MARS Applications Using
APPLI20 Development Tools:
a Case of Study

Fabio Armani
ANDrea Paladini
Claudio Rosati
IRIS s.r.l.
Parco La Selva, 151
03018 Ponzano (FR), Italy
fax: +39 (775) 533345 - tel: +39 (775) 533441
E-mail: mc28642@unilink.it

Abstract
MARS is an integrated environment in which a graphical user interface, a realtime operating system, and two general purpose digital signal processors are linked together in a interactive and realtime sound processing system.

The high level approach provided by the existing MARS development environment (EDIT20) could not satisfy all of MARS possible users. In particular, application programmers have more specific needs than those which EDIT20 can satisfy.

In order to aid software development on MARS, IRIS offers to the application programmer, APPLI20, a set of tool-libraries with a well defined Application Programmer Interface (API), available in C Language.

In APPLI20 the programmer finds tools to fully program and control the sound generation board, and a toolkit of graphic objects for building user interfaces in a uniform style, while hiding implementation details and reducing development time.

At this time, IRIS has developed applications in music, education and science fields.

The Linear Prediction Environment (LPE) application covers all aspects of application programming, from the development and control of DSP algorithms, to the programming of highly efficient and user-friendly graphical interfaces. For these reasons LPE is used in this paper as a model of good application development within MARS.

1 The APPLI20 tool-libraries

APPLI20 is a tool developed in IRIS to support MARS application programming on the host computer [Andreacci et al. 1993]. It offers a programming approach to the MARS system through a set of libraries and interfaces that allow the development of MARS applications in order to satisfy a set of general requirements. An application should:

- be an autonomous program;
- control the sound generation board without using EDIT20;
- exchange data and cooperate with EDIT20 and other applications;
- optimize the use of MARS resources, e.g. X20 microprograms and memories, when EDIT20 is not able to do it;
- be coded by means of high-level standard programming languages;
- have a uniform style in coding and presentation;
- be easily ported across different platforms;
- reuse the code previously developed and tested;
- have an easily and quickly produced user interface;

1.1 APPLI20 architecture

APPLI20 is the kernel of the EDIT20 architecture and consists of a set of three libraries (see figure 1)
that are in a hierarchical relation (SSERVER, 
MSERVER, and TSERVER), and of a graphics 
toolkit that offers graphical objects in the EIT20 
style.

![Figure 1: APPL120 architecture](image)

**TSERVER**

The Transmission Server implements access to the 
MARS DSP resources of the lower MARS data 
structure level, hiding from user the details of data 
communication between host and board. This level 
is related to the microprogram, registers, data 
memories and sample memories of the two X20 
DSPs. It also contains the virtual memory that maps 
the data memories of the two X20 chips to a single 
virtual space.

The media for transmission can be chosen at run 
time between MIDI and parallel. It is also possible 
to save on a file the "log" of a session for debugging 
or archiving purposes.

A summary of the functions offered by the 
TSERVER includes:

- initialization and test of the communication 
  media;
- read and write of X30 registers (to perform ON. 
  GPF, ...);
- initialization of X20 microprograms;
- direct read and write of algorithm parameters 
  from to X20 data memory locations, using 
  physical or virtual addresses;
- load and dump of samples;
- pack and unpack of MARS messages.

Data memory locations are shared with MARS 
realtime operating system which updates parameters 
in response to MIDI messages, it is the user’s 
responsibility to avoid conflicts with its application.

**MSERVER**

The Microprogram Server offers the programmer a 
set of objects and functions to build an Orchestra 
working environment, which offers a structured 
point of view of the DSP layer. The Orchestra is a 
collection of algorithms described in terms of:

- number of voices (clones),
- entry points (ravelopes, parameters, LFOs),
- audio routing.

Each algorithm is composed of a collection of 
cooperating and interconnected objects: the 
modules. A module is the "atomic" object, and 
implements a portion of the microprogram code, 
allocating the necessary data memory locations for 
its entry points (used for module interconnection 
and data entry).

The implemented set of functions includes:

- real-time creation and edging of algorithm 
  modules (Add, Delete, Connect, ...);
- creation and editing of the Orchestra’s 
environment as algorithm clones and their audio 
  routing;
- transmission to the board of the X20’s 
  microprograms and data memory configurations 
  using TSERVER;
- access to the Orchestra’s data files in the 
  EIT20 format.

![Figure 2: MARS data structure](image)

ICMC Proceedings 1994  
Music Workstations
SSERVER

The Structure Server extends the MSERVER objects and functions in order to allow the definition of a complete MIDI Performance Environment.

It offers an MSERVER-style approach to defining an Orchestra, adding special information for the MARS music control. All this hides the use of MSERVER details of realtime requirements, and details of ISDN aspects.

Moreover, it adds objects (dynamic and static parameters, LFOs, envelopes, data tables, tenses) and functions (Create, Delete, Edit) to give values and MIDI control to an algorithm's entry points, to create libraries of tenses and to link tenses to MIDI channels.

