Application of Fault Tolerance to Interactive Music

John Young
Peabody Conservatory of Music

Introduction

Ever since humans first fashioned tools, we have had to consider the reliability of those tools and cope with the consequences of their failure. The complexity of our tools in the digital age has given rise to the field of fault tolerance (FT), with the objective of delivering reliable operation even during sub-optimal circumstances. Over the past fifty years, FT has steadily advanced in stride with the permeation of computers into all aspects of society and human welfare. As we become more deeply enmeshed in the web of our own machinery, the repercussions of its failure grow more profound.

Digital tools have also become essential to many evolving forms of music. Interactive works are particularly dependent on electronics, which must respond appropriately to a human performer in real-time, just as a traditional instrument or ensemble partner would. While few would argue that interactive music represents the same life-or-death criticality as a manned space mission or air-traffic control system, the performance of an interactive piece has a similar quality of high-wire precision, in which everything must function perfectly to achieve success. Even on a budget somewhat below the level of key public infrastructure, the concepts and techniques developed in pursuit of FT can be usefully applied to the compositional design and performance of interactive music. The nature of sound leaves little margin for error in maintaining the suspension of reality which a flawless performance can convey. By incorporating practical elements of FT into the entire creative process, the experience of interactive music can be improved for composers, performers, and audiences alike, bringing the quest for that elusive perfect performance closer to reality.

This paper is intended more to precipitate discussion than as a rigorous treatise on specific FT modeling techniques for interactive music. There will be no analysis of comparative component failure rates, no enumeration of possible complications due to varying environmental conditions, and no estimation of explicit cost/benefit ratios for guaranteed levels of uptime, though such detailed metrics might prove worthwhile in some cases. Rather, the objective here is to present FT as a practical philosophy: a means of combating the frustration of dependence on inherently unpredictable tools, and a way of harnessing the tools themselves to monitor, diagnose, and correct their own operation. In addition to introducing some fundamentals of fault tolerance from the perspective of interactive music, concrete examples and first steps towards functional redundancy in the context of MAX/MSP will also be described.

Principles of Fault Tolerance

Basic elements of FT include defined system requirements, informed system design, fault avoidance, rigorous system evaluation, and operational fault tolerance. To avoid ambiguity, in FT a “fault” is defined as a hardware or software flaw that becomes manifested as an “error,” a departure from optimal operation that left uncorrected can result in a “failure,” where the produced results differ from those specified and expected. Faults can arise spontaneously in response to environmental changes, or as a result of human factors related either to improper design or incorrect interaction with the system. Approaching interactive music from an FT perspective helps to focus on the aspects of implementation over which we can exercise control, thus helping to better determine where to focus attention and resources for maximum reliability gains.

Defined system requirements specify the results desired from hardware and software in the context of performance. At a minimum, this would generally consist of a complete realization of an interactive work as intended by the composer. However, as this goal subsumes numerous factors beyond the realm of human influence, a more appropriate emphasis might be high “performability”. This term incorporates the comfort level of performers with the electronics, in rehearsal as well as live situations, with an ideal target of attaining the predictability of an acoustic instrument. In determining standards for success versus failure, audience expectations should be considered—the lay public will certainly be less tolerant of technical difficulties than peers at a computer music conference. Properly balancing these requirements facilitates the illusion of effortlessness that contributes to an effective performance.

Informed system design begins at the germination of a work, continuing through the compositional process, following established FT guidelines. These strategies focus on managing system complexity, optimizing use of resources, finalizing a work well in advance of performance, and weighing exposure of elements against their reliability. Choosing the most straightforward solution to each problem usually results in a system that is easier to troubleshoot, reconfigure, and enhance. Object-oriented tools can be useful for developing complex projects, as individual components can be fully tested before being integrated into larger structures. Systems universally tend to fail more frequently under heavy processing loads, so increased efficiency directly improves stability. In order to reserve time for such optimization, complex interactive systems should be feature-complete as early as possible. Each change in functionality also has the potential to introduce new faults, so adding features late in the development cycle undermines any hope of thorough testing or sufficient rehearsal. Compositionally, certain elements—grand gestures meant to trigger loud, sudden sounds, for example—are freighted with more potential for disaster than others. Awareness of such risks leads to consideration and resolution of possible conflicts between artistic intent and goals of FT. Adherence to sound principles of design is essential, as flawed design will almost certainly lead to flawed results.

