Reality Check - a framework for preserving real-time electronic music realizations

Miller Puckette
UCSD/IRCAM
Preprint - under review

ABSTRACT

A framework is proposed for protecting an ongoing music production using a continuous integration (CI) para-digm. An enabling technology named Reality Check *is described, which has been applied so far to two existing and one in-progress electronic music production. The benefits are at least two-fold: the pieces that are included in the CI system can be monitored for their continued viability; and also, the various software components used in their realization can use the pieces as unit tests to ensure their own continued back-compatiblity.*

1. INTRODUCTION

Live electronic music has a durability problem. Any realization more than a few years old is likely to require maintenance to adjust for changes in hardware or software. Pieces of live electronic music often fall into disuse for lack of the resources necessary to keep them current. Together with fixed-medium music, pieces of live electronic music can become unavailable because of deterioration of physical media, unavailability of equipment that can read the media, and/or difficulty of deciphering proprietary file formats and disk filesystems. In addition to these problems, live electronic music faces unique challenges.

Even if we suppose that we can access the primary digital materials used in a live realization, it can easily happen that essential software components used in a live electronic realization simply disappear or become unusable because of an operating system update or a lost copy protection key. Moreover, a dependency of this sort might not be evident at the time a piece is first realized - it might be hidden in a system library, plug-in, or configuration file stored somewhere else than the digital archive that supposedly constitutes the realization.

The purpose of this paper is twofold: first, to introduce a new software framework called Reality Check that aims to help keep both an electronic realization and its supporting software available over time; and second, to argue for a cultural change among practitioners of live electronic music that can help slow the processes by which the live music repertory degrades over time.

A further affordance of this work lies in its contribution to DAFNE+, a project supported by the European Commission to apply blockchain technology toward the development and versioning of collaborative artistic projects. One major problem facing this project is that items on (or pointed to from) a blockchain are fixed digital documents that could easily fall into the same time traps as any other such document. In particular, if a live electronic realization is built up over a long collaborative process among several stakeholders, the problem of ensuring the continued usefulness of the data itself is multiplied. For example, an NFT describing a realization of a piece of electronic music is worthless if the realization doesn't work. If blockchain technologies are ever to become relevant to the real art world their contents will have to offer more than GIF animations.

It will certainly not be possible for any one effort, such as this one, to permanently solve the problem of realization rot. In many situations it will simply not be possible to use the tools or ideas described here. But even to somewhat mitigate the problems we now face will be well worth the effort.

In the sections that follow we first give a brief description of recent related work; then give a functional description of Reality Check and its use both in existing and one ongoing musical production. Finally we discuss the cultural and practical issues that arise in adopting Reality Check (or another such framework) in the development and production of live electronic music realizations.

2. A STATE OF AFFAIRS

The rich literature that bemoans the constant falling into obsolescence of pieces in the live electronic music repertory is too extensive to even outline here[1]. Both hardware and software slide perpetually into the oubliette as new, more exciting hardware and software appear. New techniques and tools energize composers as they pursue new and unexplored musical possibilities. But pieces of "new" electronic music, once no longer so fresh, are forgotten even as they are replaced by ever newer ones. There is clearly a problem here.

A bespoke circle of hell opens up for those who turn to specialized hardware to overcome the limitations of standardized hardware. Until about 1990, for example, general-purpose processors could not approach the speeds necessary to generate anything beyond the most simple electronic sounds in real time. Composers, both in institutions and as individuals, turned to off-the-shelf synthesizers and sound processors. At IRCAM for example, there was even the idea that composers who wrote for "personal systems" such as computer-controlled MIDI synthesizers would find

their work much more easily portable outside the walls of the lab and, perhaps, longer-lasting as a result. The reality has been the opposite. Pieces from the late 1980s by Michael Jarrell and André Dalbavie[2] were not only never performed without IRCAM's direct involvement but quickly became obsolete, even despite their occasional revivals after much hard work on the part of later generations of computer music producers.

One might think that we are now free of such problems, since the audio signal processing required in real-time electronic music performance is normally well within the capacity of a modern CPU and can thus be realized portably. But no such luck: the new vogue for applying machine learning in real time means that some of the work is often offloaded onto "graphics processors" using specialized packages and programming languages that will certainly go the same way as the Yamaha TX816s of the 1980s. New papers along the lines of Akkerman's will continue to appear.

