CONFERENCE 1997

INTERNATIONAL COMPUTER MUSIC PROCEEDINGS

ICMC '97
VIDI- A Voice Instrument Digital Interface for Byzantine Music

Dionysios V. Politis, Ioannis A. Tsoukalas, Panagiotis Linardis (1)
Alexandros Bakalakos (2)
(1) Department of Computer Science, Aristotle University of Thessaloniki
dpolitis@athena.auth.gr, http://num.csd.auth.gr/
(2) Program of Psychoacoustics, Aristotle University of Thessaloniki
alex@alexandros.csd.auth.gr, http://alexandros.csd.auth.gr/~alex/

Abstract

VIDI-BM is a project concerning the creation of a modular integrated environment for the reproduction of traditional songs or ecclesiastical hymns written and performed in Byzantine Music notation. The environment uses multi-platform client-server architecture. The client consists of a visual editor and a compiler running under Windows/Windows NT for the composition of the musical scores with the lyrics in Greek and English. Via the Winsock and server TCP/IP sockets the interpreted MIDI-like packets are directed to the UNIX server, which is programmed to perform by employing a singer for a real time execution of the scores. Apart from the intrinsic Byzantine Music notation, the client/server environment copes with matters like the scales of East and the inherent prosodic way with glissato coarticulation of oriental tunes by using a general-purpose Voice Interface designed according to the ADPCM Δ-modulation for use by digital equipment.

1 Introduction: the semantics

The exact performance of synthetic ‘singers’ for traditional Greek songs and ecclesiastical Hymns has to cope with two major issues: the one has to do with the scales (where one finds the successors of the ancient Greek Dorian, Lydian, Phrygian, Mixolydian Modes and their plagal ones HypoDorian, Hypolydian etc) and the other with the notation.

The Byzantine Music (BM) notation, known as Parasilinikioi [1][2][5], does not mark notes explicitly, but as increment or decrement from the previous state while the same time transitional patterns state the qualitative way for the ascent or descent of the prosodic pitch. The Byzantine Music paradigm has been thoroughly examined by Spyridis and Politis as a stochastic process by using Hidden Markov Chains, and the result of the analysis has been the composition of BM hymnlike melodies by a mainframe computer [3]. This analysis methodology, however, cannot proceed with deciphering the melodic or prosodic information, since it copes with the notes of parasilinikioi as mere symbols, and with its melodies as sequences of symbols.

The musical content of BM was interpreted by Politis et al. when the singer-parasilinikioi relation was conceived as a Δ-modulation decoder scheme [4]. According to that scheme, the sequence of the parasilinikioi symbols

\[ \cdots \, \text{E} \, \text{E} \, \text{EN} \, \text{E} \, \cdots \]  \hspace{1cm} (1)

is interpreted in terms of notes, frequencies and eichomoria (the corresponding division of commas), as shown in fig. 1.

In order to interpret sequence (1) equation (2) was used

\[ x(n) = \left[ x(n-3) + \text{sign}(x(n-2)) \Delta(x(n-2)) + \alpha \right] \cdot D(x_{n-2} \ldots x_{n-1}) \]  \hspace{1cm} (2)

which interprets the delta action of each symbol \( x(n) \) on the implied note \( x(n-3) \) from the previously received sequence of symbols \([x(0), x(1), \ldots, x(n-3)]\). As one can see at fig. 1 and (2), there is a time lag corresponding to the duration of two notes, since there are time operators like - (gogion), which have a retro effective multiplication action \( D(x_{n-2}, \ldots, x_{n+1}) \) ranging from the previous two notes up to the forthcoming one in comparison with the presently received note \( x(n) \). In the

---

**Fig. 1.** Information deciphered from a BM melodic sequence in Mode 1 (Dorian) in terms of notes, eichomoria, timing and absolute frequencies for the middle octave.
case of gorgon  \( D(x_{n-3}, \ldots, x_{n+1}) = (_,*5,*5,_,) \),
meaning that when this timing operator is received with
\( x(n) \), the duration of \( x(n-1) \) and \( x(n) \) is altered by
multiplying the already deduced duration of the
deciphered notes at the present state and the previous
one.

