UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jure Turel

Tehnologije in izvedbe zvočnih vtičnikov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Dušan Kodek

Ljubljana, 2010

IZJAVA O AVTORSTVU
diplomskega dela

Spodaj podpisani Jure Turel,

z vpisno številko 63030071,

sem avtor diplomskega dela z naslovom:

Tehnologije in izvedbe zvočnih vtičnikov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Dušana Kodeka
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.), ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI"

V LJUBLJANI, DNE 15.6.2010 PODPIS AVTORJA:_____

ZAHVALA

Zahvaljujem se staršem, še posebej mami, ki mi je študij omogočila in podpirala tudi ob najtežjih trenutkih.

Špela, hvala tudi tebi, da si me spodbujala in stala ob strani.

Zahvala gre tudi profesorju dr. Dušanu Kodeku za mentorstvo in strokovne nasvete.

Kazalo

Seznam uporabljenih kratic in simbolov

VST	Virtual Studio Technology (standard vtičnikov podjetja *Steinberg*)
GUI	Graphical User Interface (grafični uporabniški vmesnik)
VPL	Visual Programming Language (programski jezik z vizualnim programiranjem)
DAW	Digital Audio Workstation (digitalna zvočna postaja)
DPS	digitalno procesiranje signalov
MIDI	Musical Instrument Digital Interface
ASIO	Audio Stream Input/Output
DX	DirectX (vtičniki bazirani na Microsoftovi tehnologiji)
AU	Audio Units (standard vtičnikov podjetja Apple)
TDM	Time Division Multiplexing
LADSPA	Linux Audio Developer's Simple Plug-in API
DSSI	Disposable Soft Synth Interface
SDK	Standard Developement Kit (standardno razvojno okolje)
DDL	Digital Delay Line (digitalna zakasnitvena linija)
KEO	filter s končnim enotinim odzivom
NEO	filter z neskončnim enotinim odzivom
API	Application Programming Interface (programski vmesnik)

Povzetek

Glasba spremlja človeka skozi njegovo celotno evolucijo. Tako kot človek, se je razvijala glasba, kot tudi tehnologije za njeno ustvarjanje. Dandanes pri snemanju in produkciji glasbe prevladujejo digitalne zvočne postaje in različna programska oprema, ki omogoča hitro, enostavno, ter visoko kvalitetno manipulacijo nad zvočnimi signali.

V pričujočem diplomskem delu sem obravnaval tehnologije zvočnih vtičnikov, ki omogočajo ustvarjanje različnih zvočnih učinkov, kot tudi proizvajanje zvoka. Pri veliki večini realizacij takšnih vtičnikov gre za posnemanje analognih naprav, ki so bile uporabljene na različnih glasbenih posnetkih, ki so tako ali drugače zaznamovali različne glasbenike, kot tudi različna glasbena obdobja.

V uvodnem delu bom govoril o tehnologiji zvočnih vtičnikov na splošno, vrstah vtičnikov in različnih formatih, ki so danes zastopani.

V osrednjem delu pa se sem se posvetil prikazu konkretne implementacije ene najpopularnejših tehnologij zvočnih vtičnikov danes, tehnologiji podjetja *Steinberg*, imenovani VST – *Virtual Studio Technology*. Prikazal sem konkretno realizacijo zvočnega vtičnika z učinkom zakasnitve. Prikaz implementacije poteka preko dveh različnih metod. Najprej sem prikazal razvoj vtičnikov s pomočjo osnovnega razvojnega orodja VST SDK v3.0, v drugem delu pa razvoj poteka s pomočjo programov, ki omogočajo programiranje s pomočjo metode vizualnega programiranja.

Ključne besede:
zvočni vtičnik, učinek zakasnitve, VST, vizualno programiranje

Abstract

Music accompanies people throughout our entire evolution. As man has evolved, so did music and technology for its creation. Nowadays, recording and producing music is dominated by digital audio workstations and various digital software, which allows fast, easy and high-quality manipulation of audio signals.

In this diploma thesis I dealt with the audio plug-in technology that enables the creation of various sound effects and also sound reproduction. The vast majority of its implementations is based on various analog devices, which have been used on different musical recordings, which marked different musicians, as well as various musical periods.

In the first part of this thesis I talk about audio plug-in technology in general and about different types and formats of plug-ins which are most common today.

In the central part I focus on a concrete implementation of one of the most popular audio plug-in technology today, Steinberg's technology called VST – Virtual Studio Technology. I was working on a realization of audio plug-in which generates the delay effect. The implementation takes place through two different methods. First I show the development of plug-ins through the basic developement kit VST SDK v3.0. In second part the development takes place through some programs that enable programming by the method of visual programming.

Keywords:
audio plug-in, delay effect, VST, visual programming

1. Uvod

Računalniki in digitalno procesiranje sta dandanes postala neizogiben del ustvarjanja, snemanja in produciranja glasbe. Digitalna tehnologija je že globoko vkoreninila svojo pot in na nekaterih področjih nadomestila analogno tehnologijo zaradi prednosti, ki jih omogoča digitalno procesiranje signalov. Analogne enote zakasnitve, ki uporabljajo neskončen, ponavljajoči se magnetni trak, bralno in pisalno glavo so bile nadomeščene z digitalnimi enotami, ki vzorčijo vhodni signal, shranjujejo vhodne vzorce dokler jih ni dovolj za realizacijo željenega zakasnitvenega časa in jih zakasnjene pošljejo na izhod enote. Z razvojem tehnik digitalnega procesiranja signalov je bilo mogoče vedno več analognih procesov, kot so optični kompresorji in učinki z lastnostmi odmeva, implementirati v digitalnem okolju. Tako je možno vsak del poti zvočnega signala realizirati z digitalnim procesiranjem, z izjemo pretvorbe akustičnega signala v digitalnega. Za to je potreben mikrofon za zajem zvoka, ki pa lahko že vsebuje analogno/digitalni pretvornik, ki opravi potrebno pretvorbo.

Digitalni sistemi so večinoma manjši, omogočajo večjo natančnost pri upravljanju z njihovimi kontrolami, nastavitve pa se lahko shranijo, ter ponovno prikličejo. Ob razvoju digitalnega procesiranja je bila kvaliteta zvoka v primerjavi z analognimi napravami slaba. Z napredovanjem digitalne tehnologije, hitrejšim procesiranjem in večjimi prostorskimi zmogljivostmi, pa je kvaliteta bistveno napredovala.

Zvočni inženirji lahko celoten studio zamenjajo s hitrim računalnikom, zvočno karto in nekaj mikrofoni. Računalnik lahko izvaja naloge mešalne mize, obdeluje podatke in jih shranjuje, kar omogoča uporabniku snemanje, miksanje in masteriziranje projektov na enostaven način. Snemalni sistemi, ki bazirajo na računalniku, so danes tudi cenejši od njihovih analognih nadomestkov. Na voljo je veliko zvočnih kart, ki imajo različno število vhodov, izhodov in omogočajo različne vzorčevalne frekvence, z različnimi analogno/digitalnimi pretvorniki. Dodaten pomnilni prostor je tudi na voljo za relativno nizko ceno, kar omogoča enostavno izdelavo zvočnega projekta tako v domačem, kot tudi profesionalnem okolju.

Seveda pa računalnik ne more izvajati teh zahtev, če mu ne podamo navodila kako izvajati naloge v obliki programske opreme. Na voljo je veliko število programov – gostiteljev (angl. *host*), ki na takšen in drugačen način omogočajo manipuliranje z zvočnim signalom. Gostitelj je odgovoren za komunikacijo z zvočno karto in pridobivanje zvočnega podatkovnega toka z nje, procesiranje le tega, shranjevanje v pomnilni prostor in pošiljanje obdelanega signala nazaj v zvočno karto. Prav tako mora zagotavljati uporabniški vmesnik, katerega uporablja zvočni inženir za urejanje vzorčenega signala in opravljanje ostalih nalog pri obdelavi.

Večina gostiteljev je sposobna upravljati z vhodnim signalom, ga urejati in obdelan signal poslati na izhod pri čemer se izognejo sposobnosti dodajanja učinkov in funkcijo procesiranja signalov prepustijo drugemu programu, ki je lahko vključen v gostitelja v obliki vtičnika (angl. *plug-in*). Zvočni vtičnik je majhen program, ki po navadi izvaja samo eno funkcijo iz množice digitalnega procesiranja signalov. Ustvarja lahko različne učinke odmeva (angl.

reverb), popačenja (angl. *distortion*), zakasnitve (angl. *delay*), kompresije... Gostitelj znotraj svojega programskega okna naloži vtičnik in mu posreduje zvočne vzorce. Vtičnik nad vhodnimi vzorci izvede definirano operacijo, modificirane vzorce pa vrne gostitelju. Pri uporabi namenske strojne opreme za generiranje učinkov smo omejeni s številom fizičnih enot, ki jih posedujemo. Z uporabo vtičnikov pa je število njihove uporabe omejeno le na procesorsko moč računalnika. Takšna fleksibilost omogoča zvočnim inženirjem večjo prostost pri izdelovanju željenih učinkov in lažje manipuliranje z zvočnim signali.

1.1 Cilji

Cilji pričujočega diplomskega dela so predstavitev tehnologije zvočnih vtičnikov, njihovega delovanja, lastnosti, zmogljivosti in realizacije.

V naslednjem poglavju bomo opisali nekaj današnjih realizacij zvočnih vtičnikov in njihove lastnosti, v kasnejših poglavjih pa se bomo osredotočili predvsem na tehnologijo podjetja *Steinberg*, *Virtual Studio Technology* – VST, ki je danes ena izmed najpopularnejših in najbolj pogosto uporabljenih. Čeprav je ta tehnologija v svetu zelo razširjena, zanjo še ne obstaja veliko literature. Tako so najboljši viri spletni forumi, samostojni razvijalci programske opreme VST in splošna literatura o digitalnemu procesiranju signalov. Osrednji del diplome vsebuje primere praktične realizacije vtičnika z učinkom zakasnitve (angl. *delay effect*), s katerimi bomo morebitnemu bralcu, ki ga to področje interesira prikazali, na kakšen način je takšno programsko opremo možno izdelati in katera programska orodja lahko uporabimo.

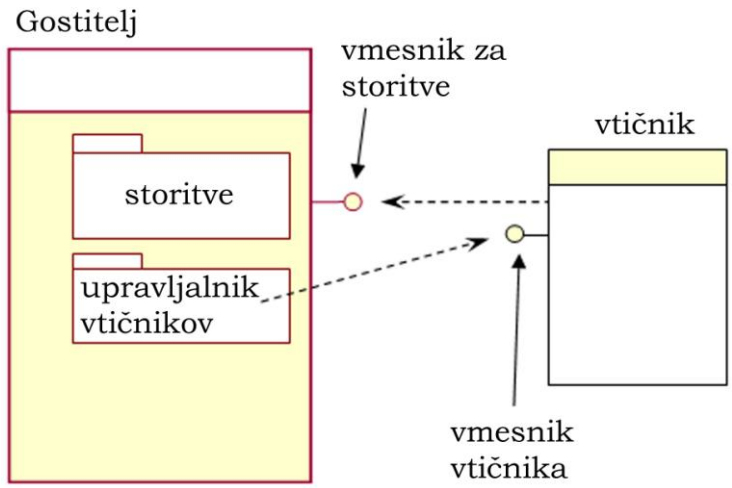
2. Pregled tehnologij

2.1 Definicija vtičnika

Pojem vtičnik (angl. *plug-in*) označuje manjši računalniški program, ki za delovanje potrebuje gostitelja, kateremu razširja funkcionalnost s specifično definirano funkcijo [1].

Razvoj vtičnikov se je začel v sedemdesetih letih prejšnjega stoletja, ko je tekstovni urejevalnik EDT[1], znotraj sebe omogočal zagon manjšega programa za manipulacijo s tekstom.

Iz strani gostitelja deluje vtičnik kot črna skrinjica, kateri gostitelj pošilja podatke, ta pa mu obdelane vrača. Gostitelj zagotavlja storitve, ki jih lahko vtičnik uporablja. Te vključujejo način, kako se vtičnik integrira v gostitelja in protokol za komunikacijo gostitelj – vtičnik. Storitve, ki jih omogoča gostitelj so za delovanje vtičnika zelo pomembne, obratno pa ne velja. Gostitelj lahko nemoteno deluje tudi brez vtičnikov.



Slika 1: Ogrodje vtičnika in povezava z gostiteljem

Namen uporabe vtičnikov in njihove implementacije v aplikacije je različen. Zunanjim razvijalcem je tako omogočeno, da razvijejo dodatne aplikacije, ki razširjajo osnovno funkcionalnost programa in tako nadgradijo programsko opremo z različnimi funkcijami, ki pri osnovnem programu niso bile realizirane. Prav tako takšna modularnost realizacije programske opreme pripomore k zmanjšanju velikosti osnovnega programa in omogoča ločitev izvorne kode vtičnika od aplikacije, zaradi nekompatibilnih licenc programske opreme.

[1] Urejevalnik teksta, ki je temeljil na ukazni vrstici, se uporabljal na operacijskem sistemu *Unisys VS/9* in je tekel na računalniku *UNIVAC 90/60*

Uporaba vtičnikov je razširjena na različnih področjih računalništva. Tako jih srečujemo pri različnih aplikacijah:

- programih za grafično urejanje;
- predvajalnikih za glasbo in video;
- klientih za elektronsko pošto;
- programih za manipuliranje z zvokom;
- orodjih za razvijanje programske opreme;
- spletnih brskalnikih.

Tu je potrebno biti pozoren na razliko med vtičnikom in razširitvijo (angl. *extension*), saj gre pri vtičniku za zunanjo in samostojno komponento, ki ima lasten vmesnik. Razširitev pa se navadno vključi v gostiteljevo ogrodje, s katerim si deli uporabniški vmesnik [1].

2.2 Vtičniki za manipuliranje z zvokom

V tem diplomskem delu se bomo osredotočili samo na vtičnike, ki omogočajo manipuliranje z zvokom in jih lahko uporabimo v digitalnih zvočnih postajah (angl. *digital audio workstation* - DAW). Navadno so modelirani po zgledu analognih naprav uporabljenih v profesionalnih studiih in v virtualnem studiu učinkovito zamenjujejo strojno opremo s programskimi nadomestki. Uporabljajo se lahko za dodajanje zvočnih učinkov pri različnih inštrumentih, katerih signal posnamemo ali predvajamo v realnem času na digitalnih postajah, vokalih, ali sempliranih in sintetiziranih zvokih. Tako lahko vtičnike za manipuliranje z zvokom ločimo na dve vrsti:

- vtičnike, ki vhodnem zvočnem signalu dodajo zvočne učinke (angl. *effect plug-ins*)

  Ta skupina vtičnikov vhodnemu zvočnemu signalu dodaja različne učinke z uporabo DPS algoritmov. Najbolj pogosti učinki so popačenje, zakasitev, frekvenčni izenačevalniki, kompresorji, učinki odmeva, učinka *chorus* in *flanger*, ter ostali. Upravljanje in spreminjanje vrednosti parametrov učinka poteka preko grafičnega uporabniškega vmesnika znotraj digitalne postaje.

Slika 2: Vtičnik za dodajanje učinkov (levo) in vtičniki za proizvajanje zvoka (desno) s pripadujočimi grafičnimi vmesniki

- vtičnike, ki proizvajajo zvok (angl. *instrument plug-ins*)

  Ti vtičniki ne omogočajo obdelovanje vhodnega toka zvoka, temveč ga generirajo. So virtualni programski ekvivalenti analognih naprav za sintetiziranje zvoka. Lahko sintetizirajo ali predvajajo različne zvoke in zvočne učinke. Tudi ti vtičniki imajo virtualni grafični uporabniški vmesnik za upravljanje s parametri. Grafični vmesnik največkrat vsebuje tudi virtualno klaviaturo. S pritiskom na tipko se generira določen ton ali določen zvočni učinek, ki je definiran s strani vtičnika. Dogodke, ki sprožijo generiranje tona imenujemo MIDI[2] dogodki. Poleg dogodka pritiska na klaviaturo, ki določa *note on* MIDI dogodek, obstajajo še ostali, ki se sprožijo ob uporabi kolesa za nastavljanje višine tona, kolesa za modificiranje zvoka in ostali kontrolni dogodki. Virtualni inštrumenti tako omogočajo priključitev zunanje MIDI klaviature, ki pripomore k enostavni uporabi in kontroli virtualnega inštrumenta. Obstaja več skupin vtičnikov za proizvajanje zvoka od enostavnih sintetizatorjev, ki predvajajo že posnete zvoke, do naprednih modularnih digitalnih sintetizatorjev, ki modelirajo zvok po vzoru popularnih analognih sintetizatorjev s povezovanjem različnih modulov. Posebna skupina vtičnikov

[2] MIDI – *Musical Instrument Digital Interface* je protokol na podlagi katerega lahko komunicirajo različne elektronske glasbene naprave. S pomočjo protokola lahko poteka upravljanje, komunikacija in sinhronizacija med različnimi napravami, pri čemer se pošiljajo dogodkovna sporočila. [5]

so virtualni ritem stroji (angl. *drum machines*), ki so posebni programski vzorčevalniki zvoka tolkal.

Slika 3: Popularni analogni sintetizator MiniMoog (levo) in njegov digitalni ekvivalent v obliki vtičnika VSTi (desno)

Zvočni vtičniki za delovanje potrebujejo digitalno postajo, ki omogoča upravljanje z zvokom. V njej je možno predvajati posnete, vzorčene in sintetizirane zvoke, snemati zvok in ga urejati. Digitalno postajo sestavljajo računalnik, analogno-digitalni in digitalno-analogni pretvornik, ter digitalni zvočni urejevalnik. Računalnik omogoča povezavo z zvočno karto, skrbi za poganjanje programske opreme in zagotavlja procesorsko moč za urejanje zvoka [2]. Zvočna karta je vmesnik, ki omogoča obdelovanje zvoka. Pretvarja analogni zvočni signal v digitalno obliko in sodeluje pri procesiranju zvoka. Pri predvajanju zvoka, pa pretvarja digitalni signal v analognega.

Del digitalne postaje s katerim ima uporabnik največ stika je digitalni zvočni urejevalnik. Uporabniški vmesnik digitalnega zvočnega urejevalnika večinoma posnema večkanalne analogne snemalnike na magnetni trak. Osnovno programsko okno lahko vsebuje več kanalov, katerega je vsakega posebej mogoče upravljati, število njih pa ni fiksno kot je to pri analognih snemalnikih. Taki urejevalniki se imenujejo večstezni (angl. *multitrack*), medtem ko so urejevalniki, ki podpirajo samo en zvočni kanal – enostezni (angl. *singletrack*). Slika 4 prikazuje nekaj danes popularnih zvočnih urejevalnikov, ki podpirajo večsteznost in uporabo vtičnikov. Na vsako zvočno stezo lahko ali posnamemo, ali uvozimo zvočni zapis, katerega je moč obdelovati s funkcijami, ki jih omogoča urejevalnik. Zvočni vtičnik lahko uporabimo na vsaki stezi ali množici stez, na katere lahko vstavljamo ali snemamo zvok.

