ClaPD: A testbed for control of multiple sound sources in interactive and participatory contexts

Cumhur Erkut
Lab. Acoustics and Audio Signal Processing
Helsinki University of Technology
Espoo, Finland
Cumhur.Erkut@tkk.fi

Koray Tahiroğlu
Media Lab.
University of Art and Design (UIAH)
Helsinki, Finland
koray.tahiroglu@uiah.fi

ABSTRACT

In this paper, we briefly outline our view on the control of multiple sound sources in interactive and participatory contexts, as will be required by the next-generation applications. As a testbed towards this view, we continuously extend the functionalities of our basic library ClaPD that meets the minimum requirements of the final system, albeit in a restricted application domain. ClaPD was written for an entirely different purpose, but it helps us to understand how we can tackle the bigger challenge. We show how we solve the problem of a better visualization strategy, and the dynamic creation of the control objects (clappers). The solutions were, in greater extent, inspired by the implementations within the pd-community.

Keywords

control, multiple sound sources, clapping, audience, ClaPD

1.INTRODUCTION

Originally released as a real-time electro-acoustic music environment [1], the application domain of PureData (pd) is being continuously expanded. A meaningful introduction to pd currently consists of sound, image, networking, and physical interaction modules (http://puredata.info/dev/pddp). Moreover, pd is considered as an audio engine in many networked, distributed, interactive, and participatory audio-visual content creation and prototyping applications, including 3D graphical model creation and rendering [2] and computer games [3].

An important characteristic of these applications is that their components communicate via control, rather than audio streams. Audio streams require a higher bandwidth in networked applications, and are more prone to packet-losses and transmission errors. This distinction between the control and the audio streams enables, for instance, the long-awaited network-based collaboration (http://www.netpd.org). Other possibilities we may soon witness in the pd community include joint audio-video synthesis of large-scale environments [4] and evolutionary music applications [5]. These applications require dynamic handling of multiple instances of similar sound sources, and are characterized by fundamentally different use cases from the current ones: N performers on M machines will need to interact with P patches that contain Q instances of S sound sources, which are controlled by C control streams and eventually converted to A audio-visual streams.

In order to manage this combinatorial complexity and to gain maximum benefit of the upcoming applications, recent studies focus on advanced control strategies and propose a dynamic mapping layer between control and audio-visual streams [6]. The advantages of separating the audio and the control streams are well-known in physics-based sound synthesis [3,7,8]. We believe that this approach can be extended by an additional layer that generates events in an hierarchical fashion. Such a three-layer system would manage multiple sound sources in interactive and participatory contexts, and would fully address the needs of the next-generation applications. It could also adapt to different application areas and scenarios. This is an ambitious project[1], but we will thoroughly and systematically explore the possibilities in the long term.

Meanwhile, besides this top-down approach, we think it is also important to consider a bottom-up approach and continuously extend the functionalities of a basic library that meets the minimum requirements of the final system, albeit in a restricted application domain. ClaPD[2] is such a library; written for an entirely different purpose, but it helps us to understand how we can tackle the bigger challenge.

ClaPD is a pd library for hand clapping synthesis and control [9]. The initial release of the ClaPD (rel-0.1) could produce individual hand-claps, or the asynchronous/synchronized applause of a group of clappers. In terms of our discussion, the hierarchical event layer corresponds to the asynchronous/synchronized applause decision, the control layer determines the tempi and the hand configurations of individual clappers, and the synthesis layer is the actual audio calculation. Therefore, the potential of the ClaPD as a testbed for control of multiple sound sources in interactive and participatory contexts has been addressed in [10], where a prioritized task list for extension has been also deduced. This list includes a better visualization strategy, extended user control, generalization of the synthesis strategy, and the dynamic creation, connection, and disconnection of the clappers. Since then, we are implementing

[1] http://www.acoustics.hut.fi/~cerkut/temp/schema-sid/

[2] ClaPD is released as a free software under the GNU Public License (GPL) and can be downloaded from http://www.acoustics.hut.fi/software/clapd

the required functionalities and use the ClaPD as a testbed for control of multiple sound sources in interactive and participatory contexts. During this activity, we typically realize that some of the required functionalities are readily available in the pd-community. In the sequel, we refer to these externals and abstractions by their relative path in the pd-CVS repository.

