Javelina: An Environment for the Development of Software for Digital Signal Processing
Kurt J. Hebel
CERL Music Project
252 Engineering Research Laboratory
103 South Mathews
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801-2977 U.S.A.
(217)-333-6766

Abstract
We are currently entering a new phase in the technology of computer music production. The newest generation of digital signal processors has the potential to perform real-time software synthesis, so far, however, approaches to the use of these systems span from patch editing to assembly-language programming.

This paper describes Javelina, a software environment for the development of software for digital signal processors. This environment translates discrete-time systems specified using mathematical notation into algorithms; the algorithms are then translated into the machine language of any of several digital signal processors. Javelina includes facilities for manipulating and measuring the discrete-time system throughout the development process.

It is proposed that Javelina could be used for the development of "unit generators" in a larger, machine-independent real-time software synthesis system.

I. Introduction
As is the case with general purpose computers, the computing power of digital signal processing hardware has been increasing exponentially, while the power of the software has only been increasing linearly. Increasingly more sophisticated digital signal processors are being built for use in real-time computer music systems; the development and management of the software for these processors is quickly becoming the major bottleneck in system development.

In this paper, I will describe Javelina, a software system which can automatically generate programs for digital signal processors from a mathematical specification. This system assists in the discovery of alternate representations for the mathematical specification, and produces software which is optimized for a particular signal processor. In addition, the system provides various performance measures to indicate the success of a particular implementation at any point in the process of converting the mathematical specification to an actual program.

II. Previous Research
Javelina brings together many different research areas in mathematics, computer science and signal processing, including symbolic (or algebraic) manipulation and computation, automatic programming, and the design, implementation and evaluation of digital signal processing systems.

In his dissertation (Kopec80), Gary Kopec presents the first work in the area of symbolic representation of discrete-time (DT) signals and systems in a computer program. He introduces symbolic digital signal processing into computer programming through the use of abstract signal types. An abstract signal type describes the behavior of a signal (or system) through its interaction with the "outside world"; that is, any signal should be able to respond to basic requests for the next sample in its stream, its bandwidth, or other common properties. Thus, all signals interact uniformly with signal processing programs, even though the internal details of implementation may change drastically between signals of differing types.

Cory Meyers extends the work of Kopec with his E-SPICE system (Meyers86). E-SPICE could perform symbolic manipulation in both the discrete-time and the continuous-frequency domains, it was able to determine a "good" alternate implementation of a DT system through symbolic manipulation and performance measurement.

[Ben85], [Cow87], and [Zib87] each present software systems which can compile a block diagram representation of a DT system into a program. A program segment must be written in advance for each
Figure 1. Block diagram of the Javelins system.

III. Overview

A block diagram for Javelins is given in Figure 1. The various components of the system are described in the paragraphs which follow.

Specification of a Discrete-Time System

The DT system can be specified using mathematical notation or through the use of special purpose editors (for example, a filter design package). Any known constraint on portions of the DT system can be entered for later use in determining alternate realizations of the given system. Figure 2 shows a filter example; a filter design program was used to generate the pole locations and the direct form of the difference equation.

Figure 2. Filter design example.

Algebraic Manipulation

The DT system can be manipulated through applications of algebraic operations to the representation stored in the computer. These manipulations might alter the form of the system entirely or merely result in mathematical simplifications. Constraints placed on the system can be used to help determine which manipulations should be allowed. For example, knowledge about the symmetry of an impulse response could be used to rewrite the convolution sum of a digital filter. An alternate form, derived from algebraic manipulation of the example filter, is shown in the lower part of Figure 2.

Abstract Computations

The algebraic specification of the DT system can be transformed into a network of computations. This network specifies, in a machine independent way, the exact sequence of computations (or operations) needed to implement the DT system. There is only a partial time ordering of the computations: the true time ordering can be derived from the computational dependencies stated in the network. Figure 3 shows the computations for the alternate form of the filter example from Figure 2.

Algorithm Manipulation

The computation network can be manipulated (manually or automatically) to improve the efficiency of the implementation. At this point approximations may be introduced into the network to take into account target machine capabilities. For example, when developing an algorithm for a target machine with no capabilities for transcendental function evaluation, a table lookup might be substituted for the transcendental function evaluation. Also, arbitrary length delays can be added to the filter.
without changing its frequency response. Sometimes the addition of delays can be used to decrease the number of computations needed. Figure 4 illustrates the new computation network when a single unit delay is added to the example filter, allowing more operations to be performed in parallel.

Real-time Simulation of the System
The computation network can then be translated into a partially time ordered set of instructions or micro-operations for the target machine; this set of instructions is automatically time ordered and optimized resulting in an executable program for the target machine. When this program is executed, the originally specified DT system will be realized in real-time.

It should be noted that all machine dependent information is contained in this and the adjacent block in Figure 1; code generation for different signal processors requires changes to these blocks only.

Interactive Optimisation
The program for the target machine can be interactively optimized to develop the most efficient realization of the computation network. This work is necessary since the computer has already tried all of the obvious "mechanical" optimizations of resource allocation; any further optimizations will require "clever tricks" on the part of a human being. For example, a linear function of time (requiring both a multiplication and an addition) can be replaced with a simple accumulating sum (requiring only additions). Optimization may require backtracking to higher level representations in order to perform either algebraic or algorithmic manipulations.

Performance Analysis
Every manipulation of the DT system's representation can, potentially, alter its performance. It is necessary, therefore, to be able to measure different aspects of the represented DT system in order to compare them with those of other implementations. These measurements might include various machine operation counts, noise and dynamic range values, and coefficient sensitivities.

It is never necessary nor desirable to try to come up with a complete list of performance measures; some measurement functions are predefined, and the user can add any other measures as they are needed.

IV. Javelina Implementation
The development of Javelina requires the availability of certain hardware and software technologies. In hardware, it was necessary to have general purpose digital signal processors. Here at CERL, we have developed such a signal processor called the Platypus [Hal87].

Software technology, in general, progresses very slowly. Most of the computer and computer languages are simple variants of one another; while it may be easier to do certain things in one language than in another language, it is typically not an order of magnitude difference. A relatively new software technology, object-oriented programming [Cox86], may yet supply that order of magnitude difference.

The first completely object-oriented language, Smalltalk ([Fox83][Kec83]), is embedded within a highly interactive and flexible environment which includes built-in compilers and debuggers [Go84]. In Smalltalk, everything is an object that can respond to messages. Computation can only be accomplished through message passing, and each object is responsible for the integrity of its private data. This allows the programmer to build highly modular subprograms which may be connected together into a much larger program in a variety of ways.

V. Conclusion
My experience with computers and signal processing have enabled me to develop a degree of signal processing "intuition". I hope to explore with Javelina the areas of adaptive filtering, quantization effects, and nonlinear signal processing with applications to digital audio and computer music in an effort to extend that intuition. Through exploration the Javelina user educates himself; the ability to interactively try out different signal processing algorithms and evaluate them in real time can lead the user to discover previously unnoticed abstract properties of signal processing structures and algorithms. I propose that Javelina could also be used to develop a set of machine independent "unit generators" for a real-time software synthesis system.

106 1987 ICMC Proceedings
Bibliography