Slika 4: Primeri večsteznih zvočnih urejevalnikov - od zgoraj navzdol: Cubase SX, Logic Pro, Pro Tools

2.2.1 Vrste zvočnih vtičnikov glede na realizacijo

Vsak vtičnik je program sestavljen iz programske kode za manipulacijo nad podatki in je lahko realiziran kot modul za procesiranje zvoka ali kot modul za sintezo zvoka. Glede na to kje poganjamo te module ločimo dva tipa vtičnikov [3]:

- vtičniki z namensko platformo (angl. *platform-based plug-ins* ali *dedicated DSP plug-ins*):

  Tu poteka procesiranje, ki ga zahteva vtičnik, s pomočjo namenskega vezja za digitalno procesiranje signalov. Tako imamo posebno DPS karto, navadno priključeno z vodilom PCI ali PCIe na digitalno postajo. Moč procesorja digitalne postaje za poganjanje vtičnikov ni pomembna. Hitrost procesiranja vtičnika je odvisna samo od zmogljivosti DPS vezja na karti.

- vtičniki, ki uporabljajo procesor digitalne postaje (angl. *native plug-ins*):

  Število in zahtevnost vtičnikov je tu pogojena s procesorsko močjo delovne postaje. Vsi vtičniki, ki tečejo v določenem trenutku na digitalni postaji, si tako delijo procesorsko moč.

Slika 5: Primer vtičnika z namensko platformo in njegove PCIe karte (desno), PCI karta z DPS vezjem (levo)

Uporaba namenskih platform poveča moč in zanesljivost sistema, vendar so vtičniki z namensko platformo dražji od vtičnikov, ki uporabljajo procesor digitalne postaje. Dolgo je veljalo, da so sistemi z namenskimi platformami boljši od *native* sistemov, vendar so dandanes te razlike skoraj izginile, zaradi vse hitrejših in zmogljivejših procesorjev. V strogo profesionalnih studiih je uporaba DPS vtičnikov precej razširjena, v manjših, pa predvsem zaradi nižje cene prevladujejo »*native*« vtičniki.

Kakovost reproduciranega zvoka je lahko pri obeh vrstah vtičnikov enaka in je odvisna od kakovosti spisanega algoritma vtičnika. Vendar lahko z uporabo *native* vtičnikov z zahtevnim algoritmom hitro prekoračimo zmogljivosti računalnika.

Z zornega kota končnega uporabnika se sistema po zunanjem izgledu ne razlikujeta. Končni uporabnik ima vedno opravka z grafičnim uporabniškim vmesnikom znotraj gostitelja. Ta je lahko enak pri obeh implementacijah vtičnikov.

2.2.2 Vtičniki glede na postavitev v digitalnem zvočnem urejevalniku

Glede na pozicijo, kamor v digitalnem urejevalniku postavimo vtičnik, t.j. na katero stezo ga postavimo, definiramo tri tipe vtičnikov [4]:

- vtičniki za posamezne steze (angl. *track based plug-in*)

  Učinek, ki ga proizvaja vtičnik vpliva samo na stezo na katero je vstavljen. V terminologiji zvočnega inženiringa se za takšne vtičnike uporablja izraz *insert plug-in.* Najbolj pogost primer takšnega vtičnika so kompresorji, saj morajo biti za pravilno delovanje nameščeni na vsak kanal posebej. Značilni predstavniki so še izenačevalniki, učinki zakasnitve, učinka *chorus* in *flanger*.

Steze na katerih so vokali, pa lahko izboljšamo z uporabo *de-esserjev*, omejevalnikov in učinkov za avtomatsko uglaševanje.

- vtičniki za vodila (angl. *bus based plug-in*)

  Tu gre za vtičnike, ki jih lahko uporabljamo na večih stezah simultano, ne vplivajo pa na celotno zvočno sliko. Imenujemo jih *send plug-ins.* Takšno uporabo lahko realiziramo z uporabo vodila, kateremu steze pošiljajo signale, na vodilo pa vstavimo vtičnik z določenim učinkom. Tipični predstavnik je učinek odmeva. Z uporabo istega učinka odmeva na različnih stezah pridobimo pri dimenziji zvočne slike – dobimo občutek, da so vsi zvoki zaigrani in posneti v isti sobi

- vtičniki za glavni kanal (angl. *mastering plug-in*)

  Vtičniki za glavni kanal vstavimo na glavni izhodni kanal. Takšen vtičnik vpliva na celoten zvok, ki ga obdelujemo v digitalnem zvočnem urejevalniku. So najbolj pomembni za zvok končnega izdelka.

2.2.3 Vtičniki glede na delovanje v realnem času

To delitev lahko naredimo samo pri vtičnikih, ki lahko lahko ustvarjajo zvočne učinke. Ločimo [3]:

- vtičnike, ki delujejo v realnem času (angl. *real time plug-ins*)

  Najbolj pogosto uporabljena skupina vtičnikov. Omogočajo uporabo vtičnikov v realnem času. Zvok in njegove modifikacije, ki jih naredi uporabnik preko grafičnega uporabniškega vmesnika, so slišne takoj po spremembi atributov.

- vtičnike z zakasnitvijo (angl. *recalculation plug-ins*)

  Ti vtičniki ne delujejo v realnem času. Vtičnik sprejme vhodne podatke, nad njimi naredi definirano operacijo in kreira novo datoteko s spremenjenimi podatki. Novo kreirana datoteka nadomesti prejšnjo, ki pa ni izbrisana, ampak se shrani in je možna ponovna uporaba. Takšen tip vtičnikov omogoča prihranke pri uporabi procesorske moči.

2.2.4 Formati zvočnih vtičnikov

Obstaja veliko formatov zvočnih vtičnikov. Podpora formatom je pri različnih zvočnih urejevalnikih različna. Najbolj poznan in razširjen je standard podjetja *Steinberg*, V*irtual StudioTechnology* (VST). Ta je široko podprt v različnih zvočnih urejevalnikih in omogoča izdelavo vtičnikov za različne operacijske sisteme. Standardu VST sledi Microsoftova tehnologija vtičnikov *DirectX* (DX), ki pa je podprta samo na sistemih *Windows*. Ekvivalent podjetja *Apple Computer* je standard *Audio-Units* (AU), ki je namenjen uporabi na operacijskem sistemu *Mac* OS X. Podjetje MOTU je razvilo svojo arhitekturo vtičnikov imenovano MAS, ki je tudi na voljo ekskluzivno na platformi *Mac*. Tu je potem še podjetje

DigiDesign, ki je začelo orati ledino na področju digitalnega procesiranja signalov v realnem času z formatom TDM, kasneje pa še RTAS in AudioSuite. Na strani Linuxovih prostokodnih predstavnikov so tehnologije LADSPA, njegov sorodnik DSSI in njun najnovejši naslednik LV2.

To so danes vse najbolj uporabljane tehnologije zvočnih vtičnikov, ki omogočajo dodajanje zvočnih učinkov. Generiranje zvoka oz. kreiranje virtualnih inštrumentov, pa ne podpirajo vse tehnologije. Tako se tehnologija LADSPA uporablja samo za ustvarjanje učinkov, medtem ko generiranje zvoka podpira njegov sorodnik DSSI. Pomembno je tudi dejstvo, da vsak format vtičnikov deluje v vseh gostiteljih, ki podpirajo njegovo tehnologijo. Tako na primer *DirectX* vtičnik, ki je bil kupljen skupaj z zvočnem urejevalnikom *Sonar*, deluje tudi v vseh drugih zvočnih urejevalnikih, ki podpirajo tehnologijo *DirectX*, kot so urejevalniki *Acid, SoundForge* ali *Vegas*.

Čeprav praktični primer realizacije vtičnika v poglavju 3 temelji na Steinbergovi tehnologiji VST, bom v naslednjih podpoglavjih opisal večino zgoraj omenjenih arhitektur, kar naj potencialnemu bralcu, ki se raziskuje svet zvočnih vtičnikov, služi za referenco in pomoč pri izbiri potrebovane arhitekture.

2.2.4.1 VST – Virtual Studio Technology

Slika 6: Logo vtičnikov VST verzije 2.x in verzije 3.x

VST ali *Virtual Studio Technology* je odprt de-facto standard podjetja *Steinberg Media Technologies* GmbH za izdelavo zvočnih vtičnikov. Projekt, začet leta 1996 je danes eden najbolj razširjenih formatov tehnologije vtičnikov. V prvi različici je bila tehnologija vključena v njihov poznan zvočni urejevalnik pod imenom *Cubase* VST. To je bila prva programska oprema, ki je uprabljala procesor gostiteljevega računalnika in podpirala uporabo učinkov v realnem času. Tekla je na *Apple-ovem Machintoshu*, ki je omogočal poganjanje 24 stez v realnem času in neomejno število MIDI stez. Leta 1997 so predstavili tehnologijo VST na osebnem računalniku. Tehnologiji VST in ASIO[3], pa sta postala odprta standarda, kar je omogočilo zunanjim razvijalcem začetek razvoja lastnih vtičnikov in zvočne opreme. Verzija VST vtičnikov 2.0 in gonilnikov ASIO 2.0, izdana leta 1999, sta predstavljala prelomno leto, zaradi uvedbe virtualnih inštrumentov v okolje VST z ozanko VSTi. Čeprav so leta 2008 predstavili za sedaj zadnjo generacijo vtičniko VST 3.0, pa je popularnost verzije 2.4 še vedno zelo visoka. To je predvsem zaradi večjega obsega podpore v različnih zvočnih urejevalnikih, ki pa je verzija 3.0 še nima in relativno majhnih prednosti nove tehnologije v primerjavi s starejšo različico.

[3] *Audio Stream Input/Output* (ASIO) je protokol gonilnika zvočne karte, ki ga je naredilo podjetje Steinberg in omogoča direktno komunikacijo med programsko opremo in zvočno karto pri nizkih latencah in visoko odzivnostjo. [6]

V splošnem vtičnik VST sprejema vhodni zvočni tok, nad njim opravi algoritem za dodajanje učinkov in rezultat vrne gostitelju. To operacijo naredi z uporabo procesorja delovne postaje in ne potrebuje dodatnih, namenskih platform za procesiranje. Vhodni zvočni tok je razdeljen na bloke, ki jih gostitelj pošilja vtičniku v zaporedju. Gostitelj in njegovo okolje določata velikost blokov. Vtičnik VST upravlja z vrednostmi lastnih parametrov. Tako gostitelj nima informacij o tem kako je vtičnik obdelal zadnji podatkovni blok, ki mu ga je poslal. Ob ustreznih sposobnostih gostitelja je omogočeno le, da ta avtomatsko spreminja parametre vtičnika. Temu pravimo, da ima parameter sposobnost avtomatizacije.

Tehnologija je podprta na več operacijskih sistemih in nastopa v obliki dinamične povezane knjižnice. V okolju *Windows* je vtičnik VST večnitna dinamično povezana knjižnica s končnico »*.dll*« (verzija 2.x), medtem ko ima verzija VST 3.x končnico »*.vst3*«. V operacijskem sistemu *Mac* OS, pa nastopa kot paket programske opreme s končnico »*.vst*«. Tu je potrebno omeniti, da vtičnike VST narejene za operacijski sistem *Windows* ni mogoče poganjati na *Mac OS* in obratno.

Programska oprema, ki podpira vtičnike VST/i je zelo obširna, predvsem zaradi dolge in uspešne zgodovine te tehnologije. Izmed množice so najbolj popularni digitalni urejevalniki *Ableton Live, Reaper, Audacity, Cubase, FL Studio* in *Sonar*. Možna je njihova uporaba tudi v nekaterih digitalnih urejevalnikih, ki ne podpirajo tehnologije VST. To dosežemo z uporabo posebnih pretvornikov – t.i. *wraperjev*, ki omogočajo pretvorbo enega formata vtičnikov v drugega. Tako na primer obstajajo pretvorniki, ki omogočijo uporabo tehnologije VST v gostiteljih, ki podpirajo samo tehnologijo *AudioUnits*, ali uporabo VST vtičnikov v gostiteljih, ki podpirajo samo RTAS vtičnike. Seveda obstajajo pretvorniki, ki omogočajo tudi obratno pretvarjanje. Velja pa poudariti, da pretvorba ne vključuje platformne prenosljivosti.

Za vtičnike VST obstajajo tudi posebni strojni gostitelji, na katerih lahko poganjamo posebne izvedbe vtičnikov. Tu gre za vtičnike z namensko platformo, ki so prenosljivi in se lahko v osnovi uporabljajo tudi brez računalnika, vendar je za urejanje in manipuliranje z zvokom še vedno potreben dodaten računalnik. Vse procesiranje nad vhodnim zvočnim tokom tako prevzame namenska platforma, ter tako pripomore k zmanjšanju porabe procesorske in pomnilne moči digitalne zvočne postaje. Predstavniki namenskih platform za poganjanje vtičnikov VST so *Plugzilla* (*Manifold Labs*) in *Receptor* (*Muse Research*).

Zaradi odprtega standarda je možen razvoj vtičnikov tudi zunanjim razvijalcem. Razvoj poteka s pomočjo *Steinbergovega* VST SDK (angl. *standard developement kit*), ki vsebuje množico razredov napisanih v C++ programskem jeziku in vsebujejo funkcije za manipuliranje z zvokom. Trenutno obstajata dve različici SDK-ja – prva je za razvijanje vtičnikov verzije VST 2.4, druga pa za najnovejšo verzijo vtičnikov VST 3. Poleg osnovnega razvojnega okolja, obstaja tudi nekaj prevedb le tega v druge programske jezike. Tako obstaja Delphi [7] različica razvojnega okolja, Java verzija [8] in .Net implementacija [9]. Za razvoj grafičnih uporabniških vmesnikov je na voljo dodatno razvojno okolje, imenovano VST-GUI. Prav tako vsebuje množico C++ razredov, ki se lahko uporabijo za gradnjo uporabniških vmesnikov po meri, ki lahko vsebujejo razrede za upravljanje z gumbi, drsniki, preklopniki,

prikazovalniki in ostalim. Razredi vsebujejo samo funkcije, ki omogočajo gradnjo uporabniških vmesnikov, medtem ko mora za izdelavo grafike poskrbeti razvijalec sam.

2.2.4.2 DirectX

DirectX ali krajše DX je arhitektura za kreiranje vtičnikov podjetja *Microsoft*, ki omogoča obdelavo zvočnih podatkovnih tokov v realnem času.

Prav tako kot predhodnik je tudi arhitektura DX odprt standard, ki omogočajo povezavo zvočnih sintetizatorjev in učinkov v zvočne urejevalnike. Vtičniki *DirectX t*emeljijo na Microsoftovem komponentnem objektnem modelu (angl. *component object model* – COM), ki je binarni vmesnik za medprocesno komunikacijo in skrbi za komunikacijo med programskimi komponentami. Arhitekturna zgradba vtičnikov pa temelji na večpredstavnostnem ogrodju in vmesniku za programiranje *DirectShow*. Ta omogoča izvajanje različnih operacij nad večpredstavnostnimi datotekami in tokovi [10].

Vtičniki DX podpirajo tako uporabo vtičnikov z lastnostmi generiranja učinkov (DX), kot vtičnikov, ki generirajo zvok – inštrumentov (DXi). Vtičnike DXi je razvilo podjetje *Cakewalk* v sodelovanju z *Microsoftom*.

Slaba stran vtičnikov DX v primerjavi z vtičniki VST je platformna neprenosljivost. Podprti so le v okolju *Windows*. Podobno kot pri standardu VST, pa obstajajo pretvorniki, ki omogočajo uporabo vtičnikov DX v gostiteljih s podporo vtičnikom VST. Digitalni urejevalniki, ki podpirajo uporabo vtičnikov *DirectX* so *Cakewalk Pro Audio, Sound Forge, Wavelab, Acid* in drugi.

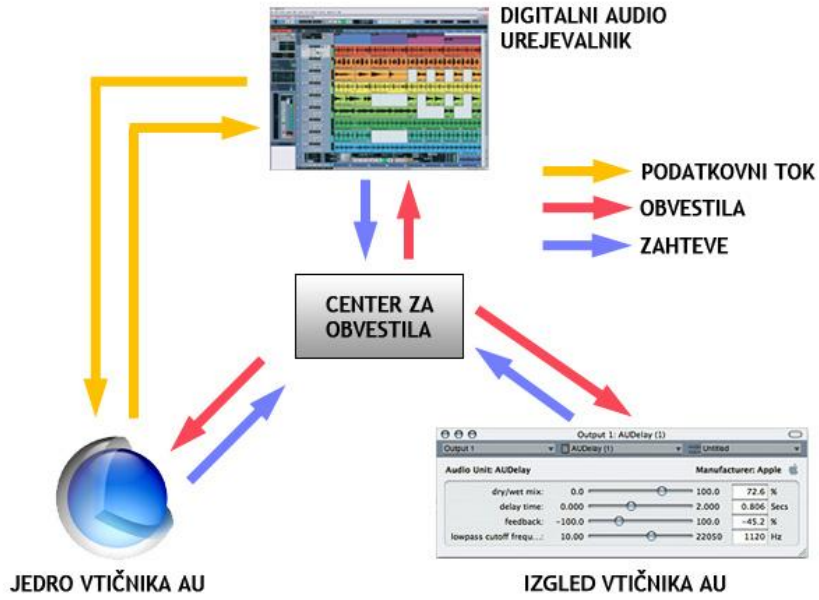
Razvoj vtičnikov *DirectX* poteka s pomočjo razvojnega okolja Microsoft *DirectX SDK* oz. njegove podmnožice *DirectX Audio*. Kot nadgradnjo tega razvojnega okolja je podjetje Sony izdalo Sony *Pictures Digital Plug-in Development Kit* – PIDK [11], ki razširja osnovni SDK. Poudarek daje na digitalnem procesiranju signalov, ki je predstavljeno v zbirki primerov in izvornih kod. Najbolj preprosta je implementacija vtičnikov z uporabo okolja DX *Plug-in Wizard* podjetja *Cakewalk*, ki omogoča enostavno kreiranje vtičnikov s pomočjo ogrodja Microsoft *Visual C++* v6.0. Obstaja tudi ekvivalent za osnovni SDK v programskem jeziku Delphi [12].

2.2.4.3 Audio Units

Slika 7: Logo vtičnikov AU podjetja Apple

Audio Units – AU so digitalni zvočni vtičniki, ki temeljijo na tehnologiji podjetja Apple, *Core Audio,* ki je temeljni del operacijskega sistem *Mac* OS X. Tako je tudi platformna vezanost uporabe vtičnikov AU omejena na *Mac* OS X.

Vtičniki v operacijskem sistemu *Mac* OS X nastopajo kot paketi programske opreme (angl. *bundle*), ki je definiran kot komponenta[4]. Tako kot predhodnika, tudi AU podpirajo zvočne učinke in inštrumente. Vsaka vtičnik AU je navznoter sestavljen iz dveh delov, ki jih prikazuje slika 8. Procesiranje zvočnih tokov omogoča jedro zvočne enote, izgled zvočne enote (angl. *audio unit view*) pa zagotavlja grafični uporabniški, preko katerega lahko uporabnik spreminja vrednosti parametrov vtičnika. Pri kreiranju AU vtičnika sta navadno oba dela vsebovana v enem paketu programske opreme, vendar sta logično ločena dela programske kode [13].



Slika 8: Komunikacija med vtičnikom AU in gostiteljem

Jedro vtičnika AU, njegov izgled in gostitelj med seboj komunicirajo s pomočjo centra za obvestila, ki je vzpostavljen s strani gostitelja. Center omogoča sinhronizacijo vseh treh delov.

Zaradi platformne vezanosti je uporaba vtičnikov AU možna v digitalnih urejevalnikih, ki tečejo na *Mac* OS X sistemu. Najbolj pogosti so *GarageBand, Logic Pro, Soundtrack Pro, Final Cut Pro* in *Ableton Live.*

Implementacija novih vtičnikov AU poteka s pomočjo *Core Audio* SDK razvojnega okolja.

