An Intelligent Music Repository

Barry M Eaglestone, Albert Verschoor,
Bradford University, Centrum voor Kennistechnologie,
Bradford, UK, Utrecht, NL,
barry@uk.ac.bradford.computing
Sound Information Technology Research Group (STRING)

Abstract: This paper argues the case for adapting integrated project support environment (IPSE) technology for use by composers, and proposes an architecture for the repository of music-IPSE. An artificial intelligence technology to be used within the IPSE for intelligent sound archiving is described, and progress towards a prototype is reported.

1. Introduction

Composition of computer music has much in common with many engineering activities (musiconology on the other hand is analogous to 'reverse engineering' [1]). However, support environments for composition are currently inferior to those used in many engineering disciplines. We propose to address this deficiency by transferring and adapting of Integrated Project Support Environment (IPSE) technology so that it may be used by composers, musicians, etc. Specifically this paper concentrates on the adaptation of the repository component of an IPSE and AI techniques for intelligent perceptron-based access to sounds represented in the repository.

2. The Composer as Engineer

The parallels between composer and engineer become apparent when the activity of composition is examined. The composer, like the conventional engineer, must transform a set of requirements into a product design, i.e. a score or realisation of a musical work, through an iterative cycle of analysis, hypothesis, building and testing. He or she is aided in this task by the various tools and techniques that may be applied to sounds and scores, e.g. analysis tools, sequencers, composition languages. Constructing or composing sounds is therefore a design process. Design is typically an activity in which we change some part of our environment, but where the set of candidate designs is infinite. To control the size of the solution space, the engineering technique of 'decomposition' is used: decomposition is the splitting up of the total product into manageable subcomponents. The subcomponents of a product are either designed themselves or are existing products, possibly modified, which have been reused. Reuse is applied for economical reasons as well as for cognitive economy. It may be cheaper to develop a product composed of existing components, than one in which each component must be separately designed.

Decomposition is used by composers so that they can cope with the overall complexity by working on small parts of the composition, and also concentrate on overall structure by abstracting out the detail. A composer will reuse design components such as compositional forms, instruments, sounds and their analyses, synthesis algorithms, etc. In fact, over the years a composer (or community of composers) may assemble a large archive of material generated during composition and of potential reuse in later works.

During the development lifecycle of a product, such as a composition, its overall design, and each of its parts, will pass through many intermediate stages. The composer, for example, will try out alternative versions and successive refinements, and will make qualitative comparative assessments of these. A prototyping approach is typically used as a basis for the assessment; the sounds will be prototyped from the design components, and this will enable the composer to audition the evolving composition.

A consequence of the above characteristics of composition is that like the engineer, the composer's life is complicated by various 'object management' and 'process management' tasks. He or she must: keep track of the different versions of a composition, of each of its parts, and of the relationships between them; assemble, maintain and utilise large archives of reusable material; develop, maintain and utilise a toolkit of programs and techniques for using them; create and maintain documentation, such as programme notes, performance instructions, annotation of the archived material for reuse, and user instruction for the various tools; and keep track of the composition process e.g. by planning and journaling.

3. Integrated Project Support Environment (IPSE)

In many engineering disciplines, particularly in software engineering, the above object and process management problems have given rise to IPSEs which help the engineers to cope with them. The underlying IPSE approach is to factor out and centralise within the IPSE various facets which are common to many applications within some engineering discipline. Specifically, an IPSE centralises knowledge about the meaning of the objects, e.g. data and code, that the engineer must utilise. It centralises meaning in terms
of structure (structural semantics), things that can be done to objects (behavioural semantics), and the way in which they can be utilised (evolutionary semantics). [See early chapters of [2] for an excellent introduction to IPSE in software engineering, and its benefits].

Environments such as UNIX may be thought of as first-generation IPSEs. UNIX objects are stored within files, and the behavioural semantics are captured in the form of sets of tools (e.g., vace, lex, cc, grep). Users may extend the behavioural semantics by writing new tools, but may also reuse tools in new combinations and contexts through a tools interconnection mechanism. Object management is through explicit use of special tools (e.g., scat and make). However, first-generation IPSEs do not include generic process management or documentation tools.

A second generation of IPSEs, broadly based on the STONE MAN architecture [3], goes some way to solving these outstanding problems by utilising extended database technology to implement a central repository in which are stored all objects utilised by the engineer. Around this repository are clustered the tools and a tools interconnection mechanism, and finally a presentation manager supports a uniform interface to all the tools. An advantage of this architecture is that the repository can present a natural representation of the stored information, thus reducing the dependence of tools and applications on physical data structures. Also, configuration and version management features can be built into the repository, so partly alleviating the engineer of this responsibility [4] for a review of version models. Documentation becomes a by-product of engineering, rather than an onerous (and sometimes omitted) chore.

4. A Model IPSE (M-IPSE)

Given the above parallels between composers and engineers, we believe IPSE technology is also applicable to music. M-IPSE users may be viewed as engineers who develop product designs, the products being compositions, instruments, performances, etc. In fact current systems for manipulating sound are typically specialisations of first-generation IPSEs, e.g., NeXT, CARL, and IRCAM musical workstation all work within UNIX or UNIX-like environments. Objects such as sounds, tools, documents, etc., are stored as files and are manipulated using toolkits (sometimes object-oriented). New tools can be programmed and/or constructed through specialisation, generalisation, or aggregation of existing ones.

This type of system has weaknesses of first generation IPSE in general, including: (i) the dependence of tools and applications upon file structures, which can lead to extensive rewrites of existing tools and/or a proliferation of file formats and conversion procedures; (ii) difficulty in keeping track of files and relationships between them; (iii) difficulty in keeping track of what has been done and what is still to be done; and (iv) difficulty in creating documentation.