Some more complex timing operators are
\[
\begin{align*}
\text{①} & \quad (\_\_\_,*5,_,) \\
\text{②} & \quad (*25,*,25,*5,_) \\
\end{align*}
\]
(3) (4)

where the factor in bold corresponds to the present state
\( x(n) \).

In order to interpret sequence (1) an encoding scheme is
needed for feeding in computers with a meaningful
ASCII or binary input; the solution is not possible by
merely assigning short integers to specific notes, as is
the case of MIDI, since each symbol of sequence (1)
does not correspond to a specific note. Very crucial for
the interpretation of (1) is the scale symbol \( \dagger \)
denoting the quantizer of Mode A (fig. 1). Had we had for
instance ①, this would denote the scale of Mode Plagal
II (Hypo-Lydian), and thus sequence (1) would an
entirely different harmonic structure at figure 1.

For the encoding, the method of Politis et al. is used [4],
which would transcribe (1) to the ASCII sequence of
\[
\begin{align*}
00 & 0100 00 00 00 \\
01 & 0000 00 00 00 \\
01 & 0000 00 00 00 \\
04 & 0000 00 00 00 \\
01 & 0100 00 00 00 \\
01 & 0000 00 00 00 \\
01 & 0000 00 00 00 \\
01 & 0000 00 00 00 \\
01 & 0010 00 00 00 \\
80 & 0010 00 00 00 \\
32 & 0100 00 00 00 \\
01 & 0100 00 00 00 \\
\ldots
\end{align*}
\]
(5)

What is important with sequence (5) is not the numerics
themselves, which are chosen deliberately to have
backward compatibility with earlier methodologies, like
that of Spyridis-Politis, but their positional
combinations, which enable the description of more than
10^2 combinations of the basic symbols of parasimanitiki
[4].

For the interpretation of the yielded sequences, a musical
compiler is devised which will produce strings
containing all the information seen at fig. 1.

In order to transcribe or compose BM songs, hymns and
melodies, a visual editor is made consolidating the
symbols of parasimanitiki. This program allows one to
compose a BM melody, even not knowing the positions
and rules of many notational elements.

The outcome of the editor feeds the compiler, and the
deciphered sequence of strings is produced. Via the
TCP/IP sockets communication, the strings are
transmitted to the real-time server, who incarnates the
synthetic ‘singer’ using Cook’s Synthesis Toolkit [6],
enriched with prosodic elements of the Greek language.

2 The client interface

At client level, the Windows 95/NT 32-bit application
platform was chosen for maximum compatibility; the
code was deployed using C, C++ and Visual Basic
avoiding non-portable routines as possible. For the
TCP/IP communication part, Winsock 1.1 was used [7].

2.1 The visual editor

An experimental visual editor was built by using Visual
Basic and employing drag-and-drop techniques [8].

The editor has a two level entry area, one for the
notational symbols and the other for the lyrics in Greek
or in English. The lyrics are quantized at morpheme
level; when entering a combination of letters not

![Fig. 2. The BM visual composer used at client level.](image-url)
reproduce a BM manuscript or compose a melody of his own. If a non-admissible combination is composed, the whole structure can be revoked.

2.2 The compiler

Once the melody is written, it is saved and the compiler is invoked. The compiler performs a second level parsing (the first assumed to be performed at the editor and its built-in expert) and confirms the musical correctness of the inserted symbols.

After that, it considers that the sequence read corresponds to a reception channel of an ADPCM Δ-decoder, with a time lag of two cycles (fig. 3).