2.2.4.4 TDM – Time Division Multiplexing

TDM so vtičniki z namensko platformo podjetja *Digidesign* ustvarjeni za njihov digitalni urejevalnik *Pro-Tools.*

V začetku osemdesetih let je podjetje *Digidesign* z uvedbo večsteznega sistema *Pro-Tools,* začelo s procesiranjem zvočnih podatkovnih tokov v realnem času z uporabo namenskih procesorjev za digitalno procesiranje signalov. Tu gre za uporabo posebnih DPS kart na

[4] Komponenta v Mac OS X je skupek kode, ki ostalim klientom zagotavlja izvajanje določene vrste storitve. Aplikacije, sistemske razširitve ali tudi druge komponente lahko uporabljajo storitve komponente.

vodilu PCI, ki jih fizično vstavimo v gostiteljev računalnik. Karte imajo poseben DPS RISC procesor, ki omogoča hitro procesiranje vhodnih zvočnih tokov. To so tako imenovane *»Digidesign farm«* karte. Število različnih vtičnikov je tako omejeno na število TDM kart, ki so vstavljene v sistem. Pri tehnologiji TDM gre za prepletanje večih digitalnih zvočnih signalov (ali tokov) na enem podatkovnem vodilu. To vodilo je fizično ločeno od sistemskega vodila in povezuje več posameznih kart TDM. Vsako TDM vodilo omogoča do 512 zvočnih kanalov, vsak z 24-bitno resolucijo podatkov [14]. Ti podatkovni tokovi tako tvorijo podatkovno pot znotraj virtualne mešalne mize.

Danes je tehnologija vtičnikov TDM odprt standard. Tako je omogočeno zunanjim razvijalcem razvoj vtičnikov, ki temeljijo na tehnologiji TDM, vendar teh implementacij ni veliko zaradi specifike realizacije vtičnikov z namenskimi platformami. Prav tako je omejitev pri uporabi vtičnikov TDM, saj so vezani na uporabo v digitalnih urejevalnikih *Pro-Tools* in *Logic Pro 9.*

Formati vtičnikov TDM so danes, v primerjavi z drugimi formati, manj prisotni. Predvsem cena tehnologije TDM je tisti parameter, ki povprečnega uporabnika odvrne od nakupa. Seveda pa se v najboljših, profesionalnih studiih poslužujejo te tehnologije, ker so njihove potrebe po strojni moči veliko večje.

Glede na današnji razvoj strojne opreme in večanjem števila procesnih jeder menim, da bo vlogo namenskega DPS procesorja kmalu lahko kompletno prevzel večjedrni procesor gostitelja, pri čemer ne bo prišlo do pomanjkanja procesorske moči ali drugih izgub v performansah.

2.2.4.5 RTAS – Real Time Audio Suite

Slika 9: Logo vtičnikov RTAS

Format vtičnikov podjetja *Digidesign,* ki uporabljajo procesor delovne postaje – so »*native*« vtičniki. Tako je uporaba vtičnikov RTAS je omejena s procesorsko močjo gostitelja.

Format RTAS je naslednik formata AS – *Audio Suite*, ki ni omogočal procesiranja v realnem času. To pomanjkljivost so nadgradili z novo različico vtičnikov RTAS. Uporabljajo se lahko znotraj sistema *Pro-Tools*, kot učinki ali inštrumenti.

2.2.4.6 MAS – MOTU Audio System

Format vtičnikov, ki omogočajo delovanje v realnem času in »*native*« načinu delovanja. So blagovna znamka podjetja *Mark of the Unicorn,* ki so bili narejeni predvsem kot format vtičnikov za lastni digitalni urejevalnik *Digital Performer* in *AudioDesk.*

2.2.4.7 LADSPA – Linux Audio Developers Simple Plug-in API

LADSPA je standard za modeliranje učinkov in zvočnih filtrov, ki je bil v osnovi kreiran za operacijski sistem Linux, vendar deluje tudi na drugih platformah. Standard je bil izdan na podlagi prostokodne licence LGPL (GNU Lesser General Public License), kar omogoča razvijalcem prosto spreminjanje in redistribucijo izvorne kode.

Ideja za razvoj prostokodnega standarda se je rodila na poštnih seznamih razvijalcev programske opreme za manipuliranje z zvokom v okolju Linux. Pri razvoju so se opirali na načela funkcionalnosti, intuitivnega razvoja, uporabe in kompatibilnosti, pri čemer naj bi bil poudarek na preprostosti, kot to pove že ime samega standarda. Tako je izpuščeno nekaj funkcij, ki jih srečamo pri drugih formatih vtičnikov. S standardom LADSPA lahko kreiramo le vtičnike, ki omogočajo dodajanje učinkov. Kreiranje zvoka je omogočeno s standardom DSSI, ki je nadgradnja standarda LADSPA. Preprostost standarda nam prav tako omogoča lahko vgrajevanje vtičnikov v gostitelje. Tako obstaja kar nekaj gostiteljev, v katerih lahko poganjamo vtičnike standarda LADSPA. Izmed mnogih so pomembnejši *Audacity*, *Ardour* in *Sweep.*

Razvoj vtičnikov poteka s pomočjo standardnega razvojnega okolja, ki vsebuje datoteko, ki nastopa kot vmesnik za programiranje in je osnovna datoteka, ki omogoča programiranje vtičnikov LADSPA.

2.2.4.8 DSSI – Disposable Soft Synth Interface

Kot že rečeno gre pri standardu DSSI za nadgradnjo standarda LADSPA. Tako vtičniki DSSI omogočajo generiranje zvoka na podlagi notnih dogodkov.

Tako je od standarda LADSPA podedoval licenco prostokodnosti, kot namembnost uporabe predvsem v gostiteljih, ki tečejo na operacijskem sistemu Linux, čeprav je možna uporaba tudi na drugih platformah. Standard LADSPA razširja z uvedbo odzivanja na notne dogodke, uporabe vnaprej definiranih programskih nastavitev in metodo za uporabo lastnih uporabniških vmesnikov.

2.2.4.9 LV2

Za odprti standard LV2 lahko rečemo, da je druga verzija standarda LADSPA. Nadgrajuje pomanjkljivosti, ki jih vsebuje LADSPA. Zasnovan je bil z možnostjo nadgrajevanja in razširjanja. Lahko dodajajo zvočne učinke, kot generirajo zvok in omogočajo delovanje v realnem času.

Vtičniki standarda LV2 nastopajo kot paketi programske opreme, ki vsebujejo knjižnice s programsko kodo vtičnika in tekstovne datoteke za opis vtičnika.

3. Implementacija vtičnikov VST

V prejšnjih poglavjih smo si pogledali vtičnike na splošno in njihove raznolike formate. V naslednjih poglavjih, pa se bomo osredotočili na primere implementacije vtičnikov Steinbergovega formata VST. Za ta format sem se odločil zaradi njegove popularnosti in razširjenosti uporabe.

V prvem delu poglavja se bomo posvetili izgradnji vtičnikov, verzije VST 3, s pomočjo razvojnega okolja SDK podjetja *Steinberg*, ki omogoča najbolj osnovni način implementacije vtičnikov, pri čemer pa ne mislimo najlažji. Z uporabo razredov znotraj razvojnega okolja implementiramo vtičnike z uporabo objektnega programskega jezika C++. Z uporabo ustreznih DPS algoritmov pa obdelujemo vhodne zvočne tokove. V zadnjih dveh poglavjih, pa si bomo pogledali realizacijo vtičnikov, verzije VST 2.4, s pomočjo modularnih programov, ki omogočajo gradnjo vtičnikov s pomočjo vizualnega programiranja. Ti programi temeljijo na vizualnem sestavljanju modulov, pri čemer ima vsak modul svojo funkcionalnost.

V vseh primerih bo šlo za modeliranje vtičnika z učinkom zakasnitve.

3.1 Učinek zakasnitve

Ideja za realizacijo vtičnika z učinkom zakasnitve je nastala kot nadgradnja projekta pri predmetu seminar, kjer sem realiziral učinek zakasnitve v obliki analognega kitarskega efekta. Ideja je bila, da se sedaj ta problem prenese in realizira v digitalni obliki. Tako je nastala tudi ideja o diplomski nalogi in raziskovanju sveta zvočnih vtičnikov.

Učinke zakasnitve srečujemo v različnih okoljih ali prostorih, ki imajo akustične lastnosti. Vsi ti sistemi vsebujejo komponente, od katerih se zvočni valovi odbijajo, lomijo. Tako razlikujemo zakasnitve, ki nastajajo na podlagi oddaljenosti površin, števila površin od katerih se lahko signali odbijajo, ukrivljenosti odbijalnih površin in različne postavitve izvornega signala. Najbolj poznan je odboj od zelo oddaljene površine, kjer je čas zakasnitve daljši od sto milisekund, pri čemer dobimo učinek odmeva. Zaradi oddaljenosti je čas odboja večji, zato je učinek zakasnitve dobro slišen. Vsi smo tak preizkus naredili tudi že v praksi, v kakšni alpski dolini, kjer se je odboj izvornega krika dobro slišal šele čez krajši čas. V manjših zaprtih prostorih lahko pride do večkratnih odbojev signala od večih odbojnih površin. Pri krajših razdaljah med izvorom in odbojno površino, pa pride do spremembe pri barvi zvoka. Tu gre za zakasnitve manjše od desetih milisekund.

S posnemanjem teh naravnih pojavov lahko v digitalnem svetu dodajamo zvokom nove dimenzije. Na glasbenem področju je učinek zakasnitve zelo razširjen in veliko uporabljen pri skoraj vseh možnih inštrumentih in vokalih. Z uporabo učinka zakasnitve postane zvok bolj poln in tako celotna glasbena kompozicija, kot bi rekli glasbeniki, »zaživi«.

3.1.1 Krožni pomnilnik

V digitalnem svetu lahko učinke zakasnitve modeliramo s t.i. digitalnimi zakasnitvenimi linijami (angl. *digital delay line* = DDL). Zakasnitvena linija dolžine *m* vzorcev je lahko implementirana s krožnim pomnilnikom, ki vsebuje množico zaporednih pomnilnih celic dolžine *M*. V vsako celico se lahko piše s kazalcem *IN* in bere s kazalcem *OUT*. Pri vsakem vzorčenju vhodnega signala se piše v pomnilnik na mesto, ki ga določa kazalec *IN* [15]:

$$IN = (OUT + m) \,\%\, M \tag{1}$$

V zgornji enačbi operacija % predstavlja operacijo po modulu *M*. Vrednost zakasnjenega signala se bere iz pomnilne celice, na katero kaže *OUT*, oba kazalca pa dobita novo vrednost po enačbah [15]:

$$IN = (IN + 1) \,\%\, M \tag{2}$$

$$OUT = (OUT + 1) \,\%\, M \tag{3}$$

Torej se kazalca povečujeta za eno za vsak vhodni vzorec, pri čemer upoštevata krožno lastnost pomnilnika. Dolžina zakasnitve je tako določena s spremenljivko *m*. Maksimalni čas zakasnitve, ki ga lahko dobimo, pa določa dolžina krožnega pomnilnika *M*.

3.1.2 KEO *comb* filter

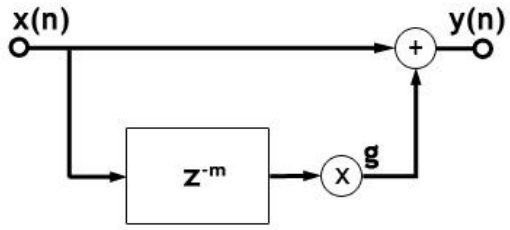
Najenostavnejši model zakasnitve je primer pri katerem pride do le enega oboja. Shema, ki simulira takšno okolje se imenuje filter s končnim enotinim odzivom - KEO (angl. *finite impulse response* = FIR). Zakasnitveni model lahko konstruiramo z zakasnitveno linijo, ki osnovni signal zakasni za *m* vzorcev. Zakasnjeni signal pomnožimo z vrednostjo koeficienta dušenja *g*, ter prištejemo osnovnemu signalu. Omenjeni model lahko opišemo z diferenčno enačbo [15]:

$$y(n) = x(n) + g * x(n - m) \tag{4}$$

Prenosna funkcija je definirana z enačbo:

$$H(z) = 1 + g * z^{-m} \tag{5}$$

Časovni odziv takega filtra je tako sestavljen iz osnovnega signala in njegove zakasnjene ekvivalence. Na podlagi prenosne funkcije (5) lahko naredimo shemo sistema KEO *comb* filtra, ki jo prikazuje slika 10 [16].



Slika 10: Shema KEO comb filtra

Kot je značilno za naravne akustične zakasnitve, tudi KEO *comb* filter vpliva na časovne in frekvenčne značilnosti vhodnega signala. Pri večjih zakasnitvah – večjih vrednostih spremenljivke *m*, lahko slišimo zakasnjen vhodni signal. Spektralni učinek takega filtra ne zaznamo, zaradi premajhne oddaljenosti frekvenc, ki jih filter ojači. Pri manjših vrednostih *m,* so časovni dogodki zelo blizu, celo tako blizu, da jih naše uho med seboj ne loči, jasno pa se sliši spektralni učinek takega filtra.

3.1.3 NEO *comb* filter

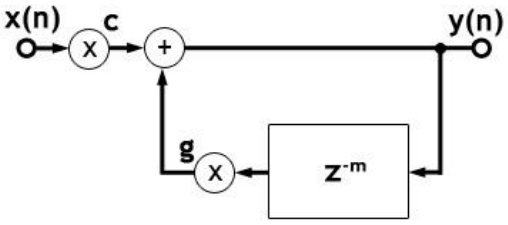
NEO *comb* filter ali filter z neskončnim enotinim odzivom - NEO (angl. *ifinite impulse response* = IRR) simulira zakasnitve, pri katerih pride do več odbojev. Pri prehodu signala preko NEO *comb* filtra se tako ustvari neskončno zaporedje odzivov na izhodu *y(n),* glede na vhod signal *x(n)*. Zgrajen je iz zakasnitvene linije dolžine *m* vzorcev in povratne ponovitvene zanke (angl. *feedback loop*) s koeficientom dušenja *g*. Tako lahko definiramo diferencialno enačbo NEO *comb* filtra [16]:

$$y(n) = c * x(n) + g * y(n-m) \tag{6}$$

Iz nje lahko izpeljemo prenosno funkcijo:

$$H(z) = \frac{c}{(1 - g * z^{-m})} \tag{7}$$

Zaradi povratne povezave lahko pride do visokega ojačanja signala, zato je potrebno vsebnost vhodnega signala na izhodu regulirati s spremenljivko *c*. Shemo NEO *comb* filtra predstavlja slika 11.



Slika 11: Shema NEO comb filtra

Zaradi povratne zanke je časovni odziv takega sistema neskončen.

3.1.4 Tipi učinkov zakasnitev

Vsak tip učinka zakasnitve je v osnovi za določen čas zakasnjen osnovni signal. Zakasnitveni čas je lahko dolg od nekaj milisekund do nekaj sekund. Če je zakasnitveni čas dolg samo nekaj milisekund, naše uho ne bo zaznalo zakasnitve, ampak spremembo v barvi zvoka glede na osnovni signal. Gre za pojav, ki smo ga opisali pri KEO *comb* filtrih.

Pri zakasnitvah brez povratne zanke, ki trajajo od 50 do 100 milisekund govorimo o učinku podvajanja (angl. *doubling*) ali učinku zakasnitve, imenovanem *slap-back delay* [17]. Z uporabo takšnih zakasnitev dobimo bolj poln zvok. Največkrat se takšen učinek uporablja pri nadgrajevanju zvoka kitare ali vokalov. Učinek podvajanja vokalov se naredi tudi tako, da pevec dvakrat posname isti del in rezultat simultano predvaja. Rezultat takšnega načina je veliko bolj naraven zvok.

O tipu zakasnitve imenovanem odmev smo že govorili. To so enojne zakasnitve daljše od 100 milisekund.

Bolj uporabljeni od enojnih zakasnitev so učinki z uporabo zakasnitvene zanke. Pri njih je določen del zakasnitvenega izhoda vezan nazaj na vhod. Z dolgimi časi zakasnitve, zakasnitvena zanka kreira ponavljajoče učinke odmeva. Pozornost velja obrniti na dolžino trajanja zakasnitve. Če so časi trajanja ponovitev predolgi, zakasnjen signal postaja vse glasnejši, kar pa lahko pripelje tudi do poškodbe zvočnikov ali sluha.

Posebej zanimive so zakasnitve, ki omogočajo procesiranje učinka zakasnitve v stereo načinu. Takim učinkom zakasnitvam pravimo stereo zakasnitve. Stereo zakasnitve so največkrat uporabljene z dolgimi zakasnitvenimi časi za kreiranje dinamike in zanimivih stereo ritmičnih efektov. Tak učinek ima dve ločeni zakasnitveni liniji in zanju ločene kontrole za levi in desni zvočni kanal. Različica stereo zakasnitve je t.i. *ping-pong delay* [17]. Tak efekt zakasnjene vzorce izmenično pošilja na levi ali desni kanal, kar da občutek, da se signal odbija.

Nadgradnja stereo zakasnitev so učinki z več zakasnitvenimi linijami – *multitap delay* [17]. Vsaka zakasnitvena linija ima svoje parametre za zakasnitveni čas, število ponovitev, izhodno glasnost in razporeditev v stereo sliko. Večina resnejših *multitap delayev* uporablja štiri ali več zakasnitvenih linij. Z različno stereo razporeditvijo in izhodne glasnosti vsake zakasnitvene linije lahko naredimo zanimive ritmične efekte.

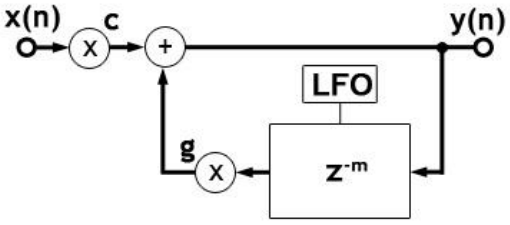
3.1.5 Razširitve učinkov zakasnitve

Veliko zvočnih učinkov, ki so dandanes uporabljeni v glasbi so nadgraditev ali zgrajeni iz zakasnitvenih linij s spreminjajočimi zakasnitvenimi časi. Zaradi teh hitrih sprememb v zakasnitvah pride do popačenja amplitude, ki nastane zaradi amplitudnih modulacij različnih frekvenčnih komponent v signalu [16].

3.1.5.1 Flanger

Učinek *flanger* je sestavljen iz NEO *comb* filtra, kjer spreminjanje zakasnitvenega časa poteka

sinusoidno med najmanjšo in največjo nastavljeno vrednostjo zakasnitve [16]. To vpliva na večanje in manjšanje harmoničnih zaporedij zapornih frekvenc frekvenčnega odziva. Za spreminjanje zakasnitvenega časa v digitalnih *flangerjih* se uporablja nizkofrekvenčni oscilator[5] (angl. *low-frequency oscillator* = LFO), ki generira sinusni ali trikotni signal.

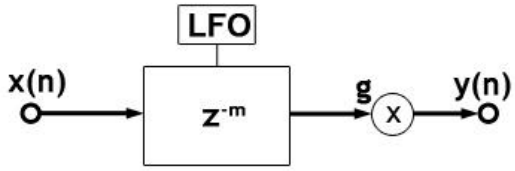


Slika 12: Shema realizacije učinka flanger z uporabo NEO comb filtra in LFO

Zaradi neprestanih spreminjanj dolžine zakasnitvenega časa lahko pride do nezveznosti in popačenj signala, zato je na zakasnitveni liniji potreben interpolacijski filter s končnim enotinim odzivom.

