Can Pipe-and-Filters architecture help creativity in Music?

Rômulo Vieira[1], **Flávio Luiz Schiavoni**[1]

[1] Arts Lab in Interfaces, Computers, and Everything Else - ALICE
Federal University of São João del-Rei - UFSJ
São João del-Rei - MG - Brazil

Abstract. *Ubiquitous Music is an emerging field of study that addresses how human agents can use computing, in the most diverse ways, to create music, an activity that is characterized by pragmatic and epistemic actions, restricted by natural and social means. In addition, this area of expertise seeks to create tools that support creativity. Thus, this paper discusses how the Pipe-and-Filters architecture, common in software development, can help in creativity and music creation, either by being present in applications that exploit this, or by the logical way in which it is structured.*

1. Introduction

A striking feature of Ubiquitous Music is that all agents in a music performance must be able to influence sound results [Keller et al. 2010]. It includes the audience's members, normally lay-musicians, taking part of the performance and the creative process of music making. It should also be noted that this technique helps in making music a social activity, without pre-defined roles, placing experts and novices side by side [Schiavoni and Costalonga 2015].

The inclusion of everyone in the creative process of music is certainly a task that can bring several challenges to be executed. Among its challenges are: assisting creativity, facilitating laypeople-musician interaction and interaction on fixed and portable devices. There are also ethical dilemmas, such as participation, inclusion, human development and sustainability involved in the decision of taking people as part of a musical presentation [Keller 2017].

To define and understand what creativity is has motivated researchers over the time. Although he was not the first to explore the subject, J. P. Guilford, then president of the American Psychological Association, is considered a pioneer in this type of research. Later, new researchers such as Margaret Boden, Ronald Beghetoo, James Kaufmann and Cecília Salles added new topics to the discussion [Beghetto and Kaufman 2007].

Boden [Boden 1991] defines creativity as "*the ability to come up with ideas or artefacts that are new, surprising and valuable*". She also attributes the characteristic as a natural condition of human beings, as well as thinking, perception and self-criticism. From there, she divides creativity into two fields: **Psychological creativity** (P-creativity) and **Historical creativity** (H-creativity). The first term is defined as a surprising, valuable and unprecedented idea for the person, even if it is a concept already known. As for H-creativity, it is defined as something new in human history [Boden 1991].

Within these two concepts, three other new forms of creativity emerge: **exploratory**, **transformational** and **combinational**. The first one involves making unknown combinations of familiar ideas. These new combinations can be generated deliberately or not, but in all cases creating this combination requires a rich reserve of knowledge. The most common examples are poetic images, collages in paintings and analogies. The second way involves the transformation of already structured concepts of thought. They are usually from social group and religion, being occasionally affected by other cultures. But in both cases, they are there, they are not originated by an individual mind. Finally, the combinational structure, as the name implies, involves the combination of already familiar thoughts, again influenced by culture and social group [Boden 1991].

Beghetto and Kaufman [Beghetto and Kaufman 2007] define creativity as "*the ability to produce innovative (that is, original, unexpected), high quality and appropriate work*". The authors further divide this field into four parts: **eminent creativity** (Big-c), which requires specialized knowledge and aims at creating professional products such as works of art or scientific theories; **professional creativity** (pro-c), which also requires specialized knowledge but whose result of the creative process does not have the same impact as eminent creativity; **everyday creativity** (little-c), which uses day-to-day experiences to generate creative products, and **personal creativity** (mini-c), characterized by internal, subjective and emotional aspects but which does not necessarily aim at artistic products.

With regard to music creation, Bennett [Bennett 1976] divides the creative process into five stages and an independent final part of revision. The stages are: **germinal idea**, **sketch**, **first draft**, **elaboration and refinement** and **final draft copying**. The independent internship is the review of all the process (Figure 1).

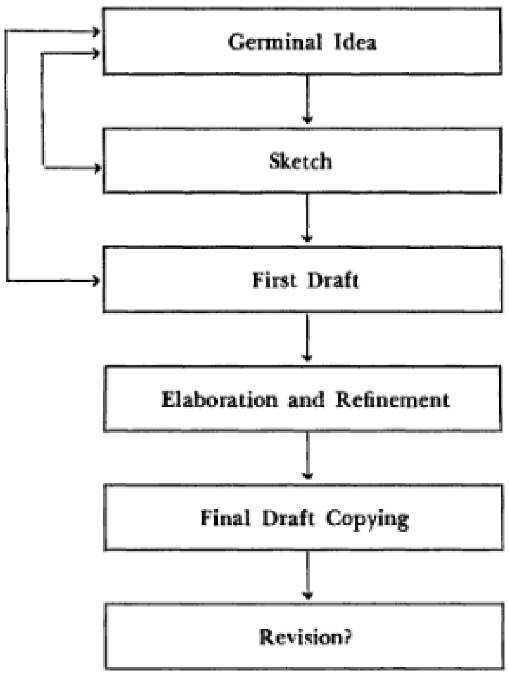


Figure 1. Creative flow elaborated by Bennett [Bennett 1976].

