The Software Architecture of the Kyma System

Kurt J. Hebel and Carla Scaletti
Symbolic Sound Corporation
Post Office Box 2530
Champaign, IL 61825-2530 USA
Tel: (217) 355-6273
Fax: (217) 355-6562
symbolic.cc@applelink.apple.com

Abstract
The Kyma System is made up of a number of cooperating software components that provide software synthesis in real time on the Capybara, a multiprocessor digital signal processor. These components range from the signal processor operating system, through low-level host driver devices, up to high-level programs written in Smalltalk-80 and other computer languages.

Introduction
The primary goal of the development of the Kyma System is the creation of a machine-independent specification that uniformly integrates both signal-level and composition-level algorithms and processes [Scaletti and Hebel, 1991]. This specification is based on abstract entities called Sound objects (or Sounds, for short) that represent audio signals. In the current release of the Kyma System, Sounds are realized in real time on the Capybara multiprocessor DSP.

Work on the Kyma software was begun by Carla Scaletti in 1986 [Scaletti, 1987]; since then it has evolved through several generations of host computer systems and digital signal processors [Scaletti, 1992].

The Kyma System is being used by composers and researchers around the world who want to work with sound at a high level while still being able to access machine-level resources when necessary. It is also being used in conjunction with other sound-related hardware and software (see, for example, [Haken, 1992], [Calloway and Haken, 1992] and [Hornor et al., 1992]).

Overview
The main software components in the current version of Kyma include:
• Smalltalk-based Kyma Sound Specification Language
• Smalltalk-based Kyma User Interface
• Smalltalk-based Kyma Sound Compiler
• Smalltalk-based Kyma System operating on the Capybara

The Kyma System software has been factored in such a way as to allow the substitution of different hardware and/or software for each of the components. For example, the DSP driver and its high-level interface, and all of the Smalltalk-80 components, are available for either the Apple Macintosh or for Microsoft Windows platforms.

In the original version of Kyma (1986), the DSP components were implemented entirely in Smalltalk. In 1987 those components were replaced by microcode programs running on the CERL Sound Group Platypus DSP [Scaletti, 1987], and, in 1989, this was extended to run on either the Capybara or the Platypus. In the future, new software and/or hardware components will almost certainly be substituted for the current components, without disturbing the basic concepts of Kyma or its software architecture.

The first three software components are written in Motorola 68000 assembly language and reside on the Capybara; the remainder of the components reside on the host computer: the device driver is written in C, the high-level software interface has been written in C; Forth, and Smalltalk-80, and the remainder of the components are written in Smalltalk-80 (see the Figure at the end of this paper).•

DSP Operating System
The DSP Operating System (DSP OS) is distributed over the several Capybara expansion cards and the Capybara motherboard. It provides basic communications facilities between
the host computer, the various Capybara input/output devices, and the active Sound objects. The DSP OS parses MIDI input, generates MIDI output, controls the output oversampling filters, and manages the audio signals going to the DA converters, coming from the A/D converters, and going to and coming from the hard disk of the host computer.

DSP Virtual Machine

The DSP Virtual Machine (DSP VM) constitutes a simulated computer whose instructions consist of signal processing and synthesis algorithms. This makes it possible for the Kyma Sound Compiler to be ignorant of the underlying architecture of the real signal processor, increasing the portability of the Compiler (Hebel, 1991).

The DSP Virtual Machine evaluates Virtual Machine instructions generated by the Kyma Sound Compiler. These instructions, called a Schedule, are contained in the Schedule Memory, an area of static RAM on the Capybara reserved for this purpose. The VM instructions in the Schedule are executed once per sample tick, and the DSP VM is responsible for maintaining the signal connection between Sounds that are evaluated on different expansion cards within the Capybara.

Unlike many other signal processing languages, Kyma's Schedule is not static—this means that the DSP VM is only evaluating instructions corresponding to active Sound objects, allowing more efficient use of the DSP in cases where the signal computation graph changes over time. This means that the DSP VM must periodically alter the Schedule of Sounds; these time-tagged alterations are stored in a sector of Capybara dynamic RAM called the Changes Memory.