We therefore advocate the adoption of a second generation IPSE technology for music users [1,5a,5b]. The creation of a second-generation M-IPSE requires:
- specialisation of the toolkit to include tools specific to sound manipulation;
- extension of the presentation manager to accommodate additional non-conventional styles of input/output, e.g., keyboard, MIDI, graphical, audio;
- extension of the repository to include sounds and knowledge about them;
- addition of a back-end DSP facility for prototyping sounds from their descriptions within the repository;
- integration of artificial intelligence (AI) into the repository and tools in order to implement mappings between user ideas about sound and the stored (physical) sound.

This extended IPSE architecture is depicted in figure 1.

5. The Repository Architecture

Our repository architecture comprises three partitions: (i) project models include data and rules relating to the sound information system's (e.g., compositions or instrument parameters) which are being engineered within current projects; (ii) archives are of relation objects and include a database of sounds which is indexed by both physical and perceptual features (SITRO is also investigating the use of neural nets as a mechanism for this perceptual indexing [5d,5b]); (iii) problem domain knowledge is the repository for knowledge about sound information systems. This will represent ideas by which perceptual features of sounds are extracted, and rules for fuzzy, perception-based retrieval, modification and generation of sounds.

Parts of the project models will form active models within the back-end DSP server, where they will be interpreted to realise the specified synthesizer and sounds. Ultimately, our objective is to implement an (object) tight binding between the active and project models, thus making it possible to control and manipulate the synthesizer and sound characteristics in real-time by manipulating their abstractions in the repository [7]. In this way, we aim to provide data and logic independence from the physical aspects of sound production.

This repository interface will support an extended SQL [8], and also a deductive mechanism for applying the represented knowledge.
6. Intelligent Retrieval from the Repository

In our view, translation of subjective experiences of sound to parameters on a synthesizer can also be treated as a design problem. The design process will include decomposition and reuse, as described above. In fact, reuse, modification and the creation of subcomponents are generic subactivities of design. Reuse is made of existing instruments of which the sound is known, and even when we make use of non-existing sounds explorations are usually made around existing sounds known to the designer. This design process is therefore actually one of modification, because it starts with an existing idea upon which modifications are made until the solution is found.

![Diagram of Integrated Project Support Environment Management System]

**Figure 1: M-IPSE Architecture**

Although retrieval is considered a simple classification task in which known attributes are fed into a computer system to retrieve a known object, when retrieving sound we do not know the attributes of the sounds because there is no standard vocabulary for sounds. Furthermore, since the object to be retrieved is new, we do not know the object itself. It does not matter to the user whether or not a sound is reused or created since the user just wants a solution to his or her problem. This is why we approach retrieval of sounds as a design problem.

Much research after knowledge involved in design is done in AI. [11] distinguishes between three types of design corresponding to the previously mentioned generic activities in design: reuse, modification and creation. In the first type, the goals are not well specified and there is no knowledge about the decomposition of the problem or plans for the subproblems. This is called open ended 'creative' design. In sound design this is the process of making a composition, without having any knowledge of the sound or structure to be used. In this case we do not know which products will be used, but their structure might be very simple.

In the second type, there is knowledge about the problem decomposition so we know about the kinds of subcomponents to be used. However, in most cases, substantial modifications need to be made to the subcomponents. In sound design we know for instance the patch but have little knowledge about timbre.

ICMC 439
The third type of design is routine: there is an effective problem decomposition, there are design plans for components, and actions for failure. This is the type of knowledge that is required for sound retrieval.

In our research we implement a system for sound retrieval in a design specification language called DSPL. [12] In DSPL, we make use of the knowledge involved in specialists' plans, tasks, steps and constraints, as explained below.

A specialist will make an attempt to design a part of a component. The specialists are typically placed in a hierarchy comprising the goal product to be designed. A specialist works use of a specialist lower in the hierarchy or has local design knowledge in the form of constraints. Accordingly, a number of plans are placed within the specialist specifications. The plans determine when each specialist will be executed. Constraints check on the integrity of the design decisions that were made.

A step in DSPL makes a decision while taking into account the constraints. For example, a step would make a decision about the amplitude of a sound. Another step would make one about the vibrato...

Tasks express the sequence of steps to be taken towards the design of one coherent section of a component. These steps are again interimpacted with constraints. The constraints will test for a relationship between two or more attributes at a particular stage of the design.

An advantage of our design approach is that the repository facilities such as configuration and versions management for engineering compositions, etc., may also be applied to the knowledge engineering task of designing sound to percept mappings.

6. Implementation

A prototype sound repository is currently being used for testing and developing our design approach to intelligent sound accessing. This prototype is built on a SUN Sparstation using EMProlog and Ingens: EM Prolog includes interfaces to the host environment and to relational databases. The Prolog provides the intelligent repository access, while Ingetns, enhanced with a version model, supports some of the design tasks.

7. Conclusion and Future Work

In this paper we have argued the need for a transferal of IPSE technology for use in music applications, and have proposed an architecture for a M-IPSE. Mapping of user sound ideas to sound system parameters is an AI problem, and we have presented a design approach based upon recent developments in AI.

The research described is part of a larger Sound Information Technology project which is being carried out jointly by the authors, Tamas Ungvary (Royal Institute of Technology, Stockholm), and Bernhard Fischer (Institut für Kommunikationswissenschaft, Berlin), with assistance from Muscep and MusSound.

8. Acknowledgement

Our thanks to Shaun Oates and Kees de Koning for their contribution of ideas and software.

References


ICMC 440