Musical Communication Modeling Methodology (MCMM): A theoretical framework for event-based Ubiquitous Music Interaction

Flávio Luiz Schiavoni[1] *

Federal University of São João Del Rei
Computer Science Department
São João Del Rei - MG - Brazil
fls@ufsj.edu.br

Abstract. *This paper introduces Musical Communication Modeling Methodology (MCMM): a theoretical framework to develop context-aware interaction in music applications with ubiquitous devices. Music is changing its context everyday and many applications are being developed without an easy way to define the interaction and the semantics. The framework uses the event-driven model to drive user-to-user interaction based on the device-to-device communication. The framework itself is a set of activities and can orient developers to create collaborative and cooperative music applications.*

Keywords: Ubiquitous Music. Theoretical framework. Context-aware music application.

1 Introduction

Computer Science is changing its paradigm from the 1980's. In the past we had the idea of one machine to one person but nowadays it is a common situation that many people have many devices embedded in a rich environment[3]. One can say that Mobile devices are everywhere and the idea of being "always on" is now part of our daily routine. This situation is explored by ubiquitous music[9], a research field where everyday devices is used to make music.

Music making is a human activity that involves social collaboration and it can be used as a good metaphor for interaction. Ubiquitous music intends to explore this natural features of music and the ubiquitous feature of devices in a integrated way, putting together multiple users, devices, sound sources, music resources and activities[9]. The ubiquitous devices can provide several types

* Author want to thank Alexandre Torres Porres for english proofreading and Antonio D. de Carvalho J. for the assistance on the idea and figures. This work has been supported by the funding agencies FAPEMIG - Minas Gerais Research Foundation and UFSJ/PROPE.

of interaction: GUI, wearable, mobile, ubiquitous, continuous and discrete. All these interactions can be thought as sensors perceiving the environment[11] that can trigger different actuators. Powered by computer networks, it can be used to expand music making activity creating new models for music interaction [7].

In this paper we propose a theoretical framework to create event-based music interaction based on the idea of several devices connected to sensors and actuators. The theoretical framework presented here is free of architecture, programming language, device type or implementation, and can be used as a guide to musical application development. The proposed framework is focused on event-based communication but it can be easily expanded to data stream communication like audio or video. Data stream commonly demands more network bandwidth, data transformations to integrate heterogeneous systems (like sample rate, endianness or bit depth change) and buffering implementation to synchronize network packets. More on network music and audio transmission can be found on [12].

The remainder of this paper is organized as follows: Section 2 presents related works and fundamentals, Section 3 presents the proposed framework activities, Section 4 presents the framework draft and Section 5 presents the Conclusion.

2 Related works and fundamentals

There are many works about music interaction in computer music research. The most discussed ideas have a focus on the interaction of a musician and a device. This scenario is important because it raises several discussions about how a user can interact with a device and make music [17,10,4]. Some activities may permit musical interaction through the sound emitted by devices at an acoustic environment without exchanging data or meaning. In this scenario one user's action hardly affects directly how other users are playing their own device. In our research, we plan to go further and focus on user-to-user musical interaction regarding the device-to-device communication. This model can also be extended to several musicians and several devices.

The main aspects of network music were already discussed in two seminal works from Barbosa and Weinberg. The first is Barbosa's classification of collaborative music using computer technologies [1]. In this work, the musical networks are distributed depending on their location and interaction, as one can see on Fig. 1. The network music systems that fit in this diagram afford to share data between users and to create a musical environment that can provide many possibilities of interaction in real time or not.

