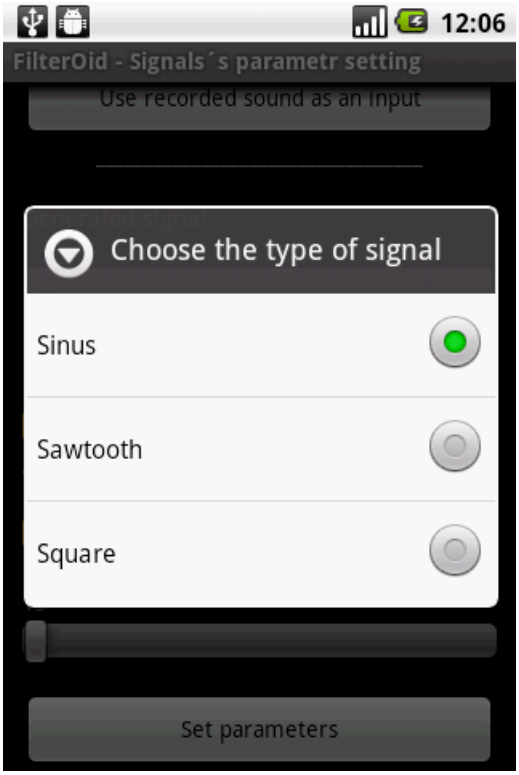
Obr. 2.7: Nápověda aplikace.



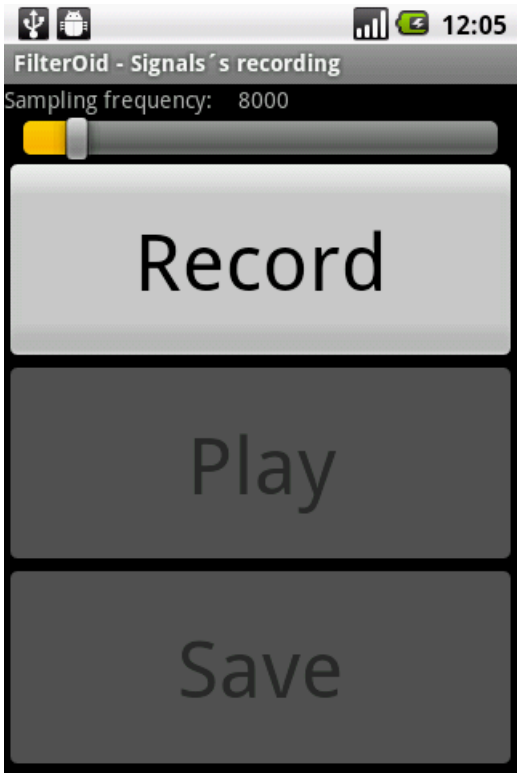
Obr. 2.8: Nastavení signálu 1.část.



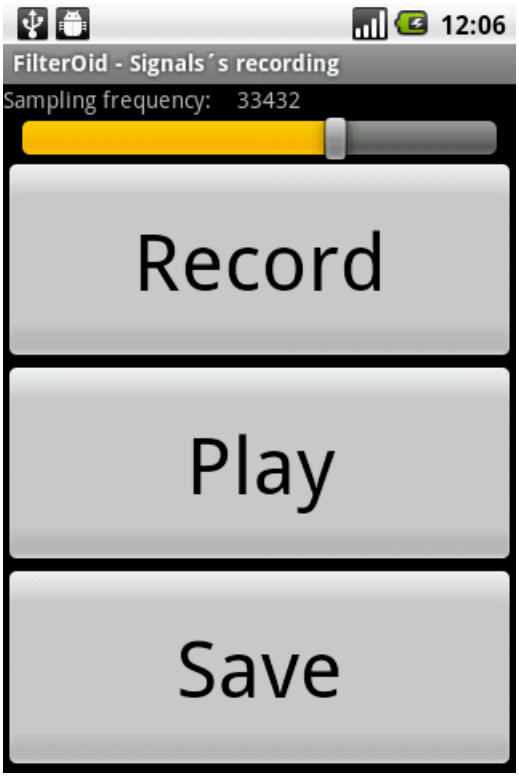
Obr. 2.9: Nastavení signálu 2.část.



