Vizualni programski jezik za sintezu zvuka i računalno stvaranje glazbe

Pošćić, Antonio

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Antonio Pošćić

VIZUALNI PROGRAMSKI JEZIK ZA SINTEZU ZVUKA I RAČUNALNO STVARANJE GLAZBE

DOKTORSKI RAD

Zagreb, 2019.

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ANTONIO POŠĆIĆ

VIZUALNI PROGRAMSKI JEZIK ZA SINTEZU ZVUKA I RAČUNALNO STVARANJE GLAZBE

DOKTORSKI RAD

Mentor:
prof. dr. sc. Danko Basch

Zagreb, 2019.

University of Zagreb
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Antonio Pošćić

VISUAL PROGRAMMING LANGUAGE FOR SOUND SYNTHESIS AND COMPUTER MUSIC CREATION

DOCTORAL THESIS

Supervisor:
Professor Danko Basch, PhD

Zagreb, 2019

Doktorski rad izrađen je na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva na Zavodu za automatiku i računalno inženjerstvo.

Mentor: prof. dr. sc. Danko Basch
Doktorski rad ima 139 stranica.
Doktorski rad br.:

O mentoru

Danko Basch rođen je u Rijeci 1967. godine. Diplomirao je u polju elektrotehnike te magistrirao i doktorirao u polju računarstva na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1991., 1994. odnosno 2000. godine.

Od svibnja 1992. godine radi na Zavodu za automatiku i računalno inženjerstvo FER-a. 2016. godine izabran je u zvanje redovitog profesora u trajnom zvanju. Sudjelovao je na tri znanstvena projekata Ministarstva znanosti, obrazovanja i sporta Republike Hrvatske, a bio je voditelj istraživačkog projekta: "Oblikovanje i implementacija programskih jezika specijalne namjene" koji je financiralo Ministarstvo znanosti, obrazovanja i sporta Republike Hrvatske. Također je sudjelovao u nekoliko projekata u suradnji s industrijom (u tri kao voditelj projekta). Objavio je tridesetak radova u časopisima i zbornicima konferencija u području modeliranja i simuliranja, simulacijskih jezika i algoritama za upravljanje memorijom računala.

Prof. Basch član je udruge ACM. Godine 1995. i 2000. primio je srebrne plakete "Josip Lončar" FER-a za posebno istaknut magistarski rad odnosno za doktorsku disertaciju.

About the Supervisor

Danko Basch was born in Rijeka in 1967. He received his B.Sc. in electrical engineering, M.Sc. and Ph.D. degrees in computer engineering from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1991, 1994 and 2000, respectively.

Since May 1992 he has been working at the Department of Control and Computer Engineering at FER. In 2016 he was promoted to full professor tenure. He participated in 3 scientific projects financed by the Ministry of Science, Education and Sports of the Republic of Croatia. He was the project leader of the research project: "Design and Implementation of Special Purpose Programming Languages" financed by the Ministry of Science, Education and Sports of the Republic of Croatia. He also participated in several projects in collaboration with industry (in three of them as a project leader). He published about 30 books and papers in journals and conference proceedings in the area of modelling and simulation, simulation languages, and computer memory management.

Prof. Basch is a member of ACM. He received silver medals "Josip Lončar" from FER for outstanding M.Sc. work in 1995, and for Ph.D. thesis in 2000.

Sažetak

Vizualni programski jezici često su zbog svojih paradigmi i značajki prilagođeni stručnjacima u neračunarskim područjima. Osmišljeni su tako da omoguće izražavanje vizualnim simbolima bliskima domeni za koju su namijenjeni, da budu jednostavni za upotrebu i da sakrivaju tehničke aspekte jezika od programera. U tom su smislu posebno zanimljivi i često korišteni vizualni jezici u području glazbe.

Postojeći vizualni jezici u domeni glazbe koriste audio signale, odnosno matematičkim formulacijama opisan zvuk kao temeljni tip podataka. Međutim, rad s audio signalima zahtijeva poznavanje obrade signala i programerske vještine kako bi se na željeni način moglo njima upravljati. Glazbenicima ovo otežava korištenje jezika. U radu se stoga predlaže jezik koji se oslanja na koncepte timbralnih atributa kako bi se približio vokabularu i predznanju glazbenika. Jezik pritom pokušava zadržati osnovni koncept rada koji je glazbenicima poznat iz postojećih jezika.

Kako bi se ostvario novi jezik, istraženi su različiti postupci preslikavanja timbralnih atributa u parametre audio sintetizatora. Zatim su timbralni atributi uklopljeni u tok podataka jezika sličan onome kod postojećih rješenja. Većina uobičajenih blokova i funkcionalnosti takvih jezika prilagođena je za rad s timbralnim atributima. Prilikom oblikovanja, validacije i evaluacije predloženog jezika korištene su metode kvantitativne i kvalitativne analize pomoću anketa te formalne analize pomoću kognitivnih dimenzija.

Dodatno, predložena su specifična poboljšanja sučelja blokova u jeziku tako da olakšaju manipulaciju timbralnim atributima. Predstavljena su i rješenja za vizualizaciju rezultata programa i izlaznog zvuka u domeni timbralnih atributa.

Konačno, predloženo je rješenje za vizualno uspoređivanja programa čiji je cilj lakše praćenje promjena među inačicama programa kao i otkrivanje pogrešaka.

Ključne riječi

vizualno programiranje, oblikovanje programskih jezika, stvaranje glazbe, sinteza zvuka, vizualizacije, timbralni atributi, glazbena tehnologija

Visual Programming Language for Sound Synthesis and Computer Music Creation

Introduction

Visual programming is a method of programming in which the user, or programmer, creates and input concepts into a computer system by manipulating graphical objects and symbols, by placing them and moving them on a workspace, and by controlling the visual characteristics of the language. Potential benefits of the visual paradigm are a consequence of the human psyche's preference towards visual representations and interactions through specific, easily understandable syntactic elements like music instrument symbols. Even if textual languages have proven to be dominant throughout the history of computing, visual programming languages distinguish themselves by being flexible and by having expressive and explicit syntaxes. Because of these features, they are adapted especially well for usages in specific domain and are, in turn, more approachable to experts in fields outside of computing. During their development, visual languages have proven especially useful in computer-enabled multimedia creation, specifically music. The latter field is in the focus of this research.

Visual programming in the context of music is a field that encompasses a large variety of approaches, from sound synthesis, interactive audio design and automation of composition techniques through music analysis to development of interactive multimedia systems. Results of existing research show that a significant number of composers, performers, and multimedia artists is inclined to use visual instead of textual languages. They find textual languages more complex and difficult to understand.

Although visual interfaces between computers and users have the potential to be more intuitive and direct than textual, they still require specific knowledge, skills, and the ability to reason analytically. Without these elements paradigms of programming language cannot be fully understood. That's why the main goal of this doctoral research is to lower the initial threshold that needs to be overcome when working with such languages in the domain of music creation and sound synthesis. After we quantitatively and formally - using the cognitive dimensions framework - analyzed existing visual programming languages in the field of music, we identified and defined elements that musicians and composers find most important. We also explored areas of visual programming in music that could be receptive of improvements and contributions. Based on the results of our analysis, we modeled the concept of a new language that seeks to mitigate shortcomings of existing solutions. The main

contribution compared to similar languages is the use of timbral attributes in place of usual audio signals in the data flow. To make this approach viable, in this work, as support for the proposed language, we also explored algorithms for mapping timbral attributes into sound synthesis parameters.

As part of this work, we present three contributions. The first and main contribution is the model of a new timbral attribute flow based visual programming language which aims to make sound synthesis and music creation more accessible and straightforward. The other two contributions include additional improvements to auxiliary elements of the visual programming languages and that can simplify its use. These improvements are implemented on elements for sound synthesizer parameter input – timbral attribute input, to be specific – elements for visualizing sound synthesis results, and finally elements for comparison of visual programs and displaying differences between them in the visual domain.

Existing research

In this chapter we present existing research pertaining to the various contributions of the doctoral work.

First we explore various papers and research in the field of visual languages for multimedia applications. Here we focus on languages that are primarily used for music creation and related processes. We explain how audio and video processing, audio signal manipulation, and interfacing with physical tools (like keyboards) are translated into syntactic and semantic elements.

When languages for music are concerned, we present five of the current most popular, influential, or important languages. The languages presented – Pure Data, Max/MSP, OpenMusic, Symbolic Sound Kyma, and Native Instruments Reaktor – have been selected based on their paradigms, user bases, and areas of application. The same languages and tools were later analyzed in detail and the new language presented in this PhD thesis is partially based on their properties. In this sense, Pure Data and Max/MSP can be considered especially important as their baseline paradigm was used as a starting point for the new language.

After presenting existing visual languages, we recapitulate the various algorithms and processes that have been explored in terms of timbral attributes and translating timbral attributes into sound synthesis parameters. Since timbral attributes are an integral part of the proposed language, this subject is explored in-depth. The most prominent methods and means of control of sound synthesis are also briefly mentioned.

Then we give an overview of different methods of sound visualization and representation divided into two categories: precise visualizations and abstract, artistic approaches. We also look at the different ways that other studies approached problems with extracting sound features used during visualization.

Finally, we present the scarce existing work on the subject of comparing visual programs in the visual domain.

Visual programming language for sound synthesis and computer music creation

The model, architecture, and evaluation of a novel visual programming language for sound synthesis based on attribute flow and for computer music creation based on time-domain manipulations are the focus points of this chapter.

Before presenting the new visual language, the chapter explains the main motivation and approach behind the language. Here we emphasize its goals: ease of use, enabling new modes of creation, and incremental improvements compared to existing languages.

Quantitative and formal analysis of existing languages (Pure Data, Max/MSP, OpenMusic, Kyma, and Reaktor) is thus presented first and is later used as basis for the new language. As part of the quantitative analysis, we conducted interviews and a survey of users of visual programming languages in music. Then we analyzed the data statistically and produced a wealth of quantitative evidence and conclusions on two topic. The first topic was the desireable aspects of the visual languages, while the second was how developers, educators, and users influenced each other, with a special emphasis put on the importance (or lack thereof) formal education.

The formal analysis of existing solutions was based on the cognitive dimensions framework. We surveyed, experimented with, and analyzed the five aforementioned languages and graded them based on the fourteen main cognitive dimensions (abstraction gradient, closeness of mapping, consistency, etc.). To better understand and analyze these languages, we also conducted additional interviews with expert users and corroborated some of our conclusions with data from the quantitative analyses mentioned in the previous paragraph. The various cognitive dimension values were presented relatively between the languages with the help of radar charts.

With the analysis of existing solutions completed, we present and document in detail both the model and modeling process of the new language. Additionally, we explain how the language builds on top of existing paradigms to maintain good practices and characteristics,

while introducing significant changes and potential benefits. Here we focus on the influence of timbral attributes and expansions like implementing a timeline element in the vein of Kyma.

Afterwards, we explore sounds synthesis based on timbral attributes. In short, we present a novel method of translation between timbral attributes and audio signals and we demonstrate how these techniques will be used within the new language. Three methods are presented of controlling the sound synthesizer: using evolutionary algorithms, using fuzzy logic, and control independent of the sound synthesis type. Here we also explain how timbral attributes fit into the visual language syntax and semantics.

After briefly explaining how the new language is to be used to synthesize sounds and create music, we explain, in detail, the architecture, elements, and basic features of the language. Programs, program elements, program executions, and additional language elements are described. The basic language objects/blocks are also analyzed in detail, with special attention to sound synthesis and input/output objects. The timeline element is explained and a sketch is presented.

Once the conceptual and formal presentation of the language are completed, the prototype of the language is briefly explained. Based on Pure Data, it tries to implement most of the earlier detailed functionalities and features. Especially important are the ways in which the prototype is modified to make use of timbral attributes. These changes are reflected both in the execution and display routines of Pure Data.

Based on the prototype, in this chapter we finally evaluate the new language using the same principles and methodolody that were used to evaluate existing solutions. The qualitative evaluation shows good acceptance of the language's novel elements, both in the population of beginner and expert users. Formally speaking, the value of certain cognitive dimensions (closeness of mapping and abstraction gradient, for example) is relatively improved in comparison to a median value of existing solutions.

Intuitive control of visual language elements and vizualization of sound synthesis results with timbral attributes

In this chapter we present models of exploration and vizualization of sound synthesis results and for the intuitive control of a sound synthesizer and its parameters.

After transitioning to timbral attributes instead of audio signals as the main flow data type, the language's syntactic elements needed to be adapted. Specifically, to make the new visual language, that uses timbral attributes instead of audio signals in its core, more user-

friendly, we had to devise and adapt the input and output interfaces of blocks and objects. We also had to create various methods of visual representation of the results of a program's execution. In other words, to translate the internal representation of data into a better user experience and improved interactions, we had to display it in an expressive manner in the visual domain and in the context of human-computer interaction. This problem was divided into two categories.

The first category includes graphical interfaces through which users input values of timbral attributes on specific blocks. These interfaces need to be well defined, clear, expressive, and intuitive. Benefits were attained by virtue of using timbral attributes instead of having to define raw sound synthesis and audio signal parameters. Additionally, various graphical elements for timbral attribute value input are considered. Mapping timbral attributes into sound synthesis parameters is a basic problem related to these procedures and is thus investigated and solved in this chapter.

The second category of problem solved in this chapter is visualizing the program execution results which are usually given in the form of synthesized sound or music. Even if languages like Pure Data and OpenMusic contain several ready made objects for the extraction and displaying of audio signal features, they are mostly oriented towards low-level information like frequency or spectrum.

For both these categories, we give an in-depth explanation of how the solutions were modeled. The evaluation of the presented solutions - VU meters, spectrograms, sliders, and knobs adapted to work with timbral attributes - was not performed separately. Instead, it was given through the evaluation of the prototype of the new language as a whole.

Comparing visual programs for sound synthesis and music creation

The idea of improved and domain-adjusted methods for visual programs comparison is explored in this chapter.

Solutions for comparison of visual programs are usually tightly coupled with specific languages. By acknowledging the lack of such tools in the field of visual programming for music and based on the quantitative analysis we conducted, we understood that such a solution was important in terms of visual language usability. It had the potential to both help novice and expert users during programming, error detection, and development of larger programs.

The main problems encountered in this part of the research were a) how to determine which changes took place between two versions of a program and b) once detected, how to display these changes in the visual domain.

We present a solution that's custom tailored for the newly introduced visual language for music. The modeling and forming process of the functionality is explained in detail and based on data collected through quantitative, qualitative, and formal analysis. The concept of the solution is based on two steps. In the first, changes are detected either in the internal structure of the program or its XML representation. Then the detected changes are shown on the newer version of the program by using colors and other graphical elements to indicate which connections have been deleted, which objects have been added, etc.

The prototype was implemented as part of the prototype of the whole language and was evaluated as such. Based on the surveyed responses, the introduction of this functionality is seen as benefitial to the overall usability of the language.

Discussion

In this chapter we first consider the positioning of the new language in the existing ecosystem of visual programming for music and we try to envision how presented solutions could be applied generally. We also explore how the various methodologies introduced in the PhD thesis provide a contribution of their own. For example, the conducted quantitative analysis can be applied to other areas of visual programming, while the data it produced can be used in future studies.

An important benefit of our research is the use and manipulation of timbral attributes which can be extended beyond the new language and applied on other problems in the field of sound synthesis.

Finally, we consider the impact of the new language on creativity. It remains unclear if the new language, which is simpler to use, hinders, improves, or affects creativity in any way since some users might relish the challenge of more difficult languages. To fully assess this perspective, the language needs to be fully implemented and made available. Only then, after a large number of users create a large number of works in it, could we try and really understand its impact on creativity.

Conclusion

By modeling and validating a new visual programming language for sound synthesis based on timbral attributes and music creation based on manipulation in the time domain and auxiliary

functionalities, we have fulfilled the premises set at the beginning of our research. These premises included the realization of significant improvements and contributions in the domain of visual programming for music, solving several typical problems of visual programs, and a general evolution of languages in the domain of music in terms of user experience and adaptability to the needs of musicians and composers.

Apart from providing a detailed analysis of the language's individual elements and basic mechanisms, we also rigorously documented, verified, and validated each step of its creation. Significant contributions have been extracted through all phases of language modeling, from the results of quantitative analysis of existing solutions to algorithms of mapping timbral attributes and their use in the data flow model.

Despite the thesis's focus being on the domains of music and multimedia, and not dealing with visual programming problems in general, some presented solutions can be generalized. The most important such contributions are the methodologies used in certain phases of the work, mechanisms of visual program comparison, and improvements to the interfaces for value input and result visualization. In general, the principles of this research of work can be applied in other domains like electronic circuit modeling.

Because of the interdisciplinary nature of the field, complexity of certain evaluations, and to facilitate the analysis of the new language through various perspectives, during the research we used methods of result verification and validation belonging to different branches of science. Apart from technical validation which was based on a minimal prototype in Pure Data, validations were mostly reliant on procedures and principles from psychology. We interviewed and surveyed users and used the raw data collected through them for quantitative and qualitative analysis. This detailed analysis produced a vast collection of data, information, and conclusions that were previously unavailable in the field and that could guide future research and development. In that sense, the contribution of this work is wider than just the model of the new language.

Keywords

visual programming, programming language design, music creation, sound synthesis, visualisations, timbral attributes, music technology

Sadržaj

1 Uvod

Vizualno programiranje način je programiranja u kojem korisnik, odnosno programer, stvara i unosi koncepte u računalni sustav manipuliranjem grafičkih objekata i simbola, njihovim razmještanjem na radnoj površini i upravljanjem vizualnim karakteristikama jezika [1]. Potencijalne prednosti vizualne paradigme proizlaze iz naklonjenosti ljudske psihe prema vizualnim reprezentacijama te interakcija putem specifičnih, lako razumljivih sintaktičkih elemenata poput simbola glazbenih instrumenata. Iako su u povijesti računarstva tekstni jezici dominantni, vizualni programski jezici ističu se prilagodljivošću te izražajnim i eksplicitnim sintaksama. Zbog tih su značajki posebno pogodni za primjene u specifičnim domenama i pristupačniji stručnjacima u neračunarskim područjima. Tijekom njihova razvoja vizualni su se jezici pokazali posebno pogodnima za korištenje za računalom podržano stvaranje multimedije, posebice glazbe. Potonje čini fokus ovog istraživanja.

Vizualno programiranje u kontekstu glazbe je područje koje obuhvaća široki raspon pristupa, od sinteze zvuka, interaktivnog audio dizajna i automatizacije skladateljskih tehnika preko glazbene analize do razvoja interaktivnih multimedijalnih sustava [2]. Rezultati prethodnih istraživanja [3] pokazali su da je značajan broj skladatelja, izvođača i multimedijalnih umjetnika sklon korištenju vizualnih umjesto tekstnih jezika koje smatraju složenijima i teže razumljivima.

Iako vizualna sučelja između računala i korisnika mogu biti intuitivnija i direktnija nego tekstna [4][5] ona i dalje zahtijevaju specifična znanja, vještine i mogućnost analitičkog razmišljanja. Bez tih se elemenata paradigme programskih jezika ne mogu u potpunosti razumjeti. Zbog toga je glavni cilj ovog doktorskog istraživanja smanjenje početnog praga koji je potrebno savladati pri radu s takvim jezicima u domeni stvaranja glazbe i sinteze zvuka. Kvantitativnom analizom postojećih jezika za vizualno programiranje u području glazbe i evaluacijom prema modelu kognitivnih dimenzija, identificirani su i definirani elementi koji su glazbenicima i skladateljima najbitniji te segmenti u kojima postoje mogućnosti za napredak i poboljšanja. Prema rezultatima tih analiza oblikovan je koncept novog jezika koji stremi poboljšanju nekih manjkavosti u postojećim rješenjima. Glavna novost u odnosu na slične jezike je korištenje timbralnih atributa umjesto uobičajenih signala kao temeljnog oblika u toku podataka. Kako bi taj pristup bio ostvariv, u radu su kao podrška za predloženi jezik također istraženi postupci za mapiranje timbralnih atributa

U okviru ovog rada prezentirana su tri doprinosa. Prvi i glavni je model vizualnog programskog jezika temeljenog na toku timbralnih atributa čija je prvotna namjena razumljivije i jednostavnije upravljanje sintezom zvuka i stvaranje glazbe. Preostali doprinosi uključuju dodatna poboljšanja popratnih elemenata vizualnog jezika koji mogu olakšati njegovo korištenje. Ta se poboljšanja odnose na elemente za unos parametara sintetizatora, odnosno unos timbralnih atributa, elemente za vizualiziranje rezultata sinteze zvuka te elemente za usporedbu različitih programa, odnosno predstavljanje razlika među programima.

U nastavku poglavlja ukratko su prezentirane najbitnije značajke i predstavnici vizualnog programiranja u glazbi, prodiskutirani su njihovi nedostatci, definirani su temeljni doprinosi rada te su na kraju pojašnjeni opseg, metodologija i struktura istraživanja.

1.1 Vizualni programski jezici i domene primjene

Vizualni programski jezici su, za razliku od tekstnih jezika, višedimenzijski budući da se za definiranje semantike programa koristi više od jedne dimenzije [6]. Najčešće je riječ o prostornim dimenzijama i vremenskom slijedu pomoću kojih se određuje uzajamna zavisnost pojedinih elemenata jezika. Upravo su spomenute dimenzije i odnosi među grafičkim elementima (simboli, poveznice, itd.) leksičke jedinke (eng. *token*) kod vizualnih programskih jezika. Nizove ili skupine leksičkih jedinki nazivamo vizualnim izrazima (eng. *visual expression*). Kod vizualnog se programiranja, dakle, programira mijenjanjem vizualnih značajki i informacija (najčešće u dvije ili tri dimenzije): pomicanjem elemenata, povezivanjem elemenata, mijenjanjem značajki elemenata, itd.

Motivaciju pri razvoju vizualnih programskih jezika treba potražiti u činjenici da su ljudi skloniji vizualnom razmišljanju te da se programiranje želi učiniti dostupnim većem broju korisnika (inženjerima građevinarstva, ekonomistima, itd.) [7]. Drugim riječima, želi se omogućiti programiranje onim korisnicima kojima primarna zadaća nije samo programiranje i razvoj programske podrške, već se žele usredotočiti na olakšano ostvarivanje nekih drugih rezultata programiranjem (npr. izračuni vezani uz značajke motora u strojarstvu). Takav pristup nazivamo programiranjem za krajnje korisnike (eng. *end-user programming*) [8].

Osim jednostavnosti korištenja za krajnje korisnike, važno je razmotriti i mogućnost da se neki zahtjevni problemi mogu lakše i učinkovitije riješiti vizualnim pristupom. Tu je posebno važno područje programskih jezika temeljenih na toku podataka koji u sprezi s grafičkim prikazom omogućuju pojednostavljeno rješavanje problema paralelizacije programa [9]. Također je važno spomenuti da je za pojedine zadaće, poput prikazivanja znanstvenih

podataka i rezultata simulacija pojedinih sustava, posebno pogodna primjena vizualnog programiranja [10].

Tijekom povijesti razvoj se vizualnih programskih jezika uglavnom temeljio na dva principa: vizualnom pristupu koji se oslanja na tradicionalno, tekstno programiranje i na potpuno novim paradigmama koje su bile pripremljene isključivo za vizualno orijentirane metode. Pokazalo se da je potonji pristup ipak nedovoljno praktičan za većinu stvarnih primjena te danas prevladavaju paradigme vizualnog programiranja koje zadržavaju neke temeljne značajke tekstnog programiranja (npr. dijagrami toka koji prate slijed naredaba) [11].

Važan čimbenik kod razvoja vizualnih jezika bila je i mogućnost da se vizualno programiranje prilagodi pojedinim domenama i pojedinim zadatcima unutar domena. Takav je pristup doveo do razvoja okruženja za vizualno programiranje (eng. *visual programming environment*). Važno je razlikovati jezike za vizualno programiranje od spomenutih okruženja. Dok su vizualni programski jezici zasebni jezici koji imaju vlastiti skup razvojnih alata (eng. *toolchain*), okruženja za vizualno programiranje često predstavljaju zasebne alate koji služe kao dodatni sloj koji se oslanja na tekstne jezike (npr. Microsoft Visual Studio za C#, C++, itd.). U tom se slučaju vizualni izrazi koriste samo kao prečaci za stvaranje ili mijenjanje kôda u tekstnom jeziku.

1.1.1 Specifičnosti paradigme

Tijekom razvoja vizualnih programskih jezika, niz istraživačkih napora posvećen je pitanju općenite definicije sintaksnih i semantičkih značajki vizualnih programskih jezika [1][4][5][7][9] čime su postavljeni temelji za sadašnji i budući razvoj vizualnih programskih jezika, a određeni su i obrasci i pristupi koje je poželjno koristiti kod razvoja novih vizualnih programskih jezika.

Cilj vizualnih programskih jezika iznesen u uvodu, a vezan uz omogućavanje širem krugu korisnika da lakše, brže, efikasnije i točnije pišu programe, ostvaruje se kroz nekoliko strategija [6]:

- Specifičnost (eng. *concretness*): za razliku od apstrakcije tekstnih jezika, nužno je pojedine elemente jezika prikazati na specifičan i jednoznačan način pomoću konkretnih instanci čime su omogućene izravne promjene nad tim elementima.
- Izravnost (eng. *directness*): omogućujemo izravne promjene nad elementima, odnosno objektima u jeziku (primjerice, korisnik izravno mijenja neku značajku umjesto tekstnog opisivanja te značajke).

- Nedvosmislenost (eng. *explicitness*): nužno je da semantika programa bude izravno definirana bez da korisnik mora sam zaključivati o odnosima objekata i vrijednosti (na primjer, jednosmjerne veze među blokovima nedvosmisleno definiraju njihovu povezanost).
- Trenutni vizualni povratni učinak (eng. *immediate visual feedback*): sve promjene učinjene u programu moraju postati odmah vidljive na prikazu programa (promjena vrijednosti svojstva nekog elementa mora uzrokovati promjenu grafičkog prikaza bloka). Postoji nekoliko različitih razina povratnog učinka koje ovise o obliku i tipu prikazanih informacija.

Uz spomenute strategije uvedeni su i pokazatelji koji definiraju uspješnost nekog vizualnog jezika s obzirom na prethodno spomenute ciljeve korištenjem kognitivnih dimenzija (eng. *cognitive dimensions*) [4]. Te dimenzije predstavljene su kroz mjere poput apstrakcijskog gradijenta (eng. *abstraction gradient*) koji definira razinu apstrakcije jezika, dosljednosti (eng. *consistency*) koja opisuje koliko su sintaksa i semantika jezika dosljedne, viskoznosti (eng. *viscosity*) koja opisuje koliko je teško učiniti jednu promjenu, itd. Pomoću ovih mjera moguće je bolje razumjeti pojedine prednosti i nedostatke jezika za vizualno programiranje, a o čemu će biti više riječi u sljedećem poglavlju.

1.1.2 Prednosti i nedostatci

Tektsni su jezici danas i dalje dominantni. Iako su vizualni programski jezici u različitim izvedbama i granama djelovanja uvedeni još osamdesetih godina dvadesetog stoljeća, nije ostvarena sveprisutnost i prodiranje do većeg broja krajnjih korisnika [12]. S druge strane, alati za vizualno potpomognuti razvoj programa u tekstnim jezicima, kao što su alati za razvoj grafičkih sučelja, posljednjih godina bilježe uzlet popularnosti. Kako bismo bolje razumjeli razloge koji su doveli do današnjeg stanja, navedimo neke specifičnosti, negativne i pozitivne, vizualnih programskih jezika [1][13][14].

Jedna od glavnih prednosti vizualnih programskih jezika nad tekstnim jezicima jest manji broj koncepata i sintaktičkih elemenata koje je potrebno usvojiti [1][7]. Ta prednost proizlazi iz strategije specifičnosti koja je opisana u prethodnom poglavlju. Korisnik nije prisiljen učiti složene izraze kako bi opisao temeljne programske elemente već su mu ti elementi neposredno predstavljeni. Ipak, neki radovi [11] osporavaju ovaj fenomen te ističu da korisnici izloženi vizualnim programskim jezicima ne postižu bolje rezultate od onih korisnika koji rade s tekstnim jezicima.

Nekoliko se pojedinačnih prednosti vizualnih programskih jezika, poput trenutnih promjena na samom grafičkom prikazu, uvida u stanje programa i veza među elementima programa, može smatrati nedjeljivom cjelinom. Skup tih značajki omogućuje izravnost pri radu s određenim programskim jezikom te smanjuje prag prisutan u trokutu između problema iz neke domene, njegovog intuitivnog rješenja te implementacije rješenja.

Osim prednosti vezanih uz olakšavanje programiranja za krajnje korisnike, vizualno programiranje prikladno je za ona područja u kojima je određene koncepte i probleme prirodnije i učinkovitije prikazati vizualnim, a ne tekstnim apstrakcijama. Najznačajniji primjer takvog područja jest paralelno programiranje gdje jednostavnim račvanjem prikazujemo paralelni tijek izvođenja programa. Kombinacijom vizualnog programiranja i jezika temeljenih na toku podataka moguće je i automatizirati paralelizaciju programa [9].

Od glavnih nedostaka jezika za vizualno programiranje, ističu se neučinkovito korištenje radnih površina kao i teško održavanje većih programa. Viskoznost vizualnih programa često je velika te je i manje promjene ponekad teško učiniti bez velikog utjecaja na grafički prikaz ostatka programa. Ti su problemi prirođeni grafičkom prikazu, ali već danas postoje mnoga rješenja koja uključuju koncepte poput gniježđenja ili enkapsulacije dijelova programa. Spajanjem objektno-orijentiranog i vizualnog pristupa također je moguće smanjiti utjecaje ovih nedostataka na uporabljivost jezika [15]. Ovisno o primjeni, mogući su i problemi s testiranjem, formalnim verificiranjem i validacijom, te ispravljanjem (eng. *debugging*) programa.

1.2 Doprinosi

Doktorski rad sadrži tri glavna i nekoliko popratnih doprinosa.

Najbitniji od glavnih doprinosa je vizualni programski jezik za sintezu zvuka temeljenu na toku atributa i računalno stvaranje glazbe temeljeno na manipulacijama u vremenskoj domeni. Predloženi jezik jedinstven je po tome što, za razliku od ostalih sličnih rješenja u području, kao temeljnu podatkovnu jedinicu koristi timbralne atribute umjesto većinski korištenih audio signala. Budući da su timbralni atributi razumljiviji umjetnicima i glazbenicima, rad s jezikom postaje intuitivniji, učenje jezika postaje jednostavnije, a produktivnost u radu se povećava. Razvoj jezika tekao je u nekoliko koraka. U prvom je koraku definiran koncept temeljen na toku timbralnih atributa.

Kako bi validirali koncept i potvrdili valjanost pretpostavki, provedene su kvantitativne, kvalitativne i formalne metode evaluacije postojećih rješenja u području [16][17][18]. Osim što je tim istraživanjem argumentirano da ima smisla stvaranje novog jezika koji neće biti

temeljen na uobičajenim audio signalima, oblikovanje jezika potom se oslanjalo na saznanja iz tih analiza te su predložena rješenja formirana tako da uzmu u obzir ograničenja i nedostatke postojećih jezika. U tom smislu, dodatni doprinos rada postaju rezultati i podatci dobiveni kroz ankete i analizu anketnih rezultata budući da se ti rezultati mogu koristiti u budućim radovima, a neiskorištena saznanja za dodatno profiliranje i poboljšanja predloženog jezika.

Osim toga, kao dodatna pretpostavka za oblikovanje jezika bilo je potrebno ostvariti sustav mapiranja timbralnih atributa u audio signale i natrag [19][20]. Sami ti postupci značajan su doprinos i moguće ih je koristiti i van samoga jezika.

Konačno, evaluacijom prototipa pokazano je da pristup temeljen na toku atributa ima značajne posljedice po jednostavnost korištenja jezika i inherentnu razumljivost.

Drugi doprinos rada jesu elementi grafičkog sučelja i model vizualizacije timbralnih atributa na temelju generiranog i sintetiziranog zvuka te načini unosa timbralnih atributa u sam jezik i upravljanje parametrima sintetizatora putem njih, kao i neki drugi oblici upravljanja sintetizatorima [19][20][21]. Predložene su modifikacije dobro poznatih metoda vizualizacije audio signala (spektralne karakteristike, VU metar) kako bi prikazivale udio pojedinog timbralnog atributa u analiziranom signalu. To rješenje oslanja se na slične arhitekturalne postavke kao i sam unos timbralnih atributa kao parametara sintetizatora, odnosno generativnih elemenata u vizualnom programskom jeziku.

Treći i posljednji doprinos rada nastao je kao poboljšanje vizualnog programskog jezika predloženog u glavnom doprinosu. Problem uspoređivanja vizualnih programa složen je i, zbog tehničkih ograničenja, teško rješiv u općem slučaju. Rješenje predloženo u ovom radu zato u obzir uzima samo usporedbe programa u novom vizualnom programskom jeziku.

1.3 Opseg istraživanja

Istraživanje provedeno u ovom radu koncentrirano je na unaprjeđenja vizualnih programskih jezika u području glazbe. Iako su neka rješenja primjenjiva i u drugim domenama, njihove prilagodbe nisu razmatrane.

U samom kontekstu vizualnog programiranja za glazbu, predloženi programski jezik donosi nekoliko inovacija isključivo vezanih uz temeljnu paradigmu toka signala umjesto kojih se predlaže tok timbralnih atributa. Također, predlažu se rješenja interaktivnog unosa timbralnih atributa, vizualizacije generiranih zvukova pomoću vrijednosti timbralnih atributa te uspoređivanja vizualnih programa. Iako su u radu identificirana i neka druga područja u kojima je moguće poboljšati postojeća rješenja (primjerice, vremenske crte), ona nisu

detaljnije razmatrana. Problemi koje je predloženi programski jezik naslijedio od postojećih rješenja te inherentna ograničenja vizualne paradigme razmotreni su u poglavljima 3.2 i 3.3.

Konačno, u radu su predloženi i predstavljeni postupci kojima se timbralni atributi mapiraju u audio signale i na taj način povezuju s postojećim okosnicama u jezicima poput Pure Date, no njihova detaljnija, zasebna primjena i proširenje značaja van primjene u vizualnom programiranju istraženi su u popratnim radovima [19][20].

1.4 Metodologija

Vizualni programski jezik predložen u okviru istraživanja oblikovan je prema konceptima proizašlima iz proučavanja trenutno postojećih rješenja. Intuitivnost ideja vezanih uz primjenu timbralnih atributa kao i temeljni elementi paradigme jezika validirani su prije samog oblikovanja jezika. Za tu je validaciju upotrijebljena višestruka evaluacija, prvo analizom postojećih jezika prema kognitivnim dimenzijama, a zatim anketiranjem njihovih korisnika. Svi elementi novog jezika osmišljeni su tako da ili zadrže dobre značajke postojećih jezika ili inoviraju i poboljšaju područja u kojima postoje nedostatci i propusti.

Nakon što je jezik osmišljen i teorijski oblikovan, a njegove temeljne ideje validirane, pripremljen je prototip. Za izradu prototipa odabrana je platforma vizualnog programskog jezika Pure Data. Izbor Pure Date uvjetovan je paradigmatskom sličnošću s predloženim jezikom te zbog činjenice da je jezik otvorenoga kôda, slobodno proširiv te široke podrške u zajednici. Zbog tih se razloga pokazao optimalnim jezikom za jednostavnu, a postojećim korisnicima blisku implementaciju prototipa.

Konačno, prototip jezika evaluiran je na sličan način na koji su evaluirani i postojeći, drugi jezici. U prvoj je fazi prototip analiziran primjenom kognitivnih dimenzija te je prema tim rezultatima uspoređen s drugim sličnim jezicima. U drugoj fazi provođeni su intervjui s iskusnim korisnicima jezika poput Pure Date ili Max/MSP-a. Uz demonstraciju prototipa jezika, kroz intervjue su prikupljeni subjektivni dojmovi i ekstrapolirane objektivne metrike jezika. Zatim, u trećoj fazi, pripremljena je anketa koja je kombinirala pitanja kvalitativnog i kvantitativnog tipa. Diseminacija ankete bila je kontrolirana i ograničena samo na iskusne korisnike pojedinih jezika. Na temelju rezultata analize ankete, podataka prikupljenih iz intervjua i kroz prizmu kognitivnih dimenzija, argumentirana su i validirana poboljšanja i inovacije predloženog jezika. Dodatno, postupak mapiranja timbralnih atributa u parametre jezika validiran je u zasebnom istraživanju i radovima [19][20][22].

Razvoj drugih predloženih rješenja u ovom radu temeljio se na nadogradnji postojećih rješenja i radova uz primjenu inovacija na specifičnu domenu te integraciju s predloženim

jezikom. Zbog toga je njihova evaluacija provedena u sklopu ispitivanja samoga jezika bez dodatnih posebnih analiza. Kad je, primjerice, riječ o vizualizaciji sinteze zvuka pomoću vrijednosti timbralnih atributa, nisu provedeni posebni postupci evaluacije, već je doprinos funkcionalno uspoređen s postojećim sličnim grafičkim elementima.

Grafičko upravljanje sintetizatorom pomoću timbralnih atributa validirano je pomoću fokus grupe – ispitivanjem sustava od strane skupine studenata muzičke akademije u Zagrebu – na način da uneseni timbralni atributi odgovaraju generiranom zvuku [19][20], no sami elementi grafičkog sučelja nisu posebno evaluirani. Konačno, evaluacija metode uspoređivanja vizualnih jezika svodi se na zadovoljavanje funkcionalne specifikacije uz uvažavanje ograničenja rješenja.

Metodologija za pojedine doprinose detaljno je opisana u odgovarajućim poglavljima doktorskog rada.

1.5 Struktura rada

U drugom su poglavlju opisana postojeća i relevantna istraživanja u području vizualnih programskih jezika u glazbi. Nakon što su definirane temeljne postavke jezika u domeni glazbe i višemedijskih pristupa, dan je pregled različitih značajnih jezika u području poput Pure Date, Max/MSP-a i OpenMusica. S obzirom na to da se jezik opisan u ovom radu oslanja na timbralne atribute kao jedinicu toka podataka, objašnjene su osnove i temeljni radovi u domeni timbralnih atributa, glazbenih sintetizatora i sinteze zvuka općenito (uključujući metode i kontrolu sinteze zvuka). Pritom je poseban naglasak stavljen na upravljanje sintezom zvuka pomoću timbralnih atributa.

U nastavku poglavlja dan je kratki pregled različitih rješenja za vizualizaciju rezultata sinteze zvuka i za uspoređivanje programa, kako u vizualnoj tako i u tekstnoj domeni.

Treće poglavlje posvećeno je detaljnoj razradi glavnog doprinosa rada - vizualnog jezika za sintezu zvuka temeljenu na toku timbralnih atributa i računalno stvaranje glazbe temeljeno na vremenskim manipulacijama. U uvodnom dijelu poglavlja definirani su temeljni koncepti i motivacija iza samoga jezika. Slijedi potpoglavlje o evaluaciji postojećih rješenja i oblikovanju programskih jezika. U tom su poglavlju prezentirani popratni doprinosi koji se odnose na podatke prikupljene anketiranjem i formalnom analizom jezika Pure Data, Max/MSP, OpenMusic, Kyma i Reaktor. Prezentirane su, primjerice, analize poželjnih značajki programskih jezika te utjecaj formalnog obrazovanja na rad s vizualnim programskim jezicima u glazbi. Na temelju tih analiza argumentira se i objašnjava proces

oblikovanja novog programskog jezika koji uvažava dobre prakse prisutne u tim jezicima te predlaže rješenja nekih njihovih problema.

Nakon što su pojašnjene osnovne pretpostavke predloženog jezika, detaljnije se razrađuju postupci koji omogućuju korištenje timbralnih atributa kao granule podataka u toku podataka u novom jeziku. Definiran je odnos timbralnih atributa i audio signala te su objašnjeni načini odabira parametara sintetizatora, preslikavanja atributa i upravljanja sintezom u novom jeziku.

Glavnina poglavlja posvećena je objašnjenju arhitekture i elemenata jezika te predstavljanju prototipa uz isticanje pojedinih bitnih detalja implementacije. Sam prikaz jezika koristi se opisnim pristupom budući da su sintaksa i prezentacija kod vizualnih programskih jezika usko povezani.

Konačno, završni dio trećeg poglavlja predstavlja rezultate evaluacije novog jezika uz korištenje kognitivnih dimenzija te kvalitativne i kvantitativne analize provedene na temelju rezultata anketa. Ukratko je jezik uspoređen i s prethodnicima poput Pure Date. Na samom kraju poglavlja, ponuđena je diskusija predloženog rješenja uz isticanje prepoznatih nedostataka i mogućih budućih istraživanja te su izvedeni odgovarajući zaključci.

Četvrto i peto poglavlje razmatraju druga dva doprinosa istraživanja. U četvrtom se poglavlju obrazlaže rješenje za probleme vizualiziranja rezultata sinteze zvuka, a na temelju vrijednosti timbralnih atributa te rješenje za intuitivno grafičko upravljanje sintetizatorom pomoću timbralnih atributa. Tu su predstavljeni elementi sučelja, mehanizmi preslikavanja timbralnih atributa, prototip i razmotrena alternativna rješenja. U diskusiji se problematizira predloženo rješenje i razmatraju moguća buduća poboljšanja.

Posljednji doprinos, uspoređivanje vizualnih programa za sintezu zvuka i stvaranje glazbe, opisan je u petom poglavlju. Nakon što su definirani temeljni problemi uspoređivanja vizualnih programa, predlaže se rješenje temeljeno na vizualnom označavanju izmjena te njihovo interaktivno predstavljanje u tekstnoj domeni. Predložena su rješenja ograničena na integraciju s novim, u ovom radu predstavljenim vizualnim programskim jezikom. Na samom kraju poglavlja, razmotreni su prijedlozi alternativnih rješenja te su ponuđeni diskusija i osvrt.

Sam rad zaključen je dvama poglavljima, Diskusija i Zaključak, koji sažimaju i zrcale najbitnije dijelove istraživanja te razmatraju njegov značaj za daljnji razvoj područja.

2 Dosadašnja povezana istraživanja

Područje vizualnih programskih jezika i konkretno njihove upotrebe u glazbi aktivno je i aktualno te u fokusu mnogih suvremenih istraživanja [23]. Porastom računalne moći osobnih i prijenosnih računala, takvi su jezici i alati postali bitnim dijelom umjetničke vizije suvremenih glazbenika [24]. Nadilazeći svoje prvotne namjene, ti alati svojim svojstvima oblikuju kreativnost korisnika, dok istovremeno napredni korisnici pronalaze nove načine za iskoristiti njihove mogućnosti. Budući da je glavni doprinos rada vizualni programski jezik koji nastoji maksimirati i optimirati te dimenzije, u nastavku poglavlja dan je uvid u trenutno stanje u području, pojašnjeni su tehničko-kognitivni razlozi njihove popularnosti te su ukratko opisani najznačajniji vizualni programski jezici u domeni glazbe.

Drugi dio poglavlja posvećen je kratkom pregledu područja kojem pripadaju drugi doprinosi proizašli iz ovog istraživanja. Iako raznorodna, ta su područja istraživanja usko povezana s glazbenom tehnologijom i vizualnim jezicima u glazbi.

2.1 Programski jezici i višemedijski pristupi

Kad je riječ o računalnoj glazbi, multimediji i interaktivnoj umjetnosti, vizualni programski jezici pokazuju se kao tehnologija koja je najpristupačnija glazbenicima, skladateljima i drugim korisnicima sličnog profila. Osim razloga vezanih uz ljudsku psihologiju opisanih u uvodu, a koji se najčešće navode u istraživanjima [3], uočljivi su još neki pokazatelji koji objašnjavaju tu poveznicu.

U ovoj je domeni digitalna obrada audio signala, slika i videa temeljni dio svakog alata, jezika i aplikacije. Većina suvremenih vizualnih programskih jezika i okruženja zato nude gotove programske elemente koji imaju funkcionalnosti i algoritme digitalne obrade signala te olakšavaju njihovu integraciju s periferijskim uređajima poput MIDI klavijatura. U kontekstu glazbe i obrade zvuka, takvi programski elementi najčešće emuliraju oscilatore, filtere, audio efekte, algoritme za praćenje notnog zapisa, *auto-tuner* itd.

Istovremeno, takvi jezici sakrivaju dio kompleksnosti tih funkcionalnosti i algoritama od krajnjih korisnika. Korištenjem gotovih elemenata, "crnih kutija", glazbenici i umjetnici ne moraju se brinuti o obradi signala na niskoj razini. Zahvaljujući toj podređenosti jezika praktičnoj upotrebi u specifičnoj domeni, vizualno programiranje široko je prihvaćeno među glazbenicima i multimedijskim umjetnicima. Ipak, kao što će se pokazati u nastavku rada,

optimalno korištenje trenutnih rješenja i dalje zahtijeva određeno poznavanje signala i razumijevanje načina njihove obrade te ograničava kreativnost korisnika.

2.1.1 Jezici u domeni glazbe i sinteze zvuka

U postojećim vizualnim programskim okruženjima za obradu glazbe i signala, audio signali najčešće sudjeluju u toku podataka. Pure Data, Max/MSP, Kyma, AudioMulch i Reaktor među najpopularnijim su okruženjima koja se oslanjaju na ovu paradigmu. Iako vizualno programiranje čiji su temelji u toku signala jamči maksimalnu fleksibilnost, ono istovremeno prisiljava korisnike da o glazbi i zvukovnoj umjetnosti razmišljaju kroz prizmu obrade signala. Glazbenici moraju razumijeti kako pojedini programski elementi utječu na audio signal te kako se vrijednosti parametara (primjerice sintetizatora) moraju postaviti i mijenjati kako bi postigli željeni izlaz, odnosno sintetizirani zvuk.

U nastavku će ukratko biti prezentirani značajni predstavnici vizualnih programskih jezika u glazbi. Oni se ističu po svojim mogućnostima, inovacijama i prihvaćenosti u zajednici.

Slika 2.1 Pure Data, Max/MSP i OpenMusic

Pure Data

Pure Data [25] je vizualni programski jezik otvorenoga kôda čija je glavna primjena obrada i generiranje zvuka te stvaranje glazbe, a općenito se koristi i za umjetničke multimedijske radove. Pure Data kao programski jezik izgrađen je oko paradigme programskog toka

podataka iako sadrži i elemente drugih paradigmi. Sam tok podataka sastoji se od kontrolnih poruka i audio signala koji se obrađuju u ugrađenim objektima. Ti objekti nude niz funkcionalnosti, od jednostavnijih poput matematičkih operacija (sumiranje) i uobičajenih programskih mogućnosti (petlje, kontrola toka) do složenih postupaka digitalne obrade signala. Jezik podržava široki skup funkcionalnosti koje su dostupne ili kroz spomenute ugrađene objekte ili kroz vanjske ekstenzije koje se nazivaju externals. Ta značajka Pure Datu čini prilagodljivim jezikom pogodnim ne samo za stvaranje zvuka, videa i drugih multimedijskih sadržaja već i za stvaranje punih programa opće namjene.

Pure Data se također ističe raznolikošću dostupnih gotovih potprograma (patches) i mogućnošću povezivanja s različitim hardverom i vanjskim softverom poput drugih alata za obradu zvuka, programskih biblioteka i sustava. Budući da se interaktivni sustavi često sastoje upravo od takvih raznorodnih komponenti, Pure Data može poslužiti kao centralno sučelje i vezivno tkivo prilikom njihova stvaranja.

Međutim, ta fleksibilnost jezika uz spomenute prednosti donosi i nedostatke. S jedne strane, korisnicima omogućuje ostvarivanje širokog raspona različitih ideja, no s druge strane traži znanje i razumijevanje programiranja na nižoj razini. To jezik čini relativno nepogodnim za početnike i stručnjake iz pojedinih domena koji nisu ujedno i programeri u užem smislu. Ti nedostatci posebno dolaze do izražaja kad je u pitanju centralni tok signala u programima.

Od ostalih nedostataka, valja spomenuti zastarjelost samog grafičkog sučelja iz perspektive dizajna i kvalitete korisničkog iskustva te postojanje samo rudimentarne podrške za funkcionalnosti više, apstraktne razine. Potonje je potrebno ručno implementirati u programima.

Max/MSP

Max/MSP [26] još je jedan jezik autora Pure Date, Millera Puckettea, čija je povijest usko vezana s Pure Datom. Max/MSP je istovremeno i preteča i nasljednik Pure Date te su suvremene inačice ta dva jezika vizualnim stilom i temeljnim konceptima vrlo slične. Oba jezika pripadaju skupini tzv. patcher jezika [27], dijele slične obrasce korištenja, namjene i značajke te su čak dijelovi njihova kôda međusobno spojivi i interoperabilni. Drugim riječima, patchevi iz Pure Date mogu se koristiti u Max/MSP-u i obratno.