The last couple of decades has seen the appearance of a few constructive responses to the overall problem, such as GAMELAN[3], ASTREE[4], and Integra[5]. None of these appear to be still available online. It appears that they have fallen into one or both of the following two traps. First, there has been a tendency to provide supporting software for the development of the piece itself (as opposed to its preservation). Integra, for example, provided an entire infrastructure including sequencing and specific types of audio effects. There is a tacit assumption that these tools would expand over time to meet the needs of an ever wider range of musical tasks, but instead, the existing structure was not one into which a great number of composers wanted to fit their work, and so there were not very many pieces produced within it.

This leads to the second problem, which is that of maintenance of the infrastructure itself. To muster the will to maintain a system such as Integra requires a critical mass of composers or other workers who are invested in the project. Such a mass seems not to have gathered, perhaps as a result of the first issue cited above.

As an alternative to building a framework for realizing new works, we propose that live electronic music realizations, both new and old, be placed into a CI framework in order to toughen them against the ravages of time. This has the advantage of not imposing a particular structure upon any given realization, other than its ability to be virtualized in order to run automated tests on it. In principle at least, not only can the structure of a piece (whether through-written, improvised, generative, or other) be freely determined by its creator, but also the software used can be chosen by the artist.

There nonetheless remains an important obstacle: any tools used in the live performance must be regenerable in some way, either by virtualizing a binary image or, much preferably, by compiling from open source. A complete realization of a piece should contain not only all documents specific to it but also the wherewithal to regenerate all the software that needs to run in real time. This includes not only the environment (Max, Pure Data, Supercollider, Csound, or whatever) but also any dependencies such as plug-ins or "externs".

This is not a trivial demand. A composer under the time pressure imposed by a looming concert date will understandably want to grab a favorite old tube compressor emulator that might be ready to hand. Such plug-ins are often encrypted and copy-protected, and there is little hope that any given one will be runnable once the mom-and-pop software house it came from inevitably withers away. Not just for CI-readiness but also in general, a composer or realizer must always choose between expediency and long-term viability.

In the following section we introduce Reality Check, one such possible CI system. It is implemented in Pure Data (Pd) and has been available in preliminary form on the web since 2020. The occasion for introducing it to a wider audience today is a new, large-scale music production that is using Reality Check as its CI system, in the hopes of guaranteeing its long-term viability.

3. REALITY CHECK

Reality Check is implemented as a Pd patch that can test other Pd patches (presumed to be realizations of musical pieces) to verify that they generate the correct output given a standardized input. The standardized input consists of a soundfile, and a text file containing all MIDI, network packet, and GUI (mouse and keyboard) events. To prepare the test, we generate a standardized output soundfile by running the piece with the standardized input.

At any later time we may then present the standardized input to the realization patch to verify that it still produces the same output, to within a predetermined tolerance (by default, one ten-millionth of maximum amplitude). If the test passes, this verifies not only that the realization patch is correct, but also that the real-time environment is still capable of playing the piece correctly.

A simplified block diagram is shown schematically in figure 1. The right-hand side of the figure shows the setup for a live performance of the piece. In the performance, microphones and other sensors (MIDI controllers and mouse or keyboard events) are sent in real time to a Pd patch which generates audio output to be played in the concert hall. During a performance of the piece (or perhaps a rehearsal or even a studio mock-up), all these inputs, and optionally the audio output, are recorded.

To set up the CI system, the realization patch is then run, virtually, in the interior of Reality Check, supplying it with the recorded inputs and in turn recording its audio outputs. The realization patch runs in a sub-process using the `pd~` object. If desired, extra output points may be inserted in the realization patch to allow finer-grained testing. This process results in an output soundfile, typically containing several audio channels.

At any subsequent time the realization patch can then be tested as shown at left in the figure. The recorded audio and control inputs are again played by the Reality Check patch into the realization patch, but this time the output is compared, sample by sample, with the earlier recorded output. If the two agree, the piece has been shown to be running correctly. If not, something has gone wrong and maintenance is needed.