The second work is Weinberg's concept of Interconnected Musical Networks, and topologies of networks in the scope of music [15]. The interaction and influence between the players are the key aspects in this work, as presented on Fig. 2. These diagrams represent a relation between centralized and decentralized topologies with the synchronous aspect of the interaction experienced by players. The hub that is presented on diagrams 2.a and 2.b points out a centralized topology where each user creates its own musical data and everything is

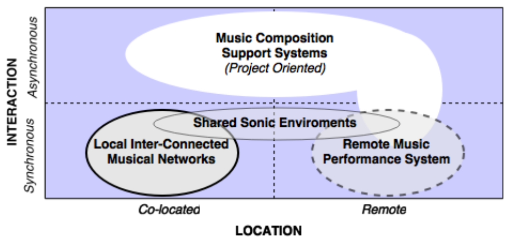


Fig. 1: Barbosa's classification space for collaborative music supported by computer systems [1]

grouped on the hub. On the other hand, the decentralized topologies presented on diagrams 2.c and 2.d indicate a possibility of interaction between users. In this way, each user can cooperate directly with the musical material created by another user, and a defined order may be considered.

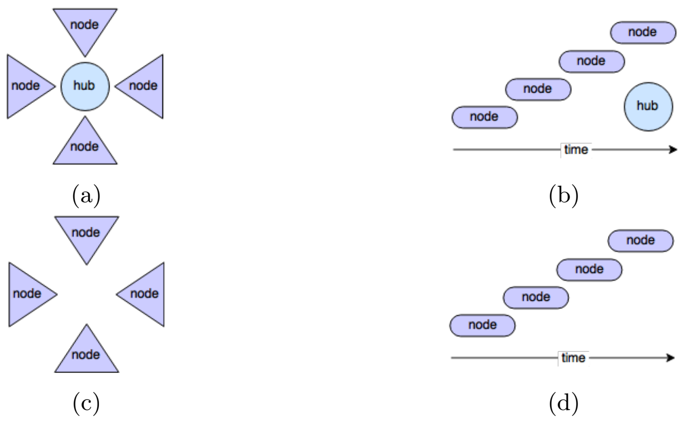


Fig. 2: Weinberg's description of network topologies regarding the social organization of players [15]

These works clarify the many spatial organizations of musical interaction systems over the network structure, but their approaches are focused on high-level classification. Although Weinberg takes care describing the social aspect of players interaction, the musical meaning of the events shared by users is set aside as much as the communication models.

In contrast to application or implementation, in which case the development can reflect a single and closed model, our approach emphasizes an open model to integrate different applications in a distributed music environment. It does not consist of a single unified model but a way to map cognitive actions to music in a collaborative way. This approach implies that the mapping outreaches the simple idea of exchanging raw data between two points, and aims at more meaningful messages.

The communication structure needs some attention at this point. A model presented in Fig.3 was created by Shannon in 1948 and is widely used on communication. This model was the basis for Information Theory and comprises the most important keys to transmit a message. Shannon's research about communication continues after that model trying to figure out the best way to communicate in two-way through the same channel in an effective way [14].

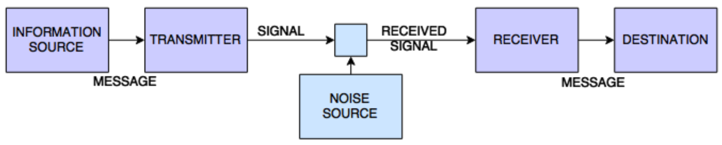


Fig. 3: Shannon model of a communication system [13]

The discussion about communication is extended depending on the context of the area. In the marketing field, the structure of the communication pays attention to the feedback from the receivers of the messages. An important diagram is presented on Fig. 4 and is very similar to the one proposed by Shannon.

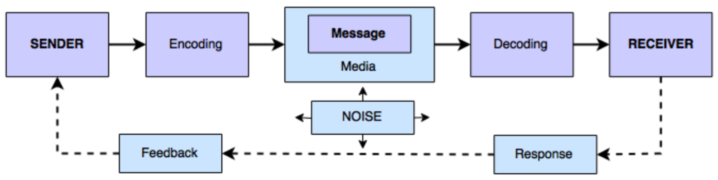


Fig. 4: Kotler diagram of communication process [8, p. 480]

These fundamentals serve as a base of the ideas presented on the framework - discussed in the next section.

3 The proposed framework activities

The proposed framework starts trying to answer a question: "If I want to to develop a collaborative musical instrument using portable devices, what should I do?". We enumerated several activities to help answering this question dividing the development into simple activities that can be realized in group or individually.