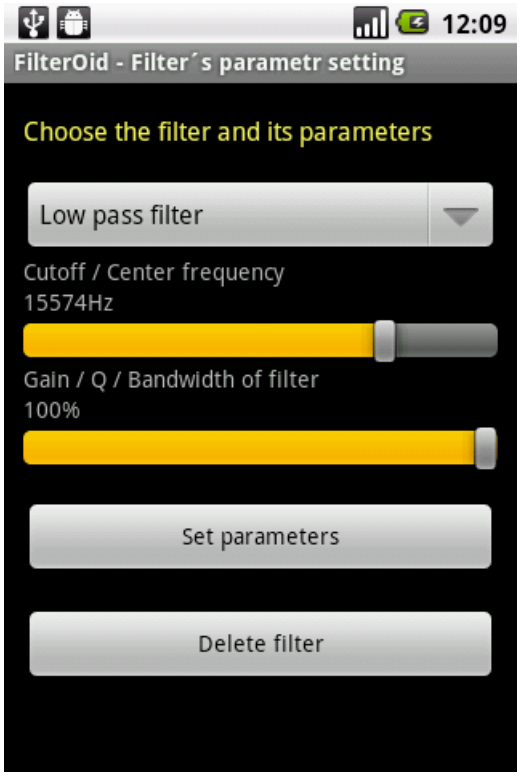
Obr. 2.10: Výběr typu signálu.



Obr. 2.11: Neaktivní tlačítka.



Obr. 2.12: Nahrávací dialog.



Obr. 2.13: Nastavení filtru.

3 ZÁVĚR

Cílem práce bylo prostudovat možnosti dostupných knihoven a balíčků v prostředí platformy Android. Dalším úkolem byla implementace algoritmů knihovny pro zpracování signálů. Celá aplikace měla tedy obsahovat knihovnu s uživatelským prostředím, které bude schopno knihovnu využívat. Knihovna také měla být snadno rozšiřitelná a využitelná samostatně pro výpočty jiných aplikací. Při jejím vývoji mělo být využito nástrojů pro automatické generování dokumentace a nástrojů pro verzování kódu.

Výsledná aplikace je po stažení do zařízení nebo nahrání do emulátoru připravena k použití. Po spuštění aplikace se spustí hlavní proces, který řídí další procesy celé aplikace. Hlavní proces tedy tvoří jakési uživatelské rozhraní mezi vlastní knihovnou a uživatelem. Pomocí různých pomocných tříd dochází k předzpracování zvoleného signálu tak, aby byl vhodně upraven. Po upravení formátu dat jsou volány vlastní knihovní funkce. Data jsou po upravení zpracována knihovními funkcemi. Obdržená výstupní data je nutno převést do vhodné podoby, které bude uživatel rozumět. Výsledkem je tedy systém, kterému uživatel určí, co se signálem má provést. Řídící proces toto provede a data uživateli zobrazí v odpovídajícím formátu. Celá aplikace byla testována na dvou zařízeních a emulátoru. Při testování byly změřeny hodnoty, které jsou komentovány v části Testování. V některých případech se podařilo změřené hodnoty analyzovat a kód přepsat. Tak bylo dosaženo lepších výsledků. Pro snadnější použití knihovny ostatními aplikacemi je možné její úplné vyřazení z projektu jako jeho součásti. Tak by získala knihovna svou úplnou samostatnost. Její následné importování do aplikace by si vyžádalo drobné změny kódu. Pro využití při tvorbě jiných aplikací, pro testování a pro práci se signály je však připravena k použití. Při tvorbě aplikace byl aktivně využíván nástroj pro automatické generování kódu. Využíván byl i systém SVN pro správu verzí kódu.

LITERATURA

[1] *Android developers* [online]. 2010 [cit. 2010-11-20]. Dostupné z WWW: <http://developer.android.com/>.

[2] *Android tutorial index* [online]. 2010 [cit. 2010-11-21]. Dostupné z WWW: <http://developerlife.com/tutorials/>.

[3] *Apache License, Version 2.0* [online]. 2010 [cit. 2010-11-27]. Dostupné z WWW: <http://www.apache.org/licenses/LICENSE-2.0>.