3.1.5.2 Vibrato

Enakomerno periodično spreminjanje višine tona imenujemo vibrato. Glasbeniki, ki igrajo godala uporabljajo učinek vibrata, da dobijo iz inštrumenta lepši ton. V digitalnem svetu, pa lahko tak učinek simuliramo s pomočjo zakasnitvene linije. Periodično spreminjanje zakasnitvenega časa proizvede učinek, ki je slišen v spreminjanju višine tona. Takšno spreminjanje lahko dosežemo z uporabo nizkofrekvenčnega oscilatorja, tako kot pri učinku *flangerja*, vendar tu na izhod peljemo samo zakasnjen signal in ne vsote obeh [16].



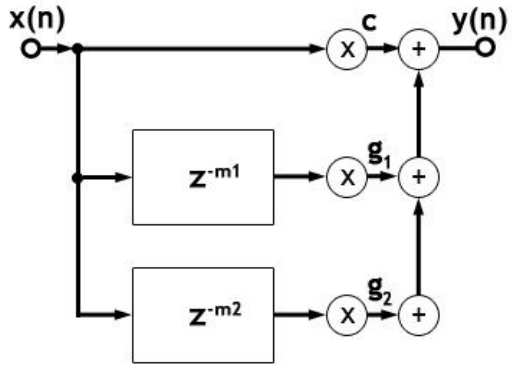
Slika 13: Shema učinka vibrata

3.1.5.3 Chorus

Z uporabo učinka *chorus* dobimo občutek, da igra več inštrumentov hkrati, čeprav je uporabljen samo en inštrument. Dva enaka inštrumenta igrana simultano nikoli ne zvenita enako, kljub enaki uglasitvi. Neskladnost se pojavi zaradi majhnih razlik v višini tonov in rahlih razlikah v sinhronizaciji, kar povzroči majhne zakasnitve. Takšen učinek lahko simuliramo z uporabo ene ali več zakasnitvenih linij, kjer se zakasnitveni čas spreminja naključno. Več zakasnitvenih linij uporabimo, bolj poln bo zvok. Zakasnitveni časi so dolgi

[5] Nizkofrekvenčni oscilator LFO je naprava, ki generira elektronski signal, navadno do frekvence 20 Hz, ki ustvarja ritmično nihanje signala in se uporablja za modulacijo sintetizatorjev, zakasnitvenih linij in drugih naprav, da ustvari učinke vibrata, temola, flangerja in ostalih učinkov, ki se uporabljajo v glasbi.

od 10 do 25 milisekund, zakasnitvene zanke pa se pri učinkih *chorus* navadno ne uporabljajo, ampak ga gradimo s pomočjo KEO *comb* filtrov [16].



Slika 14: Učinek chorus modeliran z dvema zakasnitvenima linijama

3.2 Uporabljena orodja

Pri izdelavi vtičnika s pomočjo *Steinbergovega* SDK razvojnega okolja (dostopno na naslovu [18]) sem bil v veliki dilemi, katero verzijo okolja bi uporabil. Na spletni strani podjetja *Steinberg* sta na voljo različici SDK v2.4 in SDK v3.0. Navadno je pri različnih standardih v navadi, da v uporabo sčasoma pridejo najnovejše verzije, vendar pri standardu VST še danes ni tako. Čeprav je verzija vtičnikov VST 3 izšla leta 2008 je starejša, verzija 2.4, še vedno veliko bolj popularna pri razvijalcih, kot tudi uporabnikih standarda VST. Kot že rečeno v poglavju 2, različica 3 še nima velike podpore v zvočnih urejevalnikih, poleg tega pa ne prinaša veliko ključnih sprememb v primerjavi s starejšo različico. Izkušeni uporabniki in predvsem razvijalci vtičnikov VST na raznih spletnih forumih govorijo, da je nova različica zgolj prodajna poteza podjetja, ki ne prinaša novosti in prednosti, saj je veliko večino oglašanih novosti možno narediti tudi z verzijo 2.4. Glavne prednosti verzije 3 so izboljšano, bolj efektivno delovanje procesnega dela vtičnika, omogočanje aktiviranja ali deaktiviranja zvočnih vodil po zagonu vtičnika in večje število MIDI vhodov/izhodov.

Kljub vsem razpravam sem se pri implementaciji vtičnika VST z učinkom zakasnitve odločil za implementacijo novejše verzije vtičnikov VST 3, s pomočjo razvojnega okolja SDK v3.0.

Izgradnjo celovitega vtičnika VST lahko razdelimo na dva ločena dela. Prvi del obsega implementacijo procesnega dela vtičnika, ki skrbi za procesiranje vhodnega zvočnega toka, drugi del pa se osredotoča na izdelavo grafičnega uporabniškega vmesnika. Za razvoj uporabniškega vmesnika je uporabljen dodatek k *Steinbergovem* SDK, imenovan VST-GUI, ki pa je že vključen v različico VST 3 kot ločena zbirka C++ razredov za upravljanje z grafikami. S pomočjo VST-GUI lahko upravljamo z grafikami, ki jih izdelamo v ločenem programu, jih dodajamo na panel vtičnika VST in jim dodajamo funkcionalnosti.

Glavno delovno okolje, ki sem ga uporabil za implementacijo vtičnika VST je bilo *Microsoftovo* okolje *Visual Studio* 2008, ki vsebuje tudi C++ prevajalnik. Grafike za uporabniški vmesnik sem izdelal s pomočjo 3D programa *Autodesk* 3ds Max. Delovanje

vtičnika pa sem testiral v *Steinbergovem* zvočnem urejevalniku *Cubase* 5, ki je eden izmed redkih urejevalnikov, ki omogočajo poganjanje vtičnikov verzije 3.

3.3 Psevdo algoritem učinka zakasnitve

Algoritem, ki bo omogočal zakasnjevanje vhodnega signala bo temeljil na načelih krožnega pomnilnika, ki smo ga opisali v podpoglavju 3.1.1. Preprosti algoritem bo temeljil na vmesnem pomnilniku, ki bo implementiran kot tabela (*buffer*). Ta bo delovala kot pomnilnik za shranjevanje vhodnih vzorcev digitalnega zvočnega signala. Za dostop do vrednost v tabeli bosta skrbela dva kazalca. Pisalni kazalec (*write*) bo omogočal pisanje v celice in s tem shranjeval vhodni zvočni tok. Bralni kazalec (*read*) pa bo omogočal branje iz celic. Razlika med kazalcema bo določala dolžino zakasnitve, ki jo bo učinek proizvajal. Ena izmed zahtevanih lastnosti, ki jo mora učinek upoštevati je, da učinek simulira zakasnitve z več odboji – t.j. želimo ustvariti NEO *comb* filter, moramo implementirati zakasnitveno zanko, ki bo to omogočala. Za večjo uporabnost vtičnika VST bomo implementirali še kontrolo, ki omogoča mešanje zakasnjenega signala z osnovnim in kontrolo, ki bo nadzorovala celotno izhodno glasnost vtičnika.

Vtičnik bo tako vseboval štiri parametre, ki jih bo lahko uporabnik spreminjal:

- *Delay:* s tem parametrom bo lahko uporabnik nastavil dolžino zakasnitve
- *Feedback:* parameter bo določal število ponovitev zakasnjenega signala
- *DryWet:* parameter za mešanje zakasnjenega signala z osnovnim
- *Out:* parameter za določanje izhodne glasnosti vtičnika

Ker lahko vsi parametri znotraj standarda VST zavzamejo samo realne vrednosti od [0,1], moramo posebej izračunati pozicijo bralnega indeksa po enačbi (8).

$$read = write - Delay * DolžinaTabele \tag{8}$$

Bralni indeks se vsako iteracijo algoritma, kar pomeni za vsak vhodni vzorec, poveča za ena, ko pa je večji ali enak dolžini tabele, pa se resetira na 0. Sedaj lahko s pomočjo teh dveh indeksov in parametra *Feedback* realiziramo pisanje in branje zvočnih vzorcev iz tabele s pomočjo enačbe (9).

$$temp = buffer[write] = input + buffer[read] * FeedBack \tag{9}$$

Enačba (9) določa, da se na mesto v tabeli, ki ga določa kazalec *write*, vpiše vsota vhodnega signala in deležem zakasnjenega signala, določenega s parametrom *FeedBack*. Tako dobimo model NEO *comb* filtra, ki ima spremenljivo dolžino trajanja ponovitve. Na izhod vtičnika v vsaki iteracij algoritma peljemo vrednost, ki nam jo določa enačba (10). Ta nam doda funkcionalnost mešanja vsebovanosti zakasnjenega signala z osnovnim in parameter, ki omogoča nastavljanje glasnosti izhodnega signala.

$$output = (input + temp * DryWet) * Out \quad (10)$$

Tako lahko glede na zgoraj definirane lastnosti napišemo celotni psevdo algoritem za realizacijo učinka zakasnitve:

1. inicializiraj parametre za upravljanje z lastnostmi učinka, ter bralni in pisalni kazalec;
2. inicializiraj podatkovno tabelo;
3. inicializiraj vhodni in izhodni podatkovni tok;
4. preveri ali je pisalni kazalec večji od dolžine tabele, če je ga resetiraj na 0;
5. izračunaj bralni indeks po enačbi (8);
6. preveri ali je bralni indeks manjši od 0, če je mu prištej dolžino tabele;
7. shrani vsoto vhodnega in deleža zakasnjenega signala kot določa enačba (9);
8. na izhod pošlji vrednost, ki ga določa enačba (10);
9. ponavljaj korake od 4 do 8 dokler gostitelj pošilja vzorce zvočnega signala.

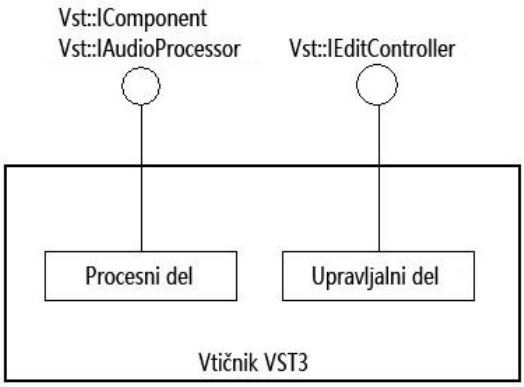
3.4 Arhitektura vtičnikov VST 3

VST SDK v3.0 vsebuje programski vmesnik (VST 3 API), pomožne razrede s pomočjo katerih lahko razvijalec implementira vtičnike VST 3, dokumentacijo standarda VST, primere uporabe in aplikacije namenjene testiranju vtičnikov. Standard VST 3 temelji na tehnologiji imenovani VST *Module Architecture*. To je komponenti modelni sistem, ki je uporabljen v *Steinbergovih* zvočnih urejevalnikih in predstavlja osnovno plast za podporo vtičnikom znotraj zvočnih urejevalnikov, kot tudi podporo notranjim komponentam njihovih urejevalnikov.

VST 3 API je zbirka vmesnikov znotraj različnih imenskih prostorov. Ti so bili narejeni za uporabo v komponentah, ki omogočajo procesiranje zvočnih podatkovnih tokov v realnem času. Takšna komponenta je lahko zvočni učinek ali inštrument. VST 3 API predstavlja vmesnik, ki omogoča komunikacijo vtičnika VST z gostiteljem. Sama implementacija teh vmesnikov pa poteka s pomočjo dodatnih, pomožnih razredov.

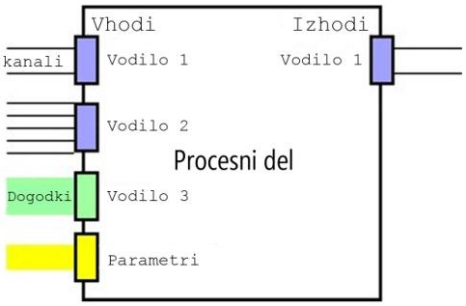
Vtičnik VST 3 je sestavljen iz dveh delov [19]:

- Procesni del (angl. *processor*) opravlja nalogo digitalnega procesiranja signalov.
- Upravljalni del (angl. *edit controller*) je odgovoren za grafični uporabniški vmesnik vtičnika.



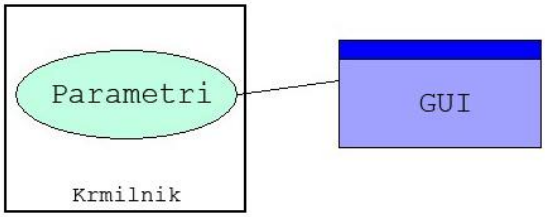
Slika 15: Zgradba vtičnika VST verzije 3

Procesni del sestavljata dva povezana vmesnika – *IAudioProcessor* in *IComponent* [19], ki se nahajata znotraj imenskega prostora *Steinberg::Vst*. Vmesnik *IAudioProcessor* poskrbi, da je procesni del vtičnika pred začetkom procesiranja ustrezno konfiguriran. Obvesti ga o parametrih, kateri se med procesiranjem ne spreminjajo, ter skrbi za dinamično spremembo števila kanalov na zvočnem vodilu (angl. *bus*). Po konfiguraciji procesnega dela je potrebna njegova aktivacija, kar omogoči izvedbo glavne procesne funkcije, ki opravlja dejansko procesiranje nad vhodnim signalom. Drugi del procesnega dela, *IComponent*, omogoča kreiranje zvočnih vodil, ki predstavljajo podatkovne vhode in izhode vtičnika, skrbi za povezovanje procesnega in upravljalnega dela vtičnika, ter omogoča aktiviranje in deaktiviranje komponente VST.



Slika 16: Sestavni deli procesnega dela vtičnika

Upravljalni del definira vmesnik *IEditController* z definicijo krmilnika, ki poskrbi za osnovni GUI vtičnika. Gostitelj mora zagotoviti povratni vmesnik, ki je nujen za komunikacijo med gostiteljem in procesnim delom. Krmilnik lahko na zahtevo prikaže uporabniški vmesnik vtičnika. Odgovoren je tudi za upravljanje s parametri. Vsaka sprememba parametrov, ki se zgodi z interakcijo uporabnika z uporabniškim vmesnikom mora biti sporočena procesnemu delu vtičnika. Za sam prenos informacije, pa je zadolžen gostitelj. Tu je treba poudariti, da ne gre za uporabniški vmesnik po meri, ampak za osnovni vmesnik, ki asimilira okenske lastnosti zvočnega urejevalnika, ki poganja vtičnik. Grafični vmesnik po meri se lahko implementira s pomočjo dodatnega orodja imenovanega VST-GUI.



Slika 17: Krmilnik poskrbi za ločen prikaz GUI-ja

Zaradi dveh ločenih delov vtičnika (procesni del in upravljalni del) obstaja potreba po komunikaciji med tema komponentama. To nalogo opravlja gostitelj – digitalni zvočni urejevalnik. Vsaka interakcija z uporabniškim vmesnikom je sporočena vmesniku *IComponentHandler*, ki omogoči gostitelju, da sporoči spremembe, ter tako omogoči sinhronizacijo upravljalnega dela s procesnim. Takšna standardna komunikacija poteka po naslednjih korakih [19]:

- Takoj po kreiranju vtičnika gostitelj predvideva, da je stanje procesnega dela sinhronizirano s stanjem parametrov krmilnika.
- V primeru, da gostitelj določi novo stanje procesorja z uporabo *IComponent::setState*, mora to biti poslano tudi krmilniku, da ta nastavi svoje parametre z uporabo funkcije *IEditController::setComponentState.*
- Ko krmilnik gostitelju posreduje spremembe svojih parametrov, gostitelj sinhronizira procesni del tako, da pošlje nove vrednosti procesnemu klicu.
- Prav tako procesni del pošilja spremembe svojih parametrov gostitelju. Te spremembe so potem posredujejo še krmilniku z uporabo *IEditController::setParamNormalized.*

Tako je definirana osnovna zgradba vtičnikov VST 3. Seveda znotraj okolja VST 3 API obstaja še veliko drugih imenskih prostorov in znotraj njih vmesniki, ki omogočajo upravljanje z vhodnimi podatkovnimi tokovi in različnimi dogodki, katere dogajanje lahko spremljamo. Za implementacijo enostavnega učinka zakasnitve, pa je dovolj poznavanje te osnove arhitekture.

3.5 Implementacija procesnega jedra vtičnika

Sedaj, ko smo pregledali teorijo, ki je v ozadju učinka zakasnitve, arhitekturo vtičnikov VST in izbrali primerna orodja za realizacijo vtičnika je potrebno narediti program, ki bo upošteval ugotovitve iz začetka poglavja in realiziral vtičnik VST z učinkom zakasnitve s pomočjo razredov znotraj *Steinbergovega* razvojnega okolja SDK v3.0. Od tu naprej je priporočeno, da ima bralec vsaj nekaj znanja o osnovah programiranja, programskem jeziku C++ in načelih metod dedovanja, saj sledi prikaz konkretne implementacije vtičnikov VST s primeri izvorne kode.

Procesno jedro lahko implementiramo z razredom *Delay*, ki deduje lastnosti pomožnega razreda *AudioEffect*. Ta razred nastopa kot privzeta implementacija zvočnega učinka VST 3 in se lahko uporabi kot osnovni razred pri izgradnji lastnih vtičnikov. Velja omeniti, da izvornih

pomožnih razredov ne smemo spreminjati, saj bi s tem porušili standard VST in tako ogrozili pravilno delovanje vtičnikov.

3.5.1 Zaglavna datoteka procesnega dela

Vsak razred, ki ga implementiramo v sklopu realizacije vtičnikov VST sestavljata 2 datoteki – zaglavna datoteka (angl. *header file*) in implementacijska datoteka. Zaglavna datoteka je zadolžena za definiranje vtičnikovega objekta, ki omogoča drugim programom interakcijo z vtičnikom brez znanja o tem, kako sam vtičnik izvrši svojo nalogo. Tako zaglavna datoteka vsebuje deklaracijo funkcij in spremenljivke s pripadajočimi podatkovnimi tipi.

```
1.  // File Delay.h
2.
3.  #ifndef __delay__
4.  #define __delay__
5.
6.  #include "public.sdk/source/vst/vstaudioeffect.h"
7.
8.  using namespace Steinberg::Vst;
9.
10. #define gComponentUID1 0x84e8de5f
11. #define gComponentUID2 0x92554f53
12. #define gComponentUID3 0x96fae413
13. #define gComponentUID4 0x3c935a18
14. #define gComponentUID  gComponentUID1, gComponentUID2, gComponentUID3,
    gComponentUID4
15.
16. #define gControllerUID1 0xD39D5B65
17. #define gControllerUID2 0xD7AF42FA
18. #define gControllerUID3 0x843F4AC8
19. #define gControllerUID4 0x41EB04F0
20. #define gControllerUID gControllerUID1, gControllerUID2,
    gControllerUID3, gControllerUID4
21.
22. class Delay: public AudioEffect
23. {
24. public:
25.    Delay ();
26.
27.    static FUnknown* createInstance (void* context)
28.    {
29.       return (IAudioProcessor*)new Delay;
30.    }
31.
32.    tresult PLUGIN_API initialize (FUnknown* context);
33.    tresult PLUGIN_API terminate  ();
34.    tresult PLUGIN_API setActive (TBool state);
35.    tresult PLUGIN_API process (ProcessData& data);
36.    tresult PLUGIN_API setState (IBStream* state);
37.    tresult PLUGIN_API getState (IBStream* state);
38.    tresult PLUGIN_API setupProcessing (ProcessSetup& newSetup);
39.
40. private:
41.
42.    float fDelay;
43.    float fFeedBack;
44.    float fOut;
45.    float fDryWet;
46.
47.    float* bufferL;
48.    float* bufferR;
49.    long bufferLength;
50.    float read;
```

```
51.     int write;
52.
53.     int32 currentProcessMode;
54.
55.     bool  bBypass;
56.  };
```

Algoritem 1: Zaglavna datoteka procesnega dela vtičnika z učinkom zakasnitve

V algoritmu 1 je celotna izvorna koda zaglavne datoteke, ki v vrstici 22 definira naš razred *Delay*. Razred mora biti definiran v predprocesorju, ki je pristojen za zbiranje vseh objektov, funkcij, spremenljivk in imen razredov na začetku prevajanja izvorne kode v strojno. V vrstici 3 preverimo, ali je naš razred v njem že definiran, če ni ga definiramo. Sledi vključevanje pomožnih razredov in definicija imenskega prostora znotraj katerega funkcije, objekte in razrede bomo uporabljali. V vrsticah od 10 do 20 definiramo enolične identifikacijske konstante, ki jih uporabimo za označitev komponente in krmilnika vtičnika. Vsak vtičnik ima svojo enolično identifikacijsko številko, kar omogoča gostitelju, da loči različne vtičnike med seboj. Preostali del kode deklarira naš razred, ter funkcije in spremenljivke, ki bodo v njem uporabljene.