In this way, the process is triggered by a twin idea that expands to a first sample of the composition. At this point, Bennett suggests that a review of the initial idea already sets up a new draft, modifying the first idea. The material is then elaborated and improved, leading to the final composition. Once the work is finished, further revisions can be made.

A point worth mentioning in the model is that backtracking is considered part of the creative process [Bennett 1976].

Keller and others [Keller et al. 2014], after reviewing eight models of creativity in music production, identified three points of relationship with the dimensions of creativity, namely: **materials**, **procedures** and **context**. From this, they identified flaws in the methods of application in music, such as domain restriction and lack of material and social reasoning. Thus, they emphasize that the availability of resources interferes with the creative process.

A common option to face these challenges is to use computer and technologies to integrate people in creative tasks. It is also known that the computer has long been considered an extremely attractive tool for creating, manipulating and analyzing sounds. Its precision and potential for automation make it a useful platform for expression and experimentation. However, it is a fact only to the extent of what we are able to express to the computer to do and how to do it. In view of this reality, the most diverse environments appear to help in the creation and execution of music, including those aimed at lay people, both in the field of music and in the field of computing.

Maybe it is possible to identify a common architecture of these systems, called Pipe-and-Filters, presented in Section 2. In this section we present several examples of how this system architecture can be used to create complex applications combining simple parts of code, from the Unix Shell to sound processing.

At the end, in Sections 3 and 4, we present some discussion about how programming environments, in particular Mosaicode, can help the creativity and the creation of ubiquitous music and what were the conclusions and experiences resulting from this work.

2. The Pipe-and-Filters architecture

According to IEEE, software architecture is defined as a fundamental organization of a system and its components and the relationship between itself and the environment that guide its design and evolution. In short, architecture is about how is the interaction between components and not how they are implemented [Maier et al. 2001].

Nowadays, applications are becoming larger, more integrated with other applications and uses to use a wide variety of technologies. It is in this context that the software architecture is very important. A good architecture works to guarantee the quality of the software product, as well as its reliability and usability. In addition, it serves as a means of communication between the entire stakeholder, represents the initial decisions of the project, is the basis for detailing a work and can be a unit for reuse, which means that your model can be transferable between systems [Bijlsma et al. 2011, Agostini et al. 2019].

There are different types of software architectures, such as: Data Abstraction and Object-Oriented Organization, Event-based, Implicit Invocation, Layered Systems and Pipe-and-Filters. The Pipe-and-Filters architecture, focus of this paper, is a division of a complex task into several sub tasks, implemented by a software component called filters. Filters have multiple inlets and outlets connected to pipes, but never know the characteristics of adjacent filters. The pipeline, on the other hand, is responsible for performing the communication between two software components and acting as data buffers between the adjacent filters. The logical functioning of this model can be seen in Figure 2

[Bijlsma et al. 2011, Avgeriou and Zdun 2005].

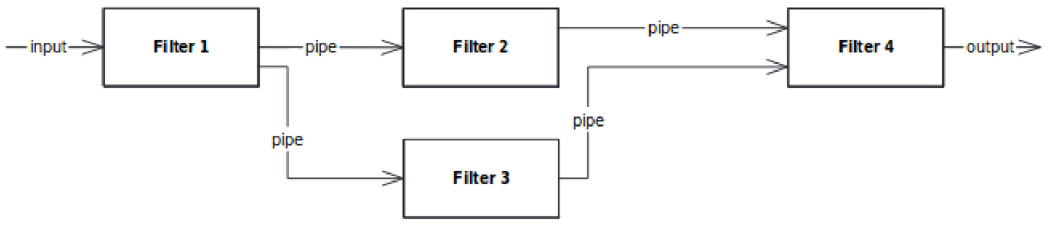


Figure 2. Pipe-and-Filters example [Bijlsma et al. 2011].

The pipe connector is considered to be a key part of the data flow transfer. Because of this, flexibility at the moment of connection is one of the main characteristics of this architecture, as it allows customized assemblies for each specific problem. With regard to data transfer, the flow is constant [Medvidovic and Taylor 2010, Bass et al. 2003].

