Microsoft DirectShow: A New Media Architecture

By Amit Chatterjee and Andrew Maltz

The desktop revolution in production and post-production has dramatically changed the way film and television programs are made, simultaneously reducing equipment costs and increasing operator efficiency. The enabling digital innovations by individual companies using standard computing platforms has come at a price—these custom implementations and closed solutions make sharing of media and hardware between applications difficult if not impossible. Microsoft's DirectShow™ Streaming Media Architecture and Windows Driver Model provide the infrastructure for today's post-production applications and hardware to truly become interoperable. This paper describes the architecture, supporting technologies, and their application in post-production scenarios.

The year 1989 marked a turning point in post-production equipment design with the introduction of desktop digital nonlinear editing systems. While IBM-compatible personal computers and Apple Macintoshes had been used in linear editing and control systems for several years prior, these new editing systems forsook analog videotape as an online storage medium and instead brought video and audio source material "inside the box" in a practical sense for the first time. Other editing equipment manufacturers quickly jumped on the desktop bandwagon. Those that didn't (and some that did) were destined to become footnotes in the annals of broadcast equipment history.

Regardless of the platform on which they were built, all of these pioneering digital nonlinear systems, as well as most of those in production today, had one thing in common: the internal media handling was performed by custom engines. Each implementation relied on proprietary and incompatible file formats, unique and incompatible programming interfaces, and nonexistent data interchange methods. Additionally, every implementation had to fight with operating system constraints and surprises, particularly in the areas of internal stream synchronization, external device synchronization, and data throughput, to name a few.

Any review of desktop production technologies would be incomplete without mentioning standardization attempts such as Apple's Quick Time™ multimedia architecture and Avid Technologies' Open Media Framework Interchange format. However, these and other efforts have not successfully addressed cross-platform issues on the Microsoft Windows operating systems or cross-application issues much beyond digital audio interchange. Microsoft's own Video for Windows multimedia architecture, designed for low bit rate consumer multimedia applications, is also insufficient for the high-performance demands of professional users.

The formation of the OpenDML committee in 1995, consisting of a very vocal group of Windows-based video tools and hardware manufacturers, underscored the desire for open standards in digital media handling and interchange. Given the "flattening" of the historically vertical tools market, a standardized multimedia architecture would enable better interchangeability of video and audio hardware with application software as well as merge traditionally separate application spaces such as video editing, video conferencing, and server-based media streaming. Other motivating factors are the new hardware buses such as the IEEE 1394 serial bus and Universal serial bus (USB), which are designed with multimedia devices in mind and promise to enable broad new classes of audio and video application programs.

To address these and other requirements, Microsoft introduced DirectShow™, a next-generation media-streaming architecture for the Windows and Macintosh platforms. In development for two and a half years, DirectShow was released in August 1996, primarily as an MPEG-1 playback vehicle for Internet applications, although the infrastructure was designed with a wide range of applications in mind. DirectShow's follow-on release, incorporating substantial input from OpenDML and key application software developers, addresses many high-end tools issues and is now in Beta release. High data rate video capture and playback, seamless nonlinear playback from multiple files, support for transitional effects, improved inter-stream synchronization, an improved media file format, external device control, and a new hardware driver model are just part of DirectShow's feature set. Other benefits include container format independence, location transparency of software modules, and hardware and software codec interchangeability and scalability.

Inside DirectShow—Supporting Technology

A look under the hood of DirectShow reveals that it builds on several Microsoft technologies, although it should not be considered "just another layer" of system software (Fig. 1). DirectShow provides functional partitioning, module connection, media type negotiation, and stream control services to applications. Other components provide high-performance media services, low-latency stream handling, hardware independence, and an overall model for component construction, loading, and intercomponent communication.

Presented at the 31st SMPTE Advanced Motion Imaging Conference (paper 31-6), New York, N.Y., February 6 to 8, 1997. Amit Chatterjee is with Microsoft Corp., Redmond, WA 98052; Andrew Maltz (who co-read the paper) is with Digital Media Technologies, Inc., Van Nuys, CA 91411. An unedited version of this paper appears in *The Age of COmpression: Nonlinear Editing, Digital Broadcasting, and Other Wonders*, SMPTE, 1997.

