POCO: AN ENVIRONMENT FOR ANALYSING, MODIFYING, AND GENERATING EXPRESSION IN MUSIC

Henk-Jan Houting

Music Department
City University
Nothampton Square
GB-London EC1V 0HB

Centre for Knowledge Technology
Utrecht School of the Arts
Lange Vierstraat 2B
NL-3511 BK Utrecht

POCO is a workbench for analysing, modifying and generating expression in music. It is aimed to use in a research context. A consistent and flexible representation of musical objects and structure was designed. The integration of existing models of expression made it possible to compare and combine these models using the same performance and score data. New tools were developed for specific "micro surgery" on expression. A lot of attention was given to the openness, integration, and expandability of the system.

INTRODUCTION

As part of our research on the modelling of expression in musical performance at City University, London, we developed a workbench named POCO. It consists of a collection of tools that can be used for the analysis, modification, and generation of expression in a research context. The research project combines three perspectives: musicological aspects (what are the rules of expression used in different styles of music), cognitive aspects (how does a good performance or interpretation facilitate the understanding of the music by the listener), and computational aspects (the design of appropriate data structures and development of programs dealing with expression). The latter will be described in this paper.

Before describing the system, the next section will give the reader a flavour of some of the problems and ideas related to this research.

DREAMS AND VISIONS

When starting a project like this there are various directions one might take. This is the moment to fantasize about the ideal system, as later on one's thoughts will probably tend to be directed by their feasibility. We will sum up a collection of dreams that we would like to see realised.

First of all, the system should incorporate existing computational models related to expression in music. These should share the same data structures so that they can be evaluated and compared. Combining these models should also be possible.

We are interested in studying a number of issues in expressive performance. For instance, how do the magnitudes, that are used in the different expressive parameters, behave in time and at different tempi, and how do they relate to the musical structure (Clarke 1988)? Because a listener cannot detect all the subtle features of a performance, we need some help. We envisage the possibility of "zooming-in" at different structural levels of a musical performance (e.g. examining the expressive timing only at the level of the piece or only at phrase level), as well as looking at its various structural units (e.g. chords or grace notes) or inter-structural relations (e.g. voice leading).

Besides analysis, we would like to perform "micro surgery" on performances: change expressive detail and shape of structural units, or, in other words, generate modified performances that have been transformed or depending on particular structural units. To give some examples: we would like to exaggerate the timing of chords without changing their spread (or the reverse), change the tempo of a piece without altering the timing of grace notes and trills, modify the timing of the melody without changing the timing of the accompaniment, remove all expressive timing except on beat level, scale specific structural elements of a performance using different magnitudes, or make a solo voice lead with respect to the rest of the music. The results of these adjustments (i.e. modified performances) could then be used in experiments where listeners have to judge the modified performance on the basis of their perceptual effectiveness.

In order to make expression in a performance a score is essential. When scores are not available (in the case of e.g. improvisations) we are helped by an automated score generator.

ICMC GLASGOW 1990 PROCEEDINGS

304
When scores are available, computer assistance is indispensable in mutually adjusting the performance and the score (e.g. taking care of performance errors, order of notes within chords, ornaments in the performance etc.), since we have to compare them on a note-to-note basis. It should also assist in transferring structural information from the score to the performance (e.g. left and right hand parts), instead of having to arrange each new performance.

Of course the possibility of recording and playing back performances of different types of instruments is an important requirement, next to having access to libraries of (expert) performances and scores, and employing graphical and textual features in editing the musical and structural information.

All the means described should be embedded in a programming environment in order to gain maximum flexibility and extendibility. The environment should support version management (keep track of different versions of data and how it was created), assistance in repetitive work (e.g. when doing the same analysis over all the data of an experiment), and the automatic generation of documentation about the system. These are just a few demands on system support.

Finally, both first-time and advanced users should feel comfortable working with the system. First time users should be able to make use of menus and dialogues, and have explanatory information on the actions that are performed. More advanced users probably want to bypass the menus and dialogues using a programmed way of manipulation. The user interface should be multi-modal, both simple and flexible, and it should be easy for advanced users to extend the environment and have their programs well integrated.

We will try to give shape to this hodgepodge of dreams and visions in the following paragraphs. The described "ideal" system is simplified into a conceptual description in Figure 1.

**DESIGN**

Earlier work on composition systems (Deakin & Honing 1988) gave us enough confidence in the importance of building POCO by using a workbench approach: a collection of tools that can be combined in a flexible way. This resulted in an architecture that embodies a relatively empty shell consisting of a closed data representation at one end and the user interface at the other. In between there is a layer of commands (or transformations) that is extensible. Communication with the outside world (e.g. sequencers and statistical packages) is supported by an I/O layer and is extensible as well (e.g. when a new medium is added or a new format is needed). This architecture is shown in Figure 2.

In the remaining half of the article we will describe this architecture layer by layer.
Communication with the outside world is implemented as transparent as possible and is modelled as streams, a combination of a medium (e.g., a file, a window, a MIDI-port) and its associated (io-)types. The system provides different (io-)types (e.g., music-text-files, standard MIDI files). All information generated by the system is encoded in the specific format or protocol used, so there is always completeness of information. A new medium and its (io-)type(s) can easily be added by providing a set of read and write functions.

We support the MIDI standard to be able to use commercial software for capturing and playback, facilitating the exchange of performances and scores between systems, and making use of the growing range of MIDI based instruments and interfaces. The format was extended to sustain completeness of information. Within the system the musical information is encoded into a more general data representation.

DATA REPRESENTATION
A consistent and flexible representation of musical objects within the environment is essential because all operations take place on this representation.

