Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Aleš Woska

M-lizer

Katedra spolehlivých a distribuovaných systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D.

Studijní program: Informatika

Studijní obor: Programování

Praha 2011

Chtěl bych poděkovat mému vedoucímu bakalářské práce a předmětu Softwarová praxe, RNDr. Janu Kofroňovi, Ph.D., za konzultaci, pomoc a vedení v průběhu tvorby práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne .....

podpis

Název práce: M-lizer

Autor: Aleš Woska

Katedra / Ústav: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D

Abstrakt: Tato bakalářská práce se zabývá problematikou vývoje programového zvukového syntetizéru nad platformou .NET. Cílem je vytvořit programový nástroj, který z MIDI dat vytvoří data, která mohou být přímo předaná zvukové kartě, nebo podobnému zařízení, a být přehrána na výstupním zvukovém zařízení s důrazem na věrohodnost zvukového výstupu. Vytvořený zvuk by se tedy měl co nejvíce podobat zvuku skutečných nástrojů. Dalším cílem je vytvořit návod, jak psát programový zvukový syntetizér podporující technologii VST.

Klíčová slova: VST, .NET, SoundFont, zvuková syntéza, instrument, syntetizér

Title: M-lizer

Author: Aleš Woska

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Jan Kofroň, Ph.D

Abstract: The thesis deals with the development of a software sound synthesizer of the .NET platform. The goal is to create a software tool which creates data, that are passed to a sound card or a similar device and that can by played on an output sound device, from MIDI data. Emphasis is placed on a authenticity of a sound output. Created sound would therefore be most resemble the sound of real instruments. Another goal is to create a guide, on how to write a software sound synthesizer that supports VST technology.

Keywords: VST, .NET, SoundFont, sound synthesis, instrument, synthesizer

Obsah

1. Úvod

Chtěl jsem vytvořit nástroj, který na základě vstupních MIDI dat bude generovat výstup, po jehož přehrání a poslechu bude pozorovatel nabývat dojem, že vygenerovaný zvuk zní stejně, jako zvuk skutečného hudebního nástroje. V současné době je k dispozici mnoho zvukových syntetizérů. Chtěl jsem navrhnout takový nástroj, aby výsledný zvuk zněl co nejvěrohodněji a zároveň byl co nejpoužitelnější a lišil se od ostatních programových zvukových syntetizérů.

Při návrhu jsem procházel několika úrovněmi problémů. Jako první přichází na řadu způsob, jakým obstarat vstup. MIDI data může program získávat buďto z nástroje připojeného k počítači nebo z MIDI souboru uloženého na paměťovém médiu počítače. Program musí mít prostředky, aby měly přístup k oběma typům vstupu.

Další problém k rozmyšlení je způsob, jak ze vstupu vytvořit data s požadovaným výsledkem. Program musí obsahovat obecný algoritmus, který řeší převod syntaktických MIDI dat na sémantická data výstupu, které nemají žádnou určitou strukturu, ale po přehrání ve zvukové kartě vznikne zvuk s požadovanými vlastnostmi. Tento proces se nazývá zvuková syntéza a program musí takovouto syntézu implementovat.

Nakonec, po vytvoření dat program musí tato data zpracovat. Data musí být programem buďto přenesena do zvukové karty, aby pak mohla být přehrána na zvukovém výstupu počítače, nebo být uložena do zvukového souboru. Program musí distribuovat prostředky pro komunikaci se zvukovým výstupem a práci se zvukovými soubory a jejich formáty.

Tato práce bude rozebírat všechny problémy, které je třeba řešit při vývoji, bude popisovat a představovat technologie potřebné pro správnou a efektivní funkčnost programu. Pro práci syntetizéru je klíčová zvuková syntéza. V práci bude vybraná a popsaná jedna metoda. Budou uvedeny porovnání vůči ostatním metodám syntézy zvuku, její výhody a nevýhody. Bude vysvětlena struktura samotného programu, vysvětlen a odůvodněn způsob řešení. Bude vysvětlen způsob využití technologií VST, MIDI, ASIO a Soundfont v práci pro efektivní a snadnější řešení problému, propojení těchto technologií do jednoho celku.

2. Analýza problému

Psaní aplikace takového typu je technologicky náročné, protože proces tvoření zvuku prochází přes různá stádia několika rozdílnými technologiemi a pro požadovaný výsledek je nutné ovládat každou technologii, přičemž tyto technologie jsou specifické a s každou se pracuje jiným způsobem. Program se proto bude skládat z několika vzájemně komunikujících částí. Každá část bude řešit rozdílnou technologii nutnou pro funkčnost celku. Při řešení problému je třeba se zaměřit na různé aspekty úlohy. Nejprve je nutné zvolit vhodnou koncepci programu. Psaní projektu jako samostatného programu má značné nevýhody. Uživatel takového programu ho musí nainstalovat. Když uživatel pracuje se zvukem, bude již pracovat se svými oblíbenými programy, navíc samotná aplikace by byla poměrně slabá. Na druhou stranu, koncepce v podobě pluginu řeší oba problémy. Zaprvé, uživatel nemusí instalovat další aplikaci, pouze použije knihovnu pluginu a nahraje si jí do své oblíbené audio aplikace. K pluginu pak bude mít přístup z této aplikace. Zadruhé, hostující aplikace bude poskytovat a nabízet vlastní prostředky a nástroje (např. nahrávání a ukládání souborů, přístup ke zvukovým efektům a kanálům, …), takže funkcionalita a použitelnost pluginu tím vzroste.

Rozhodl jsem se pro psaní pluginu, kvůli popsaným výhodám. Pro plugin je důležité, aby ho bylo schopno rozeznat a nahrát co nejvíce aplikací. Plugin by tedy měl implementovat určité rozhraní, s kterým bude schopno pracovat co nejvíce hostujících zvukových aplikací. Tento popsaný problém řeší několik technologií: Audio Units od firmy Apple, DirectX plugin od firmy Microsoft, VST od firmy Steinberg, a další. Zmíněné technologie se používají ve zvukovém průmyslu ve studiích, audio stanicích a podobně. Umožňují zvýšit funkcionalitu různých zařízení a programu za pomocí používání pluginů podporujících danou technologii. Rozhodl jsem se pro poslední zmíněnou technologii - VST, protože je ze všech zmíněných technologií nejpoužívanější. Využívá ho většina profesionálních aplikacích (1) a je k dispozici i mnoho bezplatných (freeware) programů (Audacity, Wavosaur, VSTHost, …). I to je dostačující důvod, VST má oproti podobné zmíněné technologii (DirectX plugin) navíc několik dalších výhod. VST je přenositelnější mezi platformami, protože DirectX plugin vyvíjí přímo firma Microsoft. Kód psaný pro VST je jednodušší, protože DirectX má složitou strukturu a vyžaduje množství kódu navíc. Mnoho profesionálních VST aplikací dokáže vygenerovat uživatelské rozhraní na

základě kódu. Tato funkce je ale pro tuto práci nedůležitá, protože uživatelské rozhraní jsem tvořil sám. Dalším důvodem pro tento výběr je, že technologie VST obsahuje přehledné a stručné programátorské rozhraní (VST SDK), pomocí kterého je tvorba pluginu snadnější. Mimo tyto drobné rozdíly mezi technologiemi VST a DirextX plugin není výrazný funkční rozdíl.

Pro psaní pluginu jsem si vybral jazyk C# a platformu .NET, protože tato platforma je moderní, perspektivní a umožňuje psaní bezpečného managed kódu. Problém s VST ale je, že jeho funkce a rozhraní jsou psány v jazyce C++. Pro psaní pluginu v jazyce C++ je k dispozici Software development kit (SDK), který usnadňuje vývoj pluginů VST. Technologii VST a platformu .NET kloubí dohromady projekt s názvem VST .NET. Umožňuje efektivní psaní pluginu v libovolném programovacím jazyce podporující platformu .NET. Alternativou k VST .NET je projekt s názvem Naudio, který jsem nepoužil, protože na rozdíl od VST .NET neimplementuje třídy a metody pro práci s VST. Nabízí se ještě možnost využití možností jazyka C++/CLI pro volání nativních funkcí VST SDK API přímo v C++. Tuto možnost jsem zamítl, protože bych psal kód, který je už implementovaný a optimalizovaný tvůrci VST .NET. Využitím existujících knihoven jsem získal více času a prostoru pro psaní samotné funkčnosti programu a zaměřením se na zajímavější částí problému, které samotné rozhraní VST neřeší.

Výstup pluginu bude posílán do zvukové karty a ta jej bude přehrávat pomocí vhodného výstupního zvukového zařízení. Výstup představuje pole hodnot reprezentující diskrétní průběh zvukového signálu. Vhodné metody tyto hodnoty nahrají do bufferu zvukové karty a z tohoto bufferu je zvuková karta pomocí A-D převodníku převede na spojitý zvukový signál. Plugin musí řešit pouze problém nahrání dat do bufferu zvukové karty. Proto potřebuje rozhraní, přes které bude komunikovat se zvukovou kartou a bude mít přístup k jejím zdrojům. Řešení nabízí protokol Audio Stream Input/Output (ASIO), který umožňuje přístup ke zvukové kartě s krátkou odezvou a přijatelným rozhraním. Nativní funkce systému totiž nezajišťují krátkou odezvu při nahrávání dat do bufferu, a krátká odezva je při práci se zvukem nezbytná.

Dále je nutné si rozmyslet způsob generování výstupu pluginu. Generování zvuku řeší oblast zvukové syntézy. Existuje mnoho typů zvukových syntéz (7). Rozhodl jsem se pro syntézu pomocí tzv. wavetable (waveform synthesis). Tento typ

syntézy potřebuje ke své práci již vytvořené zvukové vzorky (zvukové tabulky, wavetables). Pro jejich uložení jsem si vybral souborový formát Sound Font. Tento formát obsahuje samotná data zvukové tabulky a navíc informace důležité pro její zpracování[1].

[1] název nástroje, který data napodobují, frekvence signálu, velikost datového toku, …

3. Použité technologie

3.1. Virtual Studio Technology

Virtual Studio Technology (VST) vyvinuto firmou Steinberg GmbH je rozhraní pro integrování softwarových syntetizérů, zvukových efektů a nahrávacích systémů. VST a podobné systémy využívají digitální zpracování zvukového signálu pro simulování tradičního vybavení nahrávacích studií. VST je podporována většinou profesionálních zvukových aplikací (např. Cubase, Cakewalk, Adobe Premiere, …).

VST je platformně nezávislé. Nativní platforma pro VST je Windows, ale pluginy VST dokáží pracovat i pod platformou UNIX za použití software pro toto použití (viz http://quicktoots.linuxaudio.org/toots/vst-plugins). Podobně to je i s platformou OS X, kde pluginy VST dokáží pracovat (mimo jiné) za použití software „VST to AudioUnit Adapter“ (viz http://www.fxpansion.com/index.php?page=5&tab=22).

Programátor zpravidla s použitím technologie VST napíše plugin, který uživatel nahraje do hostující audio aplikace, která tuto technologii podporuje. VST pluginy dělíme na nástroje (instrumenty) a efekty. Nástroj je softwarová emulace některých existujících nástrojů (syntetizérů a samplerů), označují se jako VSTi. Primární funkcí nástroje je generování zvuku. Například, jako vstup plugin načítá MIDI data (získaná ze souboru, z nástroje připojeného k počítači, …) a svou činností podle něj generuje výstup reprezentující zvukový signál. Efekty na druhou stranu upravují již existující zvuková data a upravená je distribuují dále, třeba dalším efektům. Příklad efektu je ozvěna, wah-efekt nebo distorze.