Bitna prednost Max/MSP-a jest korisničko grafičko sučelje koje je doživjelo mnoge preinake i poboljšanja u smislu korisničkog iskustva, bolja organizacija programskih komponenti te dostupnost većeg broja popratnih alata poput oznaka i tražilice. Iako su ta poboljšanja periferna i ne utječu na samu funkcionalnost jezika, ona znatno pospješuju

korištenje jezika i učinkovitost pri programiranju. Grafički i vizualni dizajn s mogućnošću prilagodbe pojedinih elemenata potrebama korisnika doprinose boljoj čitljivosti i dojmu prilikom korištenja radnoga prostora. Dodatno, simboli nekih objekata na radnoj površini sadrže bogate vizualne reprezentacije koje na jednostavan način ilustriraju njihovu svrhu te čak nude izravno utjecanje na njihove parametre. Primjer takvog objekta je objekt za mijenjanje amplitude signala koji na samom simbolu sadrži tipke za povećanje i smanjenje faktora amplitude.

Budući da je u suvremenoj inačici Max/MSP plaćeni program zatvorenoga kôda [28], korisnička baza znatno je manja od Pure Datine, no istovremeno obuhvaća značajan broj profesionalnih glazbenika koji se oslanjaju na njegova poboljšanja i prednosti u odnosu na Pure Datu.

OpenMusic

U svojoj suštini, OpenMusic [29] je objektno-orijentirani vizualni programski jezik razvijen na institutu IRCAM te specijaliziran, kao što mu i ime govori, isključivo za stvaranje glazbe i sintezu zvuka. Iako nudi i niz funkcionalnosti pomoću kojih je moguće stvarati programe opće svrhe, njegovi su elementi, objekti i način programiranja podređeni algoritamskoj kompoziciji i automatizaciji skladateljskih tehnika. Najvažnije njegove značajke uključuju objekte koji emuliraju, i vizualno i funkcionalno, uređivače standardnog notnog zapisa kao i niz objekata koji se odmiču od nižih slojeva temeljenih na toku audio signala te omogućuju manipuliranje zvukovima u vremenu. Budući da je programski kôd OpenMusica temeljen na Lispu, jezik zadržava mnoge karakteristike tog tekstnog jezika te je u njemu moguće pisati proširenja i dodatke.

Unatoč činjenici da nije fleksibilan ni jednostavno proširiv kao Pure Data ili Max/MSP, OpenMusic nudi mnoge dodatne mogućnosti koje ga čine praktičnijim za korištenje u primjenama koje su usko vezane uz skladanje glazbe. Za korisnike upoznate s Lispom, OpenMusic nudi jednostavan pristup jezgrenim funkcijama te otvara mogućnost njihove modifikacije.

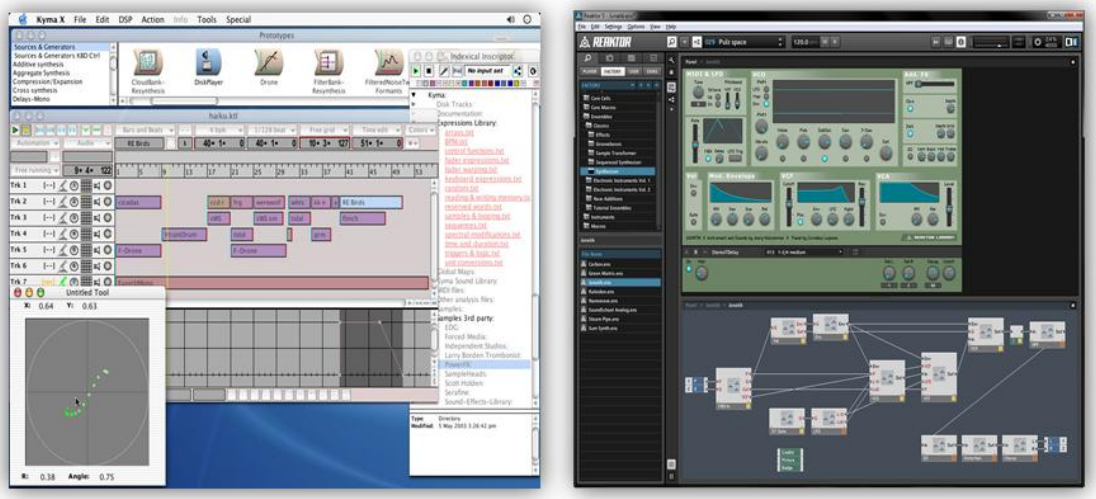
Symbolic Sound Kyma

Symbolic Sound Kyma, grafički programski jezik za "dizajn zvuka", posebno je osmišljen za računalom potpomognuto, interaktivno stvaranje zvuka u "živim", stvarnovremenskim okruženjima te postprodukciju [30]. Budući da je većina njegovih značajki i pruženih funkcionalnost koncentrirana na manipulaciju zvukovima dok su ostale dimenzije

multimedijskog sadržaja zanemarene, njegove su primjene ograničene na skladanje glazbe i niz povezanih tehnika oblikovanja glazbenog i zvučnog sadržaja.

Kymina paradigma uključuje mnoge različite aspekte koji su prethodno opisani kod Pure Date, Max/MSP-a i OpenMusica te dodatne alate (poput *timelinea*, odnosno vremenske crte) koji nisu dostupni u drugim rješenjima. Zbog tih je svojih računalnoj glazbi orijentiranih alata Kyma jednostavnija za korištenje, a proces programiranja postaje fluidnijim te otpornijim na pogreške. Posebno je tu važan tzv. timeline, odnosno vremenska crta koja omogućuje interaktivnu manipulaciju zvukova u vremenskoj domeni što je funkcionalnost slična onoj koju nude digitalne radne stanice za obradu zvuka.

Iako su doprinosi tvorkinje Kyme, Carle Scaletti, iznimno značajni za razvoj vizualnog programiranja u polju glazbe, te je Kyma proizašla iz akademskog okruženja, sam je jezik od početka bio zatvorenog kôda te iznimno skup zbog činjenice da mu je za rad potreban vanjski hardver. Kao što će rezultati našeg istraživanja pokazati u nastavku, zbog ovih je čimbenika doseg Kyme relativno uzak te ograničen na akademske ustanove i profesionalne glazbene studije.



Slika 2.2 Kyma i Reaktor

Native Instruments Reaktor

Native Instruments Reaktor [31] je modularni studio, odnosno grafičko razvojno okruženje i jezik za dizajn sintetizatora zvuka, sekvencera, samplera, audio efekata i drugih alata za oblikovanje zvuka. Za razliku od svih prethodno predstavljenih alata i programskih jezika, Reaktor nije zamišljen kao jezik za stvaranje glazbenih skladbi, interaktivnih sustava ili multimedijskih radova. Njegova je primarna namjena razvoj objekata za sintezu i obradu zvuka. Jednom stvoreni, ti se objekti mogu kombinirati, spajati i nadovezivati.

Osim objekata koji se stvaraju ili su ugrađeni i temeljeni na vizualnoj paradigmi na nekoliko različitih razina apstrakcije, u Reaktoru je moguće pregledavati i uređivati njegove interne arhitekturalne elemente korištenjem tekstnog programiranja. Iako je takva funkcionalnost usmjerena na napredne korisnike, tim se pristupom povećava fleksibilnost samog jezika van temeljne paradigme. Jednom kad je željeni "program" dovršen, sintetizatori zvuka i efekti razvijeni u Reaktoru mogu se koristiti u vanjskim sekvencerima i digitalnim radnim stanicama za obradu zvuka. Ovako oblikovani skup alata ima drukčiju ciljnu skupinu korisnika u usporedbi s drugim alatima opisanima u ovom poglavlju što utječe i na očekivanja i primjene samog jezika.

2.2 Timbralni atributi i upravljanje sintetizatorima

Budući da je područje glazbenih sintetizatora općenito van opsega ovog rada, na ovom mjestu neće biti detaljno opisani različiti njihovi oblici i načini kontrole. U nastavku su ugrubo opisane osnove glazbenih sintetizatora, dok je više pažnje posvećeno timbralnim atributima i upravljanju sintezom pomoću njih budući da se ti postupci nalaze u temeljima ovog doktorskog istraživanja i novo oblikovanog vizualnog programskog jezika.

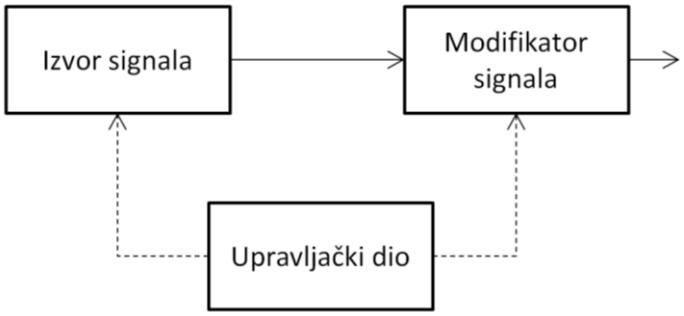
2.2.1 Sinteza zvuka

Sinteza zvuka najčešće se definira kao postupak generiranja audio signala [32]. Sinteza zvuka pritom uključuje primjenu matematičkog procesa u izračunu vrijednosti signala ili modifikaciju prethodno pripremljenih zapisa. Takav pristup kod kojeg se audio signal iz temelja generira elektroničkim sklopovima ili softverom čini bitnu razliku u odnosu na puku reprodukciju ili obradu ulaznih signala ili zapisa. Načini izračuna izlaznog signala različitih su kompleksnosti, od vrlo jednostavnih poput zbrajanja sinusnih signala do vrlo složenih i temeljenih na naprednim matematičkim tehnikama.

Iako su se prije početka razvoja računalne tehnologije glazbeni sintetizatori uglavnom sastojali od analognih elektroničkih komponenti, danas su dominantni programski sintetizatori. Ti se sintetizatori izvršavaju ili na osobnim računalima, odnosno računalima opće namjene, ili na specijaliziranom sklopovlju za digitalnu obradu signala. Računalni sintetizatori koji nisu vezani uz posebno sklopovlje najčešće se javljaju kao dodatci postojećoj programskoj potpori te ih je lako uključiti u različite druge sustave za glazbenu produkciju ili izvedbe uživo [24].

2.2.2 Glazbeni sintetizatori općenito

Unatoč značajnim razlikama među pojedinim metodama sinteze zvuka, temeljni principi rada glazbenog sintetizatora mogu se opisati modelom koji vrijedi neovisno o metodi sinteze. Model se temelji na međusobno ovisnom odnosu tri logička dijela sintetizatora. Ti su dijelovi izvor signala, modifikator signala i upravljački dio [33]. Iako nije primjenjivo općenito, za neke se metode sinteze navedeni logički dijelovi mogu izravno preslikati u module glazbenog sintetizatora. **Slika 2.3** prikazuje tok signala među pojedinim logičkim dijelovima.



Slika 2.3 Tok signala kod sintetizatora zvuka

Izvor signala generira valne oblike računanjem u stvarnom vremenu ili kombiniranjem postojećih, pripremljenih zapisa. Važnost izvora signala u procesu sinteze nalazi se u činjenici da bez odgovarajućih značajki izvornog signala, odnosno prisutnosti potrebnih harmonika [34], često nije moguće postići određene timbralne karakteristike naknadnom obradom. Generirani se signal dalje obrađuje, a proces obrade može uključivati korake kao što su filtriranje i modulacija amplitude, ali i primjenu audio efekata poput jeke, distorzije i reverberacije.

Uloga upravljačkog dijela sintetizatora jest generiranje kontrolnih signala koji upravljaju radom izvora i modifikatora audio signala. Uobičajeni primjer upravljačkog elementa jest niskofrekvencijski oscilator. On se u subtraktivnoj sintezi koristi za modulaciju osnovne frekvencije audio oscilatora radi postizanja vibrato-efekta. Često se koristi i za modulaciju izlazne razine pojačala radi postizanja tremolo efekta. Niskofrekvencijski oscilator također može upravljati filtrom kako bi se boja tona mijenjala periodički. Upravljački dio, poput ostalih elemenata glazbenog sintetizatora, najčešće ima parametre koji korisnicima omogućuju kontrolu. Kod niskofrekvencijskog oscilatora, primjerice, korisnik može birati valni oblik, frekvenciju i amplitudu kontrolnog signala.

2.2.3 Metode i kontrola sinteze zvuka

Metode sinteze

Glazbeni sintetizatori koriste različite metode sinteze zvuka. Neke od njih prilagođene su specifičnim namjenama, dok su preostale osmišljene kako bi udovoljile širem skupu potreba glazbenika. Sistematični pregled metoda sinteze može se pronaći u [2][32][35]. U nastavku poglavlja spomenuto je samo nekoliko najznačajnijih metoda.

Suptraktivna sinteza temelji se na uklanjanju neželjenih spektralnih komponenti iz signala u kojem su prisutni harmonici dobiveni kombinacijom osnovnih valnih oblika poput pravokutnog, pilastog i trokutastog [36][37]. Zbog relativno jednostavne implementacije elektroničkim sklopovima, suptraktivna sinteza bila je najzastupljenija u doba analognih sintetizatora.

Nasuprot suptraktivnoj sintezi, aditivna sinteza spektralnu strukturu audio signala postiže zbrajanjem većeg broja jednostavnih valnih oblika [38]. Kao osnovni valni oblik najčešće se koristi sinus, a frekvencije oscilatora postavljaju se u harmonijski odnos. S obzirom na to da je za sintezu složenih zvukova potreban veliki broj komponenti, aditivna sinteza računski je zahtjevna.

Frekvencijska modulacija (FM) u osnovnom se obliku ostvaruje pomoću dva audio oscilatora od kojih jedan modulira osnovnu frekvenciju drugog [34]. Kako je frekvencija modulacijskog signala iz raspona audio frekvencija, efekt modulacije ne percipira se kao vibrato, već kao zvuk bogatijeg spektra. Kod frekvencijske se modulacije s relativno malim brojem oscilatora može postići velika raznolikost sintetiziranih zvukova.

Sinteza temeljena na snimljenim zapisima ili primjercima (eng. *sample*) signala umjesto jednostavnih valnih oblika kao izvor zvuka koristi snimljene audio zapise [39]. Reprodukcija signala s više različitih osnovnih frekvencija postiže se interpolacijama, a potrebna fleksibilnost i izražajnost dobivaju se kombiniranjem većeg broja oscilatora i naprednih mogućnosti modifikacije signala. Ova metoda sinteze omogućava jednostavno oponašanje akustičnih instrumenata, pa se danas često koristi u komercijalnim glazbenim sintetizatorima.

Fizikalno modeliranje u svojoj osnovi sadrži matematičke modele stvarnih glazbenih instrumenata ili drugih fizikalno mogućih izvora zvuka [39]. Jednadžbama se opisuju pojedini dijelovi instrumenta i načini pobude poput trzanja, udaranja ili sviranja gudalom. Kako bi postupak sinteze bio računski učinkovitiji, često se koriste aproksimacije pravih fizikalnih modela.

Od ostalih metoda sinteze zvuka najčešće se javljaju formantna sinteza [40], oblikovanje valnog oblika (eng. *waveshaping*) [41] i granularna sinteza [37].

Navedene metode sinteze zvuka obilježavaju specifični timbralni karakteri sintetiziranih zvukova. Ipak, postoje i zvukovi koji se mogu postići pomoću više od jedne metode sinteze.

Kontrola sinteze

Kontrola glazbenog sintetizatora odvija se putem parametara koji najčešće imaju numeričke vrijednosti. Glazbenici mogu te vrijednosti podešavati putem klizača, kotačića i tipki izvedenih sklopovski ili simbolima koji su dijelom grafičkog korisničkog sučelja.

Problemi manipuliranja numeričkim parametrima slični su problemima vezanima uz razumijevanje audio signala od strane glazbenika i sličnih korisnika. S obzirom da parametri numerički sudjeluju u procesu generiranja i modifikacije audio signala, oni najčešće ne nose akustičko značenje. Odnosno, postoji odmak u načinu na koji glazbenici razmišljaju o zvuku i načinu na koji bi taj zvuk trebali ostvariti pomoću sintetizatora. Očiti je primjer sinteza frekvencijskom modulacijom kod koje parametri ne koreliraju na intuitivan način s timbralnim značajkama sintetiziranog zvuka. Iako FM sintetizatori imaju relativno mali broj parametara, ta činjenica čini upravljanje frekvencijskom modulacijom složenim postupkom. Drugi se primjer javlja kod aditivne sinteze. U toj se metodi sinteze definira veliki broj parametara kako bi se postigao željeni zvuk. Pokušaji grupiranja parametara u svrhu olakšavanja postupka dovode do gubitka fleksibilnosti. Korisnici takvih sintetizatora zvuka odlučuju se na uporabu unaprijed pripremljenih zvukova, izbjegavajući pritom jalovo eksperimentiranje s parametrima sinteze.

Parametri sinteze mogu imati međuzavisan utjecaj na pojedine karakteristike sintetiziranog zvuka te ih nije moguće prilagođavati pojedinačno. Ako je, primjerice, cilj postići vibrato korištenjem suptraktivne sinteze, potrebno je spojiti niskofrekvencijski oscilator tako da modulira osnovnu frekvenciju audio oscilatora te odabrati njegovu amplitudu, valni oblik i frekvenciju. Kod drugih metoda sinteze također postoje brojni primjeri međuzavisnosti parametara. Stoga je za efikasnu upotrebu glazbenog sintetizatora nužno poznavati korištenu metodu sinteze te specifičnosti implementacije sintetizatora.

Uzevši u obzir činjenicu da je među suvremenim glazbenim sintetizatorima prisutna velika raznolikost u metodama sinteze i konceptima korisničkih sučelja, poznavanje jednog sintetizatora ne jamči snalaženje s drugima. Kako bi ostvarili svoje umjetničke ideje, glazbenici često moraju izdvojiti vrijeme za proučavanje, isprobavanje i stjecanje praktičnog znanja o pojedinim sintetizatorima. Kako bi se riješio taj problem, predloženi vizualni

programski jezik koristi intuitivne postupke sinteze zvuka koje su razvijene zasebno te integrirane u model jezika.

2.2.4 Timbralni atributi

Glazbenici koriste timbralne atribute u svakodnevnoj komunikaciji kako bi opisali zvuk koji su čuli ili zamislili. Uobičajeno je reći da nešto treba odsvirati mekše ili da je određeni zvuk previše rezak. Iako su timbralni atributi subjektivni i ovisni o kontekstu, glazbenicima na intuitivan način omogućuju razmjenu i sistematizaciju ideja o timbralnim karakteristikama zvuka.

Timbralni aspekt zvuka u glazbi osobito je važan bez obzira na glazbeni stil. Promišljanje o timbralnim karakteristikama nužno je prilikom orkestracije, pripreme izražajne glazbene izvedbe, eksperimentiranja s novim zvukovima, ugađanja glazbenih instrumenata te svakom drugom glazbenom djelovanju. U suvremenim glazbenim pravcima poput aleatorike, konkretne glazbe i elektroničke glazbe, tradicionalni koncepti poput tonaliteta, harmonije i ritma ne nalaze se u prvom planu, pa su upravo timbralne karakteristike zvuka nositelji glazbenog sadržaja i estetike [42]. Kako su formalizmi za notaciju timbralnog aspekta relativno ograničeni, skladateljima često preostaje da upute glazbenicima i objašnjenja svojih ideja iznesu verbalnim opisima. U tim prilikama također uobičajeno koriste timbralne atribute.

Zbog poviše opisane intuitivnosti i zastupljenosti u glazbenoj praksi, timbralni se atributi čine pogodnima za upravljanje glazbenim sintetizatorima. Dodatno, to ih čini pogodnima za primjenu kao temeljnog nositelja informacija u vizualnim jezicima ciljanima za stvaranje glazbe i sintezu zvuka. Broj postojećih istraživanja koja su se bavila ovom temom – ostvarivanjem programskog sustava koji bi na temelju ulaznih atributa odredio odgovarajuće vrijednosti parametara sinteze tako da rezultantni zvuk ima željene karakteristike zadane riječima iz prirodnog jezika – je ograničen. Izazov za ostvarenje takvog postupka predstavlja velika složenost preslikavanja između timbralnih atributa i parametara sintetizatora. Preslikavanje je često nemoguće eksplicitno opisati s obzirom na to da nije linearno ni injektivno. Zvukovi istih ili sličnih timbralnih karakteristika mogu se postići različitim kombinacijama vrijednosti parametara sinteze. Timbralni atributi također ne mogu jednoznačno kvantificirati točku u prostoru zvukova, nego samo neizrazito i aproksimativno opisati neke od dimenzija. Timbralni karakter zvuka nije formaliziran poput tonske visine ili ritma u glazbi što predstavlja još jedan izazov u smislu interpretacije značenja pojedinih atributa.

Riječi koje služe za opisivanje zvukova mogu se podijeliti u tri skupine [43]. Prvu skupinu čini onomatopeja, odnosno riječi koje izravno oponašaju zvuk (npr. wah-wah). U sljedeću skupinu ubrajaju se taksonomski klasifikatori koji se odabiru iz postojeće taksonomije poput skupine glazbenih instrumenata (npr. trzalački) ili naziva pojedinih instrumenata (npr. saksofon). Posljednja, treća skupina sadrži opise doživljaja zvuka, tj. subjektivnu percepciju zvuka (npr. reski zvuk).

Timbar zvuka multidimenzionalna je mjera koja se ne može u potpunosti opisati atributima. U mnogim prethodnim istraživanjima autori su nastojali pronaći semantički prostor razapet timbralnim atributima koji bi omogućio kvantifikaciju usporedbe zvukova [44][45]. Za takvu je kvantifikaciju važno moći izraziti zastupljenost pojedine karakteristike u zvuku. Zbog tog se razloga uz atribut pridjeljuje i numerička vrijednost. Uobičajeni pristup je korištenje skale kojoj je jedan ekstrem promatrani timbralni atribut, a drugi ekstrem atribut suprotnog značenja. Taj se pristup naziva semantičkim diferencijalom.

Problem semantičkog diferencijala jest nejednoznačnost atributa suprotnog značenja. Primjerice, za timbralni atribut „svijetao“ postoji više atributa koji opisuju zvuk suprotne karakteristike kao što su „taman“, „zagušen“ i „tup“. Kako bi se otklonila nejednoznačnost, u radu [45] predloženo je korištenje negacija poput „svijetao“ i „ne-svijetao“. Takav način vrednovanja zastupljenosti pojedinog atributa u cjelokupnom zvuku naziva se procjena magnitude verbalnog atributa (eng. *Verbal Attribute Magnitude Estimation/VAME*). Procjena temeljena na suprotstavljanju atributa vlastitoj negaciji osigurava razumijevanje magnitude za svaki atribut. U kontekstu kontrole sinteze zvuka ulaznim atributima, procjena magnitude može se koristiti kao kvantifikator uz pojedini atribut koji čini sastavni dio ciljnog opisa zvuka.

Odabir parametara probabilističkim modelom

U radu [46] Ross Clement problemu odabira parametara pristupa analizom ključnih riječi koje se pojavljuju u nazivima ulaznih datoteka računalnog glazbenog sintetizatora. Ideja se temelji na pretpostavci da nazivi programa nose informaciju o karakteristici zvuka. Istraživanja i praktični primjeri pokazuju da je to čest slučaj, pogotovo kada je riječ o imitativnoj sintezi. Pomoću heurističke formule prvo se iz skupa naziva programa odabire deset najčešćih znakovnih podnizova koji predstavljaju ključne riječi. Za pojedinu ključnu riječ razmatraju se svi programi u čijem se nazivu ona nalazi. U tim se programima za svaki parametar prebrajaju njegove diskretne vrijednosti čime se dobivaju učestalosti, odnosno vjerojatnosti pojavljivanja pojedinih vrijednosti. Tako je za svaku ključnu riječ poznato koliko je vjerojatna pojedina

vrijednost svakog parametra sinteze. Izračunate se vjerojatnosti koriste pri odabiru vrijednosti parametara kada je potrebno sintetizirati zvuk zadan ključnim riječima.

Opisano rješenje primjenjivo je samo na jednostavne sintetizatore kod kojih je tijek kontrolnih signala uvijek jedinstven, a međuzavisnost parametara minimalna. Razlog tome je što se vjerojatnosti pojavljivanja promatraju za svaki parametar nezavisno ne razmatrajući kombinacije parametara. Među postojećim programima korištenim za grupiranje mogu se pojaviti programi obilježeni istim ključnim riječima, ali s kontradiktornim vrijednostima pojedinog parametra, što smanjuje preciznost grupiranja, a time i cjelokupni rad sustava. Problem zavisnosti parametara i mogućih suprotnosti u podatcima za grupiranje nije obuhvaćen spomenutim radom.

Odabir parametara inverznom povratnom propagacijom pogreške

U radu [47] Gounaropoulos i Johnson predlažu princip temeljen na strojnom učenju. Njihov se sustav sastoji od dva dijela. Prvi dio je umjetna neuronska mreža koja za ulazni zvuk opisan audio značajkama može odrediti timbralne karakteristike zvuka. U konkretnom slučaju opis timbralne karakteristike sastoji se od devet pridjeva, a svakome od njih dodijeljena je vrijednost između 0 i 1. Na primjer, za zvuk klavira svjetlina tona iznosi 0.6, grubost 0.1, bujnost 0.6, udaraljkaška karakteristika 1.0 itd. Neuronska mreža ima devet izlaza od kojih svaki predstavlja vrijednost pridjeva. Kao ulazi koriste se audio značajke zadanog zvuka: prvih 15 sastavnica, prosječni frekvencijski odmak sastavnica od savršene harmoničke strukture i parametri amplitudne ovojnice. Za treniranje mreže autori su upotrijebili zvukove s ručno označenim vrijednostima svih pridjeva.

Zadaća drugog dijela sustava jest biranje parametara sintetizatora zvuka tako da se postigne zvuk čija karakteristika odgovara zadanim vrijednostima pridjeva. Inicijalna je ideja bila da se upotrijebi genetski algoritam pri čemu bi neuronska mreža poslužila za računanje funkcije dobrote tako da se za svaku jedinku promatra razlika između izlaza mreže i zadanih vrijednosti pridjeva. No, zbog nedovoljno glatke funkcije dobrote genetski algoritam ne konvergira i nije primjenjiv. Zato je upotrijebljena neuronska mreža za određivanje timbralnih karakteristika zadanog zvuka. U istraživanju je korišten aditivni sintetizator kod kojeg su parametri potpuno odgovarali značajkama analiziranog signala, pa je zato bilo moguće koristiti istu mrežu. Predloženo se rješenje temelji na algoritmu povratne propagacije pogreške.

U prvoj se iteraciji na ulaz mreže postavljaju slučajno odabrane vrijednosti parametara sinteze, određuje se izlaz mreže te pogreška u odnosu na zadane vrijednosti. Zatim se po

principu povratne propagacije pogreška prenosi natrag do ulaza u mrežu, ali se ne mijenjaju naučene težine u neuronima. Algoritam se ne koristi za učenje mreže, već za optimizaciju ulaznih parametara. Postupak se ponavlja dok se pogreška ne spusti ispod zadanog praga. Opisanim se algoritmom modificirane povratne propagacije postižu znatno bolji rezultati nego genetskim algoritmom. Autori su kao moguće smjerove istraživanja istaknuli uključivanje psihoakustičke obrade ulazog zvuka prije ulaza u neuronsku mrežu i rješavanje problema promjena timbralnih karakteristika tijekom trajanja istog zvuka.

Odabir parametara stablima odlučivanja

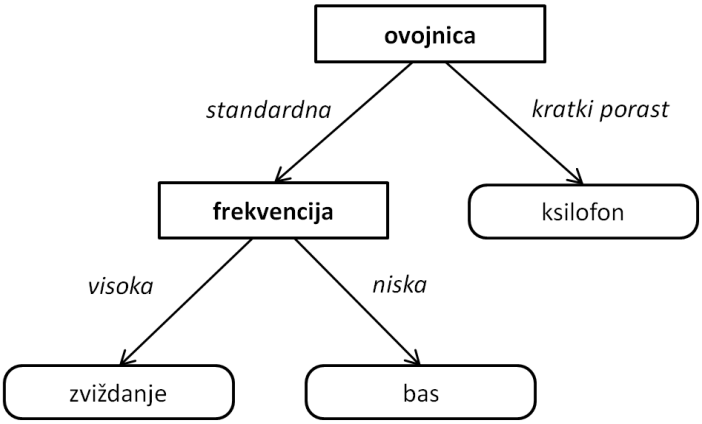
Kao preteču prethodno spomenutim radovima iz područja sinteze zvuka na temelju ulaznih pridjeva, važno je spomenuti rad [48]. Fokus rada stavljen je na shvaćanja zvuka blisko ljudima. Pretpostavljeno je da znanje o zvuku ima slojevitu strukturu i da ljudi uobičajeno koriste slojeve viših razina prilikom opisivanja zvuka. Primjer je zvuk grmljavine koji bi se mogao opisati kao glasan, oštar, šumovit, dubok i kratak. No, zbog skolonosti grupiranju informacija taj se zvuk grmljavine jednostavno opisuje kao „grmljavina“. Sljedeća je pretpostavka da se sličnost zvukova može izraziti u višedimenzionalnom prostoru atributa na način da slični zvukovi imaju iste vrijednosti većine atributa. Na poslijetku, rad promatra tezu da ljudi generaliziraju perceptualne atribute tako da zvukove karakteriziraju onim atributima koje doživljavaju važnijima.

Na temelju ovih pretpostavki, u radu se oblikuje sustav, nazvan „Artist“, za pomaganje glazbenicima pri sintezi željenih zvukova. Kako bi dobili željeni zvuk, glazbenici sustavu mogu zadati ime zvuka ili listu atributa i njihovih vrijednosti. Lista ne mora biti potpuna i može sadržavati neispravne vrijednosti atributa, no sustav će svejedno vratiti najprikladniji zvuk. U bazi znanja „Artista“ kao entiteti se pojavljuju zvukovi, atributi i parametri sinteze. Primjer skupa entiteta za definiranje zvuka:

- Zvukovi: zvuk A, zvuk B, zvuk C
- Atributi: otvorenost (široka, uska), prodornost (visoka, niska), osnovna frekvencija (duboka, srednja, visoka)
- Parametri sinteze: F1 (290 Hz, 400 Hz, 650 Hz), F2 (1028 Hz, 1700 Hz, 1870 Hz), F0 (220 Hz, 440 Hz, 880 Hz)

Atributi su „otvorenost“, „prodornost“ i „osnovna frekvencija“, a otvorenost može biti „široka ili „uska“. Konkretno su u bazi za svaki atribut zapisane vrijednosti svih parametara sinteze, a za svaki zvuk vrijednosti svih njegovih parametara i atributa.

„Artist“ koristi stablo odlučivanja koje služi za generaliziranje atributa. U čvorovima stabla nalaze se atributi, a na granama njihove vrijednosti. Listovi predstavljaju imena zvukova. Jedan primjer takvog stabla odlučivanja prikazan je na **Slika 2.4**. Treniranje stabla odvija se na primjerima koji su raspoloživi unutar baze „Artista“. Ako korisnik zatraži ime postojećeg zvuka ili unese potpunu listu atributa, sustav odabire parametre toga zvuka. No, u slučaju da korisnik preda nepotpunu listu atributa, sustav predlaže neki od postojećih zvukova koji najbolje ispunjava zahtjev.



Slika 2.4 Stablo odlučivanja za generaliziranje atributa kod sustava „Artist“

2.3 Vizualiziranje rezultata sinteze zvuka

Područje vizualizacije različitih vrsta podataka istovremeno posjeduje dugu povijest, od samih početaka razvoja računarstva, ali je i sve zastupljenije i značajnije u današnjem svijetu potpomognuto pristupom velikim skupovima podataka (eng. *big data*) [49]. U kontekstu ovog istraživanja, a vezano uz doprinos vizualizaciji sinteze zvuka u domeni timbralnih atributa, bitna su, kao prvo, ona istraživanja koja proučavaju vizualizacije različitih značajki zvuka te, kao drugo, istraživanja koja se bave izlučivanjem tih značajki koje se vizualiziraju.

2.3.1 Pristupi vizualiziranju značajki zvuka i glazbe

Povijest rudimentarnih vizualizacija aspekata zvuka zapravo seže u razdoblje prije računala kad su različiti mjerni instrumenti na audio opremi korišteni za prikazivanje, primjerice, jačine zvuka [50]. Dodatno, budući da su audio signali inherentno matematički koncepti, različite njihove aspekte moguće je opisivati grafički pomoću matematičkih metoda koje se mogu prikazati grafički, najčešće krivuljama. Primjer takvih uređaja jesu i osciloskopi. Stoga je uz razvoj računala i grafičkih računalnih sučelja došlo i do razvoja vizualizacija zvuka koje su se oslanjale na postojeće fenomene iz matematike i na hardverska rješenja.

Kad je riječ o suvremenim načinima vizualiziranja glazbe i zvuka pomoću računala, možemo ih podijeliti u dvije skupine. U prvoj se skupini nalaze metode kojima je cilj precizno odrediti neke značajke zvuka te na temelju njih omogućiti glazbenicima, skladateljima, znanstvenicima itd. promatranje promjena tih značajki ovisno o promjenama u ulaznom signalu [51]. Ovakve se vizualizacije najčešće oslanjaju na spektrograme, VU metre i slične koncepte [52]. Takvi se alati za vizualizaciju najčešće pronalaze i koriste u digitalnim radnim stanicama za obradu zvuka, vizualnim programskim jezicima u glazbi itd.

U drugoj se skupini vizualizacija nalaze apstraktne vizualizacije čija namjena nije nužno precizno oslikavanje vrijednosti nekog aspekta zvuka već postizanje umjetničkog dojma. Rješenja prezentirana, primjerice, u [53] tako generiraju psihodelične, šarene prikaze koji se mijenjaju u ovisnosti o različitim značajkama zvuka poput tonaliteta i glasnoće. Jasno je da se takve vizualizacije ne mogu koristiti za daljnje analize zvuka, već one postaju dio umjetničkog doprinosa koji prati glazbu, a često je i izravno vezan za pojedinu formu glazbe [54].

2.3.2 Izlučivanje značajki za vizualizacije

Kako bi se metode za vizualizaciju mogle primijeniti nad generiranim ili ulaznim zvukom ili glazbom, potrebno je iz tog skupa podataka izlučiti značajke poput frekvencije, amplitude, tonskih obilježja i sl. Radovi poput [55] bave se upravo algoritmima kojima se to postiže, a usmjereni su na rad u stvarnom vremenu i učinkovitost.

No za potrebe ovog istraživanja bilo je nužno usredotočiti se na izlučivanje timbralnih atributa. Ti su postupci istraženi i opisani u poglavlju 2.2.4, a posebno su značajni radovi [56] i [57] koji opisuje metode kojima se na temelju različitih poznatih parametara zvuka (poput spektra) mogu donijeti zaključci o timbralnim atributima. Te su metode, dodatno razrađene i prilagođene, poslužile kao temelj dijela ovog istraživanja opisanog u poglavlju 3.5.

2.4 Uspoređivanje programa u vizualnoj domeni

Dok je uspoređivanje tekstnih programa često korištena i dobro istražena funkcionalnost s velikim brojem različitih ostvarenja [58], isto ne vrijedi za uspoređivanje vizualnih programa. Naime, dok je kod tekstnih programa moguće sve programske jezike uspoređivati na isti način s obzirom da koriste isti medij, kod vizualnih programskih jezika ne postoji zajednički medij već svaki jezik koristi često sasvim drukčije, specifične oblike prikaza. Drugim riječima, dok su kod tekstnih jezika prezentacija i sintaksa relativno odvojene putem leksičkih značajki teksta, kod vizualnih se paradigmi sintaksa jezika usko veže za njihovu prezentaciju i zasebne

načine pohranjivanja. Stoga su rješenja uspoređivanja vizualnih programa uvijek ovisna o ciljnom jeziku.

Od malog broja dostupnih istraživanja valja istaknuti rad koji istražuje uspoređivanje programa za programabilne logičke kontrolere (eng. *Programmable Logic Controller – PLC*) po standardu IEC 61131 [59]. U tom se radu izlaže način uspoređivanja programa koji kreće usporedbom tekstne reprezentacije, odnosno zapisa vizualnih programa. Rezultati dobiveni usporedbom tekstnih zapisa zatim se kroz različite transformacije reflektiraju u vizualnoj domeni.

S obzirom na izostanak značajnijih prethodnih istraživanja, razvoj alata za uspoređivanje programa jezika predloženog u ovom radu temeljio se na sličnim principima kao prethodno opisani rad, no uvažavajući posebnosti jezika. Prezentirano rješenje nije upotrebljivo u općenitom slučaju, ali temeljni principi mogu se preslikati na druge jezike

3 Vizualni programski jezik za sintezu zvuka i računalno stvaranje glazbe

U ovom je poglavlju opisan glavni doprinos rada i istraživanja: vizualni programski jezik za sintezu zvuka temeljenu na toku atributa i računalno stvaranje glazbe temeljeno na manipulacijama u vremenskoj domeni.

U uvodnom se dijelu poglavlja razmatraju motivacija i temeljni koncepti iza predloženog programskog jezika. Zatim se, prije samog predstavljanja jezika, objašnjavaju popratni doprinosi istraživanja. Ti doprinosi uključuju analize i evaluacije postojećih rješenja. Provedene kvantitativne i kvalitativne analize same po sebi predstavljaju doprinos te daju argumentaciju i usmjerenje prilikom oblikovanja novog jezika.

U nastavku poglavlja detaljno se opisuje arhitektura i elementi predloženog jezika te popratna rješenja u vidu koncepata, postupaka i biblioteka potrebnih za njegovu realizaciju. U tom se smislu posebno ističu metode postavljanja parametara sintetizatora, načini mapiranja timbralnih atributa u takve parametre i audio signale te analiza i predstavljanje generiranih ili postojećih zvukova vrijednostima timbralnih atributa.

Konačno, na kraju poglavlja opisani su postupak i rezultati evaluacije predloženog programskog jezika. Evaluacija je provedena u tri koraka: subjektivno – razgovorom s ciljnim korisnicima, kvantitativno – anketom i formalno – kognitivnim dimenzijama.

3.1 Motivacija i temeljni koncepti

Glavna motivacija i cilj istraživanja prezentiranog u ovom radu jesu omogućavanje inovativnih pristupa glazbi kao umjetničkoj formi. Takvi se pristupi oslanjaju na nove metode stvaranja glazbe, sinteze zvuka i modeliranju zvukova. Kao način za ostvarenje tog cilja, a argumentirano evaluacijom predloženog rješenja i analizom postojećih programskih jezika, izabrana je ideja spajanja intuitivnih načina unosa i interakcije s programom (poput timbralnih značajki) s vizualnim programskim elementima i vremenski-ovisnom kontrolom toka. Kao što je već spomenuto u poglavlju 2.1, iako je u postojećim vizualnim jezicima u domeni glazbe prisutan niz koncepata temeljenih na fuziji različitih paradigmi, primjerice u jeziku Kyma [30] zvukovnim se objektima manipulira u stvarnom vremenu, većina se oslanja na slične temeljne principe i jezgru izgrađenu oko toka signala. U praktičnom smislu, ograničenja tih alata uvjetuju da glazbenici i drugi korisnici moraju poznavati i razumijevati matematičko značenje audio signala kako bi u potpunosti mogli ovladati jezikom. Za razliku od njih, predloženi

vizualni programski jezik nadograđuje i sakriva implementacijske detalje temeljene na signalima, a prema korisnicima nudi sučelja temeljena na timbralnim atributima, pojmovima koji proizlaze izravno iz glazbene teorije.

Predloženi koncept definira timbralne atribute kao glavni element toka podataka, u polju programskih jezika često korištene paradigme, dok istovremeno omogućuje njihovu manipulaciju u vremenu kroz odgovarajuće elemente grafičkog sučelja. Ta dvojakost pristupa nije prisutna u sličnim istraživanjima i jezicima te se predloženi vizualni programski jezik može smjestiti u dvije različite temeljne paradigme vizualnog programiranja [60].

Prva od te dvije paradigme vezana je uz pristup temeljen na ikonama ili simbolima [61]. U predloženom rješenju, taj se pristup ogleda u elementima grafičkog sučelja koji omogućavaju definiranje timbralnih atributa i manipuliranje njihovim vrijednostima na pojedinim generatorima i sintetizatorima zvuka.

Druga temeljna paradigma kao središnju građevnu jedinicu koristi dijagrame, odnosno usmjerene grafove [62]. Kod takvih se vizualnih programskih jezika grafovima određuju veze i međuovisnosti među objektima koji predstavljaju funkcionalnosti poput sintetizatora, VST dodataka itd. Često se vremenski-ovisni spojevi među dijelovima ciljnog slijeda zvukova uspostavljaju na način koji podsjeća na programsku potporu za uređivanje zvuka poput Audacityja [63]. U predloženom programskom jeziku, atributi su podatci koji se izmjenjuju bridovima grafa, odnosno podatci koji se prenose od objekta do objekta.

Uzevši u obzir prethodno opisane koncepte, važno je naglasiti da timbralni atributi nisu slučajno izabrani kao temeljna, atomarna granula u toku podataka već je riječ o svjesnom i istraživanjem argumentiranom izboru donesenom tijekom oblikovanja programskog jezika. Korištenjem timbralnih atributa pojačavaju se spone između ciljne paradigme vizualnog programiranja te koncepata i načina razmišljanja koji su prirodni i lako razumljivi glazbenicima i sličnim korisnicima.

Tijekom razvoja modela jezika, uzeti su u obzir uobičajeni pristupi koji se javljaju u alatima poput prethodno spomenutog Audacityja te elementi postojećih vizualnih programskih jezika poput Pure Date (na čijoj je platformi razvijen i prototip). Kombiniranjem takvih svojstava korisniku je omogućeno da učinkovito i lako mijenja i istražuje sintetizirane zvukove - konačne produkte programiranja – i u vremenskoj domeni i u domeni različitih drugih karakteristika zvuka.

Rezultat istraživanja jest jezik u kojem korištenjem koncepata koji su obično prisutni u jezicima temeljenima na dijagramima toka i toku podataka, poput mogućnosti povezivanja različitih blokova za manipuliranje tokom atributa, korisnici mogu slagati sljedove zvukova i

definirati poveznice među zvukovima. Istovremeno, korisnici mogu mijenjati pojedinačne zvukove izravnim upravljanjem parametrima blokova za sintezu zvuka.

Osim olakšavanja korištenja i razumijevanja jezika, što će primjerice glazbenicima omogućiti da se koncentriraju na glazbene izražaje umjesto na nižu razinu značajki audio signala, cilj istraživanja je da se kroz manipulaciju timbralnih atributa umjesto audio signala korisnicima otvore nove mogućnosti po pitanju kreativnosti. Drugim riječima, iz samog nekonvencionalnog, inovativnog pristupa mogu proizaći novi oblici sintetiziranog zvuka i glazbe. Primjerice, glazbenik može odlučiti generirati zvuk koji će biti "drvenast", "topao" ili "grub", istovremeno definirati timbralne promjene kroz vrijeme te postići neočekivane, ali zanimljive rezultate.

3.2 Kvantitativna analiza

Kvantitativna analiza postojećih jezika provedena je u dva koraka. U prvom koraku, provedeno je anketno ispitivanje korisnika vizualnih programskih jezika u glazbi. Pitanja početne ankete formulirana su tako da se njihovom statističkom analizom dobiju korisni podatci o poželjnim značajkama jezika. Rezultati, metodologija i zaključci ovog dijela istraživanja opisani su u potpoglavlju 3.2.1 te detaljno prezentirani u [16].

Anketa korištena u prvom koraku poslužila je kao temelj za novu anketu, ali je zbog značajnih razlika u uključenim pitanjima bilo potrebno provesti novo prikupljanje podataka. Preoblikovanjem postojećih pitanja te dodavanjem novih, konačna anketa poslužila je prikupljanju specifičnih podataka o formalnom obrazovanju korisnika, kanalima putem kojih su došli u kontakt s pojedinim jezicima, korištenim izvorima znanja prilikom savladavanja jezika te načinima korištenja jezika. Rezultati, metodologija i zaključci ovog dijela istraživanja opisani su u potpoglavlju 3.2.2 te detaljno prezentirani u [17].

3.2.1 Poželjne značajke

Kao „poželjne značajke“ vizualnih programskih jezika definiramo one elemente, funkcionalnosti i procese prilikom korištenja jezika koji pozitivno utječu na iskustvo i rezultate rada. Za potrebe istraživanja, skup poželjnih značajki uključivao je koncepte poput „otpornosti na pogreške“, jednostavnost postizanja određenih ciljeva (poput generiranja željenih izvukova) itd. Osim mogućnosti da se međusobno usporede jezici, analiza je omogućila i izlučivanje poželjnih karakteristika koje nismo prvotno razmotrili.

U nastavku su opisani detalji kvantitativne analize poželjnih značajki kod postojećih jezika na temelju ankete i statističke obrade.

Anketa

Evaluacija putem ankete provedena je u četiri faze. Tijekom prve faze, stvoreni su upitnici za izabrane programske jezike (Pure Data, Max/MSP, OpenMusic, Kyma i Reaktor). Jezici su izabrani zbog njihovog značaja i specifičnih karakteristika, a na temelju prethodnih analiza i postojećih istraživanja. Pitanja u upitniku oblikovana su tako da ekstrahiraju znanje o određenim aspektima jezika bitnima za ovo istraživanje te o njihovom utjecaju na korisnike. Kao što je ranije navedeno, cilj upitnika bio je statistikama i provjerljivim podatcima potkrijepiti temeljne premise istraživanja te omogućiti oblikovanje novog jezika na temelju zaključaka izvedenih ih rezultata upitnika.

Svaki upitnik bio je podijeljen na dva dijela. Prvi set pitanja uključivao je uobičajena pitanja koja se nisu razlikovala među jezicima. Teme pitanja u tom setu su bile demografskog tipa, načini na koji su se korisnici upoznali s jezikom, detalji o iskustvima s drugim alatima itd.

Drugi se dio upitnika sastojao isključivo od pitanja posebno osmišljenih za specifičnosti pojedinih jezika. Primjerice, upitnik za Native Instruments Reaktor nije sadržavao pitanja o algoritamskom skladanju budući da jezik ne podržava takve konstrukte.

Kako bismo poboljšali upitnike, tijekom druge faze proveli smo intervjue s iskusnim korisnicima pojedinih jezika. Za svaki od jezika izabrano je po dvoje korisnika, eksperata koji su u doticaju s jezicima svaki dan, a proizlaze ili iz akademije ili se profesionalno bave glazbom. Tako prikupljeni podatci iskorišteni su da se utvrdi i provjeri korištena terminologija, poboljša razumljivost i smislenost pitanja te dodatno pojasne ponuđeni odgovori kod pitanja zatvorenog tipa.

Treća faza uključivala je objavljivanje anketa na raznim diskusijskim stranicama na internetu, forumima i društvenim mrežama. Dodatna diseminacija anketa ostvarena je izravnim kontaktiranjem sa znanstvenicima i glazbenicima koji obično sudjeluju u istraživanja vizualnih programskih jezika u glazbi ili se njima koriste u svom profesionalnom radu. Odgovori su prikupljani, a ankete ponovno diseminirane nekoliko puta kako bi se proširio fundus odgovora.

U konačnoj, četvrtoj fazi statistički su analizirani prikupljeni podatci. Za analizu podataka korišten je IBM-ov sustav SPSS Statistics Software Package [64]. U inicijalnoj analizi čija je svrha bilo izlučivanje poželjnih značajki jezika, svaki je upitnik obrađen zasebno, bez unakrsne analize među skupovima podataka. Izlazni rezultati programske potpore ručno su korigirani kako bi se smanjio utjecaj određenih rubnih slučajeva.

Konačni skup anketiranih korisnika uključivao je 61 korisnika Pure Date, 15 korisnika Kyme, 30 korisnika Max/MSP-a, 41 korisnika NI Reaktora i 15 korisnika OpenMusica.

Rezultati i zaključci

Analizom prikupljenih podataka uočeni su razni obrasci i značajke. U tom kontekstu, jedan od bitnijih rezultata ankete i statističke analize prikupljenih podataka jest činjenica da vizualna priroda promatranih jezika umjetnicima olakšava tehničko ostvarivanje njihovih ideja te ih čini produktivnima u relativnom kratkom roku.

Promatrajući podatke postaje jasno da se većina korisnika vezuje uz samo jedan jezik te da rijetko iskušavaju alternative, nevezano uz primjenu kojom se u nekom trenutku bave. Ova tvrdnja stoji i u slučajevima kad postoje bolji, namjeni prilagođeniji jezici. Primjerice, iako je Kyma najbolje prilagođena za dizajn zvuka, a OpenMusic za algoritamsko skladanje, korisnici svejedno najčešće koriste Pure Datu za te primjene. Iz analiziranih podataka također je jasno da korisnici Pure Date i Max/MSP-a nisu svjesni komercijalno dostupnih alata poput Kyme ni novijih i manje rasprostranjenih jezika poput OpenMusica.