This paper is organized as follows. In Sec. 2, we present an overview of the ClaPD, and in Sec. 3, we recapitulate the existing problems and our current solutions. Next, we concentrate how to use these ideas in real-time performance situations. Finally, in Sec. 5, we derive our conclusions and indicate the future work.

2.OVERVIEW OF CLAPD

The ClaPD is a stochastic model in realm of [8]. It currently contains low-level synthesis and higher-level control blocks, and primitive agents for event generation, which are fine-tuned by hand-clapping statistics. ClaPD can produce expressive, human-like synthetic applause of a single clapper with adjustable hand configuration, or asynchronous or synchronous applause of a clapper population. The single clapper mode of ClaPD is discussed in detail in [5], the main focus here is the second mode of ClaPD, which is illustrated in the screenshot of the patch Ensemble-GUI in Fig. 1, where the clapper abstraction is expanded at the lower-right side of the figure. The clapper population is called audience, which consists of individual clappers. The clappers do not interact, but listen to a master clapper and adjust their clapping rates accordingly. Although technically possible, the hand-configuration control of the individual clappers of the audience is disabled due to the resulting control complexity.

Structure of ClaPD

ClaPD consists of a set of externals, abstractions, and main patches. The low-level synthesis block inClaPD follows the excitation/resonator paradigm [9]. The externals **noiseq~** and **twopz~** generate and shape the excitation signal, respectively, and **reson~** is the resonator. The parameters of these synthesis blocks are drawn randomly from a triangular distribution that is implemented as an abstraction (**trirand**). This allows a small variation between each clap. The parameters of the triangular distribution correspond to the hand configuration prototypes, and they are extracted from the analysis of recorded hand claps in an anechoic chamber [9]. The abstraction **oneclap** accommodates all the other synthesis objects and provides the main synthesis block in the ClaPD; there is only a single synthesis block regardless the number of clappers.

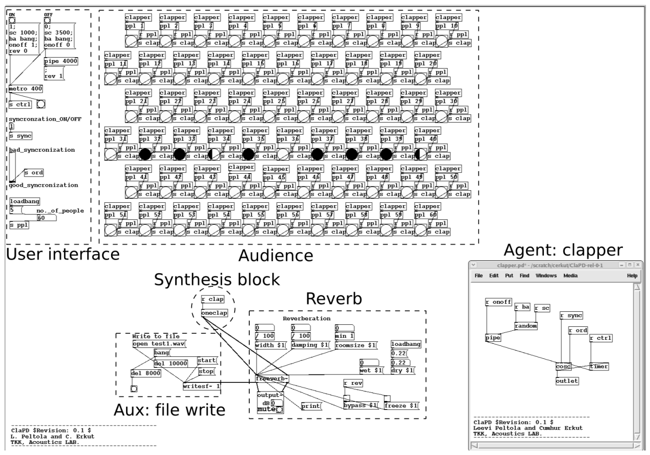

Fig. 1: The audience patch of ClaPD (rel-0.1). The bangs connected to the clappers flash when they clap. At this instant, six clappers are fully synchronized, and two are catching up.

The abstraction **clapper** is the fundamental control entity in ClaPD; its state is the delta-time until its next clap in milliseconds, and its behavior is to update its own state. The clapping mode of the audience (asynchronous or synchronous) is user-controlled by a switch and perceived by the clapper.

In the synchronous mode, the clappers listen to the master clapper, which is a native pd metro object that ticks with a fixed period around 440 ms. This imitates the period doubling of human clappers during synchronization [9]. When the master claps, a clapper calculates its offset relative to the master and determines its behavior with the help of the external object **cosc**. This control block is the most complex object in ClaPD; depending on the entrainment parameter [9] of the audience and relative offset between the master and a clapper, it enforces the clapper to accelerate/decelerate. Moreover, when the mode is switched back to asynchronous, **cosc** decouples the clapper from the master and accelerates its state until a natural clapping rate - drawn from Tri(220,70) - is achieved. The detailed dynamics of **cosc** operation can be found in [9].