Koncepci programu jsem tedy navrhl jako plugin nad technologií VST. Bude tak kompatibilní s aplikacemi i dalšími pluginy podporujícími VST. Díky tomu odpadá nutnost psát některé funkční části programu. Například není nutné řešit MIDI vstup. Obstará ho hostující aplikace, a to buď ze souboru nebo přímo ze vstupu reálného MIDI nástroje připojeného k počítači. Plugin zažádá přes rozhraní VST o data a hostující aplikace je poskytne (nebo opačně). Vše je tedy řešené na úrovní volání funkcí mezi pluginem a hostující aplikací. Dále plugin nebude muset řešit ukládání výstupních dat do souboru, to také zajistí hostující aplikace.

3.2. VST .NET

VST .NET je projekt umožňující psaní pluginu VST v libovolném .NET jazyce a vyplňuje tak mezeru mezi nativním jazykem VST, čímž je C++. Nabízí řadu rozhraní, pomocí kterých můžeme psát přehledný kód, bez podrobných znalostí standardů VST, které jsou zajištěné v rozhraních nabízející VST .NET. K dispozici jsou rozhraní na dvou úrovních, a to na úrovni jádra a na úrovni frameworku. Využití rozhraní na úrovni jádra umožňuje volání řízené (managed) verze nativních metod VST. Díky tomu bude mít kód minimální režii navíc, ale zato musí programátor podrobně znát standardy VST. Používání rozhraní na úrovni frameworku bude program stát režii navíc, zato můžeme využívat definované metody, datové struktury, které stačí předefinovat nebo lehce upravit pro správnou funkčnost. Současná verze je 0.9 a dále se pracuje na vývoji.

Umožní stavět od již existujícího základu a dodržovat tak jisté standardy pro práci s VST. Program pak bude snadno rozšiřitelný. VST .NET může v nových verzích nabízet další nebo efektivnější funkce, které se snadno naimplementují do již existujícího kódu. VST .NET se do kódu zavede připojením několika knihoven, z kterých se budou využívat její rozhraní, třídy a funkce.

Využití VST .NET má navíc několik výhod. Je snadnější na implementaci, než práce s nativním VST SDK API[2]. K správné a korektní implementaci pluginu přes VST .NET stačí implementovat předepsaná rozhraní. VST .NET také za programátora doplní nutnou výplň kódu, která v programu nemá funkční účel, ale je třeba pro identifikaci a kategorizaci[3]. V nativním API je programátor musí psát sám. Dále se může psát v libovolném .NET jazyce. VST .NET nabízí robustní základ, z kterého se dá vycházet. Existuje k němu výborná dokumentace a příklady použití.

3.3. MIDI

„Musical Instrument Digital Interface (MIDI) je mezinárodní standard používaný v hudebním průmyslu jako elektronický komunikační protokol, který dovoluje hudebním nástrojům, počítačům i dalším přístrojům komunikovat v reálném

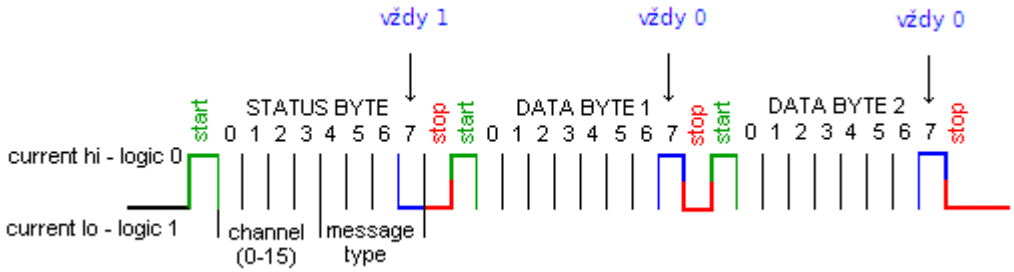
[2] Rozhraní pro práci s VST přes nativní funkce a knihovny psané v C++

[3] Je nutné například implementovat všechny třídy, které může přes VST volat hostující aplikace, i když tyto funkce plugin vůbec v průběhu svého běhu nevolá.

čase prostřednictvím definovaného sériového rozhraní. Standard spravuje organizace MIDI Manufacturers Association.“ (1)

3.3.1. Přenosový protokol

„U MIDI je použit jednoduchý asynchronní sériový přenos dat (viz RS-232C) s využitím výše popsané proudové smyčky. Data jsou rozdělena do bajtů (osmice bitů, nejdříve se přenáší bit s nejmenší váhou) doplněných o start bit a stop bit. Start bit má vždy logickou hodnotu 0, stop bit logickou hodnotu 1. Hodinová frekvence vysílače je rovna 31250 Hz, což znamená, že se celý bajt i se start bitem a stop bitem přenese za 320 µs. Jedná se o poměrně netypickou hodnotu, mnoho ostatních sériových rozhraní se drží spíše celočíselných násobků 75 Hz. Z přenášených bajtů jsou skládané MIDI zprávy. Jejich délka je rovna jednomu, dvěma či třem bajtům, ovšem jeden speciální typ zprávy má neomezenou délku. První bajt zprávy se nazývá stavový bajt (status byte). Jeho zvláštností je to, že má nastavený nejvyšší bit na logickou hodnotu 1 (jeho dekadická hodnota je vždy větší než 127). Všechny ostatní bajty mají tento bit nulový (jejich hodnota je menší než 128), takže i v případě, že MIDI zařízení ztratí synchronizaci (vytažení konektoru apod.), může poměrně jednoduše detekovat začátek další zprávy.“ (2)



Obrázek 1: *Přenos MIDI zprávy o délce tří bajtů. Tři bajty je obvyklá délka většiny MIDI zpráv, přenesení takto dlouhé zprávy trvá necelou jednu milisekundu – 3×320 µs. (2)*

3.3.2. Typ MIDI zpráv

V každé MIDI zprávě její první bajt určuje typ zprávy a číslo kanálu, pro který je příkaz určen[4]. Ve zbývajících bajtech zprávy jsou přenášeny parametry tohoto příkazu. Následující tabulka uvádí kódy a významy příkazů, které se mohou vyskytovat v MIDI zprávě.

[4] Až na kód příkazu pro nehudební příkazy (s kódem příkazu 0xf0), který v prvním bajtu neobsahuje zakódované číslo kanálu.

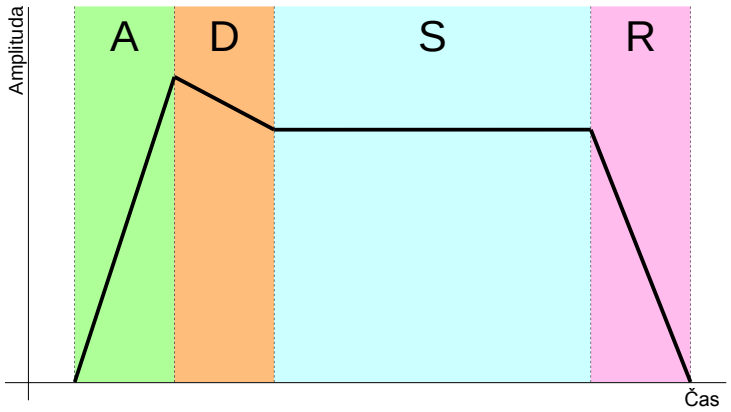
Kód příkazu (hexadecimálně)	**Název/význam**
0×80	Note Off
0×90	Note On
0×a0	Aftertouch
0×b0	Continuous controller
0×c0	Patch change
0×d0	Channel Pressure
0×e0	Pitch bend
0×f0	(non-musical commands)

Tabulka 1: Tabulka kódů a významů MIDI zpráv.

3.3.3. Obálka ADSR

Obálka ADSR popisuje závislost amplitudy generovaného signálu na čase (průběh hlasitosti). Zkratka ADSR je odvozena od názvů čtyř částí průběhu signálu: attack (náběh), decay (pokles), sustain (ustálená hodnota), release (doznění).

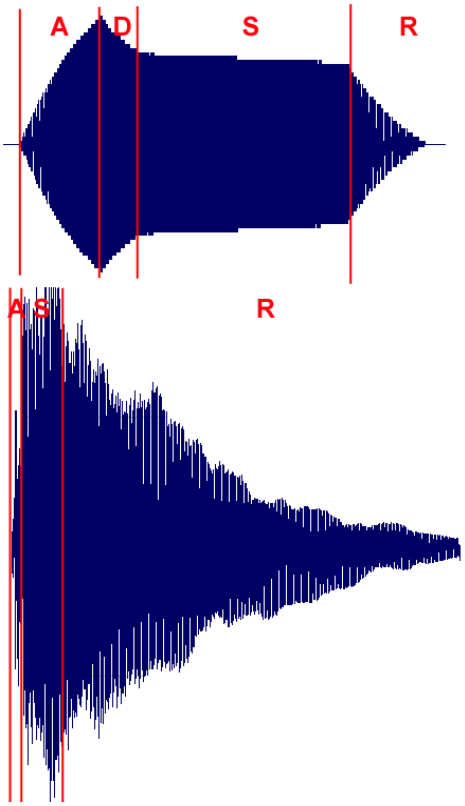
V okamžiku stisku klávesy (tím je vygenerovaná Note On MIDI zpráva, která je předána ke zpracování) nastává část A. Po této události signál zvětšuje svou amplitudu až na definovanou hodnotu, poté v části D se amplituda ustálí na hodnotu definovanou ve zprávě Note On, tím se přejde do části S. Po uvolnění klávesy (a tím vygenerování MIDI zprávy Note Off) se přejde do části R a signál začne zmenšovat svou amplitudu až na nulovou hodnotu.



Obrázek 2: Obálka ADSR signálu představujícího přehrávanou notu. Existují i jiné možnosti specifikace obálky, tato je však nejznámější, protože je použita ve velkém množství hudebních syntetizérů.

Pro lepší pochopení obálky ADSR poslouží následující obrázek, který představuje obálky ADSR dvou nástrojů. Horní část obrázku představuje roh, který

má poměrně dlouhou část S (sustain). Druhá část obrázku představuje klavír, který má naopak část S velice krátkou a téměř po celou dobu znění nástroje se signál pohybuje v části R (release). Z obálky lze tedy vyčíst povaha nástroje, roh zní dlouho dobu monotónně, kromě náběhu. Klavír zní nejsilněji při úderu kladívka a jeho hlasitost poté pouze slábne.



Obrázek 3: Obálka ADSR pro roh a klavír.

3.3.4. Přehled různých MIDI zpráv

Asi nejčastěji posílaná MIDI zpráva je Note On. Reprezentuje stisknutou klávesu noty na nástroji. Parametry této zprávy jsou číslo stisknuté noty ve druhém bajtu zprávy a rychlost přehrání začátku noty (attack) ve třetím bajtu zprávy.

Zpráva Note Off je opačná zpráva k předchozí. Význam parametrů je obdobný, tedy ve druhém bajtu zprávy je uloženo číslo noty a ve třetím bajtu zprávy je uložena doba přehrávání konce noty (release). Některé nástroje místo zprávy Note Off posílají zprávu Note On s nulovou délkou, a specifikace říká, že MIDI musí korektně rozpoznávat a reagovat na oba případy zpráv.