Fault avoidance involves minimizing opportunities for faults to arise, in coordination with system design. Preemptive techniques include selecting reliable hardware and software, using empirically proven tools, and preparing adequate documentation. Live techniques include monitoring system health in real-time, and ensuring effective communication among everyone responsible for performance success. Most of us don't have the luxury of specifying hardware, operating systems, and software explicitly to meet our needs. We usually must choose from a narrow range of general-purpose solutions, and proceed to customize them to accommodate our intentions. For those with sufficient programming expertise, open-source alternatives at least provide complete command of the software environment, although they can also carry a commensurate burden. Everyone else must rely on community knowledge and personal experience to determine which tools are most suitable. Given that interactive music frequently involves some level of collaboration, clearly documenting proper interactions, as well as procedures for recovery from errors, helps everyone involved execute more efficiently. Effective communication, both in advance and during performance, minimizes the role of human error in the overall stability of the system. Likewise, a diagnostic interface for monitoring system health in real-time can reveal a trend towards failure in time to take corrective action. Communication objectives can be thought of as getting the right information to the right person at the right time. Fault avoidance can also cover larger issues of educating composers and performers in FT practices, and giving feedback to hardware and software developers on the robustness of their systems. Attention to all forms of fault avoidance, though perhaps the least glamorous aspect of FT, is essential to reducing fault potential to a manageable level.

Rigorous system evaluation must comprehensively test all aspects of an interactive work. This requirement should influence system design to minimize the burden of testing. Evaluation includes identifying relative dependability at potential points of failure, assessing environmental factors of the venue, simulating real performance with both normal and unexpected data, and devising tests that can be run during a live event for monitoring purposes that are imperceptible to the audience. While strict quantification might be overkill, noting when a system fails more regularly during certain operations than others guides debugging effort to focus on the less reliable functions. Anticipating the impact of a venue can be difficult, but adequate preparation and rehearsal can mitigate all but the most extreme conditions. As a rule, the more unknown variables involved in a performance, the more robust a set of backup plans should be. It isn't possible to overemphasize the importance of testing—professional musicians spend hours a day practicing their instruments, and interactive musicians should demonstrate equal commitment to honing their systems. Stress-testing in advance is the best way to reduce the likelihood that an actual performance will enter uncharted territory.

Operational FT involves predictably compensating for faults when they occur. This flexibility begins with detecting the occurrence of an error. Responses may include transparent error correction, reliance on automatic redundant systems, graceful degradation of functionality, or manual intervention. FT can also encompass improvisation, which demands good feedback to a performer about the health of his/her resources, allowing human skills and ingenuity to cover for technical difficulties. Multiple levels of failure should be anticipated and corresponding response modes implemented, including for worst-case scenarios. One option in the event of serious malfunction should be to normalize the system to the next cue and continue from there. Regardless of the situation, an audience would rather have some rendition of a work, however flawed, rather than nothing at all, so even a completely pre-recorded version should be ready to go. Simply put, always have a backup plan, a backup for the backup plan, and some idea of what to do in the event of anything short of a natural disaster.

Practical Applications

The fundamentals presented above can be easily translated into concrete steps of action. For example, collected wisdom in the MAX/MSP environment has resulted in numerous “common sense” recommendations. Under the category of fault avoidance fall such preventative measures as maintaining a standard pre-performance checklist. This list might include turning AppleTalk off, turning Virtual Memory off, turning Energy Saver off, and checking Sound I/O settings, volume settings, OverDrive setting, audio vector sizes, and DSP status. Rehearsing not only the work itself, but also the physical setup of the piece, will leave maximum opportunity for troubleshooting during precious pre-performance time at the venue. Using one’s own or familiar hardware and simplifying the interface between electronics and on-site sound reinforcement likewise minimizes confusion. A recent addition to the arsenal is the option of burning an entire work to CD-ROM, including a bootable operating system and all essential software, an effective hedge against data corruption. However, variations in hardware can render this technique useless or at least high-maintenance. Further software design hints include not depending on inherent execution ordering such as right-to-left, range-checking all inputs, and setting min/max boundaries on sliders, dials, number boxes, etc. Following the rules that work best for you as a matter of course frees your mind from concern with these routine minutiae and lets you concentrate instead on the more important issues of artistry.

The graphical basis of MAX/MSP can be both blessing and curse, but it does facilitate building monitoring systems directly into a work. Designing key status indicators into the performance interface aids in all phases of testing and debugging, as does creating initialization routines for subsystems as well as the entire piece. Important cues and parameters should be visible at a glance, and ideally performers as well as technicians should have access to such status information. MAX/MSP is conducive to a modular approach, useful not only for design and testing, but also for efficiency. By disabling unused subsystems and signal paths, resources can be rationed to provide a comfortable stability buffer. One of the strengths of object-orientation is the ability to invest in your own toolbox by taking the time to perfect each of these techniques and building blocks, leveraging accumulated knowledge into the foundation of every new project.

Case Study: Redundancy

No matter how well a system is specified, designed, implemented, and tested, imperfections that lead to faults and errors will remain. Unfortunately, even the best diagnostic and repair architecture is inadequate insurance if it runs on top of current general-purpose operating systems. Thus, one of the most successful means of realizing operational FT from off-the-shelf components is through redundancy. In theory, hardware and software redundancy are considered separately, but in practice they go hand-in-hand. Complete redundancy can be expensive, but most of the benefits can usually be reaped with only partial redundancy.

In a typical configuration for interactive music, a human performer generates input, through either a physical controller or microphone, to a computer which then translates that input into sound. Out of the possible points of failure in this generic scenario—performer, instrument, microphone, MIDI interface, computer, outboard synthesis and effects modules, and sound reinforcement—the most unreliable component is probably the computer. By duplicating the function of the computer as well as its inputs and outputs, a high level of FT can be attained.

