Managing an IoMusT environment and devices

Rômulo Vieira, Flávio Luiz Schiavoni

Managing an IoMusT environment and devices

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

romulo_vieira96@yahoo.com.br, fls@ufsj.edu.br

Abstract. *The Internet of Musical Things (IoMusT) is an interdisciplinary area that encompasses concepts from the Internet of Things, ubiquitous music, new interfaces for musical expression, human-computer interaction, and participatory art. Despite being a recent field of study, it already presents well-defined contributions and challenges to the interdisciplinary areas involved on it. Among the challenges, we highlight the lack of standardization in the protocols and data that circulate in these environments and also a possible lack of interoperability between devices. To contribute to the solution of this problem, this paper presents Sunflower, a tool that allows communication between different technological and musical elements present in an IoMusT environment, focusing on its management strategies based on a parallel established with other tools that help in musical practice via the network.*

1. Introduction

The Internet of Musical Things (IoMusT) arises from the expansion of the Internet of Things (IoT) domains to musical practice [Turchet et al. 2018a]. This area of knowledge is characterized by being multidisciplinary and encompassing several devices connected in a network, to exchange musical information between musicians and their peers, and between musicians and the audience members, proposing a drastic change in the way music is created and perceived [Turchet et al. 2020, Turchet et al. 2018b]. Its infrastructure predicts a new class of devices that are connected to the network and are capable of acquiring, processing, acting, or exchanging data that serve a musical purpose. These devices are called **musical things**.

Musical things are not useful on their own. Unlike traditional instruments or audio devices, that can be played in a standalone way, these devices are not physically connected, so it is necessary to think about two important points for them to interact with each other. The first point concerns the behavior of these devices. They must be context-sensitive and allow their behavior to be changed in the face of specific characteristics of each environment. An example is to allow certain equipment to adapt its conditions to resemble other elements found in the same network. It is also important to allow the software to be updated remotely or even changed more deeply to suit the needs and limitations of users [Turchet et al. 2018a, Vieira et al. 2020b].

The second point is also the most critical and it deals with the exchange of data among devices. For audio exchange, for example, they can accept different file formats, such as MP3 and WAV, while control can take place through the Musical Instrument Digital Interface (MIDI) and Open Sound Control (OSC) protocols, which can bring some

important issues about the interoperability between them. Besides, musical things are sensitive to latency and other aspects of the network.

IoMusT is an area that probably has the ability to revolutionize several fields in the borders between music and technology, like the public participation in interactive programs, remote rehearsals, music e-learning, smart studio production, and various other aspects of musical practice. However, this field also faces some problems, such as those related to socio-environmental issues, arising from the insertion of new technologies in society, or those inherent to artistic practice. From a computational point of view, the technological problems are latency, the design of the equipment, which can be poorly ergonomic, consume a lot of energy and network resources, and mainly, the lack of standardization of these devices. Furthermore, considering that the musical things are created by different researchers and companies, they may use data and protocols of the preference of their creators, and in most cases in can be out of standard [Turchet et al. 2018a, Vieira et al. 2020a].

In this context, the authors propose an architectural prototype for an IoMusT environment, called Sunflower. Able to exchange data in real time, its main purpose is to allow musical things to exchange data with each other without having to physically configure them. As this is an environment where communication takes place over the network, a connection is required throughout its use. Its structure is arranged in four layers: audio, video data, control, and management. The focus of this paper is precisely this last layer, detailing its development, importance, and main characteristics, where the management of the environment must be similar to that performed by sound engineers in musical presentations and network administrators in computer systems. That is, it must receive information from the devices and direct the data flow to those with an interest and ability to receive it. In light of this elucidation, it is important to emphasize that details about implementation and use of the environment as a whole, as well as network issues related to latency and synchronization will not be addressed.

Sunflower's management layer is inspired in other tools that work similarly, such as Libmapper [Malloch et al. 2013] and Medusa [Schiavoni 2013], and also in tools present in musical desktops, like ALSA MIDI[1] and QJackCtl[2]. Section 2 presents these technologies to attest that this form of control is already recurrent in musical practice and can be extended to the domain of the Internet of Musical Things, while more details on the Sunflower implementation are noted in Section 3. Section 4 discusses the similarities between Sunflower and the software and libraries presented throughout the text, as well as their influences on the layer developed by us. Finally, Section 5 brings summarized conclusions and indicates the future works to be developed.

2. Related Work

This section presents some graphical tools responsible for connecting objects with different characteristics and configurations. Given their contexts of use and functionality, they served as inspiration for our project.

