MTCF: A framework for designing and coding musical tabletop applications directly in Pure Data

Carles F. Julià
Universitat Pompeu Fabra
138 Roc Boronat
Barcelona, Spain
carles.fernandez@upf.edu

Daniel Gallardo
Universitat Pompeu Fabra
138 Roc Boronat
Barcelona, Spain
daniel.gallardo@upf.edu

Sergi Jordà
Universitat Pompeu Fabra
138 Roc Boronat
Barcelona, Spain
sergi.jorda@upf.edu

ABSTRACT

In the past decade we have seen a growing presence of tabletop systems applied to music, lately with even some products becoming commercially available and being used by professional musicians in concerts. The development of this type of applications requires several demanding technical expertises such as input processing, graphical design, real time sound generation or interaction design, and because of this complexity they are usually developed by a multidisciplinary group.

In this paper we present the Musical Tabletop Coding Framework (MTCF) a framework for designing and coding musical tabletop applications by using the graphical programming language for digital sound processing Pure Data (Pd). With this framework we try to simplify the creation process of such type of interfaces, by removing the need of any programming skills other than those of Pd.

Keywords

Pure Data, tabletop, tangible, framework

1. INTRODUCTION

In the past decade we have seen a proliferation of musical tabletops. Currently, so many "tangible musical tables" are being developed that it becomes difficult to track every new proposal[1].

Independently of the relevant differences that can exist between these systems, scholars tend to agree in the benefits of interacting with large-scale tangible and multi-touch devices. Their vast screens make them excellent candidates for collaborative interaction and shared control [2][4], while favoring at the same time, real-time, multidimensional as well as explorative interaction, which makes them especially suited for both novice and expert users [6]. This last author also states that the visual feedback possibilities of this type of interfaces, makes them ideal for understanding and monitoring complex mechanisms, such as the several simultaneous musical processes that can take place in an interactive digital system for music performance [5].

This growing tabletop popularity, clearly in the musical domain but also in other fields, has increased the publicly available information for the rapid development and prototyping of these types of interfaces. Online communities of DIY builders such as the NUIGroup[2] collect large knowledge bases of resources and many easy-to-follow tutorials are publicly available [12]. The development of this type of hardware solutions has indeed become easier and affordable than ever, allowing practically anyone to experiment with tabletop computing.

From the software side, several well-known open-source solutions do also exist, both for the tracking of multi-touch fingers, such as the NUIGroup's Community Core Vision[3], or for the combined tracking of fingers and objects tagged with fiducial markers, such as reacTIVision [1]. These and other existing software tools greatly simplify the programming of the input component, essential for this type of interfaces, but this solves only one part of the problem. The visual feedback or the graphical user interfaces, which do often also include problems specific to tabletop computing, such as aligning the projector output with the camera input or correcting the distortion that results from the use of mirrors, still have to be manually programmed. Not to mention the underlying musical engine, our main reason after all for developing this type of applications.

Taking into account these considerations, it may be difficult to acquire the required skills for being capable of programming the visual interface and the audio component, even to find a single programming language or framework supporting well these two components.

A simple solution to this last problem, as presented in previous papers such as [4][3], is to divide the project into two different applications: one focused on the visual feedback and another focused on the audio and music processing. However, dividing the tasks will not eliminate the need for programming still on both sides. The system we present here has been designed for simplifying these technical difficulties.

2. MUSICAL TABLETOP CODING FRAMEWORK

Musical Tabletop Coding Framework(MTCF) is an open source framework for the creation of musical tabletop applications that takes a step forward in simplifying the creation of tangible tabletop musical and audio applications, by allowing developers to focus mainly on the audio and music programming and on designing the interaction at a conceptual level (because all the interface implementation will be done automatically).

MTCF provides a standalone program for the visual in-

[1]Kaltenbrunner, has a website devoted to Tangible Music, which includes a quite exhaustive list of devices: `http://modin.yuri.at/tangibles/`

[2]`http://nuigroup.com`

[3]`http://ccv.nuigroup.com/`

NIME'11, 30 May–1 June 2011, Oslo, Norway.

terface and the gesture recognition, which communicates directly with Pd[11], and which enables programmers to define the objects and their control parameters, as well as the potential relations and interactions between different objects, by simply instantiating a series of Pd abstractions. MTCF can be freely downloaded at github[4].