Zpráva After Touch reprezentuje sílu stisknutí přehrávané noty. Ne všechny nástroje toto podporují, protože je potřeba zvláštní senzory, které tento tlak dokáží

měřit. V obálce ADSR tato zpráva ovlivňuje amplitudu signálu části sustain. Ve druhém bajtu zprávy je uloženo číslo noty a ve třetím bajtu zprávy je uložena síla stisku noty z rozsahu 0 - 128. Některé nástroje nedokáží rozlišit všechny úrovně hlasitosti, v tomto případě musí úrovně odpovídat hodnotám násobku osmi.

Zpráva Pitch Wheel reprezentuje změnu výšky některého z tónu ve stupnici pomocí speciálního ovládacího prvku. Ve druhém a třetím bajtu zprávy je uložena čtrnáctibitová hodnota, která reprezentuje posun tónu ve stupnici o *k*(n-8192)* centů. Jeden cent odpovídá $\frac{1}{100}$ temperovaného půltónu, neboli $\frac{1}{1200}$ oktávy, kde n je přenesená hodnota, k je konstanta závislá na použitém přístroji a 8192 je zakódovaná střední hodnota polohy ovládacího prvku. Záporný výsledek výpočtu znamená posun tónu směrem dolů, kladný výsledek směrem nahoru.

3.3.5. Převod mezi výškou tónu a odpovídajícím kódem noty

Mnoho typů zpráv se odkazuje na číslo noty nebo tónu (angl. MIDI number nebo MIDI key). Každá klávesa nástroje odpovídá nějakému číslu z rozsahu 0 - 127. Tento rozsah je určený velikosti části zprávy určeného pro číslo noty - jeden bajt, kde nejvyšší bit u tohoto bajtu MIDI zprávy je vždy nulový. Do toho bajtu se dá zakódovat $2^7 = 128$ různých hodnot. Základem je tón komorní a o frekvenci 440 Hz, který odpovídá číslu tónu 69. Přechod o jednotku čísla tónu znamená změnu frekvence tónu o $2^{\frac{1}{12}}$ násobek původní frekvence. Tato konstanta byla získaná poměrem frekvenci dvou tónů ve stupnici rozdělené na 12 půltónů.

3.3.6. Využití v práci

Základem práce je zpracování MIDI vstupu, podle kterého bude program generovat výstup. Rozepsal jsem proto podrobně princip fungování MIDI, včetně vypsání důležitých typů zpráv, které bude můj program implementovat. Pro správné dekódování zprávy je potřeba znát datovou strukturu přenosu po protokolu MIDI. Typ zpráv a její kód je důležitý pro její identifikaci v programu. Princip obálky ADSR je důležitý pro realističnost generovaných dat. Popis zmíněných zpráv je důležitý pro vysvětlení jejich vlivu v programu na výsledná výstupní data.

3.4. Audio Stream Input/Output

Audio Stream Input/Output (zkratka ASIO) je platformně nezávislý vícekanálový protokol pro ovladače zvukových karet vyvinutý firmou Steinberg GmbH. Zajišťuje velmi nízkou odezvu a věrné rozhraní mezi softwarovou aplikací a zvukovou kartu. Umožňuje tedy přímý přístup ke zdrojům zvukové karty. Díky těmto vlastnostem jsou ovladače ASIO ideální pro použití s VST nástroji a pluginy. Nízká odezva je nutná pro přehrávání zvuku v reálném čase. Jestliže je odezva mezi příchozí MIDI událostí a přehrání zpracovaných dat velká, je to v hudební praxi naprosto nepoužitelné. Je proto nutné generovaná data předat bufferu zvukové karty co nejrychleji.

Pro svou funkci plugin vyžaduje instalaci ovladačů ASIO. Existují různé knihovny pro práci přímo s ASIO, nebudou ale třeba. Je pouze nutné zvolit ovladače ASIO v nastavení výstupních zařízení hostující aplikace. Vygenerovaná data jsou pak předána přes VST .NET aplikaci, která je přes ASIO rozhraní předá zvukové kartě.

Původní ovladače ASIO jsou určené pro výkonné profesionální zvukové karty a jejích používání je zpoplatněno. Pro běžné zvukové karty, a také pro zvukové čipy integrované na základních deskách jsou k dispozici nezávislé univerzální ovladače ASIO. Nejpoužívanější se nazývá ASIO4ALL, je volně ke stažení a jeho používání je bezplatné.

3.5. SoundFont

SoundFont je obchodní značka vztahující se k souborovému formátu a související technologií vytvořené pro vyplnění mezery mezi nahrávaným a syntetizovaným zvukem. SoundFont obsahuje zvukové vzorky a informace pro jejich organizaci a zpracování. Podle těchto informací může nástroj, obvykle řízený MIDI daty, generovat tóny požadovaných barev, výšek, hlasitostí a dalších parametrů. (4)

3.5.1. Struktura souborového formátu SoundFont

Souborový formát SoundFont je organizovaný do formátu nazývaný RIFF. RIFF je zkratka pro Resource Interchange File Format, je to souborový formát firmy Microsoft sloužící pro ukládání multimediálních zvukových a obrazových předloh. Formát RIFF definuje strukturu uložení dat do souboru pro různé typy a formáty dat

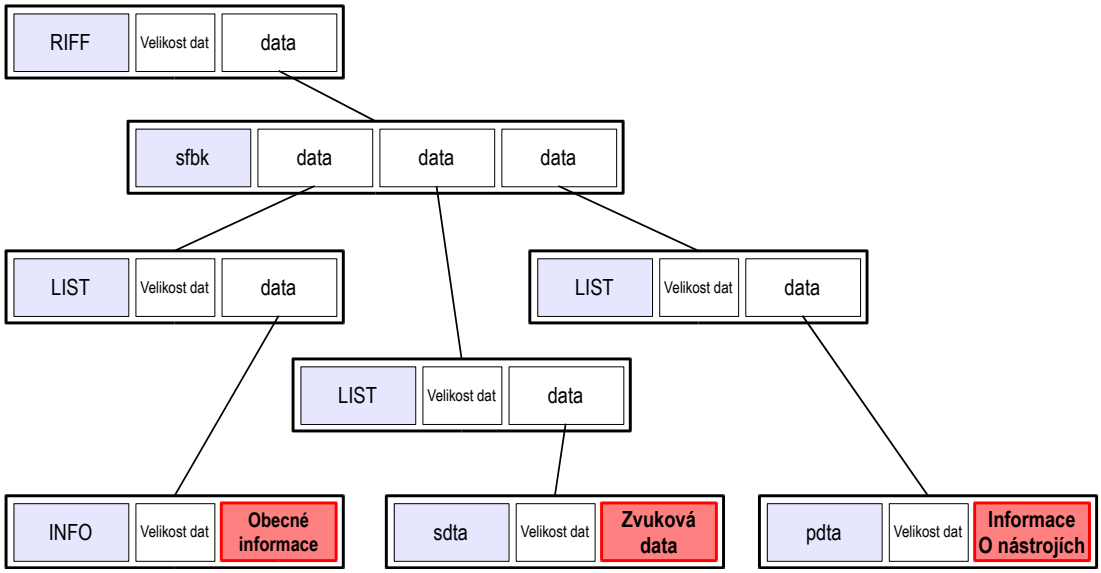
(multimediální kontejner). RIFF se skládá z datových struktur zvaných shluky (angl. chunk), každý shluk má svoji čtyřznakovou signaturu (ID) definovanou v hlavičce shluku. Za shlukem následují data do velikosti danou v hlavičce shluku. Shluk může obsahovat tzv. podshluk (subchunk). Každý RIFF soubor začíná shlukem se signaturou RIFF, dále soubor obsahuje jeden nebo několik shluků se signaturou LIST, které obsahují dodatečný identifikátor formátu dat následujících dat v souboru. Existuje ještě shluk se signaturou JUNK, používaný jako výplň dat, pro zarovnání dat na velikost čteného bloku. Data jsou v souborech řazený v pořadí little endian. Existuje i varianta RIFF ve tvaru big endian s příponou souboru RIFX. (5)

Název bloku	Velikost bloku
ChunkID	4 B
dataSize	4 B
data	dataSize B

Tabulka 2: Struktura shluku (chunku) používaná ve formátu RIFF

Celý souborový formát je tedy členěn do určitých bloků. Ze specifikace formátu RIFF musí každý takový soubor začínat se shlukem se signaturou „RIFF“. V případě souborového formátu SoundFont se datech tohoto shluku nachází podshluk se signaturou „sfbk“, který je jednoznačný identifikátor tohoto formátu. Tento shluk obsahuje tři další podshluky se signaturou „LIST“. První tento podshluk obsahuje obecné informace o souboru[5]. V jeho datech je další podshluk a jeho signatura je „INFO“. Druhý podshluk se signaturou „LIST“ jako svá data obsahuje podshluk se signaturou „sdta“, který reprezentuje surová zvuková data. Tato data jsou uložena jako jeden kus a nemají svou vlastní strukturu. Význam těchto dat určuje až třetí podshluk se signaturou „LIST“. Ten ve svých datech obsahuje podshluk se signaturou „pdta“. V tomto posledním podshluku je uložen seznam zón, generátorů, modulátorů a nástrojů. Zmíněné seznamy se odkazují na data ve druhém podshluku pomocí rozsahu bajtů, na kterém místě v těchto datech se nacházejí požadované informace. Celou situaci nejlépe vyjádří následující přehledný obrázek.

[5] verze souboru, jméno SoundFontu, datum vytvoření, komentáře, atd.



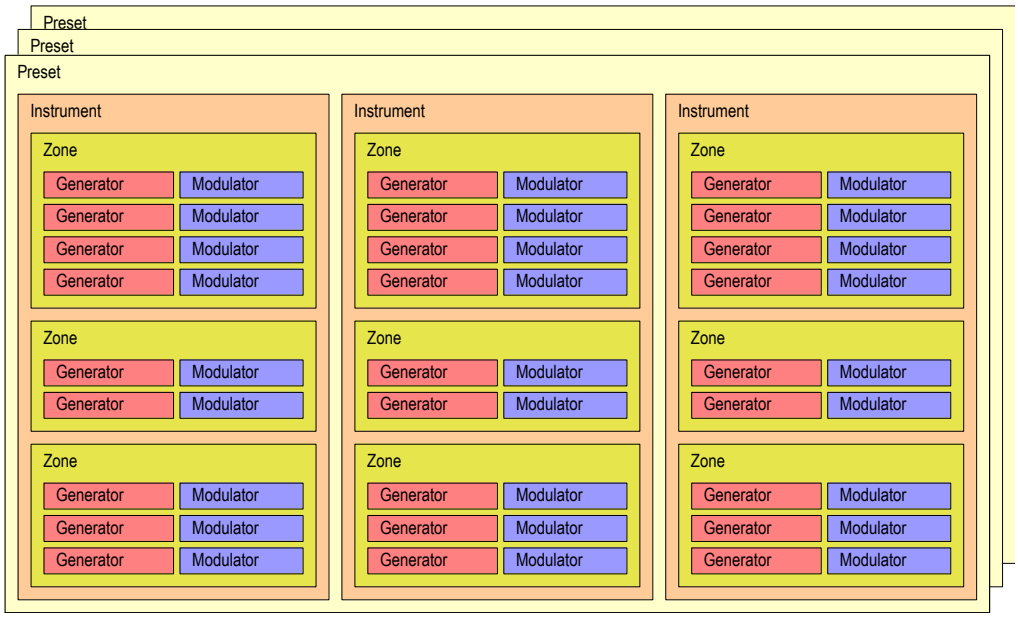
Obrázek 4: Struktura souborového formátu SoundFont

Veškeré podrobnosti a technické specifikace souborového formátu SoundFont jsou k dispozici v jeho dokumentaci[6].