Problems can come from at least three different sources: the underlying hardware and operating system, the application software, and the composer. Upgrades to the hard-

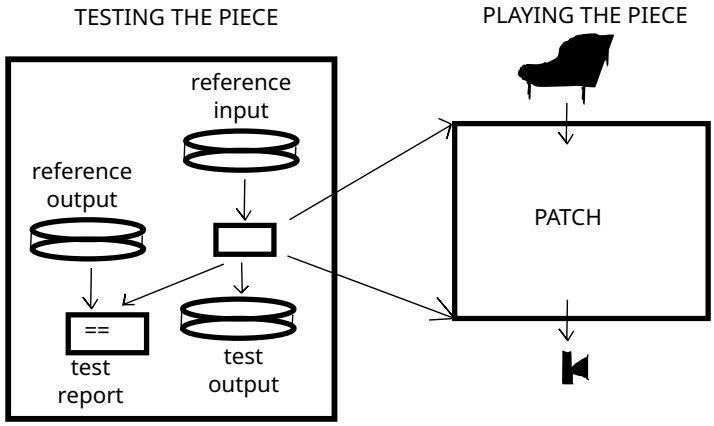

Figure 1. *Reality Check block diagram.*

ware and operating system should be made at a time when the older, working system is still available and running on a different computer. If the CI test succeeds on the older system but fails on the new one, this must be corrected in the application software. In the simplest case the software simply doesn't run on the new system and needs recompiling—for which we will need the sources to all the software at play. Most of the software components will doubtless be maintained separately, but the maintainer must be prepared for some software components eventually to be orphaned. The upkeep of all orphaned components then becomes part of the task of maintaining the realization.

Second, the application software may have become available in a new version that isn't strictly compatible with the one used in the realization. The temptation here would be to keep up the older software version, but this is dangerous, since the maintainer may end up in the position of maintaining an orphaned version of the software. The more forward-looking solution would be to use the existence of the CI test itself to apply gentle persuasion to the upstream software developers to maintain compatibility. As we will see below, this can be a practical approach in the case of widely used open-source software.

Finally, the composer might want to bring changes to bear on the piece itself. There are at least two classes of alterations that might be desired. First, the piece might be customized for a new concert hall or a new performer. A new concert hall might require reworking the spatialization of the patch's audio output. In the example currently under development, we are anticipating this by using the IRCAM spatializer software ("SPAT") as a separate process, considered as part of the specific performance production and not of the realization of the piece itself.

For another example, a performer might play along with one or more pre-recorded samples or tracks of the same instrument, and it might be desirable that the recordings be replaced by those of each new performer. This and similar adaptations can be made without bringing them back into the CI system.

On the other hand, often a composer decides to revise timings or levels or other parameters that are baked into the realization itself, either during rehearsals for a new performance or in a post-mortem studio session. In this case it is appropriate to re-introduce the changes into the CI system. And while for the later delight of musicologists we will want ot keep both the new and old versions, we can probably leave off maintaining the older version in the CI system in favor of the newer one.

4. TEST CASE

A production is underway for an opera, to premiere in 2025, that will certainly put this whole framework under an excellent stress test. The software components being used are Pure Data, Antescofo[6, 7], the `vstplugin~` Pd external object[8], and the Synful VST3 plugin[9]. Two of these systems are open-source and the other two, Antescofo and Synful, are in the process of being released in open source as well. As suggested above, the output of the realization patch is not spatialized but is instead realized as 32 audio tracks that are passed on to IRCAM's spatialization software. For purposes of testing the 32 tracks are optionally mixed down by a reference spatializer inside the Pd patch that is only adapted for a specific 6-channel speaker arrangement. There are currently only four audio inputs but as the production unfolds this will grow to about twenty. A twenty-channel DAW project will serve as a mock-up of the inputs before the actual production, and a bounce of the 20 outputs of the DAW will serve as the initial reference input for Reality Check.

Conceptually, the realization of the piece consists of a Pd patch that is already stable, plus a growing Antescofo script that constitutes the electronic "score". Since the Antescofo script is still under continuous development, it was deemed desirable to also make a test script that exercises all the elements of the patch systematically. CI tests have been built that separately verify the patch using the test script, and also verify the script for the piece itself, which last test is updated as the piece takes shape.