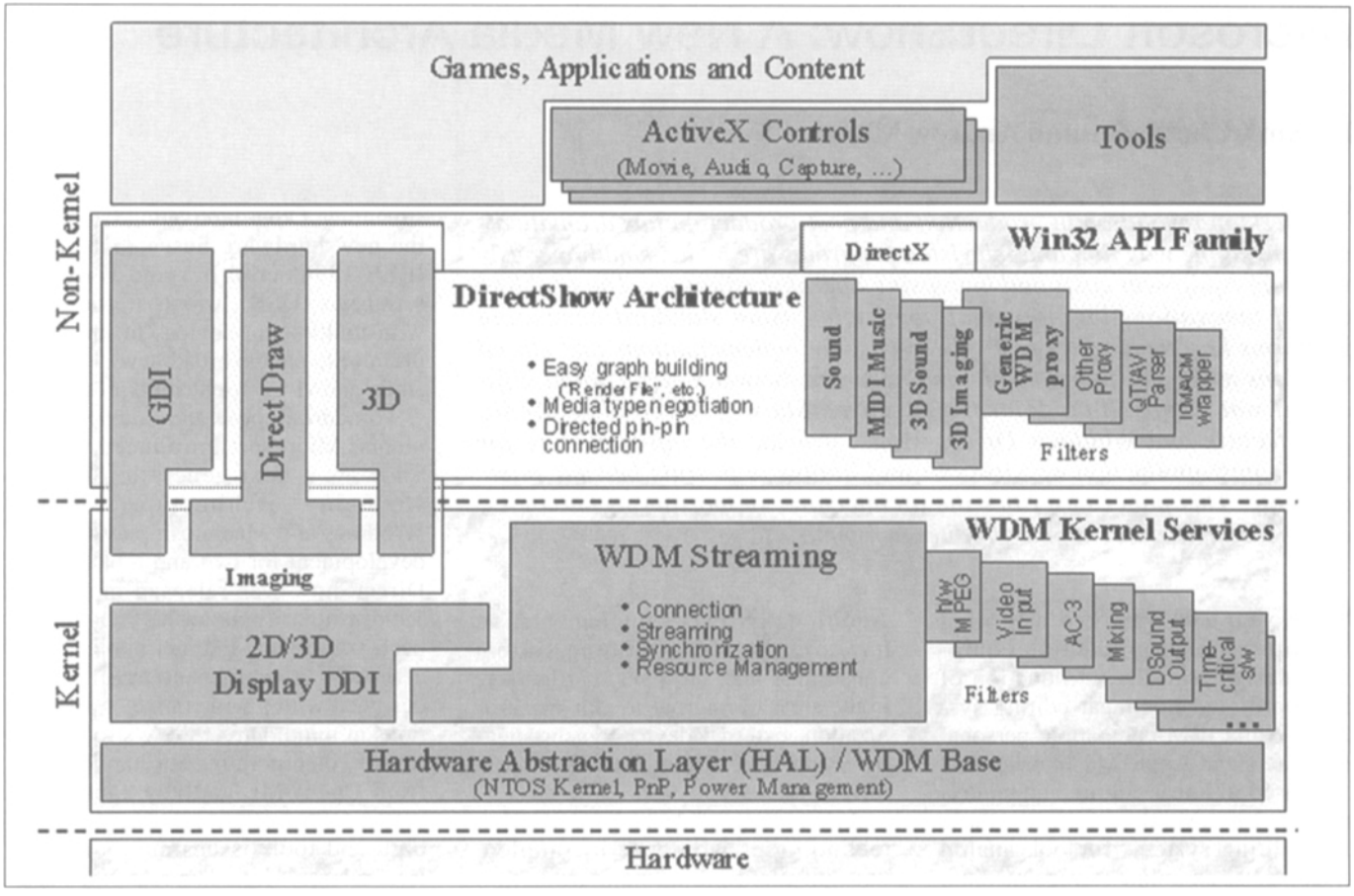

Figure 1. Microsoft multimedia architecture.

Taking a bottom-up approach, the lowest level component is the hardware abstraction layer (HAL) common to all Microsoft operating systems. As the basis for hardware independence, this is the key to having one's application run transparently on either an Intel x86, DEC Alpha, or PowerPC processor, requiring only recompilation with the proper code generator in place. Multiprocessor implementations are also handled at this level.

Also part of the lowest layer are kernel and base driver services, containing high-priority system tasks, plug-and-play support, and power management. It is at this level that low-latency and time-critical software resides. This subject will be covered in more depth further on.

At the top of the kernel services layer, and extending above it as well, are additional Microsoft DirectX Media technologies, comprised of DirectDraw for two-dimensional graphics, DirectSound for audio, Direct3D for three-dimensional graphics, DirectInput for joysticks and game consoles, and DirectPlay for multimachine, multiplayer gaming. All of the DirectX drivers were initially released for the Windows 95 operating system, and most are now being ported to Windows NT. They are designed for low-latency and high-performance control of hardware, although at present they are primarily user mode modules. This begs the question "What is user mode?"

The Windows NT operating system takes advantage of two distinct privilege modes of today's microprocessors, commonly called "user mode" and "kernel mode." The primary difference between them is that the level of access to the underlying hardware and memory space is unrestricted to software running in kernel mode and highly restricted to software running in user mode. The result of this privilege separation is a robust computing environment: application software and many operating system components run in separate and protected user mode memory spaces. Since they do not have direct access to sensitive areas of the operating system running in kernel mode, it is extremely difficult for them to "crash" the system.