3.5.2. Reprezentace nástrojů

V abstraktní struktuře SoundFontu jsou hlavním objektem nástroje (angl. instrument), které pak mohou být slučovány do větších celků, přednastavení (angl. preset). Jako preset si lze představit zvukovou banku nějakého konkrétního typu nástroje (např. klavír) a nástroje zahrnuté v tomto presetu zastupují skutečné jednotlivé nástroje (např. klasický klavír, hapsichord, varhany, atd.). Každý nástroj v sobě obsahuje identifikující informace. Dále je každý nástroj popsán několika zónami. Zóna (angl. zone) je definovaná na určitém rozsahu tónů. Pro každou zónu jsou definovány generátory a modulátory. Funkce těchto objektů je přesně taková, jak se z jejich názvů dá očekávat. Data uložená v generátoru jsou potřené pro základní generování zvuku. Obsahuje informace jako výška tónu uloženého vzorku, rozsah tónů, které z něho jdou korektně přepočítat nebo definici obálky ADSR. Data uložená v modulátoru definují vlastnosti pro úpravu zvukových dat v reálném čase, jako např. pitch wheel, frequence modulation nebo vibrato. Generátory a modulátory neobsahují funkce či algoritmy pro generování a úpravu zvuku. Jsou v nich pouze uloženy údaje, pomocí kterých syntetizér tyto funkce provádí.

[6] http://connect.creativelabs.com/developer/SoundFont/sfspec21.pdf



Obrázek 5: Abstraktní struktura obsahu souboru formátu SoundFont.

3.6. Zvuková syntéza

Zvuková syntéza je název pro proces, ve kterém se digitální technika snaží napodobit zvuk nějakého nástroje nebo zvukového zdroje ze skutečného světa. Existuje ještě druhá větev syntézy zvuku, a v této větvi se pomocí digitální techniky tvoří harmonické zvuky, které naopak skutečnosti neodpovídají a záměrně zní uměle[7]. Tato druhá větev není součástí práce, která se zabývá pouze první zmíněnou větví, tedy pomocí digitálních dat vygenerovat zvuk, který připomíná zvuk skutečného nástroje.

Nastává problém, jak takové zvuky (resp. data reprezentující zvuk, která jsou předána zvukové kartě a ta je pak pomocí vhodného zařízení přehraje a vytvoří tak zvuk) uchovávat na počítači a jakým způsobem je zpracovat.

Nejpřímější varianta je si nahrát zvuk nástroje do počítače (resp. přes nahrávací zařízení a audio-digitálního převodníku snímat zvukové vlny, ty převést na číselné hodnoty a ty poté uložit ve vhodném formátu do souboru v počítači). Tato metoda je na první pohled nevhodná. Na počítači by musela být uložená data pro každý tón nástroje. Další problém je, že uživatelé požadují a potřebují různě úrovně hlasitosti, různou délku tónů. Pro reprezentaci jednoho nástroje by bylo zapotřebí velké množství paměti. Navíc, při práci se zvukem je většinou takových nástrojů

[7] elektronická hudba, zvukové efekty

zapotřebí víc, takže paměťová náročnost této metody je obrovská. Na druhou stranu, byl by velmi nízký nárok na početní sílu počítače. Data by se pak pouze přenášela na buffer a žádná úprava dat by nebyla zapotřebí. Takový pohled na zvukovou syntézu se v praxi nepoužívá, uvedl jsem ho pouze jako příklad.

Jeden z v praxi používaných metod zvukové syntézy se nazývá aditivní (součtová). Tato metoda je založena na Fourierově transformaci. Říká, že libovolný periodický signál se dá složit pomocí množiny harmonických signálů (signály ve tvaru funkcí sinus a cosinus). Zvuk hudebního nástroje je periodický, protože každý jeho tón má určitou frekvenci. Spolu s touto frekvencí, která určuje výšku tónu, jsou ve zvuku zahrnuty i vyšší harmonické frekvence, které jsou charakteristické pro každý nástroj nebo zdroj zvuku. Právě tyto frekvence jsou zodpovědné za to, že různé nástroje při stejné výšce tónu znějí jinak. Zvuk nástroje se tedy dá tedy napodobit složením harmonických složek. Ve skutečných nástrojích se tyto harmonické složky generují pomocí oscilátorů, které se poté v nástroji sečtou a vytvoří tak zvuk, který je napodobeninou skutečného nástroje. Oscilátorů (přeneseně harmonických složek) může být jenom konečně mnoho. Z tohoto důvodu nikdy přesně nenapodobí skutečný zvuk. Metoda se ale hojně používá u mnoha syntetizérů. Tento typ jsem nezvolil, protože jsem se chtěl pokusit o co nejvěrohodnější zvuk.

3.6.1. Wavetable synthesis

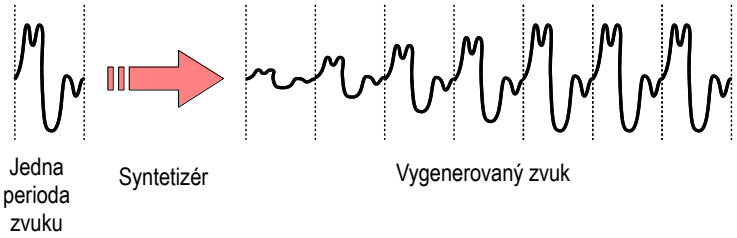
Metoda, kterou jsem zvolil, je částečně založená na předchozí aditivní metodě a byl k ní přidán nápad z první zmíněné metody, uchování vzorku zvuku. Nazývá se metoda zvukových tabulek (angl. Wavetable synthesis). Je založena na opakovaném přehrávání uloženého zvukového vzorku. Zvukový vzorek je jedna perioda zvukového signálu, neboli vlna (odtud název metody), která při opakovaném přehrávání vytvoří libovolně dlouho znějící zvuk[8]. Tento vzorek byl získán nahráním ze skutečného nástroje, proto bude dostatečně věrohodný. Skutečné nástroje ale v čase mění hodnotu signálu. Program proto bude podporovat zvukové vzorky, které mají více period. Generovaný zvuk tak bude znít realističtěji[9]. Jestli má syntetizér

[8] Opakovaným nahráváním krátkého vzorku do bufferu vznikne efekt dalšího úseku dat. Po převedení zvukovým zařízením na zvuk vznikne efekt dlouze trvajícího zvuku.

[9] V případě dlouhých zvukových vzorků se nebude jednat o syntézu v pravém smyslu. K přehrávání již nahraných vzorků nástroje slouží zařízení tzv. sampler.

k dispozici kvalitní zvukové vzorky, bude výsledný zvuk velmi věrohodný, a toho jsem chtěl v práci docílit.

Zvukových vzorků, které jsou k dispozici pro nějaký nástroj, je pouze několik a nepokrývají celý rozsah tónů, kterými disponuje napodobovaný nástroj. Převést zvukový vzorek na jiný tón (resp. jinou frekvenci) je netriviální problém a popíši ho samostatně dál v práci. Vzorek byl také nahrán při jedné úrovni hlasitosti. Změna úrovni hlasitosti odpovídá změně amplitudy signálu, což není problém spočítat[10]. Syntetizér je tedy schopný z jednoho vzorku generovat různé úrovně hlasitosti a tóny různých výšek[11].



Obrázek 6: Princip syntézy zvukových tabulek

3.6.2. Zvukové tabulky

Zvuková tabulka (angl. wavetable) je množina dat reprezentující zvuk. Nástroj vydává zvuk, což je to mechanické vlnění v látkovém prostředí. Toto vlnění je snímáno pomocí nahrávacího zařízení. Zařízení zaznamenává velikost vln v závislosti na čase, převádí spojitý průběh vlnění na diskrétní číselné hodnoty reprezentovatelné v počítači[12]. Tímto způsobem se zvuk reprezentuje a uchovává v digitálních zařízeních. Diskretizovaný a digitalizovaný signál se dále zpracovává. V případě tabulky s jednou periodou zvuku je z nahraného vzorku extrahována jedna perioda tohoto zvuku (v případě víceperiodické tabulky jich je extrahováno více). To se opakuje pro požadovaný počet vzorků. Výsledek je sada zvukových tabulek, které reprezentují nástroj.

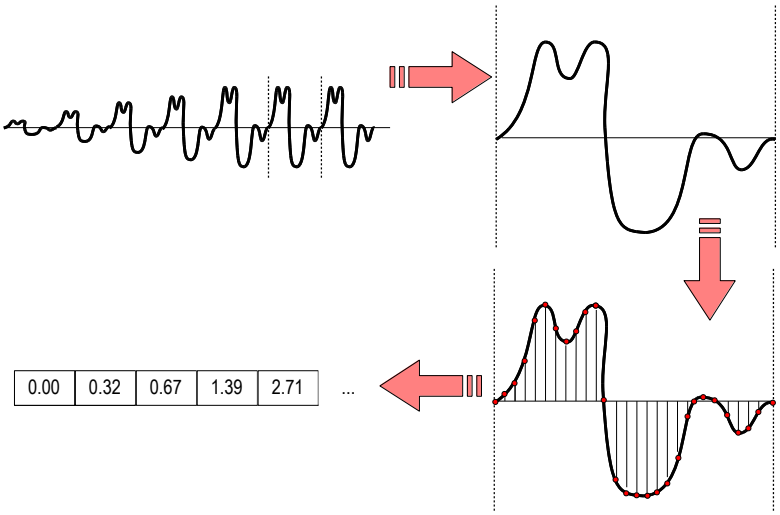
[10] Stačí všechny hodnoty tabulky vynásobit konstantou.

[11] Rozsah tónů, který lze ze vzorku generovat je samozřejmě omezený. Je to kvůli toho, aby převedený vzorek zněl realisticky.

[12] Převod spojitého signálu na diskrétní hodnoty se nazývá vzorkování.

Tyto zvukové tabulky jsou uloženy do souboru. V předchozí kapitole jsem popisoval souborový formát SoundFont. Tabulky tedy budou uloženy do souboru tohoto formátu spolu s informacemi o získaných datech a o nástroji. Soubor se zvukovými tabulkami dostane k dispozici program, který je zpracuje a nahraje do paměti. Tabulky bude přepočítávat a bude z nich generovat výsledná data reprezentující požadovaný zvuk. Souborový formát Soud Font nerozlišuje, jestli se jedná o vzorek s jednou nebo mnoha periodami. V informacích o zvukové tabulce v souboru jsou uloženy údaje, na kterém místě v datech začíná a končí zvuková tabulka vzorku daného tónu pro daný nástroj. Jestliže je délka přehrávaného tónu delší, než délka vzorku, začne se vzorek přehrávat znova od místa označeného v datech jako začátek vzorku[13]. Z tohoto důvodu program nerozlišuje, jestli je ve vzorku jedna nebo více period. Tudíž program bude umět zároveň zpracovávat zvukovou tabulku s jednou periodou zvuku i zvukovou tabulku s mnoha periodami zvuku.

Kvalitní zvukové tabulky obsahují i vzorky pro potřeby implementace obálky ADSR. Při generování dlouhého tónu jsou zohledněny jednotlivé části průběhu amplitudy v závislosti na čase. Část A a D jsou přehrány po sobě, vzorek je periodicky přehráván v části S a při ukončování přehrávání se přehraje část R.