SSERVER creates on the host an image of the three-level MARS data structure (see figure 2) for a complete MIDI Performance Environment [Andreatacci et al. 1993].

This image can be transmitted to the MARS board by means of high level object-specific functions which hide the use of TSERVER and the details of data protocol and I/O access. Furthermore, SSERVER offers special interaction modes to automatically transmit the image when modified.

The image is in EDIT20 compatible format. It can be saved on the host and retrieved for subsequent runs. This means that an image can be exchanged between EDIT20 and other SSERVER applications.

The higher level defines the MIDI Performance Environment, based on the Orchestra. It consists of tenses defined over Orchestra's algorithms, and of a Channel Map that links tenses to MIDI channels. This environment is supported by the embedded realtime MIDI management (voice and tones allocation policy, events triggering, ...).

A tense is a complete configuration of an algorithm's variables. It assigns specific values and MIDI controls to the algorithm parameters. These are described in the tense:

- envelope definitions,
- values of static parameters,
- dynamic parameters' control structures and tables,
- LFO definitions.

Graphics Toolkit

The APPLI20 graphics toolkit is a set of libraries originally created to develop a MARS application in a graphical EDIT20 style. However, it has no links with the other libraries (TSERVER, MSERVER and SSERVER), and can be used to develop non-MARS graphical applications. Thus it was extensively used at IRIS to develop graphics simulators, editors and MIDI applications.

The toolkit allows the programmer to think in terms of graphics objects with callback mechanisms. The objects it manages are windows, line detectors, sliders, buttons, LEOs, editable text fields, and graphics areas (see figure 3).

![Figure 3: Graphic toolkit objects](image)

The toolkit's current implementation is based on the Atari GEM Operating System, using its AES (Application Environment Services) and VDI (Virtual Device Interface) functions to manage the user interaction with the objects in an event-driven fashion.

In order to allow MARS applications to easily migrate to non-Atari platforms, a great effort is currently being made at IRIS to evolve the toolkit to a portable version applications [Maggi and Prestiaggiaco 1993]. This is being accomplished using multi-platform libraries (Liant, XVT toolkit) to reimplement the low-level layer of the APPLI20 graphics toolkit applications [Rosati 1994].

2 Linear Prediction Environment

In the following paragraphs we will describe a complex application called Linear Prediction Environment (LPE), which has been developed with the APPLI20 toolkit. The LPE program performs voice encoding/decoding and processing and covers all aspects of application programming.

2.1 Backgrounds

Generally, we use a model to produce a synthetic voice (see figure 4) where the vocal emission is divided into two independent stages:

Excitation The pulse train produced by the vocal folds is considered in the synthesis of voiced sound. Otherwise, in the case of voiceless consonants where the vocal folds do not take part, a noise source is used. The voice pitch depends exclusively by this excitation element.
Resonance Here we consider the filtering effect that the oral cavity makes on the sound produced by the excitation. The spectral envelope of emitted sound, which causes the difference among phonemes, depends only by this element.

The considered division in two stages derives from the fact that the oral cavity movements are slower than the vibratory motion of the vocal folds and that the two phenomena, in first approximation, do not interact with each other and can be analyzed separately.

![Figure 4: Vocal tract model](image)

A feasible simulation of the vocal tract can be obtained by sampling the cavity itself like a set of N tubes in cascade. The reciprocal width of the tubes, at a given moment, determines the cavity resonances, called formants, which characterized the produced sound.

It is possible, therefore, to draw at a given instant the cavity by means of N reflection coefficients \( k_i \) \( (1 \leq i \leq N) \) which cause, depending by tube areas, the percentage of the transmitted and reflected waves in the crossing between two tubes.

Due to its mechanical inertia the vocal tract changes slowly and therefore the produced signals can be considered stationary in a time frame of 10/50 ms. So it is sufficient to use a single parameter set every frame and update it every 10/20 ms. This time between two consecutive frames is called displacement.

At least two parameters are necessary to describe the excitation phase:

- a voiced/unvoiced flag (VUV), which allows to choose between noise or pulse train
- and, in this last case, the frequency of this pulse train.

![Figure 5: LPC analysis](image)

The analysis procedure consists of dividing the sampled vocal signal in consecutive frames of 10/50 ms length (see figure 5). For each signal i-th frame it is computed the k(i) reflection coefficients set \( \{1 \leq i \leq N\} \) using an analytic method called linear prediction code (LPC) [Rabiner and Schafer 1978, Cook 1990]. In this method the tube set is described by means of a lattice filter and the filter coefficients are calculated in order to minimize the mean square error between the speech sample and the linearly predicted ones.

A second analysis is required to decide if a frame must be considered voiced or unvoiced (VUV flag) and, in this case, to compute the value of the emitted frequency (pitch).