Na temelju broja prikupljenih odgovora na anketu u prvom prolazu, možemo zaključiti da većina korisnika daje prednost jezicima otvorenoga kôda poput Pure Date. Posljedično, Pure Data ima najraznolikiju i najširu bazu korisnika. Iako se ta pojava može objasniti činjenicom da je riječ o besplatnom softveru ili dostupnošću literature na webu, bitnu ulogu u popularnosti jezika ima i "uradi sam" priroda jezika (eng. *DIY – "do it yourself"*), zajednica korisnika koja ga okružuje te raznolikost funkcionalnosti koje nudi.

Unatoč tome što prema rezultatima našeg istraživanja ne postoji konsenzus oko njegovih ograničenja i glavnih nedostataka, većina anketiranih korisnika jeziku spočitava zastarjeli izgled grafičkog sučelja kao i korisničko iskustvo prilikom njegova korištenja. Mnogi nedostatci koji su se pojavili u anketi, poput izostanka vremenske crte, već su riješeni u jezicima s kojima korisnici Pure Date nisu upoznati, najčešće u Kymi. Upitani za faktore koji negativno utječu na njihovu produktivnost prilikom korištenja jezika, anketirani korisnici ističu izostanak vremenske dimenzije i pretjeranu oslonjenost na tok audio signala. Ostaje nejasno imaju li ti nedostatci utjecaja na kvalitetu njihova rada. Značajan broj korisnika ističe da prilikom rada s Pure Datom moraju uložiti dodatan trud i produbiti temeljno znanje programiranja kako bi ostvarili zamišljeno. Slično kao i po pitanju kvalitete rada, ostaje otvoreno pitanje kakvog utjecaja takav način rada ima na njihovu kreativnost budući da se većina izjašnjava generalno pozitivnim dojmom o Pure Dati.

Na sličan je način moguće ocijeniti i preostale promatrane jezike. Rezultati ankete pokazuju da su neke pretpostavke i predrasude točne. Primjerice, u kontekstu načina korištenja i primjene, Pure Data se najčešće koristi za izradu interaktivnih sustava i skladanje glazbe. Max/MSP, s druge strane, najčešće pronalazimo u ulozi jezika za ostvarivanje automatiziranih skladateljskih tehnika. Kyma je, pak, naučestalija u situacijama kada je potrebno skladati glazbu na formalan način i stvarati sintetizatore zvuka. Konačno, Reaktorom se najčešće grade zvučni efekti. Uzroci takve raspodjele nisu jasni iz rezultata i teško ih je obuhvatiti kvantitativnom analizom, posebno jer korisnici najčešće poznaju samo jedan jezik. Moguća objašnjenja uključuju prozaične uzroke poput toga da je riječ o marketingu, ali mogu se objasniti i kao posljedice značajki pojedinih jezika.

Unatoč tome što većina korisnika svih promatranih vizualnih programskih jezika dijeli sličnu razinu formalnog obrazovanja i iskustva, najveće razlike pronalazimo kod korisnika NI Reaktora. Kod njih je razina formalnog obrazovanja znatno niža, a posebno su odani i ekskluzivni u korištenju Reaktora. Istovremeno, zaziru od korištenja Reaktorovih naprednih funkcionalnosti kao što je uređivanje temeljnih gradivnih blokova tekstnim programiranjem.

Imajući u vidu prethodno spomenuta saznanja, jasnijom postaje činjenica da korisnici najčešće biraju, koriste i proglašavaju najbitnijima najuočljivije značajke pojedinih jezika. Na primjer, korisnici Max/MSP-a i Pure Date ističu njihovu fleksibilnost, Kymini korisnici oslanjaju se na potencijale njezinih alata za dizajn zvuka poput automatizacije parametara i uređivača zvučnog spektra, dok korisnici OpenMusica najviše vrednuju elemente za unos notnog zapisa i maquette. Iz ovih rezultata proizlazi zaključak da se većina jezika tijekom svog razvoja prilagodila očekivanjima i potrebama svojih korisnika.

Većina anketiranih glazbenika i umjetnika ne smatra da su potrebne velike paradigmatske promjene u jezicima i alatima koje koriste. Ipak, vrijedi napomenuti da mnogi jezici evoluiraju na način da dodaju i uključuju različite nove alate dok glavna, temeljna paradigma ostaje nepromijenjena. Bitna iznimka od ovog pravila je funkcionalnost vremenske crte koja je dodana u Kymu kad je Kyma kao jezik već bila u zreloj fazi. Iako je primjetan izostanak značajnih unaprijeđenja, manja poboljšanja i dorade postojećih mehanizama zbivaju se konstantno.

S obzirom na to, iz rezultata i postojećih jezika moguće je zaključiti da su dodatna poboljšanja jezika koja unaprijeđuju i pomažu glavnim funkcionalnostima zanimljivo i plodonosno područje. Primjeri takvih poboljšanja su uređivač zvukovnog spektra u Kymi te jednostavni alat za uvoz multimedijskih datoteka u Max/MSP-u. Rezultati ankete potvrđuju

intuitivnu premisu da su takvi specifični alati korisni u praksi i pri ponavljanju učestalih zadataka.

Kad je riječ o poboljšanjima drugih aspekata vizualnog programiranja najznačajnija je Kymina otpornost na pogreške. To svojstvo proizlazi iz inherentnih karakteristika Kymine paradigme i čini je privlačnom i pogodnom za neiskusne korisnike u području dizajna zvuka.

Većina problema koji su iz rezultata ankete proizašli kao bitni vezana je uz inherentne nedostatke programiranja u vizualnoj domeni [4]. Mimo toga aspekta, korisnici navode izostanak kontrolnog toka u Kymi i Reaktoru kao glavne probleme. Unatoč tome što bi za većinu proučenih jezika mogli reći da su relativno jednostavni za početnike, složeniji i kognitivno zahtjevniji zadatci traže određeno iskustvo.

Osim spomenutih specifičnih problema, javljaju se i neki zajednički nedostatci. Kao primjer navodimo sinkronizaciju događaja i zvukova u vremenu, oslonjenost na tok audio signala te izostanak vremenske dimenzije. Većinu ovih problema nemoguće je riješiti bez značajnih promjena u paradigmi poput korištenja toka timbralnih atributa umjesto toka signala.

3.2.2 Ekosustavi vizualnih programskih jezika i utjecaj formalnog obrazovanja

Kao važan sporedni doprinos doktorskog istraživanja i izvor temeljnih informacija koje su uvjetovale formiranje novog jezika, u nastavku je obrazložena kvantitativna evaluacija o ekosustavima vizualnih programskih jezika i utjecajima formalnog obrazovanja na rad s vizualnim programskim jezicima. U kontekstu istraživanja, pod „ekosustavima vizualnih programskih jezika“ podrazumijevamo međusobne odnose autora, edukatora i korisnika kao subjekata te samih jezika kao objekata u području.

Tim je istraživanjem i analizom prikupljenih podataka dobivena slika o odnosu korisnika i jezika s kojima se susreću u svakodnevnom radu. Promatrane razine uključuju dizajn interakcije ljudi i računala (eng. *human-computer interaction*) te višu, društveno-psihološku razinu. Razvoj novog jezika bio je vođen tim saznanjima.

Anketa

Kvantitativni rezultati koji su korišteni za zaključke u ovom dijelu istraživanja rezultat su analize anketnih odgovora o vizualnim programskim jezicima specijaliziranima za stvaranje glazbe. Za svaki je jezik (Pure Data, Max/MSP, OpenMusic, Kyma, NI Reaktor) stvoren upitnik s istim skupom pitanja, no pitanja su prilagođena specifičnim terminologijama

svakoga jezika. Anketirani korisnici bili su upućeni da ispunjavaju samo ankete za jezike s kojima su upoznati.

Upitnici su se sastojali od pitanja s višestrukim ponuđenim odgovorima kako bi se olakšala njihova analiza te su bili podijeljeni u tri dijela. Uvodni je dio bio fokusiran na pitanja o demografiji i obrazovanju s posebnim naglaskom na razinu formalnog obrazovanja. U drugom su dijelu upitnika bila uključena pitanja o tome kako su i gdje korisnici saznali za jezik, koje resurse su koristili dok su učili koristiti se jezikom te koliko im je dugo trebalo da postanu produktivni. Primjer pitanja s višestrukim odgovorima temeljenima na Likertovoj skali u ovom dijelu upitnika jest "Koliko često koristite jezik Pure Data?" s ponuđenim odgovorima "nikad", "rijetko", "ponekad", "često" i "svakodnevno". Primjer kategoričkog pitanja u istom dijelu upitnika jest "Kako ste saznali za jezik?" s ponuđenim odgovorima "od prijatelja ili kolege", "putem magazina, časopisa ili knjige" te s otvorenim odgovorom "ostalo". Mogućnost unosa osobnog odgovora ponuđena je zbog cjelovitosti iako su ponuđeni odgovori pažljivo izabrani, priređeni i validirani kako bi se osiguralo da korisnici u većini slučajeva pronađu željeni odgovor među ponuđenima.

Treći dio upitnika odnosio se na produktivnost korisnika i stvarna iskustva rada s jezicima. Sudionike ankete upitali smo o namjenama i ciljevima njihova korištenja jezika, o broju radova koje su njima stvorili, o oblicima glazbenih kompozicija koje su stvorili, o učestalosti pojavljivanja pogrešaka u njihovu kôdu itd. Jedan primjer pitanja u ovom dijelu uključuju "Za koje namjene koristite Pure Datu" s ponuđenim odgovorima poput "za stvaranje glazbenih kompozicija" i "za stvaranje interaktivnih sustava". Drugi primjer jest "Koliko ste programa ili glazbenih radova ostvarili u Pure Dati?". Ovo pitanje nije imalo ponuđenih odgovora već je ispitanik morao odgovoriti brojkom.

I ova anketa je pripremana u četiri faze kao i prethodna (3.2.1). U trećoj fazi anketa je objavljena na raznim diskusijskim stranicama na internetu, forumima i društvenim mrežama te izravnim kontaktiranjem kao i u prethodnoj anketi. Konačno, ankete su postavljene i na druga okupljališta korisnika vizualnih programskih jezika u glazbi poput Twittera ili SoundClouda.

Tijekom diseminacije upitnika vođeno je računa da oni dosežu podjednak broj korisnika svih uključenih jezika te da se iscrpe odgovarajući kanali. Takvim smo pristupom pokušali smanjiti utjecaj profila naših anketiranih korisnika budući da se izravnom selekcijom i kontaktiranjem korisnika ili selekcijom temeljenom na nekoj istaknutoj karakteristici uvodi pristranost odabira (eng. *selection bias*).

Konačna grupa sudionika ankete uključivala je 73 korisnika Pure Date, 21 korisnika Kyme, 47 korisnika Max/MSP-a, 54 korisnika NI Reaktora i 23 korisnika OpenMusica. Odgovori su prikupljani, a ankete iznova diseminirane nekoliko puta kako bi se proširio fundus odgovora.

Tablica 3.1 Demografske značajke anketiranih korisnika vizualnih jezika u glazbi

Jezik	**Broj anketiranih korisnika**	**Spol**	**Dob**
Pure Data	73	Ženski: 5% Muški: 95%	Raspon: 17-65 Prosjek: 32,81 Devijacija: 10,65
Kyma	21	Ženski: 5% Muški: 95%	Raspon: 25-61 Prosjek: 41,71 Devijacija: 10,17
Max/MSP	47	Ženski: 6% Muški: 94%	Raspon: 22-58 Prosjek: 37,72 Devijacija: 11,05
Reaktor	54	Ženski: 2% Muški: 98%	Raspon: 16-67 Prosjek: 38,11 Devijacija: 12,75
OpenMusic	23	Ženski: 4% Muški: 96%	Raspon: 24-60 Prosjek: 47,04 Devijacija: 10,25

Demografske značajke sudionika ukratko su prikazane u tablici (**Tablica 3.1**). Promatrajući formalno obrazovanje anketiranih korisnika, 46% ih je završilo konzervatorij ili akademiju (prvostupnik ili magistar glazbe), 17% je samoukih, 14% ih je završilo neki oblik naprednog formalnog glazbenog obrazovanja (viši razredi glazbene škole i slično), 12% posjeduje temeljnu glazbenu naobrazbu (osnovna glazbena škola, pripremna škola), 10% ih je pohađalo neke oblike satova glazbe, dok je 1% korisnika bez ikakvog glazbenog obrazovanja.

Za potrebe ovog dijela istraživanja, promotreni su razni aspekti otkrivanja, učenja i korištenja jezika vezano uz njihovu formalnu glazbenu naobrazbu te veze između ovih značajki i šireg konteksta. Značajan broj zanimljivih zaključaka u ovom dijelu istraživanja temelji se na analizi parova pitanja. Statistički testovi izabrani su na temelju tipa odgovora. Parovi kategoričkih pitanja analizirani su tablicama za nepredviđene slučajeve (eng. contingency tables) i primjenom testova hi-kvadrat hipoteza (eng. *chi-squared hypothesis tests*). Parovi kategoričkih i rednih odgovora analizirani su Wilcoxonovim testom rang-sume (eng. *Wilcoxon rank-sum test*). Korelacije među rednim odgovorima izračunati su

Spearmanovim rangom korelacija. Konačno, odnosi kategoričkih i brojčanih odgovora ispitani su ANOVA-om.

Nakon što smo temeljem analize prikupljenih podataka pronašli područja koja su posebno zanimljiva po pitanju postojećih boljki i mogućih poboljšanja, sljedeći je korak bio razmotriti odnos formalnog obrazovanja i pojedinih jezika.

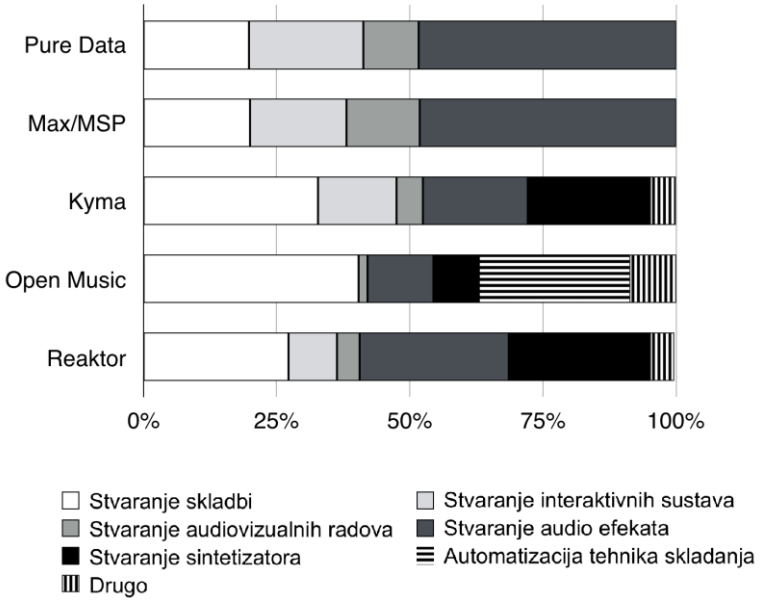
Istraženi su međusobni odnosi dimenzije formalnog obrazovanja, načina otkrivanja jezika i pristupa koji korisnici koriste tijekom interakcija s jezicima i alatima. Iz tih su odnosa proizašli zaključci o odnosima svih dimenzija koje imaju utjecaja na otkrivanje, učenje i korištenje različitih vizualnih programskih jezika. Na taj je način skicirana referentna okosnica za pedagoge, programere vizualnih jezika i krajnje korisnike koja definira poboljšanja postojećih alata i pospješuje ostvarivanje boljih, produktivnijih interakcija. Same usporedbe provedene su isključivo među promatranim vizualnim programskim jezicima dok su izvan opsega ostale usporedbe s drugim paradigmama poput tekstnih.

Tom je analizom dodatno validiran pristup u oblikovanju jezika koji se sastojao od inkrementalnog poboljšanja paradigme timbralnim atributima, ali bez zadiranja u temeljne postavke postojećih jezika poput Pure Date. Iz ove su analize također proizašla saznanja koja upućuju na koja se područja potrebno fokusirati prilikom razvoja jezika i njegova približavanja što širem krugu korisnika.

Prezentirana kvantitativna analiza temeljena je na 218 ispitanika koji su dali ukupno 3488 odgovora.

Rezultati i zaključci

Analizom rezultata ankete izlučeni su zanimljivi zaključci i trendovi vezani uz vizualne programske jezike u glazbi, njihove autore, korisnike i edukatore te uz utjecaj demografskih, obrazovnih i drugih faktora na njihovo korištenje. Detaljna analiza anketa prezentirana je u radovima [16] i [17], dok je na ovom mjestu fokus stavljen na one aspekte koji su korišteni za usmjeravanje i validiranje predloženog programskog jezika temeljenog na toku timbralnih atributa. Četiri su najvažnije stavke u tom smislu: namjene za koje se jezici koriste, bitni faktori pri učenju jezika, načini otkrivanja jezika te utjecaj demografskih i obrazovnih pretpostavki.



Slika 3.1 Primjene vizualnih programskih jezika u glazbi

Od anketiranih korisnika, najveći broj koristi vizualne jezike za stvaranje: skladbi (74%), audio efekata (58%), sintetizatora zvuka koji se zatim koriste u kompozicijama (51%), interaktivnih sustava (47%), automatizacije skladateljskih postupaka (33%) i audiovizualnih radova (26%). Rezultate prikazuje **Slika 3.1**. Budući da su pitanja vezana o namjenama korištenja jezika omogućavala odabir višestrukih odgovora, analizirali smo i parove namjena. Iz te analize proizlazi da 44% korisnika istovremeno koristi jezike za stvaranje skladbi i pripremanje audio efekata.

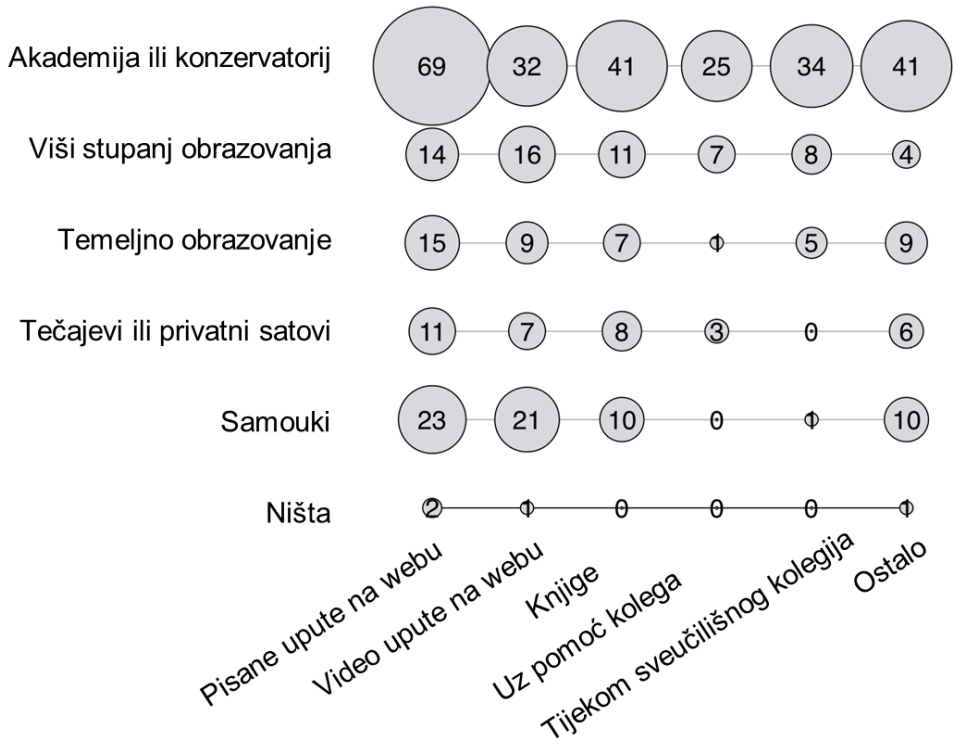
Rezultati pokazuju značajno raslojavanje primjena po jezicima što je potvrđeno hi-kvadrat testom ($p<0.1$). Korisnici OpenMusica i Kyme najćešće koriste te jezike za stvaranje skladbi, dok korisnici Reaktora pretežno realiziraju sintetizatore zvuka (28%) i

audio efekte (27%). Slično, korisnici OpenMusica najčešće koriste jezik za automatizaciju skladateljskih tehnika (28) što je i očekivano s obzirom na njegov skup značajki.

Kad je riječ o skladanju glazbe, 25% korisnika stvara algoritamske kompozicije, 20% akuzmatičnu glazbu, 17% pratnje za akustične instrumente ili ansamble, a samo 17% korisnika koristi jezike za računalne improvizacije. Dodatno, rezultati testa hi-kvadratom sugeriraju da razina formalnog obrazovanja značajno utječe na tip kompozicija koje korisnici stvaraju vizualnim programiranjem ($p<0,01$). Dok su improvizacije uz računala (eng. *laptop improvisation*) najčešće kod korisnika s nižom razinom formalnog obrazovanja, napredniji korisnici puno češće stvaraju pratnje za akustične instrumente ili ansamble te akuzmatičke kompozicije.

Kad je riječ o načinima učenja jezika, sudionici ankete najčešće su jezik učili pomoću pisanih uputa na internetu (61%), edukativnih videa (39%), knjiga (35%), kolegija na fakultetima (22%), iskusnijih korisnika (17%), radionica (11%) i ugrađene dokumentacije (6%). Najčešća kombinacija materijala za učenje pritom uključuje pisane i video upute na internetu (28%).

Analiza također pokazuje značajne razlike u načinima učenja ovisno o specifičnom jeziku što je potvrđeno hi-kvadrat testom ($p<0,01$). Korisnici Kyme, primjerice, najčešće su prilikom učenja koristili knjige (34%) i upute s interneta (24%). Nasuprot tome, korisnici Max/MSP-a većinom su učili alat putem kolegija na fakultetu (18%). Iz ovog je dijela analize jasno da, osim samih značajki jezika, dostupnost i oblici diseminacije jezika utječu na interakcije s korisnicima.



Slika 3.2 Načini učenja vizualnih programskih jezika u glazbi

Odnosi razine obrazovanja i načina učenja prikazuje **Slika 3.2**. Kao i kod načina korištenja, postoji statistička veza između formalnog obrazovanja i načina učenja, kao što je potvrđeno testom hi-kvadrata ($p<0,05$). Pritom korisnici s višim razinama formalnog obrazovanja jezike najčešće usvajaju tijekom predavanja na sveučilištima (14%), dok s druge strane samouki korisnici najčešće uče pomoću pisanih i video uputa s interneta.

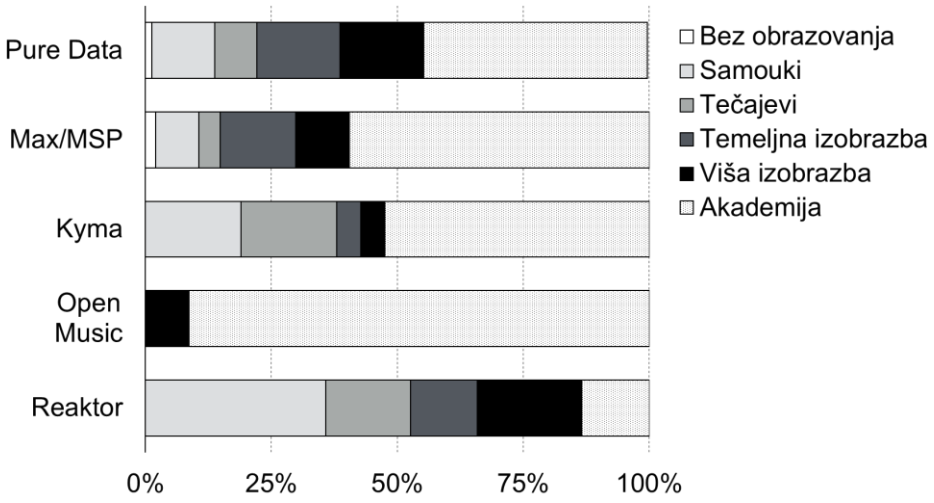
Dodatno, rezultati pokazuju da postoji i veza načina na koji korisnici otkrivaju jezike s načinima na koje ih zatim uče. Primjerice, korisnici koji za jezik saznaju tijekom predavanja na sveučilištu najčešće ne koriste druge materijale, dok samouki korisnici ili korisnici nižih razina formalnog obrazovanja najčešće uče pomoću materijala dostupnih na internetu (35%).

U kontekstu utjecaja pojedinih jezika na brzinu usvajanja jezika i postizanja produktivnosti, rezultati ankete sugeriraju da nema većih razlika među samim jezicima. No zato rezultati Spearmanova rangiranja pokazuju blagu korelaciju duljine korištenja jezika i postizanja produktivnosti ($\rho=0,17$, $p<0,01$) što znači da iskusniji korisnici smatraju da im je trebalo dulje vrijeme da postanu stvarno produktivni. Od anketiranih korisnika, 28% smatra da su postali produktivni nakon nekoliko tjedana, 21% nakon godine ili više dana, dok ostalih 17% smatra da im je za to bilo potrebno nekoliko dana.

Promatranjem odgovora na pitanja o učestalosti pogrešaka u radu, podatci pokazuju da formalno obrazovanje nema utjecaja te da korisnici čine sličan broj pogrešaka. No

istovremeno podatci pokazuju da postoji značajna korelacija značajki pojedinih jezika i činjenja pogrešaka. Primjerice, korisnici Pure Date češće čine pogreške zbog bežičnih veza, a korisnici Reaktora zbog nemogućnosti razlikovanja kontrolnih i audio spojeva. Istovremeno, Kymini korisnici čine najmanje pogrešaka što se može objasniti interaktivnošću jezika. Ovi su zaključci potvrđeni testom Wilcoxonovih suma rangova ($p<0,01$).

Anketirani korisnici su jezike najčešće otkrivali tijekom predavanja na sveučilištima (28%), osobnim preporukama (26%), iz magazina, časopisa ili knjiga (20%), na internetu (17%) itd. Test hi-kvadratom pokazuje da se načini otkrivanja značajno razliku po pojedinim jezicima. Dok se Max/MSP i OpenMusic najčešće susreću na sveučilištima (51% i 43%), Pure Data se najviše oslanja na diseminaciju osobnim preporukama (39%). Kyma i Reaktor se pak najčešće otkrivaju kroz časopise ili knjige (43% i 33%). Slično, korisnici s različitim razinama glazbenog obrazovanja otkrivaju jezike na značajno različite načine, kao što je pokazano testom hi-kvadrata ($p<0,01$). Tako 45% korisnika sa završenim konzervatorijem ili akademijom najčešće o jezicima saznaju na vlastitoj visokoobrazovnoj ustanovi, 57% korisnika s višim glazbenim obrazovanjem s jezikom se prvi put susreće kroz časopise ili knjige, 37% korisnika s temeljnim (osnovnoškolskim) glazbenim obrazovanjem pak o jeziku saznaje putem osobnih preporuka, dok većina (39%) samoukih korisnika jezike istražuje na internetu.



Slika 3.3 Razina formalnog obrazovanja korisnika pojedinih vizualnih programskih jezika u glazbi

Konačno, prema jednosmjernoj analizi ANOVA, postoje statistički značajne razlike u distribuciji dobi sudionika koji koriste različite vizualne programske jezike ($p<0,01$, $F=8,15$). Post hoc usporedbe dobivene Tukeyjevim HSD-testom pokazuju da je srednja dob korisnika

OpenMusica (M=47,05, SD=10,25) značajno viša od srednje dobi korisnika Max/MSP-a (M=37,72, SD=11,05), Pure Date (M=32,81, SD=10,65) i Reaktora (M=38,11, SD=12,75). Dodatno, korisnici Pure Date značajno su mlađi od Kyminih korisnika (M=41,71, SD=10,17). Istovremeno, postoji i značajna razlika u razini formalnog obrazovanja korisnika po jezicima. Test Wilcoxonovom sumom rangova pokazuje da OpenMusic ima najveći udio visoko obrazovanih korisnika, odnosno čak 91% korisnika završilo je konzervatorij ili akademiju. Pure Data (44%), Max/MSP (60%) i Kyma (52%) imaju slične udjele visoko obrazovanih, dok Reaktor ima značajno nižu razinu. Naime, više od polovice korisnika Reaktora je samoobrazovano ili s tek početničkom razinom obrazovanja (52%), a samo ih je 13% visokoobrazovanih. **Slika 3.3** prikazuje ukupne rezultate analize.

U ovom potpoglavlju prikazana analiza ponudila je bitna saznanja koja su korištena prilikom oblikovanja i validiranja novog programskog jezika. Posebni je naglasak stavljen na to da se jezik učini pristupačnim neovisno o razini obrazovanja i načinu učenja. Način na koji je oblikovan jezik detaljnije je opisan u poglavlju 3.4.

3.3 Evaluacija kognitivnim dimenzijama

Važan doprinos području vizualnih programskih jezika predstavljen je u radu "Usability Analysis of Visual Programming Environments - a 'cognitive dimensions' framework" [4] autora Thomasa R. G. Greena i Mariana Petrea u kojem je definiran koncept kognitivnih dimenzija. Kognitivne dimenzije uvode skup principa koji se koriste prilikom dizajna i evaluacije vizualnih programskih jezika, a temelje se na empirijskim i subjektivnim aspektima grafičkih oblika zapisa te njihovim semantičkim posljedicama. Budući da su kod vizualnih programskh jezika sintaksa i semantika dijelovi grafičkog sučelja i prikaza programa, alat poput kognitivnih dimenzija postaje izuzetno važan kako bi se programski jezici mogli međusobno uspoređivati, nove paradigme mogle evaluirati i kako bi se konačno mogli utvrditi nedostaci postojećih i doprinosi novih paradigmi.

Kognitivne dimenzije osmišljene su tako da nude istovremeno lako razumljivi, ali precizan pregled kvalitete pojedinih dizajna. One nude zajednički vokabular kojim je moguće opisivati i definirati elemente i paradigme vizualnih programskih jezika. Ipak, analiza putem kognitivnih dimenzija zadržava se na višim razinama apstrakcije dok se implementacijskim detaljima ne bavi izravno već oni proizlaze iz načela postavljenih na višim razinama. To znači da iako dva jezika mogu imati istu vrijednost određene dimenzije, njihove se manifestacije mogu značajno razlikovati.

Petre i Green definirali su četrnaest temeljnih dimenzija:

- gradijent apstrakcije (eng. *abstraction gradient*) – Definira minimalne i maksimalne razine apstrakcije u jeziku, odnosno njegovu zapisu, te mogućnost enkapsulacije. Na primjer, visoki gradijent apstrakcije manifestira se kroz mogućnost korištenja patcheva i patcheva unutar patcheva u jeziku Pure Data.
- bliskost preslikavanja (eng. *closeness of mapping*) – Daje ocjenu o tome koliko je vizualna reprezentacija jezika bliska stvarnoj domeni. Primjerice, blokovi koji predstavljaju mjerne instrumente u jeziku LabVIEW bliski su stvarnim instrumentima ponašanjem i izgledom.
- konzistentnost (eng. *consistency*) – Ovisi o tome koliko je jezik konzistentan u svojoj sintaksi i semantici. Konzistentnost se manifestira kroz razne elemente i oblike interakcije, odnosno kroz mogućnost da se učenjem izoliranog dijela jezika kasnije lako i intuitivno može odrediti („pogoditi“) kako se ponašaju ostali njegovi dijelovi. Primjer konzistentnosti je slučaj da nakon što korisnik nauči koristiti sintetizatorske blokove u Max/MSP-u, druge slične blokove može jednostavno i intuitivno razumjeti bez njihova ponovnog usvajanja "ispočetka".
- difuznost / sažetost (eng. *diffuseness / terseness*) – Koliko je simbola ili prostora na radnoj površini potrebno za postizanje određenog rezultata ili izražavanje određenog, željenog semantičkog sadržaja? Dva kontrastna primjera uočljiva su u jeziku Kyma u kojem postoje blokovi koji sadrže kompleksne funkcionalnosti poput potpunih generatora zvuka, a čiju je funkcionalnost moguće postići i nizom od više drugih blokova, odnosno kombinacijom izvora signala, obrade signala itd.
- sklonost pogreškama (eng. *error-proneness*) – Ova dimenzija promatra odnose između značajki vizualne paradigme i učestalosti pogrešaka prilikom njenog korištenja. Drugim riječima, pojedine varijacije u vizualnim jezicima mogu smanjiti ili povećati broj pogrešaka koji čine korisnici. Pozitivan primjer sklonosti pogreškama je Kyma koja omogućuje stvarnovremensku evaluaciju programa (čime se smanjuje broj pogrešaka), a negativan Pure Data koja na pojedinim blokovima nema ograničenja unosa nedozvoljenih vrijednosti. Takve se pogreške kasno i teško otkrivaju.
- kognitivno zahtjevne operacije (eng. *hard mental operations*) – Kroz ovu se dimenziju promatra koliko je intelektualnog ili kognitivnog napora potrebno utrošiti kako bi se postigao željeni rezultat u vizualnom jeziku. Posebno se razlikuju perspektive zahtjevnosti samog zapisa i nižih razina semantike. Primjerice, ako korisnik mora koristiti alate van samog jezika da bi mogao realizirati neku akciju (kao što je

korištenje bilješki u uređivaču teksta), onda taj jezik smatramo kognitivno zahtjevnim. Drugi primjer je oslanjanje na audio signale u jezicima poput Pure Date i Max/MSP-a u kojima korisnik često mora matematički izračun željenih signala (kao ulaznih parametara bloka) učiniti izvan samog jezika.

- skrivene ovisnosti (eng. *hidden dependencies*) – Dimenzija odgovara na pitanja o tome postoje li neke međuovisnosti među elementima jezika, a koje nisu prikazane grafički. Također se skrivenim ovisnostima smatraju i one ovisnosti čija usmjerenost nije jasno naznačena ili situacije u kojima nije jasno da će promjena u jednom dijelu programa utjecati na neki drugi njegov dio. Primjer toga je Max/MSP kod kojeg su sučelja s vanjskim ekstenzijama slabo definirana, pa njihovim korištenjem na radnoj površini ostaju skrivene unutarnje kompleksnosti koje mogu dovesti do nekompatibilnosti i neočekivanih rezultata.
- paralelno pregledavanje (eng. *juxtaposability*) – Određuje je li moguće dijelove vizualnog programa istovremeno pregledavati te ih međusobno uspoređivati (najčešće postavljanjem dijelova kôda jedno uz drugo). U ovom smislu, većina postojećih jezika iznimno je ograničena, a uspoređivanje programa složen je postupak i postiže se otvaranjem istog programa u više instanci.
- preuranjeno opredjeljivanje (eng. *premature commitment*) – Promatra situacije kod građenja programa te kvantificira koliko su strogo definirani redoslijedi postupaka kod programiranja. Drugim riječima, definira mogu li se isti rezultati postići na više načina. U ovom je kontekstu posebno važna situacija kad vizualni jezik od korisnika traži da donese odluke prije no što su mu dostupne sve potrebne informacije. Ako su takve odluke nužne, koliko je korisniku teško promijeniti zahvaćeni dio kôda i naknadno promijeniti odluku? Iako je većina vizualnih programskih jezika zahvaćena boljkom preuranjenog opredjeljivanja, neki jezici poput Kyme ili Reaktora omogućuju jednostavnu izmjenu ranijih dijelova programskog toka.
- progresivna evaluacija (eng. *progressive evaluation*) - U sklopu ove dimenzije vrednuje se koliko jezik nudi povratnih informacija i rezultata prije no što je program do kraja dovršen. Kao što je ranije spomenuto, Kyma nudi stvarnovremensku evaluaciju programa gdje se svakim dodanim blokom mijenja rezultat, a koji se pak konstantno prezentira korisniku.
- izražajnost uloga (eng. *role-expressiveness*) – Ako promatramo samo dijelove programa, koliko je jasna njihova uloga u ukupnom rješenju? Dajući odgovor na to

pitanje, ova dimenzija također definira i jasnoću uloga pojedinih atomskih elemenata jezika. Kod većine jezika ova dimenzija nije jednoznačna već se mijenja ovisno o promatranoj razini apstrakcije.

- sekundarna notacija (eng. *secondary notation*) – Ova se dimenzija bavi elementima koji nisu nužno usko vezani uz temeljnu sintaksu jezika, već donose dodatne korisne informacije putem komentara, boje pojedinih elemenata, rasporeda itd. Većina vizualnih jezika sadrži mogućnost zadavanja komentara, a česte su i mogućnosti (kao u OpenMusicu) da se segment blokova grupira u zajedničku vizualnu "kutiju" obojenih rubova.
- viskoznost (eng. *viscosity*) – Jedna od bitnih i najsloženijih kognitivnih dimenzija, viskoznost definira koja i kakva ograničenja postoje u vizualnom zapisu jezika te na koji način ona priječe jednostavne izmjene u programima. Dimenzija također daje odgovor na pitanje koliko je teško učiniti željene izmjene u programu. Dodatno se ova dimenzija dijeli na tri pod-dimenzije:
  - domino-efekt (eng. *knock-on viscosity*) – Promjena u nekom dijelu programa remeti unutarnju strukturu cijelog programa te uzrokuje nužnost promjena u drugim dijelovima koji zatim uzrokuju potrebu mijenjanja trećih dijelova itd.
  - ponavljanje (eng. *repetition viscosity*) – Kako bi se promijenio jednostavni semantički aspekt programa, postaje nužno učiniti mnogo istih, ponovljenih promjena kroz cijeli program. Npr. kako bi se promijenila amplituda signala u Max/MSP-u, potrebno je promijeniti vrijednost na svim blokovima koji taj signal obrađuju.
  - opseg (eng. *scope viscosity*) – Ova dimenzija javlja se u slučajevima kad je zbog promjene ulaznih podataka nužno promijeniti samu strukturu programa. Na primjer, kako bi na ulazu nekog programa u Pure Dati mogli dodati novi audio signal, potrebno je uvišestručiti sve elemente za njegovu obradu, a nije moguće samo umetnuti signal kao ulaz u postojeće blokove.
- vidljivost (eng. *visibility*) - Konačno, ova dimenzija definira koliko se brzo i jednostavno mogu pronaći željeni dijelovi programa te koliko je teško njima pristupiti i učiniti ih vidljivima.

3.3.1 Metodologija

Kako bi na temelju postojećih jezika zaključili koje su poželjne karakteristike vizualnih programskih jezika u glazbi općenito, analizirali smo Pure Datu, Max/MSP, OpenMusic, Kymu i Reaktor. Fokus je pritom bio na njihovim temeljnim konceptima, paradigmi i značajkama. Analiza je provedena primjenom kognitivnih dimenzija. Kako bismo se upoznali sa samim jezicima, u svakom od njih implementirali smo nekoliko uobičajenih programa poput jednostavnih ritam mašina te po jedan program specifičan za jezik (npr. modularni sintetizator u Reaktoru). Dodatno, proučili smo postojeću literaturu, korisničke priručnike, knjige, rasprave na internetskim forumima i druge materijale vezane uz svaki od pojedinih jezika. Budući da je praktični aspekt rada s jezicima najbitniji za shvaćanje tehničkih detalja, proučili smo fundus postojećih primjera, isprobali različite jedinstvene funkcionalnosti u svakom od jezika te kontaktirali sa zajednicama korisnika jezika.

Kao što je prethodno opisano u poglavlju 3.2, proveli smo i niz intervjua s amaterskim i profesionalnim korisnicima analiziranih jezika. Cilj tih intervjua bio je prikupljanje kvalitativnih ocjena i saznanja o korisničkim navikama i pretpostavkama u radu koje su obuhvaćene kognitivnim dimenzijama. Intervjuirali smo trojicu profesora elektroničke kompozicije s glazbene akademije, petero profesionalnih i amaterskih elektroničkih glazbenika te profesionalnog softverskog inženjera u polju glazbene tehnologije. Struktura intervjua pratila je pitanja iz anketa opisanih u poglavlju 3.2.1 te radu [16]. No za razliku od tih anketa, pitanja koja smo postavljali bila su otvorenog oblika te proširena neformalnim digresijama, anegdotama i praktičnim primjerima.

Na temelju znanja prikupljenog gore opisanim postupcima i proučavanjem sličnih radova u drugim područjima [65], sistematično smo pripremili kvalitativne formulacije o kognitivnim dimenzijama za svaki od jezika. Dodatno, radar-dijagramima prikazali smo relativne odnose među jezicima po dimenzijama.

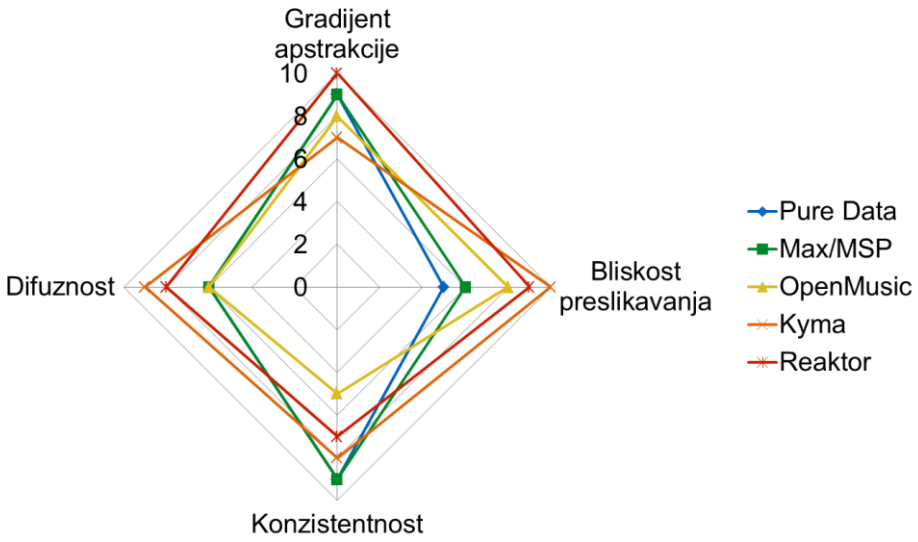
Konačno, naše zaključke o kognitivnim dimenzijama potkrijepili smo statističkom analizom pitanja iz anketa opisanih u poglavljima 3.2.1 i 3.2.2 na način da smo analizirali pojedina pitanja koja je moguće izravno vezati uz određene dimenzije. Primjerice, pitanje „Koliko često činite pogreške?“ (eng. *"How often do you make mistakes?"*) u izravnoj je vezi s dimenzijom sklonosti pogreškama (eng. *error-proneness*).

Detaljni rezultati analize kognitivnim dimenzijama prezentirani su u prilogu (

Prilog A – analiza kognitivnim dimenzijama) dok je u nastavku prezentiran sažeti pregled njihova relativnog odnosa po jezicima uz prikaz radar-dijagramima.

3.3.2 Relativni odnos kognitivnih dimenzija

Na temelju analize kognitivnim dimenzijama pripremljeni su radar-dijagrami koji olakšavaju međusobno uspoređivanje promotrenih jezika. Ocjene kvalitete po svim dimenzijama su u opsegu od 0 do 10 pri čemu 10 predstavlja najbolju ocjenu bez obzira na naziv pojedine kognitivne dimenzije. Napomena: na nekim se dijagramima vrijednosti dimenzija i njihove linije preklapaju, pa nisu jasno vidljive (npr. plava kad se preklapaju zelena i plava linija).



Slika 3.4 Relativni odnos među jezicima po kognitivnim dimenzijama gradijenta apstrakcije, difuznosti, bliskosti preslikavanja i konzistentnosti

Gradijent apstrakcije

Kad je riječ o gradijentu apstrakcije, svi promatrani jezici pokazuju dobre značajke uz mnoštvo različitih mehanizama apstrahiranja i enkapsulacije. Posebno se tu ističu Reaktor, Max/MSP i Pure Data koji nude različite modele uključivanja nižih razina programa čime je omogućeno stvaranje jednostavnih objekata koji u sebi sakrivaju implementaciju složenih procesa poput sinteze zvuka. Iako je Kymin sustav apstrakcija najnapredniji od proučenih, on ne uključuje mogućnost manipulacija i enkapsulacije sirovih podataka na najnižoj razini.

Bliskost preslikavanja

Zbog svoje izražajnosti i prilagođenosti domeni glazbe, Kyma i Reaktor pokazuju najbolje značajke u dimenziji bliskosti preslikavanja. Njihovi ugrađeni objekti izrađeni su tako da odgovaraju sučeljima iz fizičkog svijeta koji su poznati glazbenicima, a nude i dodatne

mogućnosti poput vremenske crte. Ta su sučelja glazbenicima prepoznatljiva iz drugih, često korištenih programa u svijetu audio-produkcije (npr. Ableton Live).

Iako je OpenMusic prilagođen i blizak skladateljima zbog mogućnosti korištenja notnoga zapisa, u drugim segmentima značajno je siromašniji po pitanju bliskosti preslikavanja. Nasuprot Kymi, Reaktoru i OpenMusicu, Pure Data i Max/MSP žrtvuju bliskost preslikavanja kako bi omogućili širu upotrebu jezika, primjerice za izradu vizualnih radova.

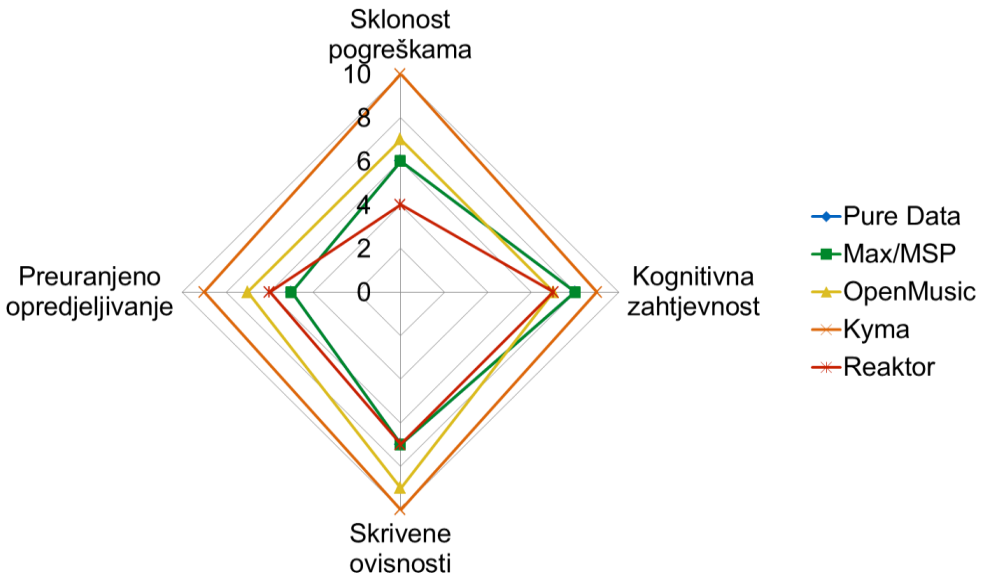
Konzistentnost

Među promatranim jezicima najkonzistentiji su Pure Data i Max/MSP jer na svim razinama i za ostvarivanje svih funkcionalnosti koriste isti sintaktičko-semantički pristup. Štoviše, čak su i datoteke za pomoć korisnicima (eng. *help files*) realizirane na isti način kao svi patchevi. Na drugoj krajnosti nalazi se OpenMusic koji sadrži višestruke, međusobno vrlo različite načine programiranja koje je potrebno i zasebno usvojiti prilikom učenja jezika.

Kad je riječ o Reaktoru, jedina se nekonzistentnost javlja na najnižoj razini programiranja gdje se proširenja kodiraju tekstnim jezikom. No budući da je to napredna funkcionalnost sakrivena od običnih korisnika, može se zanemariti njen utjecaj na ukupnu konzistentnost jezika.

Difuznost

Na najvišoj razini, Kyma i Reaktor imaju najbolju ocjenu po pitanju difuznosti jer sadrže veliki broj gotovih, specijaliziranih objekata i elemenata (primjerice ritam-mašina) koje je moguće jednostavno kombinirati za ostvarivanje željenih krajnjih funkcionalnosti. S druge strane, Pure Data, OpenMusic i Max/MSP slični su jezici opće namjene čija je difuznost posljedica postojanja velikog broja objekata koji ostvaruju funkcionalnosti niske razine. Potonje je potrebno kombinirati da bi se ostvarile složenije operacije koje su u Reaktoru ili Kymi već ugrađene. Iako negativan po pitanju difuznosti, takav pristup omogućuje da se Pure Data, Max/MSP i OpenMusic koriste za širi skup primjena.



Slika 3.5 Relativni odnos među jezicima po kognitivnim dimenzijama sklonosti pogreškama, preuranjenog opredjeljivanja, skrivenih ovisnosti i kognitivne zahtjevnosti

Sklonost pogreškama

Kyma zbog svojih mogućnosti kontinuirane progresivne evaluacije i automatskog nadovezivanja objekata tijekom uređivanja programa nudi najbolju otpornost na pogreške. Ako do pogreške i dođe, programer je može odmah uočiti jer se izlaz programa (zvuk) mijenja u stvarnom vremenu, odmah nakon učinjene promjene. Na isti način programer može i lakše otkloniti pogrešku ili eksperimentirati s različitim rješenjima.

