UNIT-7

Abstractions for programming:

⊛. Abstraction Levels:

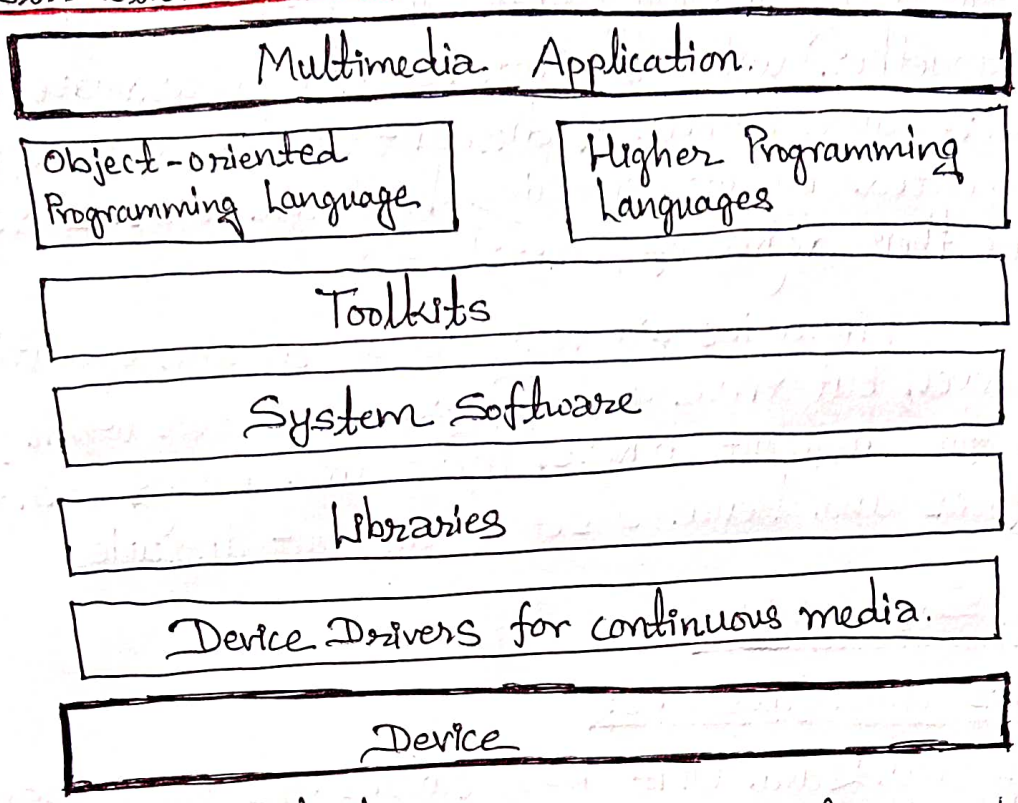

Fig: Abstraction levels of the programming of multimedia systems.

→ Abstraction levels in programming define different approaches with a varying degree of detail for representing, accessing and manipulating data.

→ A device is not part of the operating system but is directly accessible to every component and application.

→ A library, the simplest abstraction level includes the necessary functions for controlling the corresponding hardware with specific device access operation.

→ Device drivers are simply the implementation of device access and scheduling.

→ Higher programming languages contain abstractions of multimedia data.

→ An object oriented environment provides most flexibility for programmers.

→ A multimedia application may access each level.

⊛. Libraries:

The processing of continuous media is based on function or set of functions embedded into libraries. This is the usual solution for programming multimedia data. These libraries are provided together with the corresponding hardware. For example, a large number of different audio and video components were supported by corresponding hardware cards. Libraries differ in their degree of abstraction.

Libraries are very useful at the operating system level, but there is no agreement over which functions are best for different drivers. There will always be a variety of interfaces and hence, a set of different libraries.

⊛. System Software:

1) Data as Time Capsules:

→ Each Logical Data Unit (LDU) carries its time capsule along with data type, actual value and valid life span.

→ This concept is more useful for video than audio, where each frame has valid span of 40 ms.

→ Presentation rate is changed for VCR (Video Cassette Recorder) functions like fast forward, slow forward or fast rewind processes, which can be achieved by:

↳ Changing the presentation life span of a LDU.

↳ Skipping of LDUs or repetition of LDUs.

2) Data as Streams: A well-known, used and implemented abstraction at the system level is the stream. A stream denotes the continuous flow of audio and video text. The stream is established between source(s) and sink(s). This is equivalent to the setup of a connection in a networked environment. Operations can be performed on a stream such as play, fast forward, rewind and stop. In Microsoft Windows, a Media Control Interface (MCI) provides the interface for processing multimedia data. It allows access to continuous media streams and their corresponding devices.

⊛. **Toolkits:** A simpler approach in a programming environment than the system software interface for control of the audio and video data processing can be taken by using toolkits. These toolkits are used to:

→ Abstract from the actual physical layer
→ Allow a uniform interface for communication with all different devices of continuous media.
→ Introduce the client-server paradigm.
→ To hide process-structures.

Toolkits should represent interfaces at the system software level. In this case, it is possible to embed them into the programming languages or object-oriented environment.

⊛. **Higher Programming Languages:**

Higher Programming languages also called High-level languages (HLL). In HLL, the processing of continuous data is influenced by a group of similar constructed functions. These calls are mostly hardware and driver-independent. Hence their integration in HLLs leads to a wishful abstraction, supports a better programming style and increases the productivity.

⊛ **Media as Types:** One of the alternatives to programming in an HLL with libraries is the concept of media as types. Here, the data types for video and audio are defined. In the case of text, character is the type. A program can address such characters through functions and sometimes directly through operators. They can be copied, compared with other characters, deleted, created, read from a file or stored. The smallest unit can be the LDU.