The downside to this separation is that any direct hardware register request or other kernel mode call made by an application program incurs a time penalty for each user-to-kernel mode transition. This is due to the necessary saving of registers and environment state, known as a "context." High-performance applications have recourse for this, as we will see a bit later.

The Windows driver model (WDM) is a new driver model that makes development of low-latency, cross platform drivers more practical (Fig. 2). WDM drivers reside completely in kernel mode and can communicate directly with other WDM drivers, thus eliminating the performance hit associated with mode transitions. WDM drivers are modular, using a class driver/minidriver structure. This layered approach separates drivers into logical partitions, allowing a particular "stack" to be configured for a given

peripheral using a given protocol connected via a given bus.

Microsoft is supplying a class driver for streaming media that contains all of the code necessary to support plug-and-play, memory management, property and method set management, and other core services. This reduces the amount of code development for a hardware manufacturer to a relatively simple minidriver that can be written in the C programming language.

Before moving on to the details of DirectShow, one other Microsoft technology must be introduced: the component object model (COM) (Fig. 3).

COM is the underlying object-oriented software technology upon which most of Microsoft's newer products are based. It defines the communications rules by which individual software components connect, control, and transfer data. The software is indifferent to programming languages and guarantees binary compatibility between objects. COM is also cross-platform, supported on Windows, Macintosh, and UNIX operating systems.

One of the essential COM concepts is location transparency. COM objects can communicate whether physically present within the same process space on a single machine, or running on a computer connected via a network yet physically located half a world apart. Location transparency is made possible by remote procedure calls (RPC), which are part of the industry standard distributed computing environment specification. RPCs handle the actual communication between COM objects.

COM also defines a contract between the object itself and the rest of the world through interfaces. Interfaces are strongly typed and semantically related groups of functions called methods. They are described by a globally unique identifier (GUID), a 128-bit number that is pretty hard to accidentally assign to more that one interface. One of the fundamental COM rules is: interfaces are immutable. That is, if an interface changes, it gets a new name and a new GUID. This rule simplifies software versioning and guarantees that software written to use a particular interface will always work (once fully debugged, of course).

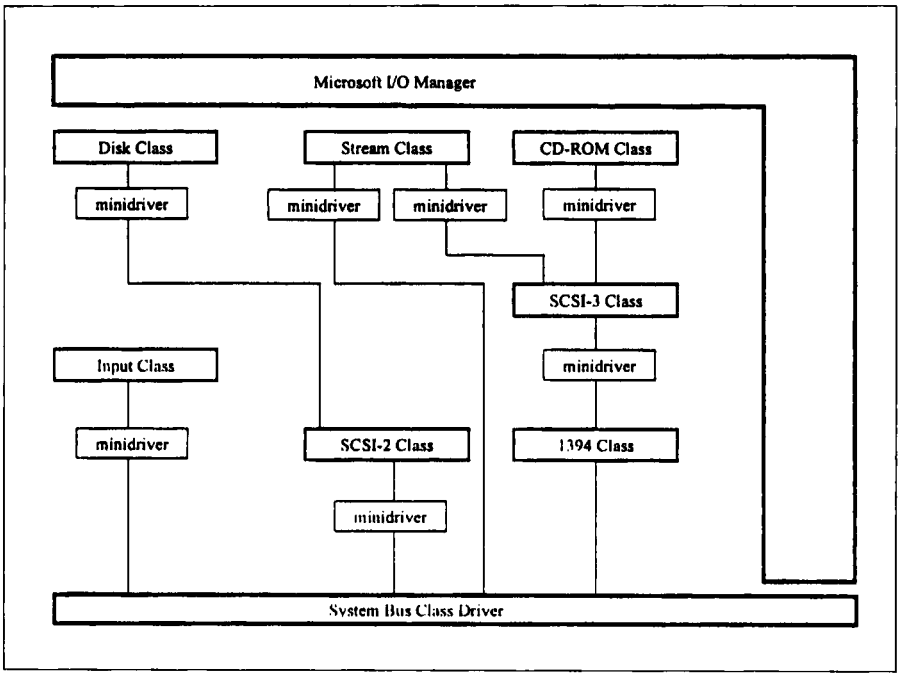

Figure 2. Windows driver model.

Inside DirectShow—Pins, Filters, and Graphs

DirectShow is a set of COM objects specifically designed to enable streaming media applications (Fig. 4). Because DirectShow is COM-based, it is truly extensible. That is, new features can be added by defining additional interfaces and new and extended implementations of DirectShow's basic objects as the need arises. These basic objects, known as filters, pins, and filter graphs, are the building blocks from which all DirectShow applications are constructed.

Filters are objects that perform a specific task such as file reading, file writing, compression, transitional effects, or image display. They are commonly grouped into three basic categories: source, transform, and renderer, although some filters defy simple categorization such as those used for external device control.