The remaining objects within the library are three main patches, the external **ppl** that controls the number of clappers within the audience and the abstraction **oneclapper** that generates the control stream in single clapper mode, which is not discussed here. One of the three main patches loads the required objects and provides a run-time interface to the user. The patch **SoloClapper** is a single clapper model, whereas the other two are the models of asynchronous/synchronous applause of an audience. Both ensemble models synthesize audio streams, both only one of them (**Ensemble-GUI**) provides primitive visualization by flashing widgets (see Fig. 1).

Use-cases of ClaPD

In the screenshot of the patch Ensemble-GUI in Fig. 1, the clapper abstraction is expanded at the lower-right side of the figure. All the patches optionally include artificial reverberation to represent a listening environment via the pd external library **freeverb~** by Olaf Matthes. Another purpose of the reverberation unit is to mask the transients due to the parameter updates of the synthesis object **oneclap.** The parameters of **freeverb~** can be controlled by the user, and its operation can be by-passed with a switch. The output of the reverberation can optionally be written to a file via an auxiliary block. The audio stream fed into **freeverb~** comes from the abstraction **oneclap**; the main synthesis block of the ClaPD. The user can set the hand configuration parameters for the entire audience by expanding this abstraction. The synthesis block receives the clapping events from individual clappers of the audience.

The clapping events are probed by a GUI widget connected to each clapper, it flashes when the clapper sends a clap event. The maximum number of clappers are fixed at 60, but the user is allowed to set it to a lower value using the interface block. Note that the simple visualization in ClaPD is not the most efficient one: with 60 clappers it uses the half of the available CPU power on a Pentium 4 machine, whereas the audio calculations use only about %3. Without visualization, the patch Ensemble-NOGUI can accommodate a much larger audience.

By using the interface, the user can start and stop the applause, determine the asynchronous/synchronous mode, set the level of entrainment, and the number of people at the run-time. Note that all these operations act on the the audience, the user has no means to access to clapper states or to influence its behavior.

3.A TESTBED: PROBLEMS AND SOLUTIONS

Improved visualization (blinkenlights)

With a modified version of the clapper abstraction **(myclapper),** we now assign each clapper a pixel in the 2D grid of the **unauthorized/blinkenlights** external by Y. Degoyon, and flash this pixel for 100 ms (adjustable) during the corresponding clap. This solution is not only CPU-friendly (about %6 for 60 clappers), it also provides a better visual description of the emerging patterns of synchronization within the audience. Moreover, the visualization can be turned on and off: the UI sends a visualization flag that is received by each clapper. Therefore currently there is no need for separate main patches with and without the visualization.

Dynamic creation of clappers (constructor)

We have originally constructed the audience manually by creating 60 static copies of the **clapper** object, including their unique **ppl** number. Below 60 clappers, the number of people in the UI disables all the clappers whose ppl number are higher, but the unneeded clappers still persist on the patch. For more clappers, we needed to create more clappers by copy and paste, edit their **ppl** number ID, and update the number of people in the UI. Our wish was to control the number of people just by using the UI, and to create the clappers dynamically, and just as needed. We have found several ways to do that: **pd-msg**, **construtor**, **nqpoly4**, **dyn~**, and the dyn system [11]. Initially, the constructor abstraction has been the most useful tool for our purposes, but later we have facilitated the native pd internal messaging system for more efficient and direct dynamic creation of the clappers.

At the lowest level, the PD system has built-in scripting facilities [11]. As it is documented in the PD extended version under the manuals, it is possible to create and control pd objects, modify them, load a patch or create a new one through messages that can be sent from another patch. Dynamic control for the internal events requires no GUI or manual intervention in edit mode.