[4] MURPHY, M. *Beginning Android.* New York : Apress, 2009. 361 s. ISBN 978-1-4302-2420-4.

[5] ZÖLZER, U., et al. *DAFX - Digital Audio Effects.* West Sussex, UK : John Wiley & Sons, Ltd, 2002. 551 s. ISBN 0471490784.

[6] ZÖLZER, U. *Digital Audio Signal Processing.* 2nd edition. West Sussex, United Kingdom : John Wiley & Sons, Ltd, 2008. 317 s. ISBN 978-0-470-99785-7.

[7] TUDOR, B.; PETTEY, C. Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010; Smartphone Sales Increased 96 Percent. *Gartner Newsroom* [online]. 2010-11-10, [cit. 2010-11-27]. Dostupné z WWW: <http://www.gartner.com/it/page.jsp?id=1466313>.

[8] PETTEY, C; STEVENS, H. *Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012 – Gartner Newsroom* [online]. 2011-04-10, [cit. 2011-05-15]. Dostupné z WWW: <http://www.gartner.com/it/page.jsp?id=1622614>.

[9] SUCHAN, J. *Google Android - platforma pro mobilní telefony* [online]. 2009 [cit. 2010-11-19]. Dostupné z WWW: <http://www.minmax.cz/blog/google-android-tutorial>.

[10] JANOVSKÝ M. *Monitorovací systém aplikací a systému linuxových serverů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2008. 44 s., 2 s. příloh. Bakalářská práce. Vedoucí práce byl Ing. Filip Janovič.

[11] MEUER, H., et al. *TOP500 Supercomputer sites, 500 nejvýkonějších počítačů na světě* [online]. Prometeus GmbH, 2010 [cit. 2010-11-23]. Dostupné z WWW: <http://www.top500.org/>.

[12] WEISSTEIN, E. *"Sawtooth Wave." From MathWorld–A Wolfram Web Resource.* [online]. 2004 [cit. 2011-5-19]. Dostupné z WWW: <http://mathworld.wolfram.com/SawtoothWave.html>.

[13] FALSTAD, P. *Digital filters applet.* [online]. 2005 [cit. 2011-5-11]. Dostupné z WWW: <http://www.falstad.com/dfilter/directions.html>.

[14] BRISTOW-JOHNSON , R. *Cookbook formulae for audio EQ biquad filter coefficients.* [online]. 2003 [cit. 2011-2-09]. Dostupné z WWW: <http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt>.

[15] *Google I/O 2010.* [online]. 2010 [cit. 2010-12-09]. Dostupné z WWW: <http://www.google.com/events/io/2010/>.

[16] MICHÁLEK, J. *Android Development.* [online]. 2011 [cit. 2011-4-12]. Dostupné z WWW: <http://www.slideshare.net/georgiksk/vvoj-pre-google-android>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

Android Developer Challenge – je soutěž vývojářů pořádaná společností Google a dotovaná až 10 milióny dolarů(2008)

Android NDK – Android Native Development Kit – balíček do prostředí Eclipse pro vývoj aplikací v nativním kódu

Android SDK – Android Software Development Kit – balíček do prostředí Eclipse pro vývoj aplikací pod platformu Android

Apache License, Version 2.0 – zdrojové kódy softwaru pod touto licencí jsou volně šiřitelné, upravitelné a se zachováním určitých pravidel dále šiřitelné i pod jinou licencí

ARM – Advanced RISC Machine – je architektura procesorů vhodných pro mobilní zařízení, vyvinutá v Británii firmou ARM Limited

Cloud computing – model vývoje a využívání počítačových technologií založený na internetu, uživatel často platí za službu a né aplikaci jako takovou, politika plať jen co užíváš s anglického pay as you go

CVS – Concurrent Version System, open source platforma pro verzování digitálních dat

Dalvik VM – Dalvik Virtual Machine – obdoba Java VM v prostředí Android, přizpůsobuje kód vyšších vrstev spodním vrstvám

DSP – Digital Signal Processing – číslicové zpracování signálů

Eclipse – vývojové prostředí pro celou řadu programovacích jazyků

Git – distribuovaný systém pro správu verzí, původně pro vývoj jádra Linuxu

GNU – Gnu´s Not Unix – projekt zabývající se vývojem open source OS

GPL – General Public License – zdrojové kódy softwaru pod touto licencí jsou volně šiřitelné, upravitelné, ale dále distribuovatelné jen pod touto licencí