2.1 Description of the system

MTCF has been designed for being used in conjunction with any type of tabletop surface that supports both the detection of marked tangible objects and multitouch interaction, although it does not force both interaction modes. The only restriction for the hardware is the output protocol used, its tracking system should complie with the TUIO protocol [9]. Otherwise it does not impose either any restriction on the size or shape of the surface, allowing to design for rectangular surfaces as well as for circular ones such as the Reactable.

Our internal test hardware is the one used for the reactable [7] and reacTIVision[8] as the tracking software (see Fig. 1). The generated data from reacTIVision (i.e. position and orientation of all the tagged pucks and fingers) is sent to MTCF using the TUIO protocol. MTCF just monitors all the incoming TUIO messages and sends them filtered to Pd by means of the Open Sound Control (OSC) protocol[13]. From Pd, control messages and waveform data are also transmitted back to MTCF, that is in charge of permanently refreshing the visual display.

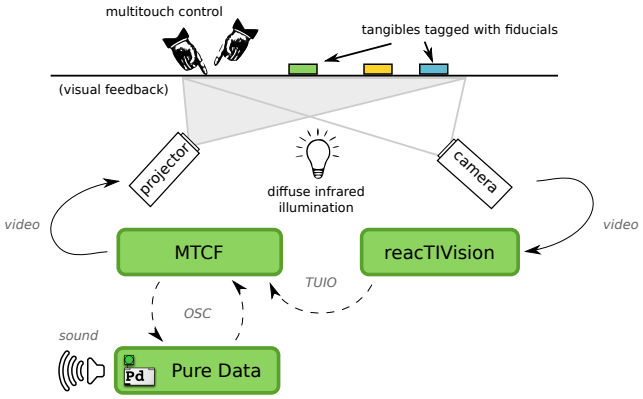

Figure 1: System diagram.

2.2 MTCF: Dealing with the Input data and with the GUI

MTCF is itself implemented on top of openFrameworks[5] (OF), a group of multi-platform libraries written in C++, specially designed for assisting creative applications programming.

MTCF also uses an external OF add-on, ofxTableGestures, which we had previously implemented with the aim of assisting multi-purpose (i.e. not necessarily musical) tabletop application design. ofxTableGestures does already solve some of the typical problems that appear in the development of generic tabletop applications, such as dealing with the tracking incoming messages or correcting the graphical output distortion or alignment. But ofxTableGestures is meant for OF programmers, which means that for using it, programming in C++ is still needed. In that sense, MTCF, built on its turn on top of ofxTableGestures, can be seen as a specialised and simplified subset of ofxTableGestures: while it does not permit all of ofxTableGestures' functionalities, it simplifies enormously the programming tasks by putting everything on the Pd side. Although no understanding of how ofxTableGestures works is needed for fully exploiting MTCF potential, next we will describe some of the basic ofxTableGestures features in order to give a clearer idea of the whole architecture.

ofxTableGestures is itself divided in two parts: TUIO input and graphics output. ofxTableGestures' TUIO input part processes the messages that arrive to the framework from any TUIO-compliant application (e.g. reacTIVision). Once these messages are processed, this component detects and generates gestural events for the top-level programmer. ofxTableGestures's graphics part on its side, helps to create drawable objects while applying the distortion correction to everything that is drawn. ofxTableGestures also includes a self-contained tabletop simulator, which simulates figures and multiple fingers interaction, allowing testing the applications without the need of a real table. (see Fig. 2). When the simulator is enabled, a right panel with a subset of figures is shown on one side of the screen. These figures are labelled with the identifier that will be reported by YUIO messages to the system. In order to maintain the fidelity between the physical table and the simulator, the figures used on the simulator match in size and shape with the real ones in our setup. ofxTableGestures includes six different figure shapes (circle, square, star, rounded square, pentagon and dodecahedron), which are defined in a configuration file that includes the figure shape, the figure identifier and the figure colour.