In this work, we divided the development of an event-based music interaction application in a set of activities to address implementation issues in an independent form. This is, in our point of view, a basic set of activities to develop a context-awareness application for devices and sensors. Our framework work flow is based on 6 different basic parts, as presented on Fig.5.

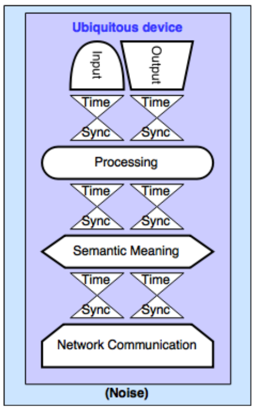


Fig. 5: The framework workflow

Like the TCP/IP stack, we arranged different parts of the framework into layers that are responsible for different activities and abstractions. So, it is possible to describe each activity and each layer as a small problem to be solved locally. Furthermore, like the TPC/IP stack, every device would have the same layered architecture to grant a communication stack and facilitate the message exchange.

As depicted on Fig.5, we enumerated 6 activities: Input, Output, Time Synchronization, Processing, Semantic Meaning and Network Communication. In this representation we have only one Input and Output to make it clean and easy to understand although some devices can have diverse sensors and also outputs. Time Synchronization are an optional task and some application development will not need to use all these activities. The idea of this workflow is to describe the full path to dispatch an event from one device to every device connected to a environment every time a new event occurs on the Input.

3.1 Input (Sensors)

Since we intend to have human interaction, the basic Input in our framework is a sensor listener. Sensors are used to map users' gestures and environment activities. It would be possible to have a software input created by an algorithm, another software, or a file, indeed. Since we intend to have users, a sensor is a computational way to capture or listen to states or changes in user's activities in the environment in analog or digital sampling. Sensors can be found embedded on mobile phones, notebooks, notepads, tablets and also can be attached to different devices like Arduino, Galileo or Raspberry Pi [6].

Different devices may have different sensors and the same sensor can have different configurations and constraints. An example for this statement is a touchscreen that can be found in different sizes depending on the device. The size of a touchscreen is a sensor constraint and it will have computational effects on the data communication, as we will see at Subsection 3.5.

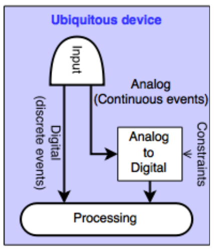


Fig. 6: Input activity

The idea of using a sensor to create and dispatch an event also follows the paradigm of monitoring a sensor to create reactive applications [5]. An observation of a sensor can bring at least two values: what changed on a sensor and when it happened. It is important to notice that a sensor value can be a parameter or a set of parameters. A touchscreen, for instance, brings X, Y position of the latest touch event while an accelerometer can have X, Y, Z values regarding the acceleration on three dimensional axes. The identification of the owner of the sensor can be a relevant data to report in some cases. Thus, we can monitor what, when, and where an event took place in the environment.

Electronically, a sensor can have a continuous or discrete signal. In the case of an analog sensor, it will need a conversion to discrete values based on some constraints before sending an event to a digital system, as presented on Fig.6. The sensor observation captures discrete instances of these parameters every period of time, or better put the sensor sample rate. Sensors with a high sample rate are normally more accurate and also more expensive than other sensors. We consider that the conversion from analog to digital is an extension of the input. This conversion is not managed at the Processing activity because this activity acts as monitor of digital events, as we will present at Section 3.4. Moreover, most of the analog sensors can be found as digital sensors, and in this case they will dispatch the events in some discrete scale.

3.2 Output (Actuators)

A device uses its output actuators to locally reflect a change in the environment. It is also possible to consider logical components in the Output activity such as an event recorder software, so one can have a score from the local sonic environment. Common actuators like sonic feedback, haptics feedback and visual feedback can be used as a context-awareness feedback for the application from the output point of view.