Based on this lowest level native mechanism, we have experimented with the **constructor** abstraction, which is developed by Ben Bogart and Cyrille Henry, and can create a 3D-matrix of abstractions (specified by the creation arguments) and connections among them (given as a list to the second inlet). We have cloned the modified clapper abstraction as a 2D-matrix with the same amount of rows and columns as the blinkenlight pixel grid and created a wrapper to pass the number of people to the creation arguments of the constructor and blinkenlight. We have not yet assigned spatial presence to the clappers, nor experimented with different interconnection strategies, although

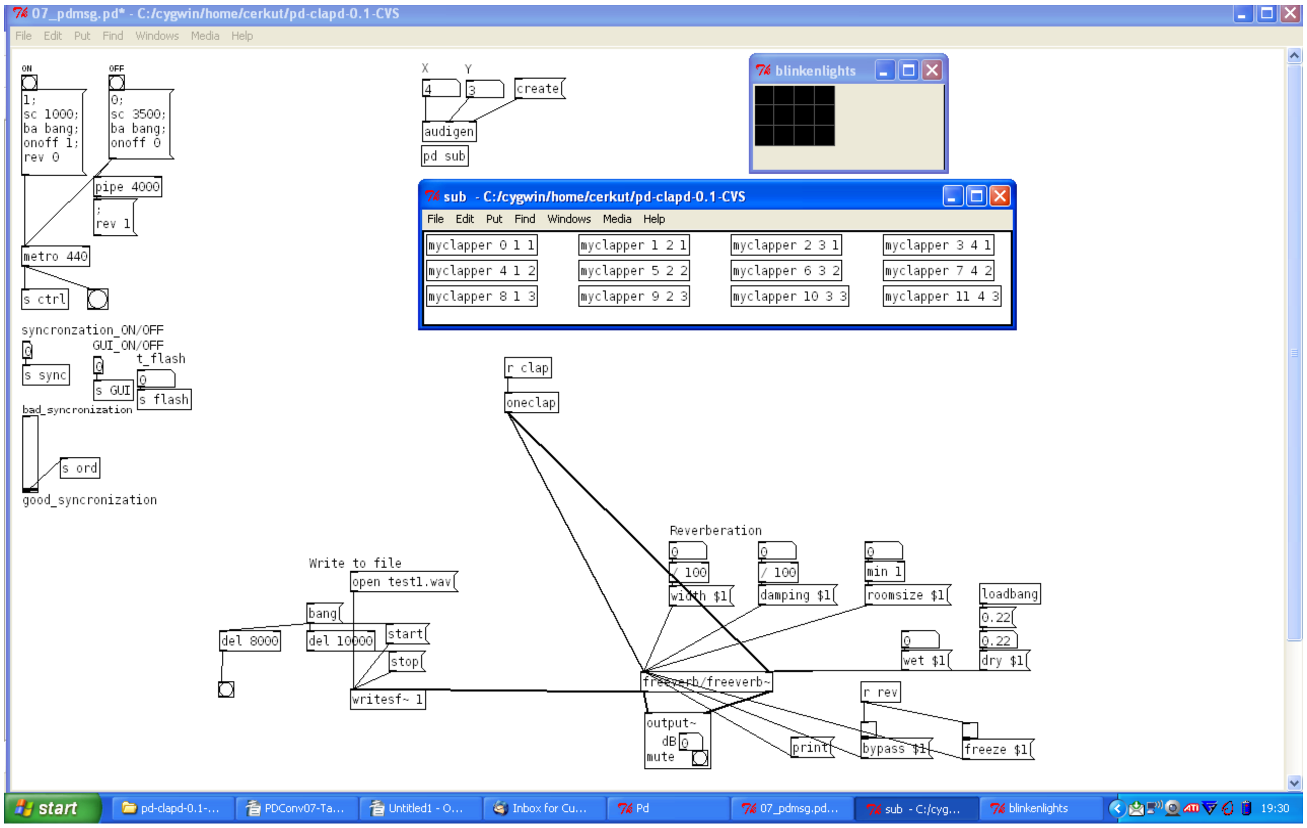

Fig. 2: Dynamically-generated audience by using the internal messaging system of pd. The clappers in the subpatch are created by audiogen, and they concurrently define the blinkenlights grid.

constructor would provide the required functionalities for these experiments.

nqpoly4 external by Steven Pickles (pix), which can take abstractions and link them together to make a polyphonic instrument, proved to be most useful for resource allocation when dynamically creating the audio objects (see below).