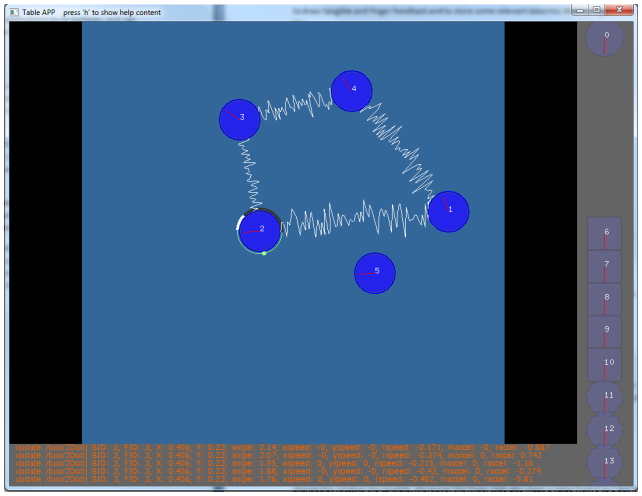

Figure 2: Simulator screen shoot.

MTCF receives data from the TUIO application, processes it, displays the graphic feedback and sends the filtered data to Pd via OSCMessages. At this stage, MTCF only draws the figure shapes and the fingers' visual feedback, all in their correct positions. The remaining graphics (such as the waveforms and the relations between the figures) are drawn in a second step, according to the additional information that is send back via OSC messages from Pd to MTCF. This will be addressed in the next section.

By default, MTCF pucks only convey three basic parameters: X position, Y position and rotary angle. Additional parameters can be enabled from Pd for any specific object. This optional additional information includes parameters resulting from the relations between pairs of pucks (distance and angle between them) as well as parameters resulting from the finger interaction onto given pucks, which

[4] https://github.com/chaosct/Musical-Tabletop-Coding-Framework/downloads
[5] http://www.openframeworks.cc/

can have two extra widgets (Object bar and finger slider) that can be activated from Pd, as shown in Fig. 3. These parameters are displayed as two semicircular lines surrounding the puck, keeping the orientation towards the centre of the table.

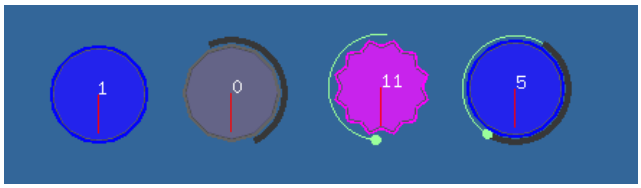

Figure 3: Tangibles with different feedbacks and controllers.

Objects' bars convey a value between 0 and 1 that can be changed by rotating the tangible. The finger slider, represented by a thinner line with a dot that can be moved using a finger, also ranges between 0 and 1. In the next section we will concentrate on the Pd side of MTCF.

2.3 Using MTCF from Pure Data

MTCF was designed to be used along with Pd, as this has become one of the most popular languages for realtime audio processing and programming. The main idea of this framework was to allow expert Pd users to interface their patches using a tangible tabletop setup. For this, MTCF provides nine Pd abstractions that transparently communicate with MTCF, and that are used to define the objects, the relations between them, and the data the programmer wants to capture from the tabletop interface. Not all of these abstractions have to be always used, as this will depend on the affordances of our musical application interface.

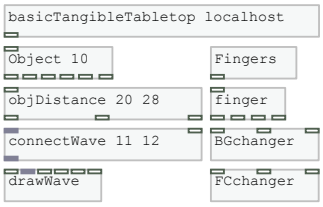

Figure 4: MTCF Pd Abstractions.

Only one abstraction is mandatory and responsible for all OSC communication between the Pd patch and MTCF: `[basicTangibleTabletop]`. Its single argument is the address of the computer running MTCF. This will typically be *localhost*, although changing this address can be useful in some situations, such as in testing several projects (on different laptops) wit only one tabletop (only running the visual part). One and only one instance of this object must exist in the Pd program.

2.3.1 Defining Objects and Parameters

Some additional abstractions will allow us to define what physical pucks will be used on the application. Instantiating `[Object` n`]` will tell the system to include the object with the id code n.