Filters transfer time-stamped data called media samples between one another through another object called a pin. Pins are exposed by a filter as either input or output, depending on whether each pin sinks or sources media samples. Pins are the point of connection between filters and are the objects that actually handle the transfer of media samples. DirectShow defines many standard video, audio, and text media types, and developers are free to define their own as required.

The method of sample transfer between filters is called a transport. DirectShow currently supports three transports: local memory-based, video overlay, and hardware-based. The local memory transport uses system memory to store media samples, and an efficient implementation passes pointers to the media samples to avoid performance-killing memory copies. The overlay transport enables traditional analog overlay and digital inlay hardware. Hardware-based transports allow an add-in adapter's own memory to be used, an example being a video capture card with on-board memory supplying media sample buffers for direct reading by a bus mastering SCSI host adapter. Custom transports can also be defined for more specialized applications.

A collection of filters connected together is called a filter graph. An application can allow DirectShow to automatically construct a filter graph for certain applications such as media file playback and capture, or it can manually assemble the filters together to serve its own special purposes. The filter graph's overall behavior is controlled by an object called the filter graph manager. In the case of MPEG

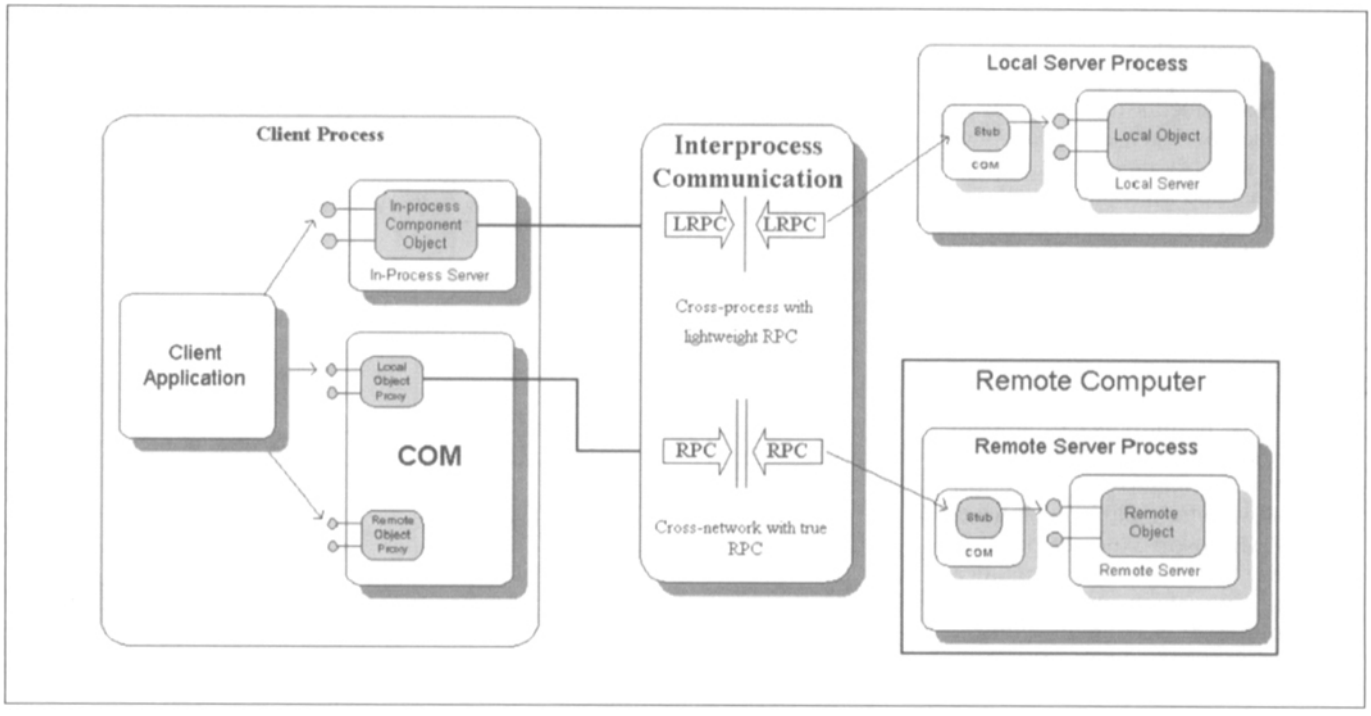

Figure 3. Component object model.

playback, for example, stream control such as run, stop, and pause are handled via the filter graph manager's control interfaces. Applications can also communicate directly with control interfaces on individual filters and pins to control specialized aspects of their behavior, such as video and audio parameter adjustments.