[1]https://www.alsa-project.org/
[2]https://qjackctl.sourceforge.io/

2.1. Libmapper

Libmapper[3] is an open-source, cross-platform library that identifies data on a network, allowing arbitrary connections between them. For this, it creates a distributed mapping network and a close collaboration between the elements of this network, limiting the scope of its values. This makes it simple to connect a device to any object in the same environment. Its main focus is to provide tools for interactive control of media synthesis [Malloch et al. 2013].

This library focuses on some crucial points, such as the automatic discovery of pairs, connections from the name and data type, and description of these devices. It can also be integrated with applications developed in C, C++, Python, Java, Max/MSP, and Pure Data [Malloch et al. 2013].

Its operating mode is "plug and play", where musical keyboards, controllers, synthesizers, and digital musical instruments announce their presence on the network, in addition to making their input and output parameters available and able to communicate, in an arbitrary way, to other elements. An example of its Graphical User Interface (GUI) and the way it is connected can be seen in the Figure 1 [Malloch et al. 2007].

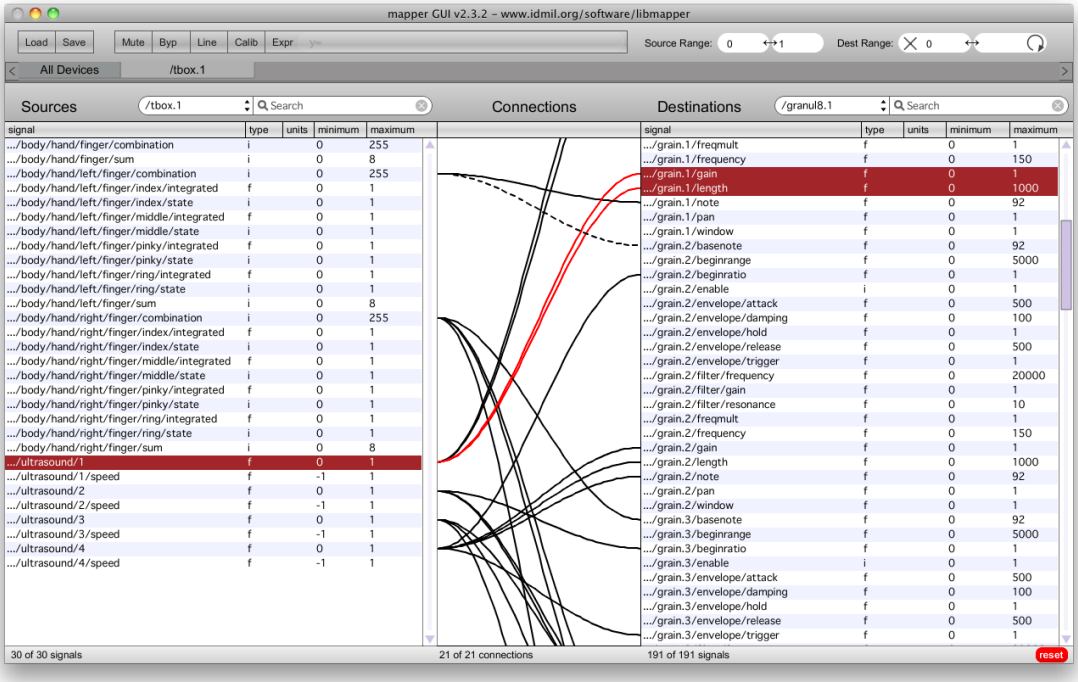

Figure 1. Libmapper's Graphical User Interface [Malloch 2010].

2.2. Medusa

Medusa [Schiavoni 2013, Schiavoni et al. 2011] is a distributed environment that allows the connection of several machines on a local network to send and receive audio and MIDI data. It does not have centralized servers, which means that each computer acts as both a server and a client. Furthermore, Medusa creates a shared environment characterized by

[3]https://libmapper.github.io/index.html

allowing sound resources to be reconfigured and used transparently. Its objective is also to allow communication between different systems and software.

This system is divided into three layers [Schiavoni et al. 2013a]: network, control, and audio. The first one is responsible for creating the connection between the network and the transmitted data. It implements different transport protocols, such as UDP, DCCP, TCP, and SCTP [Schiavoni et al. 2013b]. That way, the user can choose the one they prefer, which is faster or safer.

The control layer, in turn, is responsible for the management, packaging, and unpacking data. It distinguishes the elements of the network in consumers and suppliers. The third and final layer is more external and deals not only with audio data but also with MIDI [Schiavoni et al. 2013c]. In it, the functions of consumer and supplier are converted into plugins that exchange audio streams between machines.