This type of pattern is very popular and, because of that, adopted in several systems, mainly to manipulate different sets of data in different ways. The use of this model is advisable when little contextual information needs to be kept between the components and the filters are not able to maintain their states. They can still be an alternative to other architectures or work together with them. More directly, they are widely adopted in compilers and the UNIX shell, sound processing and sound programming.

2.1. The UNIX Shell

Created in the 1970s, for minicomputers and mainframes, the UNIX system has become increasingly popular and has moved to workstations and personal computers, going well beyond the use of it by academics or specialists. During that period, he also received several nomenclatures for variations, such as Ultrix, Xenix and Linux [Newham 1998].

In general, the shell is any interface that receives commands or text scripts from the user and converts them in instructions understandable to the operating system. Users typically interact with a Unix shell using a terminal emulator, however, direct operation via a serial hardware connection, or computer network session is common for server systems. The shell, in turn, provides basic commands, such as chaining, command substitution, control structure and conditional testing. These commands, also known as tools, do everything from counting the number of lines in a file, sending electronic mail, to displaying a calendar for any desired year [Newham 1998, Salus 1994].

Another advantage of UNIX shell is that it allows effective connection of simultaneous commands. This connection precisely follows the Pipe-and-Filters architecture. Therefore, it allows the output of one command to be directly connected to the standard input of a second command [Salus 1994, Rosen et al. 2007].

Here, filter terminology refers to any program that can receive input data and then perform some action on that input and send it to the output to be connected in another filter. More clearly, a filter is a desirable change that can occur in the communication between two programs in a pipeline. This makes shell programming extremely useful, providing a variety of tools for programmers and system administrators.

2.2. Pipe-and-Filters in sound processing

The spread of this architecture, or at least its main concept, was so vast that it also reached the field of sound processing. Take the famous Moog synthesizer (Figure 3) as an example. Created in 1964, this instrument was initially monophonic, with a reduced synthesis capacity, providing easy operation by the user, whether he is a musician or not. This feature of the instrument is analogous to the pipeline, where data is sent from a source to another [Trocco and Pinch 2004].

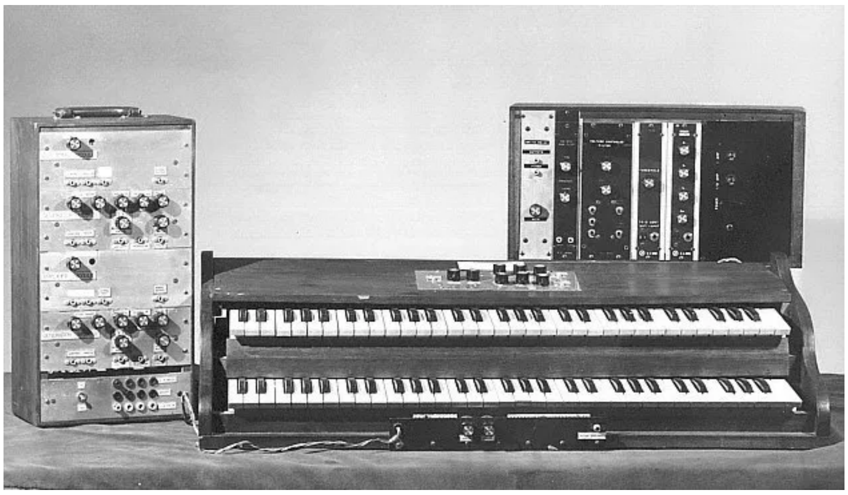


Figure 3. Early version of the Moog Modular [Trocco and Pinch 2004].

Another outstanding feature of this synthesizer is its constitution by separate modules and with specific functions, such as generating an envelope, oscillating the frequency of a note and applying a reverb to the sound. These functions are analogous to filters, capable of applying modifications to the data transmitted by the pipes, in this case, the cables physically connecting different unities. In this way, infinite possibilities have emerged with regard to the creation of sounds and music [Trocco and Pinch 2004].

In sound processing, Pipe-and-Filters architecture is not limited to synthesizers and it is present in several other situations. Effects pedals and pedalboards (Figure 4) for electric guitars work in the same way, that is, audio data generated by the resonance of the strings flows through cables until they reach the pedals. Then, data goes through an electrical circuit that changes the sound of the instrument. After transformation, data is sent to another pedal via another cable until it reaches the amplifier. Each transmission is carried out through a pipeline and each change in sound by a filter [Roberts 2019, Hunter 2013].