OpenMusic, Reaktor, Max/MSP i Pure Data pokazuju slične karakteristike kad je riječ o otpornosti na pogreške. Svaki od tih jezika sadrži neke specifične mehanizme (npr. bežične veze kod Pure Date i Max/MSP-a) koji čine pojavu pogrešaka vjerojatnijom, a izostanak alata za otkrivanje pogrešaka ili uspoređivanje vizualnih programa čini njihovu detekciju značajno težom.

Svim analiziranim jezicima zajednička je sklonost pogreškama zbog mogućeg nerazumijevanja parametara i matematičkog značaja audio signala.

Kognitivno zahtjevne operacije

Svi promatrani jezici pokazuju dobre značajke kad je riječ o kognitivnom zahtjevnim operacijama. Sve mogućnosti pojedinih jezika moguće je koristiti bez potrebe za posezanjem za vanjskim alatima ili pomagalima. Pojedini jezici nude mogućnost tekstnog programiranja na najnižoj razini (OpenMusic, Reaktor), ali ta je mogućnost izolirana, nije neophodna i njeno

je korištenje ostavljeno na izbor naprednim korisnicima. Općenito, analiza nije pokazala značajnije kognitivne prepreke ugrađene u same jezike osim korištenja audio signala kao temeljnog, glazbenicima često teško razumljivog oblika podataka.

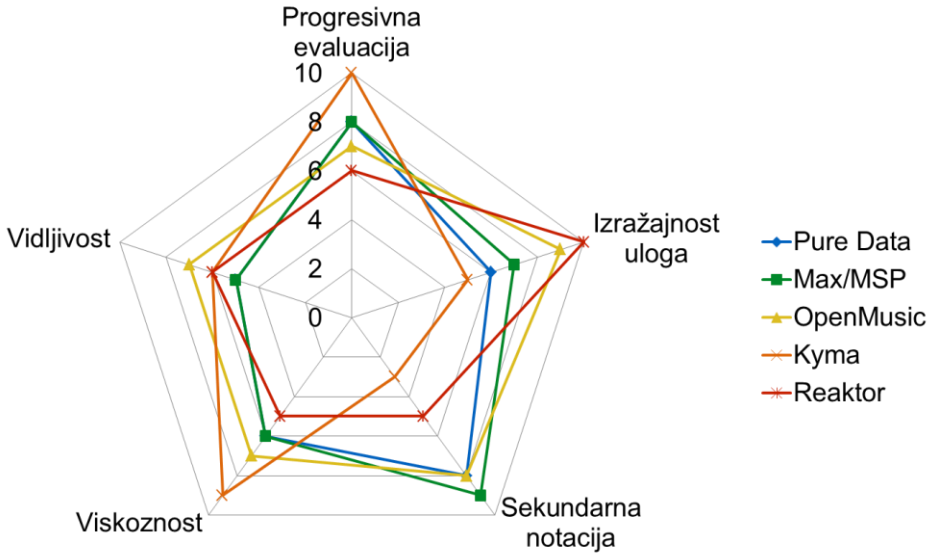
Skrivene ovisnosti

Zbog mogućnosti korištenja bežičnih veza i vanjskih proširenja, Max/MSP, Pure Data i Reaktor loše su ocijenjeni u dimenziji skrivenih ovisnosti. OpenMusic i Kyma vanjska proširenja ili ne podržavaju ili njihove ovisnosti jasno označuju.

Preuranjeno opredjeljivanje

Kyma zbog svoje mogućnosti istovremenog uređivanja i evaluacije programa postiže najbolje rezultate u kontekstu izostanka preuranjenog opredjeljivanja. Programi se grade slijedno te je njihov izlaz jasan u svakom trenutku.

S druge strane, Pure Data, Max/MSP, Reaktor te dijelom OpenMusic traže od programera da unaprijed planira strukturu programa kako bi krajnji cilj programa bio ostvariv. Pogreške u početnim fazama programiranja mogu prouzročiti značajne poteškoće pri kraju razvoja. Ovaj problem postaje posebno izražen kod složenih i iterativno poboljšavanih programa kada odluke donešene na početku razvoja mogu onemogućiti njegova daljnja poboljšanja. U tim se slučajevima korisnici često odlučuju na pisanje programa ispočetka.



Slika 3.6 Relativni odnos među jezicima po kognitivnim dimenzijama progresivne evaluacije, vidljivosti, viskoznosti, izražajnosti uloga i sekundarne notacije

Progresivna evaluacija

Slično kao kod preuranjenog opredjeljivanja, Kymina mogućnost istovremenog uređivanja i evaluiranja programa značajno olakšava progresivnu evaluaciju ostvarenih patcheva. Nasuprot Kymi, svi ostali jezici nude neki oblik progresivne evaluacije, ali uz veće ili manje prekide toka programiranja. Primjerice, u Pure Dati potrebno je privremeno prebaciti razvojno sučelje u način uređivanja i natrag.

U ovom smislu najgore karakteristike pokazuje Reaktor budući da je kod njega nužno da svi elementi na radnoj površini budu ispravno konfigurirani i povezani žicama. Iako slično vrijedi i za OpenMusic, Pure Datu i Max/MSP, kod njih je to ograničenje nešto blaže implementirano te se i djelomično neispravni programi mogu evaluirati.

Izražajnost uloga

U ovoj dimenziji Reaktor ima značajnu prednost nad ostalim analiziranim jezicima jer svaki objekt na radnoj površini i na različitim razinama apstrakcije svojim izgledom i sučeljem može biti modeliran prema sučeljima poznatima glazbenicima (analogni sintetizatori, miksete). Drugim riječima, elementi u jeziku su vizualno samoopisni. Također, zbog modularnosti pristupa tijekom izgradnje instrumenata u Reaktoru implicirana je i njihova arhetipska, dobro poznata arhitektura.

Nasuprot Reaktoru nalazi se Kyma koja nudi rudimentarne mogućnosti izražajnosti uloga. Svi su blokovi istog izgleda, a razlikovati ih se može samo prema tekstnim oznakama. Takav je pristup posljedica složenosti funkcionalnosti spomenutih blokova i manjak analognih sučelja iz stvarnoga svijeta.

Između Reaktora i Kyme smjestili su se ostali promatrani jezici gdje nešto bolju izražajnost postiže OpenMusic zbog mogućnosti korištenja notnog zapisa u pojedinim dijelovima programa (maquette), dok Pure Data i Max/MSP nude različite samoopisne grafičke elemente kroz korisnička proširenja.

Sekundarna notacija

Dok Pure Data, Max/MSP i OpenMusic nude bogate mogućnosti obilježavanja i komentiranja programa, Kyma i Reaktor ne sadrže gotovo nikakve slične mehanizme. Kod Reaktora je izostanak sekundarne notacije opravdan njegovom izražajnošću uloga gdje su sama sučelja objekata dovoljno opisna i izražajna bez potrebe za vanjskom anotacijom.

Viskoznost

Problemi s viskoznošću zajednički su analiziranim jezicima, ali i vizualnim programskim jezicima generalno. Bez prelaska u treću dimenziju (ili virtualnu stvarnost), takve je probleme u konačnici nemoguće riješiti.

Ipak, kod nekih jezika (Reaktor, Pure Data, Max/MSP) ovaj je problem izraženiji zbog činjenice da promjene na nižim razinama, u podpatchevima, mogu narušiti sučelja i zahtijevati promjene na višim razinama.

S druge strane, kod Kyme je viskoznost smanjena zbog mogućnosti korištenja vremenske crte. U tom su načinu rada kanali međusobno neovisni te se u njima zvukovi mogu kombinirati bez uvođenja novih međuovisnosti i bez da oni izravno utječu jedni na druge.

Vidljivost

Kad je riječ o vidljivosti, svi analizirani jezici dijele slične značajke koje su često boljka i vizualnih programskih jezika općenito. Vidljivost će se, zbog opsežnih mogućnosti enkapsulacije, razlikovati ovisno o složenosti pojedinih programa. Ipak, OpenMusic, Reaktor i Kyma zaslužuju nešto bolju ocjenu u ovoj dimenziji. Oni nude razne mogućnosti za eksponiranje sučelja enkapsuliranih dijelova vizualnog kôda i smanjuju potrebu za eksplicitnom vidljivošću svih segmenata programa.

3.4 Oblikovanje novog jezika na temelju prethodnih saznanja

Saznanja prikupljena u dijelovima istraživanja koji su predstavljeni u poglavljima 3.2 i 3.3 iskorištena su prilikom oblikovanja novog programskog jezika. Cilj novog programskog jezika postavljen je tako da on služi kao općeniti jezik u domeni glazbe bez restrikcija po pitanju specifičnih primjena, od algoritamske kompozicije preko stvaranja audio efekata i sintetizatora zvuka do interaktivnih sustava. Stoga su kao polazišna točka modela novog jezika poslužile paradigme Pure Date, Max/MSP-a te u nešto manjoj mjeri OpenMusic-a. Istovremeno su iz Kyme i Reaktora preuzeti pojedini koncepti, no ne i njihove generalne paradigme koje su ocijenjene kao usko fokusirane na određene primjene i poddomene.

Oblikovani jezik stoga prati općenitu i korisnicima intuitivnu i dobro poznatu paradigmu toka podataka. No, važna inovacija je da podatci u toku nisu audio signali kao u postojećim jezicima, nego se oni zamjenjuju tokom timbralnih atributa. Tom se promjenom u paradigmi riješavaju problemi vezani uz razumljivost i prilagođenost jezika glazbenicima te se povećava bliskost stvarnoj domeni. Dodatno se povećava izražajnost pojedinih blokova te se smanjuje potreba za korištenjem sekundarne notacije.

Nakon što su postavljeni temelji novog jezika, promotreni su rezultati istraživanja postojećih jezika vezanih uz poželjne značajke te načine na koji korisnici otkrivaju, uče i koriste vizualne jezike u glazbi. Na temelju tih saznanja i opservacija donešene su sljedeće najbitnije smjernice prilikom izrade modela:

- Jezik na svim razinama apstrakcije mora biti izgrađen oko iste paradigme toka timbralnih atributa.
- Kako bi se smanjila podložnost pogreškama, jezik mora omogućiti progresivnu i kontinuiranu evaluaciju programa. Analiza postojećih rješenja pokazuje da je upravo ova karakteristika među najpoželjnijima kod Pure Date, Max/MSP-a i Kyme.
- Jezik mora sadržavati mehanizme enkapsulacije i apstrakcije kako bi se mogla graditi hijerarhija programa te kako bi se na najvišoj razini omogućilo jednostavno ostvarivanje najčešćih zadataka. Primjerice, jezik mora sadržavati blok za FM-sintezu koji može ili biti realiziran drugim blokovima niže razine ili ugrađen u jezik. Enkapsulacijom se također omogućuje ponovno korištenje dijelova programa. Primjerice, jedna od prednosti koje ističu Kymini korisnici u odnosu na Pure Datu jest upravo prisutnost ugrađenih objekata koji se ponašaju kao „crne kutije“ i nude razne enkapsulirane funkcionalnosti.
- Grafički simboli pojedinih operacija moraju biti samoopisni kada je to moguće. Dodatno, korisnicima se treba omogućiti biranje simbola koji će predstavljati patcheve na razini roditeljskih patcheva. Ova funkcionalnost čini izražajne i za glazbu ciljane jezike poput Reaktora ili Kyme jednostavnijima za korištenje od, primjerice, Pure Date.
- Veze među blokovima moraju biti fleksibilne i s mogućnošću korištenja zakrivljenih segmenata. Upravo je prilagodba fleksibilnosti veza jedna od prednosti Max/MSP-a nad srodnom Pure Datom.
- Treba omogućiti da podpatchevi predstavljaju svoja unutarnja stanja izravno u svom prikazu u roditeljskom patchu. Kao dodatno poboljšanje, predlaže se omogućavanje korisničkih kontrola izravno nad simbolima na radnoj površini koji se mogu koristiti bez otvaranja novih dijaloga ili prozora korisničkog sučelja. Svi promatrani jezici nude neki oblik ove funkcionalnosti te je stoga nužno zadržati tu dobru praksu.
- Mora se osigurati jednoznačnost imena različitih varijabli, ulaza i izlaza u jeziku kako bi se izbjegli problemi prisutni kod Pure Date i Max/MSP-a

- Sekundarna notacija mora ponuditi dovoljan broj mehanizama poput komentara, bojanja dijelova programa i slično kako bi se korisnicima omogućilo dokumentiranje i anotiranje programa. Ovo je posebno bitna značajka kada se uzmu u obzir složeni, veliki i duboko ugniježđeni programi. Korisnicima Kyme najčešća je zamjerka upravo izostanak bogatih mogućnosti sekundarne notacije.
- Na temelju OpenMusica, predlaže se uvođenje posebne vrste patcha kao što je maquette, koji će eksponirati svoju unutarnju funkcionalnost kroz kontrolnu ploču te prikaz rezultata putem notnog ili sličnog zapisa (primjerice VU-metar u domeni timbralnih atributa).
- Također na temelju OpenMusica koji nudi vizualizacije rezultata grafovima, predlaže se uvođenje elementa za prikaz izlaznog signala, odnosno zvuka, u domeni timbralnih atributa.
- Na temelju Kyme, predlaže se integriranje odvojenog načina rada u jezik, čime bi se omogućilo slaganje programa koji generiraju zvukove na vremenskoj crti. Taj način rada bio bi neovisan o uređivaču programa. Važno je napomenuti da se opisano ponašanje može postići i s drugim jezicima, no to zahtijeva izlazak iz radnog procesa samog jezika i korištenje vanjskih alata.
- Potrebno je osigurati da se, uz korištenje odgovarajućih obrazaca poput paketa ili prostora imena te odgovarajućih eksplicitnih uvoza (eng. *imports*), osigura vidljivost svih vanjskih ovisnosti programa kako bi se izbjegli problemi prisutni u Pure Dati ili Max/MSP-u.
- Jezik treba biti moguće koristiti bez potrebe za vanjskim ili neparadigmatskim alatima tijekom razvoja programa.
- Predlaže se mogućnost ugrađivanja alata za otkrivanje pogrešaka u jezik. Samo definiranje značajki alata van je dosega ovog rada.
- Također se predlaže uvođenje alata za jednostavnije uspoređivanje programa kako bi se olakšalo pronalaženje pogrešaka.
- Svi navedeni elementi moraju biti posebno prilagođeni radu s timbralnim atributima.

Nažalost, neke identificirane probleme poput viskoznosti ili preranog opredjeljivanja nije moguće riješiti zbog ograničenja inherentnih paradigmi kao i tehnoloških ograničenja tradicionalnih dvodimenzionalnih računalnih sučelja. U poglavlju Diskusija dodatno su razrađeni budući smjerovi razvoja vizualnog programiranja koji bi mogli ponuditi rješenja za takve probleme.

Osim tehničkih smjernica korištenih prilikom modeliranja jezika, iz rezultata prethodno spomenutog istraživanja u vezi s učenjem, otkrivanjem i korištenjem vizualnih jezika također proizlaze neke dobre prakse i metamodeli vezani uz pozicioniranje jezika u domeni:

- U slučaju potpune implementacije jezika (izvan faze prototipa), a zbog šireg dosega jezika i bolje prihvaćenosti, predlaže se prebacivanje jezika u domenu otvorenoga kôda.
- Budući da je jezik ciljan za veliki raspon primjena i kombinacija primjena, implementacijom pokaznih programa različitih vrsta (od algoritamske kompozicije do jednostavnih sintetizatora) poboljšava se potencijal približavanja jezika širokom i često raznovrsnom skupu korisnika.
- Kako bi učenje jezika bilo što jednostavnije, predlaže se implementiranje ugrađenog, kontekstno-ovisnog sustava pomoći.
- Postojanje pisanih i video materijala dostupnih putem interneta pozitivno utječe na prihvaćenost jezika te znatno pomaže početnicima prilikom prvog susreta s jezikom.
- Suradnjom s pedagozima i edukatorima moguće je osigurati budući razvoj jezika s aspekata izloženosti što širem krugu korisnika i poboljšanja po pitanju jednostavnosti učenja.
- Suradnjom sa zajednicom glazbenika koji koriste vizualne programske jezike i zajednicom programera zainteresiranih za implementaciju samoga jezika može se omogućiti budući razvoj temelja jezika.

Kombinacijom ovih pristupa ostvaruje se fleksibilan jezik u kojem je moguće implementirati raznovrsne umjetničke ideje u domeni glazbe, a bez gubitka općenitosti. Takav jezik primjeren je i iskusnim korisnicima i početnicima.

3.5 Sinteza zvuka temeljena na toku timbralnih atributa

Kako bi se mogao realizirati jezik temeljen na toku atributa, bilo je potrebno osmisliti mehanizme kojima će se timbralni atributi preslikati u parametre sintetizatora, odnosno način na koji će se realizirati sučelje između postojećih elemenata u obradi audio signala i timbralnih atributa. Zbog nedostatne teorijske i zapisne podrške vezane uz timbar [70] i inherentne složenosti opisanog mapiranja, u sklopu ovog [19][20] i usko povezanih istraživanja [71][72] razvijeni su postupci kojima se (1) pretražuje prostor zvukova na temelju ulaznih timbralnih atributa, (2) timbralni atributi preslikavaju u parametre sinteze pomoću

neizrazite logike i (3) kontroliraju sintetizatori neovisno o postupku sinteze zvuka. U nastavku su objašnjeni glavni elementi tih postupaka i njihov odnos s ostalim dijelovima modela vizualnog programskog jezika.

3.5.1 Timbralni atributi i njihovo preslikavanje

Osnovni detalji timbralnih atributa definirani su u poglavlju 2.2. Kako bi timbralni atributi mogli poslužiti kao temeljna jedinica u toku podataka novog jezika, potrebno je ostvariti njihovu vezu, odnosno preslikavanje, s postojećim i dobro razrađenim konceptima, prvenstveno sa sintetizatorima zvuka i audio signalima. U narednim su poglavljima opisani postupci koji su korišteni kako bi se to preslikavanje i upravljanje sintetizatorom postiglo. Ti su postupci posebno bitni kad je u pitanju ostvarenje prototipa predloženog jezika. Opsežniji i detaljniji opisi predloženih postupaka predstavljeni su u [19].

Preslikavanje timbralnih atributa u parametre sinteze najčešće se ostvaruje tehnikama strojnog učenja. Zadaća strojnog učenja bila je identificirati veze između timbralnih atributa i parametara glazbenog sintetizatora na temelju zadanih primjera. No, takav je zadatak izrazito složen zbog neinjektivnosti preslikavanja koja proizlazi iz činjenice da se određena točka u prostoru timbralnih atributa može postići različitim kombinacijama parametara sinteze. Kako bi se problem pojednostavio, dosad predložena rješenja za preslikavanje atributa u parametre temeljena su na apriornom znanju o parametrima i arhitekturi sintetizatora i time ograničena na rad s jednim određenim sintetizatorom. Stoga su u radovima [20] i [19] predložena tri nova načina preslikavanja timbralnih atributa i upravljanja sintetizatorima: metoda interaktivnih evolucijskih algoritama, metoda izravnog preslikavanja i dvokoračna metoda neovisna o postupku sinteze.

Sve tri metode opisane su u nastavku, a dvije potonje su objašenje podrobnije jer je predloženi vizualni programski jezik oslonjen na njih. Sva tri pristupa detaljno su opisana i evaluirana u prethodno spomenutim radovima [19][20]. U okviru ovog rada, sami postupci nisu zasebno evaluirani već su uklopljeni u predloženi jezik te prezentirani kao njegov integralni dio.

3.5.2 Kontrola sintetizatora pomoću interaktivnih evolucijskih algoritama

Prva predložena metoda za kontrolu sintetizatora temelji se na principima interaktivnih evolucijskih algoritama. Algoritmi su prošireni tako da uzimaju u obzir prethodno definirane atribute. Korištenjem interaktivnog evolucijskog pristupa glazbenici mogu mijenjati i kombinirati dijelove programa dok ne postignu željeni rezultat. Kod primjene genetskih operatora na dijelove programa (eng. *patches*), algoritam pokušava zadržati vrijednosti parametara sinteze relevantne za postizanje željenih karakteristika zvuka. Ovaj pristup omogućuje učinkovito stvaranje novih zvukova koji sadrže značajke opisane ulaznim timbralnim atributima. Postupak za pronalaženje i zadržavanje relevantnih parametara sinteze tijekom interaktivnog istraživanja inovativni je algoritam koji spaja tehnike strojnog učenja (stabla odlučivanja) s metodama evolucijskog računarstva.

3.5.3 Kontrola sintetizatora temeljena na neizrazitoj logici

Metoda izravnog preslikavanja timbralnih atributa u parametre sinteze temelji se na neizrazitoj logici. Znanje o odnosima između atributa i parametara za određene sintetizatore pohranjeno je u obliku pravila AKO-ONDA (eng. *IF-THEN*) koja napredni korisnik sustava može samostalno definirati i prilagođavati. Za svaki atribut koji se može pojaviti na ulazu u sustav pravilima je određeno kako vrijednosti tog atributa trebaju utjecati na jedan ili više parametara sinteze. Na taj je način izbjegnuto višeznačno tumačenje pojedinog atributa. Korisnik koji je napisao pravila u njih je ugradio svoje znanje i iskustvo postizanja pojedinih karakteristika zvuka na jedinstven način. Neizrazita logika je korištenja zbog neinjektivnosti i nelinearnosti preslikavanja.

Za razliku od drugih istraživanja u kojima je ulazni opis zvuka zapravo bila lista pojedinih atributa iz skupa svih raspoloživih atributa (atribut se ili nalazi ili se ne nalazi u listi), u ovom je pristupu odabrano korištenje procjene magnitude verbalnog atributa. Na taj se način postiže veća izražajnost pri zadavanju opisa zvuka, ali i konzistentno korištenje atributa. Ulazni se opis sastoji od svih raspoloživih atributa i njima pridijeljenih vrijednosti magnitude. U slučaju da neki atribut predstavlja karakteristiku koja ne bi trebala biti zastupljena u sintetiziranom zvuku, taj će atribut imati malu vrijednost magnitude, kao što je uobičajeno u sustavima temeljenima na neizrazitoj logici.

Neizraziti model

Neizraziti model za preslikavanje timbralnih atributa u parametre sinteze definiran je u jeziku FCL. Ulaz u model čini 9 timbralnih atributa s vrijednostima iz raspona od 0 do 1, dok se izlazni vektor sastoji od 24 realna broja koji predstavljaju parametre sinteze, a poprimaju vrijednosti iz različitih raspona. Definicija nekoliko izdvojenih ulaznih i izlaznih varijabli u FCL modelu prikazana je sljedećim primjerom:

```
VAR_INPUT
bright   :   REAL; (* RANGE(0 .. 1) *)
warm     :   REAL; (* RANGE(0 .. 1) *)
harsh    :   REAL; (* RANGE(0 .. 1) *)
      ...
END_VAR
VAR_OUTPUT
osc1type     :      REAL; (* RANGE(0 .. 2) *)
      pulsewidth1  :      REAL; (* RANGE(0.02 .. 0.5) *)
      pitchlfo     :      REAL; (* RANGE(0.01 .. 1) *)
      chorus       :      REAL; (* RANGE(0 .. 1) *)
      ...
END_VAR
```

Za svaku ulaznu varijablu definiran je proces „fuzifikacije“, odnosno pretvorbe numeričke varijable u jezičnu pomoću funkcija pripadnosti. Kao vrijednosti jezične varijable za sve timbralne atribute korišteni su kvantifikatori „vrlo malo“, „malo“, „umjereno“, „prominentno“ i „vrlo prominentno“. Sve su funkcije pripadnosti za ulazne varijable trokutastog oblika. Primjer definicije jedne takve trokutaste funkcije za atribut „svijetao“ (eng. *bright*) nalazi se u nastavku:

```
FUZZIFY bright
TERM very_little := (0, 1) (0.1, 1) (0.2, 0);
TERM little := (0.1, 0) (0.3, 1) (0.4, 0);
TERM moderate := (0.3, 0) (0.5, 1) (0.7, 0);
      TERM prominent := (0.5, 0)  (0.7, 1) (0.9, 0);
      TERM very_prominent := (0.7, 0) (0.9, 1) (1, 1);
END_FUZZIFY
```

Za svaku izlaznu varijablu također je definirana funkcija preslikavanja iz jezične varijable u numeričku. Pritom su korištene trapezoidalne funkcije pripadnosti, a kao metoda defuzifikacije odabrana je ona temeljena na centru ravnoteže.

Skup pravila za mapiranje timbralnih atributa u parametre sinteze sadrži ukupno 85 pravila za čije je podešavanje bilo potrebno nekoliko iteracija subjektivnog testiranja od strane autora. Komponenta za neizrazitu logiku može učitati bilo koju datoteku ispravne FCL

sintakse, što čini dodavanje novih ili promjenu postojećih pravila jednostavnim. Potrebno je urediti tekstualnu datoteku i ponovno je učitati u komponentu. Primjeri pravila iz FCL modela nalaze se u nastavku:

```
IF harsh IS little THEN filterlfo_r IS small;
IF hit IS prominent OR hit IS very_prominent THEN volume_s IS very_small;
IF warm IS prominent OR warm IS very_prominent THEN filterfreq_r IS moderate;
IF warm IS prominent OR warm IS very_prominent THEN osc1type_r IS square;
IF woody IS very_prominent OR woody IS prominent OR woody IS moderate THEN
    osc1type IS square;
```

Svako pravilo u neizrazitom modelu, zbog inherentnih ograničenja, odnosi se na samo jedan atribut, što drugim riječima znači da se na lijevoj strani implikacije uvijek nalaze jednostavni uvjeti nad istom lingvističkom varijablom. Tako koncipirani model lakše je razumjeti, održavati, testirati i nadograđivati. Dodavanje novog atributa značilo bi unos novog skupa pravila bez potrebe za modifikacijom postojećih. Također, podjela pravila po atributima omogućila je iterativno podešavanje funkcija pripadnosti za jedan po jedan atribut na način da su se izlazne vrijednosti neizrazitog modela uspoređivale s očekivanim vrijednostima.

3.5.4 Kontrola sintetizatora na temelju atributa neovisna o postupku sinteze zvuka

Inherentna složenost preslikavanja timbralnih atributa u parametre sinteze, kao i nedostatak formalizirane uporabe atributa za opisivanje timbralnih karakteristika zvuka, čine problem automatskog odabira parametara sinteze izazovnim. Dosadašnja srodna istraživanja opisana u poglavlju 2.2 problemu su pristupala kombinacijom strojnog učenja i interaktivnih algoritama te neizrazitom logikom. Navedeni su pristupi s jedne strane pokazali valjane rezultate, no s druge strane i nezanemarive nedostatke, a posebn prilikom ostvarivanja sustava za automatski odabir parametara koji je potpuno neovisan o korištenom sintetizatoru. Problem strojnog učenja bili su nezadovoljavajući rezultati u slučaju prisustva konfliktnih ili nedostatnih podataka u skupu za učenje, dok se neizrazita logika pokazala ograničavajućom za pojedine metode sinteze. Dodatno, za mapiranje putem neizrazite logike bilo je potrebno pohraniti apriorna znanja o korištenom glazbenom sintetizatoru u obliku neizrazitih pravila. Iz dosadašnjih istraživanja može se zaključiti da je izravno preslikavanje timbralnih atributa u parametre glazbenog sintetizatora izrazito složeno, a osobito bez oslanjanja na apriorno znanje o metodi sinteze te o arhitekturi i parametrima korištenog glazbenog sintetizatora.

Kako bi se taj složeni problem pojednostavio, ideja je podijeliti ga na dva koraka. Prvi je korak usmjeren na interpretaciju značenja ulaznih atributa i njihovih vrijednosti, dok je zadaća

drugog koraka pronaći odgovarajuće parametre sinteze. Kao veza između ta dva koraka koriste se audio značajke. One se čine pogodnim posrednikom između timbralnih atributa i parametara sinteze jer s jedne strane mogu kvantitativno reprezentirati atribute, a s druge strane mogu se izlučiti iz sintetiziranog audio signala. Korelacija određenih audio značajki s ljudskom slušnom percepcijom demonstrirana je prethodnim radovima [73]. S druge strane, izračun audio značajki izravno iz sintetiziranog signala omogućuje njihovo korištenje za konstrukciju funkcije dobrote za evolucijski algoritam primijenjen za automatski odabir atributa [74].

Prema tome, zadaća je prvog koraka preslikati ulazne atribute u odgovarajući vektor audio značajki. Primjerice, ako je ulaznim atributom specificirano da sintetizirani zvuk treba biti vrlo svijetao, očekivana vrijednost spektralnog centroida treba biti visoka. U drugom je koraku cilj odabrati parametre sinteze tako da razlika između audio značajki sintetiziranog zvuka i audio značajki generiranih prvim korakom bude što manja. Time je postignuto da se svaki korak rješava zasebno najpogodnijom metodom. Prednost je takvog postupka njegova neovisnost o korištenom glazbenom sintetizatoru.

3.5.5 Prototip kontrole sintetizatora

U sklopu rada [19] prezentirano je rješenje pisano u jeziku Java koje je sadržavalo implementaciju algoritama opisanih u ovom poglavlju. Tijekom inicijalne, izdvojene evaluacije algoritama za preslikavanje timbralnih atributa u parametre sintetizatora, taj je zasebni alat korišten kao prvi korak u kvalitativnom i kvantitativnom vrednovanju rezultata procesa preslikavanja. Riječ je o jednostavnom alatu bez grafičkog sučelja čiji se parametri zadaju putem tekstnih datoteka, a koji na temelju ulaznih timbralnih atributa upravlja različitim sintetizatorima i generira izlazni zvuk.

Algoritmi implementirani u tom alatu poslužili su kao temelj postupka translacije timbralnih atributa u audio signale i natrag koji je korišten prilikom izrade prototipa vizualnog jezika i njegovih sučelja za unos timbralnih atributa i vizualiziranje zvuka pomoću timbralnih atributa.

3.5.6 Atributi i vizualno programiranje

Kao što je ranije opisano, postupci preslikavanja timbralnih atributa u audio signale putem kontrole parametara sintetizatora postaju glavnim elementom novog, predloženog jezika. Ti se postupci koriste i u pojedinim blokovima kao sučelja između korisnika koji definiraju timbralne atribute te nižih slojeva koji obrađuju audio signale, kao što je pokazano

u poglavlju 4.1. Također, timbralni atributi postaju novim sudionicima u kontroli toka i toku podataka te se prenose među pojedinim blokovima tijekom izvođenja vizualnog programa.

U blokovima za vizualizaciju, koji ne generiraju zvuk, postupci preslikavanja koriste se za izlučivanje značajki iz ciljnog zvuka, definiranje njihovih vrijednosti u prostoru timbralnih atributa te njihovo predstavljanje u obliku numeričkih i grafičkih reprezentacija. Ovaj je pristup detaljnije razrađen u poglavlju 4.2.

3.6 Stvaranje glazbe vizualnim programiranjem

Glazbenici i skladatelji stvaranju glazbe pomoću vizualnih programskih jezika pristupaju na različite načine. Uočeni su tako pristupi u kojima se programski jezici koriste samo za definiranje pomoćnih generatora zvukova (primjerice ritam-mašina) koji se zatim koriste kao gotovi elementi prilikom izvedbi uživo, slaganju skladbi u alatima poput digitalnih radnih stanica za obradu zvuka (npr. Ableton) ili kao izvori zvukova za obogaćivanje tradicionalnih muzičkih formi. Dodatno, neki korisnici vizualne programske jezike koriste kao improvizacijske alate kod izvedbi uživo, samostalno ili u spoju s tradicionalnim instrumentima pomoću elektromehaničkih sučelja [75].

Ipak, u kontekstu novog jezika temeljenog na timbralnim atributima, najzanimljiviji su oni pristupi u kojima se cijele kompozicije, odnosno ukupnost glazbe, stvara u samom vizualnom programskom jeziku. Kod takvog oblika stvaranja glazbe, glazbenici i skladatelji koriste gotove elemente i patcheve, ali i vlastite odsječke programa, kako bi njihovim slaganjem i povezivanjem stvorili željena glazbena djela. Pritom primjenjuju obrasce eksperimentiranja i metode pokušaja i promašaja (eng. *trial and error*) kod kojih neprestanim promjenama ili unaprijed pripremljenih ili samostalno isprogramiranih komponenata postižu finu granulaciju rezultata programa, odnosno stvorene glazbe.

Korištenjem timbralnih atributa fokus je stavljen upravo na pojednostavljivanje, poboljšavanje i unaprjeđivanje potonjeg procesa. Praćenjem toka atributa kroz cijeli dijagram vizualnog programa moguće je uočiti načine na koje pojedini blokovi djeluju na generirani zvuk te se tako smanjuje disonanca između zvuka koji glazbenik čuje i njegove reprezentacije diskretnim vrijednostima, objektima i vezama u programu. Drugim riječima, glazbenicima se omogućuje da svoje ideje izravnije pretoče u programski jezik te se smanjuje broj iteracija i modifikacija programa potrebnih za postizanje željenih rezultata.

Iako se temeljni način korištenja novog jezika pritom značajno ne razlikuje od programiranja u, primjerice, OpenMusicu, prisutnost timbralnih atributa može dovesti do

značajno različitih rezultata, često bližih idejama s kojima glazbenici kreću u proces stvaranja glazbe.

3.7 Arhitektura, elementi i temeljne značajke jezika

Predloženi vizualni programski jezik za sintezu zvuka temeljenu na toku timbralnih atributa i računalno stvaranje glazbe temeljeno na vremenskim manipulacijama modelom je i arhitekturom sličan postojećim jezicima u domeni. Tu se ističu Pure Data, Max/MSP i OpenMusic.

Zadržavanjem dobrih praksi i paradigmatskih načela postojećih rješenja postižu se dva cilja jezika. Prvi i najbitniji je taj da novi jezik ponudi značajna poboljšanja, no da istovremeno svojim značajkama i temeljnom paradigmom ostane blizak i razumljiv korisnicima koji već imaju nekog iskustva u području. Naime, povijest razvoja programskih jezika, i tekstnih i vizualnih, pokazuje da su najpoželjnije promjene uvijek evolucijskog, a ne revolucijskog karaktera te se razvoj postiže postupnim nadogradnjama. Drugi cilj je taj da se zadrže dobre i provjerene prakse programiranja koje su uvedene, a zatim destljećima usavršavane u Pure Dati, OpenMusicu i sl. Ti su se jezici pokazali dobro prihvaćenima među glazbenicima, skladateljima i sličnim korisnicima koji ih svakodnevno koriste za mnoštvo različitih namjena. Stoga ih nije potrebno dekonstruirati i iz temelja ponovno osmišljavati, već samo nadopuniti.

U nastavku će biti opisani temeljna arhitektura i elementi jezika. Budući da su kod vizualnih programskih jezika sintaksa i model jezika bliski pojmovi, a uvažavajući izostanak formalnih obrazaca za njihovo opisivanje, sama definicija jezika dana je opisno i primjerima, skiciranjem pojedinih sintaksnih i semantičkih elemenata.

Programi i elementi programa

Na najvišoj razini apstrakcije, u novom se jeziku vizualni programi stvaraju manipulacijom i povezivanjem grafičkih elemenata na radnoj površini. Ti elementi uključuju blokove, temeljne gradivne jedinice vizualnih programa, te spojeve, veze među njima kojima se stvara tok. Vizualni program (ili patch) temeljni je oblik predstavljanja funkcijskog izraza programskog jezika, a sastoji se od blokova povezanih spojevima. Svaki blok sadrži određene funkcionalnosti ili semantički značaj nekog funkcionalnog elementa jezika poput sintetizatora, raznih transformacija itd. Skup međusobno spojenih blokova predstavlja konzistentni, usmjereni aciklički graf.

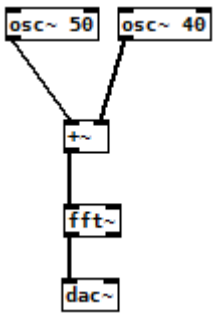
Grafički gledano, svaki se blok sastoji od temeljnog dijela ili tijela simbola te ulaza i izlaza. U tijelu mogu biti izravno ponuđene grafičke kontrole kojima se upravlja funkcionalnostima bloka te ulazima i izlazima.

Svaki blok, neovisno o funkcionalnosti, sadrži ulaze i izlaze jer su oni nužni za povezivanje u program, a podatci koji se njima putem spojeva prenose do drugih blokova su timbralni atributi. Ulazi bloka najčešće će odgovarati argumentima funkcije koja je blokom predstavljena, dok će njegovi izlazi odgovarati povratnim vrijednostima funkcije. Tijekom postavljanja blokova na radnu površinu, korisnik utječe na postavke i funkcije bloka ili izravnim manipuliranjem grafičkih kontrola na grafičkoj reprezentaciji bloka (kad je ona dostupna) ili putem odvojenih prozora/dijaloga (eng. *dialog*) korisničkog sučelja.

Broj ulaza i izlaza ovisi o funkcionalnosti i potrebama pojedinog bloka. Položaj samih ulaza i izlaza u grafičkoj reprezentaciji ovisi o implementaciji jezika te graf jezika može teći s lijeva na desno ili od gore prema dolje. Ipak, na temelju prethodnih istraživanja [3][4] boljim se izborom pokazuje postavljanje ulaza i izlaza na vrh i dno blokova. Takvim se pristupom postiže bolja čitljivost kôda i smanjuje viskoznost.

Osim atomarnih blokova, jezik sadrži blokove u kojima se mogu gnijezditi drugi blokovi. Takav se koncept rada – u kojem se bilo koja funkcionalna jedinica programa može ponovno iskoristiti na drugom mjestu ili na drugoj razini – često naziva slaganjem patcheva, po uzoru na Max koji je bio preteča suvremenih vizualnih programskih jezika u glazbi [26][27].

Funkcionalnosti temeljnih blokova uključenih u jezik bit će opisane u sljedećem poglavlju. Primjer jednostavnog programa koji demonstrira blokove s temeljnim aritmetičkim operacijama prikazuje **Slika 3.7**.



Slika 3.7 Jednostavno zbrajanje i Fourierova transformacija zvuka u prototipu jezika.

Uz blokove i spojeve, jezik uključuje razne druge grafičke elemente poput dijaloga pomoću kojih se upravlja postavkama blokova te popratne alate koji su integrirani u jezik, ali nisu njegov temeljni dio. Predviđena je i proširivost jezika kroz korisničke dodatke. Kako bi

se poboljšala sekundarna notacija (jedna od kognitivnih dimenzija opisanih u poglavlju 3.3), vrijedi razmotriti i mogućnost označavanja blokova i spojeva raznim bojama.

Izvršavanje programa

Predloženi je jezik interpretiranoga tipa što znači da se ostvareni programi izvršavaju u stvarnom vremenu te ih je moguće dinamički mijenjati bez potrebe za prevođenjem u međukôd.

Tijekom izvršavanja programa, funkcije specificirane u blokovima, a koje mogu biti i vrlo jednostavne i vrlo složene, izvode se redoslijedom koji je definiran vezama među njima te se istovremeno generiraju izlazi. Drugim riječima, izvodi se prateći graf vizualnog programa, a rekurzivne funkcije izvode se od dna prema vrhu. Izvođenje programa najčešće je kontinuirano, no to ovisi o samim funkcionalnostima koje su uključene u pojedini program. Primjerice, program koji na početku ima sintetizator zvuka čiji se izlaz dalje obrađuje moguće je izvoditi bez prekida dok se drugi programi mogu izvesti jednom ili nekoliko puta ovisno o grafu blokova, zadanim parametrima i elementima kontrole toka.

Kao što je spomenuto ranije, svaki blok semantički odgovara nekoj funkcionalnosti jezika koja može biti različitog tipa, od kontrolne poput petlji, preko aritmetičke do složene poput sintetizatora i generatora zvuka. Prilikom izvođenja programa, funkcija sadržana u bloku počinje se izvršavati tako što se poziva enkapsulirana funkcionalnost. Važno je napomenuti da zbog toga što blokovi prikazuju informacije tijekom samog izvođenja, nije moguće ostvariti potpunu odvojenost prezentacije od nižih slojeva.

Istovremeno, tijekom izvođenja programa, spojevi – zapravo bridovi usmjerenog grafa – definiraju kako se podatci, odnosno timbralni atributi prenose od jednog ka drugim blokovima. Drugim riječima, spojevi definiraju funkcijsku kompoziciju programa. Spajanje izlaza jednog bloka na ulaz drugog bloka ekvivalentno je kompoziciji funkcija gdje se rezultati jedne funkcije koriste kao parametri sljedeće.

Tijekom izvođenja bloka, odnosno tijekom evaluacije njegove funkcije, vrijednosti ulaza se prikupljaju prije samog ulaska u tijelo funkcije. Ulazi su definirani vrijednošću i spojem s prethodnim blokom čije izvođenje mora unaprijed dovršiti trenutni ciklus i pripremiti nove podatke za daljnje korištenje. Ako ulaz nije spojen na izlaz nekog drugog bloka, pretpostavlja se da vrijednost ulaznih timbralnih atributa nije postavljena. Takva situacija može biti dozvoljena ili ne, ovisno o tome kako je pojedini blok koncipiran. Ove značajke čine ulaze i izlaze bloka ključnim elementima u kontroli toka. Oni pokreću evaluaciju na drugim blokovima koji zatim dohvaćaju rezultate evaluacije prethodnog bloka. Tako dohvaćene

vrijednosti timbralnih atributa postavljene na izlazu bloka mogu postati ulazni argumenti funkcije sljedećeg bloka.

Blokovi također mogu sadržavati logiku za odgađanje evaluacije na temelju nekih vanjskih parametara. To može biti posebno korisno u situacijama kad se čekaju podatci koji dolaze iz nekog vanjskog, neprogramskog izvora. Primjerice, ako pretpostavimo postojanje bloka za interakciju s uslugama na webu (eng. *web service*), možemo pretpostaviti da bi takav blok odgodio svoje izvođenje do trenutka uspješnog dohvaćanja podataka.

Budući da se izvođenje jezika temelji na propagaciji podataka i kontrole toka, a s obzirom na poželjnu značajku stvarnovremene evaluacije izvođenja programa, kao što je pokazano u poglavljima 2.1.1 i 3.3, moguće je ostvariti progresivno ispitivanje ispravnosti programa. Drugim riječima, moguće je zaustaviti izvođenje u trenutku kad neki određeni blok detektira nekonzistentnost u ulaznim podatcima ili neki drugi oblik nepravilnosti.

Temeljem analize opisane u poglavlju 3.2.1, ustanovljeno je da je smanjivanje broja pogrešaka za korisnike bitan aspekt rada s vizualnim programskim jezicima. Stoga je potrebno osigurati da jezik sam detektira, prijavljuje i sprječava sintaktičke pogreške prije i tijekom izvođenja programa. Tu se posebno ističu nedozvoljeni slučajevi programskoga toka, primjerice kružnih poziva ili povezivanja višestrukih izlaza na isti ulaz, ili postavljanje nedozvoljenih vrijednosti parametara pojedinih blokova.

Dodatni elementi jezika

Rezultati istraživanja prikazani u 3.2.1 i 3.2.2 pokazuju da su korisnicima iznimno bitni određeni elementi jezika koji nisu nužno izravno vezani uz osnovnu funkcionalnost, ali su ipak dio jezika. Stoga predloženi jezik sadržava mogućnost ostavlja komentara, odnosno anotiranja blokova, spojeva i čitavih programa. Unešeni komentari se prikazuju različito, ovisno o elementu koji se pregledava, ali može se ostvariti zasebnim prozorima i dijalozima ili malim oblačićima s informacijama (eng. *tooltip*) koji se pojavljuju prilikom prelaska miša preko elementa.

Nadalje, definirani su posebni moduli za grafički unos timbralnih atributa koji će biti prezentiran u poglavlju 4.1 te modul za vizualizaciju rezultata programa, odnosno sinteze zvuka koji će biti prezentiran u poglavlju 4.2.

Konačno, kako bi se olakšalo praćenje promjena u programima te omogućila usporedba programa, predlaže se vanjski alat za usporedbu programa, a koji se temelji na preslikavanju unutarnje strukture programa u tekstni oblik i natrag. Preslikavanje je oslonjeno na svođenje usmjerenog grafa programa, svih pojedinosti o postavkama blokova i spojeva te njihovih

položaja u dvodimenzionalnom prostoru radne površine u odgovarajuću tekstnu reprezentaciju. Modul i pripadajući postupci opisani su u poglavlju 5.

3.7.1 Blokovi jezika

Skup blokova, odnosno objekata, koje nudi jezik praktički je neograničen budući da je predviđena podrška za korisničke blokove i nadogradnje. U načelu, zadržani su korisnicima poznati koncepti blokova i funkcionalnosti prisutnih u jezicima poput Pure Date i OpenMusica, ali uz njihovu prilagodbu za rad s timbralnim atributima. Slijede primjeri kategorija blokova uz opis njihovih funkcionalnosti:

- temeljni blokovi
    - nude funkcionalnosti poput aritmetičkih i kombinatoričkih operacija, različitih krivuljnih funkcija, objekata za ulaz i izlaz u tekstne datoteke itd.
- blokovi za obradu zvuka
    - najvažniji objekt u ovoj skupini jest generički objekt koji predstavlja sintetizator, a koji svojom implementacijom može nuditi različite vrste sintetizatora, od FM do granularne sinteze, a njime se upravlja putem timbralnih atributa
    - osim sintetizatora, očekuje se da jezik nudi različite druge metode obrade zvuka poput normalizacije glasnoće, generiranja vibrata i šuma, promjene frekvencije uzorkovanja itd.
- blokovi za ulaz i izlaz zvuka
    - na ulazu, jezik može procesirati različite zapise zvuka i glazbe (primjerice u formatu mp3-ja)
    - na izlazu, jezik može zapisivati rezultate izvođenja, odnosno rezultantni zvuk i glazbu, u različite zapise (primjerice u formatu FLAC-a)
    - blokovi za analizu i vizualiziranje različitih značajki zvuka s posebnim naglaskom na timbralne atribute
    - ponuđeni su i alati za izlaz zvuka putem sklopovlja računala, odnosno različitih zvučnih kartica i audio-sučelja
- blokovi s matematičkim funkcijama
    - sadrži složenije matematičke postupke poput Fourierovih transformacija, Markovljevih lanaca, Kroneckerove delta funkcije itd.
- pomoćni blokovi

- ponuđeni su blokovi za manipulaciju formata MIDI, integraciju s drugim alatima putem signala OSC itd.

Osim blokova koji su ponuđeni kao dio temeljnog modela, arhitektura novog jezika predviđa korisnička proširenja i vanjske blokove koji mogu nuditi različite funkcionalnosti, od složenih izračuna i transformacija do bogatih vizualizacija podataka.

Razvijateljima vanjskih blokova nužno je, kroz odgovarajuća sučelja, ponuditi pristup unutarnjim strukturama jezika kako bi se njihova integracija učinila što lakšom i prirodnijom (eng. *seamless*). Dodatno, u punoj implementaciji jezika preporuča se uključivanje alata za upravljanje vanjskim paketima i dodatcima (eng. *packet manager*) kako bi se korisnicima olakšala razmjena i uključivanje raznih proširenja.

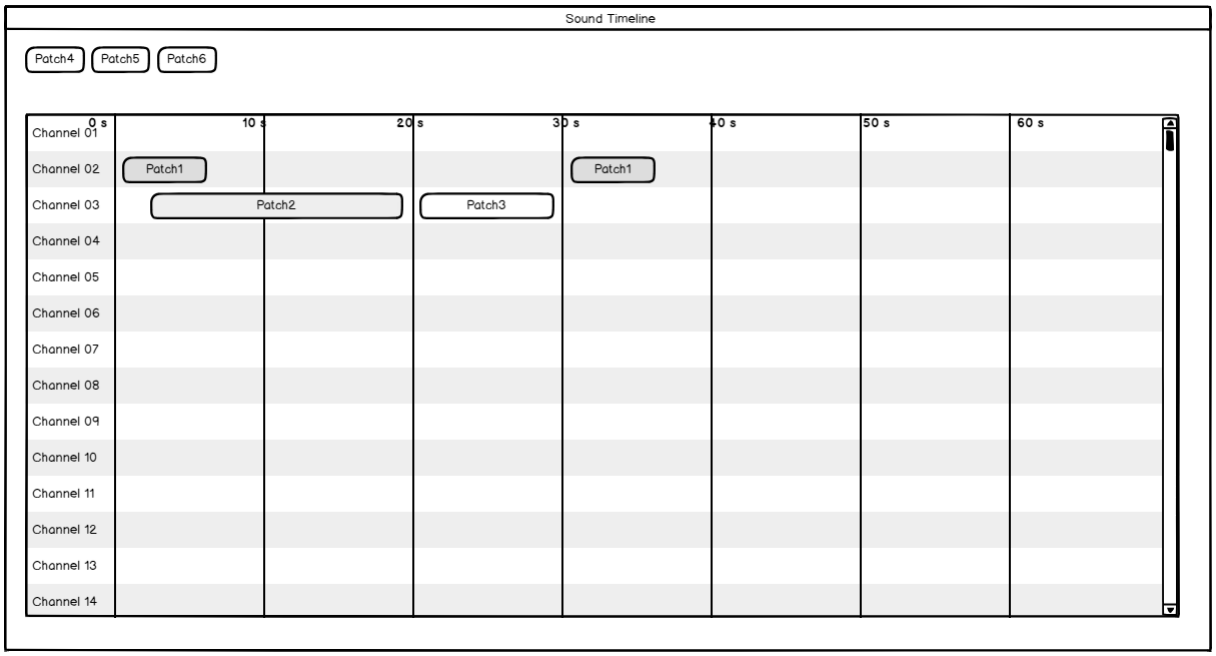
3.7.2 Vremenska crta

Budući da se novi jezik temelji na paradigmi koja predviđa razne blokove za opće programerske funkcije poput petlji te blokove za ulančavanje zvukova, podrazumijeva se da se sintetizirane ili obrađene zvukove može nizati i njima manipulirati u vremenskoj dimenziji. Time se od pojedinačnih zvukova mogu graditi cjelovita glazbena djela. Takva se funkcionalnost u novom jeziku, kao primjerice u Pure Dati, postiže korištenjem brojača, polja podataka i sl.