GPS – Global Positioning System – globální družicový polohový systém s jehož pomocí je možno určit polohu a čas kdekoliv na Zemi

OOP – objektově orientované programování

Open Handset Alliance – je sdružení telekomunikačních operátorů, výrobců mobilních telefonů a technologických firem, které stojí za vývojem platformy Android pro chytré mobilní telefony

OS – Operating System – Operační Systém

OSS – Open Source Software – software s veřejnými zdrojovými kódy

PCM – Pulse Code Modulation – formát digitální reprezentace analogových signálů

PID – Process Identifier – každý proces či úloha běžící v linuxovém OS má přiřazen vlastní identifikátor

PNG – grafický formát pro uchování obrazových dat

SCM – Software Configuration Management, je zastřešující název pro všechny techniky spojené s verzováním digitálních dat

SQL – Structured Query Language – strukturovaný dotazovací jazyk pro vytváření dotazů nad databází

SQLite – databázový dotazovací jazyk vycházející z SQL, nepotřebuje k běhu server

SVN – Subversion, open source platforma pro verzování digitálních dat

TOP500 – žebříček 500 nejvýkonnějších počítačů světa http://www.top500.org

XML – eXtensible Markup Language – rozšiřitelný značkovací jazyk, uživatel si může vytvářet vlastní tagy

aapt – Android Asset Packaging Tool – nástroj pro tvorbu aplikace s koncovkou (*.apt) spustitelné na platformě Android

binární formy software – předkompilovaný software

dx – nástroj upravující soubory jazyka Java na soubory s příponou (*.dex) spustitelné pomocí Dalvik VM