When concerning audio in studios and concerts, the same idea is present. A lot of sound sources, like microphones and instruments are connected by pipes, in this case, audio cables. The data is the audio signal and several different devices, like direct boxes, compressors, equalizers, and noise gates, act like filters changing the audio signal. The mixer receives all the pipes and connect them in a complex system, allowing inserts, effects and other filters to be connected. Sometimes, a patch bay is present to simplify the addition or removing of a new component. As an example of the pipes and filters architecture, in this system a audio processing unit is included in the mesh and does not know what is after or before it in the audio chain.

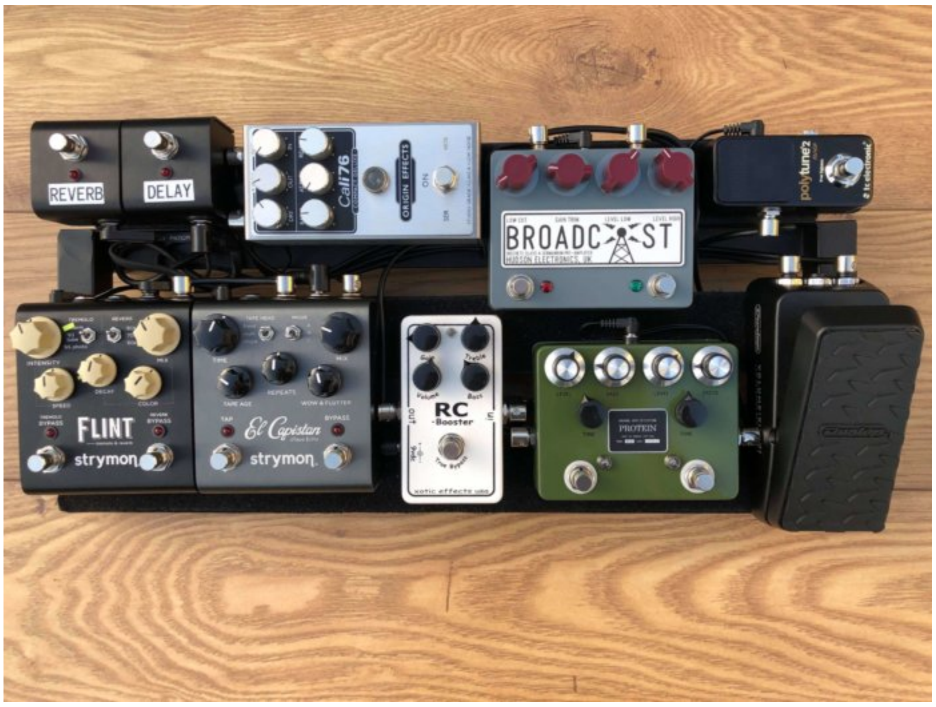


Figure 4. Pedalboard, a set of guitar pedals [Roberts 2019].

2.3. Pipe-and-Filters to music programming

Part of the musical creation can be done based on this architecture, especially using environments or languages of music programming. Even though it is not so explicit, Pipe-and-Filters is present in audio programming languages and environments, such as Pure Data, FAUST, SuperCollider, ChucK, and Mosaicode, for instance. For laymen, both in music and in programming, the advantage is that they can create something from the interconnection, combination and alteration of components, as well as their parameters.

Pure Data (Figure 5) is a graphical interface for real-time programming of audio and video. Its functionality is directed at an interaction similar to manipulating objects in the real environment, moving and connecting them in order to generate new combinations. For this, the language uses the interconnection of boxes that contain some command, forming a flow of information. At this point that it correlates with the Pipe-and-Filter, since the output from one control box will serve as an input to another box, and yet, the data undergoes some change by a filter. Pure Data can be found in several applications, like Reactable, a musical interface aimed at creating collaborative music, RJDJ app and the Networked Resources for Collaborative Improvisation (NRCI) library [Puckette 1998, Puckette 2011].

Functional AUdio STream, popularly known as **FAUST**, is a high-level language, focused on digital signal processing, with special support for real-time audio applications and plugins on various software platforms, including mobile systems. Among its advantages are: easy creation and reading of command blocks, support for Graphical User Interface (GUI) and generation of C ++ code from the created blocks [Michon and Smith 2011, Michon et al. 2019]. Its function is to describe a signal processor, which applies a mathematical equation to the input signals to produce output signals.