Along with the ongoing production, we are starting a CI regime in which the piece, along with three other pre-existing ones, are run automatically at regular intervals, thus flagging any problems that crop up. This CI system will have to be periodically updated in the future as hardware, operating systems, and software components evolve.

This regime has already borne fruit by catching, on two occasions, regressions in Pd that were since corrected, as well as calling attention to variations in floating-point rounding that are not yet resolved.

5. OBSERVATIONS AND CONCLUSIONS

A CI system such as this one can only work if some person or institution will take on the task of maintaining it. In our particular case, the host institution has a team of several permanent computer music designers who understand from years of bitter experience why this will be a good thing to do. Furthermore, we hope that once the procedures described here become widely used inside our own house that other centers that maintain repertories of live electronic music will adopt this or a similar methodology. If adoption is widespread enough, software developers will be able to take advantage of separately maintained CI repertories as a very useful collection of unit tests that their own software can benefit from to maintain backward compatibility.

For Reality Check or any other CI system to work in this way requires that the software components used be able to run in a virtual environment (such as inside `pd~`) and also be deterministically able to guarantee exact reproducibility of audio output from one run to another, given identical inputs. This is apparently the case for the components used in our main test case, as shown by the fact that the CI system currently passes the realization. Precisely which real-time computer music environments offer this possibility is not known.

This requirement of determinism does not forbid the use of stochastic processes in a realization, provided they rely on pseudo-random number generators, as are almost universally used in computer music software. It would nonetheless be forbidden, for instance, to use the current date or room temperature as a random seed; such a seed would have to be fixed. Certain other non-deterministic operations are also forbidden, such as measuring CPU time or querying the current OS version.

There is also the issue of truncation errors that arise from rounding in floating-point computations. In the pieces being tested so far these give variations in the audio output that are very small in magnitude and fall beneath the declared error threshold in the CI system. But running an unstable process such as a Lorenz attractor would give rise to wide divergences in output for different rounding schemes. This is currently a problem for the developers of Pd, Antescofo, and Synful to attend to.

We have nonetheless shown that a CI system can be devised that helps in the maintenance of live electronic music realizations. The system not only flags problems, but its very existence is an invitation to composers to develop their music in CI-controllable form, using deterministic and reproducible software components. It is already also useful as a tool for software maintainers to catch and correct any incompatibilities that might crop up in progressive versions of their programs.

Acknowledgments

This work is supported by IRCAM and by the DAFNE+ project under Horizon Europe Grant Agreement number 101061548.

6. REFERENCES

[1] S. Lemouton, "The electroacoustic repertoire: Is there a librarian?" *array.*, pp. 7–14, 2020.

[2] M. Akkerman, "'This hardware is now obsolete.' Marc-André Dalbavie's Diadèmes." in *Proceedings of the International Computer Music Conference*. Ann Arbor: International Computer Music Association, 2017.

[3] A. Vincent, A. Bonardi, and J. Barthélemy, "Pourrons-nous préserver la musique avec dispositif électronique?" in *Sciences humaines et patrimoine numérique*, 2010, pp. 1–1.

[4] A. Bonardi, "Rejouer une œuvre avec l'électronique temps réel: enjeux informatiques et musicologiques," 2015.

[5] J. Rudi and J. Bullock, "The Integra project," in *Proceedings of the 2011 Electroacoustic Music Society Conference*, 2011.

[6] A. Cont, "ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music." in *International Computer Music Conference (ICMC)*, 2008, pp. 33–40.

[7] J.-L. Giavitto, J.-M. Echeveste, A. Cont, and P. Cuvillier, "Time, Timelines and Temporal Scopes in the Antescofo DSL v1. 0," in *International Computer Music Conference (ICMC)*, 2017.

[8] C. Ressi, "[vstplugin~]–A Pd external for hosting VST plugins," *Revista Vórtex*, vol. 9, no. 2, pp. 20–20, 2021.

[9] E. Lindemann, "Music synthesis with reconstructive phrase modeling," *IEEE Signal Processing Magazine*, vol. 24, no. 2, pp. 80–91, 2007.