Integrated Development Environment for Computer Music Composition

Jon Drummond
Joint Research Center for Advanced Systems Engineering
CSIRO - Macquarie University
jond@mpce.mq.edu.au, http://www.jrcase.mq.edu.au/~jond

Abstract

This paper describes an integrated development environment for computer music composition. While a large number of software tool sets are currently available to assist in different aspects and stages of the composition process, there exists a significant lack in software support for the processes involved in the design of a new composition. An integrated tool set for computer music composition is described. The tool set addresses these issues by providing support for different aspects of the composition process and by integrating existing software tool sets.

1 Introduction

The possibilities and potentials presented by computer music have long been an intoxicating dream for composers. The application of computers to music offers unique creative possibilities for music composition. Computer music systems enable the composer to work with sound directly. The analogy of a sound sculptor is frequently used to describe the process. The composer can work with any aspects of pitch, volume, timbre, spectrums and spatialisation, all with equal ease. The composition can exist through performance, created as the result of interactions with the software composition. The composition process itself becomes an interaction between the composer/performer and the software.

While current computer music systems have achieved some of the possibilities promised by the new technology, a number of significant aspects remain unsupported and difficult to accomplish. Limited software support is provided for the design process of the composition. Many existing software languages for computer music require an extensive learning period. The models used within such systems are often unrelated to the composition task being performed.

Effective interface design is an integral part of creating flexible and useable high-level integrated tool sets. Poor interface design can greatly reduce the useability of a computer music application. Inappropriate visual representations can hinder the composition process. This results in convoluted representations and prevent the exploration of alternate compositional solutions.

While high-level software support can enhance the composition process, it also has the potential to restrict the flexibility and creative freedom possible within a system. Brün [1] observed that computer music systems are not equal. Every system constrains musicians to a restricted set of operations; every view on a piece is a filter that biases the viewer’s attention to a particular perspective. Perhaps, as Roads [4] has noted, the ‘holy grail’ of a universal representation is antithetical to the goals of creative music. Music is constantly evolving and one definitive solution to questions of music representation may not be particularly beneficial. An integrated and extensible tool set can provide the software support needed for computer music composition while maintaining the flexibility required by composition.

2 Interface and Composition

The interface in this context is defined by the interplay of both the visual representations used and the underlying model of the software. The composer defines compositional processes and functions through interaction with the visual interface (text and/or graphic). The underlying model of the software defines the nature of the possible interactions with the software. The nature of the modes of interaction determines the extent of possibilities the user has within any particular system. This contributes to the flexibility and creativity offered by any particular software tool set.
2.1 Composition Requirements

An effective interface design should address the needs of composers and provide representations that relate to the processes being utilised. To achieve this, a set of interface requirements for composition is required. A number of studies have been carried out to create such a set of composers’ needs ([2] and [3]). The following are key issues arising from such studies.

Interfaces should allow the composer to work at different levels of complexity. Depending on the nature of the project a composer could customise such a system to hide or reveal different levels of complexity as required. In many situations too much detail can clutter the interface and impede the task being performed.

Software should support classification and automation of appropriate tasks, based on the composer’s requirements. A multi-level interface of this type would allow the software interface to be customised to meet the needs of the composer and of the specific project.

Support is needed for both formal and informal graphic notations. Informal graphic representations are frequently used as effective high-level descriptions of structures and form.

Software support is required for work on form independently of the materials within it. With appropriate software support, the development and manipulation of musical forms may be carried out prior to, concurrent with and independent of the development of musical materials.

2.2 Interactions Between Representations

To design effective interfaces for computer composition we need to identify the processes and interactions that are involved. The composition process is a creative activity and as such is elusive to being modelled. Methods of composition vary with the individual and the type of project. Despite this, it is possible to make some generalisations. When working with software in composition there exists interactions between three key representations of the work. (Figure 1) These representations are the-

- Mental model or concept of the composition.
- Visual representations provided by the interface.
- Performance of the current version/s.

These components interact in a feedback loop. Material is developed and entered into the software score via the interface (Figure 2), this material is performed and auditioned both through computer sonification and internal performance. As a result of this auditioning process the material is edited via the interface and subsequently reauditioned. The overall concept of the piece can also be updated as a consequence of this process with the resultant changes filtering down to the lower levels of the structure.

3 The Tool Set

The tool set consists of an integrated and extensible set of applications including an event editor, patch builder, object link viewer, and an envelope editor. The tool set integrates with an expandable set of existing software applications including audio editors, sound file playback utilities and synthesis languages. The tool set is written primarily in Tk Tcl and runs on a Silicon Graphics under IRIX 5.3. It currently integrates with the Csoud synthesis language. The following provides an overview of the key features of the tool set.

3.1 Event Editor

The event editor forms the central interface to the system (Figure 3). From this window, objects are
created, defined, edited and placed in time with respect to each other. The event editor also provides access to all other parts of the tool set.

The graphic layout of the time line event editor references the studio metaphor of the multi-track recorder. The interface extends the analogy to allow multi-track style manipulations to be applied not only to sound events but to any type of event defined by the system to be an object.

Sound objects and functions are defined within event containers. An object can be thought of as a flexible building block. Any primitive function supported by the system can be thought of as an object. An object can even be another time line function.

An event or container is placed on one of the tracks by double clicking at the desired location. Containers are moved, re-sized, and placed on different tracks graphically via mouse click and drag operations. Specific event times can also be entered numerically. Tracks operate as visual place holders.

Event editor containers are assigned functional names by the user. The object names provide the means by which other parts of the system can access event editor objects. The event editor also provides links to all other tools in the system.

3.2 Patch Builder

The patch builder is a graphic interface in which the contents of a time line event are defined. Compositional functions are built up from functions provided by the tool set. These include control functions, envelopes, filters and synthesis routines. A function can also be a container for another patch. Function parameters are entered via dialogue widows. Each function object can be assigned a user defined name. Referencing function names allows other timeline events to access specific functions in other containers.

3.3 Envelope Editor

The graphic envelope editor allows control streams and data sets to be defined via mouse input. (Figure 5) The function of the control stream is defined via the patch editor. The spline based drawing tool can be used to define points or linear/exponential envelopes.

4 Future Directions

The function set of the software will be expanded. Methods for integrating user defined functions will be defined. A limited version will be ported to the Power PC platform.

5 References