Beseda *public* v 24 vrstici pove prevajalniku, da so sledeče funkcije in podatki dostopni drugim objektom in razredom. Tako lahko naš razred *Delay* enostavno uporabimo tudi v drugih implementacijah vtičnikov.

Za inicializacijo spremenljivk in funkcij razreda potrebujemo konstruktor, ki je poklican, kadar vtičnik poženemo znotraj gostitelja.

V vrstici 27 definiramo funkcijo, ki omogoča kreiranje novih inačic istega vtičnika VST. V primeru, da v zvočnem urejevalniku potrebujemo več instanc enakega vtičnika se bo uporabila definirana funkcija in vsaki zahtevani zvočni stezi generirala svojo instanco istega vtičnika. Ti bodo med seboj logično ločeni in omogočali ločeno upravljanje s svojimi parametri. Definicije v naslednjih vrsticah opisujejo javne funkcije, ki direktno določajo upravljanje in funkcionalnost vtičnika. Podrobneje si jih bomo ogledali ob razlagi implementacijske datoteke.

Vrstica 40, z rezervirano besedo *private*, označuje začetek spremenljivk, do katerih lahko dostopajo samo lastniški razred in prijatelji razreda. Tu definiramo spremenljivke, katere bodo služile kot parametri, ki jih bo končni uporabnik lahko spreminjal in ostale, katere bodo služile kot pomožni deli pri modeliranju učinka zakasnitve. Tako so spremenljivke *fDelay, fFeedBack, fDryWet* in *fOut* tisti parametri, ki jih bo končni uporabnik uporabljal za spremembo dolžine zakasnitve, števila ponovitve signala, količine vsebovanosti zakasnjenega signala v primerjavi z osnovnim in parameter za določanje izhodne glasnosti, kot smo jih definirali v poglavju 3.3. Tu velja še enkrat ponoviti, da lahko vsi parametri pri standardu VST zavzemajo le realna števila v množici [0,1], zato so vsi ti parametri podatkovnega tipa *float.* V vrstici 47 sledi definicija dveh podatkovnih tabel, ki bosta služili za shranjevanje vhodnega signala. S spremenljivko *bufferLength* lahko enostavno določamo dolžino pomnilnih tabel, kar nam omogoča hitro nastavljanje maksimalne dolžine zakasnitve. Kazalca s katerimi bomo dostopali do vrednosti v tabeli pa sta definirana kot *write* in *read.*

3.5.2 Implementacijska datoteka procesnega dela

V implementacijski datoteki, delay.cpp, implementiramo konstruktor našega razreda in definirane funkcije v zaglavni datoteki, med kateri je tudi glavni procesni klic, ki opravlja dejansko procesiranje nad vhodnim podatkovnim tokom.

```
14. Delay::Delay ()
15. : fDelay (0.5f),
16.  fFeedBack (0.4f),
17.  fOut (1.f),
18.  fDryWet (1.f),
19.  currentProcessMode (-1),
20.  bufferLength (44100),
21.  read (0.0f),
22.  write(0),
23.  bBypass (false),
24. {
25.       setControllerClass (FUID (gControllerUID));
26. }
```

Algoritem 2: Kostruktor razreda Delay z inicializacijo parametrov

Konstruktor inicializira vrednosti parametrov, ki jih vsebuje naš razred. Te vrednosti bodo uporabljene kot začetna nastavitev našega vtičnika. Spremenljivka *currentProcessMode* določa v kakšnem stanju procesiranje je vtičnik. Na začetku je spremenljivka postavljena na vrednost -1, kar označuje da način procesiranja še ni inicializiran. Pred začetkom procesiranja se pokliče funkcija, ki omogoča spremembo načina procesiranja. Izbiramo lahko med načini procesiranja v realnem času, procesiranja z zakasnitvijo in načinu *prefetch.*

Naslednji dve funkciji naredita inicializacijo procesnega dela in terminacijo, ki omogoča sproščanje zasedenega pomnilnega prostora, ki ga je vtičnik zasedel, ob njegovem zaprtju.

```
28. tresult PLUGIN_API Delay::initialize (FUnknown* context)
29. {
30.       tresult result = AudioEffect::initialize (context);
31.
32.       if (result != kResultOk)
33.       {
34.             return result;
35.       }
36.
37.       bufferL = new float[bufferLength];
38.       bufferR = new float[bufferLength];
39.
40.       for (int32 i = 0; i < bufferLength; i++) {
41.             bufferL[i]=0;
42.             bufferR[i]=0;
43.       }
44.
45.       addAudioInput  (USTRING ("Stereo In"),  SpeakerArr::kStereo);
46.       addAudioOutput (USTRING ("Stereo Out"), SpeakerArr::kStereo);
47.
48.       return kResultOk;
49. }
50.
51. tresult PLUGIN_API Delay::terminate  ()
52. {
53.       if(bufferL)
54.             delete[] bufferL;
55.       If(bufferR)
```

```
56.              delete[] bufferR;
57.
58.        return AudioEffect::terminate ();
59.  }
```

Algoritem 3: Funkciji za inicializacijo in terminiranje komponente

Inicializacijo komponente izvrši gostitelj preko vmesnika *IPluginBase*, ki vsebuje ti dve funkciji. Večje alokacije pomnilnega prostora je priporočljivo inicializirati ravno znotraj funkcije *initialize* in ne znotraj razrednega konstruktorja. Tako tukaj inicializiramo našo podatkovno tabelo za shranjevanje vzorcev signala. Dolžino tabele določa spremenljivka *bufferLength*, ki je incializirana na vrednost 44100. Takšna dolžina nam bo pri vhodnem signalu, vzorčenem s frekvenco 44100Hz, omogočala zakasnitve za natanko 1 sekundo – enačba 11.

$$maksimalni\ čas\ zakasnitve\ = \frac{dožina\ pomnilne\ tabele}{frekvenca\ vzorčenja} \tag{11}$$

V vrsticah 45 in 46 definiramo vhodno in izhodno podatkovno vodilo in število kanalov vsakega. Seveda želimo, da bo učinek zmožen procesiranja vsaj stereo vhodnega signala in rezultat peljal prav tako peljal na stereo izhod.

Po uspešni konfiguraciji komponente in njenih sestavnih delov je potrebna aktivacija procesnega dela vtičnika, da bo lahko začel s procesiranjem signalov. Ta aktivacijski klic označuje, da so se vse konfiguracije zaključile. Preden gostitelj začne z izvajanjem procesnih klicev, mora to zahtevo sporočiti s klicanjem funkcije *setProcessing(true)*. Po končanem procesiranju, pa mora zaključek sporočiti s klicem *setProcessing(false)*.

Sledi glavni, procesni klic, ki implementira dejanski učinek zakasnitve – algoritem 4.

```
66.  tresult PLUGIN_API Delay::process (ProcessData& data)
67.  {
68.     IParameterChanges* paramChanges = data.inputParameterChanges;
69.     if (paramChanges)
70.     {
71.        int32 numParamsChanged = paramChanges->getParameterCount ();
72.        for (int32 i = 0; i < numParamsChanged; i++)
73.        {
74.          IParamValueQueue* paramQueue = paramChanges->getParameterData(i)
75.           if (paramQueue)
76.           {
77.              int32 offsetSamples;
78.              double value;
79.              int32 numPoints = paramQueue->getPointCount ();

80.              switch (paramQueue->getParameterId ())
81.              {
82.                 case kDelayId:
83.                    if (paramQueue->getPoint (numPoints - 1,  offsetSamples,
     value) == kResultTrue)
84.                    {
85.                       fDelay = (float)value;
86.                    }
87.                 break;
88.
...                 //checks the change for each of the rest parameters
116.
```

```
            }
         }
      }
   }
   //if no inputs
   if (data.numInputs == 0 || data.numOutputs == 0)
   {
      return kResultOk;
   }

   int32 numChannels = data.inputs[0].numChannels;

   float** in  = data.inputs[0].channelBuffers32;
   float** out = data.outputs[0].channelBuffers32;

   //---check if silence---------------
   if (data.inputs[0].silenceFlags != 0)
   {
      data.outputs[0].silenceFlags = data.inputs[0].silenceFlags;

      int32 sampleFrames = data.numSamples;
      for (int32 i = 0; i < numChannels; i++)
      {
         if (in[i] != out[i])
         {
            memset (out[i], 0, sampleFrames * sizeof (float));
         }
      }
      return kResultOk;
   }

   //---in bypass mode outputs = inputs-----
   if (bBypass)
   {
      int32 sampleFrames = data.numSamples;
      for (int32 i = 0; i < numChannels; i++)
      {
         if (in[i] != out[i])
         {
            memcpy (out[i], in[i], sampleFrames * sizeof (float));
         }
      }
   }
   else
   {
      //if mono track
      if(numChannels == 1) {
         int32 sampleFrames = data.numSamples;

         float* ptrInMono = in[0];
         float* ptrOutMono = out[0];

         while (--sampleFrames >= 0)
         {
            float in = *ptrInMono++;
            if(write > bufferLength)
               write = 0;

            read = write - (fDelay * bufferLength);

            if( read < 0 )
               read += bufferLength;

            float out;
            out = bufferL[write++] = in+(bufferL[(int)read]*fFeedBack);

             (*ptrOutMono++) = (in + out * fDryWet)*fOut;
         }
```

```
185.        }
186.
187.        //if stereo track
188.        else {
189.           int32 sampleFrames = data.numSamples;
190.
191.           float* ptrInL = in[0];
192.           float* ptrInR = in[1];
193.
194.           float* ptrOutL = out[0];
195.           float* ptrOutR = out[1];
196.
197.
198.           while (--sampleFrames >= 0)
199.           {
200.              float inL = *ptrInL++;
201.              float inR = *ptrInR++;
202.
203.              if(write > bufferLength)
204.                 write = 0;
205.
206.              read = write - (fDelay * bufferLength);
207.
208.              if( read < 0 )
209.                 read += bufferLength;
210.
211.              float outL;
212.              float outR;
213.              outL=bufferL[write] = inL+(bufferL[(int)read]*fFeedBack);
214.              outR=bufferR[write++]= inR+(bufferR[(int)read]*fFeedBack);
215.
216.              (*ptrOutL++) = (inL + outL * fDryWet)*fOut;
217.              (*ptrOutR++) = (inR + outR * fDryWet)*fOut;
218.           }
219.        }
220.     }
221.     return kResultOk;
222. }
```

Algoritem 4: Glavni procesni klic, ki vsebuje algoritem za dodajanje učinka zakasnitve

Podatki, nad katerimi je potrebno narediti operacijo, so metodi *process* poslani kot argument. Na začetku funkcije se preveri ali je prišlo do sprememb parametrov. Spremembe parametrov se zgodijo zaradi interakcije uporabnika z grafičnim uporabniškim vmesnikom. Vsaka sprememba parametra se zapiše v vrsto. Algoritem 4 v vrsticah 68 do 120 preveri pri katerih parametrih se je zgodila sprememba in novo vrednost zapiše v ustrezno spremenljivko.

Za vsako zvočno vodilo, ki smo ga definirali v vtičniku, mora gostitelj priskrbeti ustrezne podatke. Tako v vrstici 129 in 130 definiramo kazalca za vhodne in izhodne zvočne medpomnilnike, preko katerih bomo dobili podatke za procesiranje.

Z vrstico 133 preverimo ali so postavljene t.i. tihe zastavice (angl. *silent flags*), ki se uporabljajo pri optimizaciji in omogočajo vtičniku zmanjšanje porabljene procesne moči v primeru praznih vhodnih vodil. S pogojnim stavkom v vrstici 149 preverimo, ali je morebiti vključen *bypass* in v primeru izpolnitve pogoja kopiramo vhodne podatke direktno na izhod, brez dodajanja učinka zakasnitve. Če pogoj ni izpolnjen, pa se izvede procesiranje učinka zakasnitve.

Ločimo dve verziji enakega algoritma za dodajanje učinka zakasnitve. Razlikujemo algoritem za uporabo na mono vhodnem kanalu (vrstice od 163 do 185) in algoritem za uporabo na stereo kanalu (vrstice od 188 do 220). Primera moramo ločiti zaradi različnega števila uporabljenih pomnilnih tabel za shranjevanje signala. V primeru, da je vhodni signal mono uporabljamo samo eno pomnilno tabelo, saj imamo le en zvočni kanal. V stereo načinu, pa moramo signal vsakega kanala posebej shranjevati v svojo tabelo. Bralni in pisalni indeks je seveda za obe tabeli enak. V obeh primerih pa je algoritem enak tistemu, ki smo ga definirali v poglavju 3.3.

Na koncu implementacijske datoteke sledi še reimplementacija funkcij *setState* in *getState*, ki služita za shranjevanje in pridobivanje informacije o stanju parametrov komponente.

3.6 Implementacija upravljalnega dela vtičnika

Po uspešni implementaciji procesnega dela je težji del že za nami. Sedaj je potrebno narediti še uporabniški vmesnik preko katerega bo uporabnik dostopal in spreminjal parametre, ki vplivajo na lastnosti učinka zakasnitve. Za implementacijo uporabniškega vmesnika so nam zopet v pomoč vmesniki znotraj VST 3 API. Definicijo krmilnika zopet sestavljata zaglavna in implementacijska datoteka. V tem poglavju si bomo pogledali implementacijo krmilnika, ki skrbi za uporabniški vmesnik in vsebino njegove implementacijske datoteke.

Standardni uporabniški vmesnik implementiramo preko vmesnika *IEditController*. Če ne implementiramo lastnega uporabniškega vmesnika s pomočjo razredov iz okolja VST-GUI, gostitelj avtomatično kreira standardni uporabniški vmesnik, ki bazira na svojih okenskih lastnostih. Za to operacijo razvijalcu ni potrebno definirati uporabniškega vmesnika, temveč mora v krmilniku definirati samo vse tiste parametre, ki sodijo v uporabniški vmesnik. Gostitelj pa te prikaže v vmesniku avtomatično.

Med privatnimi spremenljivkami, ki smo jih definirali znotraj razreda *Delay*, moramo tako določiti tiste, ki so dejansko odgovorne za spremembo lastnosti učinka, da jih gostitelj lahko prikaže v uporabniškem vmesniku. Spomnimo se, to so parametri definirani v poglavju 3.3. Za vsak tak parameter moramo tako definirati razred, ki podeduje lastnosti pomožnega razreda *Parameter*. V konstruktorju vsakega parametra definiramo lastnosti, ki mu jih želimo dodati (ime, enota), s prepisom metod *toString()* in *fromString()*, pa omogočimo izpis parametra v uporabniškem vmesniku s pravilnim formatom.

```
 88.  class OutParameter : public Parameter
 89.  {
 90.  public:
 91.     OutParameter (int32 flags, int32 id);
 92.
 93.     virtual void toString (ParamValue normValue, String128 string) const;
 94.     virtual bool fromString(const TChar* string,ParamValue& normValue) const;
 95.  };
 96.
 97.  OutParameter::OutParameter (int32 flags, int32 id)
 98.  {
 99.     Steinberg::UString (info.title, USTRINGSIZE (info.title)).assign (USTRING
      ("Level"));
100.     Steinberg::UString (info.units, USTRINGSIZE (info.units)).assign (USTRING
```

```
    ("%"));
101.
102.    info.flags = flags;
103.    info.id = id;
104.    info.stepCount = 0;
105.    info.defaultNormalizedValue = 0.5f;
106.    info.unitId = kRootUnitId;
107. }
108.
109. void OutParameter::toString (ParamValue normValue, String128 string) const
110. {
111.    char text[32];
112.    sprintf (text, "%d", (int)(100.f * (float)normValue));
113.    Steinberg::UString (string, 128).fromAscii (text);
114. }
115.
116. bool OutParameter::fromString (const TChar* string, ParamValue& normValue)
    const
117. {
118.        Steinberg::UString wrapper ((TChar*)string, -1);
119.        double tmp = 0.0;
120.        if (wrapper.scanFloat (tmp))
121.        {
122.              normValue = (float)tmp / 100.f;
123.              return true;
124.        }
125.        return false;
126. }
```

Algoritem 5: Primer definicije avtomatiziranega parametra za nastavljanje izhodne glasnosti vtičnika

V algoritmu 5 je prikaz definicije samo enega parametra, parametra, ki nam določa izhodno glasnost vtičnika. Definicija ostalih treh parametrov, ki jih bomo uporabljali pri našem vtičniku je ekvivalentna zgornjemu primeru.

Sedaj ko imamo vse parametre definirane je znotraj krmilnikove funkcije *initialize*, potrebna kreacija novih instanc vsakega razreda parametrov. Vsakem primerku razreda, torej parametru, dodamo lastnost avtomatizacije in ga dodamo v vsebovalnik parametrov z ukazom *addParameter*.

```
187.    OutParameter* outParam = new OutParameter (ParameterInfo::kCanAutomate,
    kOutId);
188.    parameters.addParameter (outParam);
```

Algoritem 6: Kreiranje in dodajanje parametra v vsebovalnik

Po uspešni implementaciji vsakega parametra je vtičnik že funkcionalen in ga je možno pognati znotraj gostitelja. Prikazan bo s preprostim uporabniškim vmesnikom, katerega parametre je možno spreminjati preko svojih drsnikov. Standardna komunikacija med procesnim in upravljalnim delom poteka tako, kot smo opisali v zadnjem delu podpoglavja 3.4. Da bo vtičnik ob zagonu pravilno prikazoval trenutno stanje parametrov v procesnem delu, je potrebno prepisati še funkcijo *setComponentState*, ki omogoča sinhronizacijo obeh delov.

```
206. tresult PLUGIN_API DelayController::setComponentState (IBStream* state)
207. {
208.    if (state)
209.    {
```

```
210.        float savedDelay = 0.f;
211.        if (state->read (&savedDelay, 4) != kResultOk)
212.        {
213.           return kResultFalse;
214.        }
215.
216.        setParamNormalized (kDelayId, savedDelay);
217.
218.        float savedFb = 0.f;
219.        if (state->read (&savedFb, 4) != kResultOk)
220.        {
221.           return kResultFalse;
222.        }
223.
224.        setParamNormalized (kFeedBackId, savedFb);
225.
226.        float savedOut = 0.f;
227.        if (state->read (&savedOut, 4) != kResultOk)
228.        {
229.           return kResultFalse;
230.        }
231.
232.        setParamNormalized (kOutId, savedOut);
233.
234.        float savedDryWet = 0.f;
235.        if (state->read (&savedDryWet, 4) != kResultOk)
236.        {
237.           return kResultFalse;
238.        }
239.
240.        setParamNormalized (kDryWetId, savedDryWet);
241.     }
242.     return kResultOk;
243. }
```

Algoritem 7: Funkcija, ki omogoča krmilniku sinhronizacijo s procesnim delom vtičnika

S tem je vtičnik s standardnim uporabniškim vmesnikom popolnoma delujoč. Slika 18 prikazuje tak dokončan vtičnik. Ker uporabniški vmesnik zagotovi gostitelj ima vmesnik okenske lastnosti gostitelja. Primer na sliki 18 je bil pognan v zvočnem urejevalniku Cubase 5, v drugih urejevalnikih bi standardni pogled izgledal drugače.