Obrázek 7: Znázornění reprezentace zvukové tabulky

[13] S přihlédnutím k obálce ADSR.

4. Návrh řešení

4.1. Struktura VST .NET

VST .NET se skládá ze tří knihoven, a to Vst.Interop, Vst.Core a Vst.Framework. Všechny objekty těchto knihoven jsou rozděleny do tří jmenných prostorů, které odpovídají názvům knihoven, tj. Vst.Interop, Vst.Core a Vst.Framework.

Jak název napovídá, v knihovně Vst.Interop se nachází třídy a metody zodpovědné za přenos dat mezi pluginem a hostující aplikací. Dále jsou funkce knihovny zodpovědné za správnou komunikaci mezi těmito dvěma částmi a také za konverzi datových struktur a návratových hodnot funkcí na správný tvar. Tato knihovna obsahuje funkci VSTPluginMain, která je volaná hostující aplikací. Vrací datovou strukturu AEffect identifikující plugin. Tato datová struktura obsahuje několik ukazatelů na funkce volané hostující aplikací. Knihovna Vst.Interop zachytí volání těchto metod pomocí tříd Plugin Command Proxy a Plugin Command Stub, vykoná funkci pluginu a provede konverzi datových typů a návratových hodnot na správné hodnoty, aby je hostující aplikace přijala.

Knihovna Vst.Core obsahuje definici běžných řízených datových struktur, které jsou sdílené mezi pluginem a knihovnou Vst.Interop. Všechny tyto struktury jsou reprezentací nativních neřízených struktur z C++. Dále obsahuje rozhraní, která implementují všechna volání metod, která mohou poslat hostující aplikace pluginu nebo naopak. Jako poslední obsahuje rozhraní pro přímý přístup ke zvukovým bufferům, které jsou použity během zpracování zvukových dat.

Poslední knihovna, Vst.Framework, zajišťuje třídy a metody pro členěnou práci s VST. To zahrnuje implementaci několika klíčových rozhraní. Ze všech knihoven je její funkčnost nejmenší, zajistí ale pohodlnější a přehlednější práci.

4.2. Práce s VST .NET

Pro rozeznání a načtení pluginu hostující aplikací musí program implementovat určitá rozhraní.

4.2.1. Plugin Command Stub

Plugin Command Stub je třída implementující rozhraní IVstCommandStub. V knihovně Vst.Interop se nachází třída PluginCommandProxy, která volá metody třídy PluginCommandStub, proto se musí v inicializaci pluginu vytvořit instance této třídy. Jelikož je tato třída volaná knihovnou Vst.Interop, je třeba jí nastavit veřejnou viditelnost (public). Jsou dvě možnosti, jak tuto třídu vytvořit. První možností je implementace rozhraní IVstPluginCommandStub třídou PluginCommandStub nacházející se ve jmenném prostoru Vst.Core, v tom případě se musí implementovat všechny metody, které rozhraní definuje. Většina těchto metod nevykonává v pluginu žádnou funkci a můžou vracet nulovou hodnotu, proto je výhodnější druhá možnost implementace, odvození od třídy StdPluginCommandStub, která se nachází ve jmenném prostoru Vst.Framework. V druhém případě implementace stačí předefinovat metodu CreatePluginInstance a jako parametry ji předat základní údaje identifikující plugin. Ve třídě tohoto jmenného prostoru jsou již implementované všechny metody definované rozhraním. V obou případech je pak potřeba vytvořit instanci této třídy. Tímto je vytvořena instance, která je nezbytná pro správné načtení a identifikaci pluginu. Metoda CreatePluginInstance musí jako hodnotu vracet instanci třídy Plugin, reprezentující samotný plugin.

4.2.2. Plugin

Implementace rozhraní IVstPlugin je povinné pro všechny VST .NET pluginy. Právě implementace tohoto rozhraní obsahuje nejvíce funkčního kódu samotného programu. Třída Plugin musí toto rozhraní implementovat. Stejně jako v případě implementace třídy Plugin Command Stub, je na výběr několik možností, jak toto provést. První možnost je ruční implementace tohoto rozhraní. Druhá možnost je odvození od některých ze tříd VstPluginBase nebo VstPluginWithInterfaceManagerBase. První z uvedených tříd implementuje všechny metody rozhraní na úrovni výchozích hodnot. Druhá z uvedených tříd je navíc odvozena od třídy Plugin Interface Manager Base, která implementuje metody rozhraní IExtension. Díky tomu již budou definované metody CreateX, kde X je název třídy. Jsou to třídy PluginEditor, AudioProcessor a MidiProcessor. Tyto třídy plní důležité funkce v pluginu. Vyplatí se odvození od druhé zmíněné třídy VstPluginWithInterfaceManagerBase, protože metody definované touto třídou budou užitečné při tvorbě pluginu.

4.2.3. Audio Processor

Tato třída je obecně třeba pro generování zvukových dat. Pro napsání VST pluginu efekt i instrument je třeba implementovat tuto třídu. To se provede implementací rozhraní IVstPluginAudioProcessor nebo IvstPluginAudioPrecisionProcessor, přičemž druhé rozhraní se liší jen v tom, že používá čísla s pohyblivou desetinnou čárkou s dvojitou přesností. Zvláštní pozornost je třeba věnovat metodě Process, která je volaná pokaždé, když je třeba generovat data reprezentující zvuk do sdíleného bufferu (tato volání pochází od hostující aplikace přes Vst.Interop) a je zodpovědná za veškerý audio výstup v pluginu.

4.2.4. Midi Processor

Instance této třídy není obecně potřeba pro běh pluginu, v tomto případě je ale nezbytná, protože dodává programu informace o příchozích MIDI událostech. Jako v případě předchozí třídy Audio Processor, i zde plní hlavní funkci metoda Process. Je volána přes Vst.Interop pokaždé, když je třeba zpracovat data z MIDI vstupu. Jako argument funkce je předána kolekce speciálních událostí VstEventCollection. Tyto události se nazývají VstMidiEvent. Každá událost obsahuje informace o MIDI události, které jsou třeba pro zpracování této události. Těchto událostí může být v danou chvíli samozřejmě více, proto je jako argument předávaná kolekce a ne jednotlivé události. Třída je vytvořena implementací rozhraní IVstMidiProcessor. Tentokrát je pro implementaci k dispozici pouze toto rozhraní, narozdíl od jiných předešlých tříd.

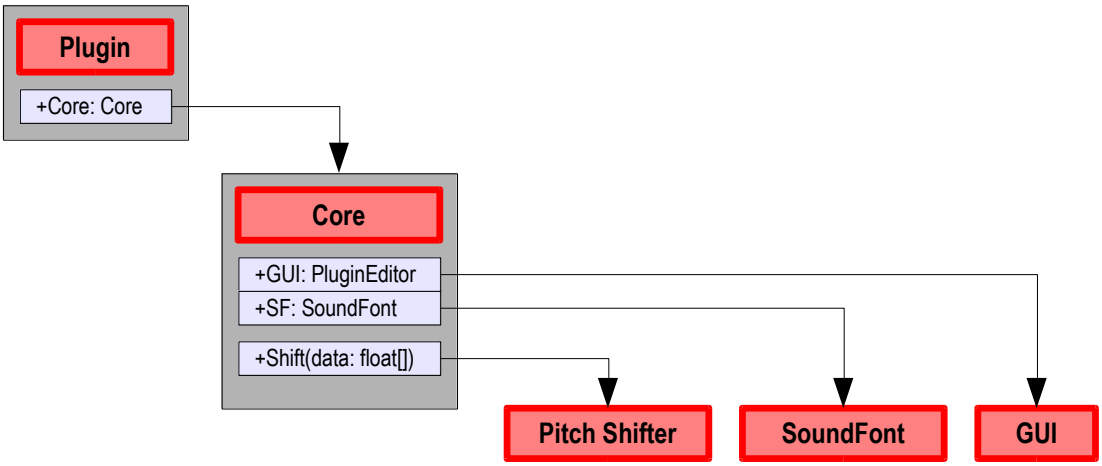
4.2.5. Plugin Editor

Poslední třída důležitá pro psaní pluginu je Plugin Editor. Je vhodné uživateli zobrazit přívětivé a přehledné grafické prostředí. Grafické prostředí je možné implementovat právě pomocí této třídy. Je na ni důležité jen několik věcí, ostatní se programují přímo v grafickém okně. Jako první se vytvoří instance třídy WinFormsControlWrapper, která reprezentuje zobrazení prvku User Control do hostující aplikace. V případě tohoto projektu je vytvořená User Control prvek s názvem GUI. Třída WinFormsControlWrapper je generická a musí se natypovat na vytvořený User Control prvek, to je GUI. Hlavní metoda této třídy je ProcessIdle, přes kterou plugin komunikuje s grafickým prvkem. Může ji předat reference na

objekty, přes které pak může toto okno předávat informace pluginu. Konkrétně tato metoda nekoná hlavní funkčnost, slouží jen jako prostředník pro předání dat metodě se stejným názvem, která se nachází ve třídě WinFormsWrapper v objektu SafeInstance. Tato metoda pak provádí konkrétní operace v ovládacím prvku.

5. Programátorská dokumentace

5.1. Komunikace mezi součástmi programu



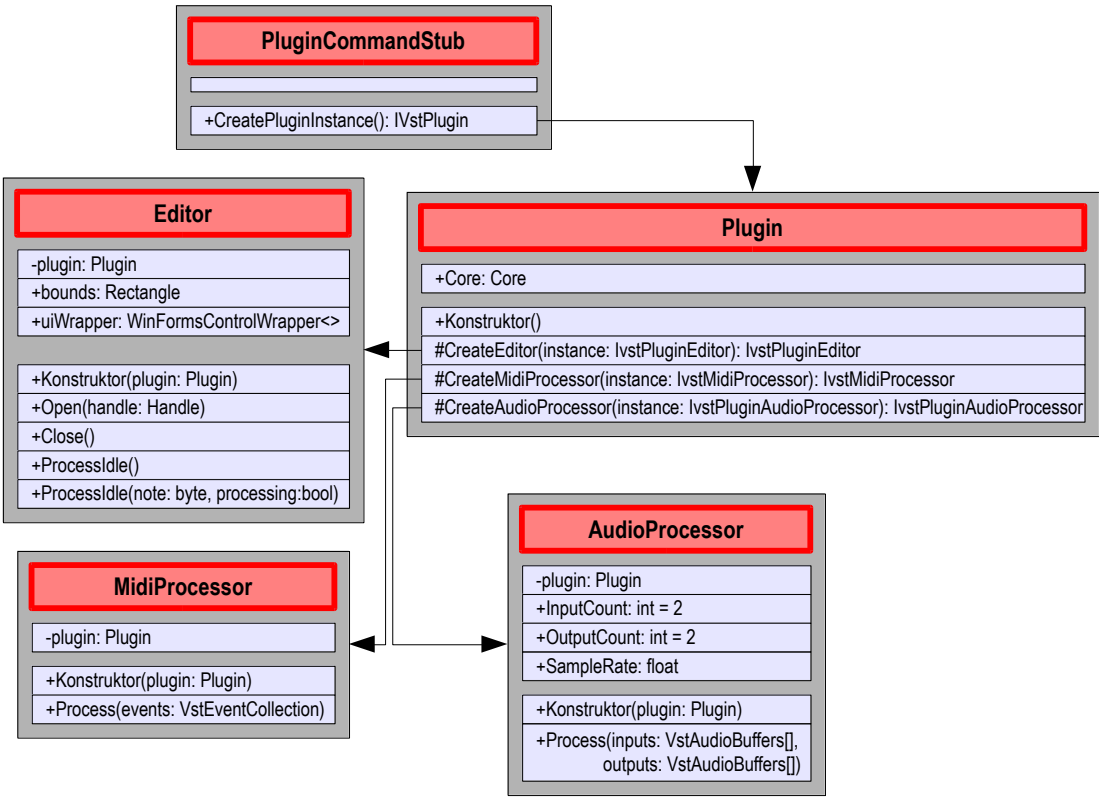
Obrázek 8: Zobrazení závislosti tříd různých částí programu pomocí UML schématu. Zobrazeny jsou pouze datové struktury a metody, které bezprostředně využívají objekty jiné třídy.