It is clear that different devices can have different actuators. Also, just like the sensors, actuators can have different constraints that influence the outcome. It leads us to some local decisions about how to react to some received message. One application can choose to have a sonic output to a message while another application can have only a visual output to the same message. The example depicted on Fig.7 illustrate some possibilities of actuators output in a certain device.

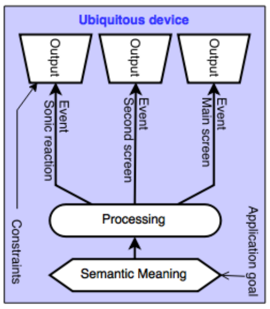


Fig. 7: Output activity

Output is also used to give feedback to the user's own actions. User's interaction with the device can be reported with some feedback to the user and this feedback can also be merged with the environment messages.

3.3 Time Synchronization

Music can be explained as sound events organized in time. Since our focus is musical interaction, time synchronization can be necessary. This layer appears several times on Fig.5 because Time Synchronization activity can be necessary to synchronize different data. The basic idea of time sync is to have event ordering. In several scenarios, the order of the events is important and it is undesirable to have the second event before the first [3].

If the timing of users' action or environment changes is critical, the time synchronization needs to occur right before the Processing. In this framework proposal we assumed that the Processing can have multiple threads or dispatch events in different order. On the other hand, if the timing is not so critical, and the Processing is expected to change the sampling rate or discard some events, it is better to have a Time Synchronization activity after the Processing or avoid using this activity.

Since network communication can bring latency and jitter to the application, it can also be necessary to synchronize event messages on the sender / receiver and keep the order of the events when necessary. A time-stamp field can also be used with a local ring buffer to synchronize received events with different latency (jitter). In musical interaction through the network, the packet loss is a common drawback of unreliable connections and the medium can be the source of most noise and interference at the communication process. One can imagine a message to start a sound and another message to stop a sound as a common Use Case to play a musical note, but if the second message is lost, the note will be played forever. We can also illustrate the same result if the Network Communication delivers the second packet before the first one. On the other hand, some applications may not have this problem if the events are independent, like *play A4 for 1s*, and the sonic result is not time aligned.

Another important point is the necessity of defining a clock for the synchronization process. Synchronization can be done based on a local clock, a global clock, a relative time clock, and also a virtual clock adjusted on periodically. A single representation of Time Synchronization activity is presented at Fig.8

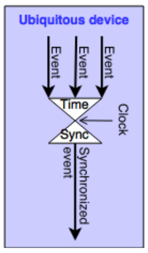


Fig. 8: Time Synchronization activity

Furthermore, message ordering can be done using an auto-increment for every new event or attaching a time-stamp field on the event. This activity can hold events on a buffer before sending in order to assure the synchronization.

3.4 Processing

Sometimes, an Input sensor will keep the same value for a long period of time. In this situation, it can be necessary to avoid the generation of several messages to the environment with the same value. For this reason, a processing of the input value can help to guarantee a new event to update the environment only if the change in the value is really significant, for example.

Also, it is possible to a) convert a continuous value to a discrete value, b) convert a discrete value into a constrained value based on some rules, c) generate an event only if a change is really significant, d) apply the input constraint to a value to decide if it is possible to react to an environment change. From digital signal processing research we can grab plenty of filters that may be used here in case the data from an event differs from the type of data required to be sent.

A threshold can be applied to a sensor value to estimate a minimum change that will be reported. A Low Pass Filter can be used to avoid reporting drastic changes of a value and to create a smooth event output. In this way, the Processing is responsible to map every user interaction to an event or a group of events, independently of the interaction model[6]. A representation of the Processing activity in two different situations is presented on Fig.9

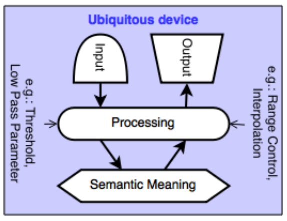

Ubiquitous device
Input
Output
Processing
Semantic Meaning
e.g.: Threshold, Low Pass Parameter
e.g.: Range Control, Interpolation

Fig. 9: Processing activity

This activity can fit better after an Input or Semantic meaning, depending on the application context in each device. The events dispatched from Semantic Meaning activity can also be filtered by Processing activity since a message may need an adjustment before being sent to the Output. The Processing may be used to redefine the range of values that an Output will receive after some point. A drastic pitch change, for instance, can create a glitch effect on the sound, and for this reason, even when every message changes the pitch value, it can be necessary to have an interpolation ramp between the previous value and a new one.