Slika 18: Vtičnik z učinkom zakasnitve, prikazan s standardnim uporabniškim vmesnikom v programu Cubase 5.

3.7 Implementacija uporabniškega vmesnika po meri

Čeprav je vtičnik VST že polno funkcionalen, bomo izdelali še ločen uporabniški vmesnik, ki bo naredil vtičnik bolj privlačen in omogočil lažje upravljanje. Ideja je, da se naredi vmesnik, ki bo omogočal nastavljanje parametrov preko gumbov. Tako moramo v ločenem programu

izdelati grafike za ozadje vtičnika in grafiko za gumb. Poleg gumba, bo lahko uporabnik vrednosti parametrov nastavljal preko enostavnega polja za vnos.

Kreiranje uporabniškega vmesnika z VST-GUI je enostavno. Najprej je potrebno kreirati razred, ki podeduje lastnosti razreda *VSTGUIEditor* in prepisati osnovni metodi *open* in *close*. Znotraj metode *open* kreiramo okvir, enake velikosti kot je grafična datoteka z ozadjem. S kreiranjem nove instance našega grafičnega razreda znotraj krmilnika upravljalnega dela vtičnika, pa imamo tako definiran osnovo za grafični vmesnik po meri. Sedaj mu je potrebno dodati še komponente, kontrole in in dogodkovne poslušalce. To storimo z uporabo razredov *CBitmap* (za nalaganje in uporabo grafik), *CKnob* (za ustvarjanje gumbov), CTextEdit (za ustvarjanje vnosnih polj) in *CControlListener* (za ustvarjanje dogodkovnih poslušalcev). Seveda so ti poslušalci namenjeni spremljanju dogajanja pri spremembi parametrov in sinhronizacije uporabniškega vmesnika s krmilnikom. Vsaka sprememba, ki jo uporabnik naredi z interakcijo z vmesnikom, bodisi z gumbom, bodisi z vnosom v vnosno polje, mora biti upoštevana in povzročiti spremembo parametrov krmilnika. Ta seveda komunicira naprej s procesnim delom po že prej opisanih pravilih.

Slika 19: Končni uporabniški vmesnik vtičnika z učinkom zakasnitve

S tem je zaključena implementacija vtičnika z učinkom zakasnitve. Realizirali smo vse potrebne sestavne dele, ki jih mora tak vtičnik vsebovati.

3.8 Nadaljnje delo

Z zgornjo implementacijo smo dobili delujoči vtičnik VST 3, ki omogoča ustvarjanje učinka zakasnitve. Ima štiri parametre za upravljanje in omogoča zakasnitve do 1 sekunde, vendar tudi eno pomanjkljivost. Problem se pojavi pri hitrem premikanju parametra za nastavljanje dolžine zakasnitve (parameter *Delay*). Pri spremembi parametra časa se spremeni tudi bralni indeks in s tem je vsebina zakasnjenega signala drugačna – bralna glava se zelo hitro pomakne na drugo celico v podatkovni tabeli, ki vsebuje drug zvočni vzorec. Hitra sprememba parametra iz vrednosti *x* v *y* pomeni, da bo vtičnik takoj želel predvajati vse vzorce, ki so v podatkovni tabeli na vseh pozicijah med *x* in *y*. Takšno hitro premikanje bralnega indeksa se odraža kot popačenje ali pokanje izhodnega signala, ki traja vsak krog ponovitve zakasnitve dokler ne izzveni, kar je določeno s parametrom *FeedBack*.

Kot napotke za nadaljnje delo bom podal nekaj možnih rešitev problema, ki so lahko realizirani pri nadgraditvi projekta.

- Najbolj osnova rešitev problema bi bila, da se v času premikanja problemskega parametra učinek zakasnitve enostavno ne predvaja. Tako bi v tem času na izhodu dobili samo osnovni vhodni signal brez zakasnitev. Ko pa se parameter ustali, pa se učinek zakasnitve znova prišteje vhodnemu signalu.
- Druga rešitev bi bila, da se uporabita dva bralna indeksa, recimo jim A in B. Na začetku beremo vrednosti iz pozicij obeh kazalcev, ki imata enako vrednost, pri čemer določimo vsebnost signala A 100% in signala B 0%. Ko se parameter spremeni, to novo vrednost shranimo v indeks B in počasi spreminjamo vsebnost iz A v B. Ko doseže vsebnost signala B 100% shranimo njegovo pozicijo na A in se pripravimo za naslednjo menjavo, ki pa bo potekala ravno obratno. Tako dosežemo neprekinjeno predvajanje učinka zakasnitve.

Vredno je tudi razmisliti ali je takšna izpopolnitev vredna trudu vloženega dela. To trditev bi argumentiral s tem, da se v veliki večini primerov čas zakasnitve med samo uporabo učinka ne spreminja. Ko uporabnik nastavi željen učinek, ga med samo izvedbo skladbe ne spreminja, ampak tekom igranja uporablja enake nastavitve. Tako uporabnika med samim igranjem pomanjkljivost našega učinka zakasnitve ne moti. Funkcionalnost vtičnika lahko uporablja brez težav.

Drug primer, ki pa govori v prid izboljšavi je avtomatizacija parametrov. Tu lahko gostitelj vtičnika VST, tekom predvajanja skladbe, dinamično spreminja parameter zakasnitve po pravilih, ki jih je določil uporabnik. V tem primeru se zopet pokaže slabost implementiranega vtičnika. Ne glede na vse, pa bi bilo priporočljivo pomanjkljivost vtičnika VST z nadaljnjo implementacijo popraviti in tako dobiti brezhiben izdelek, ki je primeren za vsakršnjo uporabo.

4. Realizacija vtičnikov s pomočjo programov za vizualno programiranje

Po prikazu implementacije vtičnikov VST 3 s pomočjo razvojnega okolja VST SDK v3.0, bi rad predstavil še nekaj alternativ izgradnje vtičnikov. Tu gre za dva zanimiva programa, ki v primerjavi s prejšnjo metodo izgradnje v poglavju 3, omogočata relativno enostavno izgradnjo vtičnikov. Programa omogočata programiranje vtičnikov s pomočjo metode vizualnega programiranja. Osnovne gradnike tako povezujemo v celoto in tako gradimo aplikacijo željenih lastnosti. Za razliko od prejšnjega poglavja, opisana programa omogočata izdelovanje samo vtičnikov starejše verzije 2.4.

4.1 SynthEdit

Program SynthEdit je modularni sintetizator zvoka, ki ga je razvil Jeff McClintock [20]. Obsega veliko področje pri ustvarjanju glasbe. Na osnovnem nivoju lahko z njim generiramo enostaven zvočni učinek, v svojem največjem obsegu pa lahko poganja celotno zvočno sliko od virtualnih sintetizatorjev, naprav za generiranje ritma in procesorjev za zvočne učinke.

Sledi načelom analognih sintetizatorjev zvoka. Ti so proizvajali zvok tako, da je bilo potrebno v celoto povezovati različne module, kot so oscilatorji, filtri, envelope generatorji in ojačevalniki, ki so vsak po svoje nekako modificirali zvok. Čeprav je bil ta proces dolgotrajen in težak, pa je omogočal svobodo pri kreiranju zvoka po željah ustvarjalcev.

Tako SynthEdit omogoča fleksibilnost analognih sintetizatorjev zvoka, pri čemer dodaja moderne funkcionalnosti, kot so podpora MIDI in podpora vtičnikom VST verzije 2.4. To fleksibilnost dosega z uporabo modularnega VPL (angl. v*isual programming language*) programskega jezika, ki omogoča programiranje z vnaprej definiranimi komponentami in ne s tekstovnim programiranjem vsakega objekta. Tako omogoča programiranje z vizualnimi izrazi, prostorskimi ureditvami in grafičnimi simboli. Tako vrsto programiranja poimenujemo tudi diagramsko programiranje.

Programska oprema SynthEdit omogoča [22]:

- modeliranje zvočnih sintetizatorjev;
- modeliranje vtičnikov VST (učinki VST in inštrumenti VST);
- implementacijo grafičnega vmesnika za vtičnike VST;
- popolno podporo tehnologiji MIDI;
- procesiranje zvočnega toka v realnem času;
- generiranje programske kode posameznih modulov v C++ programskem jeziku;
- uvoz in izdelavo dodatnih modulov (dodatni moduli so na voljo na [22]).

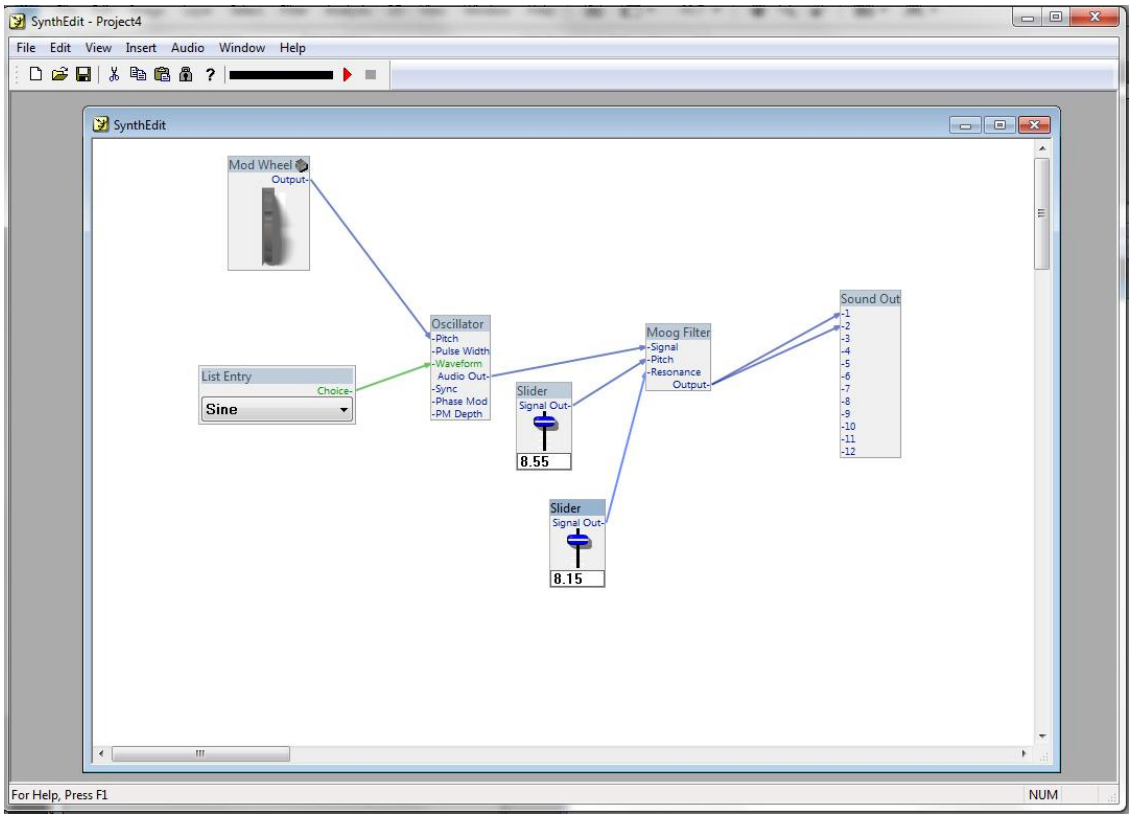
Program je na voljo kot *shareware*, kar pomeni, da ima brez nakupa licence nekatere omejitve pri shranjevanju in podpisovanju izdelkov, vendar časovne omejitve uporabe nima.

Tehnične specifikacije [22]:

- procesiranje notranjih signalov v 32 bitni plavajoči vejici;
- zapis izhodnih vzorcev s 16 bitnim celim številom ali 32 bitno plavajočo vejico;
- podpira vse frekvence vzorčenja;
- uporaba najvišje možne frekvence vzorčenja pri vseh notranjih signalih, kar pripomore k kvalitetnejšemu zvoku;
- za zvočni izhod uporablja tehnologiji *DirectSound* ali ASIO, kar omogoča nizko izhodno latenco;
- 6 glasna polifonija – omogoča predvajanje šestih zvok naenkrat, kar lahko povečamo na 128 z uporabo tehnologije MIDI;
- procesiranje vzorcev v blokih.

4.1.1 Delovno okolje

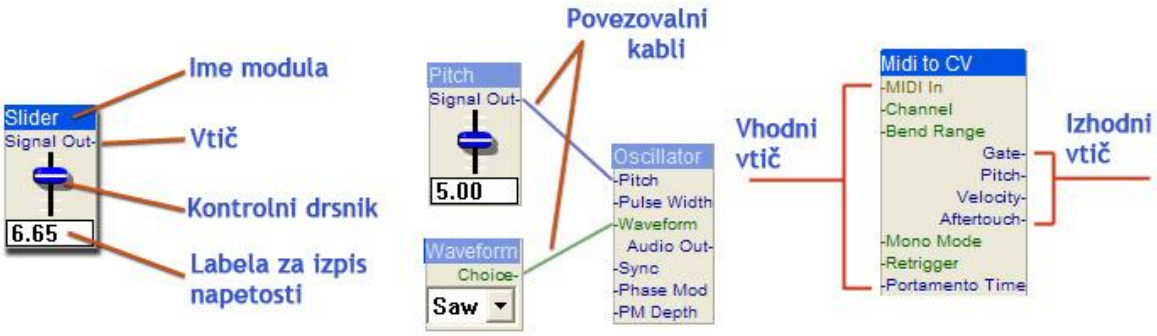
Za vsak nov model moramo ustvariti nov projekt, ki odpre novo, prazno strukturno okno. Vanj lahko sedaj vstavljamo module, kontrole modulov in povezovalne kable, ki bodo gradili virtualni sitetizator ali zvočni učinek.



Slika 20: Delovno okno programa SynthEdit s primerom enostavnega sintetizatorja zvoka

Slika 20 prikazuje osnovno delovno okolje programa SynthEdit in primer realizacije enostavnega sintetizatorja zvoka. Ta temelji na modulu oscilatorja, ki je element za generiranje tona. S spustnim menijem lahko definiramo obliko signala (sinusna, kvadratna, trikotna ali žagasta krivulja), z modifikacijskim kolesom pa višino tona (angl. *pitch*). Iz oscilatorja peljemo signal v filter, katerega lahko modificiramo z dvema drsnikoma.

4.1.1.1 Moduli, vtiči in povezovalni kabli



Slika 21: Zgradba in sestavni deli modula

Modul je osnovni gradbeni element. V strukturnem oknu nastopa kot pravokotnik z različnimi vhodi in izhodi. Lahko generira ali procesira zvočne in MIDI podatke, lahko je kontrolni element (drsnik, gumb), ali pa je kompleten sintetizator zvoka. Na sliki 21 je predstavljen modul oscilatorja, ki ima šest vhodnih vtičev na levi in en izhodni na desni strani. V modulu predstavlja vtič povezavo na katerega lahko s povezovalnim kablom (angl. *patch cord*) priključimo drug modul.

Obstaja veliko vrst prednastavljenih standardnih modulov, ki so logično razdeljeni v skupine glede na njihove lastnosti:

- moduli za upravljanje: gumb, drsnik, spustni meni, klaviatura,...;
- moduli za konvertiranje podatkovnih tipov: bool to float, bool to int, float to int, list to int, text to bool,...;
- moduli za učinke: zakasnitve, modulacije, popačenje,... ;
- moduli za logične operacije;
- moduli za filtre: HPF – *high pass filter*, LPF – *low pass filter*, *all pass filter*, *moog filter*,...;
- moduli za matematične operacije;
- moduli za ostale posebne kontrole ali funkcije.

Poleg standardnih modulov obstajajo tudi ostali moduli, ki so jih ustvarili uporabniki programa z uporabo SynthEdit SDK in so povečini na voljo na spletu brezplačno [23]. Ti prav tako obsegajo celotno področje pri obdelavi in sintezi zvoka (npr: enkoderji/dekoderji prostorskega zvoka, različni filtri, moduli, ki pomagajo pri razhroščevanju sitetizatorjev,...).

Vtiči v modul se razlikujejo po tipu podatka, ki ga lahko vanj vodimo. Vrste vtičev ločimo po barvi.

Vrste vtičev:

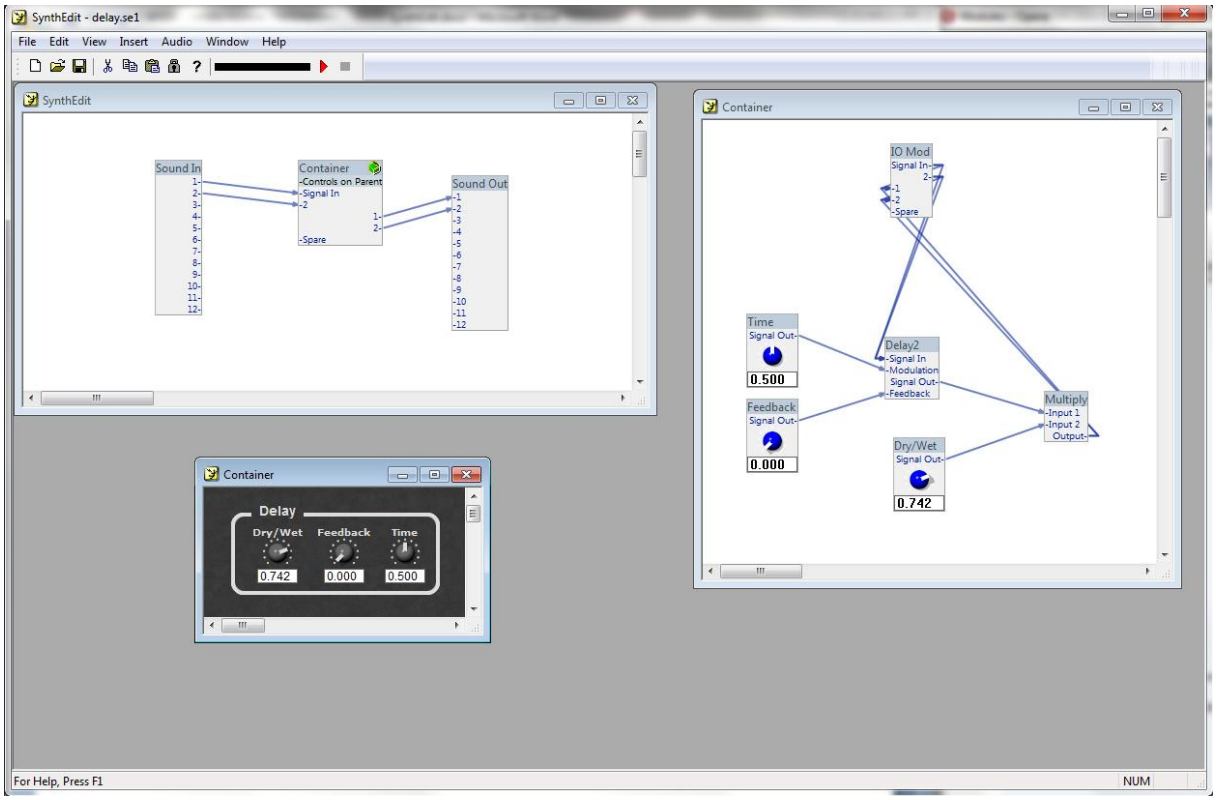
- napetostni vtiči: služijo za zvočne signale in kontrolne napetosti in imajo razpon 0 do 10V za kontrolne signale, ter od -10V do 10V za zvočne signale;
- vtiči za izbiro iz spustnega menija;
- vtiči za podporo plavajoči vejici;
- MIDI vtiči;
- tekstovni vtiči;
- rezervni vtiči;
- vtiči za grafični uporabniški vmesnik.