Professional Application Issues

The bane of most software media engines' existence is proper synchronization, both interstream and with external references. In the video world, almost every piece of hardware has the ubiquitous reference or genlock input connector, and phase-locked loops handle the job of synchronizing to a reference video signal quite well. The computer world has consistently ignored this basic requirement of video (and audio) systems design by trusting free-running and unstable audio cards to supply the master reference clock. If streams slipped, then dropping a frame or five here and there would bring things back together. On top of this, applications generally were not informed of this poor state of affairs until it was too late. Hardware-locked video and audio sample clocks are standard features on high-end hardware today, but this solution generally requires integrated audio and video hardware from the same manufacturer, which limits end user choice. This approach also does not enable applications that want to display real-time video on a free-running video graphic array (VGA) display without artifacts such as "tearing" caused by the asynchronous video clocks.

DirectShow addresses this problem on several fronts. First, it allows any filter to supply a time-stamped reference clock to drive the filter graph. A video capture card, SMPTE time code reader, or other device capable of exposing a clocking signal can then be the system master. The time stamp is usually the system time, and since that is always kept current, a filter or application can determine the amount of time that has passed since the reference clock value was updated. This feature is very important for determining the existence of latencies and compensating for them. Second, new facilities are provided for interstream rate matching, particularly with audio. This is done at three levels: coarse filtering of audio samples, dynamic hardware sample clock rate adjustment, and low-latency sample rate conversion using kernel mode filters. The technique selected depends on the required quality, hardware capabilities, and available computing horsepower.

Reference clock suppliers also provide notification services to other filters using events. Events are operating system objects that trigger when preset conditions are met, and can be configured in one-shot or multivibrator modes. The net effect of a triggered event is to unblock a waiting thread of execution in a filter, thus synchronizing that filter's processing with the clock source.

DirectShow's concept of time is important because it impacts synchronization and is the basis for the quality management mechanism described below. DirectShow and its applications maintain four distinct yet related time values:

- Media Time: a temporal position within a seekable medium, i.e., the byte position with a data file.
- Reference Time: absolute or "wall clock" time is established by a reference clock. It is always counting regardless of graph state.
- Stream Time: the offset from the time the graph was started; also called relative reference time.
- Presentation Time: the reference time at which media samples should be presented. This is tied directly to the reference time and is calculated by

the following formula:

[(Media Time - Starting Media Time) / Playback Rate] + Starting Reference Time

This value is used by a rendering filter to determine exactly when to render a sample.

The third synchronization technique used by DirectShow is a rich set of quality management controls, which are driven by the time model previously described. Based on comparisons of presentation times and reference time, filters can report buffer overflow, underflow, or watermark conditions to other filters in the stream so that adjustments can be made before samples must be dropped or duplicated. For example, in the case of video capture to disk, a compressor filter can be told to increase its compression ratio by the downstream filewriter filter if the disk subsystem falls behind. When the "flood" condition corrects itself, the compressor can be told to restore its previous settings.

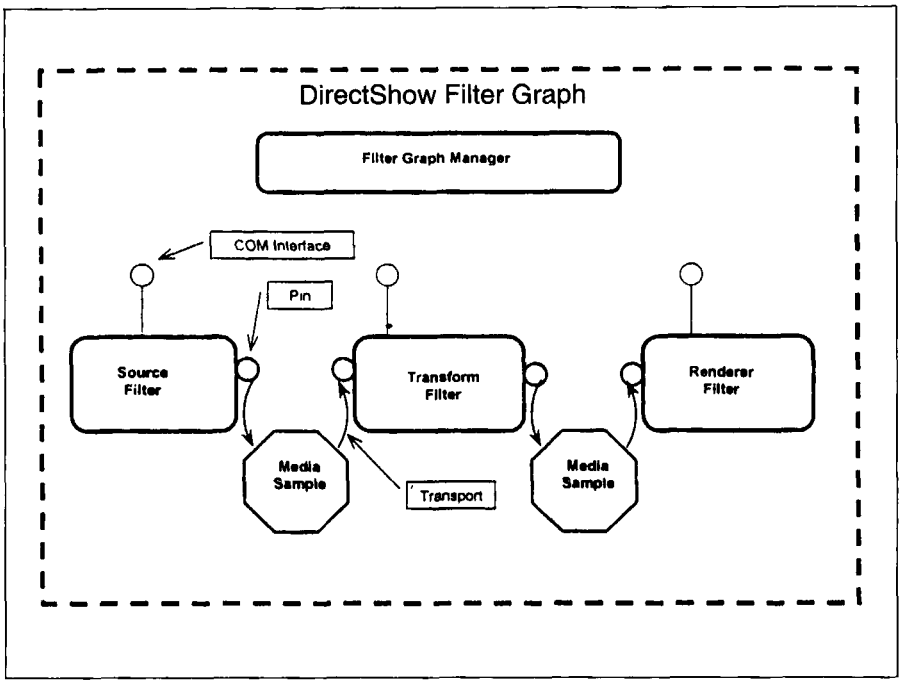

Figure 4. DirectShow components.

