**Multimedia Abstraction**

Multimedia applications should have higher abstraction from multimedia hardware details. It’s a state of the art of programming that shield the hardware details and complexities from software applications and users.

Generally application code is highly dependent on hardware. In such cases, change of multimedia devices means reconstruction or re-implementation of applications.

For e.g.

Following example is taken from Linux kernel code of i386 architecture. The code or instruction used here seem to hardware dependent instruction.

**static unsigned long read\_timer(void)**

{

/\* example taken from Linux kernel code - i386 architecture \*/

unsigned long t, flags;

int i;

\_\_save\_flags(flags);

\_\_cli();

t = jiffies \* 11932;

outb\_p(0, 0x43);

i = inb\_p(0x40);

i |= inb(0x40) << 8;

\_\_restore\_flags(flags);

return (t - i);

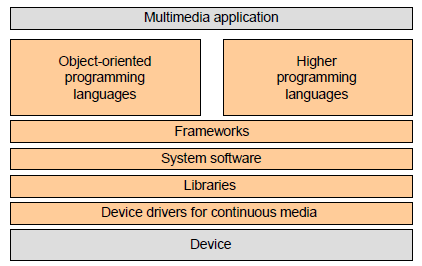
}

Strong hardware dependency may cause following problems:

1. Portability
2. Reusability
3. Coding Efficiency

**Abstraction Layer**

* Common operating system extensions try to solve this type of dependency problems. And different programming approaches are avilabel for accessing and representing multimedia data:
* Following diagram shows different layer of abstractions and component used at those layers for accessing multimedia data.



**1. Libraries**

* A library is a collection of implementations of behavior, written in terms of a language, that has a well-defined interface by which the behavior is invoked.
* In addition, the behavior is provided for reuse by multiple independent programs.
* A program invokes the library-provided behavior via a mechanism of the language.
* For example, in a simple  language such as C, the behavior in a library is invoked by using C's normal function-call.
* What distinguishes the call as being to a library, versus being to another function in the same program, is the way that the code is organized in the system.
* ***Processing of continuous media based on functions are embedded in libraries***
* Libraries differ in their degree of abstraction
* For Example from IBM’s early Audio Visual Connection (AVC):

acb.channel = AAPI\_CHNA

acb.mode = AAPI\_PLAY

...

aud\_init(&acb) /\* acb is the audio control block \*/

...

audrc = fab\_open(AudioFullFileName,AAFB\_OPEN,AAFB\_EXNO, 0,&fab,0,0,0,0);

fork(START in PARALLEL)

aud\_strt(&acb)

displayPosition(RelativeStarttime, Duration)

**...**

**Feature of Libraries**

* Organized Code ( Dependency low)
* Reusable : modular coding fashion

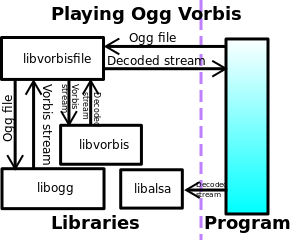


Fig: Illustration of an application which uses libvorbisfile to play an Ogg Vorbis file

**Libraries – OpenGL**

It’s a 2D and 3D graphics API developed by Silicon Graphics

• Basic idea is “write applications once, deploy across many platforms”:

* PCs
* Workstations
* Super Computers

• Benefits of OpenGL Libraries:

* Stable
* Reliable and Portable
* Evolving
* Scalable (Features like Zoom, Rectangle handling ...)
* Well documented and easy to use

• This can be integrated with:

* Windows 95/NT/2000/XP
* UNIX X Window System

**2. System Software**

System software (or systems software) is computer software designed to operate and control the computer hardware and to provide a platform for running application software.

System software includes the following:

* The *operating system* (prominent examples being z/OS, Microsoft Windows, Mac OS X and Linux), allows the parts of a computer to work together by performing tasks like transferring data between memory and disks or rendering output onto a display device. It also provides a platform to run high-level system software and application software.
* *Utility software* helps to analyze, configure, optimize and maintain the computer.
* Device drivers such as computer BIOS and device firmware provide basic functionality to operate and control the hardware connected to or built into the computer.
* A *user interface* "allows users to interact with a computer."Since the 1980s the graphical user interface (GUI) has been perhaps the most common user interface technology. The command-line interface is still a commonly used alternative.