4.1.2 Implementacija vtičnika VST z učinkom zakasnitve s pomočjo programa SynthEdit

4.1.2.1 Modeliranje osnovne funkcionalnosti

Izdelava učinka zakasnitve s programom SynthEdit je enostavna. Večji del problema lahko rešimo z uporabo modula, katerega funkcionalnost je učinek zakasnitve. Modul zakasnitve ima 3 vhode - vhodni signal, modulacija (čas zakasnitve signala) in število ponovitev zakasnitve. Znotraj lastnosti modula lahko nastavimo maksimalni zakasnitveni čas v sekundah. Ker je glavni namen izdelave uporaba vtičnika VST z električno kitaro, je maksimalni zakasnitveni čas nastavljen na 1 sekundo. Modul sicer omogoča nastavitev časa zakasnitve do 10 sekund, vendar je sekunda zakasnitvenega časa dovolj za večino kitaristov, če le ne igrajo sila eksperimentalne glasbe.

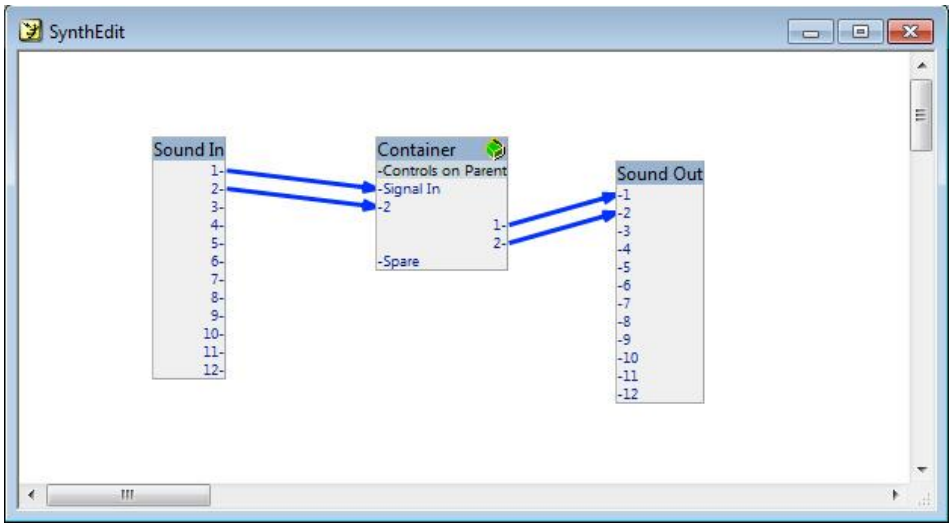
Vrednosti modulacije in trajanja zakasnitve lahko poljubno nastavljamo. To najlažje storimo z uporabo drsnikov, ki nam bodo omogočali tudi spreminjanje vrednosti v realnem času. Na zakasnitveni modul sta tako priključena dva drsnika, ki vsak posebej omogočata nastavljanje omenjenih parametrov. Ker vtičnik VST temelji na analognem modelu je v navadi, da se čas zakasnitve in trajanje nadzira z gumbi in ne z drsniki. SynthEdit omogoča enostavno pretvorbo drsnika v gumb z nekaj kliki, saj je analogija drsnika in gumba enaka, le izgled je drugačen. Videz tako spremenimo s posegom v lastnosti modula. Tako povezani moduli sedaj tvorijo funkcionalno celoto učinka zakasnitve.



Slika 22: Model učinka zakasnitve v programu SynthEdit

Iz modula zakasnitve peljemo signal v modul za množenje in množimo z vrednostjo novega drsnika, z razponom od 0 do 1. Vrednost drsnika nam pove kolikšen delež zakasnjenega signala peljemo na izhod. Tako dobimo kontrolo za učinek »*dry/wet*«, ki omogoča mešanje osnovnega, nezamaknjenega signala s sprocesiranim, zamaknjenim signalom. Seštejemo izhod množilnika in osnovni vhodni signal ter vsoto peljemo na izhod.

SynthEdit zahteva, da vsako strukturo, ki jo želimo shraniti kot vtičnik VST grupiramo v vsebovalnik (angl. *container*). Vsebovalnik je gradnik, ki lahko vsebuje več modulov. Ti definirajo določen sklop funkcionalnosti sintetizatorja, zvočnega učinka ali kontrolne enote. Znotraj vsakega vsebovalnika je vhodno/izhodni modul (angl. I/O *module*), ki oskrbuje vsebovalnik z zvočnim signalom. Taka je tudi struktura zmodeliranega učinka zakasnitve v vsebovalniku – slika 23. Vanj peljemo dva vhodna signala, ki prihajata iz modula *Sound In*, ki je modul za direkten dostop do zvočnih kanalov zvočne karte. Zaradi dveh vhodnih signalov lahko vtičnik VST procesira stereo vhodni signal. Prav tako je izhod enote stereo, kar omogoča modul *Sound Out*, v katerega pripeljemo dva enaka izhoda iz vsebovalnika.



Slika 23: Vsebovalnik in V/I moduli

4.1.2.2 Grafični uporabniški vmesnik

SynthEdit nam omogoča enostavno ustvarjanje grafičnih uporabniških vmesnikov. Program vse module, ki jih lahko spreminjamo v realnem času, doda v poseben vsebovalnik, ki je namenjen gradnji grafičnega vmesnika. Te lahko potem premikamo po delovni površini in poljubno razporejamo v skupine. Sama velikost okna nam tudi določa končno velikost enote VST. Podpira tudi nalaganje preoblek, ki imajo že prednastavljene grafike in tako omogoča enostavno izgradnjo grafičnih vmesnikov.

Slika 24: Grafični uporabniški vmesnik izdelan s programom SynthEdit

4.1.2.3 Kreiranje vtičnika VST

Tako zasnovan in oblikovan model lahko sedaj enostavno pretvorimo v vtičnik VST s funkcijo »Shrani kot VST«. Odpre se pogovorno okno, ki omogoča definiranje imena vtiča VST, imena datoteke (program kreira datoteko s končnico ».dll«), števila prednastavljenih programov (v neregistrirani različici števila ni mogoče spreminjati) in štirimestne identifikacijske kode izdelka, ki mora biti za vsak vtič edinstvena. V primeru, da je vtičnik generator zvoka (VSTi), je potrebno označiti parameter »*Plug-in is synth*«, ki bo obvestil gostitelja, da ima vtičnik sposobnost generiranja zvoka. Izmed ostalih je najpomembnejše še

potrditveno polje, ki omogoča, da v vtičnik peljemo mono signal, čeprav ima vtičnik dva vhoda. V tem primeru gostitelj pošlje vtičniku zgolj en kanal.

4.2 SynthMaker

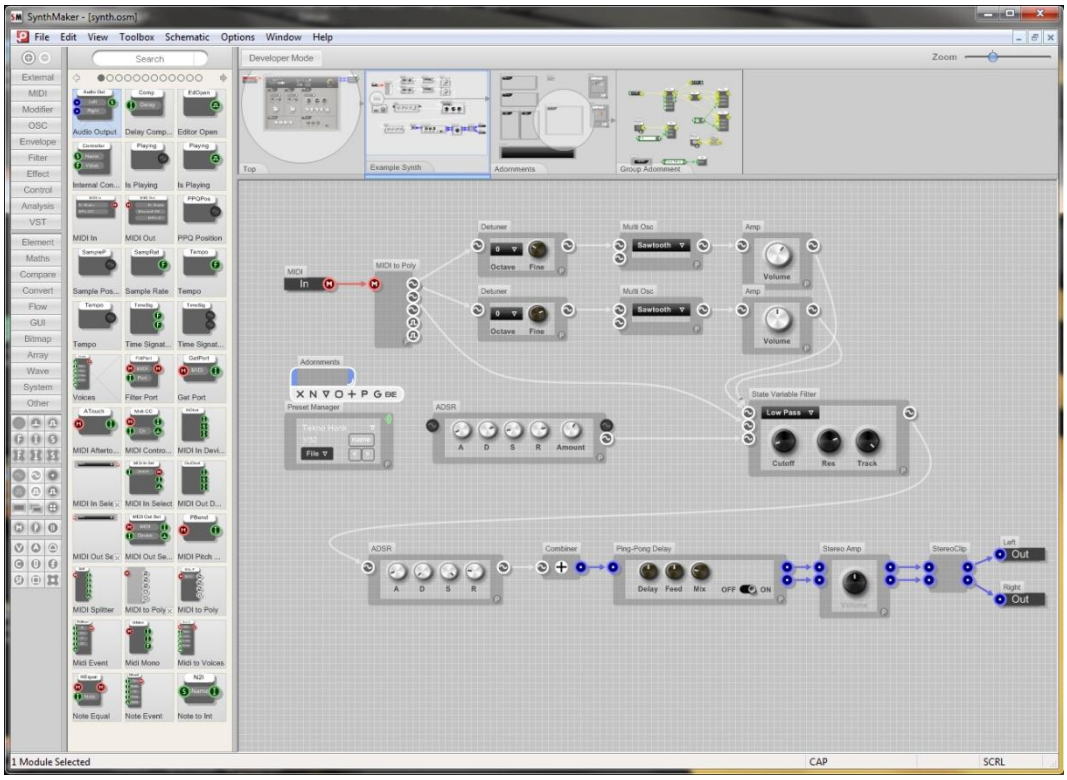
SynthMaker je grafično programsko okolje za kreiranje in manipuliranje z digitalnim zvokom. Razvilo ga je podjetje OutSim [24]. Funkcionalnost programa obsega kreiranje zvokov, virtualnih inštrumentov in učinkov s pomočjo vizualnega programiranja. Omogoča izvoz vseh projektov kot samostojnih inštrumentov VST ali učinkov VST, kot tudi samostojnih aplikacij. Podpora vtičnikom VST je tudi tukaj omejena samo na različico 2.4.

Osnovne lastnosti in zmožnosti programa SynthMaker so zelo podobne konkurentu SynthEditu. Omogoča modeliranje zvočnih sintetizatorjev, vtičnikov VST, razvoj grafičnega vmesnika, podporo tehnologiji MIDI. Bistvena razlika pa je ta, da je SynthMaker plačljiv. Na voljo je demo različica, ki se brez nakupa licence zaklene v 30 dneh, generirani vtičniki VST pa proizvajajo šumeč zvok.

4.2.1 Delovno okolje

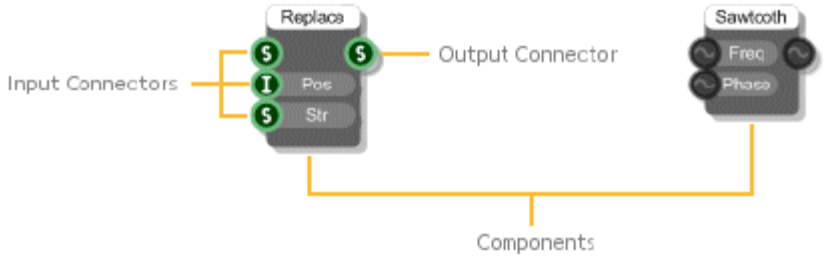
Tu je potrebno omeniti, da je poimenovanje elementov, ki imajo načeloma enako funkcionalnost, v dokumentaciji programa SynthMaker drugačno kot pri programu SynthEdit. Zamenjana sta pojma modula in vsebovalnika. Tako pri programu SynthMaker za poimenovanje osnovnih gradnikov (modulov pri SynthEdit) uporabljajo naziv komponenta. Pojem modula pri programu SynthMaker pa predstavlja vsebovalnik, ki vsebuje nek zaokrožen del sheme. Vtič je tu poimenovan konektor, povezovalni kabel pa zgolj povezava. Od tu naprej se bomo sklicevali na dokumentacijo programa SynthMaker in uporabljali njihovo terminologijo, kar pa naj bralca ne zmede.

Slika 25 prikazuje grafični uporabniški vmesnik programa SynthMaker z implementacijo sintetizatorja zvoka. Že na prvi pogled je očitno, da gre za podoben princip kot pri SynthEditu. V osrednjem oknu gradimo shemo željenega učinka ali inštrumenta z uporabo komponent iz knjižnice v levem delu programskega okna.



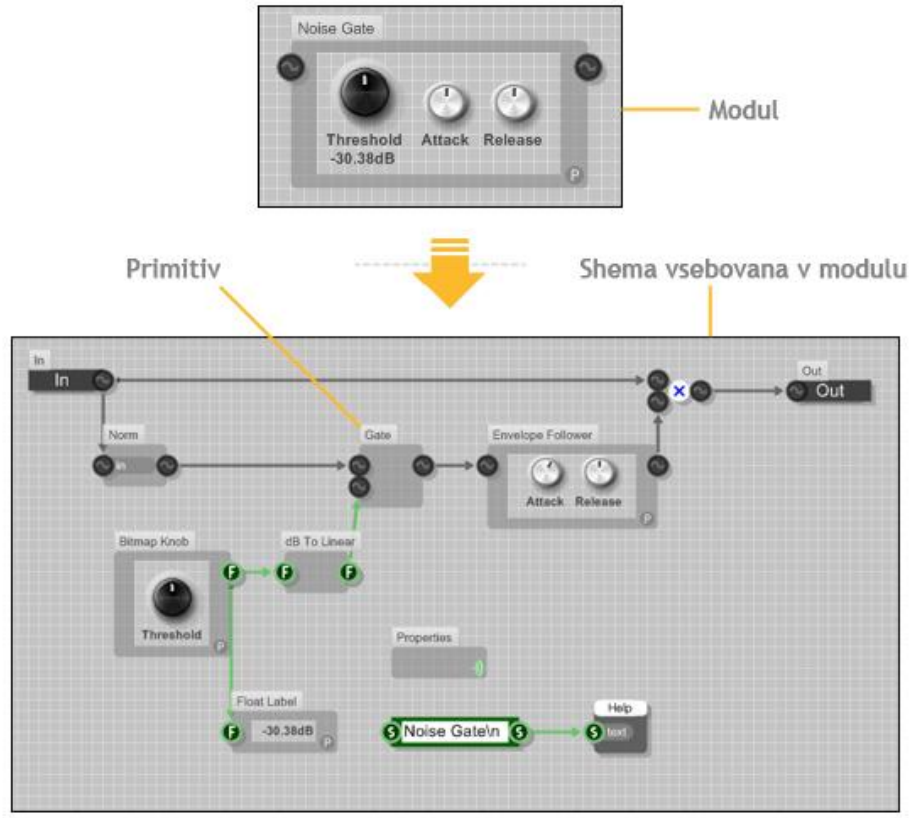
Slika 25: Osnovno okno programa SynthMaker in implementacija sintetizatorja zvoka

4.2.1.1 Komponente, konektorji in povezave



Slika 26: Osnovni komponenti in njihovi sestavni deli

Tu so osnovni gradniki komponente. Te lahko razdelimo na dva tipa – osnovne ali primitive in module. Osnovni imajo definirano notranjo funkcijo in lahko nanje gledamo kot na črno skrinjico, katere obnašanje ne moremo spreminjati. Predstavljeni so s pravokotnimi bloki znotraj katerih so konektorji za priključitev drugih komponent. Moduli pa so vsebovalniki, ki imajo funkcionalnost definirano z vsebujočo shemo zgrajeno iz primitivnih komponent in jo lahko spreminjamo.



Slika 27: Primer modula in njegove notranje sheme

SyntMaker ima prednaloženo knjižnico, ki vsebuje veliko število komponent. Baza teh komponent obsega oscilatorje, elemente s katerimi lahko upravljamo MIDI dogodke, envelope generatorje, filtre, učinke, kontrolne elemente, gradnike za osnovne matematične operacije, module za osnovne matematične operacije,...

Slika 28: Primerjava konektorjev (od leve proti desni): večvhodni konektor za zvočni signal, konektor za podatkovni tok s plavajočo vejico, konektor za modificiranje grafičnega vmesnika, mono konektor za zvočni signal, MIDI konektor

Na sliki 28 so predstavljeni osnovni tipi konektorjev. Za vsak tip obstaja podmnožica konektorjev, ki oskrbujejo komponento s potrebnimi informacijami. Tudi tu gre za podobne skupine kot smo jih imeli pri SynthEditu.

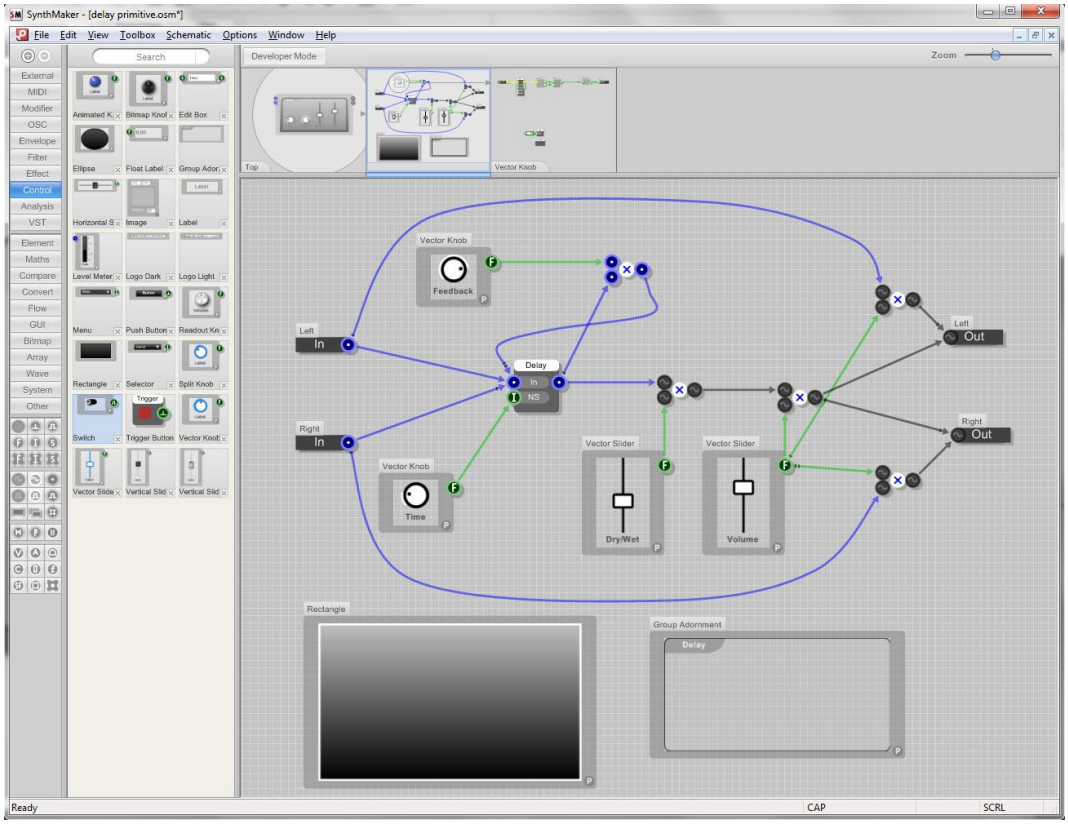
4.2.2 Implementacija vtičnika VST z učinkom zakasnitve s pomočjo programa SynthMaker

SynthMaker nam zaradi velikega števila vgrajenih komponent in modulov omogoča realizacijo enostavnega učinka zakasnitve na več različnih načinov. Obstajata dve komponenti in trije moduli, ki so namenjeni samo za implementacijo učinka zakasnitve. Seveda je možna

realizacija tudi z drugimi komponentami in primitivi, vendar za osnovno uporabo in študijo implementacije zadostujejo omenjene komponente in moduli.