Creating a redundant system using multiple computers involves its own complications. The computers must communicate their status to each other, maintain synchronization, and transfer processing quickly in the event of failure. Like FT itself, these clustering techniques run the gamut from basic to industrial-strength. For the purposes of exploring redundancy in interactive music, I developed a simple set of FT patches in MAX/MSP which were demonstrated at SEAMUS 2001. The configuration consisted of two Macintosh laptops communicating over Ethernet, receiving identical input from a MIDI controller via a splitter, with audio outputs merged to a single stereo pair fed into local sound reinforcement. The laptops, though not identical models, ran basically the same operating system and software versions. The MAX/MSP patches differed only in the TCP/IP address used to transmit data to the other machine. The most important data consisted of a "heartbeat," a bang occurring at a regular interval, adjustable between 250 and 1500 milliseconds. Whichever machine first initiated the FT sequence automatically became "active," with all inputs and outputs enabled, forcing the other into "standby" mode, with inputs and outputs disabled by gate objects. If the standby computer did not receive a heartbeat from the active computer within the expected timeframe, it would switch itself to active status, enabling its outputs and continuing the performance.

To more accurately simulate a realistic interactive work, in which many functions operate independently of input, this basic redundancy in function was augmented with full parameter synchronization. Again using the network link, a "snapshot" of all time-varying parameters was sent from the active machine to the standby with every heartbeat, so in the event of failure, the standby would take over at almost the exact point at which the other had left off. Using these techniques, even advanced processes like real-time convolution can be transferred from one computer to another relatively seamlessly. Using a 250 ms heartbeat, the fail-over is barely noticeable, far superior to the work crashing to a halt. Even at that level, the CPU overhead is nominal compared to real-time DSP, so the redundancy itself doesn't detract appreciably from system stability. Retrofitting such a framework onto an existing piece would be difficult, given the need to tap into and mirror every significant parameter. However, incorporating redundancy into the design from the beginning does not involve much extra complexity, and the payoff could be the difference between disappointed silence and enthusiastic applause.

Future Developments

Investigation into more sophisticated redundancy methods continues. Possible paths of additional research include greater automation, remote control, and more complex bi-directional communication between systems. In its current form, the system requires some manual intervention, such as optimizing the heartbeat interval, which could be performed automatically. Controlling multiple systems from a single location might also be very useful, in cases where redundant computers could not be located in close physical proximity. This feature could easily extend to spreading workload across multiple machines or dedicating each to a specific task, commanded by a central console—much like an orchestra and conductor. These possibilities step well into the realm of distributed computing, where procedures for inter-process communication often involve greater autonomy and negotiation amongst machines to match services and resources amidst dynamic conditions at the most granular levels. As the role of the Internet in interactive music grows, these techniques will surely come more into play.

Conclusions

It seems that time is always the scarcest resource in any project. While integrating FT into interactive music may be valuable protection against the forces of entropy, if it compromises time that would be better invested in a more complete artistic vision of a work, then the project might still be considered a failure despite functioning flawlessly. This is the challenge of applying a discipline founded on quantitative measurement to an artistic endeavor where the judgement of success is infinitely more subtle. As always in interactive music, balance between the technical and the creative is essential. The principles and applications of FT presented here are merely another set of tools at our disposal, with the distinction that they have the power to make every other tool we use more dependable. The real reason to apply FT to interactive music is to have confidence in the systems supporting the creation of musical experience. When a performer trusts a computer as much as a piano, only then will the stage be set for realization of a masterpiece.

Acknowledgements

This paper would not have been possible without the inspiration and knowledge of McGregor Boyle, Ichiro Fujinaga, and Geoffrey Wright. Their decades of experience in connection with interactive music were essential to the progress of my research in this area. Thanks also go to Margaret Schedel, who provided invaluable assistance in developing the redundancy demonstration.

The experimental MAX/MSP patches described herein may be found at www.netmuse.org.

Bibliography

Cristian, F. "Understanding Fault-Tolerant Distributed Systems." *Communications of the ACM* 34(2): 56-78 (1991).

Gray, J. "Why Do Computers Stop and What Can Be Done About It?" In *Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems*, 3-12. Los Angeles, California: IEEE Computer Society Press, 1986.

Siewiorek, D., and R. Swarz. *Reliable Computer Systems: Design and Evaluation*. 3rd ed. Natick, Massachusetts: A K Peters, 1998.

Stankovic, J. "Distributed Real-Time Computing: The Next Generation." *Journal of the Society of Instrument and Control Engineers of Japan* 31(7): 726-736 (1992).

Werner, M., A. Polze, and M. Malek. "The Unstoppable Orchestra: A Responsive Distributed Application." In *Proceedings of the Third International Conference on Configurable Distributed Systems*, 154-60. Los Alamitos, California: IEEE Computer Society Press, 1996.