**How OS access devices and multimedia data ?**

Device access becomes part of the operating system:

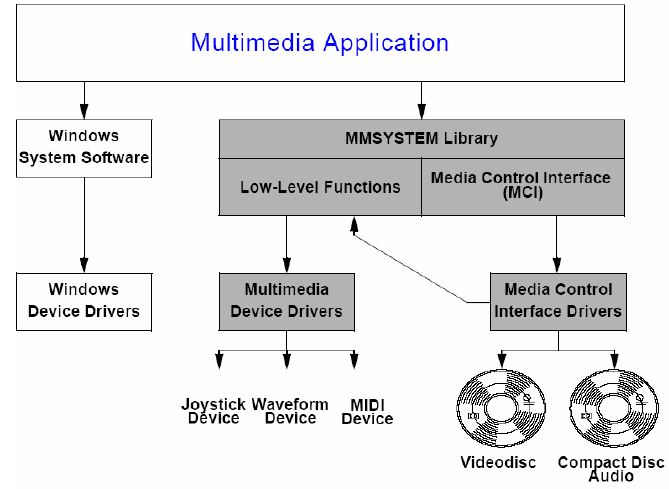
**• Data as *time capsules* (file extensions)**

* + - Each Logical Data Unit (LDU) carries in its time capsule its data type, actual value and valid life span
    - Useful concept for video, where each frame has a valid life span of 40ms (rate of read access during a normal presentation)
    - Presentation rate is changed for VCR (Video Cassette Recorder) functions like fast forward, slow forward or fast rewind by
      * Changing the presentation life span of a LDU
      * Skipping of LDUs or repetition of LDUs

**• Data as *streams***

* A stream denotes the continuous flow of audio and video data between a source and a sink
* Prior to the flow the stream is established equivalent to the setup of a connection in a networked environment

System Software :Windows *Media Control Interface* (MCI):



Above diagram shows MMSYSTEM library for extensibility and device independence

**System Software :MCI**

* The **Media Control Interface** — **MCI** for short — is a high-level API developed by Microsoft and IBM for controlling multimedia peripherals connected to a Microsoft Windows orOS/2 computer, such as CD-ROM players and audio controllers.
* MCI makes it very simple to write a program which can play a wide variety of media files and even to record sound by just passing commands as strings.
* It uses relations described in Windows registries or in the [MCI] section of the file system.ini.
* One advantage of this API is that MCI commands can be transmitted both from the programming language and from the scripting language (open script, lingo aso). Example of such commands are mciSendCommand or mciSendString.
* For a number of years, the MCI interface has been phased out in favor of the DirectX APIs first released in 1995.

## MCI Devices

The Media Control Interface consists of 4 parts:

* CD audio
* Digital video
* Overlay
* sequencer
* VCR
* Video disc
* Wave audio

Each of these so-called MCI devices (e.g. CD-ROM or VCD player) can play a certain type of files, e.g. AVIVideo plays .avi files, CDAudio plays CD-DA tracks among others. Other MCI devices have also been made available over time.

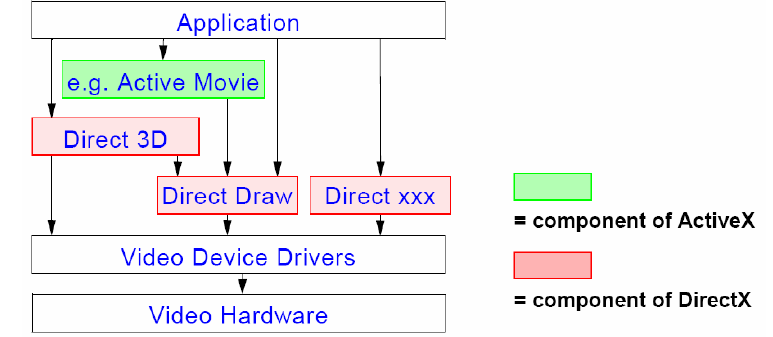
## Playing media through the MCI interface

To play a type of media, it needs to be initialized correctly using MCI commands. These commands are subdivided into categories:

* System Commands
* Required Commands
* Basic Commands
* Extended Commands

**System Software - DirectX**

* Low-level APIs and libraries for high-performance applications
  + Especially games - formerly known as the "Game SDK"
* Direct access to hardware services
  + E.g. audio & video cards, hardware accelerators
  + “DirectX” = “direct access”
  + Strong relationship/interaction with ActiveX/DCOM



Different Components of DirectX:

• DirectDraw - 2 dimensional graphics capabilities

• Direct3D - extensively functional 3D graphics programming API

• DirectSound - (3D) sound, mixing and playback of multiple streams

• DirectPlay - for network multiplayer game development

• DirectInput - input from various peripherals, e.g. joysticks, data gloves

**3. Framework**

Additional level of abstraction:

* Combination of different media
* New data types
* E.g. mixed.video := orig.video + subtitle
* Granularity problematic (pixel, frame, image group, sequence, video, ...)
* Reuse of modules:
* Synchronization
* Scaling
* (De)compression
* File formats
* …

Application frame given: generic problem solution

Creation of specialized application components: media containers, media processors

**Framework :JMF**

Framework controls execution

* E.g. myApplicationComponent.init(); myApplication.start();
* Example: Java Media Framework (JMF)