No, kako bi se stvaranje glazbe u kontekstu novog jezika učinilo jednostavnijim i praktičnijim, predlaže se uvođenje alata za rad s vremenskom crtom po uzoru na Kymu, OpenMusic (maquette) i digitalne radne stanice za obradu zvuka poput Audacityja. Taj alat postoji izvan uobičajenih radnih površina patcheva te je u njemu moguće slagati, pomicati, proširivati i sužavati zvukove i slijedove zvukova koje generiraju pojedini patchevi.

Svaki od segmenata na vremenskoj crti predstavlja jedan patch u novom jeziku i te segmente je moguće proizvoljno razmještati među kanalima te zatim pomicati unutar pojedinih kanala. Postavljanje dva patcha u paralelne kanale, a tako da se preklapaju u vremenskoj dimenziji, rezultirat će istovremenim izvođenjem ta dva patcha. Postavljanje dva patcha u isti kanal, jedan iza drugoga u vremenskoj dimenziji, uzrokuje slijedno izvođenje patcheva. Proširivanjem segmenta koji predstavlja patch u vremenskoj dimenziji postiže se dulje trajanje njegova izvođenja.

Tako će se u primjeru skiciranom na slici (**Slika 3.8**) prvo početi izvoditi *Patch1*, pri kraju njegova trajanja istovremeno će krenuti izvedba *Patch2*, a nakon njegova završetka pokrenut će se *Patch3*. Na slici se također može primijetiti da će trajanje *Patch2* biti znatno dulje od ostalih patcheva.



Slika 3.8 Skica vremenske crte za slaganje zvukova/patcheva

Kao dodatno poboljšanje predlaže se da se na pojedinim grafičkim elementima koji predstavljaju patcheve na vremenskoj crti omogući izravna manipulacija njihovim timbralnim atributima. U tom bi se načinu rada unutar samih patcheva mapirale kontrole timbralnih atributa prema istovjetnim kontrolama na roditeljskom patchu ili vremenskoj crti (slično kao kod funkcionalnosti „crtanja na roditelju“ kod Pure Date).

3.7.3 Tok atributa

U predloženom modelu jezika pretpostavlja se da se tok podataka u potpunosti oslanja na timbralne atribute. U tom smislu svi blokovi međusobno komuniciraju razmjenom vrijednosti timbralnih atributa, a sam audio signal generira se samo u blokovima za izlaz zvuka.

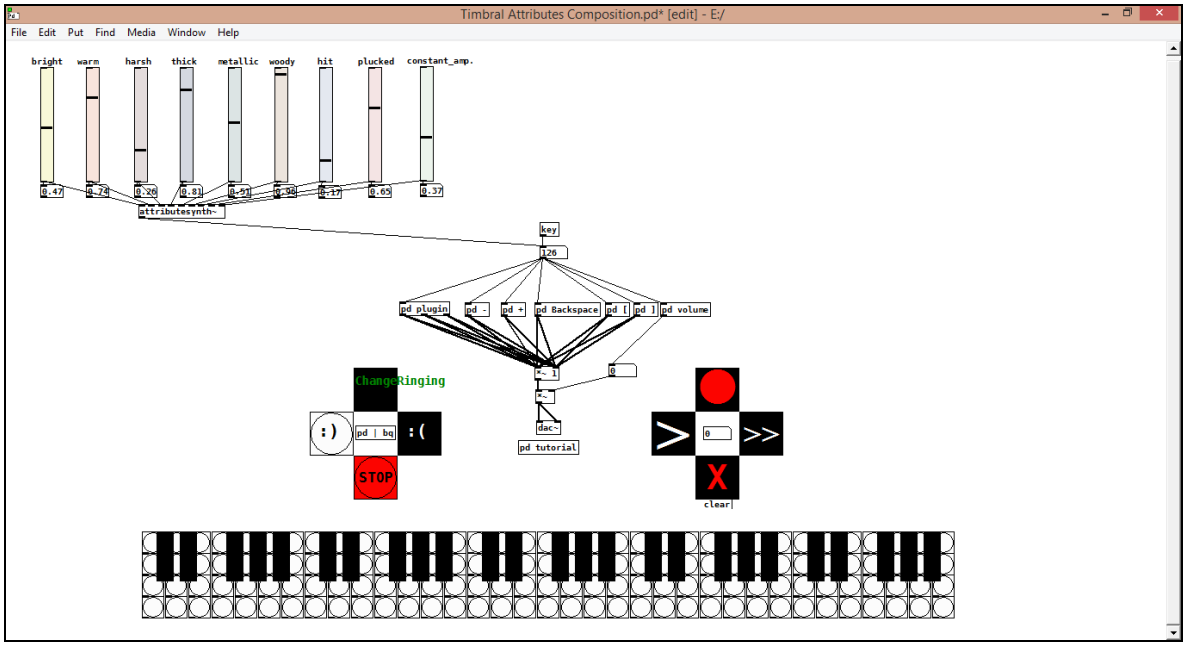
Kao alternativno rješenje koje bi omogućilo istovremenu manipulaciju audio signalima i korištenje timbralnih atributa, predlaže se da jezik istovremeno među blokovima prenosi i timbralne atribute i odgovarajuće vrijednosti sintetiziranog audio signala. Drugim riječima, bilo koji blok koji čini promjene nad tokom timbralnih atributa, nužno mijenja i tok audio signala prema vrijednostima timbralnih atributa. Odnosno, svaki blok iznova generira audio signal na temelju novih vrijednosti atributa.

Iako je takav pristup značajno računalno intenzivniji i redundantan, njime je omogućeno da se u jezik uključe i tradicionalni tipovi blokova koji obrađuju signale. Takav pristup čini jezik fleksibilnijim i otvorenijim prema spajanju s vanjskim alatima i drugim jezicima.

3.8 Prototip

Puna implementacija predloženog jezika tražila bi višegodišnji rad nekolicine programera te bi uključivala ostvarivanje svih funkcionalnosti iz temelja. Stoga smo, kako bi dokazali ostvarivost predloženog jezika, napravili prototip (temeljen na platformi postojećeg jezika) čija je svrha demonstriranje najbitnijih poboljšanja i doprinosa novog jezika. Prototip je također poslužio za validiranje i evaluaciju predloženog modela, arhitekture i poboljšanja jezika.

Budući da su najznačajniji doprinosi predloženog jezika u njegovom korištenju toka timbralnih atributa, a da je u drugim segmentima paradigme sličan jezicima Pure Data, Max/MSP i OpenMusic, prototip jezika implementiran je kroz unutarnja (intervencijama u izvornom kôdu) i vanjska proširenja Pure Date (ostvarivanjem dodataka, patcheva i pluginova u jeziku). Iako je svojom paradigmom i određenim proširenjima jezik bliži Max/MSP-u, Max/MSP je jezik komercijalne prirode i zatvorenoga kôda te nije mogao poslužiti za izradu prototipa. Istovremeno, fokus OpenMusica je na specifičnim ulogama vizualnog programiranja u skladanju i algoritamskoj kompoziciji te bi zahtijevao veći broj intervencija kako bi ga se prilagodilo predloženoj paradigmi toka timbralnih atributa. Slika 3.9 prikazuje pokazni program za generiranje glazbe temeljeno na Markovljevim lancima i prošireno s timbralnim atributima u prototipu jezika.



Slika 3.9 Pokazni program u prototipu jezika na operacijskom sustavu Microsoft Windows 8.1

3.8.1 Pure Data

Kao što je prethodno opisano u poglavljima 2.1.1 i **Error! Reference source not found.**, Pure Data je alat otvorenog kôda koji je razvio Miller Puckette tijekom devedesetih godina prošloga stoljeća [25]. Pure Data je namijenjen stvaranju računalne glazbe te interaktivnih i multimedijskih radova. Zbog toga je osnovna paradigma programiranja temeljena na toku podataka koja je bliska domeni obrade i sinteze audio signala. Također, Pure Data sadrži velik broj komponenata za digitalnu obradu signala i integraciju putem tipičnih protokola za kontrolu digitalnih glazbenih instrumenata. Primjenom postojećih komponenata korisnici mogu stvarati vlastite sintetizatore, audio efekte i interaktivne sustave, ali i programe šire primjene.

3.8.2 Audio signali i timbralni atributi

Budući da je Pure Data u svojoj jezgri orijentirana na obradbu audio signala i budući da joj se tok podataka temelji na njihovu prijenosu, u prototipu se audio signali i timbralni atributi istovremeno prenose među blokovima. No, za razliku od pristupa opisanog u 3.7.3, za potrebe prototipa nije nužno sve blokove prilagoditi za rad s timbralnim atributima.

Stoga je jedini zahtjev taj da svi blokovi, odnosno objekti, prilagođeni za rad s timbralnim atributima istovremeno generiraju i audio signale kako bi se mogli spajati s „običnim“ blokovima. Istovremeno se interne strukture koje čine temelj svih blokova u Pure Dati proširuju tako da prosljeđuju timbralne atribute čak i ako ih nisu izravno modificirali.

Nedostatak ovakve implementacije je što u pojedinim slučajevima može doći do privremenog nesrazmjera vrijednosti audio signala i timbralnih atributa, no budući da će se vrijednosti timbralnih atributa po potrebi ponovno izračunati na sljedećem bloku koji podržavaju rad s njima, taj je problem zanemariv u kontekstu prototipa.

S druge strane, prototip ostvaren na ovaj način prenosi i timbralne atribute i audio signale što više odgovara alternativnom prijedlogu iz 3.7.3, koji bi omogućio veću fleksibilnost u upotrebi jezika.

3.8.3 Implementacija prototipa

Implementacija prototipa u Pure Dati temelji se na tri komponente koje su ili stvorene za potrebe prototipa ili dobivene modifikacijom i nadopunom postojećeg kôda:

1. Jedan od temeljnih blokova za sintetiziranje, odnosno generiranje zvuka prerađen je tako da sučelje u kojemu se upravlja parametrima sintetizatora omogućuje unošenje vrijednosti timbralnih atributa. Korištenjem postupaka mapiranja opisanih u 3.5, te se

vrijednosti tijekom izvođenja programa prevode u vrijednosti koje su razumljive izvornoj implementaciji bloka u Pure Dati. U općenitom bi slučaju bilo potrebno sve postojeće blokove Pure Date modificirati na opisani način, no za potrebe evaluacije prototipa izabran je samo najreprezentabilniji među njima.

2. Dio kôda koji nadgleda tok podataka i upravlja prijenosom podataka među blokovima, uključujući njihove ulaze i izlaze, nadograđen je tako da uz originalni audio signal prenosi i vrijednosti timbralnih atributa. To znači da bez obzira što u prototipu, kao što je naglašeno u prethodnoj točci, svi blokovi nisu prilagođeni za rad s timbralnim atributima, njihovo uključivanje u program neće uzrokovati gubitak semantičkih informacija o atributima.
3. Korištenjem uobičajenih proširenja jezika (eng. *plugin*), objekata, biblioteka i dodatnih patcheva ostvarene su potrebne funkcionalnosti za mapiranje timbralnih atributa koje se u pojedinim slučajevima oslanjaju na vanjske biblioteke poput komponente za neizrazitu logiku.

Priprema blokova za rad s timbralnim atributima

Za potrebe prototipa, stvoren je novi blok za sintezu zvuka koji je na nekoliko načina prilagođen radu s timbralnim atributa. Za razliku od originalnog bloka za sintezu zvuka temeljenu na matematičkim parametrima sintetizatora, ulazno sučelje novog bloka temelji se na timbralnim atributima. Oni se definiraju pomoću apsolutnih vrijednosti pojedinih atributa na skali od 0 do 100. Na izlaznoj strani, pripremljene su varijacije blokova poput objekta *VU meter* i *spectrum analyzer* tako da oni prikazuju vrijednosti timbralnih atributa u generiranom zvuku umjesto frekvencije i amplitude signala u temeljnim inačicama.

Kako bi se ostvarila ova funkcionalnost, u implementaciju bloka dodan je sloj, koji pomoću algoritama opisanih u 3.5 i pomoću komponente za neizrazitu logiku, transformira timbralne atribute u parametre ciljnog sintetizatora audio signala.

Ovaj je model i obrazac translacije atributa u parametre moguće prilagoditi i primijeniti na svim ostalim blokovima, po potrebi u smjeru timbralnih atributa prema audio signalima ili od audio signala prema timbralnim atributima.

Prijenos timbralnih atributa

Kao što je spomenuto u prethodnom potpoglavlju, u prototipu se tok podataka istovremeno temelji i na timbralnim atributima i na audio signalima. S obzirom na arhitekturu Pure Date koja je fokusirana na audio signale, timbralni atributi su sekundarni podatci u toku te se prenose uz glavne podatke, audio signale. To znači da je u prototipu novog jezika moguće

koristiti sve objekte i mogućnosti Pure Date, neovisno o tome jesu li isti prilagođeni za rad s timbralnim atributima.

Implementacijski gledano, prijenos timbralnih atributa ostvaren je zahvatima u jezgri Pure Date gdje su sve funkcije, metode i razredi koji se koriste za prijenos podataka prošireni tako da primaju dodatne parametre, odnosno iznose timbralnih atributa.

Ovakav pristup u praksi donosi usporenje izvođenja u stvarnom vremenu te može prouzročiti pojavu latencija.

Vanjska komponenta za neizrazitu logiku

S obzirom da standardni paket programskog jezika Pure Data ne sadrži komponentu s funkcionalnošću neizrazite logike, bilo je potrebno koristiti vanjsku komponentu čija je uloga bila da prihvaća listu timbralnih atributa s pridijeljenim magnitudama te korištenjem neizrazitog modela pohranjenog u datoteci generira vrijednosti parametara glazbenog sintetizatora. Implementacija opisane funkcionalnosti spominje se u radu Cádiza i Kendalla [76], a oslanja se na biblioteku libfuzzy za programski jezik C koja koristi Fuzzy Inference System (FIS) notaciju za pripremu neizrazitih modela. No, spomenuta komponenta nije bila dostupna za ponovno korištenje pa je za potrebe ovog rada razvijena nova komponenta nazvana fuzzyext.

Kao jezik za definiranje neizrazitog modela odabran je Fuzzy Control Language (FCL) jer je definiran standardom Međunarodne elektrotehničke komisije (IEC 61131-7), a odlikuje se jednostavnom sintaksom. Za razliku od FIS notacije kod koje se prilikom definiranja pravila moraju koristiti indeksi varijabli, FCL notacija koristi standardni IF-THEN oblik pravila. Zahvaljujući tome, FIS modele mogu nakon relativno kratke faze učenja koristiti tipični korisnici koji se prije nisu susretali s tekstnim programiranjem.

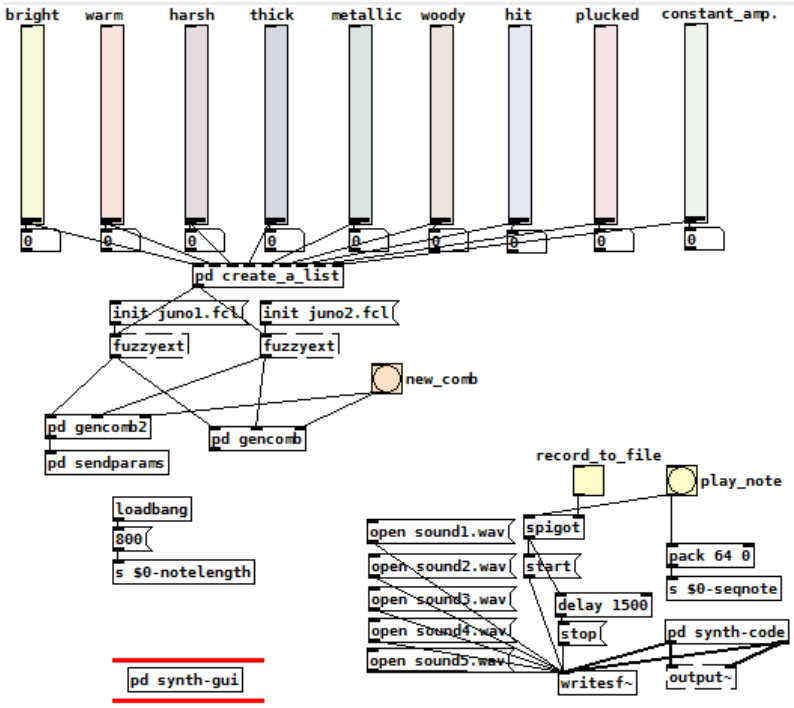
Vanjske komponente za vizualni jezik Pure Data moguće je razviti isključivo u programskom jeziku C pa su iz tog razloga razmotrene brojne postojeće biblioteke za neizrazitu logiku napisane za C i C++. No, konačni izbor sveo se na jFuzzyLogic biblioteku za programski jezik Java jer ostale razmotrene biblioteke nisu podržavale FCL notaciju niti dodatne mogućnosti vizualizacije i evaluacije neizrazitih modela [77].

Implementacija vanjske komponente temeljena na jFuzzyLogic biblioteci zahtijevala je povezivanje biblioteke za programski jezik Java s kôdom napisanim u jeziku C. Zbog toga su razvijene posebne funkcije koje omogućuju komunikaciju između izvornog C programa i Java virtualnog stroja putem radnog okvira Java Native Interface [78]. Funkcije su poslužile kao vezivni kôd između osnovnih funkcionalnosti komponente napisanih u C-u i biblioteke za

neizrazitu logiku jFuzzyLogic. Kako bi osigurale pouzdanu i efikasnu integraciju, funkcije također imaju riješeno rukovanje pogreškama, kao i mehanizam privremene pohrane podataka (eng. *caching*). Tim se mehanizmom reducira broj ponovnih poziva i kalkulacija za vrijednosti ulaza koje su se nedavno pojavile, što posljedično povećava brzinu rada. Privremena pohrana u memoriju ostvarena je pomoću biblioteke sglib [79].

Objekt za sintezu zvuka

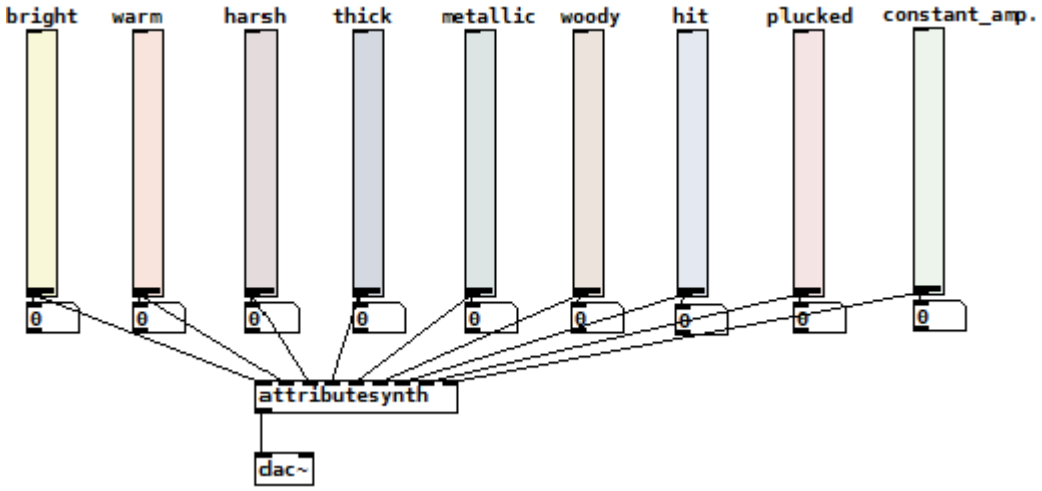
Najznačajniji dio prototipa jest objekt za sintezu zvuka. Taj je objekt u sklopu istraživanja implementiran na dva načina.



Slika 3.10 Objekt za sintezu zvuka na temelju ulaznih atributa implementiran elementima jezika Pure Data

Prva izvedba objekta za sintezu zvuka zapravo je u potpunosti ostvarena ugrađenim mogućnostima jezika Pure Data (**Slika 3.10**). Drugim riječima, objekt je ostvaren programiranjem u Pure Dati i slaganjem postojećih elemenata jezika u patch kako bi se postigla izvedba različitih vrsta sintetizatora (implementirane su FM i granularna sinteza). Pritom su razvijeni vanjski dodatci poput prethodno spomenute komponente za neizrazitu logiku kako bi se mapirale vrijednosti timbralnih atributa u interne parametre izrađenih sintetizatora. Tako implementirani patch pretvoren je u podpatch te ga je, zahvaljujući svojstvima enkapsulacije, moguće koristiti iz bilo kojeg drugog patcha njegovim jednostavnim uključivanjem u tok signala.

Druga izvedba oslanja se na implementaciju istih tih algoritama prezentiranih u 3.5.3, ali tako da su oni izvedeni u jeziku C intervencijama nad jezgrom Pure Date (**Slika 3.11**). Tako je stvoren novi objekt *attributesynth~* koji na ulazu prima vrijednosti timbralnih atributa, a na izlazu generira audio signal prema tim vrijednostima. Objekt se koristi kao i svi drugi slični objekti u jeziku, primjerice poput objekta *osc~*.



Slika 3.11 Puna izvedba sintetizatora kojim se upravlja timbralnim atributima u jezgri jezika

3.9 Evaluacija jezika

Predloženi je jezik evaluiran slično kao postojeća rješenja opisana u poglavljima 3.2 i 3.3. Kombinacijom subjektivnih intervjua, kvantitativnom i kvalitativnom analizom statističkih rezultata ankete te formalnom evaluacijom pomoću okosnice kognitivnih dimenzija, ispitana je uspješnost novog jezika. U nastavku je dan detaljan pregled pojedinih koraka evaluacije.

3.9.1 Intervjuiranje eksperata

U prvom su koraku provedeni intervjui sa šestero stručnjaka, glazbenika i istraživača u području računalom potpomognutog stvaranja glazbe. Ti su stručnjaci izabrani tako da je svaki od njih bio u doticaju s međusobno neovisnim tehnologijama te se njihova područja ekspertize nisu preklapala. Skup intervjuiranih korisnika uključivao je profesore s akademije, profesionalne glazbenike, istraživače u području improviziranog programiranja (eng. *live coding*) i multimedijske umjetnike. Tijekom intervjua stručnjacima je prezentiran prototip programskog jezika te su istaknute najbitnije razlike u odnosu na njima poznate jezike. Stručnjaci su pritom nudili komentare i opaske te tražili pojašnjenja pojedinih mehanizama. Primjerice, na temelju povratnih informacija ove faze izmijenjen je osnovni skup timbralnih atributa te su jasnije istaknute uloge pojedinih elemenata sučelja.

Ova je faza ispitivanja jezika također poslužila za kreiranje kvalitativnih pitanja koja su činila anketu korištenu u sljedećoj fazi.

3.9.2 Anketa

Kao što je spomenuto u prethodnom poglavlju, pitanja za anketu kojom je evaluiran jezik djelomično su oblikovana u skladu s dojmovima i najbitnijim opaskama prikupljenima iz intervjua sa stručnjacima. Posebna je pozornost usmjerena na ona područja u kojima se predloženi jezik značajno razlikuje od postojećih poput Pure Date i Max/MSP-a te je dodatni naglasak stavljen na evaluaciju i validaciju korištena toka timbralnih atributa.

S obzirom na tip evaluacije i manjak objektivnih, kvantitavnih mjerila za ispitivanje novog vizualnog programskog jezika, pitanja su većim dijelom bila kvalitativnog tipa. Primjerice, upitnik je sadržavao zatvorena pitanja poput "Smatrate li da korištenje timbralnih atributa poboljšava razumljivost patcheva?" s ponuđenim brojčanim odgovorima od 1 do 10 gdje vrijednost 1 odgovara ocjeni "nimalo", a 10 "značajno". Osim zatvorenih pitanja, upitnik je sadržavao pitanja otvorenog tipa poput "Koje prednosti novog jezika uočavate?" ili "Koje nedostatke novog jezika uočavate?". Dodatno, u upitnik su uključena pitanja o demografiji i iskustvu s drugim jezicima. Zbog sažimanja opsega istraživanja, pripremljen je samo jedan upitnik te nisu odvojeno razmatrani odnosi pojedinih jezika (Pure Date, Max/MSP-a itd.) s novim jezikom, već samo generalna pozicija novog jezika u domeni.

Cilj upitnika bio je dvojak. S jedne strane ispitivanjem stavova iskusnih korisnika poboljšati predloženi model, a s druge na širem skupu korisnika validirati predložene ideje. Stoga je anketa provođena u dva kruga. U prvom su krugu anketirani samo vrlo iskusni korisnici. Iako su rezultati takve ankete podložni predrasudama pri odabiru (eng. *selection bias*) i time kontaminirani, takav je pristup u ovom slučaju opravdan spomenutim ciljem izlučivanja mogućih poboljšanja. Anketu je u prvom koraku ispunilo 17 iskusnih korisnika vizualnih programskih jezika u glazbi.

U drugom su koraku ankete postavljene u iste diseminacijske kanale kao u 3.2 (internetski forumi, društvene mreže, zajednice glazbenika i sl.) te je anketiran raznolik skup korisnika. Cilj je drugog koraka bio statistikama i provjerljivim podatcima potkrijepiti valjanost predloženog modela. Anketu je u drugom koraku ispunilo 77 korisnika različitih profila, od početnika do visoko obrazovanih glazbenika.

Kako bi se anketiranim osobama omogućio uvid u jezik, uz ankete je distribuiran i niz kratkih video zapisa koji su demonstrirali korištenje jezika u uobičajenim situacijama poput

izrade jednostavnog sintetizatora. Video zapisi popraćeni su kratkim tekstovima o načinu rada s jezikom i njegovim bitnim značajkama.

Rezultati anketa u konačnici su analizirani IBM-ovim sustavom za statističku analizu SPSS u onim segmentima koji su se na taj način mogli analizirati. Pitanja otvorenog tipa pritom su klasificirana u različite skupine, statistički obrađena, ali i pojedinačno razmotrena. Analiza rezultata slijedi u nastavku poglavlja.

Rezultati i zaključci

Tijekom intervjuiranja eksperata te analizom rezultata anketa isključivo naprednih korisnika izlučeni su zaključci koji validiraju i opravdavaju temeljne postavke novostvorenog jezika.

Uvođenje timbralnih atributa većina intervjuiranih i anketiranih korisnika subjektivno smatra značajnim unaprjeđenjem. Posebno se u tom smislu ističe mogućnost zanemarivanja detalja niske razine audio signala uz zadržavanje mogućnosti manipulacija i u toj domeni. Osim uvođenja timbralnih atributa, intervjuirani eksperti smatraju poželjnim zadržavanje poznatih paradigmi iz jezika poput Pure Date ili Max/MSP-a. Smatraju da na taj način mogu svoja prethodna znanja na jednostavan način primijeniti i u novom jeziku čime se značajno smanjuje period učenja jezika.

Po pitanju utjecaja na kreativnost, većina eksperata smatra da će utjecaj biti blago pozitivan, ponajviše zahvaljujući činjenici da se kod kreativnog procesa oslobađa dio kognitivnog tereta kojega uobičajeno donose manipulacije audio signalima. Nekolicina intervjuiranih izrazila je skepsu po pitanju utjecaja na kreativnost te izrazila mišljenje da složenost rada s audio signalima često dovodi do neočekivanih, no zanimljivih rezultata te da se uvođenjem timbralnih atributa gubi element svojevrsne slučajnosti. Ipak, za točnije ocjene utjecaja novog jezika na kreativnost potrebna su dulja razdoblja korištenja i dodatna dublja kvalitativna i kvantitativna analiza.

Od potencijalnih problema intervjuirani su i anketirani stručnjaci istaknuli poglavito probleme koji se obično vezuju uz paradigmu toka podataka te su istaknuli neke manjkavosti grafičkog sučelja prototipa jezika implementiranog u Pure Dati. Izražena je i želja da se omogući jednostavna automatizirana gradnja programa kao što je to u Kymi, no takva su poboljšanja bila izvan opsega istraživanja te se njima treba baviti u budućim istraživanjima.

Uvođenje mogućnosti upravljanja sintezom zvuka pomoću timbralnih atributa te vizualizaciju rezultata izvođenja programa u istoj dimenziji eksperti također smatraju poboljšanjem u odnosu na postojeće jezike. Pritom ističu da odabir temeljnog skupa atributa može značajno utjecati na uporabljivost jezika.

Konačno, mogućnost da se uspoređuju različite inačice istog programa istaknuta je kao bitan faktor kod razvoja programa i traženja pogrešaka. Pritom su u odnosu na implementirani rudimentarni prototip mehanizma zatražena poboljšanja poput usporednog prikaza dva programa, izražajniji i detaljniji oblici prijavljivanja promjena i slično. Takva su značajna poboljšanja van dosega ovoga rada te ih je potrebno razmotriti prilikom izlaska jezika iz faze prototipa.

Kad je riječ o statističkoj analizi ankete koju su ispunjavali glazbenici i umjetnici različitih profila, dobiveni rezultati potvrđuju prethodno iznesene ocjene. Jezik je pozitivno ocijenjen prosječnom ocjenom 7,34 u kontekstu razumljivosti, 6,93 u kontekstu unaprjeđenja u odnosu na prethodno korištene jezike, 6,84 u kontekstu utjecaja toka timbralnih atributa na kreativnost te 7,89 po pitanju poboljšanja procesa programiranja uzevši u obzir sva uvedena poboljšanja.

Budući da je od anketiranih korisnika najveći broj njih imao prethodno iskustvo s Pure Datom (21), Max/MSP-om (16) i Reaktorom (7), dodatno su razmotrene ocjene jezika tih skupina korisnika. U kategorijama unaprjeđenja u odnosu na prethodno korištene jezike, korisnici Pure Date i Max/MSP-a dali su iznadprosječne ocjene (8,33 i 7,21) u odnosu na ukupno anketirane korisnike. Isto vrijedi i za ocjenu poboljšanja procesa programiranja (8,87 i 8,19). Također, analizom njihovih odgovora na otvorena pitanja pokazuje se da predložene nadogradnje smatraju poželjnima i dobrodošlima u kontekstu Pure Date i Max/MSP-a. Ovi su rezultati bili i intuitivno očekivani s obzirom na sličnosti paradigme novog jezika i implementaciju prototipa u Pure Dati. S obzirom na rezultate, vrijedi razmotriti mogućnost da se poboljšanja predloženog jezika preslikaju u proširenja Pure Date i Max/MSP-a te i na taj način, a van izrade sasvim novog jezika, ponude korisnicima.

Za razliku od upravo spomenutih, nešto niže ocjene dali su korisnici Reaktora po dimenzijama utjecaja toka timbralnih atributa na kreativnost (4,93), razumljivosti (6,45) i unaprjeđenja u odnosu na prethodno korištene jezike (4,12). To se može objasniti činjenicom da je, kao što je pokazano u 3.2.2, zajednica korisnika Reaktora primarno orijentirana na korištenje gotovih elemenata na visokoj razini apstrakcije te stvaranje sintetizatora i sekvencera koji se zatim koriste izvan samoga jezika. Stoga je očekivano da tim korisnicima rad na složenim programima na nižoj razini, čak i kad je potpomognut tokom timbralnih atributa, predstavlja neuobičajeni i kognitivno zahtjevniji način rada nego što su navikli.

Neovisno o prethodno korištenom jeziku, rezultati anketa pokazuju pozitivne stavove prema proširenjima za usporedbu vizualnih programa (8,22) i vizualizaciju izlaza programa VU-metrom s vrijednostima timbralnih atributa (6,33).

Promatranjem ukupnosti relevantnih rezultata (**Tablica 3.2**) možemo reći da je kvalitativno ispitivanje validiralo temeljne pretpostavke istraživanja. Tok timbralnih atributa i ostala predložena poboljšanja pozitivno su evaluirana i od strane početnika i od strane iskusnih korisnika. Pritom treba uvažiti da je za dublje i bolje razumijevanje kvaliteta jezika potrebno ostvariti punu implementaciju jezika i učiniti je dostupnom širokom krugu korisnika budući da se neke manjkavosti i prednosti pokazuju tek kroz dulje periode korištenja i stvaranja.

Tablica 3.2 Rezultati kvalitativne evaluacije jezika po pojedinim karakteristikama

Karakteristika	**Prosječna ocjena - svi**	**Prosječna ocjena - početnici**
Razumljivost jezika	7,34	7,11
Primjenjivost jezika	7,77	8,45
Ergonomija i jednostavnost upotrebe	7,63	6,78
Unaprjeđenja u odnosu na druge jezike	6,93	-
Poželjnost timbralnih atributa	8,94	9,09
Utjecaj timbralnih atributa na kreativnost	6,84	7,48
Vizualizacije zvuka timbralnim atributima	6,14	7,67
Sučelja za unos timbralnih atributa	6,11	7,39
Vizualno uspoređivanje programa	9,14	9,17
Iskustvo programiranja u odnosu na druge jezike	7,89	-

3.9.3 Kognitivne dimenzije

Jezik je nakon evaluacije anketama analiziran i kroz prizmu kognitivnih dimenzija. Pritom su kod ocjenjivanja po pojedinim dimenzijama uzeti u obzir rezultati anketa, mišljenja prikupljena kroz intervjue te vrijednosti dimenzija kod prethodno evaluiranih jezika. Drugim riječima, jezik je u analizi po kognitivnim dimenzijama smješten u relativni odnos s Pure Datom, Max/MSP-om, OpenMusicom, Kymom i Reaktorom.

Gradijent apstrakcije

Predloženi jezik zadržava sve dobre značajke demonstrirane u Pure Dati, Max/MSP-u i OpenMusicu te nudi enkapsuliranje u potprograme i apstrahiranje funkcionalnosti. Predloženi model jezika također predviđa eksponiranje pojedinih sučelja enkapsulirane funkcionalnosti na višim razinama, ekvivalentno crtanju u roditelju kod Pure Date.

Promjene u paradigmi predložene ovim radom u načelu ne utječu na gradijent apstrakcije budući da evaluacija postojećih jezika nije pokazala postojanje značajnijih problema u tom aspektu.

Bliskost preslikavanja

Po pitanju bliskosti preslikavanja novi jezik nudi značajna poboljšanja budući da se odmakom od audio signala ka timbralnim atributima približava vokabularu i načinu razmišljanja glazbenika. Istovremeno jezik zadržava dobre elemente Pure Date i Max/MSP-a te nudi razne elemente za generiranje i obradu zvuka.

Potencijalni problem novog jezika jest zadržavanje mogućnosti proceduralnog ili općenitog programiranja koje odudaraju od kognitivnog modela bliskog glazbenicima, ali zato omogućuje korištenje jezika za široki raspon namjena. Budući da korištenje tih mogućnosti nije nužno, njihovo uključivanje smatra se prihvatljivim kompromisom.

Konzistentnost

Budući da je oblikovan prema naputcima danima u 3.4, novi jezik možemo smatrati relativno konzistentnim. Ocjena ove dimenzije slična je Pure Dati uz poboljšanja u vidu mogućnosti skrivanja kontrolnih signala, odnosno njihovo implicitno korištenje uz eksplicitni tok timbralnih atributa. Pritom se na svim razinama apstrakcije i svugdje u hijerarhiji patcha primjenjuje istovjetna paradigma.

Difuznost / sažetost

Difuznost novog jezika može se relativno smjestiti između Pure Date, Max/MSP-a i OpenMusica na jednoj te Kyme i Reaktora na drugoj strani. Relativno poboljšanje u ovoj dimenziji u odnosu na Pure Datu nastaje zbog bliskosti koncepta timbralnih atributa načinu razmišljanja glazbenika te predviđenom sveobuhvatnijem skupu temeljnih objekata.

S druge strane, u ovoj će dimenziji novi jezik uvijek biti iza Kyme zbog činjenice da je Kyma specijaliziranija i konciznija na najvišoj razini apstrakcije. Na temelju rezultata ankete i usporedbe s drugim jezicima, može se zaključiti da je u ovoj dimenziji ostvaren prihvatljiv kompromis.

Sklonost pogreškama

Sklonost pogreškama značajno je smanjena uvođenjem koncepta prostora imena za varijable, ograničenjem dosega bežičnih veza, eksplicitnim definiranjem redoslijeda operacija te korištenjem timbralnih atributa kao razumljivijeg i eksplicitnijeg tipa u toku podataka.

Uz zadržavanje dobrih osobina Pure Date (visoka razina konzistentnosti, preglednost grafičkog prikaza, progresivna evaluacija), novi jezik tako postaje relativno najuspješniji u ovoj dimenziji od dosad promotrenih. Iako ta značajka izostaje u prototipu, uvođenje alata za otkrivanje pogrešaka dodatno bi pridonijelo pozitivnoj ocjeni jezika u ovoj dimenziji.

Kognitivno zahtjevne operacije

Kognitivno zahtjevne operacije još su jedna dimenzija u kojoj se očituju pozitivni utjecaji korištenja timbralnih atributa u toku podataka. Naime, budući da glazbenici ne moraju razmišljati o audio signalima već o intuitivno razumljivim vrijednostima timbralnih atributa, smanjuje se njihova kognitivna opterećenost prilikom programiranja. Osim tog poboljšanja, jezik zadržava sve dobre osobine Pure Date. Kao i kod Pure Date, korištenjem općenitih, proceduralnih programskih konstrukata, poput petlji, povećava se kognitivna zahtjevnost.

Skrivene ovisnosti

Uvođenjem prostora imena te inzistiranjem na eksplicitnom uvozu svih vanjskih proširenja koja su zatim vidljiva u popisu prikazanom u odvojenom prozoru, jezik riješava probleme skrivenih ovisnosti prisutne u Pure Dati i Max/MSP-u. Upravljanje ovisnostima moguće je dodatno poboljšati ostvarivanjem specifičnog alata za tu namjenu koji bi pratio sve datoteke uključene u projekt kao i njihove rekurzivne ovisnosti. Takvo je proširenje van opsega ovog rada.

Preuranjeno opredjeljivanje

Problem preuranjenog opredjeljivanja u novom je jeziku podjednak kao u Pure Dati i Max/MSP-u te je ovisan o inherentnim ograničenja paradigme toka podataka. Analiza ponuđena za Pure Datu i Max/MSP vrijedi i u ovom slučaju. Rješavanjem problema preuranjenog opredjeljivanja ovaj rad se nije bavio.

Progresivna evaluacija

Predloženi jezik zadržava sve dobre osobine progresivne evaluacije prisutne kod Pure Date, Max/MSP-a i OpenMusica te ih dodatno poboljšava uvođenjem elementa za stvarnovremensku vizualizaciju vrijednosti timbralnih atributa izlaza programa (opisano u poglavlju 4.2). Slično kao kod Pure Date, patch, odnosno program, mora biti valjan i u potpunosti spojen da bi se mogao evaluirati.

Izražajnost uloga

Izražajnost uloga poboljšana je uvođenjem timbralnih atributa koji su jasniji i eksplicitniji od raznih uobičajenih parametara audio signala. Pritom je od OpenMusica i Reaktora preuzeta veća izražajnost samih simbola podpatcheva koje korisnici mogu proizvoljno postavljati. Konačno, pojedini su blokovi svojim izgledom bliži stvarnim elementima koje modeliraju (npr. oscilatori, sintetizatori itd.).

Sekundarna notacija

Predloženi jezik zadržava sve dobre osobine sekundarne notacije iz Pure Date bez većih proširenja ili poboljšanja. Potreba za sekundarnom notacijom smanjuje se poboljšanjima u dimenzijama izražajnosti uloga i bliskosti preslikavanja.

Viskoznost

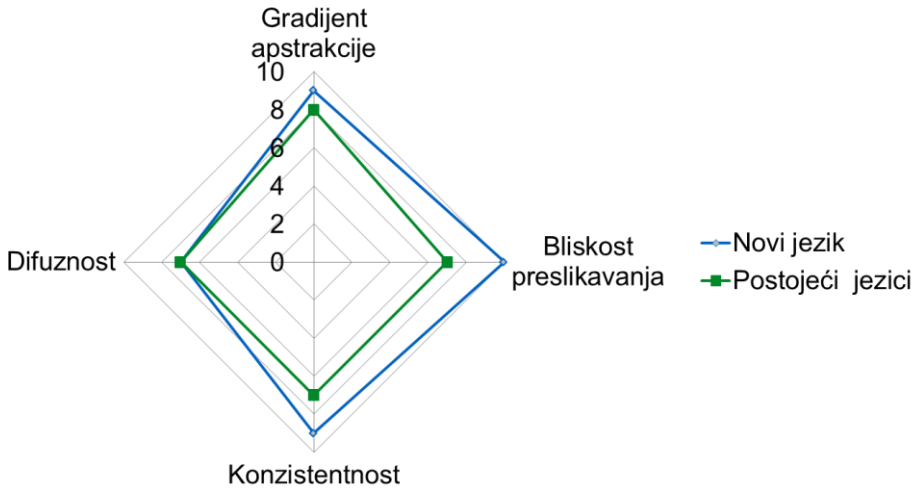
Viskoznost se pokazuje najsloženijom dimenzijom kod vizualnih programskih jezika kad su u pitanju poboljšanja. Novi jezik zadržava značajke Pure Date, OpenMusica i Max/MSP-a te ne uvodi inovacije koje bi utjecale na ovu dimenziju. U poglavlju 6 razmotrena su neka moguća poboljšanja kojim bi se mogla baviti buduća istraživanja.

Vidljivost

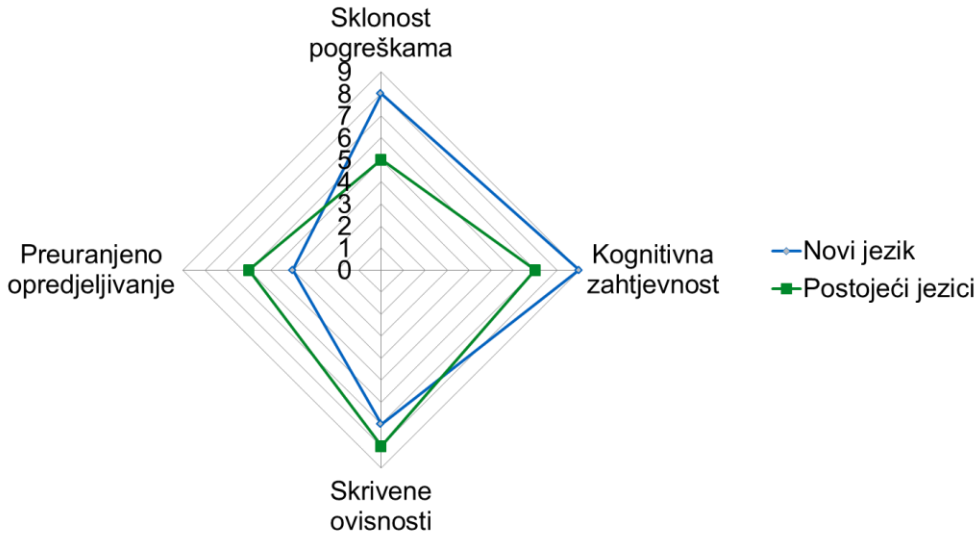
Razine apstrakcije predložene u novom jeziku impliciraju i veliku mogućnost skrivanja pojedinih dijelova programa te će vidljivost uvelike ovisiti o programeru. Što se statičkih pokazatelja vidljivosti tiče, jezik zadržava osobine Pure Date i Max/MSP-a uz određena poboljšanja u slučaju implementacije elementa poput maquettea ili vremenske crte.

3.9.4 Usporedba radar-dijagramima

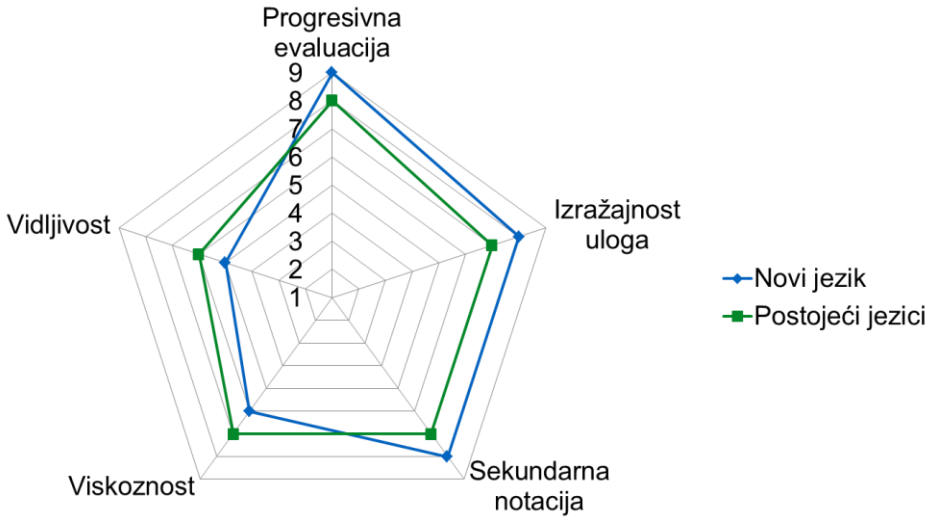
Na temelju analize prezentirane u prethodnom poglavlju, izrađeni su radar-dijagrami (**Slika 3.12**, **Slika 3.13** i **Slika 3.14**) koji pokazuju relativne vrijednosti dimenzija novoga jezika u usporedbi s prosjekom za Pure Datu, Max/MSP, OpenMusic, Reaktor i Kymu. Važno je napomenuti da na dijagramima pojedine vrijednosti dimenzija odgovaraju ocjenama dobrote, odnosno veće je uvijek bolje.



Slika 3.12 Relativne ocjene novog jezika po dimenzijama gradijenta apstrakcije, bliskosti preslikavanja, konzistentnosti i difuznosti



Slika 3.13 Relativne ocjene novog jezika po dimenzijama sklonosti pogreškama, kognitivne zahtjevnosti, skrivenih ovisnosti i preuranjenog opredjeljivanja



Slika 3.14 Relativne ocjene novog jezika po dimenzijama progresivne evaluacije, izražajnosti uloga, sekundarne notacije, viskoznosti i vidljivosti

3.10 Osvrt

Evaluacijom i validacijom pomoću kognitivnih dimenzija, kvantitativnom i kvalitativnom analizom pokazano je da predloženi novi jezik donosi poboljšanja u odnosu na prethodna rješenja u području. Iako su poboljšanja koja donosi inkrementalna, njihov zbroj donosi

značajan potencijal unaprjeđenja iskustva korištenja jezika u izradi glazbenih skladbi, sintetizatora zvuka itd.

Prihvaćanjem kompromisa i određenih ograničenja, odnosno svojevrsnom specijalizacijom – cilj novog jezika nije nadomještanje uloge NI Reaktora ili Audacityja – te uvrštavanjem funkcionalnosti poput mogućnosti uspoređivanja programa postignuta su poboljšanja u specifičnim područjima i primjenama vizualnog programiranja u kontekstu glazbe. Primarna poboljšanja pritom ostvarena su po pitanju intuitivnosti korištenja, bliskosti paradigmama koje su poznate glazbenicima, dok je niz drugih dimenzija poboljšan posredno. Primjerice, zbog bolje bliskosti preslikavanja i izražajnosti uloga očekuje se smanjenje učestalosti činjenja pogrešaka.

Kao sljedeći korak u budućim istraživanje predlaže se izrada pune implementacije jezika, njegova diseminacija među što većim brojem glazbenika te dodatne kvantitativne i kvalitativne analize nakon dužih perioda korištenja jezika.