This division guarantees the desirable characteristics of Medusa, which are: transparency, heterogeneity, graphic display with connection status, various types of input and output data, integration with legacy software, and sound processing capability.

2.3. ALSA MIDI

Since its creation, the Linux audio system was based on the Open Sound System (OSS), including all its limitations and restrictions. To remedy this problem, the Advanced Linux Sound Architecture (ALSA) [Kysela 2000] was created, which initially focused on the automatic configuration of the soundcard and manipulation of multiple devices in a system, but has evolved to support professional audio, 3D surround sound, advanced MIDI functions, software mixing, and audio multiplexing [Phillips 2005, Yu et al. 2015].

Its outstanding features are efficient to support the types of audio interfaces, fully modularized drivers, a library that simplifies the programming of top-level applications, and compatibility with programs made in OSS. As for the main objectives of the ALSA project, the following stand out: creating an open and modularized sound system for Linux; maintain backward compatibility with legacy applications; allow easy development of libraries; interactive configuration for the driver, etc.

Each soundcard can be identified by its input, output, sound control or by a string. The parameters of the soundboard, such as sample rate (44.1 kHz for stereos, and 48 kHz for home theater), bit depth (8, 16, 24, or 32 bits), and channel numbers are also informed.

Inspiration has also been extended to the control screen. Firstly, through aconnect, a textual system for connecting or disconnecting ports on an ALSA sequencer. Secondly, via aconnect-gui, a graphical utility that performs the same function, representing an evolution of aconnect.

2.4. QJackCtl

For network music practice, it can be useful to use an audio server. From it, it is possible to share files and functions, besides the synchronous execution of low latency clients. To monitor audio flows between clients and servers, the presence of graphical interfaces is required. A tool that plays this role is QJackCtl (Figure 2), a free and open-source software which provides a dialog box for managing the transport and connection status of network elements [Silva 2016].

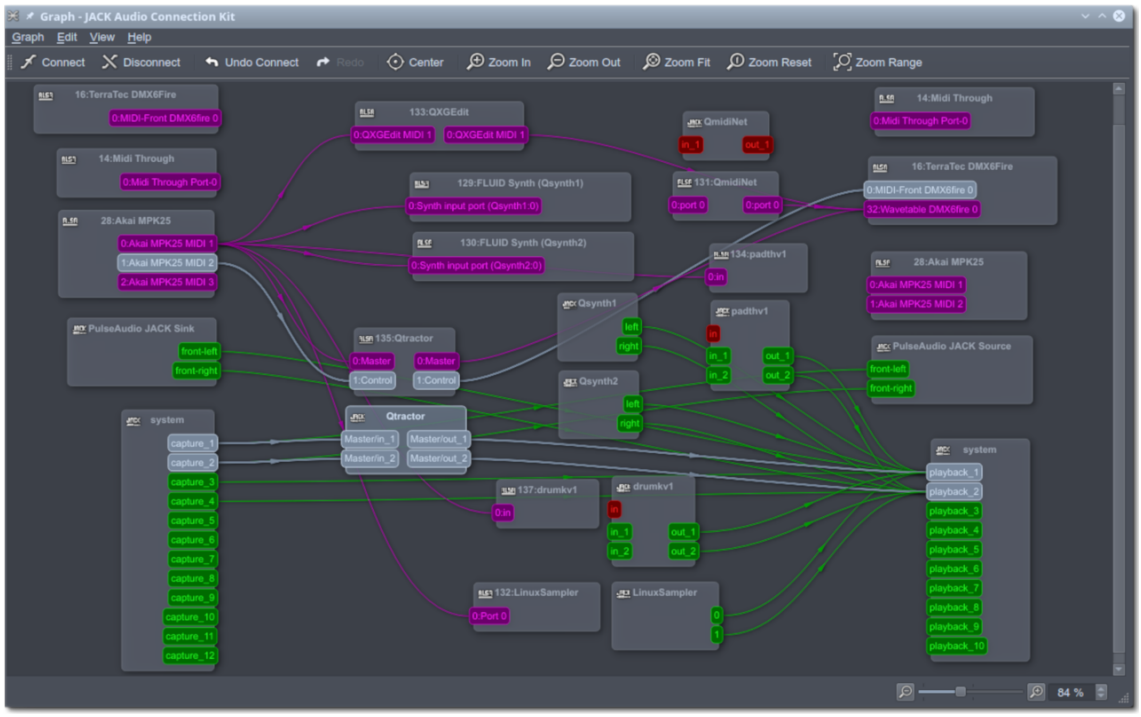

Figure 2. QJackCtl's Graphical User Interface.

