The IRCAM "Real-Time Platform"

François Déchelle, Maurizio de Cecco, Miller Puckette, David Zicarelli

The Ircam ``Real-Time Platform'': Evolution and Perspectives

François Déchelle, Maurizio De Cecco, Miller Puckette, David Zicarelli

ICMC 94, Aarhus (Danemark), 1994

Abstract

This paper describes the current evolutions and perspectives of the FTS real-time executive as the colonne vertébrale of the Ircam ``Real-Time Platform'' project, which is a software oriented continuation of the ``Ircam Musical Workstation'' project.

1 Introduction

The most popular real-time DSP oriented software for the well-known Ircam Musical Workstation [Lindemann91] is FTS [Puckette91], a distributed real-time executive. FTS is a message-oriented system, in which objects' activations are scheduled statically for DSP computations and dynamically for control computations. FTS has usually been used as a back end to the NeXTStep version of MAX.

2 The Open System approach

The Next Cube history proves once more that a long-lived technology must be as independent as possible of any particular hardware or software platform, and from vendor dependent choices. The current evolution of FTS is in the direction of a standard open system: FTS is becoming a hardware-independent system that can run on multiple hardware and software platforms, and interact with user applications in a well-defined framework.

2.1 FTS as an open server

The chosen approach is a client/server architecture, in which FTS is the real-time server of one or more client applications connected to it via a well-defined hardware independent protocol. The communication itself can use different transport layers: standard UNIX pipes, TCP connections or shared memory in specialized hardware architectures. The chosen transport is completely transparent with respect to the client application; the same application can use different transport layers in different sessions, depending on the configuration. The kind of server used, and the platform on which the FTS server runs, is also completely transparent to the application: the same client can connect to a standard Ircam Musical Workstation card, an Alpha workstation or a TMS320C40-based system in different sessions, without any difference from the client's point of view. Of course, different servers can include different features, like different sets of objects, different performance levels or a different audio I/O structure. A well-behaved client should not make assumptions as to the existence of a particular server feature, but the FTS protocol includes support to publish the main features of a server so that the client can actually know which features are supported by a particular server. One of the benefits of this approach is that the real-time architecture and graphical development environments are more fully decoupled, and can run on physically different machines connected through Ethernet or another network. A well-defined interface between the two components allows specialized applications, using FTS for their DSP computations, to be developed in programming languages other than Max. Currently, language bindings for the FTS library are provided

in C, C++, Common Lisp and Dylan; other utilities developed include a shell command line interface to control the FTS server and FTS applications, and a Motif library to build specialized application interfaces connected to FTS. The support for multiple clients for the same server allows the use of Max as the development environment for the FTS part, while the final application interface is developed as an independent client by other means.

2.2 Multiple architectures

The immediate goal is the porting of FTS to commercially available hardware architectures, which are currently:

- PC with add-on Texas Instruments TMS320C40 DSP boards providing the scalability required by large musical applications,
- Unix workstations with audio devices and high processing power, such as Silicon Graphics and DEC Alpha.

3 Evolution of FTS

A number of longer-term issues are also under consideration; a number of case studies with FTS users both in and outside of Ircam showed two areas of the system where improvements or added functionalities would be worthwhile. These are automatic partitioning and compilation of control algorithms.

3.1 Partitioning

Like money, computing power is never plentiful enough; most of the applications developed at Ircam with the Musical Workstation use the maximally expanded configuration (three cards, i.e. six *i860* processors), and use the available computing resources, CPU and memory, to their very limits. The main consequence is that the partitioning of the different tasks on different processors is at best a critical process; in practice, once close to the machine limits, finding a good partitioning of the algorithm between the processors becomes the most time-consuming development activity. The other consequence is that working close to the limits implies that a number of stricly architectural and platform-dependent optimization choices are performed by the FTS programmer, and are hardcoded in the patch itself. In the anticipated multi-platform scenario, this is clearly a problem that will increase the cost of porting applications across platforms. In this scenario, a system support for application partitioning is seen as unavoidable. A development activity in this area has only just started; the goal is to reduce the development cost of distributed FTS applications: in the short term, we will give more accurate metering tools to monitor the load of the different parts of the system; in the medium term, we will develop semi-automated utilities for partitioning of FTS programs.

3.2 Compilation

The FTS DSP engine is currently highly optimized, at least at the single DSP operation level. However, the control algorithm is essentially interpreted. Investigation of finished works shows that the control part is not negligible and is sometimes the bottleneck, especially when the system reacts to external inputs. FTS developers often use workarounds, like extending the system audio FIFOs to cope with peak loads (which introduces audio delays that are sometimes difficult to handle), and splitting the control computation between different ticks by means of delay lines. In order to better optimize the control computation, the idea of compiling the control part of a patch instead of interpreting it, has been considered. Given the nature of the FTS programming language (simple data types, simple recursive control structure), compiling it directly to a low level byte code (or in theory to directly executable machine code), is rather straightforward; a feasibility analysis showed that for patches made of standard control objects (like `int`, `float`, `trigger` and `select`), using standard optimization techniques such as inlining, peep-hole optimization, and a bit of data-flow analysis, we achieve translation ratios around 1-2

machine instructions per FTS object, giving a theoretically possible improvement in control performance between one and two orders of magnitude over the existing system, depending on the target compilation language. The compiling algorithm is also compatible with an incremental implementation, something that is essential to avoid any impact on the ease of use of the Max programming environment. Compiling the control algorithm also allows for a better integration of the DSP computation (currently compiled to an intermediate language) and the control computation, permitting a more flexible semantic of the interaction between the two, more flexible scheduling strategies and also the application of global optimization to the DSP part (like elimination of `sig`objects for instance). Using a compiled approach also allows the implementation of delivery-only versions of the FTS kernel, for use in embedded or dedicated applications.

4 Summary

We described in this paper the short-term and long-term evolution of the FTS real-time executive, making it an open server and adding new features such as partitioning tools for faster multiprocessor development and optimization of control computation.

References

[Lindemann91]
Eric Lindemann, François Déchelle, Michel Starkier, Bennett Smith. The Architecture of the IRCAM Musical Workstation. Computer Music Journal, 15(3): pp. 41-50.

[Puckette91]
Miller Puckette. FTS: A Real-Time Monitor for Multiprocessor Music Synthesis. Computer Music Journal, 15(3): pp. 58-68.