⊛. **Media as Files:** Another possibility of programming continuous data is the consideration of continuous media streams as files instead of data types. Read and write functions are based on continuous data behaviour. Continuous data are often played from source of non-persistant data. A microphone and camera are examples

of such sources. The most device units are handeled at their interfaces to the applications as files.

Using this kind of programming of continuous data, the number and functionality of the operations with continuous data is limited. This approach can be seen as the programming of data streams.

⊛. Media as processes: The processing of continuous data contains a time-dependency because the life span of a process equals to the life span of a connection(s) between source(s) and destination(s). A connection can exist locally, as well as remotely. The processing can be done either once or continuously, meaning that during the entire transmission of continuous data:

→ The loudness is determined by a device driver call. The driver loads a certain storage content which is used by the running process controlling an audio board.

→ If the main processor passes the audio data further from a file to the communication system, then the loudness can be changed. Thus, the compression and coding must be considered.

⊛. Programming Language Requirements:

1) 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 program through identification of media.

⊛. InterProcess Communication Mechanism [IPC]:

Different process must be able to communicate through an interprocessing communication and must be able to:
→ understand implicitly/prior specified time requirement.
→ transmit continuous data according to requirement.
→ initiate processing of received continuous process on time.

⊛. Language: The authors see no demand for the development of a new dedicated language. A partial language replacement is also quite difficult because cooperation between the real-time environment and the remaining programs requires semantic changes in the programming languages. The IPC must be designed and implemented in real-time, the current IPC can be omitted.

A language extension is the solution proposed here. An example of such a language is OCCAM-2. Some real-time systems are implemented in this parallel programming language today.

⊛. Object-Oriented Approaches:

⊛. Class: A class is a collection of objects of similar type. Class is user-defined data type and behaves like a built-in type of a programming language. A class is a data type having data member and member functions. The syntax for class is as follows:

```
class class_name{ private:
                  // data member and member functions.
                  public:
                  // data member and member functions.
                  protected:
                  // data member and member functions.
                };
```

⊛. Object: Once a class has been defined, we can create any number of objects belonging to that class. Objects are member variable of a class which are user-defined data types.

Example:

```
class person{ char name[20];
              int id;          } data members
              public:
                void getdata(){} // member function
            };
int main(){ person p1; // p1 is the object of type person
          }
```

⊛. **Inheritance:** Inheritance is the process by which the objects of one class acquire the properties of objects of another class. It helps to share common characteristics with the class from which it is derived. For Example: The bird 'Robin' is a part of class 'flying bird' which is again a part of class 'bird'.

The concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it.

⊛. **Polymorphism:** Polymorphism is the ability to make more than one form. Any operation may show different behaviours in different instances. For example: Consider the operation of addition. For two numbers, the operation will generate sum but if operands are strings, then the operation will produce third string by concatenation.

⊛. **Application-specific Metaphores as Classes:**

An application-specific class hierarchy introduces abstractions specifically designed for a particular application. Thus it is not necessary to consider other class hierarchies. This approach leads to a number of different class hierarchies.

Unfortunately, this is currently the most used solution, which has led to different kinds of class hierarchies. Although, for similar applications, similar class hierarchies can be implemented.

⊛. **Application-generic Metaphores as Classes:**

Another approach is to combine similar functionalities of all applications. These properties of functions can be defined and implemented as classes for all applications. An application is defined only through a binding of this class. For example, basic functions or functional units can create classes. In theory this approach sounds easy to follow. In practice we have not yet a very successful generic application classes, because they only work well for a very restricted set of applications.

⊛. Devices as classes:

The devices are assigned to objects which represent their behaviour and interface. Methods with similar semantics, which interact with different devices, should be defined in a device-independent manner.

The concept of devices as class hierarchies provides a simple parallel performance of the methods. Synchronization is not supported in this hierarchy and must be provided through other components; multiple inheritance is often needed.

⊛. Processing units as classes:

This abstraction consists of source objects, destination objects, and combined source destination objects which perform intermediate processing of continuous data. With this approach, a kind of "lego" system is created which allows for the creation of a data flow path through a connection of objects. The outputs of objects are connected with inputs of other objects, either directly or through channel objects.

⊛. Distribution of BMOs and CMOs:

The channel object with the methods create_connection and delete_connection supports the possibility to manage connections of several media together. A CMO can consist of objects which are distributed over different computer nodes. At the moment when the CMO lexicon is actived, a text of the text object is displayed. The CMO animation is a composition of audio and video objects.

Destinations and Combined source-destination objects can also be specified as BMOs and CMOs. They may represent:

→ Output device such as windows, monitors or speakers.
→ Files of internal secondry storage devices or external storage devices.
→ Processing units of continuous media with input and output ports.

⊛. Media as Classes:

The media class hierarchy defines a hierarchical relation for different media. Different class hierarchies are better suited for different applications. A specific property of all multimedia objects is the continuous change of their internal states during their life spans. Data transfer of continuous media is performed as long as the corresponding connection is actived. A connection can be either a connection for local data transfer between source and destination, or a connection for remote data transfer. Besides, the class hierarchy, the main attributes need to be considered for different classes.

⊛. Communication-specific Metaphores and Classes:

These approaches consider objects in a distributed environment through an explicit specification of classes and objects tied to a communication system. The information, contained in the information objects, can build a presentation object which is later used for presentation of information. Information objects can be converted to transport objects for transmission purposes.

Information is often processed differently. It depends on whether the information should be presented, transmitted or stored. With storage objects, it is necessary to consider the different storage formats. Relevant formats are the coding and compression formats, format of interleaved data streams and formats such as CD-ROM ISO 9660.