Among its advantages are the fact that it is intuitive, as connections are made by point-and-click, connecting the output of one device to the input of another; it is structured, where the software guides make it clear which devices can connect; and offers a quick and easy way to integrate audio and MIDI.

As for the weak points, we can mention the inability to store the state; segmentation can cause doubts about those who are new to this type of system, and has some difficulty connecting or disconnecting cables.

3. Sunflower management environment

Defining the architecture of a system is an essential step in its development, as it indicates how the components of that system will interact with each other, as well as the details for its implementation. This section, therefore, is dedicated to presenting the architecture and the main characteristics of the Sunflower environment. Although it is still in the prototype phase, it is already capable of supporting numerous instruments and musical devices, in addition to having defined roles, being controlled by the network, and presenting data patterns to be exchanged.

Its structure is based on the Pipes-and-Filters software architecture model, since its functioning can be observed in various musical activities and tools. In this way, musical things will behave like filters, acting independently and without prior knowledge of their neighbors [Vieira and Schiavoni 2020]. The means used to send information, on the other hand, will behave similarly to the pipelines, which are solely responsible for sending information over the network, not changing the data that travels through them.

However, when faced with a large variety of data, this mode of operation faces three problems: inability to deal with data in different formats, possible overloads and

difficulty in reusing filters. To alleviate these problems, the system was divided into layers.

Several tools use layered architecture, such as operating systems and the TCP/IP protocol stack. In most cases, this structure is used to isolate implementation details from one layer to another, providing some maintainability for the system. In contrast to this, the layers in Sunflower are independent and do not need to be placed in the order in which they are presented in this paper.

We define layers according to the type of data exchanged between your devices in an IoMusT environment: an audio layer, a video layer, a control layer and a management layer. More details about each of them are shown below.

3.1. Audio Layer

The audio layer, as the name implies, is responsible for generating and sending audio data to the environment. For this project, a multitude of patches was created, responsible for simulating common objects in musical practice, such as a loudspeaker, a drum machine, and a pitchfork. They were also responsible for allowing the connection of microphones and instruments (guitar and bass) to the computer, sending their respective data to the network, and making the transformation from analog to digital audio and vice versa.

The audio engine was made in Pure Data, a programming language created by Miller Puckette, in the 1990s, belonging to a branch of the programming language family known as Max (which includes Max/FTS, ISPW Max, Max/MSP, etc). Initially conceived to remedy the deficiencies of these languages, its use has expanded and today it can be applied in musical creation, live coding, audio synthesis, composition assistant, and multimedia works [Puckette 1996, Vieira and Schiavoni 2020]. Pure Data was chosen due to its ability to have its properties edited while it is running, for allowing data to be sent to the network, and for being simple and easy to use.

Another point worth mentioning is the externals (or external libraries), which are developed by the community and expand Pure Data's functionalities [Puckette 1996]. One of the externals used in the audio layer was mrpeach[4]. This module is responsible for sending the audio over the network, in a UDP socket.

3.2. Video Layer

An element that can enhance a musical performance is video transmission. It can be used to capture, in real-time, the movements of the musician and/or the audience, or to display a videoclip that works together with the music to produce a desired artistic effect, in addition to offering the audience a new layer of control, unlike those inherent in audio.

Each of these examples uses an image encoding and decoding system, as it will travel over the network in bit format. This transformation is accomplished through a computational element. Here, Pure Data's multimedia capability was used to perform this function, in particular, Graphics Environment for Multimedia (GEM)[5] external, which is in charge of graphic processing in real-time, as well as reproducing data from DVD players, laptops, webcams or any other medium that has compatible input.

[4]https://puredata.info/downloads/mrpeach
[5]https://puredata.info/downloads/gem/

For the Sunflower environment, the video comes from an external webcam, with an image captured by the patch shown in the Figure 3, and sent via multicast to all machines with interest and the ability to receive this type of data.