3.5 Semantic meaning

The Semantic Meaning activity will map the received message to the specific end for each actuator to assure a local reflection of this environment change, and in the end we will have only final meaningful events. This final event must have a semantic meaning instead of a raw value because an isolated event discards its original context and lacks a semantic meaning. For this reason, a semantic model is necessary to associate a particular user interaction and a desired sonic result. This association is required to: map one or more event to an acoustic result; create a message with semantic meaning, and; normalize or standardize the data representation [7]. In addiction, semantic models are more abstract than notations and can decouple the gesture from the acoustic result creating a common agreement to the environment[3].

Another reason to use a semantic meaning layer considers the device constraints. If a drum set is played in a touchscreen, for instance, the touch position must be mapped to the desired drum pad and this message must be sent to the network and not a single two-dimensional parameter because the drum position on the screen can vary depending on the device size settings. So far, it is necessary to give locally a semantic meaning because only the user's device knows its own settings.

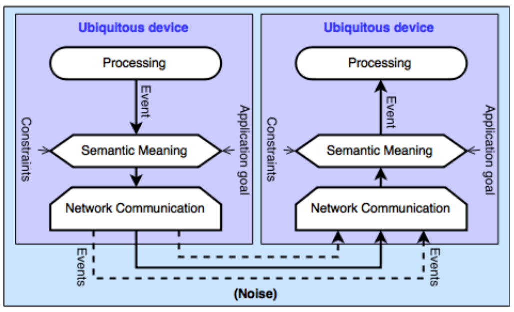


Fig. 10: Semantic Meaning activity

Since a device can have several sensors, a semantic model can also group several events from one or more sensors into one single message in order to generate a new unique event. Using this fusion paradigm it is possible to have a more accurate vision of the environment and reflect a more accurate sonic result. It means that one can acquire user's position, gesture, localization and other events from available sensors and merge in one single message in the end. This single message can be used to change the audio volume of an instrument in another point and therefore change the environment with a sonic result.

Another reason to use a semantic meaning layer is related to the application constraint. A touch position from one device can be used to play the piano, the drums, the xylophone or to control a sequencer, an audio effect or any other musical instance in other devices. These messages can eventually be interpreted by the second device without any additional mapping since the network message has semantic meaning and not X,Y positioning.

From the communication point of view, the event needs to be interpreted by the Semantic Meaning activity considering the application context at the receiver environment. Although all participants need to talk in the same language to ensure the communication, the semantic meaning can be used to adapt the message to another context at a specific environment. In this case, the Semantic Meaning activity at the receiver may or may not share the same semantic model from the sender, but will act in the same way, receiving and mapping several events

It can also use different techniques to respond softly to an environment event. Imagining a common instrument played by several musicians, the note pitch can be defined as an average of the actual value and a received value or it can change its value on every received message. Altogether, the Semantic Meaning activity will define a **group of messages** and send different events through the network in order to notify the whole distributed environment. Fig.10 presents an overview of this activity in some possible situations.

Once the group of messages is defined, it is also necessary to define a network format to exchange these messages. A network message should be encoded in common formats, like text-plain, JSON, XML and OSC, before being sent. The latter is considered the most used network data format in music context [16] since it is supported by different music programming languages like Pure Data, CSound or Supercollider and several music applications. Other formats may be applied, like the byte chunks used by MIDI or any serialization method.

At this point, the codification of the message can also include any cryptography depending on the medium used. The receiver should be aware of the message format, decode the event, interpret the message based on its own semantic model, and create another event to be dispatched by the Semantic Meaning activity.