Musical Objects

There are two kinds of basic objects in this representation: time-points and time-intervals. Time-intervals are note, rest, and segment (denoting structure). Time-points are midi (e.g., MIDI controller information), comments (for representing comments and other timed textual information), and begin-of-stream and end-of-stream (to model上下, to calculate the length of a piece, to cut sections out of performances, to merge and concatenate them etc.).

Structure

One of the main deficiencies of low level representations of music (e.g., MIDI files, note lists) is the absence of structural descriptions. In our representation we use a simple and flexible way of representing structure called segmentation or collections. The basic musical objects can be grouped using a general 'part-of' relation to build hierarchical, horizontal, vertical, associative or even mutual ambiguous structural units. This representation proved to suffice in rebuilding wildly different models. Each unit is named to be able to provide a hook onto which any other knowledge (outside the definition of the musical representation) can be attached. When constructing a complete model of expressive timing, information is needed from a harmonic or metrical nature. Although it is tempting to incorporate musical knowledge, as done in most AI approaches to modelling of musical knowledge, it specializes the model and makes it less modular. With structural annotation there is no need to incorporate all this domain (i.e. style) specific information in the system, because it can be communicated through a layer of structural information (see also Honing 1990).

USER INTERFACE

POCO is implemented in Allegro Common Lisp making use of program generators. They facilitate the easy integration of user code. When a new command is added to the system, it automatically propagates information to the right menu's and dialogs and provides information for the automatic documentation generator (a facility that is almost indispensable in a larger system).

The user interface supports multiple modes of communication, consisting of menu's and dialogs, Lisp program equivalents, and natural language descriptions (see figure 3). The system keeps a history of all actions that took place. They are available as normal Lisp expressions that can be re-evaluated and edited. Data files generated by the system contain information describing what transformations were used and their parameters (i.e. the LISP expression that generated it).

TRANSFORMATIONS

Transformations are a (still growing) collection of tools that generate new or modified musical information. There is a matcher for comparing, cleaning-up, and mutually adjusting scores and performances, a filter system (using a general pattern language) to retrieve special information (e.g. all notes that are part of a chord, the whole piece except the ornaments, all notes in the left hand of a piano performance in the second phrase etc.), tools that allow scaling of timing, ars-imitation and dynamics of musical objects (e.g. amplifying, translating or inverting the expressive timing profiles), and transformations to merge or concatenate performances or scores.

ICMC GLASGOW 1990 PROCEEDINGS
366
Another set of tools embodies some well-known models of expression. Longuet-Higgins' mental parser (1987), Todd model of rubato (1989), the Sundberg (1989) expression generating rule system ('van Oosten 1990), and the Desain and Hornig connectionist quantizer (1989) are examples of transformations that are available.

A typical path
To give an idea of both the complexity of an expressive transformation, which might seem simple at first sight, and the support given by the system in the realisation of such transformation, we will describe a typical path from an original piano performance to a new version with a modified expressive timing profile depending on the musical structure.

To be able to look at the expressive timing of the performance we need a score. Either we use a score available in one of the other standard formats or we can make a new one from a recorded performance using one of the quantizers (Desain & Hornig 1988; Longuet-Higgins 1987) resulting in a first version of the score. Then we probably need to do some editing of the score, for instance, add (more) structural information, correct errors etc. This can be done by using the editors outside the system, after converting the score to a convenient format. But before we can do any transformation we should be matched to the performance under examination (removing errors in the performance, altering order of notes within chords etc.). All non-note (e.g. text, comments) and structural information annotated in the score is merged into the performance and vice versa. The result is a matched performance and score, both with all the available structural information. These form the basic input to our transformations. We can now, for instance, exaggerate the timing of the bars, without changing the timing of the other structural units (e.g. chords and phrases). The modified performance is written to an external file that can be played by a sequencer.

The environment offers different kinds of help that makes traveling along this path, with all its intermediate steps, easier and repeatable (see User Interface).

ANALYSES
Analysis is a category of transformation that generates statistical data (instead of musical information). The analyses comprises special analytical methods that provide the user with visual or numerical information. It will be written to a dedicated medium (e.g file, window) and can be used by programs outside the system (e.g. statistical packages), were additional analysis can be done, graphs can be plotted etc.

Examples of analyses provided are the use of autocorrelation in analysing expressive timing (Desain and De Vos 1990), analysis that produce tables of timing data related to structure that facilitate the study of e.g. voice-leading and chord timing. These are among other more straightforward analyses.

CONCLUSION
Although we, of course, didn't succeed in realising all the dreams of a perfect system, we provided a sound basis for further development. We do think we have made the right decisions in what should be inside and what should be outside the system. The possibility to use structure in examining and manipulating expression proved to be very powerful. The facilities described in the User Interface turned out too to be just luxurious, but improved the usability and maintainability of the system.

POCO is currently used by the institutes involved in the project, but is still in development. We now work on stabilizing the system. A version for distribution is still not available.

Our own use of the system is directed towards understanding the relation between expression and structure, hopefully resulting in more insights of how to model expression in music. This would enable us to design editors that can manipulate musical information in a more psychologically and perceptually relevant way. In the end we hope to contribute to the design of composition and interactive computer systems in need of models for the production and perception of musical performance.

ACKNOWLEDGMENTS
Both design and realization of POCO were done in collaboration with Peter Desain. The research was done together with Eric Clarke at City University, London and was made possible by an ESPRIT grant under number A412354004.

Thanks to Steve McAdams for fruitful discussions during the development of the system. Also thanks to Peter van Oosten, Klaus de Rijck, Jeroen Sotul, Siebe de Vos, and our other colleagues at

ICMC GLASGOW 1990 PROCEEDINGS
REFERENCES

ICMC GLASGOW 1990 PROCEEDINGS
308