V této kapitole budu popisovat pouze třídy a datové struktury, které nemají přímou návaznost na VST, ale jsou klíčové pro vlastní funkčnost pluginu, se zaměřením na propojení mezi těmito třídami a strukturami. Propojení s VST budou popisovat následující kapitoly. Klíčová je třída Plugin, která (kromě metod a datových struktur nutné pro funkčnost VST .NET) obsahuje instanci třídy Core. Ve třídě Core se odehrává hlavní funkčnost programu. Na tuto třídu se odkazují především jiné části programu (ať už na žádost uživatele nebo na žádost hostující aplikace) a řídí tím tok dat. Obsahuje instanci třídy SoundFont, ve které jsou uložené data ze souboru formátu SoundFont. Obsah tohoto soubor je načtený do paměti, protože pro práci se zvukem je nezbytná rychlá odezva. Dále tato třída obsahuje instanci třídy GUI, která reprezentuje grafické rozhraní programu a načítá z ní parametry, které zadal uživatel. Jako poslední se tato třída odkazuje na metodu statické třídy Pitch Shifter, jejíž úkolem je převádět zvukový vzorek reprezentující zvuk o určité výšce tónu na jinou výšku. Propojením těchto zmíněných tříd vznikne základní funkční kostra programu.

5.2. Struktura programu

Program se skládá z pěti částí. Každá část má na starost nezávislou oblast činnosti programu.

5.2.1. Plugin.cs



Obrázek 9: UML schéma zdrojového souboru Plugin.cs

Zdrojový kód první části se nachází v souboru Plugin.cs. Právě tuto část volá Vst.Interop jako první. Je to způsobené tím, že se v ní nachází definice třídy PluginCommandStub. V ní je jediná metoda CreatePluginInstance, která vrací objekt implementující rozhraní IVstPlugin. Jacobi.Interop při inicializaci pluginu volá právě tuto metodu. V tomto případě je rozhraní IVstPlugin implementované třídou Plugin. Metoda CreatePluginInstance proto vrací instanci třídy Plugin.

Třída Plugin obsahuje instanci třídy Core. Ta představuje jádro programu, ve kterém se odehrávají veškeré důležité a zajímavé procesy. Třída dále přepisuje metody CreateEditor, CreateAudioProcessor a CreateMidiProcessor. Tyto metody jsou volané knihovnou Vst.Interop na předané instanci třídy Plugin. Tyto metody vracejí instance tříd PluginEditor, AudioProcessor a MidiProcessor. Tyto třídy jsou zodpovědné za chování jistých částí VST. Konkrétně třída PluginEditor má na starosti grafické rozhraní. Třída AudioProcessor má na starosti generování dat reprezentujících výstupní zvuk. Třída MidiProcessor zpracovává MIDI události přicházející z hostující aplikace.

Nejdůležitější na třídě MidiProcessor je metoda Process. Tu volá VST pokaždé, když má nová MIDI událost být zpracována. Jako parametr předá funkci kolekci VST událostí. Tato funkce slouží jako prostředník pro jádro programu. Každou událost zkontroluje, jestli se jedná opravdu o VST MIDI událost. Zjistí, jaký je typ této události a podle typu zavolá obslužnou funkci v jádru.

Podobné to je i u třídy AudioProcessor v její nejdůležitější metodě Process. VST předá této funkci jako parametry reference na vstupní a výstupní buffer[14]. Referenci na vstupní buffer program nepotřebuje, protože za vstup výstup se generuje na základě MIDI událostí a nikoliv na základě dat ze zvukového vstupu. Referenci na výstupní buffer předá jádru programu, které do tohoto bufferu vygeneruje požadovaná data.

Grafické rozhraní pluginu reprezentované třídou PluginEditor má již víc metod. Metody Open a Close jsou volány VST, když byl obdržen požadavek na otevření nebo zavření okna pluginu. Metoda ProcessIdle je volaná VST, když je třeba zpracovat okno pluginu. Třída obsahuje referenci na objekt typu WinForm Control. Tento objekt reprezentuje okno aplikace a nachází se v něm ovládací prvky pluginu.

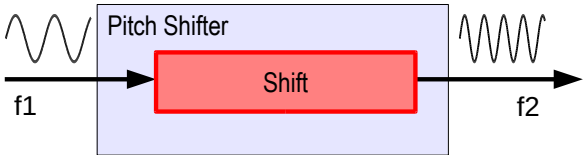
5.2.2. PitchShifter.cs

Jak jsem zmínil v kapitolách o zvukových tabulkách a souborovém formátu SoundFont, zvukové vzorky zpravidla nejsou uchovávány pro všechny výšky tónů nástroje. Program proto potřebuje nástroj, který data reprezentující tón o určité výšce přepočte na data reprezentující tón o různé, požadované výšce. Protože se jedná o komplexní netriviální problém, umístil jsem kód tohoto přepočtu do samostatné části programu.

Zdrojový soubor PitchShifter obsahuje pouze jednu statickou třídu PitchShifter. Tato třída obsahuje pomocné metody pro přepočet dat, jsou ale privátní a neviditelné pro zbytek programu. Má jedinou veřejnou statickou metodu Shift. Ta jako parametry přebírá data pro přepočet (zvukový vzorek) reprezentovaná polem hodnot, výšku originálního tónu, výšku požadovaného tónu a velikost datového toku

[14] V tomto bufferu jsou uložená data reprezentující zvuk. Data jsou pak předána zvukové kartě pro převedení na výsledný zvuk.

vzorku[15]. Funkce této metody spočívá v přepočtu pole tak, aby nová data reprezentovala tón o změněné výšce.



Obrázek 10: Vizualizace funkce zdrojového kódu PitchShifter.cs

5.2.3. GUI.cs

Zdrojový soubor GUI.cs není funkčně zajímavý a není klíčový pro chod programu. Definuje objekt odvozený od typu UserControl. Je to grafický panel, což je obdoba klasického okna (form). Tento panel se jako grafický prvek zobrazí v uživatelském rozhraní pluginu v hostující aplikaci. Grafické rozhraní se nesmí implementovat jako samostatné okno. Důvod je ten, že hostující aplikace si sama vyžádá grafický panel pluginu a dosadí ho do okna, které sama vygenerovala.

Ve zmíněném objektu je reference na instanci třídy Core, do kterého se pomocí metod objektu ukládají data získaná uživatelem. První z těchto metod předá instanci třídy Core cestu k souboru formátu SoundFont. Druhá metoda předá instanci třídy Core nástroj zvolený uživatelem a vyšle jí žádost o otevření zadaného souboru a uložení ho do paměti.

Poslední funkce, kterou GUI nabízí, je vizualizace uživateli, jaká část souboru se již načetla.

5.2.4. Soundfont.cs

Soubor formátu SoundFont je třeba rozčlenit a nahrát do paměti programu. K tomuto účelu slouží zdrojový kód s názvem Soundfont.cs. Je v něm definovaná třída Soundfont. Jejímu konstruktoru program předá název souboru zvoleného uživatelem. S instancí třídy Soundfont poté pracuje plugin při generování zvuku.

Organizace třídy Soundfont odpovídá členění dat v souboru tohoto formátu. Obsahuje instance podtříd InfoChunk, PresetsChunk a SampleDataChunk. Podtřída InfoChunk obsahuje základní informace o souboru, které nejsou funkční a slouží

[15] Datový tok je počet bajtů zpracovaných za jednotku času. V tomto případě to odpovídá počtu bajtů, které z bufferu za jednotku času (vteřinu) zpracuje zvuková karta. Přeneseně, kolik bajtů karta přehraje za vteřinu.

pouze k identifikace souboru. Podtřída PresetsChunk obsahuje logické členění struktury. Jsou v ní informace o presetech (přednastavení), nástrojích, zónách, generátorech a modulátorech. Odpovídá to struktuře souborového formátu SoundFont. V poslední podtřídě SampleDataChunk jsou uložená samotná data, ke kterým plugin přistupuje pomocí informací uložených ve třídě PresetsChunk.

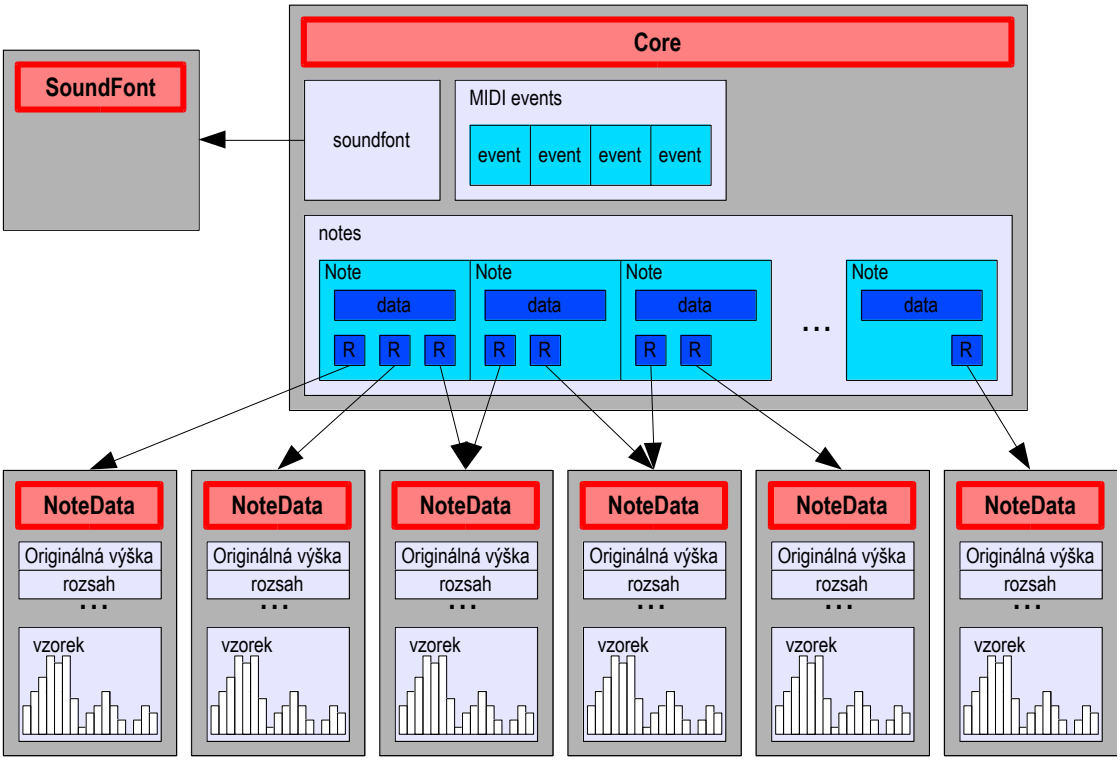
5.2.5. Core.cs

V části programu Core.cs se nachází nejvíce funkčního kódu a je pro jeho chod nejpodstatnější. Nachází se vní instance třídy Core, která reprezentuje jádro programu.