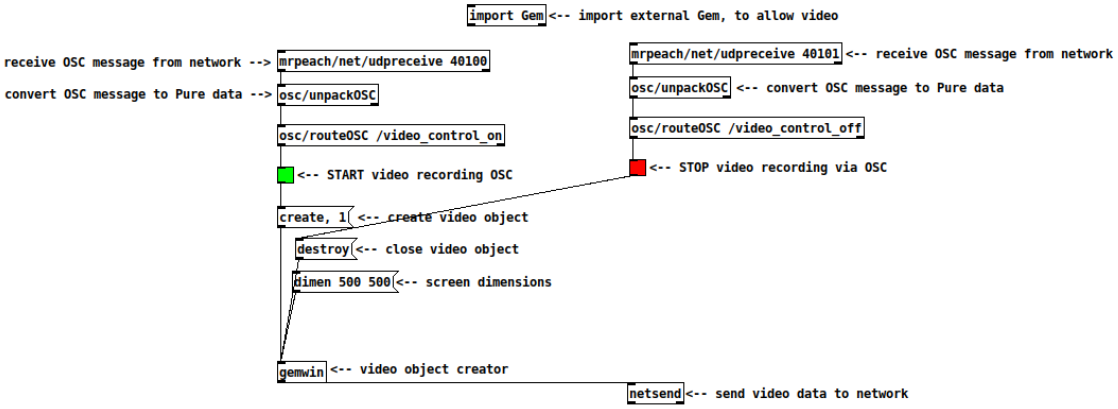

Figure 3. Patch responsible for capturing and sending video data over the network.

3.3. Control Layer

The control layer is made up of manager objects. It controls different actions in the audio patches, such as turning a sound source ON/OFF; change the drum machine's beats per minute (BPM); control the volume of the microphone and instruments; start or pause the recording; open files, and several others actions.

It is possible to exercise general or specific control over the patches that have this property. All of this is done by the network, sending these control messages packaged in the OSC[6] format.

OSC was created in 1997, by Adrian Freed and Matt Wright, and was initially developed for communication between computers, synthesizers, and multimedia devices. Among its advantages, interoperability, flexibility, organization, and robust documentation can be observed [Wright and Freed 1997].

OSC messages can be sent using UDP, without the need for prior knowledge of the host's IP, channel, or path, making it accessible either over the local network or the Internet. Because of this, he was chosen to compose the environment shown here.

This protocol can also be used as an alternative to MIDI, especially when a higher resolution and amount of data are needed. This is not to say that one protocol is better than another, just that it has slightly different characteristics and applications.

Figure 4 shows a microphone control patch, also made in Pure Data. It is important to highlight that for each element of the audio layer there is also an OSC controller, with small differences that reflect the particularities of the operation of each object. However, the operating logic and controllers data transmission are the same for everyone.

[6]http://opensoundcontrol.org/

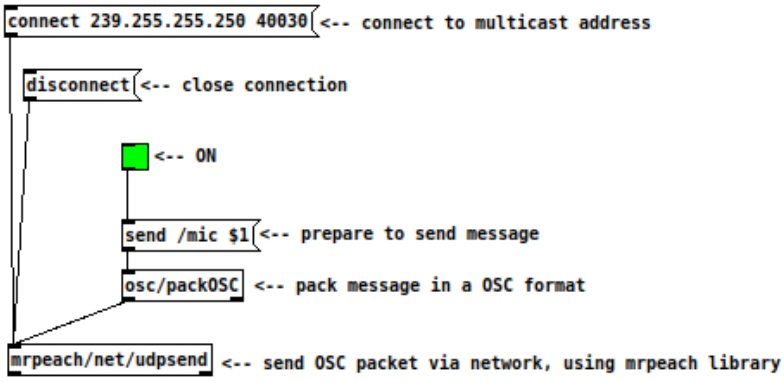

Figure 4. Microphone control via OSC.

3.4. Management Layer

In the computer networks architecture, it is essential to have an administrator who acts both in helping users, as well as in the configuration and maintenance of the network infrastructure. At concerts, there is the technician or sound engineer, who provides the necessary support to musicians. Here, these two roles are inseparable, with the administrator being responsible for ensuring the usability of the network and also taking care of sound aspects.

For this, the management layer was created to receive information from the audio devices available in the network, such as the connection status, indicating when a device entered or left the environment; port number, a key part in the connection, and responsible for indicating which port was chosen to receive input data; protocols, such as UDP for sending packets over the network or OSC, for controlling musical information; and bit depth, sample rate, and audio format, which convey information about the sound properties of the devices. In this way, the behavior of objects becomes analogous to the publish/subscribe method, where all tools publish their attribute and indicate their connection interests.