4 Intuitivno upravljanje elementima jezika i vizualiziranje rezultata sinteze zvuka pomoću timbralnih atributa

Kako bi se vizualni programski jezik, u čijoj se srži nalaze timbralni atributi, učinio pristupačnijim korisnicima, bilo je potrebno osmisliti i prilagoditi ulazna i izlazna sučelja pojedinih blokova, odnosno objekata, ali i metode vizualnog predstavljanja rezultata izvođenja programa. Drugim riječima, da bi unutarnja reprezentacija podataka rezultirala boljom interakcijom s korisnicima, potrebno ju je prikazati na izražajan način u vizualnoj domeni i u kontekstu interakcije čovjeka s računalom (eng. *human-computer interaction*). Ovaj se problem može podijeliti u dvije podkategorije.

U prvoj se nalaze grafička sučelja pomoću kojih korisnici određuju ulazne vrijednosti timbralnih atributa na pojedinim blokovima. Ta sučelja moraju biti jasno definirana, izražajna i intuitivna. Samim korištenjem timbralnih atributa nasuprot definiranju sirovih parametara audio signala postignuta su značajna poboljšanja, a pritom su razmotreni različiti grafički elementi za unos vrijednosti atributa. Kao jedan od temeljnih problema ovakvog pristupa javlja se mapiranje timbralnih u konačne vrijednosti audio signala.

Druga podkategorija problema jest prikazivanje rezultata izvođenja programa, odnosno sintetiziranog zvuka ili stvorene glazbe. Iako u jezicima poput Pure Date i OpenMusica postoji niz gotovih objekata za izlučivanje i prikazivanje značajki audio signala, svi su oni primarno orijentirani na informacije niske razine poput frekvencije ili spektra.

Iako se ove dvije kategorije oslanjaju na slične mehanizme mapiranja, u nastavku će biti izložene i analizirane zasebno.

4.1 Sučelja temeljena na timbralnim atributima

Tijekom istraživanja predstavljenog u disertaciji, razmotren je niz postojećih varijanti grafičkih elemenata za unos vrijednosti. Glavni faktor prilikom njihove analize bila je mogućnost prilagodbe potrebama unosa timbralnih atributa i općenito kontekstu stvaranja glazbe. U tom je smislu razmotren široki raspon rješenja, od onih temeljenih na vrlo jednostavnim sučeljima poput običnih brojčanih polja do složenijih ulaznih elemenata poput grafičkih prikaza potenciometara i interaktivnih radar-dijagrama.

Sva rješenja predstavljena u ovom poglavlju pretpostavljaju postojanje mehanizama za preslikavanje izlaznih vrijednosti grafičkih elemenata, prvo u brojčane iznose timbralnih

atributa, a zatim transformaciju tih vrijednosti u parametre sintetizatora i drugih funkcionalnosti objekata u jeziku.

4.1.1 Od timbralnih atributa do audio signala

Kako bi se timbralne atribute moglo upotrijebiti u ulozi parametara na različitim objektima u jeziku (primarno na sintetizatorima), bilo je potrebno osmisliti mehanizme njihova preslikavanja u parametre sintetizatora te omogućiti njihovo manipuliranje u drugim objektima koji su inače predviđeni za rad sa sirovim audio signalima. Drugim riječima, svaki objekt u novom jeziku koji na svom ulazu prima timbralne atribute ili koji omogućava mijenjanje njihovih vrijednosti (poput zbrajala), u svojoj implementaciji sadrži translacije iz timbralnih atributa u audio signale.

Korištene metode detaljno su opisane u poglavlju 3.5 kao i radu [19] te stoga neće biti podrobnije razmatrane na ovom mjestu.

4.1.2 Oblikovanje rješenja

Oblikovanje rješenja oslanjalo se na saznanja prikupljena kvantitativnom analizom (3.2) i analizom temeljenom na kognitivnim dimenzijama (3.3) te na prethodne radove poput [80] i [81] koji su razmatrali ergonomiju unosa vrijednosti u vizualnom programiranju. Pritom je posebni naglasak stavljen na dimenzije poput bliskosti preslikavanja fizičkih i glazbenicima poznatih sučelja u grafičke elemente jezika.

Kao što je vidljivo iz rezultata kvantitativne analize te iz intervjua s naprednim korisnicima, sučelja koja oponašaju stvarne uređaje (poput sintetizatora) na glazbenike djeluju dvojako:

- Omogućavaju im unos vrijednosti na poznat i intuitivan način te im garantiraju predvidljivo ponašanje. Primjerice, u trivijalnom slučaju, okretanjem potenciometra u smjeru kazaljke na satu povećava se vrijednost onog timbralnog atributa koji je na taj potenciometar preslikan. No dok je takav pristup iznimno pristupačan i lako razumljiv, može predstavljati ograničenja iskusnijim korisnicima. Često se takvi korisnici žele oslanjati na finiji i precizniji unos numeričkog oblika vrijednosti. Stoga je razmotren i takav pristup.
- Svojim izgledom i prepoznatljivošću demistificiraju proces programiranja i smanjuju kognitivne zapreke, a pritom i sakrivaju kompleksnosti audio signala koji su uvijek rezultat rada programa. Na taj se način smanjuje ukupna zahtjevnost i potencijalni zazor od procesa programiranja.

Od promotrenog skupa grafičkih elemenata za unos vrijednosti, detaljnije su razmotrena tri koja mogu poslužiti kao arhetipovi pojedinih kategorija ulaznih elemenata.

Izravni numerički unos

Najjednostavniji i najizravniji pristup unosa timbralnih atributa uključuje jednostavno definiranje specifičnih timbralnih atributa specificiranjem njihovih apsolutnih vrijednosti na zajedničkoj skali. U tom smislu definira se vrijednost od 0 do 100 za svaki pojedini atribut. Vrijednosti pojedinih atributa pritom su neovisne jedne o drugima što može dovesti do neželjenih situacija u kojima se međusobno komplementarni atributi (npr. svijetao i taman) poništavaju. Unos putem takvog slobodnog formata može dovesti do neželjenih, neočekivanih rezultata.

Stoga je razmotreno i poboljšano rješenje u kojem se atributi uparuju po ključu komplementarnosti. Povećavanjem vrijednosti jednog atributa za isti relativni iznos smanjuje se iznos komplementarnog atributa. Nedostatak ovakvog pristupa je ograničavanje dijela fleksibilnosti unosa. Kao i u drugim segmentima jezika, riječ je o kompromisu između intuitivnosti i lakoće korištenja (čime se posredno smanjuje i učestalost pogrešaka) s jedne strane te potpune slobode programskog izražavanja s druge.

Iako nudi najveću fleksibilnost i mogućnost preciznog definiranja vrijednosti, riječ je o ergonomski rudimentarnom načinu unošenja podataka, posebno prilikom eksperimentiranja tijekom progresivne evaluacije. Uočljivo je pritom i odsustvo takvih sučelja u svijetu fizičkih uređaja za sintetiziranje i stvaranje glazbe čime se smanjuje izražajnost uloga i bliskost preslikavanja.

Istovremeno, ovakvo sučelje je nužno jer omogućuje jednostavno definiranje velikog broja vrijednosti te će biti korisnije naprednim korisnicima.

Klizači i potenciometri

Iako je naoko riječ o različitim elementima unosa, klizači i potenciometri svojim karakteristikama pripadaju istoj skupini. Njihova je glavna, funkcionalno trivijalna razlika u tome mijenja li se ulazna vrijednost povlačenjem grafičkih elementa gore-dolje ili okretanjem lijevo-desno.

Zbog bliskosti sučeljima na uređajima poput analognih sintetizatora i ploča za miksanje audio signala, njihova je bliskost preslikavanja i izražajnost uloga iznimno visoka. Ipak, zbog svoje veličine i nesažetog vizualnog prikaza zauzimaju više prostora na radnoj površini te mogu dodatno narušiti viskoznost i druge povezane dimenzije vizualnog programa.

Kao i kod izravnog numeričkog unosa, potenciometre i klizače moguće je definirati tako da njihove vrijednosti budu apsolutne, na skali od 0 do 100 – što odgovara krajnjim položajima klizača, odnosno potenciometra. U kontekstu sinteze zvuka i stvaranja glazbe takav je pristup prirodniji i odgovara sličnim kontrolama svjetline i basa na glazbenim instrumentima i drugoj audio opremi.

Ipak, u kontekstu upravljanja parametrima u programskom jeziku, a kako bi se smanjila učestalost činjenja pogrešaka, promotreno je i rješenje kod kojeg se krajnji položaji potenciometra i klizača preslikavaju u relativne vrijednosti komplementarnih atributa. Primjerice, ako donji položaj klizača proglasimo "drvenastim", a gornji "metalnim", pomicanje klizača od dna prema vrhu smanjivat će vrijednost atributa koji definira da je zvuk "drvenast", a za istu vrijednost povećavati udio timbralnog atributa koji se tumači kao "metalan".

Interaktivni radar dijagrami

Ovaj se način unosa može promatrati kao dvodimenzionalna varijanta klizača i potenciometara uz istodobno kontroliranje dva do četiri timbralna atributa. Grafički gledano, element se predstavlja kao dvodimenzionalni kartezijev koordinatni sustav podijeljen u četiri sektora. Vrijednosti atributa nalaze se na apscisi odnosno ordinati.

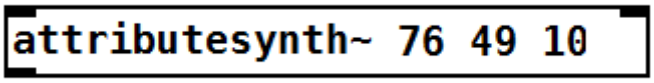
U prvom su slučaju ordinata i apscisa svaka vezane uz jedan timbralni atribut. Pomicanjem točke odabira vrijednosti u tom prostoru mijenjaju se iznosi atributa, neovisno jedni o drugima. Primjerice, ako na ordinati imamo vrijednost atributa "svijetao", a na apscisi "drvenast", pomicanje točke iz lijevog donjeg sektor u gornji desni učinit će zvuk istovremeno svijetlijim i drvenastijim.

U drugom se slučaju, na sličan način kao što je demonstrirano kod klizača i potenciometara, apscisa i ordinata vezuju uz po dva timbralna atributa. Primjerice, na apscisi se mogu nalaziti "taman" (lijevo od ishodišne točke) i "svijetao" (desno od ishodišne točke), a na ordinati "metalan" (ispod ishodišne točke) i "drvenast" iznad ishodišne točke. U tom se slučaju pomicanjem točke koja simbolizira trenutne vrijednosti atributa iz donjeg desnog sektora u gornji lijevi, zvuk čini istovremeno manje svijetlim, tamnijim, manje metalnim, a više drvenastim.

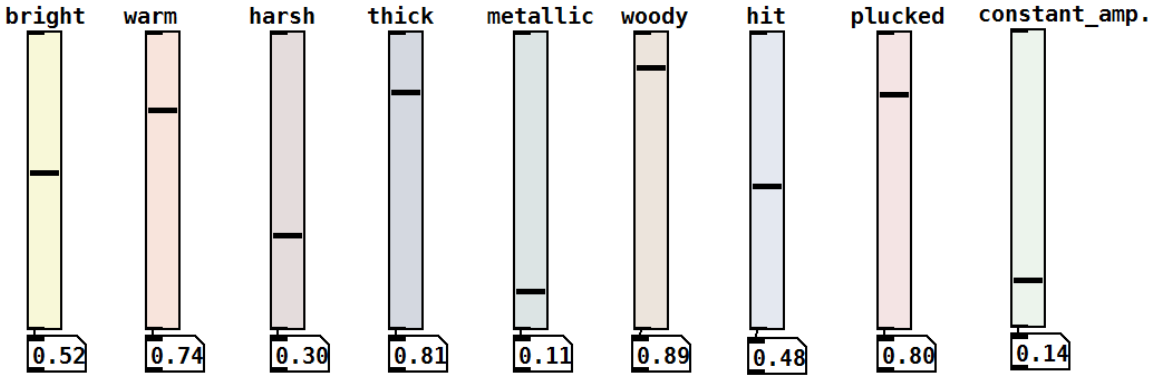
Poopćavanjem ovog modela dodavanjem treće dimenzije, moguće je istovremeno manipulirati s najviše šest timbralnih atributa, no to zahtijeva i mogućnost korištenja trodimenzionalnih grafičkih sučelja u programskom jeziku.

4.1.3 Prototip

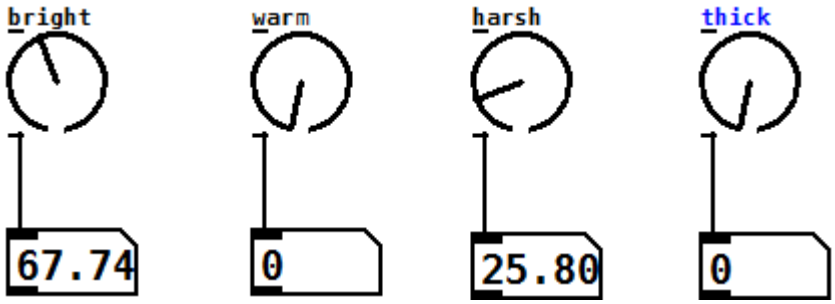
Na slikama **Slika 4.1**, **Slika 4.2** i **Slika 4.3** prikazane su implementacije prva dva pristupa za kontrolu timbralnih atributa opisana u prethodnom poglavlju. Implementacija se temeljila na modifikaciji postojećih sličnih grafičkih elemenata u jeziku Pure Data te na njihovoj nadogradnji mehanizmima za translaciju timbralnih atributa.



Slika 4.1 Određivanje vrijednosti ulaznih timbralnih atributa izravno na objektu sintetizatora



Slika 4.2 Klizači za određivanje vrijednosti timbralnih atributa



Slika 4.3 Potenciometri za određivanje vrijednosti timbralnih atributa

Iako nije provedena zasebna evaluacija ovih elemenata, njihova su poboljšanja validirana i verificirana kroz evaluaciju prototipa novog jezika. Anketirani i intervjuirani korisnici pritom su istaknuli vrijednost takvog pristupa, a slični pokazatelji dobiveni su i kvantitativnom analizom rezultata anketa.

4.2 Vizualiziranje rezultata u domeni timbralnih atributa

Kao što je pokazano u poglavlju 2.3, postoje mnogobrojna rješenja za vizualizaciju zvuka. U najjednostavnijem slučaju, možemo reći da se ulazni elementi prikazani u prethodnom

poglavlju mogu izokrenuti te koristiti za prikazivanje rezultata, tako da ulazni grafički elementi postanu izlazni. No takvo je rješenje krnje i neprilagođeno potrebama glazbenika koji su naviknuti na prikaze poput spektrograma, mjerače jačine zvuka (eng. *VU meter*) i sl. Stoga su za potrebe novog vizualnog programskog jezika razmotrena samo rješenja bliska postojećem semantičkom vokabularu glazbenika i skladatelja.

4.2.1 Od audio signala do timbralnih atributa

Kako bi bilo moguće vizualizirati zvuk u domeni timbralnih atributa, potrebno je iz generiranog zvuka, odnosno rezultata izvođenja programa, izlučiti vrijednosti timbralnih atributa. Ovaj je postupak nedeterministički i neizrazit u smislu da su dobivene vrijednosti aproksimacije. Takav zaključak proizlazi iz nemogućnosti potpuno diskretnog i predvidivog opisivanja zvuka timbralnim atributima, odnosno izostankom bijekcije u lancu ulaznih timbralnih atributa, generiranog zvuka i timbralnih atributa izlučenih iz generiranog zvuka.

Budući da su ovi postupci detaljno opisani u radovima [19][20] i pojašnjeni u 3.5, na ovom mjestu neće biti detaljnije predstavljeni. Pretpostavlja se da se oni koriste za ostvarivanje svih predloženih rješenja vizualizacije rezultata predstavljenih u ovom poglavlju.

4.2.2 Oblikovanje rješenja

Prilikom oblikovanja rješenja uzeti su u obzir rezultati kvantitativne analize opisane u poglavlju 3.2, analize pomoću kognitivnih dimenzija predstavljene u poglavlju 3.3 i intervjua s ekspertima iz područja. Iz tih je rezultata izlučeno da je glazbenicima prilikom vizualiziranja najbitniji intuitivno razumljiv prikaz zvuka u vizualnoj domeni. Odnosno, najpoželjnijom se pokazuje bliskost vizualizacijama poput spektrograma i jednostavnih mjerača amplitude različitih značajki zvuka. Takve su vizualizacije uobičajene kako u hardverskim rješenjima, tako i u digitalnim radnim stanicama za obradu zvuka.

U idealnom slučaju, vizualiziranje se odvija u stvarnom vremenu te trenutno reagira na promjene ulaznih vrijednosti timbralnih atributa koji se koriste za manipulaciju zvukom.

Spektrogram

Spektrogramom nazivamo vizualni prikaz spektra frekvencija signala koji se mijenja u vremenu. Preneseno u domenu timbralnih atributa, spektrogram se sastoji od niza diskretnih segmenata za svaki minimalni vremenski odsječak audio signala te prikazuje odsječak povijesti u kontekstu promjena atributa. Promatrani vremenski odsječak, odnosno prozor, može biti unaprijed određen ili konfigurabilan, ovisno o značajkama ciljnog računalnog sustava, arhitekture i sl.

Za svaki od spomenutih vremenskih odsječaka određuju se prosječne vrijednosti promatranih atributa. Kako se kod uobičajenih značajki zvuka promatra samo jedna vrijednost – najčešće je to frekvencija – korištenje spektrograma u domeni timbralnih atributa nešto je složenije budući da je poželjno istovremeno pregledavanje vrijednosti više od jednog atributa.

Stoga se nameću tri moguća rješenja:

- Jednostavni spektrogram koji može prikazivati promjenu određenog timbralnog atributa u vremenu. Kako bi istovremeno pregledavali promjene više od jednog atributa, potrebno je u program uključiti više blokova za prikaz spektralne analize, po jedan za svaki željeni atribut. Takav pristup značajno narušava viskoznost i preglednost programa.
- Spektrogram kod kojih pojedini segment definira vrijednosti komplementarnih atributa. Drugim riječima, apsolutna amplituda pojedinog odsječka dobiva se oduzimanjem vrijednosti međusobno suprotnih atributa poput „svijetao“ i „taman“. Kod ovakvog pristupa jedan blok za prikaz spektralne analize zapravo daje informacije o dva atributa. Kao i u prethodnom slučaju, u program je moguće uključiti višestruke blokove kako bi se pokrilo više atributa.
- Složeni spektrogram kod kojeg je svaki segment prikazan različitim gradijentima boja čije prisustvo ili jača izraženost definira veće prisustvo određenog atributa. Riječ je o vrlo složenom načinu prikaza koji zahtijeva visoku razlučivost grafičkog sučelja.

Na kraju valja napomenuti da iako ovo rješenje u poglavlju nazivamo „spektrogramom“, zbog njegova opisnog značaja i prepoznatljivosti, provedenim modifikacijama za rad s timbralnim atributima zapravo se više izravno ne predstavlja spektar kao karakteristika audio signala.

VU metar

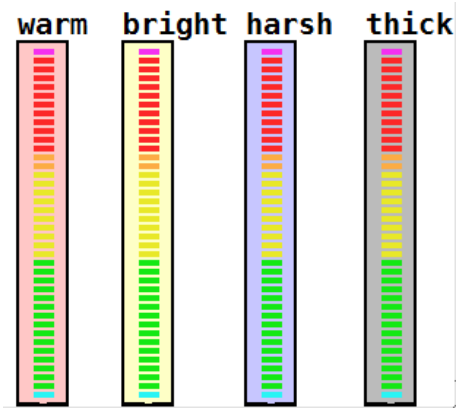
U izvornom smislu, VU metar je uređaj koji prikazuje iznos ili amplitudu signala i najčešće se pronalazi na glazbenoj i audio opremi poput pojačala ili ploča za miksanje. U kontekstu računarstva i vizualnog programiranja u glazbi, objekti koji simuliraju VU metar najčešće mogu grafički prikazivati različite diskretne vrijednosti unutar programa.

Za potrebe vizualnog programskog jezika temeljenog na atributima, VU metar se prilagođava tako da pokazuje vrijednosti pojedinih timbralnih atributa. Riječ je o jednostavnom, ali vrlo izražajnom načinu vizualizacije. Zbog svog relativno kompaktnog grafičkog prikaza, na radnu je površinu moguće postaviti više VU metara, po jedan za svaki praćeni atribut, bez gubitka prostora ili značajnog povećanja viskoznosti.

Za razliku od spektrograma, VU metar pokazuje samo trenutno stanje signala tijekom progresivne evaluacije te ne bilježi povijesna stanja osim što postoji mogućnost da se dodatnim grafičkim indikatorom označi minimalna i maksimalna vrijednost pojedinog atributa u određenom vremenskom prozoru.

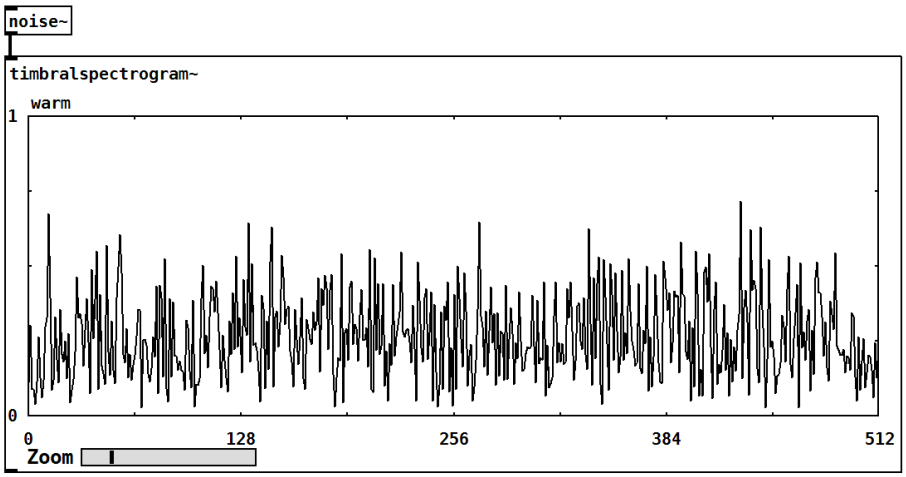
4.2.3 Prototip

Prilikom izrade prototipa spektrograma prilagođenog tibralnim atributima, a zbog ograničenja grafičkog podsustava Pure Date, izabran je najjednostavniji model rješenja, u kojemu je spektrogramom moguće pratiti samo jedan timbralni atribut.



Slika 4.4 Vizualiziranje vrijednosti timbralnih atributa u analiziranom zvuku pomoću VU metara

Spektrogram (**Slika 4.5**) i VU metar (**Slika 4.4**) temeljeni na timbralnim atributima u prototip su ugrađeni prilagodbom postojećih rješenja za rad s audio signalima. Zbog složenosti algoritama koji izlučuju atribute iz audio signala i ekstrapoliraju vrijednosti timbralnih atributa potrebne za vizualizaciju, kod progresivne evaluacije postoji minimalna latencija između promjena rezultantnog signala i njegove vizualizacije.



Slika 4.5 Spektrogram s prikazom vrijednosti timbralnog atributa „topao“ (eng. *warm*) ulaznog signala

4.3 Osvrt

Evaluacija novog jezika temeljenog na timbralnim atributima pokazala je da su funkcionalnosti opisane u ovom poglavlju ključne za uspjeh ciljanih poboljšanja jezika. Učinkovit i intuitivan rad s jezikom oslanja se na dobro prilagođena ulazna i izlazna sučelja ključnih blokova. Iako zasebna evaluacija predloženih rješenja specificiranja ulaznih vrijednosti timbralnih atributa i vizualiziranja rezultata sinteze zvuka i stvaranja glazbe nije provedena, njihova je vrijednost potpuno vidljiva tek u sklopu novog jezika.

Modifikacijom blokova tako da se vrijednosti parametara definiraju na ergonomičan i izražajan način glazbenicima je omogućeno da željeni zvuk opisuju njima prepoznatljivim vokabularom. S druge strane, omogućavanjem vizualizacije izlaznog signala u domeni istih vrijednosti timbralnih atributa ostvarena je jasno vidljiva i lako razumljiva povratna veza između modifikacija programa, odnosno ulaznih atributa, i značajki izlaznoga zvuka.

U tom se smislu jezik posebno prilagođava onim glazbenicima i skladateljima koji vizualno programiranje koriste u kreativnom procesu od početka do kraja (eng. *top to bottom*).

5 Uspoređivanje vizualnih programa za sintezu zvuka i stvaranje glazbe

Kao što je spomenuto u poglavlju 2.4, rješenja za vizualno uspoređivanje programa najčešće su vezana uz pojedinačne programske jezike. Izostankom alata primjenjivih generalno u vizualnoj dimenziji te analizom sličnih vizualnih programskih jezika u domeni glazbe, utvrđeno je da bi funkcionalnost za uspoređivanje programa znatno olakšala rad s jezikom te pomogla pri otkrivanju pogrešaka. Takav zaključak proizlazi i iz komentara korisnika koji često ističu nemogućnost uspoređivanja programa kao bitan nedostatak postojećih rješenja.

U nastavku je prezentirano specifično rješenje za predloženi programski jezik te implementacija rješenja nad prototipom realiziranim proširenjem Pure Date.

5.1 Temeljni problemi vizualnog uspoređivanja

Nad programima tekstnih jezika može se primijeniti bilo koji od alata za usporedbu teksta [82][83] budući da su oni najčešće agnostični kada je riječ o vrsti programskog jezika koji se uspoređuje. Nasuprot tome, svaki vizualni programski jezik definira svoju specifičnu sintaksu u vizualnom mediju i popratnim strukturama na datotečnom sustavu. Iako je i u tom slučaju često moguće uspoređivati tekstne reprezentacije vizualnih programa (najčešće u XML-u) korištenjem prethodno spomenutih alata za usporedbu teksta, riječ je o ergonomski nedostatnom načinu rada koji dodatno traži upotrebu vanjskih alata. Kod složenijih i većih programa ova metoda nije upotrebljiva. Uzrok toga nalazi se u kognitivnoj zahtjevnosti povezivanja grafičkog prikaza programa i njegova tekstnog zapisa na datotečnom sustavu. Prilikom korištenja takvih ručnih metoda usporedbe korisnik mora razumjeti strukturu i značenje zapisa programa, a isti često nisu namijenjeni ljudima (eng. *human-readable*).

Vizualne je programe stoga potrebno uspoređivati na način koji bi bio bliži njihovoj paradigmi i načinu programiranja. Kad je riječ o problemima takvoga pristupa, možemo ih podijeliti u dvije skupine. U prvoj se skupini nalaze poteškoće u određivanju pojedinačnih promjena koje su se zbile u programu. Budući da su vizualni programi dvo ili višedimenzionalni, nužno je u obzir, osim atomskih promjena (npr. modifikacija brojčanih parametara objekata), uzeti i promjene poput položaja objekata na radnoj površini, veza između njih itd. Ovisno o načinu zapisa vizualnog programa na datotečnom sustavu, mogući su različiti pristupi u određivanju vektora razlike između inačica programa.

U drugu skupinu pripadaju problemi vezani uz povezivanje vizualne s tekstnom reprezentacijom programa te poteškoće vezane uz samu vizualizaciju i prikaz promjena korisnicima. Jasno je da je sveobuhvatno rješenje takvog problema iznimno složeno te da je ograničeno nedostatcima same programske paradigme i mogućnostima grafičkog prikaza.

5.2 Oblikovanje rješenja

Slično kao i sam jezik, rješenje za uspoređivanje programa razrađeno je kroz nekoliko faza. U prvoj su fazi promotreni rezultati analize postojećih jezika i sličnih rješenja. Zatim su na temelju tih saznanja i prema specifičnostima novog jezika predstavljenog u 3.7 određene osnovne značajke i zahtjevi koje model rješenja mora zadovoljiti. U trećoj je fazi na temelju značajki oblikovana arhitektura i model mehanizama za uspoređivanje jezika te su određeni koraci koje rješenje mora sadržavati i koji opisuju njegovo ponašanje. Konačno, ostvaren je prototip koji elemente predloženog rješenja ugrađuje u prototip jezika u Pure Dati.

5.2.1 Analiza postojećih istraživanja i izlučivanje zahtjeva

Prilikom prve faze izrade arhitekture i modela rješenja za uspoređivanje vizualnih programa koji ostvaraju glazbene sintetizatore i skladbe, uzeto je u obzir nekoliko faktora na temelju prethodno provedenih analiza i postojećih istraživanja u području.

Rezultati kvantitativne analize

Prvo su promotreni rezultati analiza anketa prezentiranih u 3.2. Te ankete istaknule su važnost mogućnosti kontinuirane evaluacije programa te značajno smanjenje učestalosti pogrešaka kod onih jezika koji sadrže funkcionalnosti za lakše otkrivanje pogrešaka. Dodatno, pokazuje se da tijekom učenja jezika korisnici često koriste učenje putem postojećih primjera. Pritom svoje prve programe ostvaruju modifikacijom postojećih patcheva i programa.

U oba ova slučaja mogućnost uspoređivanja programa pokazuje se bitnim poboljšanjem. U prvom slučaju korisnici mogu uspoređivati različite inačice svog programa te pregledom promjena lakše pronaći područja u kojima su pogreške mogle nastati. U drugom slučaju usporedbom originalnih programa preuzetih, primjerice, putem interneta i vlastitih modifikacija programa korisnici mogu lakše shvatiti kako manipulacije pojedinim blokovima ili njihovim parametrima utječu na rezultate izvođenja programa.

Iz činjenice da korisnici, kako iskusniji tako i početnici, često koriste mehanizme sekundarne notacije proizlazi potreba da se vizualni prikaz promjena učini izražajnim.

Mogućnosti kao istovremeni prikaz prethodnih i trenutnih vrijednosti te dodatne informacije o promjenama (npr. kad su učinjene) također pripomažu prilikom korištenja jezika.

Rezultati kvalitativne analize

Ankete korištene prilikom kvantitativnih analiza također su sadržavale pitanja otvorenog tipa. Iako pregled takvih odgovara nudi isključivo anegdotalno znanje i ne može biti temeljem za statističku evaluaciju, ipak nudi uvid u određene probleme s kojima se susreću korisnici te rješenja koja bi htjeli vidjeti u jezicima koje koriste.

Posebno se u tom smislu ističu korisnici Pure Date i Max/MSP-a koji tijekom razvoja kompleksnijih programa često nailaze na probleme vezane uz sljedivost vlastitih rješenja. Drugim riječima, postaje im teško pratiti promjene koje su činili tijekom razvoja programa. Ta činjenica dodatno može uzrokovati nekonzistentnosti i potencijalne pogreške. Primjerice, ranija promjena određenog parametra u podpatchu može dovesti do kasnijih promjena rezultata u roditeljskom patchu. Otkrivanje takvih pogrešaka može biti složeno i dugotrajno. Vizualiziranjem promjena između pojedinih inačica programa i upućivanjem korisnika na točan dio programa u kojoj se promjena zbila, smanjuje se utjecaj ranijih promjena na željene rezultate.

Konačno, pokazuje se da korisnici posebno vrednuju mehanizme i dodatne alate koji zadržavaju bliskost domeni primjene iz čega proizlazi da uspoređivanje vizualnih programa novog jezika mora posebno istaknuti promjene među vrijednostima timbralnih atributa.

Kognitivne dimenzije

Rješenje za uspoređivanje programa značajno utječe na kognitivne dimenzije poput sklonosti pogreškama, progresivne evaluacije, kognitivno zahtjevnih operacija i preuranjenog opredjeljivanja.

Kad je riječ o sklonosti pogreškama, a kako je već spomenuto tijekom poglavlja, omogućavanjem da se uspoređuju različite inačice istog programa postaje jednostavnije otkrivati već učinjene pogreške, ali i smanjuje pojavu novih pogrešaka budući da korisnici na temelju prethodnih usporedbi uče o područjima u kojima najviše čine pogreške. Time im je omogućeno da u daljnjem radu obrate više pozornosti na one akcije koje su u prošlosti dovele do pogrešaka.

Predloženo rješenje mora omogućiti usporedno uspoređivanje, uređivanje i izvršavanje programa. Time se omogućuje da se u stvarnom vremenu vide utjecaji pojedinih promjena programa na njegove rezultate čime se ostvaruju značajna poboljšanja u dimenziji progresivne

evaluacije. Dodatno, model može predvidjeti postojanje izlaznih blokova koji bi uspoređivali izlaze programa.

Jasno je da će postojanje alata za uspoređivanje programa značajno umanjiti zahtjeve po pitanju kognitivnih napora tijekom razvoja programa. Umjesto da korisnici ručno uspoređuju dva programa – primjerice istovremenim pregledavanjem inačica programa u dvije radne površine – omogućuje im se interaktivna i vizualno efektna identifikacija promjena.

Konačno, zbog mogućnosti da se prate promjene kroz različite inačice programa, smanjuje se negativni utjecaj preuranjenog opredjeljivanja koji je, kao što je prikazano u ranijoj analizi u poglavlju 3.9, jedan od i dalje prisutnih problema u predloženom jeziku.

Iako mogućnost uspoređivanja programa ne utječe izravno na viskoznost i difuznost jezika, potencijalno omogućuje lakše snalaženje u iteracijskom razvoju složenih programa kod kojih su negativni utjecaji viskoznosti i difuznosti posebno prisutni.

Postojeća istraživanja i rješenja

Iako ne postoje generalno primjenjiva rješenja za uspoređivanje vizualnih programa, kao što je objašnjeno u 2.4 i na početku ovog poglavlja, razmotrena su slična rješenja u srodnim jezicima te su translatirane dobre prakse prisutne kod uspoređivanja tekstnih jezika.

Radovi poput [59] i [84] uzeti su kao primjeri uspoređivanja specifičnih vizualnih programa te su iz njih preuzeti apstraktni postupci na visokoj razini pomoću kojih se programi uspoređuju. Riječ je o općenitim načelima koja se u konačnici značajno razlikuju od rješenja predloženih u ovom radu, no s njima dijele slične pretpostavke.

S druge strane, radovi poput [85] istražuju kognitivne i tehničke implikacije uspoređivanja tekstnih programa te predlažu dodatna poboljšanja u tom području. Ovi radovi su uzeti kao nacrt poželjnih značajki kad je riječ o uspoređivanju programa općenito. Ta su rješenja poslužila kao apstraktno nadahnuće te su preuzeti pojedini konceptualni elementi, no jasno da zbog drukčijeg medija njihovi zaključci nisu izravno primjenjivi.

5.2.2 Osnovne značajke i preduvjeti

Na temelju analize prezentirane u prethodnom poglavlju, definirane su osnovne značajke i preduvjeti rješenja za uspoređivanje programa:

1. Uspoređivanje programa mora se u potpunosti odvijati u vizualnoj domeni.
2. Proces usporedbe programa mora biti jednostavan i dio temeljnog radnog procesa s novim jezikom, odnosno treba biti konzistentan te ne odstupati od temeljnog vizualnog prikaza jezika.

3. Rezultati usporedbe programa moraju biti prezentirani na grafički izražajan i intuitivno razumljiv način.
4. Usporedba se odvija na različitim razinama apstrakcije i detalja služeći se pritom specifičnim mehanizmima na svakoj od razina.
    a. Primjerice, ako je modificiran parametar nekog bloka, taj blok mora biti označen na razini glavnog grafa programa, a otvaranjem pregleda bloka označava se specifični parametar koji je izmjenjen, njegova prethodna i trenutna vrijednost.
    b. Kao dodatna mogućnost, izmjenjeni se parametri mogu istaknuti sekundarnom notacijom na samim simbolima blokova na radnoj površini.
5. Programe mora biti moguće superponirati tako da su na radnoj površini istovremeno prikazani blokovi starije i novije inačice. Blokovi različitih inačica razlikuju se drukčijim prikazom grafičkih elemenata. Primjerice, blokovi starije inačice mogu biti prikazani iscrtkanim linijama ili drugom, kontrastnom bojom.
6. Dijelovi programa koji nisu trenutno vidljivi moraju biti označeni na druge načine, dajući do znanja korisniku gdje su se zbile promjene.
7. Programe mora biti moguće uspoređivati dok traje njihovo izvođenje.
8. Programe mora biti moguće uređivati tijekom uspoređivanja.
9. Model uspoređivanja mora biti kompatibilan sa sustavima verzioniranja.
    a. Prvo moguće rješenje koristi integraciju s postojećim alatima te se oslanja na pohranjivanja stanja vizualnog programa u tekstni zapis poput XML-a
    b. Drugo rješenje se oslanja na interni sustav verzioniranja, odnosno pohranjivanje stanja programa u binarnom obliku.

Kao dodatni način uspoređivanja programa, ali izvan opsega ovoga rada, predlaže se mogućnost uspoređivanja izlaza programa u domeni timbralnih atributa.

5.2.3 Model i arhitektura

Uvažavanjem osnovnih značajki i preduvjeta iznesenih u prethodnom poglavlju te na temelju specifičnosti novog jezika, oblikovan je model rješenja.

Model uključuje tri razine rješenja.

Detektiranje promjena

Prva razina rješenja definira unutarnje mehanizme koji omogućuju uspoređivanje jezika, odnosno postupke kojima se otkrivanju promjene između dva programa. Prema istraživanjima

poput [83][85] jasno je da se programi mogu uspoređivati na dva načina. Prvi od njih uključuje uspoređivanje jezika tako što se struktura programa pohrani u vanjskom obliku na datotečnom sustavu, primjerice XML-u, te se zatim korištenjem dobro poznatih alata za uspoređivanje na tekstnoj razini ili razini XML-a izluče razlike u programima. Zatim se na temelju tih usporedbi označavaju elementi u grafičkom sučelju.

Nedostatak ovakvog pristupa je što je teško ostvariti istovremenu mogućnost da se uspoređuju, uređuju i izvode programi budući da je potrebno generirati statičke reprezentacije vizualnih programa. Na neki način, uvodi se korak prevođenja jezika u XML. S druge strane, prednost ovakvog pristupa mogućnost je korištenja vanjskih alata za verzioniranje poput *gita*. Integracija s jezikom je u tom slučaju jednostavna i svodi se na korištenje neke od biblioteka za korištenje funkcionalnosti verzioniranja koje nudi *git*. Korisnik pritom bira inačice programa koje želi usporediti pregledom stabla promjena.

Kod drugog se rješenja programi uspoređuju tako što se obilaze interne strukture podataka kojima je program predstavljen i privremeno pohranjen u radnoj memoriji računala. Ovim pristupom moguće je ostvariti konstantno uspoređivanje tijekom uređivanja i izvođenja programa. Zbog oslanjanja na interne strukture podataka, verzioniranje je također potrebno ostvariti internim mehanizmama. Pritom se stanja programa na zahtjev korisnika ili automatski pohranjuju u tzv. *snapshotovima*. Korisnik zatim može izabrati *snapshotove* koje želi uspoređivati.

Prikazivanje promjena u strukturi programa

Promjene se u jeziku, kao što je spomenuto ranije, prikazuju na različite načine ovisno o razini na kojoj korisnik pregledava program.

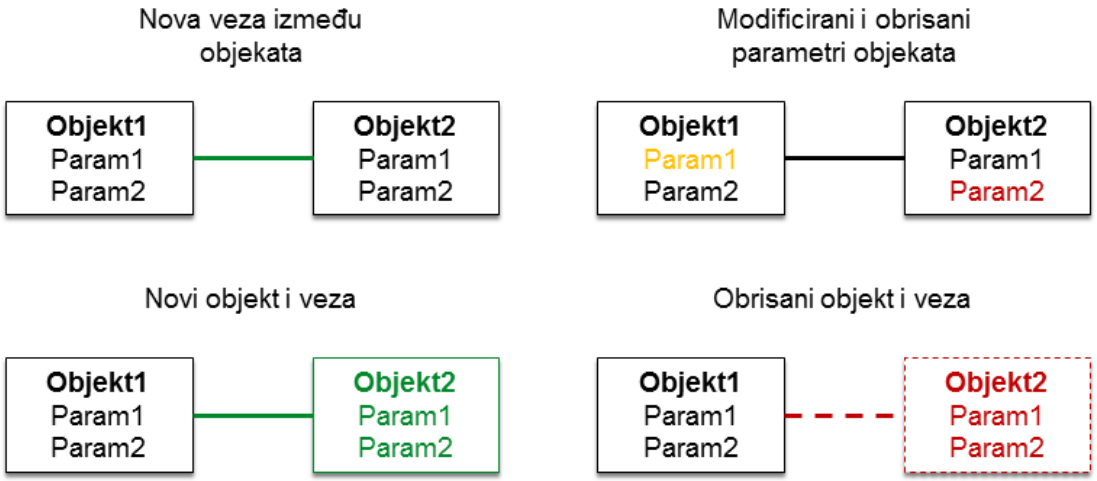
Na razini patcha, odnosno grafa programa, model može podržavati dva načina rada. U prvom se na radnoj površini prikazuju blokovi i stanje novije inačice programa uz dodatno označene i iscrtane razlike u odnosu na stariju inačicu. Ako se primjerice u novijoj inačici promijeni vrijednost parametra jednog bloka te doda novi blok, onda će oba bloka biti vizualno označena uz potencijalnu prezentaciju promjena parametara na samom simbolu bloka u prvom slučaju te jasnim, konzistentnim signaliziranjem da je blok dodan u drugom slučaju.

U drugom se načinu rada istovremeno prikazuju svi blokovi prve i druge inačice programa. U tom je scenariju nužno vizualno razlikovati blokove različitih inačica programa korištenjem grafičkih elemenata poput iscrtkanih linija, sjena i različitih boja. Ovaj pristup

može biti kognitivno zahtjevan za praćenje i dezorijentirajući zbog ograničenja viskoznosti kod vizualnih programa.

U oba načina rada vrijedi da se interakcijom s izmijenjenim elementima korisniku prezentiraju dodatne informacije u obliku kontekstualnih poruka. Te informacije mogu biti ime korisnika koji je promjenu učinio, kad je promjena učinjena itd. Također, u oba se načina rada na ovoj razini označavaju sljedeće promjene (**Slika 5.1**):

1. dodani blokovi
2. obrisani blokovi
3. nove veze među blokovima
4. obrisane veze među blokovima
5. izmjene parametera blokova.



Slika 5.1 Označavanje različitih tipova izmjena na radnoj površini vizualnog programskog jezika

Sljedeća razina prikazivanja razlika među programima odnosi se na značajke pojedinih blokova. Otvaranjem dijaloga koji prikazuju parametre blokova (ili njihovim prikazom na samim grafičkim simbolima blokova) označavaju se pojedini parametri čije su se vrijednosti mijenjale. Korisničkom interakcijom s tim parametrima, primjerice zadržavanjem miša iznad njih, otvara se poruka koja pokazuje promjenu vrijednosti parametra, ime korisnika koji ju je učinio i vrijeme kad se zbila. Potonje se ostvaruje integracijom sa sustavom za verzioniranje.

Konačno, korištenjem dodatnih vizualnih pokazivača i popratnih alata korisnik mora moći na jednostavan način pristupiti promjenama koje nisu u trenutno vidljivom području programa. To se ostvaruje dodatnim prozorom koji sadrži popis svih promjena te oznakama na simbolima potprograma (podpatcheva) koji upućuju da je tijelo potprograma mijenjano.

Posebno je na ovoj razini uspoređivanja potrebno uzeti u obzir specifičnosti timbralnih atributa te na temelju njihovih specifičnosti i konteksta istaknuti pojedine promjene.

Primjerice, značajne promjene atributa mogu značiti prisutnost nenamjernih pogrešaka. Naime, u običajenom radu očekuju se manji relativni pomaci vrijednosti timbralnih atributa.

Uspoređivanje rezultata programa

Iako izlazi iz okvira uspoređivanja vizualnih programa prezentiranog u ovom radu, važno je istaknuti dodatni potencijalni način uspoređivanja rezultata izvođenja programa u domeni timbralnih atributa.

U tom bi se načinu uspoređivanja naglasak stavio na usporedni pregled izlaza programa vizualizacijama audio signala te predstavljanjem istog vrijednostima timbralnih atributa. Ovaj je način vizualiziranja srodan vizualizacijama prikazanima u poglavlju 4.2, no uz dodatno uključivanje usporednog vizualiziranja dva različita programa. Rezultati bi se mogli vizualizirati usporedno u dva prozora s rezultatima prvog i drugog programa ili istovremeno na istom grafu koji bi superponirao rezultate. Pritom bi se za vizualizacije koristilo inverzno mapiranje, odnosno preslikavanje audio signala u timbralne atribute.

U ovoj domeni je moguće uspoređivati i programe koji se svojom implementacijom značajno razlikuju. Zbog složenosti postupaka potrebnih za ostvarenje i evaluaciju ovog načina rada, on izlazi van okvira ove disertacije te ostaje tema za buduća istraživanja.

5.3 Prototip

Kako bi u prototipu predloženog jezika bilo moguće ostvariti alat za uspoređivanje programa, prihvaćena su sljedeća pojednostavljenja i dodatna pravila:

- U tekstnom se prikazu programa detektiraju sve promjene te se, ako korisnik tako želi, izlučuju u zasebnu datoteku. U idealnom slučaju, međukorak koji se oslanja na tekstni prikaz ne bi bio nužan već bi se programi uspoređivali na temeljnu njihovih zapisa u memoriji računala, odnosno korištenjem ugrađenih struktura podataka.
- Zbog načinja na koji je ostvareno uspoređivanje programa i ograničenja jezgre Pure Date, nije moguće istovremeno uspoređivati i uređivati ili izvoditi programe.
- Na temelju detektiranih promjena u vizualnoj se domeni označavaju blokovi koji su mijenjani ili dodani te se ističu modificirane vrijednosti parametara blokova.
- Iako model i implementacija prototipa dopuštaju daljnja proširenja, u vizualnoj se domeni neće označavati obrisani blokovi i obrisane veze.
- Vizualni indikatori parametara blokova pokazuju da je određena vrijednost promijenjena, no ne omogućuju usporedni pregled stare i nove vrijednosti.

- Budući da je paralelni prikaz dva programa koje se uspoređuje kognitivno zahtjevan za korisnika i kao takav manje značajan, odlučeno je da se ovaj način prikaza neće koristiti u prototipu, nego će se sve detektirane promjene prikazivati na konačnoj, novijoj inačici programa.
- U algoritam određivanja promjena ugrađena je granična vrijednost vektora promjena nakon koje se programi proglašavaju međusobno previše različitima te se usporedba ne vizualizira u grafičkom prikazu programa.
- Izuzeta je mogućnost korištenja verzioniranja.
- Nije implementiran modus uspoređivanja rezultata programa, ali je takvo rješenje moguće ostvariti i u samom jeziku Pure Data.

5.3.1 Otkrivanje izmjena u tekstnoj domeni

Uspoređivanje programa novog jezika u tekstnoj domeni izvršava se u nekoliko koraka: preoblikovanjem zapisa (obično sadržanih u datotekama s ekstenzijom .pd) dvaju inačica programa u XML, usporedbom tako dobivenih XML-ova pomoću vanjske biblioteke te generiranjem izlaznog XML-a koji reflektira promjene među inačicama programa. Nakon toga se otkrivene promjene vizualiziraju (što je opisano u poglavlju 5.3.2) čime se dovršava postupak uspoređivanja.