Other possibility for internal events management in pd is the **dyn~** external by Thomas Grill. Even though **dyn~** external looks similar with pd-msg description, it provides the advantages of a better object and interconnection addressing. Finally, the dyn API [11] seems to be the most comprehensive system we have encountered for dynamic object management.

In the last release of the ClaPD, we create the final dynamic patch by using the internal messaging system of pd. Through a main patch, we distribute the clappers as a matrix, concurrently define the row and column values of the blinkenlights grid, and send them as a creation argument to the **audigen** (audience generator) abstraction object. Each clapper then receives a unique ID, together with a row and a column index to specify its appearance both in a subpatch and in blinkenlights grid; **myclapper** objects receive these values as creation arguments. An example of the resulting subpatch is illustrated in Fig. 2.

Generalization of the synthesis strategy

As in the previous release of the ClaPD, each clapper sends a bang to the main synthesis block (**oneclap,** shown within a dashed circle in Fig. 1) according to its own timing mechanism. Since the audio and event generators are decoupled, the clapping synthesis block can be replaced with any other sound generation block to generate other collective scenes (e.g,, a crowd of people yelling, pack of dogs barking, or even some kind of harmonic musical material). The only requirement is to construct an abstraction that can be triggered with a bang. Optionally, a matched control block with a more advanced timing mechanism (e.g., locking to *p*/*q**master, where *p and q* are integers) can be also constructed.

A related question is how to mix multiple synthesis models (e.g., mix a crowd of clapping with some people whistling). After the dynamic creation of the subpatch, it is possible to manually edit any clapper, and reroute its firing to another synthesis block. The steps involved in this operations are as follows:

1. Create another synthesis block; connect its input to a **receive** (for instance, [r whistle]) and its output to the **freeverb~**.
2. Open any of the **myclapper** abstractions in the dynamically-generated subpatch, update the **send** object connected to **cosc** (for instance, [s whistle]).
3. Optionally, insert a *p*/*q* ratio to alter the timer operation.
4. Optionally, change the [pixon $1 $2(message to [pixel $1 $2 127 0 0(for a red appearance of the corresponding object on the blinkenlights grid.
5. Save this abstraction with a different name (for instance, **mywhistler**).
6. Edit the name of the desired objects in the subpatch.

The created abstractions can then be used as a library; once such a library is available, only steps 1 and 6 would be required. Currently, we are extending the library of different sound generators. Automatization of this procedure is considered as a future work.

An example of mixing multiple synthesis models is illustrated in Fig. 3, where the dynamically-generated subpatch of 12 clappers have been changed by the operations outlined above to 10 clappers and 2 whistlers.

Resource allocation (nqpoly4)

The separation of the audio and control streams has an obvious advantage when dealing with a high number of objects. To illustrate this advantage, we have conducted a simple experiment. Based on the example of the pd-msg documentation, we have created *n* instances an oscillator and a single clapper (control and audio), and measured the CPU load on a P4 machine both using the PD tool and from the command line. We have found that the CPU load percentage was 0.2*n* in the first case, and 0.36*n* in the second. This means, we can create at most 500 instances of an oscillator, and 277 instances of the clapper object on that machine, if both audio and control streams are used. On the other hand, in the extreme case of a single audio object in ClaPD **(oneclap),** we create thousands of instances of the control-only clappers, but loose the flexibility of parametric control, and observe some transient effects when multiple clappers excite this single audio object. An optimum number of the audio objects is between these two extremes, and the second outlet of the **nqploy4** that outputs a bang when the audio object has finished playing and is ready to receive another trigger is a great tool for further experiments with generalization and resource allocation. Note that, in the case of multiple sound generators (e.g., clap and whistle), **nqploy4** should be utilized per generator.

4.CLAPD IN PERFORMANCE

ClaPD is a platform for controlling multiple clap sound objects in interactive contexts. The control structure in ClaPD currently relies on the one-to-one relation between each clapper and the master clapper; however, in the future it will rely on the event generation of each clapper, which will be controlled and manipulated by the events of the all other clappers. Each clapper will be thus an agent that listens and interacts with all the others [10], and the behavior of these agents will be controlled by the events of the other agents [5]. With additional synthesis objects, ClaPD will be then a sound performance system, which can be used in various musical art events. The authors will implement more musical events to be explored by the agents during a performance in the future.