Zaradi velikega števila komponent sem imel pri implementaciji učinka zakasnitve težave zaradi neintuitivne razporeditve komponent v knjižnici. Tudi iskalnik komponent ne pomaga veliko, saj ob iskanju osnovnih tipov vrne veliko množico komponent in izbora ne omeji, kar oteži upravljanje z njimi.

4.2.2.1 Modeliranje osnovne funkcionalnosti



Slika 29: Model učinka zakasnitve v programu SynthMaker

Glavni gradnik je primitiv *Delay*. Ta deluje tako, da shrani vhodne podatke v tabelo in jih zakasni za število vzorcev, definiranih z vhodom *NumberOfSamples*. Zgornja meja dolžine zakasnitve – dolžine tabele, pri tej komponenti je 262144 vzorcev, kar pri vzorčenju s frekvenco 44100 Hz predstavlja malo manj kot 6 sekund – enačba 11.

Ker je dolžina zakasnitve najbolj pomembna vrednost pri modeliranju tega učinka, mora biti uporabniku omogočeno spreminjanje te vrednosti. To realiziramo z uporabo gumba. Tu zopet ni smiselna uporaba celotnega časa zakasnitve, zato omejimo obseg vrednosti, ki jih vrača gumb na vrednosti od 0 do 88200. To pomeni, da lahko dosežemo zakasnitev za največ 88200 vzorcev, kar pri vzorčenju s 44100 Hz pomeni 2 sekundi – enačba 11. Tako pridobimo večji nadzor pri nastavljanju vrednosti zakasnitve, kot tudi manjšo porabo pomnilnika zaradi manjše tabele za shranjevanje vzorcev.

Takšna komponenta nam sedaj omogoči zakasnitev samo za eno ponovitev. Da bi lahko imeli več ponovitvenih vzorcev moramo narediti povratno zanko. Povratna zanka je sistem, ki ima izhodno povezavo vezano nazaj na vhod.

Pri gradnji povratne zanke je bilo za spreminjanje dolžine trajanja ponovitve potrebno uvesti gumb. Ta omogoča nastavljanje vrednosti od 0 do 1 v plavajoči vejici. Tako z množenjem izhoda komponente *Delay* in vrednosti gumba dobimo delež signala, ki ga peljemo nazaj na vhod komponente *Delay*. Tako modeliran sistem nam sedaj omogoča nastavljanje dolžine trajanja ponovitve vhodnega signala.

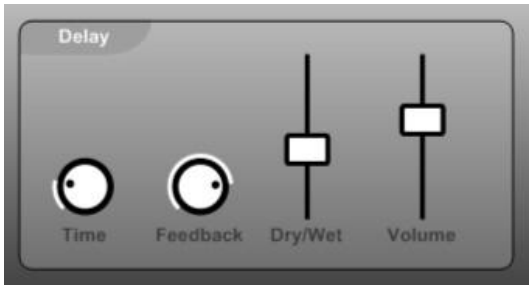
Kreiranje povratnih zank je prednost programa SynthMaker pred SynthEditom, saj slednji tega ne omogoča in ob poskusu njene realizacije vrne napako. SynthMaker omogoča procesiranje vhodnih podatkov s frekvenco vsakega vhodnega vzorca, medtem ko SynthEdit procesira vhodne podatke glede na blok – vsak blok vsebuje okoli 100 vzorcev. Tako bi SynthEdit ob uporabi povratnih zank ustvarjal zakasnitev, kar ne omogoča uporabe pravilnih povratnih zank.

Podobno kot pri realizaciji v programu SynthEdit, tudi tu uporabimo drsnik za učinek »*dry/wet*«. Vrednost izhoda iz komponente zakasnitve množimo z vrednostjo drsnika.

Tu je dodana še kontrola za nastavljanje glasnosti izhoda vtičnika. Zopet gre za množenje vrednosti novega drsnika z vsemi signalnimi potmi, ki jih peljemo na izhod.

4.2.2.2 Grafični uporabniški vmesnik

Vsak modul ima lahko uporabniški panel (angl. *front panel*) in predstavlja okno, kamor lahko dodajamo kontrole in prikazujemo grafike. Vsebovane komponente na panelu so definirane s shemo znotraj modula. Tako imamo v našem primeru na panelu dva guma in dva drsnika. Izgled gumba in drsnikov določimo znotraj sheme. Knjižnica vsebuje množico gumbov in drsnikov, katerim lahko spreminjamo lastnosti in izgled. Z uporabo grafične komponente na panel vstavimo še gradientno ozadje in komponento za združevanje posameznih objektov na panelu, ki služita le za lepši videz končnega grafičnega vmesnika.

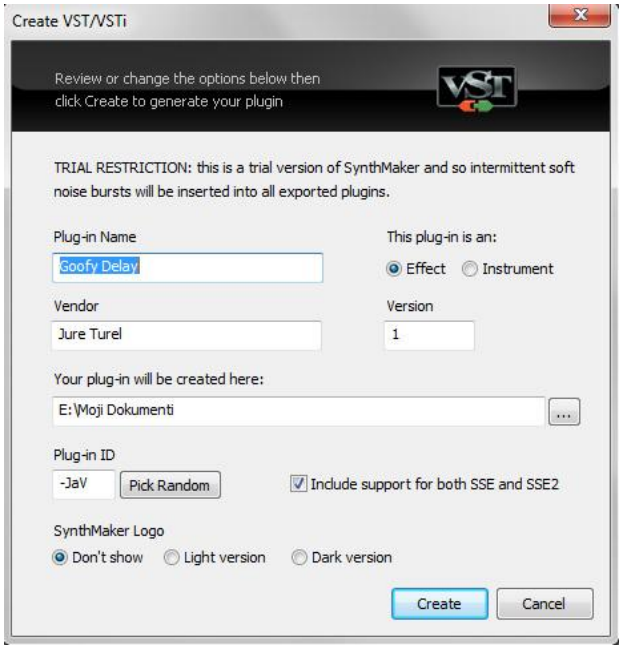


Slika 30: Grafični uporabniški vmesnik vtičnika VST izdelanega s programom SynthMaker

4.2.2.3 Kreiranje vtičnika VST

V nadzorni vrstici panela se nahaja gumb, ki omogoča izvoz narejene strukture v obliko vtičnika VST ali inštrumenta VST. Zopet je potrebno definirati osnovne parametre, ki

določajo lastnosti vtičnika – ime, tip vtičnika (učinek VST, inštrument VST), avtorja, verzijo, identifikacijsko številko in omogočenost podpore naborom ukazov SSE in SSE2.



Slika 31: Pogovorno okno za izvoz vtičnika VST

4.3 Program SynthEdit v primerjavi s SynthMaker

Programa sta si v svojih osnovnih sposobnostih, funkcionalnostih in načinu dela z njim zelo podobna. Pri obeh gre za orodje za izdelavo zvočnih sintetizatorjev in učinkov s pomočjo vizualnega programiranja. Ker imata oba vgrajene podobne komponente je z obema možno zgraditi enake aplikacije, enakih lastnosti. Pri uporabi dodatnih, uporabniško definiranih komponent pa ima SynthEdit rahlo prednost, saj obstaja velika množica modulov, ki omogočajo nove funkcionalnosti. SynthMaker ima prednosti v sposobnosti izvoza realiziranega modela kot samostojne aplikacije in pri uporabi povratnih zank.

Glede na osebno izkušnjo, bi izbral SynthMaker. Zaradi modernejšega programskega vmesnika mi osebno bolj odgovarja, saj omogoča lažje in preglednejše delo. Sicer vsebuje nekoliko nepregledno knjižnico elementov, vendar se ob vsakdanji uporabi ta pomanjkljivost ne pokaže.

	SynthEdit	SynthMaker
vrsta licence	»*shareware*« - plačljive nekatere dodatne funkcionalnosti	plačljiv – na voljo je 30 dnevna demo različica
omogoča modeliranje vtičnikov VST in inštrumentov VSTi	✓	✓
podpora tehnologiji MIDI	✓	✓

možnost izvoza modela učinka ali sintetizatorja kot samostojne aplikacije	✗	✓
kakovost in številčnost vgrajenih modulov/komponent	manjša množica - bolj pregledna	velika množica – neučinkovit sistem iskanja komponent
razvoj lastnih modulov/komponent	✓	✗
omogoča predvajanje zvoka sintetizatorja znotraj aplikacije	✓	✗
frekvenca procesiranja vhodnih podatkov	procesiranje v blokih (približno 100 vzorcev/blok)	procesiranje s frekvenco vsakega vhodnega vzorca
uporaba povratnih zank	✗	✓
ostalo	nekoliko zastarel programski uporabniški vmesnik	novejši programski uporabniški vmesnik

Tabela 1: Primerjava programov SynthEdit in SynthMaker

5. Sklepne ugotovitve

V diplomskem delu sem skušal bralcu približati področje zvočnih vtičnikov in omogočiti vpogled v nekatere današnje tehnologije in vrste implementacij le-teh. Na začetku smo si pogledali formate teh tehnologij, v naslednjem poglavju pa predstavili konkretne načine implementacije ene izmed najpopularnejših danes, tehnologijo podjetja *Steinberg*, VST – Virtual Studio Technology.

Čeprav smo primer implementacije učinka zakasnitve prikazali samo za format VST, bi lahko problem enostavno prenesli tudi na druge formate zvočnih vtičnikov. Opisano rešitev osnovnega algoritma učinka zakasnitve, implementiranega s pomočjo krožnega pomnilnika, bi lahko enostavno uporabili tudi pri drugih formatih, saj je ta enaka pri vseh različicah. Razlike bi nastajale v sami implementaciji procesnega in upravljalnega dela zaradi različnih arhitekturnih značilnosti ostalih formatov, osnovni algoritem učinka zakasnitve pa bi lahko z manjšimi popravki enostavno prepisali.

Za izbor formata je ključen razmislek na katerem operacijskem sistemu in v katerem digitalnem zvočnem urejevalniku se bo vtičnik uporabljal. Tako bi bil za operacijski sistem *Mac* OS X najprimernejši format *Audio Units*. Za operacijski sistem Linux pa bi lahko uporabili formata LADSPA ali LV2, vendar snemanje in produciranje glasbe v veliki večini primerov poteka v zvočnih urejevalnikih, ki tečejo na operacijskih sistemih *Windows* ali *Mac*. Tako podjetja, ki izdelujejo zvočne vtičnike v komercialne namene, običajno vsak tip vtičnika prevedejo v več formatov. To jim omogoča večjo tržno pokritost in višje prodajne odstotke.

Kot nadgraditev pričujočega dela bi lahko realizirali vtičnik z učinkom zakasnitve še v drugih formatih. Večina formatov vtičnikov uporablja odprt standard, kar pomeni, da obstaja razvojno okolje s pomočjo katerega lahko izdelujemo vtičnike določenega formata. Teoretično bi bil najbolj zanimiv (in najtežji) prenos v format TDM, kar pomeni, da bi bilo potrebno narediti namensko platformo v obliki strojne opreme, ki bi implementirala naš učinek zakasnitve. Po drugi strani pa bi bila takšna realizacija zaradi relativne enostavnosti in nizke procesorske zahtevnosti našega vtičnika nesmiselna. Veliko pomembnejša bi bila izboljšava, ki bi izpopolnila osnovni algoritem učinka zakasnitve. Poleg izboljšave omenjene v poglavju 3.8, ki bi odstranila popačenje izhodnega signala ob premikanju parametra za nastavljanje dolžine zakasnitve, bi lahko dodali nove parametre, ki bi omogočali nastavljanje nizko-prepustnega in visoko-prepustnega filtra. Takšni filtri so pri implementacijah učinka zakasnitve zelo pogosti. Vplivajo na prepustnost frekvenc ponavljajočega se signala in tako prispevajo k bolj bogati zvočni sliki, ki jo naredi uporaba takšnega učinka. Uporabna bi bila še funkcija, ki omogoča sinhronizacijo ponovitev s tempom, ki je določen znotraj zvočnega urejevalnika. Tako bi lahko uporabnik avtomatično nastavil čas zakasnitve v skladu s tempom.

Največja težava, na katero sem pri izdelavi naletel, je bilo pomanjkanje strokovne literature o vtičnikih. To sem najbolj pogrešal pri implementaciji s pomočjo razvojnega okolja SDK v3.0. Poleg tega, da je verzija vtičnikov VST 3 dokaj nov standard, je tudi njegova dokumentacija

narejena rahlo pomanjkljivo, kar otežuje delo z vmesniki, ki jih vsebuje. Tako so najboljši viri pridobivanja informacij razni spletni forumi, ki jih uporabljajo ostali razvijalci zvočnih vtičnikov.

Menim, da sem s predstavitvijo tehnologije, formatov in prikazom konkretne izdelave vtičnikov zastavljene cilje dosegel. Tako je lahko pričujoče delo bralcu v pomoč pri raziskovanju zvočnih vtičnikov.

Dodatek

Opis vsebine na priloženi zgoščenki

Na priloženi zgoščenki se nahaja celotna izvorna koda implementacije vtičnika s pomočjo VST SDK v3.0, kot tudi projektni datoteki programov SynthEdit in SynthMaker. Vsak del je v svoji mapi in vsebuje tako projektne datoteke, kot tudi končno različico izdelanega vtičnika VST.

- **Vsebina mape »VSTSDK3_Delay_Project«**

  Mapa, ki vsebuje izvorno kodo implementacije s pomočjo VST SDK v3.0 in končno verzijo vtičnika VST 3 (datoteka »delay.vst3«).

  Razvojno okolje in implementacijske datoteke se nahajajo znotraj mape »VSTSDK3_Delay_Project/vstsdk3«. Dokumentacija celotnega razvojnega okolja se nahaja v mapi »VSTSDK3_Delay_Project/vstsdk3/doc«.

  Vmesniki, ki se uporabljajo pri implementaciji, se nahajajo v »VSTSDK3_Delay_Project/vstsdk3/pluginterfaces«.

  Znotraj mape »VSTSDK3_Delay_Project/vstsdk3/public.sdk/samples/vst/delay/win« se nahaja projektna datoteka (»delay_vs2008.vcproj«) programa *Microsoft Visual Studio 2008*, ki povezuje vse potrebne razrede za implementacijo vtičnika VST 3. Sami razredi, ki vsebujejo izvorno kodo, pa se nahajajo znotraj mape »VSTSDK3_Delay_Project/vstsdk3/public.sdk/samples/vst/delay/source«. Ta vsebuje zaglavne, kot tudi implementacijske datoteke tako procesnega, kot upravljalnega dela vtičnika.

  Mapa »VSTSDK3_Delay_Project/vstsdk3/vstgui.sf« vsebuje vmesnike, ki omogočajo izdelavo grafičnega vmesnika po meri.

- **Vsebina mape »SynthEdit_Delay_Project«**

  Znotraj mape se nahaja izvorna datoteka (»delay.se1«), ki vsebuje projekt izdelanega učinka zakasnitve v programu SynthEdit. Izdelana je bila s pomočjo različice 1.1770 programa SynthEdit.

  Datoteka »Delay.dll« je končna verzija vtičnika VST 2.4.

- **Vsebina mape »SynthMaker_Delay_Project«**

  Znotraj mape se nahaja izvorna datoteka (»delay.osm«), ki vsebuje projekt izdelanega učinka zakasnitve v programu SynthMaker. Izdelana je bila s pomočjo različice 1.1.2 programa SynthMaker.

  Datoteka »Delay_sm.dll« je končna verzija vtičnika VST 2.4, narejena s pomočjo programa SynthMaker.

Seznam slik

Literatura

[1] (2010) *Plug-in (Computing).* Dostopno na: http://en.wikipedia.org/wiki/Plug-in_(computing)#Plug-ins_and_extensions/.

[2] (2010) *Digital audio workstation*. Dostopno na: http://en.wikipedia.org/wiki/Digital_audio_workstation/.

[3] (2003) *The audio Plug-ins*. Dostopno na: http://www.macmusic.org/articles/view.php/lang/en/id/62/The-Audio-Plug-ins/.

[4] (2010) *Using plug-in processors for professional sound*. Dostopno na: http://www.tweakheadz.com/plugins_for_audio.html/.

[5] (2010) *Musical Instrument Digital Interface*. Dostopno na: http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface/.

[6] (2010) *Steinberg's Technology*. Dostopno na: http://www.steinberg.net/en/company/steinberg_technology.html/.

[7] (2009) *Delphi VST*. Dostopno na: http://www.axiworld.be/vst.html/.

[8] (2010) *jVSTwRapper, Java-Based Audio Plug-ins*. Dostopno na: http://jvstwrapper.sourceforge.net/.

[9] (2010) *VST.NET*. Dostopno na: http://vstnet.codeplex.com/.

[10] (2009) *DirectX Plugin*. Dostopno na: http://en.wikipedia.org/wiki/DirectX_plugin/.

[11] (2010) *Sony Audio Plug-in Development Kit*. Dostopno na: http://www.sonycreativesoftware.com/download/devkits/.

[12] (2003) *Delphi DirectX SDK V4*. Dostopno na: http://www.cloneensemble.com/ddx_sdk.htm/.

[13] (2007) *AudioUnit Programming Guide.* Dostopno na: http://developer.apple.com/mac/library/documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40003278-CH1-SW2/.

[14] (2009) *Logic Pro 9, TDM.* Dostopno na: http://documentation.apple.com/en/logicpro/tdmguide/index.html#chapter=2%26section=0/.

[15] D. Rocchesso, *Introduction to sound processing*, 2003, pogl. 3 Dostopno na: http://profs.sci.univr.it/~rocchess/SP/sp.pdf/.

[16] U. Zolzer, *DAFX – Digital Audio Effects*, John Wiley & Sons, 2002, pogl. 3.

[17] (2010) *Understanding Delay Effects*. Dostopno na: http://emusician.com/mag/emusic_better_late/.

[18] (2010) *Steinberg's 3rd Party Developers*. Dostopno na: http://www.steinberg.net/index.php?id=48&L=1/.

[19] (2009) *Software Developement Kit Documentation*. Dostopno na: http://www.steinberg.net/index.php?id=48&L=1/.

[20] (2010) SynthEdit. Dostopno na: http://en.wikipedia.org/wiki/SynthEdit/.

[21] H.G. Fortune, P. Schaffauzer, D. Haupt, *Visual VST/i programming*, Wizoo Publishing, 2007.

[22] (2009) *SynthEdit Modules*. Dostopno na: http://www.synthedit.com/modules.htm/.

[23] (2010) *The SynthEdit resources page.* Dostopno na: http://www.numerisson.com/novaflash/synthedit_resources.htm/.

[24] (2010) *SynthMaker*. Dostopno na: http://en.wikipedia.org/wiki/SynthMaker/.