Among the various algorithms applying the Δ-modulation decoding scheme, the one implementing the FIFO buffer is dominant. Its functionality is shown with an example: Assuming that we have the sequence ΔΔ Δ Δ, which corresponds to the encoded tuplet

\[
\begin{align*}
31 & 0100 00 00 04 \\
31 & 1000 00 00 00
\end{align*}
\]

(6) (7)

the buffer is loaded with (6) as seen in fig. 4.

The time operator 1000 is analyzed as \( D(1000) \rightarrow ([*0.5,*0.5,\ldots]) \). The previous steps are repeated and the Δ-action queue and duration queue are updated; the retro-effective action of gorgon changes once more the time span; While this rotation takes place, the first deciphered string of (6) \([\Delta = -1, T = 1, \ldots]\) can be transmitted to the real time server with a time lag of 2 slots.
2.3 The client/server network

The information deciphered is transmitted to the 'singer' at the server using Winsock 1.1, which is an implementation of the BSD Unix sockets on the TCP and IP layers.

On the server side, a socket is created, it is binded on a local address and listens for connection on a predefined port. When the client attempts to connect to the port, the server accepts the connection and creates (or, better say, spawns) another process to service the client. Then, this new child process talks directly to the server, using a predefined protocol.

While Unix feels at home working with processes and sockets, on the client side things are more complicated, largely because the original Winsock implementation was based on the single-tasking, event-driven platform of the earlier Windows 3.1. While Windows 95 have implemented multi-threading with Winsock version 2.0, for compatibility reasons we use version 1.1.

Still, the idea is the same: get local and remote IP addresses, create a new socket, bind that socket to the local port and address, and establish messaging handling for the transmission of the ASCII string coming out of the compiler decoder-compiler.

4 The Real Time Server

The server side is configured so that it accepts line after line of input data in the format specified above via a socket channel of type SOCK_STREAM. Once a connection is initiated data is passed at ethernet speeds which are sufficient for the real time implementation. The data is then fed to the SKINI input handler of the Synthesis Toolkit [6]. It is then transliterated to SKINI (Synthesis toolKit Instrument Network Interface) format to subsequently drive the 'singer' engine. A More detailed explanation of the SKINI format can be found at [6].

5 The Singer

The singer module is comprised of a main control program which accepts the SKINI code, parses it and feeds it to the main synthesizer routine. This routine uses a physical modeling approach [6,10] to synthesize the corresponding phrases using the data coming from the server and a set of glottis and shape files created by the SPASM software [10]. The glottis and shape files are created beforehand in such a way that they preserve the articulation and phonation parameters of the greek language. Additional sets of those files were created extending the work of the IGDIS project [9]. The control parameters of the physical model synthesizer routine are carefully chosen to reflect the case of a male byzantine speaker/singer by extending the length of the vocal tract since the original model was that of female singer/speaker. The whole system behaves in real time writing the output directly to the audio port of the SGI server.

6 Conclusions

The vast and largely unknown world of Byzantine Music is deciphered by using an ADPCM Δ-modulation encoding-decoding scheme. Upon this methodology a real-time synthetic 'singer' is established and demonstrated using an O2 SGI server and a Windows client. For the user of the system a visual editor is employed where one can write BM melodies, compile them and send via the TCP/IP network the musical information to the performer. The 'singing' is the final stage of the system, and it is achieved using the Synthesis Toolkit in C++ by Perry Cook, and more specifically the module that does physical modeling synthesis of the singing voice [6]. The whole system is tuned for performing Greek Traditional Songs and BM Hymns by using the appropriate scales, especially the chromatic ones, and by performing Greek phonemes, as they have been estimated by five subjects, all holding a diploma in BM studies.

7 Acknowledgments

The authors would like to acknowledge the help of Stephen Arnold and Allen Strange. The model sounds have been estimated from the voices of N. Lefkopoulus, D. Papageorgiou, N. Kougiou, D. Veltsios and S. Triantafyllidis. We are grateful for their assistance. The authors would also like to acknowledge the support and help of Perry R. Cook throughout this project.
References