A sequence of commands responsible for capturing this data and sending it to the network is done in Pure Data. For practicality, all information is displayed on a screen made in Python. The management script containing the actions of some musical patches is present in the Figure 5.

```
/hello ['Hello']
/publish/thing Drum Machine ID#002
/publish/input INPUT: ID#002 -port: 40010 (ON/OFF) 40011 (BPM) 40012 (kick & snare & hi-hats) 40013 (Drum pattern) -protcol: UDP & OSC
/publish/output OUTPUT: ID#002 -audio(port 3000) -sample rate: 44100 Hz -bit depth: 16 bits -audio file format: wav
/goodbye Drum Machine says Goodbye!
/hello ['Hello']
/publish/thing Pitchfork ID#006
```

Figure 5. Part of the management screen, showing the patches that have been connected to the network.

Before addressing interoperability issues at this layer, it is important to use the

way a traditional audio environment works as an example. So, imagine a concert by a band, which has a microphone, guitar, bass, drums, and an audio system. It is not possible to directly connect the guitar's audio output to the microphone input. Or, make the sound generated in the bass to be played by some component of the drums. Given these restrictions on connections in musical practice, it is noteworthy that a musical thing cannot connect with all devices present in the environment. For this to happen, they must have some degree of compatibility. That said, the importance of this layer is highlighted, which even in a simple way, displays all the main connection characteristics of a musical thing, and it is up to the administrator to make the connection.

Figure 6 summarizes all these layers, showing their main elements, files and protocols.

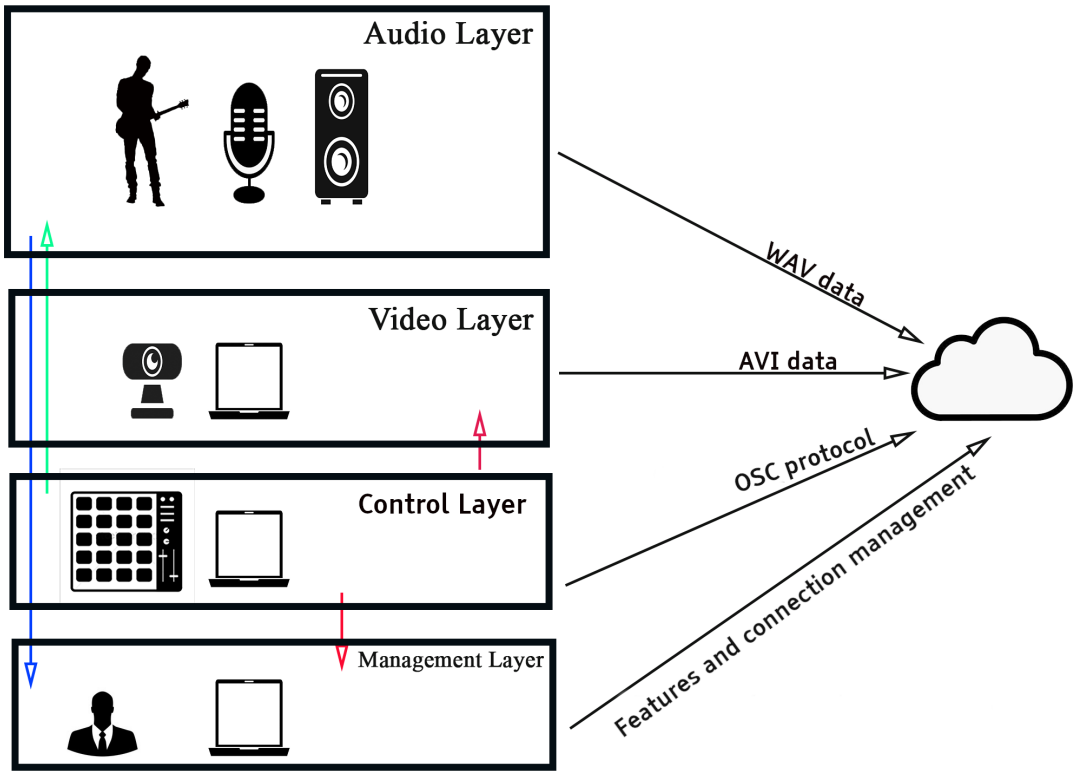

Figure 6. Basic Sunflower Architecture.

Practice Tests

After completing the development of the environment, tests were carried out to verify its functionality and measure the results obtained. Therefore, the authors ran tests on two machines, an Acer Predator Helios 300 laptop (computer A) with a 2TB hard drive, 16GB RAM, and an Intel i7 processor, and a Dell Inspirion 14 3442 3000 series laptop (computer B) with a 1TB hard drive, 8GB RAM and Intel i5 processor. The sending of information occurred in three different ways: in localhost, in a wired connection, using a twisted-pair cable, and in a wireless connection, via Wi-Fi.