Pretvaranje ugrađenog zapisa u XML

Kao preduvjet otkrivanju izmjena u tekstnoj domeni, bilo je potrebno prilagoditi Pure Datu tako da uz uobičajeni zapis programa generira i XML. Takav je pristup bio nužan zato što je ugrađeni način zapisivanja programa u Pure Dati nestrukturiran te kao takav nije pogodan za usporedbe. Dodatno, korištenjem XML-zapisa omogućeno je korištenje prethodno istraženih algoritama i postojećih biblioteka za uspoređivanje XML-a. Primjer originalnog zapisa dijela patcha koji sadrži dva objekta u Pure Dati:

```
#X obj id101 95 298 bng 15 250 50 0 empty empty empty 17 7 0 10 -262144 -1 -1;
#X obj id102 56 260 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 0 1;
```

Primjer istog zapisa translatiranog u XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<obj id="101" x="95" y="298" type="bng">
	<parameters>
		<param id="1">15</param>
		<param id="2">250</param>
		<param id="3">50</param>
		<param id="4">0</param>
		<param id="5">empty</param>
		<param id="6">empty</param>
		<param id="7">empty</param>
		<param id="8">17</param>
		<param id="9">7</param>
		<param id="10">0</param>
		<param id="11">10</param>
		<param id="12">-262144</param>
		<param id="13">-1</param>
		<param id="14">-1</param>
	</parameters>
</obj>
<obj id="102" x="56" y="260" type="tgl">
	<parameters>
		<param id="1">15</param>
		<param id="2">0</param>
		<param id="3">empty</param>
		<param id="4">empty</param>
		<param id="5">empty</param>
		<param id="6">17</param>
		<param id="7">7</param>
		<param id="8">0</param>
		<param id="9">10</param>
		<param id="10">-262144</param>
		<param id="11">-1</param>
		<param id="12">-1</param>
		<param id="13">0</param>
		<param id="14">1</param>
	</parameters>
</obj>
```

Važno je primjetiti da je objektima u izvornom zapisu dodan identifikacijski broj kako bi se omogućila obostrana veza elemenata u XML-u s elementima u izvornom zapisu. Ta funkcionalnost enumeriranja nije ugrađena u samom jeziku Pure Data, nego su napravljene izmjene unutar jezgrene datoteke „m_class.c“ te na mjestima u kojima se unutarnja struktura podataka zapisuje u datoteku.

Uspoređivanje XML-prikaza programa

Nakon što su programi iz ugrađenog zapisa prebačeni u XML, moguće ih je uspoređivati raznim postupcima za određivanje razlika među dokumentima u XML-u [86][87]. Za potrebe implementacije prototipa nije iznova implementiran neki od prethodno istraženih algoritama već je korištena vanjska biblioteka libxmldiff [88]. Nakon što se kôd programa pripremi u XML-u, korištenje biblioteke vrlo je jednostavno:

```
...
setDefaultXmldiffOptions(diffOptions);
diffXmlFiles(firstProgram, secondProgram, diffOutput, diffOptions);
...
```

Primjer izlaznog XML-a koji sadrži opis promjena na jednom bloku u dvije inačice istog programa:

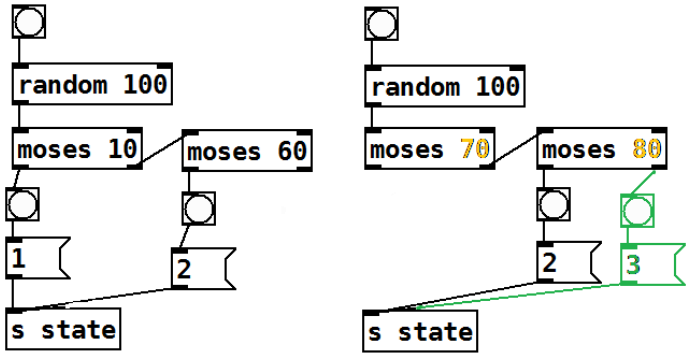
```
<?xml version="1.0" encoding="UTF-8"?>
<obj xmlns:diff="http://www.via.ecp.fr/~remi/soft/xml/xmldiff" id="101"
  x="95|90" y="298" type="bng" diff:status="modified">
  <parameters diff:status="below">
    <param id="1">15</param>
    <param id="2">250</param>
    <param id="3" diff:status="modified">50|52</param>
    <param id="4">0</param>
    <param id="5">empty</param>
    <param id="6">empty</param>
    <param id="7">empty</param>
    <param id="8" diff:status="modified">17|20</param>
    <param id="9" diff:status="modified">7|14</param>
    <param id="10">0</param>
    <param id="11">10</param>
    <param id="12">-262144</param>
    <param id="13">-1</param>
    <param id="14">-1</param>
  </parameters>
</obj>
```

5.3.2 Označavanje izmjena u vizualnoj domeni

Prethodni korak generira izlazni XML s rezultatima usporedbe. Taj se dokument iznova parsira i učitava. Pretragom pomoću *XPatha* po elementima koji sadrže atribut `diff:status="modified"` moguće je izlučiti popis izmijenjenih blokova, veza itd.