DSP Virtual Machine Instruction Set

Each instruction in the DSP Virtual Machine Instruction Set is implemented as a 50001 assembly language subroutine. These VM instructions constitute the set of primitive signal processing operations for Kyma.

Host-based Low-level DSP Device Driver

The low-level DSP device driver provides host computer services to control the Capybara. Services include loading and running programs in the Capybara, examining and changing Capybara memory, loading and running VM programs (called compiled Sounds), and samples file playback and recording.

On the Macintosh, the device driver is a Macintosh driver that is loaded by an INIT when the computer is started and is linked dynamically; in Windows, the device driver is that can be linked either at compile-time or at run-time.

Host-based High-level DSP Software Interface

The high-level DSP software interface provides an interface to the low-level DSP device driver services that is independent of both the DSP and the host platform. This layer of software hides the implementation details of the driver and DSP interface from the user, and allows higher level control of the DSP.

All of the low-level device driver services are available either directly or indirectly through this software interface. Examples of high-level services include downloading a compiled Sound file, stopping or starting the execution of the Sound, and reading and writing the Sound parameters.

Both Kyma and third party programs use this software interface to control the Capybara.

In the Smalltalk-80-based Kyma environment, this high-level interface to the DSP device driver is implemented in the HostDriverInterface class hierarchy. This class and its subclasses provide a domain-independent interface to the host operating system, providing among other things, services that control the naming of files and file types and the presentation of standard user interface elements such as menus and dialogs.

Smalltalk-based Kyma Sound Specification Language

The Kyma language itself is implemented by a class Sound and its subclasses (Scaletti, 1987). A Sound object represents a function of constant parameters and zero or more other Sound objects (Scaletti and Hebel, 1991).

Sound objects can be implemented in two ways: either as an abstraction of a Sound structure that includes variable parameters (Scaletti, 1990b), or directly as a VM instruction (the Sound Compiler includes a DSP assembler for this purpose) (Hebel, 1991).

Smalltalk-based Kyma User Interface

The Kyma user interface presents a graphical representation of the Sound objects and provides a way to directly manipulate and construct new Sound objects.
New user interfaces to Sound objects can be created without programming using the Sound Class Editor [Sealetti, 1991].

Smalltalk-based Kyma Sound Compiler

Whenever a Sound object is played within Kyma, the Kyma Sound Compiler is used to compile the Sound into a VM program. Cappara execution of this VM program can usually begin before compilation is completed on the host.

The Sound Compiler can also be directed to store the VM program in a file. This file can then be used in conjunction with the high-level DSP software interface to generate and process signals in real time.

The Sound Compiler is implemented by a set of resource management classes and by the GenericSignalProcessor class hierarchy.

There are classes that automatically manage waveable and delay-line memory, schedule samples for playback, schedule output from MIDI Sounds and MIDI files, manage the assignment of MIDI input channels to Sound objects, and allocate the virtual input/output signals that correspond to ExternalInput and ExternalOutput Sounds.

Subclasses of GenericSignalProcessor implement Sound Compilers for specific DSP architectures. Classes exist for both the single processor Platypus DSP [Sealetti, 1991a], as well as the multiprocessor Cappara.

For the Cappara, the Sound Compiler automatically manages processor allocation; this means that a complex Sound structure is automatically split up into simpler components, each of which is assigned to a Cappara expansion card with available memory and processing time.

Summary

The underlying philosophy of the Kyma System software architecture has been to provide uniform structured access to the signal processing and synthesis capabilities of DSP hardware. As much as possible, we have tried to maintain "object-orientedness" throughout the layers of software.

The development of the Kyma software has been a large project consisting of 66001 assembly language (~306 K source), C (~195 K source), and Smalltalk (~3980 K source, ~575 classes, ~6700 methods). However, it has been factored to allow for portability and evolution as hardware and software technologies improve.

References