Due to the Covid-19 pandemic, the authors were unable to run the tests with a group of people. Therefore, it was up to them to verify the results obtained. To keep the

focus of this paper, only the behavior of the management layer will be discussed in the next section.

4. Discussion

The purpose of creating Sunflower is to present a new way of contributing to what may be the biggest challenges faced by the IoMusT area, which are the lack of standardization of the data that travels on the network and the difficulty of obtaining interoperable objects. Thus, the environment in question intends to allow the use of different computational devices, from an architecture with similar functioning to Pipes-and-Filters and divided into layers, an approach different from those traditionally used in the area [Turchet et al. 2018b, Turchet and Barthet 2018, Centenaro et al. 2020].

For this, some desirable characteristics for the system were mapped, such as the client-server architecture, the self-management of objects and the types of messages that would travel on the network and, mainly, how the management layer should behave.

To justify this work, a systematic research was carried out to find tools with similar functioning. The first one is Libmapper, which associates with Sunflower by allowing the connection of different objects, treating them as filters with the capacity to receive and process data, which will later be sent to its adjacent through pipelines. Libmapper also has the ability to adapt OSC data to or from different ranges and dynamically change an OSC namespace. Using distributed mapping and identification by name to manage the objects were also taken into account in the environment developed here.

In the sequence, the Medusa environment corroborates with its well-defined division of layers. Like its management layer, Sunflower also distinguishes objects between suppliers and/or consumers of resources. Furthermore, it is assisted with its distribution of resources on the network, which enables the occurrence of a musical performance along these lines. Medusa also uses network messages to set up a musical environment and has different forms to exchange these messages using, for instance, command line and graphical user interfaces to provide these functionalities to lay users and experts too.

While the collaboration of the previous tools took place in the structure of the data and the performance, ALSA contributed mainly to the aesthetic aspect. Again, ALSA MIDI has different ways to manage the device connections, presenting the aconnect-gui management screen and aconnect command line interface. Using the same strategy, Sunflower also presents, so far, a screen that displays only textual information about what elements are present in the environment, as well as the ports and protocols it uses for connection and their sound properties.

QJackCtl presents, in a way, all the features listed previously, but its main contribution was concerning the possibility of listing the data present on the network. Although simple, this way of displaying data helps in organizing the system and also makes it easier for laypeople to understand.

So far, Sunflower does not have a data conversion layer. Hence the importance of musical things expressing their characteristics when they connect to the network. Thus, they will be able to find and send data to those devices that support the same types of audio configuration and also use the same network and music information protocols, ensuring that different gadgets can communicate based on this minimal similarity.

5. Conclusion and Future Work

The development of a tool for IoMusT proved to be a complex activity since it requires knowledge in several areas of computing, such as computer networks, signal processing, software engineering, and sound design, as well as the music itself. The target audience of these devices is formed of sound engineers and technicians, musicians, music teachers and students, and audience members who are interested in participating in the show. The tool must also contemplate the artistic-musical creation process and deal with any network problems.

From a general analysis, it was observed that the performances that happen in-network, as is the case of those intended by IoMusT, the identification of the elements by name and/or characteristics, the availability to publish or subscribe to certain services, and a control screen help a lot in the management of resources present in the environment. The division of it into layers with well-defined performance and attributions also helps in the process, since it is easier to identify and fix possible failures.

In-depth research showed that the techniques adopted in Sunflower were previously used, allowing musical connections and management of the environment in scenarios with different objects, ensuring robustness to the system. The Sunflower's strengths revolve around its ability to link the various instruments and audio patches connected to it, as well as allowing the administrator to have general and precise control of the environment. Among the weaknesses, we highlight the simplicity of the management layer, which only displays textual data, and the need for the administrator to manually execute the code.

This work was more concerned about how to manage an environment with such diverse aspects. To this end, it focused its attention both on the main characteristics of the devices, such as the protocols and audio formats used in the network communication, as well as on their ways of identifying themselves on the network and showing their connection status. Parallels have also been drawn between it and other technologies that work similarly.

Once the influences of related works are explained, importance is given to the architectural and sonic characteristics of musical things and the role to be played by the administrator of this system, the contribution of the Sunflower management layer to ensure the interoperability of the environments is observed. That makes the features of musical things clear, and it is only up to the administrator to connect them. The fact that they are identified and inform of their connection ports also facilitates this process. But as noted, it is common for this type of management to be done graphically and not just in a textual way. Consequently, for future work, it is intended to develop an interface along these lines for the Sunflower, making it even more intuitive and contributing to a more efficient workflow.