As described in the previous section, a slider plus a $[0, 1]$ rotatory parameter can be activated around any puck. The (de)activation of these extra controllers is done in Pd, by sending messages to their associated `[Object]`. Only when these elements are active Pd will receive this additional information.

Outlets in `[Object]` output the presence of the puck (Boolean), its position, orientation, and if activated, its slider and rotary parameter values.

Inspired by the Reactable paradigm, which allows the creation of audio processing chains by connecting different objects (such as generators and filters), MTCF also permits to use the relations between different pucks and can make them explicit. However, unlike the Reactable, MTCF is not limited to the creation of modular, subtractive synthesis processing chains; any object can relate to any other object independently of their nature. This allows for example to easily create and fully control a tangible Frequency Modulation synthesiser, by assigning each carrier or each modulator oscillator to a different physical object; or a Karplus-Strong plucked string synthesiserby controlling the extremes of a virtual string with two separate physical objects.

On the counterpart, MTCF does not yet permit dynamic patching [10], so it is not capable of producing a fully functional Reactable clone, neither was this its main objective. In MTCF, the connections between the pucks have to be made explicitly by the programmer in the Pd programming phase. This is attained by using `[objDistance` m n`]`, which continuously updates about the status of this connection, and (if existent) about the angle and distance between objects m and n. The programmer can also specify whether she wants this distance parameter to be drawn on the table by sending a Boolean value into the `[objDistance]` inlet.

2.3.2 Drawing Waves

Also inspired by the Reactable, MTCF can easily show the "sound waves" going from one object to another. This can be achieved by using the `[connectWave]` object. This abstraction takes two parameters that indicate the two object numbers between which the wave should be drawn. As indicated before, this waveform does not necessarily indicate the sound coming from one object into the other, but can rather represent the sound resulting from the interaction between two combined objects, or from any other sound thread from the Pd patch.

An audio inlet and an outlet are used to take the waveform and to act as a gate, allowing the audio to pass, only if the two pucks (and therefore the waveform) are on the surface. This ensures that no unintended sound will be processed neither shown when its control objects are removed. Additionally, a control inlet lets the patch to activate and deactivate this connection.

This way of drawing waveforms has some consequences: first, waveforms are drawn by default between pucks, difficulting the drawing of waveforms between two arbitrary points, or from one object to the centre, as Reactable does. This can be overcome by using a simpler Pd abstraction, `[drawWave]`, which has exactly this very purpose: drawing waves between two points.

The second but very important consequence is that the audio connection between two physical pucks is a Pd object itself. Instead of having Pd audio connections between `[Object]` abstractions, the programmer must therefore use `[connectWave]` abstractions, which simply send the waveform information to MTCF for drawing it. This can be confusing, specially when chaining multiple physical pucks imitating an audio processing chain, since the programmer must then consider all the possible combinations (Fig. 5).

2.3.3 Extra features

For more advanced interaction, additional abstractions are also provided. `[Fingers]` gives full information of the position of all fingers detected on the table, while `[finger]` can be used to extract individual fingers information (see Fig. 6). These abstractions can be used to control less obvious parameters.

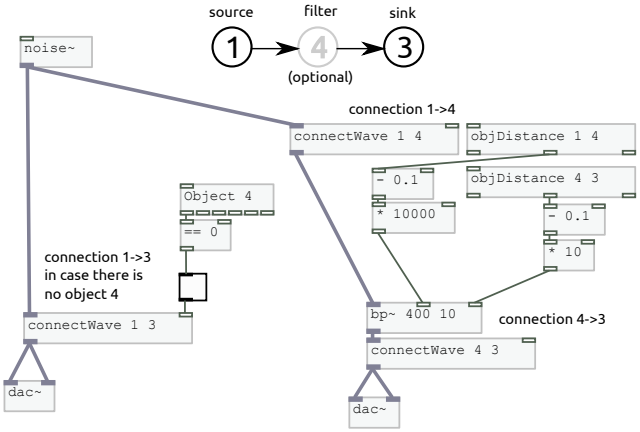

Figure 5: A processing chain example. Puck 1 is a noise generator, puck 4 is a filter, and puck 3 is an audio sink (i.e. the speakers). The programmer must consider the connections both when puck 4 is present ($1 \rightarrow 4 \rightarrow 3$) and when it is not ($1 \rightarrow 3$).