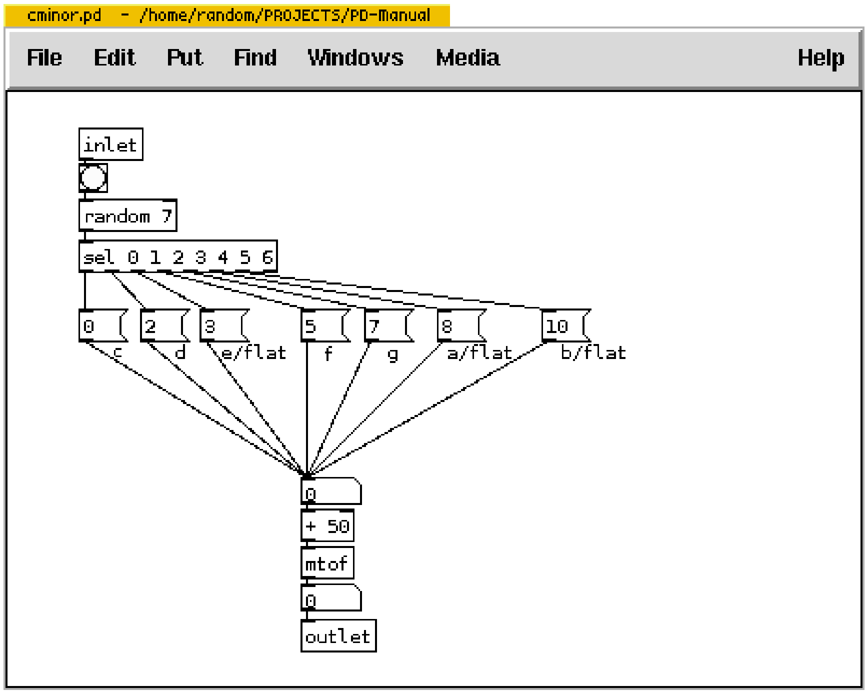


Figure 5. Pure Data interface.

The functional programming approach provides a natural framework for signal processing. Digital signals are modeled as discrete functions of time and signal processors as second order functions that operate on them. It can generate code to the most diverse plugin architecture, including formats such as LADSPA, MAX/MSP and VST, in addition to mobile applications for Android and iOS [Smith 2011, Letz et al. 2010].

SuperCollider is an object-oriented language, with a focus on algorithmic composition and audio synthesis, requiring the least possible effort. Its main characteristic is to transform musical concepts into functions or methods, creating music from the manipulation of these elements through blocks, as well [Wilson et al. 2011]. Among its main qualities are dynamism and brevity. The dynamism allows the user to create structures that generate events in a nested way. Patches can be built dynamically and parameterized not only by floating point numbers, but also by other unit generator graphics. The patch can also be built algorithmically in real time [McCartney 1998]. Since SuperCollider is a Object-oriented programming language, it can be an example of the combination of two architectures, Object-Oriented as a programming paradigm and Pipes-and-Filters as the flow of audio data.

Another language focused on live coding, composition and performance in real time is **ChucK**. Its objective is to be proactive and interactive to the user, however, this causes some restrictions, such as less absolute performance. In addition, ChucK presents a highly visible and centralized view of logical time (via keyword) that reconciles logical time with real time. Another highlight is the ability to edit the code while the program is running [Wang 2008, Wang and Cook 2003]. ChucK natively supports multiple simultaneous and dynamic control rates and the ability to add, remove and modify code on-the-fly, and its application ranges from algorithmic composition to sound synthesis, recording and visual effects display. As a whole, ChucK allows the programmer to write code quickly and easily to synchronize and communicate with input devices. It can be a good tool for easy use or the rapid prototype of new controllers, or for writing network code or synchronization. This programming language has been used in performances by the Princeton Laptop Orchestra (PLOrk) and for developing the musical applications [Wang 2008, Dean 2009].

Mosaicode is an open-source programming environment, created by a group of researchers at the Federal University of São João del-Rei, Brazil. Focused on a visual environment, this tool is used for development of systems common to digital art and music in web browsers, extending its use to mobile devices [Schiavoni and Goncalves 2017, Schiavoni et al. 2018a, Schiavoni et al. 2018b]. The similarity with the other techniques is in the fact that it also uses blocks and connections to create a complex code, inspired in the Pipe-and-Filters architecture. A block represents a single unity of code in the system and it can have static and dynamic properties. The dynamic properties is represented by an input port. The output processing is represented by an output port. These ports define the programming data flow and can be connected like presented in Figure 6 [Schiavoni and Goncalves 2017, Schiavoni et al. 2017].