Editing Support

Editing applications have additional requirements beyond basic media handling. One of the most basic is support for nonlinear playback of material from multiple sources. Since multiple source filters can exist in a filter graph, an application can easily mix material sourced from an audio video interlead (AVI) file, a wave (WAV) file, an MPEG-1 (MPG) file, and even a QuickTime MOoVie file. Additionally, DirectShow provides interfaces that allow for cut-list-driven playback through a low-level technique known as dynamic graph reconstruction (Fig. 5).

Playback segments can be divided into reusable portions of filter graphs, or graphlets. In the diagrammed case, the graphlets consist of source filters, parsers, and stub filters that are connected to downstream target filters by the filter graph manager at reference times determined by a DirectShow playlist. This technique is also quite useful for reordering complex effects and compositing filter graphs, where hardware codecs or effects processors must appear in different positions at different reference times.

The internal and target filters shown in Fig. 5 have another benefit. They allow an application to directly remove or introduce media samples from a running filter graph. This can be very useful if an application wants to read time code values directly or source images to and sink from non-DirectShow applications.

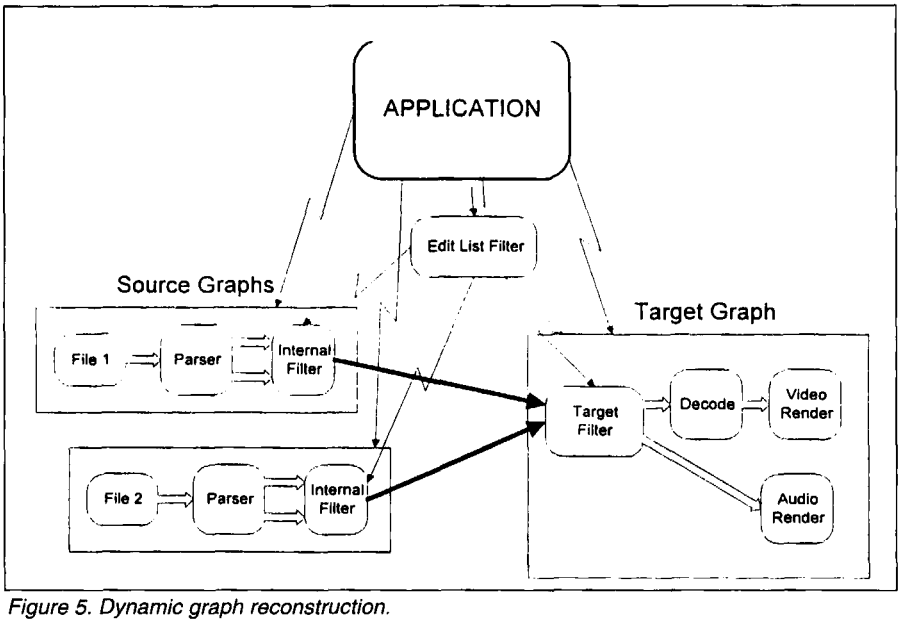

Figure 5. Dynamic graph reconstruction.

Other higher level compositional notations are currently under consideration for applications that do not require such fine control of filter graph construction.

Improvements have also been made to the AVI file format to support video editing. Extremely large files can now be created, and the resulting large frame indexes are distributed throughout the file for efficient access. Other refinements include support for video fields, stream interleave control, and SMPTE time code and film source information.

Since video and audio data are typically supplied to editing systems on videotape, and these systems output to videotape as well, frame accurate control of external devices is yet another

requirement. DirectShow defines interfaces for complex machine control, and filter implementations for a wide selection of professional and industrial videotape machines will soon be available. SMPTE time code is also defined as a standard media type and control interfaces are defined to enable time code acquisition and generation.

Sample Filter Graphs

Figures 6 and 7 show sample filter graphs for various applications possible with DirectShow. In Fig. 6 the important filter is the capture source filter shown on the left. It connects via private interfaces to the Microsoft Stream Class Driver, which in turn talks to the capture hardware through vendor-supplied minidrivers. While only a single source filter is shown, it can also be broken down into separate audio and video capture filters. The tee filters are also interesting because they can be installed ad infinitum, up to the practical processing limit of the host CPU.

Distributed applications (Fig. 7) are installed in many facilities today, with centralized image or stream storage supplied by remotely located file servers. For news editing and even episodic television editing, which are both multi-editor and short-turnaround time programs, DirectShow enables the simultaneous capturing and playback required for efficient throughput.

High-Performance Applications—WDM Streaming

DirectShow addresses a great many of the systems issues for streaming media applications. Its services and clients, however, run in user mode. In some situations, the latencies and nondeterministic behavior of a user mode process can get in the way of achieving "really timely" performance. In these cases, applications can be developed using a new, high-performance Microsoft technology: the WDM Connection and Streaming Architecture, or WDM Streaming for short. WDM Streaming extends DirectShow streaming services down into the kernel, where filters can take advantage of more predictable scheduling services and can pass media samples directly between drivers without costly mode transitions.