By no means do the authors intend to close the discussion on IoMusT environments, especially with regard to their forms of management. This work, therefore, is another contribution to the area, which can lead to improvements in live performances, ubimus environments with different technological artifacts, studio recordings, and music e-learning, as it will indicate the possibilities of connecting the musical objects in these contexts and make them easier.

Acknowledgments

Authors would like to thanks to all ALICE members. 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

Centenaro, M., Casari, P., and Turchet, L. (2020). Towards a 5g communication architecture for the internet of musical things. In *27th Conference of Open Innovations Association (FRUCT)*, pages 38–45.

Kysela, J. (2000). Advanced linux sound architecture (alsa) the future for the linux sound!? *Journées d'Informatique Musicale*, pages 1–7.

Malloch, J. (2010). Joseph Malloch libmapper. `https://josephmalloch.wordpress.com/portfolio/libmapper/`. Accessed: 2021-04-05.

Malloch, J., Sinclair, S., and Wanderley, M. (2007). A network-based framework for collaborative development and performance of digital musical instruments. In *Computer Music Modeling and Retrieval. Sense of Sounds. 4th International Symposium, CMMR 2007*, volume 4969, pages 401–425.

Malloch, J., Sinclair, S., and Wanderley, M. (2013). Libmapper (a library for connecting things). In *CHI 2013 Extended Abstracts*.

Phillips, D. (2005). A user's guide to alsa. *Linux Journal*.

Puckette, M. (1996). Pure data. In *Second Intercollege Computer Music Concerts*, pages 37–41.

Schiavoni, F. (2013). *Medusa: um ambiente musical distribuído*. PhD thesis, Universidade de São Paulo, Brazil.

Schiavoni, F., Queiroz, M., and Iazzetta, F. (2011). Medusa -a distributed sound environment. In *Linux Audio Conference*.

Schiavoni, F., Queiroz, M., and Wanderley, M. (2013a). Network music with medusa: A comparison of tempo alignment in existing midi apis. In *Sound and Music Computing Conference*.

Schiavoni, F. L., Queiroz, M., and Wanderley, M. (2013b). Alternatives in network transport protocols for audio streaming applications. In *Proceedings of the International Computer Music Conference*, Perth, Australia.

Schiavoni, F. L., Queiroz, M., and Wanderley, M. (2013c). Network music with medusa: A comparison of tempo alignment in existing midi apis. In *Proceedings of the Sound and Music Computing Conference*, Stockholm, Sweden.

Silva, F. A. F. (2016). Camadas tecnológicas da música feita através da rede de internet. In *Simpósio Brasileiro de Pós-Graduandos em Música*.

Turchet, L., Antoniazzi, F., Viola, F., Giunchiglia, F., and Fazekas, G. (2020). The internet of musical things ontology. *Journal of Web Semantics*, 60:100548.

Turchet, L. and Barthet, M. (2018). Jamming with a smart mandolin and freesound-based accompaniment. In *IEEE Open Innovations Association FRUCT*.

Turchet, L., Fischione, C., Essl, G., Keller, D., and Barthet, M. (2018a). Internet of musical things: Vision and challenges. *IEEE Access*, 6:61994–62017.

Turchet, L., Viola, F., Fazekas, G., and Barthet, M. (2018b). Towards a semantic architecture for the internet of musical things. In *IEEE Open Innovations Association*.

Vieira, R., Barthet, M., and Schiavoni, F. L. (2020a). Everyday use of the internet of musical things: Intersections with ubiquitous music. In *Proceedings of the Workshop on Ubiquitous Music 2020*, pages 60–71, Porto Seguro, BA, Brasil. Zenodo.

Vieira, R., Goncalves, L., and Schiavoni, F. (2020b). The things of the internet of musical things: defining the difficulties to standardize the behavior of these devices. In *2020 X Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–7.

Vieira, R. and Schiavoni, F. L. (2020). Can pipe-and-filters architecture help creativity in music? In *Proceedings of the Workshop on Ubiquitous Music 2020*, pages 109–120, Porto Seguro, BA, Brasil. Zenodo.

Wright, M. and Freed, A. (1997). Open soundcontrol: A new protocol for communicating with sound synthesizers. In *ICMC*.

Yu, H.-X., Xu, S.-J., and Fu, C.-K. (2015). An alsa audio system based on the linux system. In *International Conference on Advances in Mechanical Engineering and Industrial Informatics*.