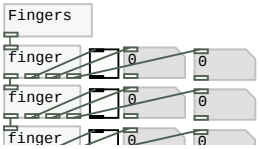

Figure 6: A Pd structure to receive information of the several fingers on the surface.

Two additional abstractions can be used for visual purposes: `[BGchanger]` and `[FCchanger]` respectively allow changing the background colour of the tabletop and the colour of the fingers' trailing shadows. Changing colours, for example according to audio features, can create very compelling effects.

3. CONCLUSIONS

The experience we have gained until now from using MTCF on two short half-day workshops, indicate that MTCF is not only a very valuable tool for the quick development and prototyping of musical tabletop applications, but also an interesting system for empowering discussion and brainstorming over some concepts of software synthesis control and interaction.

We are also aware that there are many issues that can still be improved. While Pd experts quickly understand the framework's mechanisms and take full profit from it producing interesting results in very short times, a few advanced users missed some higher level control possibilities. At its current stage, MTCF is clearly very oriented towards real-time sound synthesis and processing control, lacking of higher level and more structural controls that could communicate with Pd entities such as data arrays or sequences. In a near future, we therefore plan to include more graphical interface features, probably making a more extensive use of multi-touch interaction, in order to be able to control time-oriented and structured data such as envelopes or sequences of events.

4. ACKNOWLEDGMENTS

This work has been partially supported by TEC2010-11599-E (Ministerio de Ciencia e Innovación, Gobierno de España) and by Microsoft Research Cambridge.

5. REFERENCES

[1] R. Bencina, M. Kaltenbrunner, and S. Jordà. Improved topological fiducial tracking in the reactivision system. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, page 99. Ieee, 2005.

[2] Y. Fernaeus, J. Tholander, and M. Jonsson. Beyond representations: towards an action-centric perspective on tangible interaction. *International Journal of Arts and Technology*, 1(3):249–267, 2008.

[3] L. Fyfe, S. Lynch, C. Hull, and S. Carpendale. Surfacemusic: Mapping virtual touch-based instruments to physical models. In *Proceedings of the 2010 conference on New interfaces for musical expression*, pages 360–363. Sydney, Australia, June 2010.

[4] J. Hochenbaum, O. Vallis, D. Diakopulos, J. Murphy, and A. Kapuy. Designing expressive musical interfaces for tabletop surfaces. In *Proceedings of the 2010 conference on New interfaces for musical expression*, pages 315–318. Sydney, Australia, June 2010.

[5] S. Jordà. Sonigraphical instruments: from fmol to the reactable. In *Proceedings of the 2003 conference on New interfaces for musical expression*, NIME '03, pages 70–76, Singapore, Singapore, 2003. National University of Singapore.

[6] S. Jordà. On stage: the reactable and other musical tangibles go real. *International Journal of Arts and Technology*, 1:268–287, 2008.

[7] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st international Conference on Tangible and Embedded interaction*, pages 139–146. ACM, 2007.

[8] M. Kaltenbrunner and R. Bencina. reacTIVision: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 69–74. ACM, 2007.

[9] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza. Tuio-a protocol for table based tangible user interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005), Vannes, France*, 2005.

[10] M. Kaltenbrunner, G. Geiger, and S. Jordà. Dynamic patches for live musical performance. In *Proceedings of the 2004 conference on New interfaces for musical expression*, NIME '04, pages 19–22, Singapore, Singapore, 2004. National University of Singapore.

[11] M. Puckette. Pure Data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

[12] J. Schöning, P. Brandl, F. Daiber, F. Echtler, O. Hilliges, J. Hook, M. Löchtefeld, N. Motamedi, L. Muller, P. Olivier, et al. Multi-touch surfaces: A technical guide. *Technical Reports of the Technical University of Munich*, 2008.

[13] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference*, pages 101–104, 1997.