inkrementace – proces při kterém proměnná zvýší svou hodnotu o 1 krok

kernel – jádro operačního systému

super – identifikátor třídy rodiče v prostředí jazyka Java

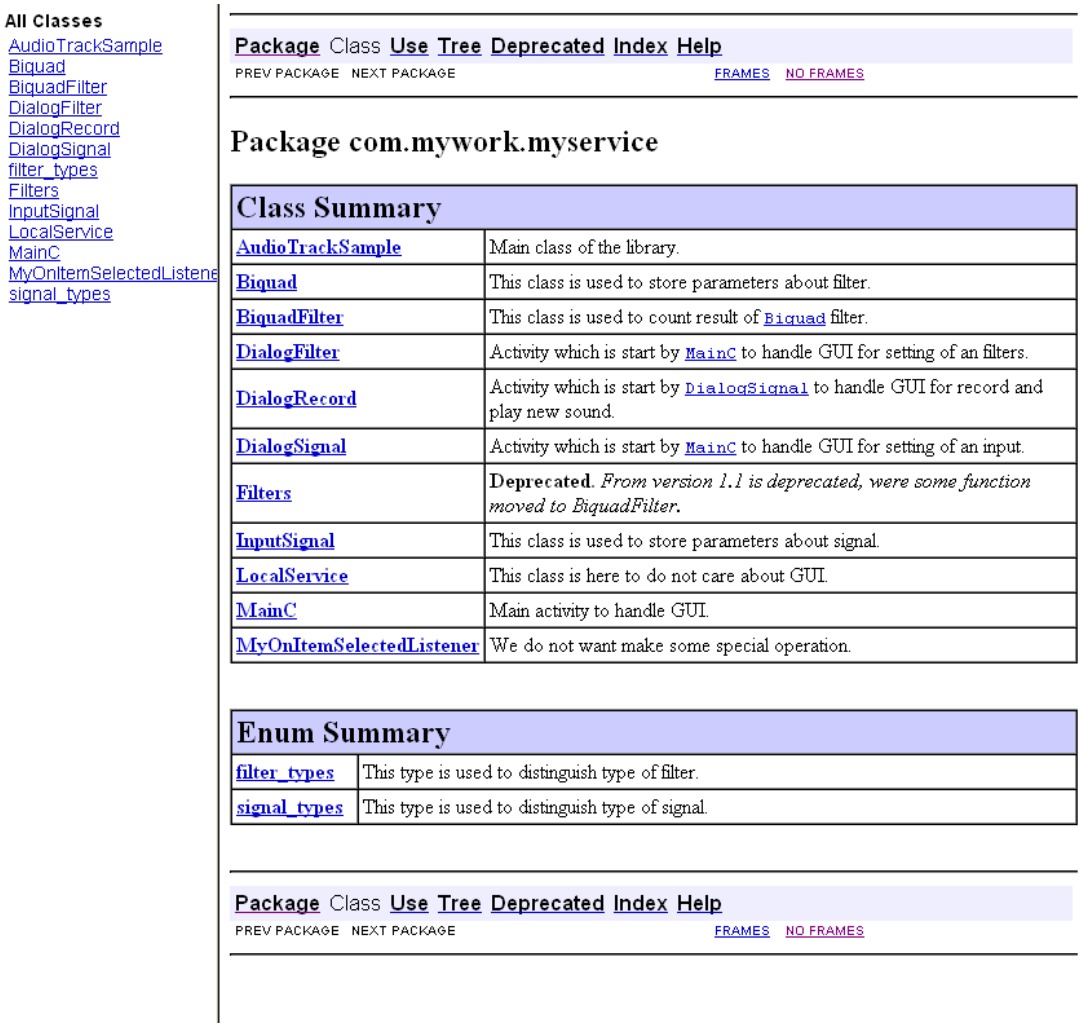
SEZNAM PŘÍLOH

A ELEKTRONICKÁ VERZE PRÁCE NA CD

CD s elektronickou verzí semestrálního projektu

- semstrální projekt

B DALŠÍ OBRÁZKY



All Classes
AudioTrackSample
Biquad
BiquadFilter
DialogFilter
DialogRecord
DialogSignal
filter_types
Filters
InputSignal
LocalService
MainC
MyOnItemSelectedListene
signal_types

Package Class Use Tree Deprecated Index Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