Some background on performance issues with respect to the Windows NT operating system is appropriate at this point. There are many priority levels at which various software components will execute, and a given system's performance is directly related to how the various tasks are distributed among them. At the risk of over simplification, the order of execution priority from highest to lowest goes something like this:

- Interrupt Service Routines (ISR): usually triggered by a hardware event and runs to completion unless interrupted by a higher priority interrupt. ISRs are typically very short pieces of code that usually read registers or collect data for later processing.
- Deferred Procedure Call (DPC): usually scheduled by an ISR and runs to completion unless interrupted by an ISR. DPCs usually handle non-time critical processing.
- System Thread: can also be scheduled for lengthier driver processing. System threads are preemptible by other system threads and can be inter-

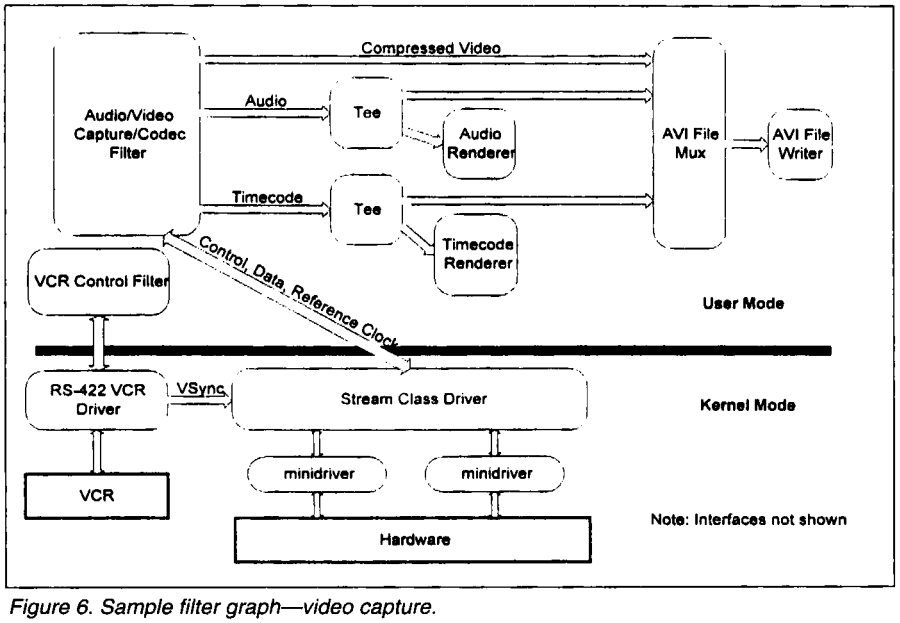

Figure 6. Sample filter graph—video capture.

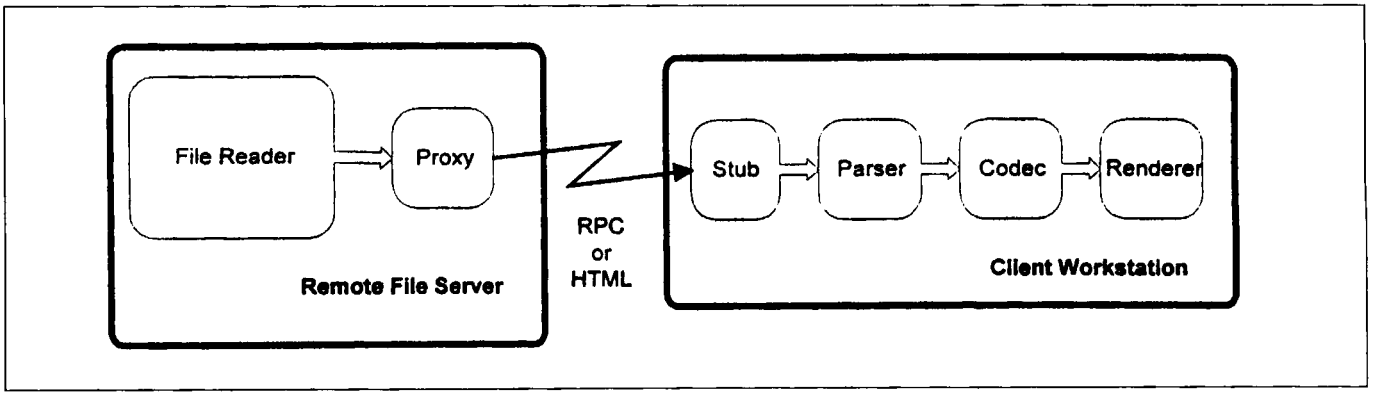

Figure 7. Sample filter graph—distributed application.