3.6 Network communication

We need to exchange messages to other users to notify the environment about some new event. In the presented framework, the network communication layer is responsible to exchange messages on the environment. As our aim is to ensure communication between many users, a group communication paradigm should be used to fulfill our proposal with specific solutions depending on the communication medium.

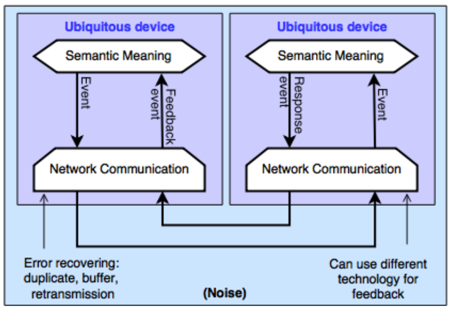


Fig. 11: Network Communication activity

In Local Area Networks (LAN), a Broadcast or Multicast addressing methodology can be used, and for World Area Networks (WAN) communication, a central relay server is a common solution. Hybrid solutions can mix different network addressing methodologies depending on the desired performance structure. The *buzzword* Cloud Computing can be another specific solution that extends the functionality of WAN with the addition of distributed servers and cloud services. The Network Communication activity can also be used with technologies that interconnect devices in a more direct way. One can cite the wired connections, or the wireless options like Infrared, Bluetooth, and the Wi-Fi Direct.

Network communication turns out to be a critical part of any framework when a performance using a musical instrument controlled through the network requires very precise timing. Thus, network latency and jitter can interfere adversely on the application usage. Normally, LAN has lower latency and jitter than WAN but it varies depending on the network infrastructure, number of devices, protocol choice and others implementation choices.

All of these alternatives have their own constraints. The selection of the technology depends on the interfaces supported by devices at the sonic environment, and some devices may have more than one interface that can be used at the same time. Fig. 11 shows the main characteristics of Network Communication activity.

4 The framework draft

In this section we present a draft of the MCMM activities. Here we will present a guideline to start a new application based on this framework. It is necessary to describe each activity details prior to start developing. In this way you can have an open view of the constraints of the musical environment, evaluate the application goal, come up with solutions for the issues, and have a general representation of the application developed beforehand.

This theoretical framework comprises a group of ideas that can support the study and development of applications for music interaction based on events. We can have some graphical representation like the ones on Figure 12 in order to better presenting the applications, and we need to use textual descriptions based on the draft below if we need to specify the details regarding each activity included in the framework We may need to describe other applications that can be connected or interact with the musical application during a performance in case we want to suggest possible ubiquitous interaction.

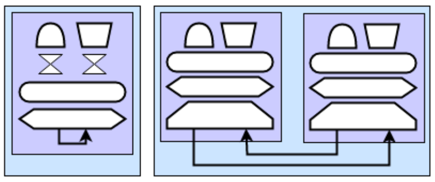


Fig. 12: Examples of graphical representation using the framework

1. **Input**: Read and dispatch values from a sensor or a set of sensors that can monitor user's gesture and the environment.
   - To Define: Sensors and Devices.
   - Input: Continuous or discrete sensor value.
   - Output: Discrete sensor value.
2. **Output**: Use actuators to reflect an environment change or a user's action.
   - To Define: Actuators.
   - Input: Discrete value.
   - Output: Discrete or continuous value.

3. **Time Synchronization**: Synchronize events based on a clock. Incoming events can be buffered in order to wait for a specific time.
   - To Define: Time sync clock. Buffer type. Localization of the syncs.
   - Input: Event with or without timing information.
   - Output: Event with or without timing information. Event on time.
4. **Processing**: Filter and monitor values.
   - To Define: Threshold, ramps and filters.
   - Input: Raw data in discrete format.
   - Output: Raw data in discrete format.
5. **Semantic Meaning**: Transform a raw data into a musical message associating a semantic meaning to an event, or decode a musical message into raw data before sending to an output. This activity is also called **mapping** and depends on the application goal.
   - To Define: Application goal. Message format. Message mapping. Context.
   - Input: Raw data, Message with meaning
   - Output: Message with meaning, Raw data
6. **Network Communication**: Send and receive messages from and to the environment.
   - To Define: Network addressing methodology. Network transport protocol.
   - Input: Message event
   - Output: Message event