Performing for live electronics in real-time constitutes a need for an overall control on the performance system. The multi-agent control systems give possibility to have an accompaniment system during the live performance. The dynamic event generation modules enrich the sound synthesis of the musical performance compared to the predefined structures. Generative multi-agent modules are driven by improvisational acts of a human performer; however, their autonomous behavior is a benefit in terms of generative aspect of music events.

The agent-based control system in ClaPD can be used for two different control strategies. Performers can control each agent (clappers) individually or control function can be on master clap agent, which has an overall control on the events of the generated

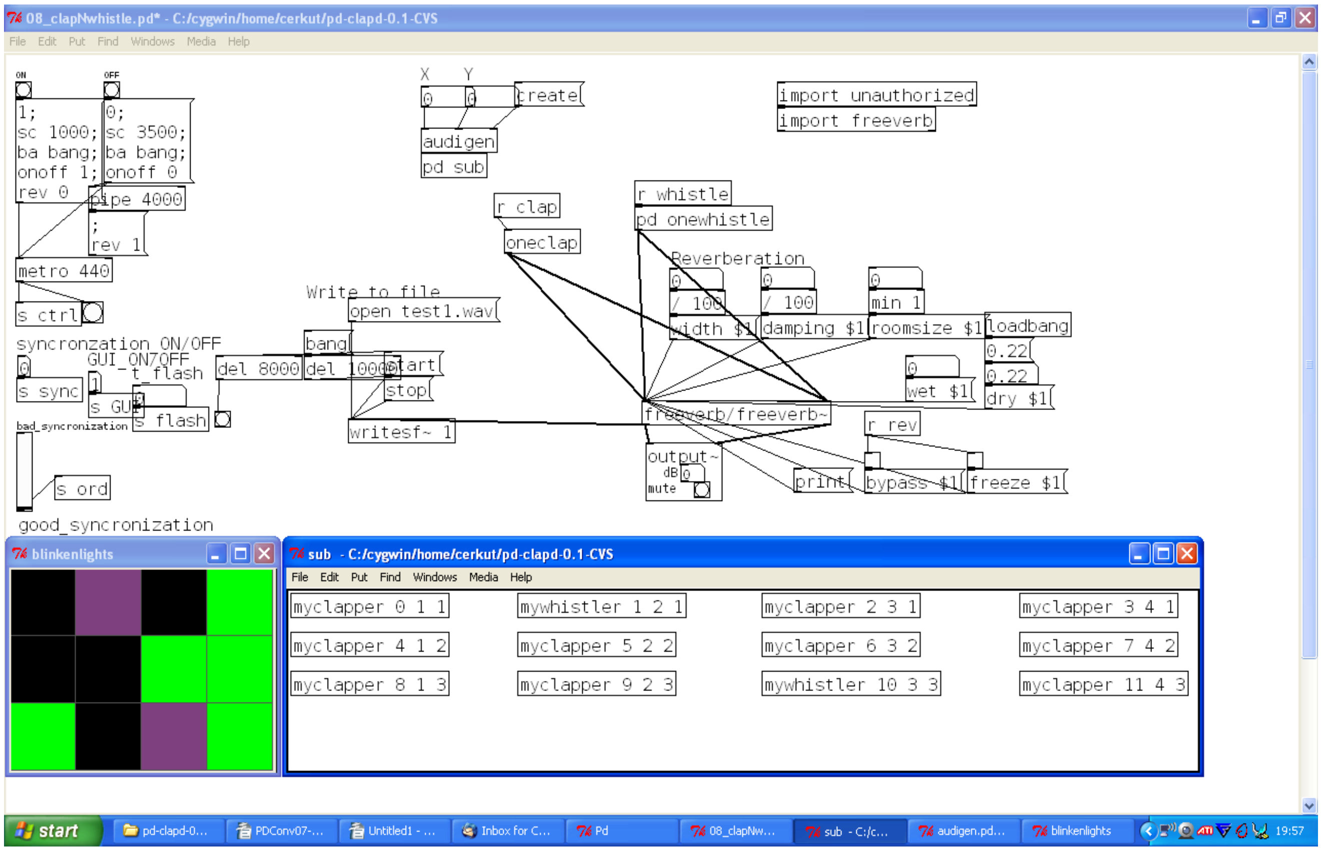

Fig. 3: An example of mixing multiple synthesis models: 10 clappers and 2 whistlers