In our application the acquisition algorithm under-samples the signal at 20kHz sampling rate. The LPC analysis program performs a 25th order LPC analysis using the Durbin method on approximately 25 ms frames displaced by 15 ms while the pitch analysis uses the Gold-Rabiner parallel algorithm [Gold and Rabiner 1969].
After the calculation of these parameters it is possible to obtain a voice resynthesis where the emitted frequency (therefore the melodic line in the singing and the prosody in the speech) is controlled independently from the articulation of the vocal tract.

2.2 Application description

The Linear Prediction Environment (LPE) program was developed using the APPL20 tool library, and runs on an ATARI personal computer linked to a MARS board.

It is a real-time environment that allows numerous and interesting musical and pedagogical applications. For example it is possible to resynthesize a singing voice by synchronizing it to an external time source or redefining the singing melody without altering its naturalness.

The host computer has the interface functions of event control and data displaying while it performs the LPC and pitch analysis. MARS is configured with a two-algorithm orchestra. The first algorithm samples and stores the voice signal while the second one performs the LPC synthesis. This orchestra has a tonemap containing the timbres required to perform the LPC synthesis. These timbres can be controlled using standard MIDI devices or by the LPE application interface HotF.

![Figure 6: Pitch shape and VUV flags](image)

The steps allowed by the LPE program are the following:

**Initialization** The described orchestra is sent to MARS.

**Acquisition** The program starts the recording on MARS in auto trigger mode and stores the signal. The user can control the stop of this recording and may hear the sampled signal.

**Analysis** This signal is transmitted to the host in order to perform the analysis. The program first displays the extracted pitch shape and the VUV flag (see figure 6). Then it computes, frame by frame, the 26 LPC prediction coefficients displaying the corresponding tube set (see figure 7). Every frame the program sends these 26+2 coefficients to the MARS which plays the displayed configuration for an acoustic check.

**Synthesis** The whole LPC coefficient set is sent to the MARS sample memory. The synthesis algorithm reads each coefficient with an interpolated oscillator from its sample memory. It uses a 25-order IIR filter with doubled delay taps to achieve a synthetic speech at 40KHz sampling rate. The excitation source can be chosen among the usual VUV oscillator, all noise or an external signal from ADC. The pulse train is produced by a bandwidth controlled oscillator reading a wavetable. The frequency of this oscillator can be taken from the preanalyzed pitch or from a MIDI controller.

**File operation** The user can save the LPC coefficients in a binary file loadable from the EDIT20 environment in order to use the LPC orchestra in a standard EDIT20 session. The application allows also to load and play a pre-stored analysis file.

![Figure 7: LPC prediction coefficients and tube set](image)

During voice synthesis, the user can control all the algorithm parameters. Among these the most interesting are:

**Excitation type** The user can choose the function red by the VUV oscillator using an impulsive similar to that produced by the vocal folds in order to obtain a realistic voice or using any periodic waveform. It is also possible to switch from the VUV oscillator to a noise source, to obtain the same effect we have by speaking without the vocal folds, or to the ADC to modulate with the vocal tract, for example, the sound of an orchestra.

**Frame velocity** A forward or backward sentence flowing can be controlled varying the frequency of the oscillation which needs the synthesis parameters. Using the LPC synthesis in fact the pitch and the spectral shape do not depend by the velocity at which the tubes describing the vocal tract are moved.

Music/Workstations 234

ICMC Proceedings 1994
3 Conclusions and Further Developments

APPL20 has proven to be a powerful tool to easily and quickly develop complex applications, like the LPE program.

Currently IRIS is working on the multi-platform porting of the MARS developing environment and, of course, of the APPL20 tools and applications [Maggi and Prestigiacomo 1993]. The Macintosh and Windows versions of the APPL20 libraries are already available in a beta version, and allow the development of MARS applications using the native Graphical User Interface (GUI) system.

Meanwhile the LPE program is under redevelopment on both Macintosh and Windows using the Lant’s Cviews 3.0 portable software development kit [Rosati 1994], because it offers a single Application Programming Interface (API) on both operating system and underlying GUI.

Further improvements of the LPE program functions are suggested by musical and scientific requirements and include the possibility of frame-by-frame graphical editing of:

- pitch shape on the basis of the reference computed by the program.
- amplitude envelope to obtain a control on the dynamic of a sentence or of a melody.

Deeper editing could be performed also by stopping the program during the LPE analysis in order to:

- change by hand the time and pitch diameter with an acoustical feedback of the obtained results.
- save the LFC coefficients and the data related to the first five formants of the current vocal tract configuration, computed from the LFC data [McCandless 1974] in order to allow the use of this analysis for a singing voice formant synthesis algorithm [Sandberg 1987].
- interpolate between two previously saved configurations with the possibility of choosing among different interpolation methods.

References


Department of Music, Stanford University, Stanford, California, 1990.