JMF supports platform-neutral synchronized playback of multimedia data

* Local or via network:
* Pull protocols - e.g. HTTP
* Push protocols - e.g. RTP

Easy integration into platforms native environment and core packages; supported media

formats:

* MPEG, QuickTime, AVI
* WAV, AU, MIDI

Extensible:

* New Data Sources (e.g. for FTP or Video on Demand (VOD) protocols)
* New Players

**4. Higher Programming Language Requirements**

* The high-level language should support parallel processing, because the processing of continuous data is
  + controlled by the language through pure asynchronous instructions
  + an integral part of a program through the identification of media
* Different processes must be able to communicate through an inter-process communication mechanism, which must be able to:
  + Understand a priori and/or implicitly specified time requirements (QoS parameters or
  + Transmit the continuous data according to the requirements
  + Initiate the processing of the received continuous process on time

**5. Object-Oriented Approaches**

Basic ideas of object-oriented programming is data encapsulation in connection with class and object definitions

• Abstract Type Definition (definition of data types through abstract interfaces)

• Class (implementation of a abstract data type)

• Object (instance of a class)

Other important properties of object-oriented systems are:

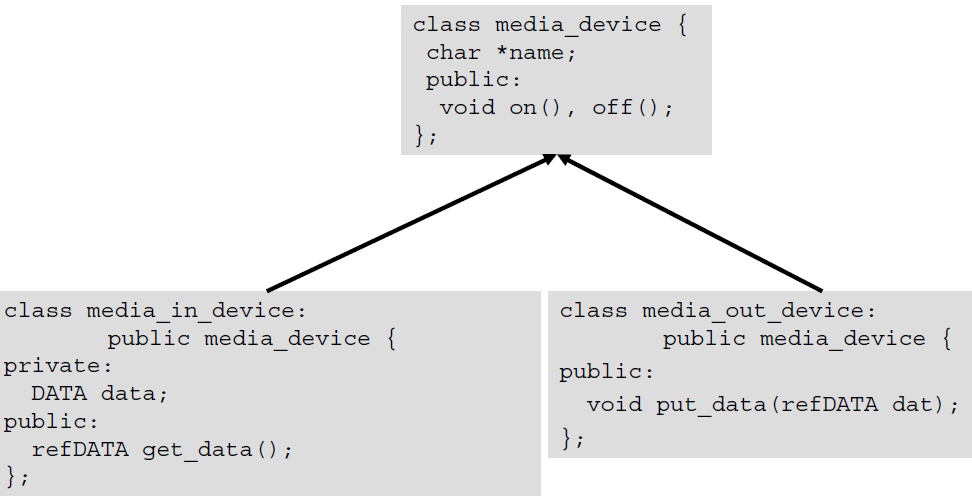
• Inheritance

• Polymorphism

Object Oriented Approach

Devices as classes: devices are assigned to objects which represent their behavior

and interface



**Processing units as classes:**

• Three main objects:

\_ Source objects

\_ Destination objects

\_ Combined source-destination objects allows the creation of data flow paths through connection of objects

• Multimedia object

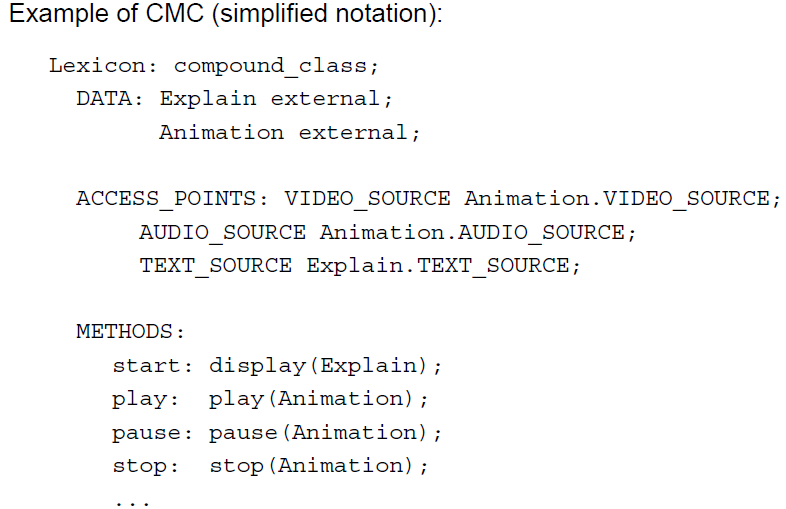
\_ Basic Multimedia Classes (BMCs) /

Basic Multimedia Objects (BMOs)

\_ Compound Multimedia Classes (CMCs) /

Compound Multimedia Objects (CMO), which are compound of BMCs / BMOs and other CMCs/CMOs

\_ BMOs and CMOs can be distributed over different computer nodes



Media as classes:

• Media Class Hierarchies define hierarchical relations for different media

• Different class hierarchies are better suited for different applications

Example :