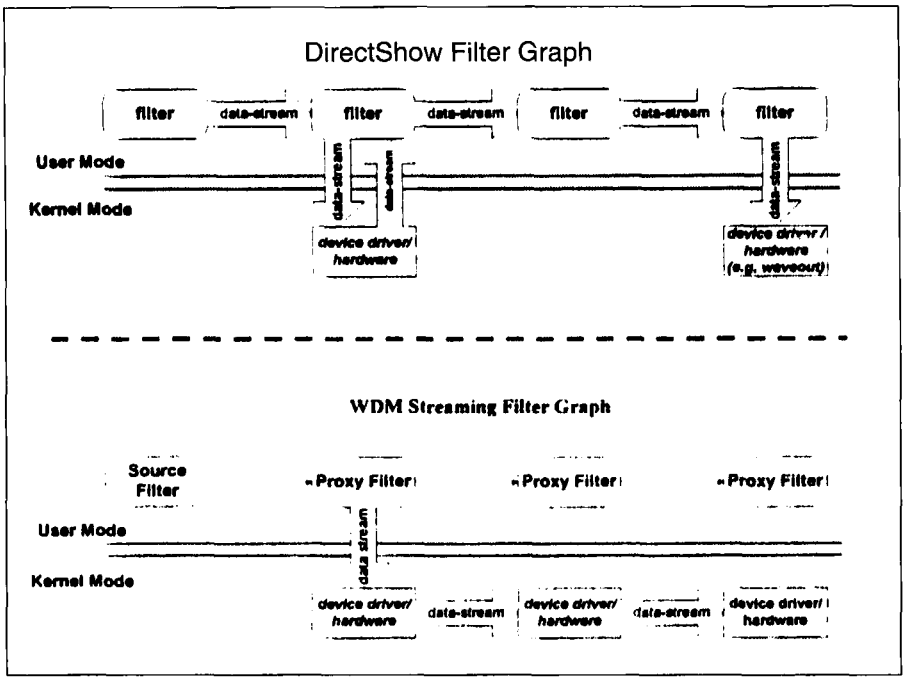

Figure 8. WDM Streaming vs. DirectShow.

rupted by ISRs.

• User Mode Thread: the standard execution object for an application program. User mode threads are assigned one of 32 priority levels and can be both interrupted and preempted. User mode threads generally have the least deterministic behavior and as such are poor choices for time-sensitive applications.

The first four execution components operate strictly in the kernel, and thus are generally not available to application programs.

In a typical DirectShow playback graph, incoming samples from a piece of hardware must cross the boundary into user mode for processing by downstream filters (Fig. 8). The samples must then cross back again into kernel mode for output. And as previously stated, user mode processes are scheduled at relatively low priorities and can incur varied and unacceptably long latencies if higher priority and time-consuming tasks are also running.

In a WDM Streaming playback graph, kernel mode drivers appear to applications via a DirectShow proxy filter. Microsoft supplies a generic proxy filter that translates a set of control interfaces into kernel mode commands called input/output controls (IOCTLS). The proxy is also responsible for marshalling the kernel filter's responses back to the application. WDM Streaming also defines another transport called device input/output interface (IDevIO). IDevIO is a memory-based transport similar to that of DirectShow's, but it enables the direct, kernel driver-to-kernel driver media streaming. The net result is that high-priority and low-latency execution components are now made more available to applications through the WDM Streaming architecture.

A natural use of WDM Streaming is for real-time audio echo cancellation in desktop teleconferencing applications. This has latency requirements in the 1-msec range. While very difficult to achieve in a user mode filter graph, this is certainly possible with WDM Streaming.

Such high performance, however, is not without its costs. Developing kernel mode components is one of the more difficult software development challenges and carries with it the potential for compromising overall system stability if not done correctly.

Conclusion

Microsoft DirectShow and its related multimedia technologies provide a rich, robust, and extensible platform for high-performance media applications. The acceptance of these technologies in the professional tools market is made evident by the substantial applications expected to be available in 1997.

Acknowledgments

The authors would like to thank the Microsoft teams responsible for the ongoing development of the technologies described herein. We would also like to thank the many independent software and hardware vendors that contributed valuable input and feedback to Microsoft.

THE AUTHORS

Amit Chatterjee completed his Bachelor of Technology Degree with honors in electronics and electrical communications from the Indian Institute of Technology, Kharagpur, in 1984. He received the President of India Gold Medal for being judged the best graduating student for the period 1979-1984. Amit completed his Master of Technology Degree in computer engineering from the same institute in 1985 and received the Technology Alumni Association Gold Medal. Chatterjee has been working at the Microsoft Corp. since 1988. During this period he has been part of the Windows development team that shipped Windows 95, 3.1, 3.0, and 2.1. He is currently the development lead for Microsoft's DirectShow multimedia architecture.

Andrew Maltz is an entertainment technology consultant in Los Angeles. He pioneered nonlinear video editing systems as a principal developer of the Emmy award-winning Ediflex and its digital successor, developed computer animation and digital audio systems, and has worked in film and television production and postproduction. Maltz's current work is in media engines for the Microsoft Windows platforms, digital television equipment for film mastering, and video compression for high-quality desktop applications. Maltz has a BSEE from SUNY Buffalo and is a member of several SMPTE Working Groups.