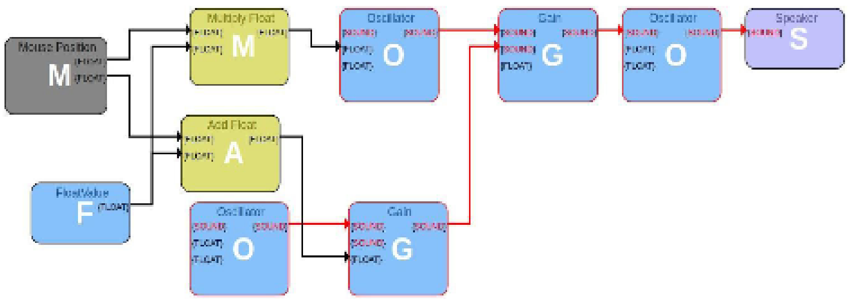


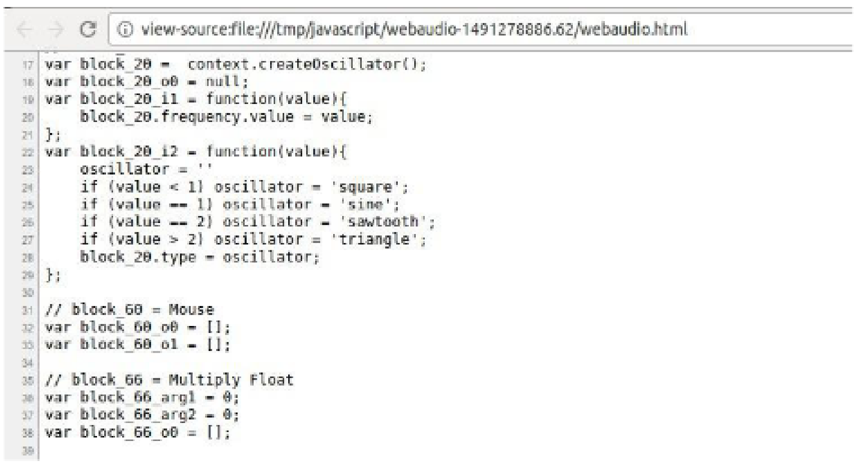
Figure 6. Block interconnection in Mosaicode.

However, Mosaicode is not an environment for interpreting code, but for generating it from the combination of blocks created by the user. A code generated in the tool is shown in Figure 7.

3. Discussion

To try to ensure the participation of lay-programmers and lay-musicians in creative process, we propose to use the Pipes-and-Filters architecture in this process. Thus, the creation of a complex music, piece or instrument can be done combining a set of simple objects, the Filters. In this case, we are understanding it in a general way, and it does not matter if one can use guitar pedals, Moog devices, SuperCollider code or Pure Data objects.

Each Filter has a internal configuration, just like the parameters passed to a shell command, the parameters passed to a code function or the knobs and buttons present in a studio compressor, a Moog box or in a guitar pedal. Sometimes we can have a single data type and a single pipe type, like in Unix shell, or different pipes with different data type,



```
view-source:file:///tmp/javascript/webaudio-1491278886.62/webaudio.html
```

```
var block_20 =  context.createOscillator();
var block_20_o0 = null;
var block_20_i1 = function(value){
    block_20.frequency.value = value;
};
var block_20_i2 = function(value){
    oscillator = ''
    if (value < 1) oscillator = 'square';
    if (value == 1) oscillator = 'sine';
    if (value == 2) oscillator = 'sawtooth';
    if (value > 2) oscillator = 'triangle';
    block_20.type = oscillator;
};

// block_60 = Mouse
var block_60_o0 = [];
var block_60_o1 = [];

// block_66 = Multiply Float
var block_66_arg1 = 0;
var block_66_arg2 = 0;
var block_66_o0 = [];
```

Figure 7. Example of JavaScript source code of a webaudio application generated by Mosaicode.

like in Pure Data and Mosaicode. The creativity task here is to combine a set of Filters to create an art piece.

A good point to start thinking about creativity and Pipes-and-Filters it that the creation of a complex program in this architecture is based on the combination of a set of less complex filters. Thus, the basic idea is to reuse components that are already known, combining them and using their predefined feature to compose a new idea.

Using the concepts presented by Boden, the Pipe-and-Filter architecture can be used in the three forms of creativity, starting with the **exploratory** form. Exploratory creativity involves making unprecedented combinations of familiar ideas. This model can be highly explored in a Pipe-and-Filters system where the familiar ideas are the Filters and the possibility to combine them is the main form of programming. In this context, music creation tools redefine the use of this architecture, assigning its concepts to a completely different field.