Additionally, some skills may be required by the developers and users of this framework. For input reading, some skills on sensors and microelectronic may help to deal with the technical aspects of the electronics components. Experience with message packing syntax may help at the mapping activity, and a good knowledge on network implementation will be necessary to assure a good communication between devices. Depending on the application goal, the developers may need some skills in synthesizer implementation. In addition, the output manipulation depends on the expertise regarding the actuators, while the processing will be mostly based on DSP fundamentals.

4.1 Case study I: Sensors2OSC

Some applications from music interaction field can be described using this theoretical framework. In order present our theoretical framework applied in a real case, we are going to describe an application named Sensors2OSC [2] already developed by the authors.

Sensors2OSC is a mobile application that sends all sensors events using OSC through the network. A user just needs to select the sensors and set an IP and Port in order to start sending the new events. At another point, we can receive the OSC messages using any program and use the values to control whatever we want.

Figure 13 presents the application structure with two instances of Sensors2OSC on each side, and a computer music program in the middle. The final context of an interactive application using Sensors2OSC is described on Table 1. We believe that both representations are sufficient in any case.

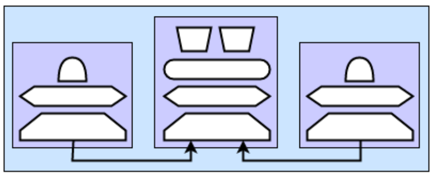


Fig. 13: Sensors2OSC presented with this framework

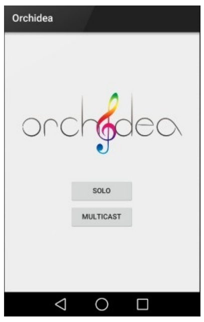
Table 1: Sensors2OSC description using MCMM

Input	To define	The sensor
	Input	Continuous values
	Output	Digital events
Semantic Meaning	To define	OSC address correlated to the sensor
	Input	The sensor ID and value
	Output	Message with OSC address
Network Communication	To define	Unicast or Multicast
	Input	TCP or UDP packets
	Output	TCP or UDP packets
Semantic Meaning	To define	Interpret OSC addresses
	Input	Message with OSC address
	Output	Raw value of the sensor event
Processing	To define	Optionally the user can filter the value
	Input	Value of the sensor event
	Output	Value of the sensor event
Output	To define	Any synthesizer or program
	Input	OSC messages
	Output	Continuous audio signal or program update

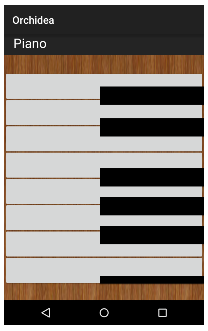
4.2 Case study II: Orchidea

The Orchidea is a project focused in the development of an Orchestra of Mobile (Android) Devices, presented in Fig. 14. This project is using MCMM as a base to the development.

Orchidea Input, initially, is a cellphone touchscreen. Other sensors can be used but our initial development used only the touchscreen. The output is the sound and uses libpd and Puredata patches as the synthesizer. Thus, it was



(a)



(b)

Fig. 14: Orchidea GUI

possible to detach the sound design from the programming. There are a message mapping by semantic meaning by instrument development and the network communication uses OSC and multicast.

The development of a new instrument in Orchidea depends on a) the creation of a touchscreen, b) the message definition, c) the synthesizer creation on Pure Data.

MCMM activities helped this project definition and development and worked as a guide to the application development.

5 Conclusion

In principle, mobile phones were developed for people communication purpose. Once they became pocket computers, they have being used to music making activities. Several music making applications were developed focused on a single user activity and progressively taking advantage of the communication capability of devices.