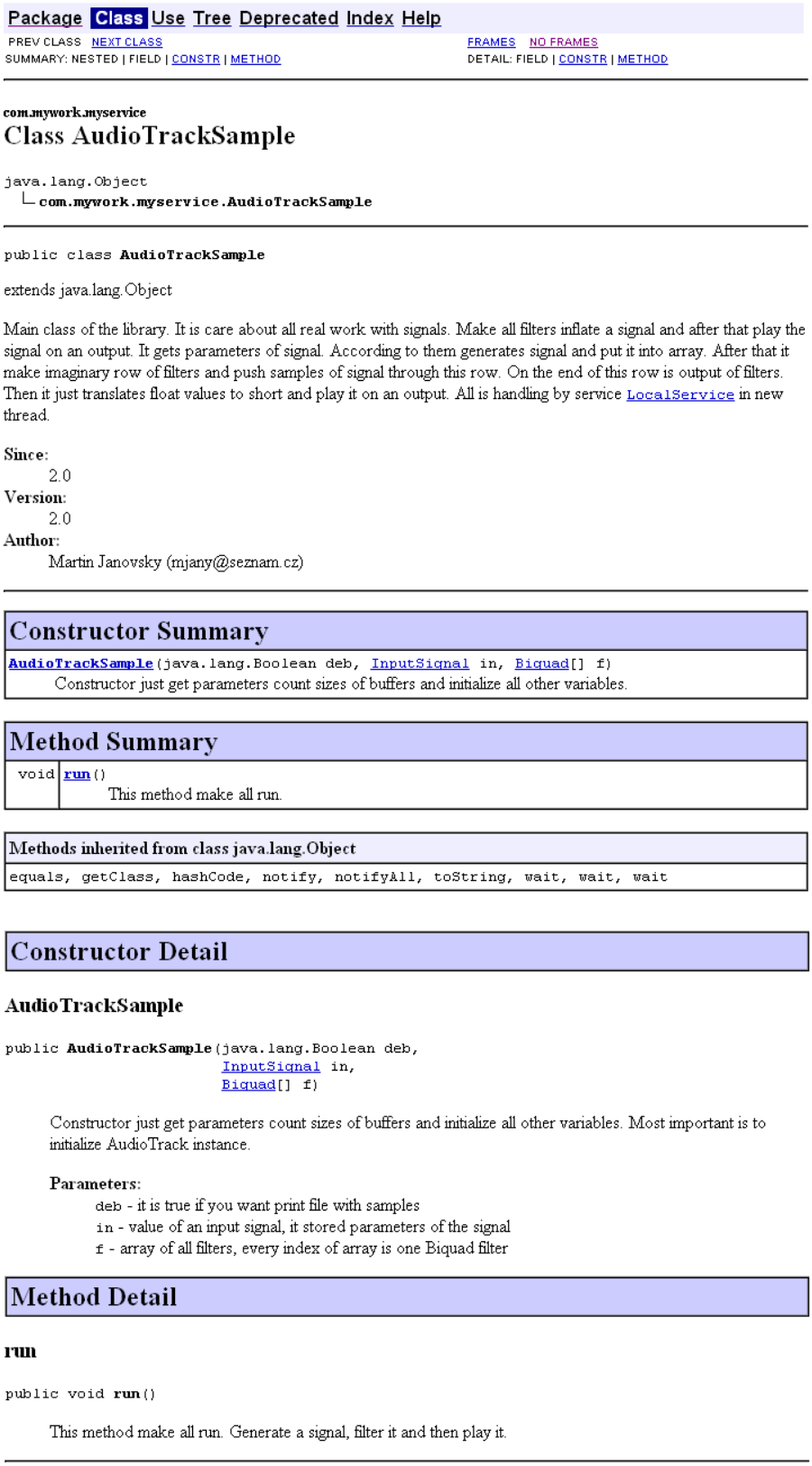
Package com.mywork.myservice

Class Summary	
AudioTrackSample	Main class of the library.
Biquad	This class is used to store parameters about filter.
BiquadFilter	This class is used to count result of Biquad filter.
DialogFilter	Activity which is start by MainC to handle GUI for setting of an filters.
DialogRecord	Activity which is start by DialogSignal to handle GUI for record and play new sound.
DialogSignal	Activity which is start by MainC to handle GUI for setting of an input.
Filters	**Deprecated.** *From version 1.1 is deprecated, were some function moved to BiquadFilter.*
InputSignal	This class is used to store parameters about signal.
LocalService	This class is here to do not care about GUI.
MainC	Main activity to handle GUI.
MyOnItemSelectedListener	We do not want make some special operation.

Enum Summary	
filter_types	This type is used to distinguish type of filter.
signal_types	This type is used to distinguish type of signal.

Package Class Use Tree Deprecated Index Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Obr. B.1: Náhled do indexu dokumentace.



Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.mywork.myservice

Class AudioTrackSample

java.lang.Object
└ com.mywork.myservice.AudioTrackSample

public class **AudioTrackSample**

extends java.lang.Object

Main class of the library. It is care about all real work with signals. Make all filters inflate a signal and after that play the signal on an output. It gets parameters of signal. According to them generates signal and put it into array. After that it make imaginary row of filters and push samples of signal through this row. On the end of this row is output of filters. Then it just translates float values to short and play it on an output. All is handling by service LocalService in new thread.

Since:
2.0
Version:
2.0
Author:
Martin Janovsky (mjany@seznam.cz)

Constructor Summary

AudioTrackSample(java.lang.Boolean deb, InputSignal in, Biquad[] f) Constructor just get parameters count sizes of buffers and initialize all other variables.

Method Summary

void	**run**() This method make all run.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AudioTrackSample

public **AudioTrackSample**(java.lang.Boolean deb,
InputSignal in,
Biquad[] f)

Constructor just get parameters count sizes of buffers and initialize all other variables. Most important is to initialize AudioTrack instance.

Parameters:
deb - it is true if you want print file with samples
in - value of an input signal, it stored parameters of the signal
f - array of all filters, every index of array is one Biquad filter

Method Detail

run

public void **run**()

This method make all run. Generate a signal, filter it and then play it.

Obr. B.2: Náhled do dokumentace třídy AudioTrackSample.

Obr. B.3: Náhled výpisu verze v prostředí systému Assembla.

Obr. B.4: Náhled výpisu verzí v prostředí Eclipse.

Obr. B.5: Graf vývoje verzí kódu v prostředí Eclipse.