Třída Core zahrnuje několik důležitých datových struktur. Nejpřímočařejší je reprezentace souboru SoundFont načteného do paměti. Implementuje ho třída SoundFont, která je členěna do jednotlivých struktur odpovídající členění souboru do zón, nástrojů, apod. Další datová struktura je fronta MIDI událostí. Její obsluha je jednoduchá. Pokaždé, když hostující aplikace předá pluginu informaci o nové MIDI události, třída Plugin předá obslužné metodě ve třídě Core tuto událost, která ji zařadí do fronty MIDI událostí. Při generování zvuku se pak prochází tato fronta a na základě událostí v ní uložených vykonává svou funkci. Plugin implementuje reakce pouze MIDI zprávy Note On, Note Off a Pitch bend. Poslední z důležitých datových struktur je pole instancí tříd Note s názvem notes. Pole má velikost přesně 128, což odpovídá velikosti stupnice definovanou MIDI normou. Třída Note reprezentuje informace a data pro daný tón (index odpovídá hodnotě tónu). Její obsah je je seznam instancí tříd NoteData. Třída NoteData je odvozená od třídy Zone (tato třída je obsažena ve struktuře SoundFont) a obsahuje data o daném tónu. Zón může být pro daný tón více, proto i třída Note obsahuje seznam instancí tříd NoteData. Třída NoteData obsahuje pouze data, která jsou zpravidla společné pro více tónů[16]. Třída Note pak na základě dat ze seznamu instancí tříd NoteData vygeneruje data vyžadované MIDI událostmi pro konkrétní tón. Operace, které musí provádět jsou

[16] Více tónu v rozsahu jedné zóny odkazují referencí na stejnou instanci třídy NoteData. Dává to smysl, protože noty v rozsahu mají stejný základ, z kterého se odvozují data pro konkrétní tón. Navíc to šetří místem, protože velikost reference je zanedbatelná (řádově bajty), na rozdíl od velikost dat pro tón (řádově kilobajty).

například změna výšky tónu (pitch shifting) nebo skládání vzorků více zón do jednoho.



Obrázek 11: Organizace datových struktur ve třídě Core

Kromě zmíněných datových struktur třída Core obsahuje metody, které jsou volané aplikací skrz třídu Plugin. Jsou to LoadSoundfont, ProcessMidiEvent a PlayAudio. Metoda LoadSoundFont má dvě fáze, v první uživatel vybere požadovaný soubor, ve druhém vybere nástroj, který chce používat. Metoda načte data ze souboru do již zmíněné datové struktury Soundfont. Metoda ProcessMidiEvent dostane jako argument MIDI událost, kterou pak zařadí do fronty MIDI událostí. Podle požadavků různých typů událostí provede i jinou akci. Například po přijetí události Note Off se z fronty odebere událost odpovídající Note On pro daný tón. Jiné typy událostí mohou například měnit celkovou hlasitost, v tomto případě se do fronty nepřidává událost, ale upraví se odpovídající proměnná. Metoda PlayAudio ovládá veškeré generování zvuku. Jako parametr bere dva buffery (reprezentující levý a pravý zvukový kanál, nakopírováním dat do těchto bufferů předá metoda výstup aplikaci). Je nezávislá na metodě ProcessMidiEvent. Metoda projde frontu MIDI událostí a vybere tóny, které se mají přehrávat. Jestli se jedná o zvláštní událost typu pitch wheel, vykoná zvláštní obslužné funkce. Dál metoda

požádá o výstupní data pro jednotlivé tóny, které sloučí a nakopíruje do výstupních bufferů.

5.3. Příprava dat a generování výstupu

V tomto programu vstup reprezentuje příchozí MIDI událost. Program tuto informaci získá přes argument metody Process nacházející se ve třídě MidiProcessor. Toto volání uskutečňuje hostující aplikace. Metodu Process volá pokaždé, když je třeba zpracovat příchozí MIDI zprávu. Instanci třídy reprezentující MIDI zprávu (MidiEvent) předá aplikace jako argument metody. Činnost této metody je poměrně jednoduchá. Ve třídě MidiEvent zjistí, o jaký typ MIDI zprávy jde a podle typu rozhodne, jak s ní naloží:

1. Nejprve zjistí, jestli uživatel nezvolil ignorování tohoto typu zprávy. V tom případě metoda končí.
2. Jestliže typ zprávy není ignorovaný, metoda zjistí, jestli je zpracování tohoto typu zprávy implementován a jestli ano, vykoná jednu z následujících možností:
3. Jestliže se jedná a Note On zprávu, pak přidá do fronty přehrávaných tónu právě tón, který je zmíněný ve zprávě.
4. Jestliže se jedná o Note Off, potom ukončí přehrávání daného tónu (přehrávání přejde v obálce ADSR do oblasti Release) a odebere tón ze seznamu přehrávaných tónů.
5. Jestliže se jedná o Pitch Wheel, potom upraví konstantu, která řídí rychlost přehrávání vzorků a tím upraví výšku přehrávaných tónů ovlivněnou kontrolerem Pitch Wheel.

Informace o aktuálním tónu jsou uloženy v MIDI zprávě (viz kapitola o MIDI). Tímto je ošetřen MIDI vstup pro plugin. V budoucnu je možné vylepšit funkčnost plugin implementováním dalších typů MIDI zpráv.

Hostující aplikace žádá o vygenerování výstupu voláním metody Process nacházející se ve třídě AudioProcessor. Jako parametry ji předá reference na pole VSTBufferů. Typicky jsou dva, a to pro levý a pravý kanál. Tyto VSTBuffery slouží jako prostředník pro buffer zvukové karty. V důsledku tedy plugin nakopírováním

dat do VSTBufferů předal data zvukové kartě. Metoda Process v této třídě slouží pouze jako prostředník a její jediný účel je volání metody PlayAudio ve třídě Core. Předá ji reference na VST Buffery a o nakopírování dat do těchto bufferů se postará zmíněná metoda PlayAudio. Funkce této metody popisuje algoritmus:

1. Pro všechny přehrávané tóny vykonej:
2. Nakopíruj do bufferu maximální množství dat:
3. Jestliže je vzorek dostatečně dlouhý, je to v pořádku.
4. Jestliže vzorek není dostatečně dlouhý, pokračuj v kopírování s daty od začátku smyčky[17].
5. Jestliže je originální tón vzorku o jiné výšce, než výška přehrávaného tónu, změň jeho výšku (proveď Pitch Shifting).

Při kopírování dat do bufferu se hodnoty na pozicích bufferu sčítají. Je to kvůli přehrávání více tónů najednou. Kdyby se hodnoty pouze přepisovaly, informace o předešlých tónech by se ztrácely. Buffer je před touto operací samozřejmě nutné vynulovat. V případě přehrávání mnoha tónů hrozí nebezpečí přetížení (způsobené přičítáním hodnot mnoha vzorků až do hodnoty, které zvuková karta nedokáže správně přehrát). Konstantní úroveň hlasitosti metoda zajistí vydělení hodnotou počtem přehrávaných tónů. Například, jestli je v danou chvíli přehráváno pět tónů, amplituda signálu tím vzroste pětkrát a proto metoda hodnoty v bufferech vydělí pěti. Tím amplituda klesne zpět na původní úroveň. Opakovaným voláním metody Process aplikace zajistí nepřetržité kopírování dat do bufferu a tím zvukový výstup.

Pro aplikaci Pitch Shiftingu jsem použil následující trik. Zvukový vzorek reprezentující tón o určité výšce je do bufferu kopírován 1:1. Jestliže při kopírování metoda změní tento poměr, docílí tím změnu výšky přehrávaného tónu. Tím se ale změní i době přehrávaného tónu. Tento problém řeší smyčky, které kopírují data znova od vyznačeného místa, pokud při kopírování došlo na konec vzorku. Při kopírování se do bufferu ze vzorku nakopíruje každá i-tá hodnota[18], kde $i = 2^{\frac{n}{12}}$ a n

[17] Smyčku ve vzorku definuje Sound Font.

[18] Po zaokrouhlení, protože pozice v bufferu musí být celočíselné, je také možno použít nějaký druh interpolace hodnot, například lineární.

je rozdíl mezi výškou originálního a požadovaného tónu. Tato hodnota je získaná porovnáním frekvencí tónů různých výšek ve stupnici o dvanácti půltónech.

5.4. Další část dokumentace

Další nedílná část programátorské dokumentace se nachází ve zdrojových souborech pluginu v podobě komentářů.

6. Závěr

6.1. Přínos práce

Plugin využijí zejména uživatelé pracující s MIDI nástroji (existují i například MIDI snímače pro kytary, ale použití kláves je běžnější). Po jejich připojení k počítači mohou používat tento plugin jako zvukovou banku nástrojů. Možností ignorovat některé MIDI zprávy může uživatel dosáhnout zajímavých efektů a upravit chování nástroje. Protože plugin podporuje VST technologii, může být použit spolu s dalšími pluginy (například další efekty, ekvalizéry, a podobně) pro vylepšení a doladění výsledného zvuku.

6.2. Výhody

Největší nespornou výhodou je podpora VST. Plugin mohou rozpoznat všechny aplikace podporující tuto technologii, takže se dá použít v mnoha neplacených i komerčních zvukových aplikacích (včetně velkých studiových stanic). Důsledek podpory VST je možnost zapojení pluginu do většího řetězce dalších pluginů (ať už dalších nástrojů nebo efektů), které mohou dále zpracovávat a upravovat výstup tohoto pluginu.

Další výhodou je používání souborů formátu Soundfont. Obsahuje mimo zvukových vzorků další informace, které pomohou identifikovat nástroje, které má soubor reprezentovat. Jediný problém je získat kvalitní SoundFont, protože kvalitní vzorky bývají zpoplatněné. Našel jsem ale soundfonty, které jsou ke stažení zdarma a jejich kvalita je dostačující. Jestliže uživatel získá kvalitní soundfont, pak plugin dokáže vygenerovat velice kvalitní výstup, protože jeho kvalita nejvíce závisí právě na soundfontu.

Poslední výhodou je možnost zakázat reakci na některé druhy MIDI zpráv. Reakci na zprávu Note On zakázat nelze, protože potom by plugin nemohl tvořit žádný výstup. Zakázáním zprávy Note Off by uživatel mohl docílit zajímavého efektu například u zvuku klavíru. Zakázáním dalších typů zpráv může uživatel ovlivnit nepohodlné chování jeho nástroje. Jestli například nechce používat kontroler Pitch wheel nebo nechce měnit program, který se předtím pracně navolil, může toto vypnout v nastavení pluginu.

6.3. Nevýhody

Mírnou nevýhodou je koncepce pluginu, protože uživatel potřebuje hostující aplikace, aby plugin načetl a spustil. Výše popsané výhody ale převyšují tuto nevýhodu. Jak jsem taky uvedl, uživatelé pracující s MIDI nástroji zpravidla mají k dispozici aplikace, která VST podporuje.

Plugin dokáže načíst v jednu chvíli pouze jeden soundfont. V takovém souboru se nachází pouze jeden typ nástroje (byť s různými modifikacemi). Proto plugin dokáže v jednu chvíli generovat zvuk pouze jednoho nástroje. Je proto vhodný jako navolení zvuku nástroje pro nějaký MIDI nástroj, ale nelze použít, pokud by chtěl uživatel přehrát MIDI soubor (ve kterém je uložena například píseň) tak, aby zněl realisticky.