This paper presented MCMM, a theoretical framework to develop Event-based music applications with a communication layer to allow user-to-user interaction based on device-to-device communication. This framework enumerated a group of activities, defined a development workflow and presented some technical issues in every activity.

Since our goal was not focused on implementation details, this framework can be used with any programming language, device type or music application type. Moreover, it can put together in the same musical environment different applications and devices, from desktop application to notebooks, mobiles or other devices. It is also important to notice that we can also use this framework to describe most of the applications already developed for musical interaction. Authors encourage the idea that this framework will serve as an starting point for instructing developers and musicians on modeling and sharing the structure of their applications with lay audience and users in a near future.

References

1. Barbosa, A.: Displaced soundscapes: A survey of network systems for music and sonic art creation. In: Leonardo Music Journal 13, pp: 53–59 (2003)
2. De Carvalho Junior, A. D. and T. Mayer.: Sensors2OSC. In: Sound and Music Computing Conference, pp. 209-213. Maynooth (2015)
3. Dix, A. J.: Towards a Ubiquitous Semantics of Interaction: Phenomenology, Scenarios, and Traces. In: Proceedings of the 9th International Workshop on Interactive Systems, Design, Specification, and Verification, DSV-IS 02, pp. 238–252. London, UK, UK, Springer-Verlag (2002)
4. Luciano V. Flores, Marcelo S. Pimenta, Damián Keller. Patterns of Musical Interaction with Computing Devices. In: Proceedings of the III Ubiquitous Music Workshop (III UbiMus). São Paulo, SP, Brazil: Ubiquitous Music Group (2012)
5. Hinze, A., K. Sachs, and A. Buchmann.: Event-based Applications and Enabling Technologies. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS '09, pp. 1:1–1:15. New York, NY, USA, ACM (2009)
6. Kernchen, R., M. Presser, K. Mossner, and R. Tafazolli.: Multimodal user interfaces in ubiquitous sensorised environments. In: Intelligent Sensors, Sensor Networks and Information Processing Conference, pp. 397–401 (2004)
7. Malloch, J., S. Sinclair, and M. M. Wanderley.: Libmapper: (a Library for Connecting Things). In: CHI '13 Extended Abstracts on Human Factors in Computing Systems, CHI EA '13, pp. 3087–3090. New York, NY, USA: ACM (2013)
8. Kotler, P.: Marketing Management, 14th Edition. Prentice Hall (2014)
9. Pimenta, Marcelo S., Flores, Luciano V., Capasso, Ariadna, Tinajero, Patricia, and Keller, Damián.: Ubiquitous Music: Concepts and Metaphors. In: In Proceedings of the 12th Brazilian Symp. on Computer Music, pp. 139–150. Recife, Brazil (2009)
10. Radanovitsck, E. A. A.: mixDroid: Compondo através de dispositivos móveis. Ph.D. thesis, Universidade Federal do Rio Grande do Sul (2011)
11. Realinho, V., T. Romão, and A. E. Dias.: An Event-driven Workflow Framework to Develop Context-aware Mobile Applications. In: Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia, MUM '12, pp. 22:1–22:10. New York, NY, USA, ACM (2012)
12. Schiavoni, F. L., and M. Queiroz.: Network distribution in music applications with Medusa. In: Proceedings of the Linux Audio Conference, pp. 9–14. Stanford, USA (2012)
13. Shannon, C. 1948.: A mathematical theory of communication. In: Bell System Technical Journal, The, 27(3):379–423 (1948)
14. Shannon, C. E., et al.: Two-way communication channels. In: Proceedings of 4th Berkeley Symp. Math. Stat. Prob, Volume 1., pp. 611–644 (1961)
15. Weinberg, G.: Interconnected Musical Networks: Toward a Theoretical Framework. In: Computer Music Journal, pp. 29:23–39 (2005)
16. Wright, M.: Open Sound Control: an enabling technology for musical networking. In: Organised Sound, pp. 10:193–200 (2005)
17. Young, J. P.: Using the Web for live interactive music. In: Proceedings of International Computer Music Conference, pp. 302–305. Habana, Cuba (2001)