```
...
<param id="8" diff:status="modified">17|20</param>
<param id="9" diff:status="modified">7|14</param>
...
```

Sljedeći i posljednji korak jest da se prolaskom kroz sve pronađene izmjene iste označe u grafičkom prikazu jezika. Korištenjem identifikatora pronalaze se odgovarajući objekti, veze i parametri u unutarnjoj strukturi patcha, odnosno programa, te im se mijenjaju parametri poput boje i debljine kako bi se označile promjene u odnosu na program koji je izabran za usporedbu:



Slika 5.2 Originalni odsječak programa (lijevo) i modificirana verzija (desno) s označenim promjenama

Važno je primijetiti da nije moguće činiti daljnje promjene u programu dok je aktivan način pregledavanja razlika programa, već je to potrebno ponovno omogućiti pomoću odgovarajuće tipke u korisničkom sučelju.

5.4 Osvrt

Generalna evaluacija opisanog mehanizma i modela uspoređivanja vizualnih programa u novom jeziku uključena je u ispitivanje samog jezika. Ta je analiza predstavljena u poglavlju 3.9. Model prezentiran u poglavlju 5.2 sveobuhvatan je i optimiran za značajke i očekivane obrasce korištenja novog jezika. On uključuje usporedbe starih i novih vrijednosti parametara po pojedinim blokovima, pregledavanje blokova i veza koje su u međuvremenu izbrisane, slojevito prikazivanje promjena grafičkom superpozicijom različitih inačica programa te istovremeno uređivanje i uspoređivanje programa.

Uključivanjem navedenih aspekata, predloženi model maksimira funkcionalnost uspoređivanja u kontekstu kognitivnih dimenzija. U modelu i arhitekturi novog programskog jezika, mehanizmi uspoređivanja programa bitan su element jer značajno poboljšavaju iskustvo iterativnog razvoja programa te olakšavaju otkrivanje pogrešaka koje su se pojavile između iteracija.

Zbog značajki i ograničenja jezika Pure Date na kojem je temeljen prototip jezika, rješenje prezentirano u 5.3 sadrži podskup funkcionalnosti predloženog modela koji je dovoljan za njegovu evaluaciju. Zbog tehničkih ograničenja i grafičke rudimentarnosti Pure Date, izražajnost prikaza promjena prvenstveno se oslanja na komunikaciju putem boja i debljine crta. U slučaju punog tehničkog ostvarenja jezika, omogućilo bi se vidljivije i jasnije označavanje pogrešaka putem bogatijih mehanizama sekundarne notacije. Konačno, ograničenje prototipa je i to da se programi ne mogu uređivati tijekom usporedbi što je također posljedica implementacije prototipa u Pure Dati.

Stoga je kod potpune implementacije predloženog modela jezika, izvan faze prototipa, od početka razvoja nužno omogućiti bogatiji grafički leksikon te odgovarajuće interne strukture koje bi omogućile usporedbe tijekom izvođenja programa (eng. *runtime*). Takav je pristup nužan preduvjet kako bi se u potpunosti mogao ostvariti model prezentiran ranije u ovom poglavlju.

6 Diskusija

Model vizualnog programskog jezika temeljenog na toku timbralnih atributa, uključujući ostale predložene doprinose, smješten je u interdisciplinarni kontekst. U tom se smislu novi jezik nalazi na sjecištu vizualnog programiranja, sinteze zvuka, računalom podržanog skladanja i suvremenih umjetničkih praksi, posebice u glazbi. Istraživanje predstavljeno u ovom radu stoga je, osim kroz znanstveno-tehnički aspekt, potrebno primarno analizirati kroz prizmu utjecaja tehnologije u umjetničkim praksama.

6.1 Generalni utjecaj na vizualne programske jezike

S obzirom da je svaki vizualni programski jezik specijaliziran, pa tako i jezik predstavljen u ovom radu, glavna poboljšanja i doprinosi nisu toliko izraženi u općenitom području vizualnog programiranja. Unatoč toj činjenici, važno je razmotriti a) način na koji se sam jezik uklapa u postojeća rješenja, b) obrasce putem kojih se predstavljena rješenja mogu poopćiti i primijeniti na druge domene te c) važnost i provjerljivost korištenih metodologija, kako u oblikovanju tako i u analizi vizualnih jezika.

6.1.1 Pozicija novog jezika u području

Kao što je predstavljeno kroz kvalitativnu i kvantitativnu analizu te usporedbama s postojećim sličnim jezicima, predloženi model novog jezika donosi bitna paradigmatska poboljšanja dok istovremeno zadržava poželjne i provjerene elemente drugih rješenja poput Pure Date, Max/MSP-a i OpenMusica. U tom smislu, novi je jezik evolucija, a ne revolucija.

Ta ga činjenica čini lako usvojivim i razumljivim onim glazbenicima koji su se prethodno susretali s vizualnim programiranjem u glazbi. Istovremeno, zbog oslanjanja na intuitivno razumljive timbralne atribute, jezik je bliži i potpunim početnicima u području. Oslanjanjem na timbralne atribute rješavaju se mnogi problemi prisutni kod drugih jezika, a koji su najčešće vezani uz nerazumijevanje i dodatno učenje potrebno da bi se usvojila paradigma audio signala.

Dodatno, uz samu jezgru jezika predložena su rješenja čiji je cilj umanjiti utjecaj negativnih značajki koje su inherentne vizualnoj domeni. Funkcionalnost uspoređivanja vizualnih programa tako pomaže kod pisanja kompleksnih programa, ali i olakšava otkrivanje pogrešaka. Grafički elementi za unos vrijednosti timbralnih atributa ergonomski su i lako razumljivi te omogućuju jednostavnu manipulaciju zvuka i dobivanje željenih izlaza. Konačno, elementi koji izlučuju timbralne atribute iz generiranog zvuka te ga predstavljaju u

vizualnoj domeni omogućuju izražajni i bogati pregled sintetiziranih zvukova i stvorene glazbe.

S druge strane, promatramo li potencijalne mane, vizualni jezik zadržava boljke inherentne većini vizualnih programskih jezika, poput problema s prikazom i izostankom lako dostupnih manipulacija složenih programa, poteškoće prilikom preuređivanja programa itd. Njihovo rješavanje nije usko vezano uz područje glazbe te stoga nije bilo fokus ovoga rada.

Uvaživši gore navedene pozitivne i negativne aspekte modela novoga jezika i bez obzira što on nudi bitne prednosti u odnosu na postojeća rješenja, uspjeh u potpunosti implementiranog jezika ovisio bi također o drugim dimenzijama samog jezika poput uspješnosti tehničke realizacije (moderni izgled sučelja, web-inačica), razgranatosti diseminacijskih kanala (korištenje u nastavnim procesima na akademijama, zajednice amaterskih i profesionalnih glazbenika), dostupnosti materijala za učenje (upute u pisanom i multimedijskom obliku) itd.

Konačno, u budućim istraživanjima vrijedilo bi razmotriti mogućnost da se jezik ostvari kao nadogradnja nekog od postojećih jezika otvorenoga kôda poput Pure Date ili OpenMusica. Takvim bi se pristupom postigao kompromis između bržeg dosega nauštrb nepotpune realizacije svih ideja i principa iznesenih u modelu jezika.

6.1.2 Poopćavanje predstavljenih rješenja

Osim kroz usku domenu glazbe, neka od rješenja predstavljenih u ovom istraživanju mogu poslužiti kao nadahnuće i predložak za buduća istraživanja u drugim područjima.

Primjerice, funkcionalnost vizualnog uspoređivanja jezika u ovom je radu primijenjena isključivo na novi jezik za sintezu zvuka i stvaranje glazbe, no isti bi se principi mogli prenijeti i primijeniti u većini sličnih jezika u domeni glazbe, ali i drugim domenama.

Primjerice, na način na koji su u ovom radu u toku podataka audio signali zamijenjeni timbralnim atributima, u drugim bi se domenama kao temeljni oblik podataka mogli koristiti neki za pojedinu domenu specifični oblici podataka.

Nadalje, predstavljeno rješenje usmjereno je na uočavanje promjena timbralnih atributa kao ulaznih parametara objekata jezika i na razlike u topografiji programa te je stoga usko vezano uz sintaksu i semantiku. Unatoč tome, međukoraci korišteni za ostvarivanje te funkcionalnosti – odnosno preslikavanje unutarnje strukture programa u XML te uspoređivanje u toj domeni – kao i temeljni principi poput prikazivanja promjena na grafičkim simbolima objekata mogu poslužiti općenito. Ipak, konačnu funkcionalnost treba baždariti prema potrebama domene te bi stoga slično rješenje u, primjerice, domeni

elektroničkih sklopova bilo prilagođeno usporedbama iznosa napona, struje i sličnih vrijednosti na otpornicima, kondenzatorima i sličnim elementima.

Konačno, principi predstavljeni kod elemenata za unos timbralnih atributa i njihovu vizualizaciju izlučivanjem iz rezultata programa moguće je također poopćiti, no uvijek uvažavajući specifičnosti domene. Te će specifičnosti uvjetovati odabir načina prikaza i elemenata za unos (tipkala, grafovi) kao i vizualizacije (radar dijagrami, jednostavne skale).

6.1.3 Korištene metodologije

U ovom su radu korištene različite i raznorodne znanstvene metodologije kroz sve faze razvoja jezika. Dok su neke od njih – poput kognitivnih dimenzija i izrade prototipa za validiranje modela jezika – relativno uobičajene u području, druge se – poput kvalitativnih i kvantitativnih analiza jezika – rijetko susreću te je njihova primjena značajno doprinijela provjerljivosti i vrijednosti validiranja odluka prilikom oblikovanja jezika.

Analiza kognitivnim dimenzijama tako je uz potporu kvalitativne i kvantitativne analize rezultata anketa prvo korištena kako bi se odredila područja vizualnog programiranja u glazbi u kojima su potrebna poboljšanja, a zatim za validaciju stvorenog modela novog jezika. Pritom su identificirane specifične dimenzije koje je novi jezik unaprijedio. Time je ostvarena sljedivost i provjerljivost poboljšanja.

Kvalitativna i kvantitativna analiza rezultata ankete kojima su se prikupljala mišljenja korisnika vizualnih programskih jezika u glazbi korištena je, osim kao potpora analizi kognitivnim dimenzijama, i zasebno kako bi se izlučile korisnicima najbitnije značajke. Konzultiranjem tih rezultata i ponavljanjem ankete za novi jezik dodatno su provjerene i validirane hipoteze o poželjnosti i valjanosti predloženih rješenja.

Konačno, prototip jezika korišten je u sprezi s anketom i intervjuima s naprednim korisnicima kako bi se ukratko demonstrirale značajke u stvarnom svijetu i u kontekstu dobro poznatog jezika Pure Date.

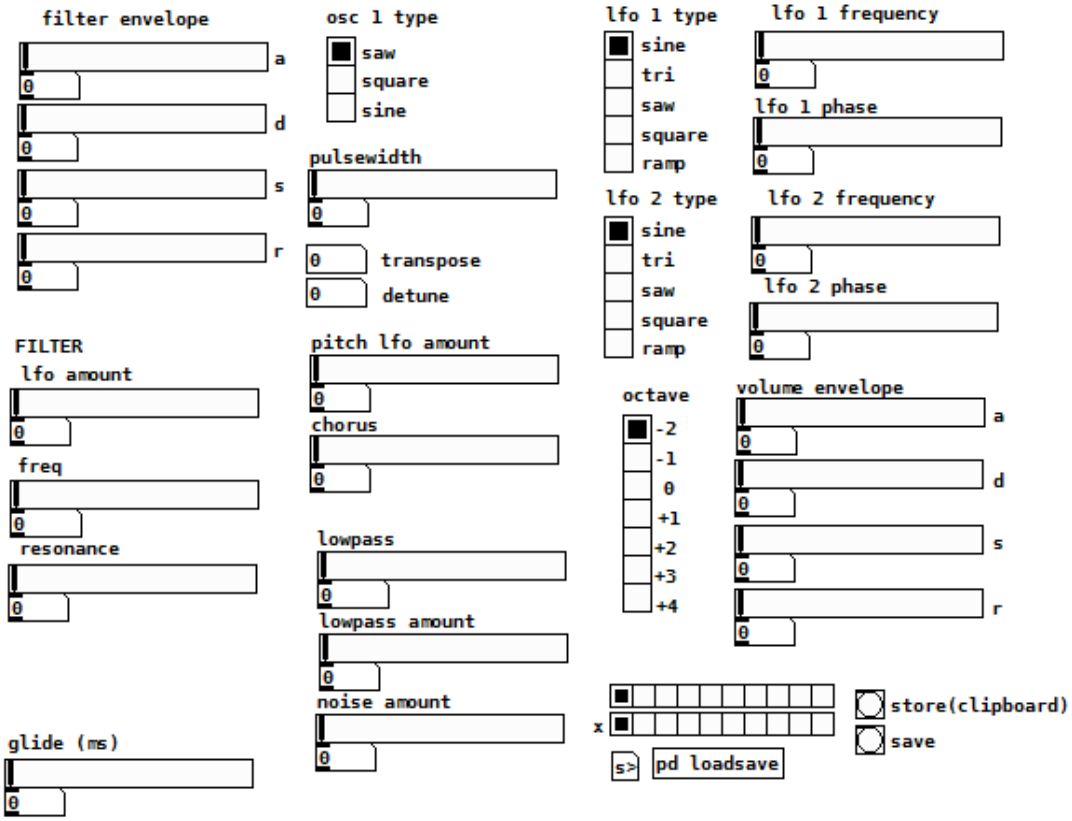
Uz opisane pojedinačne elemente metodologije koja je uključivala od tehničkih do kognitivnih mjerila, pristup prilikom oblikovanja jezika bio je prvenstveno holistički i sveobuhvatan. Tako su u svim fazama istraživanja provođeni intervjui s ekspertima iz područja te je na temelju njih usmjeravan tijek istraživanja.

Buduća istraživanja u području vizualnih programskih jezika, neovisno o domeni primjene, mogu se stoga poslužiti istom ili sličnom metodologijom koja se pokazala uspješnom u a) izlučivanju potreba i zahtjeva korisnika te b) validiranju predloženih

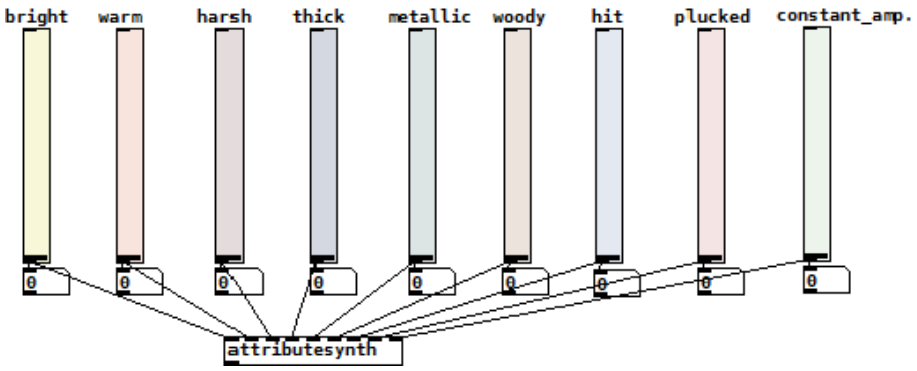
inovativnih rješenja. Dodatno, rezultati kvantitativne analize jezika moguće je izravno koristiti za daljnja slična istraživanja u području.

6.2 Korištenje timbralnih atributa

Jedna od temeljnih pretpostavki za ostvarivanje vizualnog programskog jezika temeljenog na atributima jest postojanje mehanizama preslikavanja timbralnih atributa u audio signale te izlučivanje vrijednosti timbralnih atributa iz postojećih signala. Implikacije tog dijela istraživanja opširnije su opisane u radovima [22] i [19], a njegovi su rezultati i doprinosi značajni i primjenjivi izvan svijeta vizualnog programiranja.



Slika 6.1 Sučelje za sintetizator u domeni audio signala



Slika 6.2 Sučelje za sintetizator u domeni timbralnih atributa

Važno je pritom istaknuti način na koji timbralni atributi olakšavaju temeljno upravljanje sintetizatorima ostvarenima unutar vizualnog programskog jezika. **Slika 6.1** tako prikazuje kako se sintetizatorom ostvarenim u Pure Dati upravlja pomoću uobičajenih parametara u domeni audio signala. Nasuprot nje, **Slika 6.2** demonstrira kako se istim sintetizatorom upravlja pomoću timbralnih atributa. Iz primjera je vidljivo da je upravljanje timbralnim atributima puno jednostavnije, razumljivije i bliže glazbenicima. Također, takav pristup riješava problem međusobne ovisnosti i nejasnih efekata uobičajenih parametara sintetizatora koji od glazbenika traže razumijevanje i poznavanje postupaka obradbe audio signala.

6.3 Glazbenici i kreativnost

Kao što je demonstrirano analizama u prethodnim poglavljima, proširenjem uobičajenih paradigmi vizualnog programiranja u glazbi te zamjenom audio signala timbralnim atributima kao glavnim nositeljem informacija, poboljšava se ukupna ergonomija i interaktivnost vizualnog programiranja. Anketiranjem korisnika, kvalitativnom analizom i provjerom kognitivnim dimenzijama pokazuje se da značajke novog jezika unaprjeđuju lakoću korištenja jezika, intuitivnost i smanjuju vjerojatnost činjenja pogrešaka.

No ti aspekti ne govore o krajnjem utjecaju korištenja novog jezika na kreativne procese kod glazbenika [42]. Za takvu je analizu potrebno realizirati punu implementaciju jezika te je prepustiti korisnicima na dulje razdoblje. Iako prezentirane preliminarne analize ukazuju na pozitivne konotacije korištenja novog jezika, umjetnički proces izrazito je stohastički i generalno neograničen čvrstim faktorima. Iz tog je razloga teško ocjenjivati i kvantitativno vrednovati interakcije, pozitivne i negativne utjecaje u sprezi s tehnologijom.

Primjerice, neki od anketiranih i intervjuiranih glazbenika svojom su prilagodbom svijetu audio signala i kreativnim pristupom postojećim jezicima postigli neočekivane, ali dobrodošle umjetničke rezultate. Drugim riječima, umjetnički i estetski vrijedni rezultati uvjetovani su upravo njihovim neiskustvom kad su u pitanju audio signali. Korištenjem timbralnih atributa sužava se prostor za takve inovativne intervencije, neočekivane ishode i "hakiranje", no istovremeno se olakšava izražavanje glazbenih misli čime se otvaraju nove mogućnosti u umjetničkom radu.

Stoga bi, kao u demonstriranom prototipu jezika, trebalo razmotriti mogućnost da se zadrži paralelni tok timbralnih atributa i audio signala te omogući njihovo miješanje. Takvim pristupom istovremeno glazbenicima i skladateljima olakšavamo temeljno i početno korištenje jezika, ali ostavljamo mogućnost za naprednije i detaljnije intervencije u programima manipulacijama audio signala. Konačno, u budućim bi istraživanjima valjalo

razmotriti proširenja toka podataka i drugim opisnim, visoko izražajnim vrijednostima osim timbralnih atributa, a vezanih uz shvaćanje glazbe iz perspektive glazbenika i skladatelja.

7 Zaključak

Oblikovanjem i validacijom novog vizualnog programskog jezika za sintezu zvuka temeljenu na timbralnim atributima i stvaranje glazbe temeljeno na manipulacijama u vremenu te popratnih funkcionalnosti ostvarene su pretpostavke postavljene na početku istraživanja. Te su pretpostavke uključivale realizaciju značajnog poboljšanja u domeni vizualnog programiranja u glazbi, rješavanje nekih tipičnih problema vizualnog programiranja i generalnu evoluciju jezika u domeni glazbe po pitanju iskustva korištenja i prilagođenosti glazbenicima i skladateljima.

U radu su, osim detaljnih obrazloženja pojedinih elemenata jezika i temeljnih mehanizama njegovih ostvarenja, detaljno i rigozorno dokumentirani, verificirani i validirani svi koraci stvaranja novog jezika. Pritom su značajni doprinosi dobiveni u svakoj od faza oblikovanja jezika, od rezultata kvantitativne analize postojećih rješenja do mehanizama za preslikavanje timbralnih atributa i njihovo korištenje u paradigmi toka podataka.

Iako je fokus rada uglavnom stavljen na domenu glazbe i multimedije, a ne na rješavanje problema u vizualnom programiranju generalno, neka od predstavljenih rješenja moguće je poopćiti. Pritom se tu posebno ističu metodologije korištene u pojedinim fazama rada, mehanizmi uspoređivanja vizualnih programa te poboljšanja sučelja za unos i vizualizaciju rezultata programa. Općenito je principe ovoga istraživanja i rada moguće primijeniti i u drugim domenima poput modeliranja elektroničkih sklopova.

Zbog interdisciplinarnosti područja i složenosti pojedinih evaluacija, te kako bi se ostvareni jezik promotrio kroz više perspektiva, tijekom istraživanja korišteni su načini verificiranja i validiranja rezultata iz različitih područja znanosti. Osim tehničke validacije ostvarivanjem minimalnog prototipa u Pure Dati, validacija se najvećim svojim dijelom oslanjala na postupke i principe iz psihologije. Korištene su ankete i intervjui s korisnicima te kvantitativna i kvalitativna analiza tako dobivenih rezultata i prikupljenih podataka. Te su analize producirale u području dosad nepostojeće rezultate, a koji bi mogli usmjeriti njegov daljnji razvoj. Time doprinos ovog rada nadilazi sam model jezika.

8 Popis literature

[1] Whitley, K. N., "Visual programming languages and the empirical evidence for and against", J. Vis. Lang. Comput., Vol. 8, No. 1, February 1997, str. 109-142.

[2] Roads, C., "The Computer Music Tutorial", MIT Press, Cambridge, 1996.

[3] Blackwell, A. F., Collins, N., "The Programming Language as a Musical Instrument", Proceedings of PPIG 2005, Brighton, United Kingdom, 2005., str. 120-130.

[4] Green, T. R. G., Petre, M., "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework", Journal of visual languages and computing, Vol. 7, No. 2, June 1996, str. 131-174.

[5] Shneiderman, B., "Direct manipulation: A step beyond programming languages", Computer, Vol. 8, August 1998, str. 57-69.

[6] Burnett, M. M., "Visual programming", Encyclopedia of Electrical and Electronics Engineering, John Wiley & Sons, Inc., New York, 1999.

[7] Lewis, C., Olson, G., "Can principles of cognition lower the barriers to programming?", Empirical Studies of Programmers: Second Workshop, Ablex Publishing Corp., Norwood, USA, 1987.

[8] Scaffidi, C., Shaw, M., Myers, B., "Estimating the Numbers of End Users and End User Programmers", Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, Dallas, USA, 1995, str. 207-214.

[9] Browne, J. C., Hyder, S. I., Dongarra, J., Moore, K., Newton, P., "Visual programming and debugging for parallel computing", IEEE Parallel & Distributed Technology: Systems & Technology, Vol. 3, No. 1, 1995, str. 75-83.

[10] Takatsuka, M., Gahegan, M., "GeoVISTA studio: a codeless visual programming environment for geoscientific data analysis and visualization", Computers & Geosciences, Vol. 28, No. 10, December 2002, pp. 1131-1144.

[11] Myers, B. A., Ko, A. J., "The past, present and future of programming in HCI", Proceedings of Human-Computer Interaction Consortium (HCIC'09), Winter Park, Colorad, USA, 2009.

[12] TIOBE Software, "TIOBE Index for January 2019", dostupno na: https://www.tiobe.com/tiobe-index/ (siječanj, 2019.)

[13] Myers, B. A., "Taxonomies of visual programming and program vizualization", Journal of Visual Languages and Computing, Vol. 1, No. 1, 1990, str. 97-123.

[14] Shu, N. C., "Visual programming", Van Nostrand Reinfold Company, New York, 1988.

[15] Hu, C. H., Wang, F. J., "Towards a practical visual object-oriented programming environment: desirable functionalities and their implementation", Journal of Information Science and Engineering, Vol. 15, No. 4, 1989, str. 585-614.

[16] Pošćić, A., Kreković, G., Butković, A., "Desirable Aspects of Visual Programming Languages for Different Applications in Music Creation", Proceedings of the Sound and Music Computing Conference, Maynooth, Ireland, 2015.

[17] Pošćić, A., Kreković, G., „Ecosystems of Visual Programming Languages for Music Creation: A Quantitative Study“, Journal of the Audio Engineering Society, 2018., DOI: https://doi.org/10.17743/jaes.2018.0028

[18] Pošćić, A., Kreković, G., „The Frailty of Formal Education: Visual Paradigms and Music Creation“, Proceedings of the Audio Mostly 2017, London, Ujedinjeno Kraljestvo, 2017.

[19] Kreković, G., Pošćić, A., Petrinović, D., “An Algorithm for Controlling Arbitrary Sound Synthesizers Using Adjectives”, Journal of New Music Research, 2016.

[20] Pošćić, A., Kreković, G., “Controlling a Sound Synthesizer Using Timbral Attributes”, Proceedings of the Sound and Music Computing Conference, Stockholm, Sweden, 2013., str. 467-472.

[21] Kreković, G., Pošćić, A., “Shaping Microsound Using Physical Gestures”, Proceedings of the International Conference on Computation, Communication, Aesthetics and X, Glasgow, UK, 2015.

[22] Kreković, G., "Postupci intuitivnoga upravljanja sintezom zvuka", doktorski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, Hrvatska, 2016.

[23] Burlet, G., Hindle, A., "An empirical study of end-user programmers in the computer music community", Proceedings of the 12th Working Conference on Mining Software Repositories, Florence, Italy, May 2015, str. 292-302.

[24] Collins, M., “Professional Guide to Audio Plug-ins and Virtual Instruments”, Focal Press, Burligton, 2003.

[25] Puckette, M., “Pure Data”, Proceedings of the International Computer Music Conference, Hong Kong, China, 1996.

[26] Puckette, M., "Max at seventeen", Computer Music Journal, Vol. 26, No. 4, 2002, str. 31-43.

[27] Puckette, M., "The patcher", Proceedings of the 1986 International Computer Music Conference, San Francisco, USA, 1986.

[28] Cycling '74, „Cycling '74: Max 8“, dostupno na: https://cycling74.com/support/faq (siječanj, 2019.)

[29] Bresson, J., Agon, C., Assayag, G., "OpenMusic–visual programming environment for music composition, analysis and research", Proceedings of the 19th ACM international conference on Multimedia, Scottsdale, USA, 2011.

[30] Scaletti, C., “Composing Sound Objects in Kyma”, Perspectives of New Music, 1989. str. 42-69.

[31] Walker, M., “NI Reaktor 5: Virtual Modular Synth”, Sound On Sound, September 2005.

[32] Russ, M., “Sound Synthesis and Sampling”, Focal Press, Oxford, 2004.

[33] Reid, G., “The Secret of the Big Red Button”, Sound On Sound, July 2004.

[34] Chowning, J., “The Synthesis of Complex Audio Spectra by Means of Frequency Modulation”, Journal of the Audio Engineering Society, Vol. 21, No. 7, 1973., str. 526-534.

[35] Miranda, E., “Computer Sound Design: Synthesis Techniques and Programming”, Focal Press, Oxford, 2002.

[36] Moog, R., “A Voltage-Controlled Low-Pass High-Pass Filter for Audio Signal Processing”, Audio Engineering Society Convention 17, 1965.

[37] Cook, P. R., “Real Sound Synthesis for Interactive Applications”, CRC Press, 2003.

[38] Kleczowski, P., “Group Additive Synthesis”, Computer Music Journal, Vol. 13, No. 1, 1989., str. 12-20.

[39] Smith, J. O., “Physical Modeling Synthesis Update”, Computer Music Journal, Vol. 20, No. 2, 1996., str. 44-56.

[40] Rodet, X., “Time-Domain Formant-Wave-Function Synthesis”, Computer Music Journal, Vol. 8, No. 3, 1984., str. 9-14.

[41] Arfib, D.,“Digital Synthesis of Complex Spectra by Means of Multiplication of Non-Linear Distorted Sound Waves”, Journal of the Audio Engineering Society, No. 27, 1979., str. 757-768.

[42] Roads, C., “Composing Electronic Music: A New Aesthetic”, Oxford University Press, New York, 2015.

[43] Drake, G., “Assessment of Timbre Using Verbal Attributes”, Proceedings of the Conference on Interdisciplinary Musicology, Canada, 2005.

[44] Grey, J. M., Gordon, J.W., “Perceptual Effects of Spectral Modifications on Musical Timbres”, The Journal of the Acoustical Society of America, Vol. 63, No. 5, 1978., str. 1493-1500.

[45] Kendall, R. A., Carterette, E. C. “Verbal Attributes of Simultaneous Wind Instrument Timbres: I. von Bismarck's Adjectives”, Music Perception, 1993., str. 445-467.

[46] Clement, R., “Automatic Synthesiser Programming”, Proceedings of the International Computer Music Conference, Universitiy of Huddersfield, UK, 2011., str. 155-158.

[47] Gounaropoulos, A., Johnson, C., “Synthesising Timbres and Timbre Changes from Adjectives/Adverbs” in P. Collet et al. Applications of Evolutionary Computing, Springer-Verlag, Berlin, 2006.

[48] Miranda, E., “An Artificial Intelligence Approach to Sound Design”, Computer Music Journal, Vol. 19, No. 2, 1995., str. 59–75

[49] Keim, D., Qu, H., Ma, K.L., "Big-data visualization", IEEE Computer Graphics and Applications, Vol. 33, No. 4, 2013, str. 20-21.

[50] Lobdell, B.E., Allen, J.B., "A model of the VU (volume-unit) meter, with speech applications", The Journal of the Acoustical Society of America, Vol. 121, No. 1, 2007, str. 279-285.

[51] Piszczalski, M., Galler, B., Bossemeyer, R., Hatamian, M., Looft, F., "Performed music: analysis, synthesis, and display by computer", Journal of the Audio Engineering Society, Vol. 29, No. 1/2, 1981, str. 38-46.

[52] Adams, N., "Visualization of musical signals", Analytical Methods of Electroacoustic Music, Routledge, London, 2016.

[53] Wagner, M.G., Carroll, S., "Deepwave: Visualizing music with VRML", Proceedings of the Seventh International Conference on Virtual Systems and Multimedia, Berkeley, CA, USA, 2001, str. 590-596.

[54] Uehara, M., Itoh, T., "Pop music visualization based on acoustic features and chord progression patterns applying dual scatterplots", Proceedings of Sound and Music Computing Conference (SMC2015), Maynooth, Ireland, 2015, str. 43-48.

[55] Stables, R., De Man, B., Enderby, S., Reiss, J.D., Fazekas, G., Wilmering, T., "Semantic description of timbral transformations in music production", Proceedings of the 2016 ACM on Multimedia Conference, Amsteram, Netherlands, 2016, str. 337-341.

[56] Antoine, A., Williams, D., Miranda, E.R., "Towards a timbral classification system for musical excerpts", Proceedings of the 2nd AES Workshop on Intelligent Music Production, Birmingham City, UK, 2016.

[57] James, S. "A classification of multi-point spectral sound shapes", Proceedings of the Sonic Environments Conference, Brisbane, Australia, 2016, str. 57-64.

[58] McKee, S., Nelson, N., Sarma, A.. Dig, D., "Software practitioner perspectives on merge conflicts and resolutions", Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 2017, str. 467-478.

[59] Nedvěd, M., Vrba, P.. Obitko, M., "Tool for visual difference display of programs in IEC 61131-3 ladder diagrams", Proceedings of the International Conference on Industrial Technology (ICIT), Seville, Spain, 2015, str. 2994-2999.

[60] Golin, E. J., Reiss, S. P., "The specification of visual language syntax", Journal of Visual Languages & Computing, Vol. 1, No. 2, 1990, str. 141-157.

[61] Erradi, M., Frasson, C., "Interaction with IBS", Advanced Computer Graphics, Springer, Tokyo, 1986.

[62] Engels, G., Hausmann, J. H., Heckel, R., Sauer, S., "Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML", Proceedings of the International Conference on the Unified Modeling Language, York, UK, 2000, str. 323-337.

[63] Brown, A., Brown, A. R., "Computers in music education: Amplifying musicality", Routledge, London, 2012.

[64] Field, A., "Discovering statistics using IBM SPSS statistics", SAGE Publications Ltd, Thousand Oaks, CA, USA, 2013.

[65] Kutar, M., Britton, C., Barker, T., "A comparison of empirical study and cognitive dimensions analysis in the evaluation of UML diagrams", Proceedings of the 14th Workshop of the Psychology of Programming Interest Group (PPIG 14), London, UK, 2002.

[66] Chung, B.W., "Multimedia Programming with Pure Data: A comprehensive guide for digital artists for creating rich interactive multimedia applications using Pure Data", Packt Publishing Ltd, Birmingham, UK, 2013.

[67] Cycling '74, "bpatcher Reference", dostupno na: https://docs.cycling74.com/max5/refpages/max-ref/bpatcher.html (siječanj, 2019.)

[68] Bresson, J., Agon, C., Assayag, G., "Visual lisp/clos programming in openmusic", Higher-Order and Symbolic Computation, Vol. 22, No. 1, 2009, str. 81-111.

[69] Hetrick, M.L.S., "Modular Understanding: A Taxonomy and Toolkit for Designing Modularity in Audio Software and Hardware", doktorski rad, University of California, Santa Barbara, SAD, 2017.

[70] Wishart, T., "On sonic art", Routledge, London, UK, 1996.

[71] Kreković, G., Petrinović, D., "Intelligent Exploration of Sound Spaces Using Decision Trees and Evolutionary Approach", Proceedings of the International Computer Music Conference joint with Sound and Music Computing Conference, Athens, Greece, 2014., str. 1263–1270.

[72] Kreković, G., Petrinović, D., "Automated Control of Sound Synthesis in Live Musical Performances Using Modified Online Time Warping", Proceedings of 54th International Symposium ELMAR, Zadar, Croatia, 2012.

[73] Jensen, K.,"Timber Models of Musical Sounds", doktorski rad, Department of Computer Science, University of Copenhagen, 1999.

[74] McDermott, J., "Evolutionary Computation Applied to the Control of Sound Synthesis", doktorski rad, University of Limerick, Ireland, 2008.

[75] Eldridge, A., Kiefer, C., "The self-resonating feedback cello: interfacing gestural and generative processes in improvised performance", Proceedings of New Interfaces for Music Expression 2017, Aalborg, Denmark, 2017, str. 25-29.

[76] Cádiz, R., Kendall, G., "Fuzzy Logic Control Tool Kit: Real-Time Fuzzy Control for Max/MSP and Pd", Proceedings of the International Computer Music Conference, New Orleans, 2006.

[77] Cingolani, P., Alcalá Fernández J., "jFuzzyLogic: A Robust and Flexible Fuzzy-Logic Inference System Language Implementation", Proceedings of the 2012 IEEE International Conference on Fuzzy Systems, Brisbane, 2012., str. 1-8.

[78] Oracle, "Java Native Interface", dostupno na: https://docs.oracle.com/javase/10/docs/specs/jni/index.html (siječanj, 2019.)

[79] Vittek, M., Borovansky, P., Moreau, P.-E., "A Simple Generic Library for C, in Reuse of Off-the-Shelf Components", Proceedings of the 9th International Conference on Software Reuse, Turin, 2006., str. 423-426.

[80] Leitão, A., Santos, L., Lopes, J., "Programming languages for generative design: a comparative study", International Journal of Architectural Computing, 2012, Vol. 10, No. 1, str. 139-162.

[81] Ichikawa, T., Jungert, E., Korfhage, R.R., "Visual languages and applications", Springer Science & Business Media, Berlin, Germany, 2013.

[82] Wikipedia, "Comparison of file comparison tools", dostupno na: https://en.wikipedia.org/wiki/Comparison_of_file_comparison_tools (siječanj, 2019.)

[83] Hashimoto, M., Mori, A., "Diff/TS: A tool for fine-grained structural change analysis.", Proceedings of the 15th Working Conference on Reverse Engineering, Antwerp, Belgium, 2008., str. 279-288.

[84] Jetley. R., "An approach for comparison of IEC 61131-3 graphical programs", Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 2013.

[85] Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E. "Comparison and Evaluation of Clone Detection Tools", IEEE Transactions onSoftware Engineering, Vol. 33, No. 9, September 2007, str. 577-591.

[86] Zhang, S., Dyreson, C., Snodgrass, R.T., "Schema-less, semantics-based change detection for XML documents", Proceedings of the International Conference on Web Information Systems Engineering, Brisbane, Australia, 2004., str. 279-290.

[87] Peters, L., "Change detection in XML trees: a survey", Proceedings of the 3rd Twente Student Conference on IT, Twente, Netherlands, 2005.

[88] Peyronnet, R., "a simple library to diff XML files", dostupno na: https://github.com/rpeyron/libxmldiff (siječanj, 2019.)

Prilog A – analiza kognitivnim dimenzijama

U nastavku slijedi detaljna analiza postojećih jezika po kognitivnim dimenzijama.

Pure Data

Uz opis jezika koji je dan u poglavlju 2.1.1, za analizu Pure Date važno je spomenuti neke tehničke detalje. Najvažniji od njih je taj da je Pure Data jezik temeljen na toku podataka, što znači da podatci svojim „kretanjem“ kroz graf programa pokreću računalne operacije u pojedinim blokovima koji čine sam program. Pure Data se oslanja na dvije vrste toka podataka - jedan je tok audio signala stvorenih i obrađenih na određenoj frekvenciji uzorkovanja, a drugi je tok kontrolnih signala. Na svaki skup od 64 audio uzoraka dolazi jedan kontrolni signal. Podržani tipovi za kontrolne signale su brojevi s pomičnim zarezom (eng. *float*), simboli (eng. *symbols*) te pokazivači na strukture podataka koje mogu sadržavati proizvoljan broj atomskih tipova podataka. Dodatno, kolekcije podataka moguće je pohraniti u liste, polja, grafove i tablice.

Programi razvijeni u Pure Dati često se nazivaju *patches*, a njihovi su temeljni elementi objekti, poruke, brojevi i komentari. Objekti su funkcionalni elementi koji mogu biti jednostavni (matematičke operacije i općenite funkcionalnosti poput kontrole toka) ili vrlo složeni (obrada signala). Veze među objektima ostvaraju se njihovim ulazima i izlazima, a samim funkcijama sadržanima u objektima upravlja se ulaznim argumentima. Neovisne objekte koji nisu dio samog jezika, već su razvijeni od treće strane, nazivamo *externals*.

Gradijent apstrakcije

Pure Data podržava nekoliko enkapsulacijskih mehanizama koji smanjuju složenost i poboljšavaju čitljivost kôda. Segmenti kôda mogu se umetnuti u apstrakcije i potprograme (eng. *subpatches*) koji, kao i atomski blokovi, imaju ulaze i izlaze za komunikaciju s drugim dijelovima programa. Iako su sadržaji potprograma i apstrakcija obično sakriveni, svaki od njih može se otvoriti u zasebnom prozoru, a pojedini elementi iz enkapsuliranog bloka mogu se izravno prikazati u roditeljskom patchu. Potonje se najčešće koristi kako bi se razni elementi za korisnički unos (poput gumba ili klizača) te elementi za prikaz podataka (poput numeričkog ili grafičkog prikaza) eksponirali na najvišoj razini te time poboljšali korisnost temeljnog, roditeljskog patcha. Uzevši u obzir ove značajke, moguće je zaključiti da je Pure Data dobro dizajniran jezik po pitanju podrške za rekurzivno apstrahiranje kôda uz zadržavanje iste programske paradigme na svim razinama apstrakcije.

Bliskost preslikavanja

Tok podataka u Pure Dati izgledom podsjeća na arhitekturu ranih analognih sintetizatora. Možemo reći da u tom smislu sami blokovi u jeziku odgovaraju modulima sintetizatora, dok veze među blokovima predstavljaju kabele kojima su se prenosili signali. Mnogi temeljni objekti u jeziku nude funkcionalnosti generiranja i obrade signala izravno preslikane iz analognoga svijeta poput raznih oscilatora. Dodatno, klizači i tipke koji su prisutni u grafičkom sučelju jezika često izgledom i ponašanjem odgovaraju klizačima i tipkama na fizičkim sintesajzerima i uređajima koji su poznati glazbenicima. Konačno, razdvojenost toka audio signala i kontrolnih signala također je preuzeta iz svijeta analognih sintetizatora.

Ipak, Pure Data nudi znatnu fleksibilnost koja nadilazi samo stvaranje modularnih softverskih sintetizatora i audio efekata. Iako u tom smislu tok podataka i operacije obrade signala blisko prate kognitivni model razumljiv glazbenicima, neki drugi elementi jezika znatno odudaraju od tog principa. Primjerice, složene mogućnosti upravljanja i usmjeravanja kontrolnih signala zahtijevaju odstupanje od onoga što se smatra očekivanim ponašanjem kod toka audio signala. Primjer takvog ponašanja su tzv. zamrznuti ulazi (eng. *cold inlets*) koji neće pokrenuti izvršavanje izlaza nakon što se objektu proslijedi nova ulazna vrijednost. Neki korisnici također smatraju da objekti poput vrata (eng. *gate*) ili rute (eng. *route*) nisu dovoljno intuitivni zbog njihovih specifičnih primjena i ponašanja. Istovremeno, proceduralni koncepti poput petlji i redoslijeda operacija ne odgovaraju kognitivnom modelu budući da je za ostvarivanje slijeda ili ponavljanja koraka neke operacije potrebno odustati od intuitivnog proceduralnog kognitivnog pristupa te se fokusirati na modeliranje toka podataka.

Konzistentnost

Razlike među konceptima i paradigmama u dva paralelna toka podataka u Pure Dati mogu poslužiti kao indikator generalne konzistentnosti jezika. U oba toka podataka način upravljanja signalima veoma je sličan. Podatci se među objektima razmjenjuju žičanim vezama ili "bežičnim" implicitnim spojevima te se zatim u njima i obrađuju. S obzirom da su kontrolni signali diskretne vrijednosti, Pure Data nudi dodatnu logiku vezanu uz njihovo usmjeravanje i upravljanje, no ona je diskretno uklopljena u sam jezik te ne remeti konzistentnost jezika.

Konzistentnost jezika vidljiva je i u razlikama među konceptima i paradigmama na različitim razinama apstrakcije. Glavni načini enkapsulacije u jeziku (patches, subpatches i apstrakcije) međusobno su zamjenjivi. Svaki potprogram, odnosno subpatch može se pohraniti kao zasebni program te tako postaje apstrakcija, a svaka apstrakcija može postati

subpatch tako što se njen sadržaj premjesti u neki drugi patch. Apstrakcije i subpatchevi također mogu postati patchevi ako se iskopiraju u prazno tijelo glavnog patcha. Takva zamjenjivost ostvariva je zato što su svi koncepti i paradigme konzistentni te na taj način olakšavaju učenje jezika.

Difuznost / sažetost

Promatrajući dimenziju difuznosti ugrađenih objekata, pokazuje se da Pure Data ima sličnu razinu sažetosti prikaza kao i Max/MSP i OpenMusic, no istovremeno je znatno difuznija od NI Reaktora i Kyme. Objašnjenje tih razlika možemo pronaći u činjenici da su i Kyma i Ni Reaktor na najvišoj razini specijaliziraniji jezici za stvaranje glazbe te se stoga i njihovi objekti nalaze na višim razinama apstrakcije. Kod Pure Date je pak taj gubitak u dimenziji sažetosti nužan i opravdan jer omogućuje njegovo korištenje za mnogo širi skup namjena.

Sklonost pogreškama

U općem slučaju, možemo reći da kod Pure Date visoka razina konzistentnosti i preglednosti grafičkog prikaza smanjuju vjerojatnost činjenja pogrešaka. Unatoč tome, postoji nekoliko značajki koje mogu negativno utjecati na ovu dimenziju.

Prva od tih značajki je izostanak kontrole dosega varijabli (eng. *variable scope*) kod slanja i primanja poruka među objektima. Naime, Pure Data podržava slanje i primanje kontrolnih signala korištenjem "bežičnih", implicitnih veza. Kod takvog načina rada, svaki objekt pošiljatelj koristi ime veze kao argument, a objekt primatelj definira isto to ime kako bi se pretplatio na signale koje hvata. To znači da će dva objekta koja koriste isto ime veze prilikom slanja zapravo slati podatke u sliv iste bežične veze. Njihovi se poslani podatci mogu dohvatiti iz svih dijelova istoga patcha (uključujući podpatcheve i apstrakcije) što može dovesti do stvaranja pogrešaka koje je vrlo teško otkriti, posebno ako patch sadrži eksterne apstrakcije. Ovaj je problem moguće zaobići korištenjem prefiksa "$0-" u imenu veze. Prefiks se prilikom izvođenja programa zamjenjuje jedinstvenim brojem. Iako koristan, ovaj koncept nije usporediv s prostorima imena (eng. *namespaces*) i dosegom varijabli koji su dostupni u drugim jezicima.

Dodatni potencijalni izvor pogrešaka u Pure Dati vezan je uz redoslijed operacija. On je određen ugrađenim pravilima koja nisu nužno intuitivno razumljiva programerima. U nekim slučajevima postaje teško razlučiti kako će se program ponašati ako programer u potpunosti ne razumije sljedeća pravila: odnos zamrznutih i aktivnih ulaza, redoslijed spajanja ulaznih veza i princip izvođenja od dna do vrha (eng. *depth-first*) [66]. Konačno, Pure Data ne nudi alate za otkrivanje pogrešaka.

Kognitivno zahtjevne operacije

Pure Data je relativno izražajan vizualni programski jezik kad je riječ o računskim, logičkim i proceduralnim mogućnostima u kontroli toka podataka. Ta izražajnost omogućava programerima ostvarivanje interakcija i algoritama koje nisu moguće u drugim jezicima specijaliziranima za ovu domenu poput Kyme ili NI Reaktora. Kako bi se kontrola toka podataka organizirala proceduralno, programerima su ponekad potrebne složene strukture podataka ili algoritamski obrasci poput ugniježđenih petlji ili rekurzija. Iako sve ove obrasce nije moguće ostvariti u Pure Dati, oni koji se mogu implementirati često zahtijevaju pažljiv i promišljen pristup. Ugniježđene petlje i prekidi petlji primjeri su algoritamskih koncepata koje je u Pure Dati značajno teže ostvariti u odnosu na proceduralne programske jezike. Jezici poput spomenute Kyme ili NI Reaktora zaobilaze probleme takvih koncepata, no istovremeno ograničavaju skup svojih primjena.

Skrivene ovisnosti

Potpuni patch ili program u Pure Dati može se sastojati od nekoliko datoteka budući da se apstrakcije i eksterni objekti spremaju u zasebne datoteke. Zato je nužno sve datoteke držati na okupu te se na njih pažljivo referencirati kako bi se zadržala konzistentnost programa. Pure Data ne nudi ugrađene alate za kontrolu ili upravljanje programskim ovisnostima (eng. *dependency management*). Budući da sve apstrakcije i eksterni objekti izgledaju isto kao i temeljni objekti ili podpatchevi u glavnom patchu, potrebne eksterne datoteke nije moguće uočiti pukim pregledavanjem glavnog patcha. Dodatno, eksterni objekti i apstrakcije mogu imati svoje daljnje ovisnosti što dovodi do rekurzivnog problema.

Na nižim razinama, rizik skrivenih ovisnosti vezan je uz prethodno spomenuti izostanak prostora imena te probleme bežičnih veza čime se dodatno maskiraju potencijalne ovisnosti o drugim datotekama programa.

Preuranjeno opredjeljivanje

Veze u Pure Dati predstavljene su ravnim crtama te se zato relativna topologija niza međusobno spojenih objekata mora prilagoditi kako bi se oslobodio prostor za jasne i nedvosmislene veze. Dodatni izazovi nastaju uvođenjem novih objekata u prethodno formirani patch. Budući da programeri često progresivno izvode i evaluiraju patcheve, patchevi su u potpunosti spojeni i funkcionalni tijekom svih faza razvoja. Dodavanje novih objekata stoga najčešće izazva i izmjenu postojećih veza, premještanje postojećih objekata te prepravljanje njihovih spojeva.

Progresivna evaluacija

Pure Data podržava dva načina interakcije s patchevima. Korisnici mogu ili mijenjati sam patch ili upravljati njegovim parametrima putem grafičkih elemenata poput tipki i klizača. Budući da se patch izvodi kontinuirano, neovisno o izabranom načinu rada, lako je evaluirati rezultate izvođenja tijekom svih faza razvoja jednostavnom promjenom načina rada.

Kako bi se postigla konačna evaluacija patcha, svi potrebni objekti koji čine cjelinu moraju biti 1) međusobno ispravno povezani, 2) evaluirani. Izlazni objekti za signale (npr. prema zvučnim sučeljima) ili objekti za ispis (npr. grafovi) koriste se kako bi se rezultat (odnosno rezultantni audio signal) učinio čujnim ili vidljivim. Budući da Pure Data ne sadrži alate za otkrivanje pogrešaka, progresivna evaluacija koristi se za rano otkrivanje pogrešaka, a sličan se pristup koristi i za otkrivanje pogrešaka jednom kad su one već nastale.

Izražajnost uloga

Pure Data ne nudi aspekte paradigme ili alate koji bi inherentno poboljšali izražajnost uloga. Ugrađeni objekti, eksterni objekti, apstrakcije i podpatchevi predstavljeni su istim simbolima te je ulogu pojedinih dijelova patcha na prvi pogled teško shvatiti. Stoga izražajnost uloga uvelike ovisi o disciplini programera, odnosno o korištenju konvencija imenovanja, komentiranja, sekundarnih zapisa, grupiranja objekata i pažljivog slaganja topologije objekata. Iako bežične veze pomažu vidljivosti i čitkosti mogu značajno negativno utjecati na izražajnost uloga, posebno kad su na taj način ostvarene veze među entitetima u različitim podpatchevima.

Sekundarna notacija

Kako bi pridonijeli razumljivosti patcheva, programeri mogu koristiti komentare, mijenjati vizualni prikaz elemenata u patchu te grupirati objekte u jasne topologije ili podpatcheve, a uvijek slijedeći konzistentnu logiku. Komentari u Pure Dati sadrže obični tekst te se mogu smjestiti bilo gdje unutar patcha. Mogućnost modificiranja vizualnog prikaza elemenata ponajviše se odnosi na upravljačke elemente koji mogu biti različitih boja, veličina i uz koje se mogu vezati labele. Sekundarni zapis na upravljačkim elementima pomaže korisnicima patcha u interakciji s programom. Konačno, Pure Data podržava koncept "prikaz na roditelju" (eng. *graph on parent*) putem kojeg podpatch ili apstrakcija mogu imati posebni prikaz u tijelu roditeljskog patcha.

Viskoznost

Dodavanje novog objekta ili upravljačkog elementa u patch ne narušuje unutarnju konzistentnost ili ograničenja jezika, ali istovremeno ne proizvodi nikakve efekte dok se element ne spoji u tok podataka. Budući da su veze predstavljene ravnim crtama, spajanje novog elementa često zahtijeva izmjenu topologije postojećih elemenata. Brisanje postojećih elemenata može negativno utjecati na funkcionalnost patcha budući da se sve ulazne i izlazne veze također automatski brišu.

Vidljivost

Pure Data je fleksibilan jezik kad je u pitanju organizacija patcheva, pa programeri moraju biti pažljivi i disciplinirani kako bi osigurali dobru vidljivost i preglednost patcheva. Pure Data dozvoljava postavljanje elemenata van trenutno vidljivog dijela radne površine. U tom slučaju pojavljuju se elementi za pomicanje po radnoj površini (eng. *scrollbars*), no time se smanjuje vidljivost. Dodatno, pretjeranim ili pogrešnim korištenjem podpatcheva i apstrakcija moguće je dijelove programa učiniti slabije vidljivima i teže dostupnima.

Max/MSP

Kao što je objašnjeno u poglavlju 2.1, razvoj Max/MSP-a usko je vezan uz Pure Datu. Najbitnije razlike između ta dva jezika jesu otvorenost kôda Pure Date (dok je Max/MSP komercijalan) te dodatni skup alata i proširenja paradigme kojima je Max/MSP nadograđen tijekom godina.

Max/MSP je, kao i Pure Data, izgrađen na modularnoj softverskoj arhitekturi te dozvoljava razvoj dodatnih funkcionalnosti kroz vanjska proširenja. Sama paradigma, podržani tipovi podataka i ugrađeni objekti pritom su gotovo identični kao u Pure Dati.

Kao zanimljivo i potencijalno korisno proširenje prisutno u Max/MSP-u treba spomenuti ugrađenu mogućnost da se patchevi kompiliraju u aplikacije koje se mogu izravno pokretati na Microsoftovim Windowsima i Mac OS X-u, bez prisustva Max/MSP-a.

Daljnja analiza pretpostavlja istovjetnost većine dimenzija između Pure Date i Max/MSP-a te je stoga fokusirana samo na njihove međusobne razlike.

Gradijent apstrakcije

Enkapsulacijski mehanizmi u Max/MSP-u omogućuju programerima stvaranje blokova kôda te njihovo pohranjivanje ili u roditeljskim patchevima ili vanjskim datotekama. U prvom se slučaju takvi blokovi nazivaju subpatchers te odgovoraju konceptu podpatcheva u Pure Dati. U drugom slučaju, odnosno kod pohranjivanja u vanjskim datotekama, riječ je o

apstrakcijama. Posebna značajka Max/MSP-a je mogućnost stvaranja vidljivih sučelja takvih umetnutih blokova korištenjem bpatchera [67]. Umjesto umetanja statičkih i zatvorenih prikaza apstrakcija u glavnom patchu (principom crne kutije), kod umetanja putem bpatchera moguće je prikazati dijelove njihova vizualnog prikaza u samom roditeljskom patchu. Grafički element koji prikazuje sučelje umetnutog subpatchera ili apstrakcije može sadržavati klizače koji omogućuju pregledavanje i kretanje po ukupnom sadržaju umetnutog bloka bez njegova otvaranja u zasebnom prozoru. Ova je značajka slična funkcionalnosti "crtanja na roditelju" u Pure Dati.

Bliskost preslikavanja

Kao i u Pure Dati, programiranje u Max/MSP-u fokusirano je na stvaranje tokova podataka korištenjem objekata i veza. No dok su vizualni prikazi u Pure Dati vrlo jednostavni s temeljnim geometrijskim oblicima i rudimentarnim izgledom, vizualni uređivač u Max/MSP-u nudi mnogo više detalja. Veze između objekata nisu više samo ravne crte, već pomoću sjena i zakrivljenih dijelova emuliraju kabele prisutne u profesionalnoj audio opremi. Također, mnoge komponente prikazane su izražajnim, bogatim simbolima (npr. audio ulaz prikazan je mikrofonom) što izgled patcheva čini bliskima onome kako glazbenici zamišljaju tok signala u fizičkom svijetu. Max/MSP također sadrži alate poput uređivača filtera, spektrograma, grafičkog prikaza podataka i preglednika valnog oblika. Ti dodatni alati ponuđeni su kroz intuitivna, korisniku orijentirana vizualna sučelja.

Slično kao i u Pure Dati, mapiranje između ljudskog kognitivnog modela i vizualnog prikaza bliže je obradi signala nego stvaranju glazbe. Ipak, Max/MSP ima prednost nad Pure Datom zbog ugrađenih alata, dostupnih proširenja i ponuđenih gotovih odsječaka kôda.

Konzistentnost

Po ovoj se dimenziji Max/MSP ne razlikuje od Pure Date.

Difuznost / sažetost

Za Max/MSP vrijede sva ograničenja i značajke kao i za Pure Datu.

Sklonost pogreškama

Sve opisane manjkavosti i obrazloženja vezana uz sklonost pogreškama kod Pure Date vrijede i za Max/MSP (izostanak definiranja dosega varijabli, nejasan redoslijed poruka i zamrznuti/aktivni ulazi). No za razliku od Pure Date, Max/MSP ima ugrađeni alat za otkrivanje pogrešaka čime se smanjuje utjecaj nekih od tih problema te korisnicima omogućuje njihovo lakše rješavanje.

Alat za otkrivanje pogrešaka (eng. *debugging tool*) prikazuje informacije o unutarnjim stanjima programa tijekom izvođenja. Postavljanjem točaka prekida (eng. *breakpoint*) moguće je promotriti kretanje poruka od objekta do objekta putem raznih veza. Alat također nudi temeljne mogućnost pronalaženja pogrešaka poput zaustavljanja izvođenja programa te analize trenutnih stanja pošiljatelja i primatelja poruka.

Kognitivno zahtjevne operacije

Analiza ponuđena za Pure Datu vrijedi i za Max/MSP.

Skrivene ovisnosti

Max/MSP ne nudi poboljšanja u području definiranja dosega varijabli (nema ugrađenih prostora imena) te kao i Pure Data omogućuje bežične veze koje mogu dodatno otežati otkrivanje skrivenih ovisnosti. Stoga je ocjena skrivenih ovisnosti ista kao kod Pure Date.

Preuranjeno opredjeljivanje

Mogućnost bolje topološke organizacije objekata unutar patchera jedno je od poboljšanja koja čine bolju ocjenu preranog opredjeljivanja kod Max/MSP-a. U usporedbi s Pure Datom, veze među objektima nisu ravne crte već, zbog emulacije fleksibilnih kabela, mogu imati zakrivljenja i zavoje što čini vizualno praćenje veza jednostavnijim. Također, smanjuje se broj križanja veza čime se pospješuje preglednost, olakšava pomicanje blokova te stvara dojam urednosti programa. Istovremeno, kako se povećava broj objekata u patcherima, njihova topološka organizacija postajem tim važnija kako bi logika pojedinog patchera bila razumljiva i laka za pratiti. U konačnici, osim blage prednosti zbog zakrivljenih veza, uzroci i razina preranog opredjeljivanja slični su u Max/MSP-u i Pure Dati.

Progresivna evaluacija

Progresivna evaluacija bitan je aspekt prilikom stvaranja patchera u Max/MSP-u neovisno o tome je li namjena evaluacije eksperimentiranje ili pronalazak pogrešaka. Za razliku od Pure Date, Max/MSP sadrži alat za otkrivanje pogrešaka i dodatni prezentacijski način rada. Te mogućnosti motiviraju korisnike da kontinuirano evaluiraju programe. Principi progresivne evaluacije analogni su onima kod Pure Date: objekti u programu moraju biti ispravno povezani, a uključivanje i isključivanje načina uređivanja omogućuje programerima da jednostavno prelaze iz procesa uređivanja objekata u manipuliranje njihovim izlazima i natrag.

Izražajnost uloga

Visoka razina izražajnosti uloga nije inherentna Max/MSP-u, njegovim značajkama ili programskoj paradigmi. Iako je Max/MSP tu opet sličan Pure Dati, postoji bitna razlika: prisustvo bogate palete sekundarnih notacija. Ako se ispravno koriste, metode sekundarne notacije značajno poboljšavaju izražajnost uloga u programu. Korištenjem tih funkcionalnosti, disciplinirani programeri mogu vrlo jasno istaknuti uloge pojedinih komponenata.

Važno je istaknuti da Max/MSP sadrži prezentacijski način rada u kojem programeri mogu stvoriti specifično korisničko sučelje koje će se koristiti prilikom izvedbi uživo, a koje ne mora odgovarati programskom modelu. U tom smislu, izražajnost uloga u programu ne mora nužno odgovarati izražajnosti uloga njegove konačne prezentacije.

Sekundarna notacija

Kao što je spomenuto u prethodnom odlomku, mogućnosti sekundarne notacije značajno su bogatije u Max/MSP-u nego u Pure Dati. Svaki objekt posjeduje skup značajki koje opisuju njegov izgled i ponašanje. Korisnici mogu mijenjati boje i vrstu znakovlja kako bi njihovi patchevi bili razumljiviji. Dodatno, objekti se mogu anotirati korištenjem dvije vrste opisa. Te se vrste opisa u jeziku nazivaju annotations i hints. Kao i Pure Data, Max/MSP nudi mogućnost ostavljanja komentara bilo gdje u patchu. No za razliku od rudimentarnih komentara u Pure Dati, u Max/MSP-u oni mogu koristiti različite vrste znakovlja, veličine i boje.

Nadalje, topologija objekata se zbog zakrivljenih veza može koristiti kao sekundarna notacija, a neke značajke vizualnog uređivača olakšavaju programerima razmještaj objekata u patcheru. Primjerice, moguće je automatski poravnati objekte, raspoređivati ih ujednačeno, pomicati ih naprijed ili natrag, vezati ih uz rešetku ili usmjeravati veze. Prezentacijski način rada i prethodno spomenuti bpatcher nude i dodatne mogućnosti za notaciju patchera.

Viskoznost

Operacije dodavanja i brisanja objekata Max/MSP-a konceptualno su slične istim operacijama u svim ostalim promatranim jezicima osim Kyme. Nakon takvih operacija, program se neće ponašati na željeni način dok se sve veze iznova ispravno ne spoje.

Vidljivost

Slično kao kod izražajnosti uloga, vidljivost patchera ovisi o disciplini i vještini programera. Max/MSP nudi niz funkcionalnosti koje se mogu koristiti za skrivanje dijelova kôda te eksponiranje drugih. Bogati alati sekundarne notacije, prezentacijski način rada i realističniji

vizualni prikazi komponenata omogućuju stvaranje urednih i dobro organiziranih patcheva. Zbog toga je po pitanju vidljivosti Max/MSP u prednosti pred Pure Datom.

OpenMusic

Gradijent apstrakcije

U kontekstu apstrakcijskog gradijenta, OpenMusic se nalazi između Pure Date i Max/MSP-a na jednom kraju te Kyme i Reaktora na drugom. Jezik nudi nekoliko različitih razina apstrakcije, od jednostavnih primitivnih tipova do razreda. Enkapsulacija je podržana kroz subpatcheve koji mogu biti temeljeni ili na paradigmi toka podataka ili na notnom zapisu.

Maquette je pritom posebna vrsta enkapsulacije, svojevrsni "nadpatch", koji omogućuje smještanje objekata u vremenskoj dimenziji. Dok patchevi i podpatchevi koriste paradigmu toka podataka kod koje su izračuni jednokratni i ne ponavljaju se u stvarnom vremenu, maquette se temeljni na konceptima kontinuiranog vremenskog toka.

Na nižim razinama programiranja, razredi i funkcije mogu se grupirati u pakete te se tako postiže relativno visoka razina apstrakcije po pitanju enkapsulacije. Ipak, na razini samog patcha, obrada podataka najčešće je oslonjena na primitivne tipove poput listi.

Bliskost preslikavanja

Budući da OpenMusic, kao što je prethodno spomenuto, može raditi ili sa signalima ili s glazbenom notacijom (notnim zapisom), bliskost preslikavanja razlikuje se ovisno o načinu rada. S obzirom na to da je OpenMusic primarno alat za skladanje, korisnici najčešće upotrebljavaju drugi način rada, odnosno vremensku dimenziju koja je podržana na najvišoj razini hijerarhije te se oslanja na niz objekata za operacije nad notama, akordima, ritmom itd.

Gledano s najniže razine, OpenMusic je izgrađen pomoću koncepata tekstnog programskog jezika Lisp [68] te je za izrađivanje temeljnih elemenata potrebno znati programirati u njemu. Ponašanje vizualnih programa u OpenMusicu također nasljeđuje neke značajke funkcijske paradigme iz Lispa. Jasno je da su to bitni nedostatci jezika kad je u pitanju njegova prilagođenost glazbenicima te zato oni uglavnom koriste unaprijed pripremljene objekte za kompoziciju i manipulaciju tonovima.

Konzistentnost

OpenMusic je, zbog svojih višestrukih načina rada koji se preklapaju, relativno nekonzistentan jezik. Dok su ti načini rada i oblici prikaza programa konzistentni unutar njih samih, globalno unose značajne nedosljednosti. Preklapanje je posebno vidljivo u usporedbi

vremenske dimenzije u maquetteu s razredima i funkcijama na niskoj razini te u spojevima Lispove logike i sintakse s grafičkim dijelovima jezika.

Difuznost / sažetost

Difuznost u OpenMusicu ne razlikuje se značajno od difuznosti Pure Date i Max/MSP-a. Grafički prikaz patcha oslanja se na objekte i veze između njih te je stoga podložan istim poopćenim problemima takvog prikaza.

Sklonost pogreškama

Zbog miješanja različitih paradigmi i značajki Lispa, OpenMusic je od promatranih jezika najpodložniji pogreškama. Pogreške se mogu dogoditi i na nižim razinama. Primjerice, razredi imaju statuse pa se kao izvor pogrešaka javlja zaključavanje u krivi status. Ikone statusa u grafičkom prikazu relativno su neprimjetne pa je moguće da ih korisnici previde i ne uključe u odgovarajući status.

Kognitivno zahtjevne operacije

Rad na niskim razinama na kojima su osjetni utjecaji Lispa te posljedični rad s listama podataka zahtijeva značajne kognitivne napore. Na višim razinama OpenMusic je po pitanju ove dimenzije usporediv s Pure Datom i Max/MSP-om.

Skrivene ovisnosti

Zbog izostanka bežičnih veza, OpenMusic nije podložan problemima sa skrivenim ovisnostima koji su uočljivi kod Pure Date i Max/MSP-a. Također, zbog postojanja paketa, svaki razred i funkcija imaju jasno označene pripadnosti. Drugim riječima, paketi osim za enkapsulaciju istovremeno služe i za definiranje prostora imena. Kako bi se izbjegle kolizije s ugrađenim razredima i funkcijama, korisnički se kôd sprema u zasebni paket.

Preuranjeno opredjeljivanje

U dimenziji preranog opredjeljivanja, OpenMusic ne odudara od dosad promotrenih jezika. Primjerice, položaj objekata u odnosu na topologiju postaje to problematičniji što je patch veći. Također, postoje određeni problemi s povezivanjem objekata koji nisu u međusobno povoljnom položaju i odnosu što dovodi do križanja veza i neurednog izgleda programa. Stoga redoslijed inicijalnog postavljanja objekata može značajno utjecati na kasniju preglednost i funkcionalnost patcha.

Progresivna evaluacija

Progresivna evaluacija djelomično je podržana u OpenMusicu. Dok se pojedini razredi kao i dijelovi patcheva mogu jednostavno evaluirati na zahtjev, funkcije nije moguće evaluirati dok se ne spoje do kraja. To znači da će i progresivna evaluacija cjelovitog patcha ovisiti o tome jesu li sve njegove funkcije pravilno spojene.

Ipak, OpenMusic nudi pomoćni mehanizam koji omogućuje progresivnu evaluaciju nedovršenih patcheva. To se ostvaruje privremenim spajanjem nedovršenih dijelova na izlazne elemente notnog zapisa ili na funkciju za ispis vrijednosti.

Izražajnost uloga

Izražajnost uloga OpenMusica jest prosječna i na razini Pure Date. Iz podpatcheva nije nužno jasna njihova uloga te je teško učiniti simbole podpatcheva izražajnijima. Ipak, maquette je dio OpenMusica koji je iznimno izražajan jer omogućuje pregled cjelovite slike i uloga pojedinih dijelova podpatcha. Naravno, korištenje maquettea nije nužno te ovisi o disciplini i namjerama programera.

Sekundarna notacija

Sekundarna notacija OpenMusica relativno je bogata. Osim organizacije kôda u podpatcheve, moguće je mijenjati veličinu razreda u patchu čime se olakšava isticanje bitnijih razreda. Razredi i funkcije također imaju svoje ikone, a korisničkim funkcijama moguće je dodijeliti proizvoljnu ikonu. Konačno, OpenMusic nudi komentare koji imaju osnovne mogućnosti stiliziranja (podebljana i nakošena slova).

Viskoznost

Ocjene viskoznosti jednake su kao kod Pure Date.

Vidljivost

Od nedostataka OpenMusica po pitanju vidljivosti treba istaknuti da ugniježđivanje, koje se često koristi, smanjuje vidljivost. Također, podpatchevi često zbog svoje složenosti i obima mogu izlaziti izvan okvira prozora čime određeni njegovi dijelovi postaju privremeno nevidljivi. S druge strane, prednost OpenMusica očituje se kod vidljivosti rezultata izvršavanja jer se rezultati najčešće pohranjuju u objektima koji nude vizualnu reprezentaciju prosljeđenih podataka, najčešće u obliku notnog zapisa.

Kyma

Gradijent apstrakcije

Kyma ima razrađen i fleksibilan sustav apstrakcija. Osim raznih ugrađenih elemenata koji se mogu izravno koristiti, Kyma omogućuje stvaranje duboko ugniježđenih struktura i patcheva pomoću radnog procesa temeljenog na vremenskoj domeni. Taj je vremenski slijed osnovni pristup programiranju u Kymi, a u njega se mogu uključiti brojni postojeći patchevi.

Svaki postojeći blok može se također koristiti prilikom izgradnje drugih, složenijih blokova. Uređivačem toka signala blokove je moguće slagati u patcheve. Pritom svaki patch možemo promatrati kao spremnik ili crnu kutiju koja kroz kontrolnu ploču eksponira određene ulaze i izlaze te korisničke kontrole prema unutarnjim funkcionalnostima, ali ih pritom prezentira korisniku na intuitivan i grafički dosljedan način. Svaki od tih ulaza, izlaza i grafičkih elemenata moguće je dodatno mijenjati. Paradigmu koju Kyma koristi na grafičkoj razini moguće je poistovjetiti s enkapsulacijom kod objektno-orijentiranih jezika.

Zbog navedenih značajki, kao i činjenice da je svaki zvuk zasebni objekt, te mogućnosti da se na višim razinama apstrakcije ne treba brinuti kako su zvukovi ostvareni, za Kymu možemo zaključiti da ima najbolji gradijent apstrakcije od svih promatranih jezika.

Bliskost preslikavanja

Posebnost Kyme nalazi se u različitim načinima rada koji omogućuju glazbenicima da o programiranju i programskom komponiranju razmišljaju slično kao kod tradicionalnih oblika organizacije kompozicija. Kod korištenja vremenskog slijeda događaja, sučelje je slično raznim digitalnim radnim stanicama za obradu zvuka te će model rada biti intuitivno razumljiv glazbenicima koji su se s tim alatima susretali ranije.

Dodatno, Kyma nudi mogućnost da, ovisno o trenutnim potrebama, korisnik upravlja radnim procesom u domeni vremena ili toka podataka. Ta su dva načina rada međusobno povezana te se svi zvukovi i elementi pripremljeni u načinu rada s tokom podataka mogu koristiti u načinu rada s vremenskom crtom.

Konačno, kontrola pojedinih zvukova i patcheva te automatizacija parametara odgovara praksi rada s efektima i sintetizatorima zvuka. Ukupno gledano, Kyma je jedan od uspješnijih jezika po pitanju bliskosti preslikavanja.

Konzistentnost

Ako promatramo način rada s tokom audio signala, Kyma pokazuje značajno bolju konzistentnost od Pure Date i Max/MSP-a. Kako je ranije objašnjeno, kod Max/MSP-a i Pure

Date zvučni i kontrolni signali konceptualno su razdvojeni te se odvojeno i obrađuju. Posljedično, kod Max/MSP-a i Pure Date stvaraju se patchevi koji služe ili za generiranje ili za obradu zvuka. Iako se takvi patchevi te elementi za obradu i generiranje mogu kombinirati, konačni rezultat uvijek je ili generiranje ili obrada signala, nikada oboje istovremeno. Kyma, s druge strane, nudi jedinstveni i objedinjujući konstrukt: sam zvuk. Stoga svaki programski segment u Kymi u konačnici odgovara zvuku, neovisno o tome je li riječ o jednostavnoj reprodukciji zapisa ili složenoj mreži efekata. Tako pripremljeni zvukovi postaju temelj za građenje daljnjih zvukova. Jednom kad je zvuk stvoren, način kojim je stvoren postaje nebitan.

Kao minus u dimenziji konzistentnosti valja spomenuti različite načine rada, razne alate koji odudaraju od glavnog načina programiranja itd. Ipak, dobrobit tih proširenja temeljne paradigme značajnija je od nedostataka koje donose, a mnoga od njih nisu nužna u radu s jezikom. Također, ta proširenja ne narušavaju nužno konzistentnost jer ostaju bliski domeni kao što je obrazloženo u poglavlju o bliskosti preslikavanja.

Difuznost / sažetost

Kyma je koncizan jezik kod kojeg se, na najvišoj razini apstrakcije, mnoge složene operacije mogu svesti na jedan blok. No te su operacije ograničene na obradu audio signala što Kymu istovremeno čini znatno ograničenijim jezikom od Pure Date ili Max/MSP-a.

Sklonost pogreškama

Zbog apstrakcija na visokoj razini, unaprijed pripremljenih blokova za razne složene funkcionalnosti te načine rada u kojem se program konstantno evaluira, a novi blokovi automatski smještaju u tok audio signala ili na vremensku crtu, Kyma je otpornija na pogreške korisnika. Istovremeno, zbog izostanka općih programskih konstrukata, smanjuje se opseg u kojem može doći do pogrešaka logičkog tipa koje narušavaju izvođenje programa. Nasuprot tome, najčešćim "pogreškama" u Kymi smatramo neočekivane ili neželjene rezultate izvođenja (npr. zvuk neodgovarajuće frekvencije).

Kognitivno zahtjevne operacije

Kyma nudi niz popratnih, ugrađenih alata koji olakšavaju svaki korak kod računalnog skladanja glazbe, sinteze i obrade zvuka. Dodatno, izostankom općenitih programskih mogućnosti, poput petlji, dodatno se sužava prostor za postojanje kognitivno zahtjevnih operacija.

Skrivene ovisnosti

Kod Kyme sve su međusobne ovisnosti dijelova programa i različitih programa jasno istaknute i eksplicitne. Ipak, treba uočiti da zato što je Kyma mogućnostima svoje paradigme bliža digitalnim radnim stanicama za obradu zvuka i sličnim alatima negoli jezicima opće namjene, izostanak skrivenih ovisnosti dolazi nauštrb općenite fleksibilnosti.

Preuranjeno opredjeljivanje

Kod korištenja načina rada s tokom audio signala, Kyma pokazuje slične probleme kao Pure Data, Max/MSP i OpenMusic. Prvenstveno je riječ o problemima vezanima uz redoslijed kojim se dodaju patchevi za obradu audio signala budući da naknadno mijenjanje, brisanje ili dodavanje patcheva traži značajne napore.

S druge strane, prilikom rada u načinu s vremenskom crtom ograničenja gotovo da ne postoje zahvaljujući mogućnosti da se zvukovi smještaju u različite paralelne kanale ili trake koje se zatim pojedinačno mogu uključivati ili isključivati.

Progresivna evaluacija

Po pitanju progresivne evaluacije, Kyma nudi način programiranja koji je znatno različit od drugih dosad prikazanih jezika. Bilo da se jezik koristi u načinu rada s manipulacijama u vremenskoj domeni ili u načinu manipuliranja tokom audio signala, sve promjene učinjene u programu imaju trenutni učinak na izlaz. Kako bi se to postiglo, u Kymi se novi blokovi automatski dodaju na postojeće konektore te nije potrebno eksplicitno spajati blokove. Time je osigurano da patch u svakom trenutku bude spojen i proizvodi zvuk.

Dodatno, kod patcheva koji očekuju neki vanjski okidač ili ulaz kao što su datoteke s audio podatcima (npr. mp3), moguće je te ulaze uključiti u izvođenje patcha bez da se on zatvori i odvojeno pokrene. Takvo se ponašanje može postići, primjerice, dovlačenjem (eng. *drag and drop*) odabrane datoteke izravno na ulaz u patch. Na sličan se način i različiti drugi alati unutar jezika (poput prikaza valnog oblika i spektrograma) mogu primijeniti izravno na patchu.

Kad je pak riječ o progresivnoj evaluaciji na razini vremenske crte, ona je implicitna i uvijek prisutna jer je sam način rada s vremenskom crtom inherentno interaktivan i stvarnovremenski orijentiran.

Izražajnost uloga

Budući da blokovi u Kymi često enkapsuliraju vrlo složene operacije, simboli nisu grafički izražajni i samoopisni kao primjerice u Reaktoru, nego su na sličnoj razini kao kod Pure Date

ili Max/MSP-a. Istovremeno, njihova je funkcija jasna iz naziva i ulaznih grafičkih sučelja. Sami patchevi u Kymi generalno su manji (sadrže manji broj blokova) zbog obima funkcionalnosti enkapsuliranih u pojedinim blokovima. Zbog toga se izostanak visoke izražajnosti uloga ne reflektira negativno na iskustvo korištenja jezika.

Sekundarna notacija

Kyma ne podržava niti jedan oblik sekundarne notacije te se po toj dimenziji ne može uspoređivati s drugim jezicima. Izostanak sekundarne notacije primjetan je kod stvaranja složenijih patcheva.

Viskoznost

Viskoznost se u Kymi razlikuje ovisno o tome promatramo li način rada s tokom audio signala ili s vremenskom crtom. U prvom slučaju, u uređivaču patcha, viskoznost je slična kao kod Pure Date, Max/MSP-a i OpenMusica. Zbog načina na koji se blokovi automatizirano dodaju u patch, a ovisno o scenariju korištenja, taj aspekt može negativno utjecati na viskoznost.

U slučaju rada s vremenskom crtom, viskoznost je značajno manja zbog izostanka međuovisnosti različitih kanala i zvukova postavljenih na njih. Zvukovi se dodaju i evaluiraju neovisno jedni o drugima. Micanje postojećih ili dodavanje novih zvukova ne utječe na ukupnu strukturu programa.

Vidljivost

Zbog mogućnosti enkapsulacije, vidljivost se može razlikovati ovisno o složenosti programa. Neki patchevi mogu biti duboko ugniježđeni te eksponirati samo dio funkcionalnosti kroz razna sučelja. Ipak, uzimajući u obzir da se korisnik može proizvoljno kretati kroz hijerarhiju patcheva, možemo zaključiti da je vidljivost dobra, no ne na razini Max/MSP-a.

Reaktor

Reaktor je razvojno okruženje za dizajniranje sintetizatora zvuka, sekvencera, semplera, audio efekata i drugih alata u području dizajna zvuka. Njegovo modularno grafičko sučelje omogućuje učinkovito korištenje različitih oblika sinteze i obrade zvuka na višim i nižim razinama apstrakcije. Reaktor uključuje mnoge, u cijelosti implementirane i za upotrebu spremne instrumente, sekvencere i efekte s tisućama predefiniranih zvukova i postavki. Zbog toga, Reaktorovi korisnici mogu stvarati glazbu jednostavno i brzo kombiniranjem postojećih

elemenata. Dodatno, nove je instrumente moguće preuzeti iz rastuće biblioteke korisničkih proširenja.

Za razliku od svih dosad spomenutih vizualnih programskih jezika, Reaktor nije namijenjen stvaranju skladbi, interaktivnih sustava ili multimedijskih radova. Primarna su Reaktorova namjena stvaranje alata za sintezu i obradu zvuka, alata za generiranje glazbenih sekvenci i slično. Kako bi se njima ostvarila skladba ili izvedba, tako ostvareni programi moraju se spojiti u domaćinske programe (poput digitalnih radnih stanica za obradu zvuka) ili stvarnovremenske upravljače. Proceduralni i općeniti programski koncepti u jeziku iznimno su ograničeni. Iz navedenog je jasno da je Reaktorova ciljana skupina korisnika različita u odnosu na dosad opisane jezike, no nasuprot njima nudi značajnu fleksibilnost i lakoću uporabe u kontekstu digitalnih radnih stanica za obradu zvuka i sekvencera.

Gradijent apstrakcije

Reaktor nudi nekoliko različitih mehanizama enkapsulacije. Svaka od njih nalazi se u hijerarhiji apstrakcija od onih vrlo visoke razine koje su konceptualno jednostavne i razumljive glazbenicima do onih najniže razine koje omogućuju manipulaciju samom jezgrom jezika. Reaktor tako nudi: ansamble (eng. *ensembles*) – međusobno ovisne skupove instrumenata, instrumente (eng. *instruments*) – objekte koji grafičkim sučeljem eksponiraju određene funkcionalnosti poput sintetizatora, blokove (eng. *blocks*) – topološki organizirane skupove objekata kao u patchevima, makroe (eng. *macros*) – odsječke nekoliko objekata koji se mogu ponovno koristiti, temeljne ćelije (eng. *core cells*) i temeljnu razinu (eng. *core level*). Svaku od tih razina apstrakcije moguće je koristiti u svim razinama iznad nje.

Dok ansambli i instrumenti svojim vizualnim prikazom i funkcionalnošću blisko prate instrumente i uređaje iz stvarnog svijeta, apstrakcije niže razine poput makroa i temeljnih ćelija bit će razumljivije iskusnim programerima. Taj zaključak posebno vrijedi kad je riječ o temeljnim ćelijama koje napuštaju vizualnu paradigmu te se programiraju tekstnim jezikom.

Uzevši u obzir slojevitost enkapsulacijskih mehanizama u Reaktoru, možemo zaključiti da je riječ o jednom od najnaprednijih jezika u dimenziji gradijenta apstrakcije.

Bliskost preslikavanja

Budući da je primjenom najspecifičniji i najfokusiraniji jezik među promatranima, Reaktor je u cijelosti dizajniran na temelju koncepata, simbola i terminologije iz svijeta sintetizatora zvuka i audio efekata. Imena i vizualni prikazi predefiniranih modula odgovaraju njihovim pandanima iz stvarnoga svijeta. Programiranje u Reaktoru na najvišoj razini stoga je slično

građenju modularnog sintetizatora ili lanca audio efekata dok je na najnižoj razini programiranje blisko obradi signala te slično onome što nude Pure Data ili Max/MSP.

Novije verzije Reaktora uvele su koncept blokova - gotovih komponenti poput oscilatora, filtera, efekata, modulatora i sekvencera koji se mogu koristiti u sprezi s modulima u svrhu izrađivanja instrumenata i efekata na intuitivniji način. Ova značajka dodatno je približila Reaktor analogijama iz stvarnoga svijeta.

Konzistentnost

Dok se Pure Data i Max/MSP oslanjaju na jedinstvenu paradigmu na svim razinama apstrakcije, to ne vrijedi za Reaktor. Panelni pregled je specifični dio korisničkog sučelja koji se ne može smatrati programskom radnom površinom. Međutim, usporedba primarnog načina rada i jezgrenog uređivača otkriva razlike koje mogu usporiti učenje jezika. Naime, na jezgrenoj razini postoje izdvojena, drukčija pravila redoslijeda izvođenja te se istovremeno kontrolni signali razdvajaju od audio signala.

Reaktor je orijentiran ka građenju instrumenata i efekata te, unatoč razlikama u programskoj paradigmi, općeniti koncepti uvijek prate slične principe koji su intuitivni korisnicima koji su radili sa sintetizatorima zvuka i audio efektima. Za razliku od Kyme, Reaktor nema vremensku crtu, no može se koristiti kao vanjsko proširenje digitalne radne stanice za obradu zvuka koje su najčešće izgrađene oko vremenskih crta. Na taj se način Reaktorom može upravljati korištenjem automatizacije parametara.

Difuznost / sažetost

Reaktor podržava veliki broj predefiniranih modula i blokova koji se mogu koristiti za stvaranje instrumenata i efekata. Stoga je na primarnoj razini (odnosno na strukturnoj razini instrumenata) difuznost Reaktora manja u odnosu na Pure Datu, Max/MSP i OpenMusic bez vanjskih proširenja. Nasuprot tome, na najnižoj, jezgrenoj razini, Reaktor pokazuje sličnu difuznost kao prethodno spomenuti jezici.

Sklonost pogreškama

Redoslijed izvođenja operacija zajednički je problem mnogih vizualnih programskih jezika. Kada postoje višestruke paralelne grane u putu audio signala, mora postojati pravilo koje određuje koja će se grana prva izvršiti. Slično kao i u Pure Dati i Max/MSP-u, Reaktor određuje redoslijed događa na temelju redoslijeda kojim su spajani ulazi i izlazi. Ipak, dok navedeno vrijedi na primarnoj razini, na jezgrenoj razini uvode se drukčija pravila koja dozvoljavaju paralelno izvođenje grana. Pogreške nastaju ako korisnici nenamjerno zanemare

redoslijed izvođenja ili propuste razmisliti o razlikama između programiranja na razini instrumenata i na jezgrenoj razini. Tako stvorene pogreške mogu biti iznimno teške za pronaći i otkloniti.

Dodatni izvor pogrešaka proizlazi iz nemogućnosti razlikovanja kontrolnih i audio veza, ulaza i izlaza koji se koriste za spajanje modula. Oni se mogu međusobno spajati iako su raznorodni. Neočekivane rezultate također je moguće dobiti spajanje polifonijskih na monofonijske makroe ili ako se propusti koristiti kombinatore glasa (eng. *voice combiners*) u slučajevima kad je to nužno u polifonijskim sintetizatorima [69].

Kognitivno zahtjevne operacije

Složenost korištenja jezika povećava se spuštanjem po njegovoj hijerarhiji apstrakcija. Programiranje na najvišoj razini (ansambli, instrumenti) značajno je jednostavnije od programiranja na najnižoj (jezgrenoj) razini. Na najvišoj razini, Reaktor možemo promatrati kao domaćinski softver (eng. *host software*) za instrumente i efekte budući da prikazuje korisnicima prilagođene panele s polugama, klizačima, potenciometrima, tipkama, ekranima i drugim uobičajenim grafičkim elementima.

Niže razine pak otkrivaju unutarnju strukturu instrumenata i efekata. Čine to na način koji je intuitivan glazbenicima koji imaju iskustva sa sintetizatorima zvuka, audio efektima i modularnim sustavima. Na najnižoj razini nalazi se domena obrade signala koju je prosječnom korisniku najteže naučiti i koristiti. Operacije na jezgrenoj razini zahtijevaju slične kognitivne napore kao i programiranje u Pure Dati ili Max/MSP-u. Ipak, zbog Reaktorove specifične primjene, jezgrena se razina rijetko koristi te je stoga zaključak da, generalno gledano, Reaktor postavlja najniže kognitivne prepreke od svih promatranih jezika.

Skrivene ovisnosti

Slično kao i Pure Data i Max/MSP, Reaktor podržava bežične veze (npr. funkcionalnost slanja/primanja poruka) koje mogu doprinijeti problemima sa skrivenim ovisnostima u programu. Izvan tog mehanizma, Reaktor nije podložan skrivenim ovisnostima budući da su sve veze među instrumentima, ansamblima itd. vrlo jasno i eksplicitno izražene.

Preuranjeno opredjeljivanje

Kad je riječ o preuranjenom opredjeljivanju, Reaktor je sličan Pure Dati, Max/MSP-u i OpenMusicu. Ulazi se nalazi na lijevoj strani modula, a izlazi na desnoj strani, pa je tok signala organiziran s lijeva na desno. Grananja i paralelni tokovi ostvareni su u vertikalnoj dimenziji. Problemi nastaju kada programi postaju sve složeniji, a time i patchevi teže

razumljivi zbog prevelikog broja veza. Tada programeri često imaju potrebu mijenjati topologiju modula. Prvi se od dva problema može riješiti grupiranjem modula u makroe ili korištenjem ugniježđenih makroa. Drugi se problem može riješiti premještanjem modula i makroa tako da se veze međusobno ne križaju.

Progresivna evaluacija

Za razliku od Kyme, instrumenti i efekti razvijeni u Reaktoru nisu unaprijed ožičeni. Kako bi se moglo evaluirati određeno stanje programa, nužno je povezati module za generiranje zvuka na odgovarajuće audio izlaze. Ovaj je princip sličan Pure Dati, Max/MSP-u i OpenMusicu. Reaktor sadrži i modul za reprodukciju audio zapisa koji se može koristiti za progresivnu evaluaciju kod razvoja audio efekata. Slično, ponuđen je i modul za snimanje podataka u datoteku kojim je moguće pohraniti sve rezultate evaluacije.

Izražajnost uloga

Za razliku od ostalih opisanih jezika, te unatoč tome što nudi značajnu fleksibilnost prilikom stvaranja instrumenata, efekata i sekvencera, Reaktor je usko specijalizirani jezik. Postojeći ugrađeni moduli su jasno imenovani te vezani uz odgovarajuće vizualne prikaze, a istovremeno korištenje modula prilikom gradnje instrumenata često implicira i građenje arhetipske arhitekture. Iz navedenog proizlazi visoka razina izražajnosti uloga.

Sekundarna notacija

Reaktor ne nudi mogućnost anotiranja dijelova instrumenata, modula ili makroa. Njih je moguće samo grupirati. Ipak, ugrađeni moduli imaju opisne i intuitivne vizualne reprezentacije koje smanjuju potrebu za dodatnim komentarima.

Viskoznost

S obzirom na uobičajeno duboku hijerarhiju makroa, promjene na nižim razinama često uvjetuju ulančane promjene na višim razinama. Primjer takve međuovisnosti je dodavanje funkcije na jezgrenoj razini kojom se upravlja pomoću ulaza na instrumentovu panelu. Novi ulazi moraju se dodati i pravilno spojiti u svim makroima u hijerarhiji, od jezgrene razine sve do panela. Osim problema koje donosi hijerarhija različitih ugrađenih razina apstrakcije, viskoznost Reaktora slična je viskoznosti Pure Date i Max/MSP-a.

Vidljivost

Modularno grafičko sučelje doprinosi općenitoj vidljivosti instrumenata razvijenih u Reaktoru. Korisnik može otvoriti višestruke prozore, postaviti jedne do drugih te tako stvoriti

pregled panela instrumenta i njegovih modula na različitim razinama apstrakcije. Razvojno okruženje dobro je organizirano te pomaže korisnicima u ostvarivanju detaljnog pregleda strukture instrumenata. Najznačajniji uočeni problem vezan je uz tok podataka i veza s lijeva na desno budući da veći instrumenti ili makroi zahtijevaju mnogo prostora i horizontalnog pomicanja.

Životopis autora

Antonio Pošćić rođen je 1985. godine u Rijeci. Nakon završetka prirodoslovno-matematičke gimnazije u rodnome gradu, seli se u Zagreb gdje 2008. na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu stječe titulu prvostupnika računarstva, a zatim 2010. i titulu magistra računarstva uz najviše pohvale (*summa cum laude*). Iste godine dodijeljena mu je i brončana plaketa Nagrade "Josip Lončar" za najuspješnijeg studenta računarstva na diplomskom studiju.

Doktorski studij računarstva na Fakultetu elektrotehnike i računarstva upisuje 2011., a glavna tema istraživanja jesu inovativni pristupi u polju vizualnog programiranja za umjetničke primjene, posebno sintezu zvuka i stvaranje glazbe. Drugi istraživački interesi uključuju interdisciplinarne teme na međi umjetnosti, posebice glazbe, i tehnologije. Objavio je dva članka u znanstvenim časopisima A-kategorije, sedam članaka na međunarodno recenziranim konferencijama i sveučilišni priručnik o programiranju u jeziku Python.

Van znanstvenoga rada, Antonio Pošćić zaposlen je kao softverski inženjer i vođa tima u kompaniji Neota Logic Inc. čija je zadaća pomaganje ekspertima u pružanju usluga i automatizaciji znanja pomoću aplikacija pogonjenih umjetnom inteligencijom.

Konačno, kao glazbeni i filmski kritičar, Antonio Pošćić pisao je za publikacije poput The Wirea, The Quietusa, Kulturpunkta, Jazzwisea, Bandcamp Dailyja i PopMattersa. Motiviran ljubavlju prema eksperimentalnim i avangardnim oblicima glazbe, njegov je cilj, i kao znanstvenika i kao glazbenog kritičara, bolje razumijevanje i poboljšanje veze tehnologije i umjetnosti.

Popis objavljenih radova

Radovi iz područja istraživanja doktorskog rada:

- Pošćić, A., Kreković, G., „Ecosystems of Visual Programming Languages for Music Creation: A Quantitative Study“, Journal of the Audio Engineering Society, 2018., DOI: https://doi.org/10.17743/jaes.2018.0028
- Kreković, G., Pošćić, A., Petrinović, D., “An Algorithm for Controlling Arbitrary Sound Synthesizers Using Adjectives”, Journal of New Music Research, 2016., https://doi.org/10.1080/09298215.2016.1204325
- Pošćić, A., Kreković, G., „The Frailty of Formal Education: Visual Paradigms and Music Creation“, Proceedings of the Audio Mostly 2017, London, Ujedinjeno Kraljestvo, 2017.

- Pošćić, A., Kreković, G., Butković, A., “Desirable Aspects of Visual Programming Languages for Different Applications in Music Creation”, Proceedings of the Sound and Music Computing Conference, Maynooth, Ireland, 2015.
- Pošćić, A., Kreković, G., “Controlling a Sound Synthesizer Using Timbral Attributes”, Proceedings of the Sound and Music Computing Conference, Stockholm, Sweden, 2013., str. 467–472.

Ostali radovi:

- Kalafatić, Z., Pošćić, A., Šegvić, S., Šribar, J., „Python za znatiželjne“, Element, Zagreb, 2016.
- Kreković, G., Pošćić, A., Manojlović, S., „Chatbots as a Novel Interactive Medium for Poetry“, Proceedings of the Sixth Conference on Computation, Communication, Aesthetics and X, Madrid, Španjolska, 2018.
- Kreković, G., Pošćić, A., “Shaping Microsound Using Physical Gestures”, Proceedings of the International Conference on Computation, Communication, Aesthetics and X, Glasgow, UK, 2015.
- Bosnić, I., Pošćić, A., Ačkar, I., Žibrat, Z., Žagar, M., „Online Collaborative Presentations“, Proceedings of the 32nd International Conference on Information Technology Interfaces (ITI 2010), Cavtat, Hrvatska, 2010., str. 337–342.

Biography of the Author

Antonio Pošćić was born in 1985 in Rijeka, Croatia. After finishing high school in his hometown, he moved to Zagreb where in 2008 he received his B.S. and in 2010 his M.S. (summa cum laude and valedictorian) degrees in computer engineering at the Faculty of Electrical Engineering and Computing, University of Zagreb.

Antonio Pošćić enrolled in a PhD programme in Computing at the Faculty of Electrical Engineering and Computing in 2011. His PhD research is focused on innovative techniques within the field of specialized visual programming languages for artistic applications, especially sound synthesis and music creation. He is also interested in other interdisciplinary fields that investigate the fusion of art, especially music, and technology. He has published two scientific papers in journals, seven papers in conference proceedings, and a book on programming in Python.

Outside his scientific research, Antonio Pošćić works as a software engineer and team lead at Neota Logic Inc., a company devoted to helping professionals provide their services with AI-powered applications that intelligently automate expertise, workflow, and documents.

Finally, as a music writer, Antonio Pošćić has contributed to publications like The Wire, The Quietus, Kulturpunkt, Jazzwise, Bandcamp Daily, and PopMatters. Motivated by his love of experimental and avant-garde musical forms, his goal both as academic researcher and freelance music writer is to help better understand and improve the link between technology and art.