In **transformational** creativity, Pipe-and-Filter is no longer just a hidden part of the software, to appear graphically to the user, allowing one to create and manipulate the system inputs, encapsulate them in pipes and modify them through filters, just like in Mosaicode and Pure Data. Thus, they are still able to create a new parameter, since the application provides a code from the block interconnections.

In the **combinational** structure, the concept of combining different ideas is much broader than in the previous ones. For those who are only initiated in computing, they have already encountered this form of command in their field of action and now they are faced with the technique for creating music. For those who are musicians, the application of filters to input signals occurs on guitars, synthesizers and other electro-acoustic instruments. For those who are laymen in both subjects, they have certainly come across equipment that needed a correct fit to work, especially since this is a recurring activity for children, such as assembling puzzles or Lego pieces. Thus, common ideas are used to create music as well.

The division of creativity forms, presented by Beghetto and Kaufman [Beghetto and Kaufman 2007] can be useful to understand how Pipes-and-Filters can help creativity and also gives a clue about teaching and learning based on this architecture. Maybe it is possible to understand that the main difference between an agent that can be considered a Big-c and an agent classified as mini-c is the knowledge about how to create a product. In a Pipe-and-Filters system, this knowledge can be understood as how many filters do you know and how deep do you know their features. Certainly, a Pure Data user that knows hundreds objects should not be classified as more creative user than a user that knows a few, but certainly it is possible to classify them as a more expert user and a less expert based on it. Thus, it is possible to define a learning process dividing it in learning several filters and how to combine them.

The creative process presented by Bennett [Bennett 1976], divided into five main stages can also be helped by a Pipe-and-Filters architecture and concept. In this case, the refinement stage and the sketch can be done adding, reordering and removing filters, keeping the main idea and infrastructure and changing components of the final product. Also, it is possible to reuse parts of the systems in other works and to divide the creative process in several tasks, just like in Pipe-and-Filters architecture.

As for the programming languages that use this architecture to create code and consequently music, some peculiar characteristics to each type emerge. Pure Data, for example, has a similar approach to the real world of audio stomp boxes, where the connection of elements in a given order will result in a new value, in addition to being aimed at real-time collaboration between networks and live coding. FAUST, in turn, is marked by not describing a sound or a group of sounds, but rather a signal processor, often in the form of a plugin or standalone application. SuperCollider is characterized by the generation of audio synthesis and algorithmic composition in real time through functions and methods. ChucK is also focused on audio synthesis, composition and performance, but it presents the differential of privileging flexibility over performance. Finally, Mosaicode can be defined as a visual programming language and a domain-based language, which allows the user to develop systems using two-dimensional notation and interaction with code through graphics, in addition to developing systems within a well-defined scope.

4. Conclusion

This paper proposes to analyze how Pipe-and-Filters can help in the creation of Ubiquitous Music by people without musical or technological knowledge. Applications built from this architecture, or with a similar operating logic, were presented, such as Moog synthesizers and guitar pedals, beyond that which assist in musical composition, such as Mosaicode, Pure Data and FAUST.

Among the creative processes, this structure of sending and manipulating data correlates with exploratory, transformational and combinational types. In short, it is a technique that supports the tools used to generate some type of art and also composes the creative method to perform this task.

The various forms of creativity focused on music are also taken into consideration and, once again, Pipe-and-Filters can help so that more and more people can express themselves through music, in a simple, playful and graphic (or sometimes textual) way.

To understand how creativity can be related to the combination of simple filters to create a complex system is an interdisciplinary field of study and has an important role in education, social and technological integration, the exploration of the most diverse techniques helps in the growth of Ubiquitous Music, making it sustainable and accessible to the most diverse people, from the most diverse social classes.

Acknowledgment

Authors would like to thanks to all ALICE[1] members that made this research and development possible. The authors would like also to thank the support of the funding agencies CNPq, (Grant Number 151975/2019-1), CAPES (Grant Number 88887.486097/2020-00) and FAPEMIG.

References

Agostini, A., Ghisi, D., and Giavitto, J. L. (2019). Programming in style with bach. In *International Symposium on CMMR,*, pages 91–102.

Avgeriou, P. and Zdun, U. (2005). Architectural patterns revisited - a pattern language. In *EuroPLoP' 2005, Tenth European Conference on Pattern Languages of Programs*, volume 81, pages 431–470.

Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture In Practice*, pages 23–726. Addison-Wesley Professional.