Největší nevýhodou je občasná odezva při přehrávání tónů. Toto je pravděpodobně způsobeno prací Garbage collectoru. Může se stát, že plugin určitou dobu přehrává tóny správně, ale po nějaké době se začíná opožďovat. Kvůli tomu klesne použitelnost pluginu v praxi, protože krátká odezva je nezbytná.

6.4. Podobné implementace

K nalezení je velká skupina programů, tzv. soundfont přehrávačů (soundfont players), které umí otevřít a zpracovat soubory formátu soundfont. Vyznačují se virtuálním nástrojem (většinou klávesy) v jeho okně a tím, že program simuluje nástroj s použitím načteného soundfont souboru. Některé aplikaci mají samozřejmě funkce navíc. Jako příklady soundfont přehrávačů poslouží programy Extreme Sample Converter, SoundFont Player, Terry West nebo Font!SF2.

Pro operační systém OS X je k dispozici plugin do zvukových aplikací s názvem SoundFont Synth. Funkčností je obdobný mému pluginu, tedy slouží jako programový zvukový syntetizér využívající soubory formátu Soundfont, ale nepodporuje technologii VST.

Pro operační systém Windows a technologii VST jsem nalezl několik pluginů, které funkčností odpovídají mému pluginu. Tedy stejně jako můj plugin, se i ostatní nahrají do hostující zvukové aplikace, využívají MIDI vstup a to buď ze souboru nebo z MIDI vstupu počítače, využívají soubory formátu Soundfont k syntéze zvuku a svůj výstup posílají přes aplikaci do zvukové karty. Navíc většina

těchto pluginů obsahují funkce, které můj plugin neimplementuje. Jako příklad jsou v těchto pluginech funkce ladění tónů, zvukové efekty, ekvalizéry, ovládání obálky ADSR a podobně. Některé z těchto pluginů jsou: Font!, SF Synth 2, Safwan Soundfont Player, SafFron Soundfont Player, a jiné.

Implementací softwarového syntetizéru s použitím souborů soundfont je mnoho, většina z nich má funkce navíc. Uživatel má na výběr z mnoha alternativ zdarma. Jediná funkce, kterou jsem neviděl u žádných dalších implementací, je možnost ignorovat vybrané MIDI zprávy. Uživateli může vyhovovat, jestliže syntetizér bude ignorovat některé typy zpráv a lépe si tak může nastavit jeho chování. Touto implementací je můj plugin jedinečný.

6.5. Návrhy na vylepšení

Nezbytné je plugin vylepšit tak, aby při práci nevznikaly žádné prodlevy při generování výstupu. U ostatních pluginů je běžná přítomnost ekvalizéru, pomocí kterého může uživatel nastavit hlasitost různých frekvenčních pásem a výsledného zvuku. Plugin by mohl implementovat ekvalizér, i když nezbytné to není, protože se dá simulovat nahráním dalšího pluginu do aplikace, který ekvalizér zajistí. V souvislosti s MIDI by bylo vhodné doimplementovat všechny zbývající zprávy, protože plugin neimplementuje všechny používané typy zpráv.

Výrazné vylepšení by bylo výsledkem možností nahrání více soundfontů a možnost je přiřadit různým MIDI kanálům. V tom případě by si uživatel navolil požadované nástroje a ve výsledku by mohl simulovat realističnost vícekanálového MIDI souboru (jestliže by chtěl přehrát píseň uloženou v MIDI souboru tak, aby zněla realisticky). Zajímavým vylepšením by byla možnost kombinovat více soundfontů na jeden kanál. Výsledkem by byla kombinace více nástrojů pro lepší nebo zajímavější zvuk.

Za zmínku by stál návrh opustit vývoj programu pod platformou .NET a pokračovat ve vývoji pod nativním VST SDK API za použití programovacího jazyka C++. Výhodou by byla větší rychlost programu způsobená přímou komunikací s VST voláním nativních funkcí. Jazyk C++ se navíc v současné době používá pro vývoj aplikací nebo pluginů podobného typu nejčastěji.

7. Reference

7.1. Seznam zdrojů a citací

1. http://en.wikipedia.org/wiki/Virtual_Studio_Technology#Software. [Online]

2. http://cs.wikipedia.org/wiki/MIDI. [Online]

3. http://cs.wikipedia.org/wiki/Resource_Interchange_File_Format. [Online]

4. http://cs.wikipedia.org/wiki/SoundFont. [Online]

5. http://www.root.cz/clanky/rozhrani-midi-na-osobnich-pocitacich. [Online]

6. http://www.root.cz/clanky/rozhrani-midi-na-osobnich-pocitacich-ii. [Online]

7. http://en.wikipedia.org/wiki/Sound_synthesis#Types_of_synthesis. [Online]

7.2. Seznam obrázků

7.3. Seznam tabulek

8. Seznam použitých zkratek

A-D ..... Audio-digitální (například A-D převodník)

ADSR ..... Attack/Decay/Sustain/Release, různé zóny v obálce ADSR

API ..... Application Programming Interface, rozhraní sloužící ke snadnějšímu programování a prací s nějakou platformou nebo aplikací

ASIO ..... Audio Stream Input/Output, speciální ovladače zvukových karet

GAC ..... Global assembly cache

GUI ..... Graphic User Interface, grafické uživatelské rozhraní

MIDI ..... Musical Instrument Digital Interface

RIFF ..... Resource Interchange File Format, souborový formát

SDK..... Software Development Kit

SF ..... Sound font, souborový formát

VST ..... Virtual Studio Technology

VSTi ..... Virtual Studio Technology Instrument

9. Přílohy

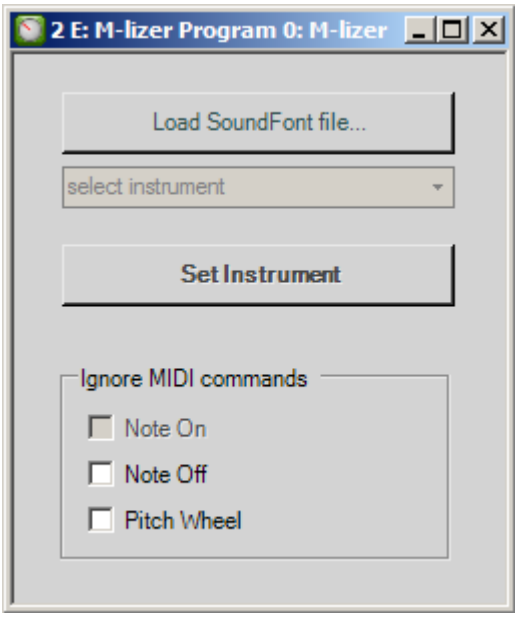
9.1. Uživatelská dokumentace

Instalace pluginu je jednoduchá a odpovídá typické instalaci obecného VST pluginu. Stačí nakopírovat knihovny pluginu[19] do adresáře s VST pluginy adresáře hostující aplikace. Knihovny mohou, ale nemusí být ve zvláštním adresáři. Důležité však je, aby všechny knihovny pluginu byly ve stejném adresáři. Aplikace typicky mívá všechny své pluginy v jednom adresáři (v tom případě stačí nakopírovat plugin přímo do tohoto adresáře), nebo má zvláštní skupinu pluginů pro VST instrumenty, instalace se provede nakopírováním knihoven do adresáře pro VSTi označenou jako „Synth“. Některé aplikace nevyžadují kopírování knihoven do adresáře, ale stačí je načíst manuálně v aplikaci.

V hostující aplikaci uživatel otevře plugin M-lizer (nachází se v části instrumentů, podskupině Synth)[20] a v závislosti na nastavení aplikace otevře uživatelské rozhraní pluginu ručně nebo aplikace načte rozhraní automaticky. Po otevření okna je k dispozici pouze jedno aktivní tlačítko („Load SoundFont file…“), po jehož stisknutí se zobrazí dialog pro zvolení požadovaného souboru SoundFont. Plugin podporuje pouze soubory formátu Sound Font 2.0. V příloze práce je k dispozici několik volně šiřitelných souborů SoundFont. Navíc, na internetu je k nalezení mnoho volně šiřitelných soundfontů. Navíc je k dispozici několik zaškrtávací polí (check boxů), pomocí kterých může uživatel vypnout reakci na některé typy MIDI zpráv. Reakci na určitý typ MIDI zprávy uživatel vypne zaškrtnutím příslušného políčka.

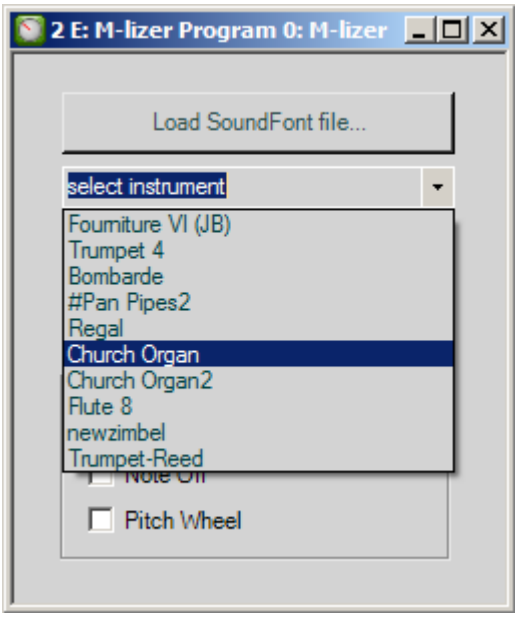
[19] Plugin sestává ze čtyř knihoven, M-lizer.dll, M-lizer.net.dll, Jacobi.Vst.Core.dll a Jacobi.Vst.Framework.dll a pro svou funkčnost vyžaduje přítomnost všech těchto knihoven vedle sebe. Poslední dvě zmíněné knihovny nemusí být přítomny s prvními dvěma za podmínky, že jsou nahrány v GAC (Global assembly cache).

[20] Způsob otevření či načtení pluginu se liší od aplikace. Uživatel si musí zjistit, jak se s pluginy zachází v jeho aplikaci.



Obrázek 12: Uživatelské rozhraní pluginu M-lizer před otevřením souboru SoundFont.

Po otevření souboru se zpřístupní rozbalovací nabídka a další tlačítko. Z rozbalovací nabídky si uživatel zvolí jeden z nástrojů, které obsahuje Sound font. Tento zvolený nástroj se do pluginu nahraje stisknutím tlačítka „Set instrument“ a tento bude používán po dobu běhu pluginu. Jiný soubor nebo nástroj uživatel vybere stejným způsobem, jako je popsán výše.



Obrázek 13: Uživatelské rozhraní pluginu po otevření souboru a rozbalené nabídce nástrojů.

Nyní je plugin připraven k použití a veškeré další operace (práce s MIDI) se odehrávají v hostující aplikaci.

9.2. Obsah přílohového CD

BP/BP.doc - Bakalářská práce ve formátu Microsoft Office 1997 - 2003

BP/BP.pdf - Bakalářská práce ve formátu PDF

Soundfont/* - několik volně šiřitelných souborů ve formátu Sound Font 2.0

VSTHost/ - hostující volně šiřitelná aplikace (licence GNU GPL) podporující VST

M-lizer/ - 4 knihovny pluginu nutné pro jeho spuštění

M-lizer (sources)/ - zdrojové soubory a projekt pluginu