clappers. The current ClaPD architecture can be extended into a real-time music synthesis performance tool by mapping sound sources as agents instead on the clapper abstractions. Controlling single agent behavior or master controller for multi-agent event generation can be turned into organizing sonic relationships over time in a real-time performance.

5.CONCLUSIONS AND FUTURE WORK

We have briefly outlined our view on the control of multiple sound sources in interactive and participatory contexts, as will be required by the next-generation applications. The realization of these ideas will be based on the distinction between the events, control, and audio streams. While working in the long term to implement these functionalities in the top-down fashion, we are continuously extending the functionalities of our hand-clapping synthesizer ClaPD in the bottom-up direction. In this paper, we have tackled the problem of a better visualization strategy and the dynamic creation of the control objects (clappers). The current solutions were inspired by the existing implementations provided by the pd-community.

In the future, we will fully utilize the ideas, found for example in **nqpoly4,** for the generalization of the synthesis strategy and resource allocation. Meanwhile, reducing the dependencies to the other externals will be a guiding principle for us.

We have kept so far the one-to-one relationship of a clapper and the master clapper; in the future we will experiment with different interconnection strategies and assign spatial presence to the clappers. We will then switch to the extended user control, and be best tool we can think of for this task is the **OSCx.** Finally, a functional performance system that demonstrates our view is yet another important future challenge.

6.ACKNOWLEDGEMENTS

This work is supported by the Academy of Finland (projects 105651 and 120583 Schema-SID). Our thanks to Ben Bogart for introducing us the constructor patch, and to Leevi Peltola for developing the initial version of the ClaPD in a very short time and GPL'ing it for extension and maintenance.

7.REFERENCES

[1] M. S. Puckette. The Theory and Technique of Electronic Music. World Scientific Press. May, 2007.

[2] R. Kajastila, S. Siltanen, P. Lundén, T. Lokki and L. Savioja. A distributed real-time virtual acoustic rendering system for dynamic geometries. In *Proc. AES 122nd Convention,* Vienna, Austria, May 2007.

[3] A. J. Farnell. Sound synthesis for games. In *Sounding Out Symp*., Sunderland, UK, July 2006. Available online at http://obiwannabe.co.uk

[4] N. Raghuvanshi and M. C. Lin. Interactive sound synthesis for large scale environments. In *SI3D '06: Proc. Symp. on*

Interactive 3D graphics and games, pages 101–108, New York, NY, USA, 2006. ACM Press.

[5] E.R. Miranda and J.A. Biles. Evolutionary Computer Music. Springer, 2007. See also http://www.livealgorithms.org

[6] A. Momeni and C. Henry. Dynamic independent mapping layers for concurrent control of audio and video synthesis. *Computer Music J.*, 30(1):49–66, 2006.

[7] C. Henry. PMPD: Physical modelling for Pure Data. In *Proc. Int. Computer Music Conf.*, Coral Gables, Florida, November 2004.

[8] D. Rocchesso and F. Fontana, editors. The Sounding Object. Edizioni di Mondo Estremo, Firenze, Italy, 2003.

[9] L. Peltola, C. Erkut, P. R. Cook, and V. Välimäki. Synthesis of hand clapping sounds. *IEEE Trans. Audio, Speech and Language Proc.*, 15(3):1021–1029, 2007.

[10] C. Erkut. Towards physics-based control and sound synthesis of multi-agent systems: Application to synthetic hand clapping. In *Proc. Nordic Music Technology Conf.*, Trondheim, Norway, October 2006.

[11] T. Grill. dyn: Dynamic object management. In *Proc. 1st Intl. pd Conv*., Graz, Austria, Sept. 2004.