Beghetto, R. and Kaufman, J. (2007). Toward a broader conception of creativity: A case for mini-c creativity. *Psychology of Aesthetics, Creativity, and the Arts*, 1:73–79.

Bennett, S. (1976). The process of musical creation: Interviews with eight composers. *Journal of Research in Music Education*, 24(1):3–13.

Bijlsma, A., Heeren, B., Roubtsova, E., and Stuurman, S. (2011). *Software Architecture*. Free Technology Academy.

Boden, M. (1991). *The Creative Mind: Myths and Mechanisms*. Basic Books, Inc., USA.

Dean, R. T. (2009). *The Oxford Handbook of Computer Music*. OUP USA.

Hunter, D. (2013). *Guitar Effects Pedals: The Practical Handbook*. Backbeat Books.

Keller, D. (2017). Challenges for a second decade of ubimus research.

Keller, D., Barreiro, D., Queiroz, M., and Pimenta, M. (2010). Anchoring in ubiquitous musical activities. *International Computer Music Conference*.

Keller, D., Lazzarini, V., and Pimenta, M. (2014). *Ubimus Through the Lens of Creativity Theories*, pages 3–23. Springer.

Letz, S., Orlarey, Y., and Fober, D. (2010). Work stealing scheduler for automatic parallelization in faust. *Linux Audio Conference*.

Maier, M., Emery, D., and Hilliard, R. (2001). Software architecture: introducing ieee standard 1471. *Computer*, 34:107 – 109.

McCartney, J. (1998). Continued evolution of the supercollider real time synthesis environment. *ICMC*, page 4.

[1] `https://alice.dcomp.ufsj.edu.br/`

Medvidovic, N. and Taylor, R. (2010). Software architecture: Foundations, theory, and practice. In *Proceedings - International Conference on Software Engineering*, pages 471–472.

Michon, R., Orlarey, Y., Letz, S., Fober, D., and Dumistracu, C. (2019). Mobile music with the faust programming language. In *International Symposium on CMMR*, pages 371–382.

Michon, R. and Smith, J. O. (2011). Faust-stk: A set of linear and nonlinear physical models for the faust programming language. *Proceedings of the 14th International Conference on Digital Audio Effects*, pages 199–204.

Newham, C. (1998). *Learning the Bash Shell*. OReilly.

Puckette, M. (1998). Pure data: Recent progress. *International Computer Music Conference*.

Puckette, M. (2011). *Pure Data*. Pure Data Development Team.

Roberts, S. (2019). Your board: Joe coombs' go anywhere session pedalboard.

Rosen, K. K., Host, D. A., Klee, R., and Rosinski, R. (2007). *UNIX: The Complete Reference, Second Edition*. McGraw-Hill Education.

Salus, P. H. (1994). *A Quarter Century of UNIX*. Addison-Wesley Professional.

Schiavoni, F., Cardoso, T., Gomes, A., Resende, F., and Sandy, J. (2018a). Utilização do ambiente mosaicode como ferramenta de apoio para o ensino de computação musical. In *8th Workshop on Ubiquitous Music (UbiMus)*, pages 33–44.

Schiavoni, F. and Costalonga, L. (2015). Ubiquitous music: A computer science approach. *Journal of Cases on Information Technology*, 17:20–28.

Schiavoni, F. and Goncalves, L. (2017). From virtual reality to digital arts with mosaicode. In *2017 19th Symposium on Virtual and Augmented Reality (SVR)*, pages 107–112.

Schiavoni, F., Gonçalves, L., and Gomes, A. (2017). Web audio application development with mosaicode. In *16th Brazilian Symposium on Computer Music*, pages 107–112.

Schiavoni, F., Gonçalves, L., and Sandy, J. (2018b). Mosaicode and the visual programming of web application for music and multimedia. *Revista Música Hodie*, 18:132.

Smith, J. O. (2011). Audio signal processing in faust. *Center for Computer Research in Music and Acoustics (CCRMA) Publications*, pages 1–54.

Trocco, F. and Pinch, T. (2004). *Analog Days: The Invention and Impact of the Moog Synthesizer*. Harvard University Press.

Wang, G. (2008). *The ChucK Audio ProgrammingLanguage "A Strongly-timed and On-the-flyEnviron/mentality"*. PhD thesis, Princeton University.

Wang, G. and Cook, P. (2003). Chuck: A concurrent, on-the-fly audio programming language. In *ICMC*, page 8.

Wilson, S., Cottle, D., and Collins, N. (2011). *The SuperCollider Book*. MIT Press.