Unit 7: Abstractions for programming

The state of the art of programming

Most of the current commercially available multimedia applications are implemented in procedure-oriented programming languages. Application code is still highly dependent on hardware Change of multimedia devices still often requires re- implementation. Common operating system extensions try to attack these problems. Different programming possibilities for accessing and representing multimedia data.

1 ABSTRACTIONS LEVELS

Abstraction levels in programming define different approaches with a varying degree of detail for representing, accessing and manipulating data. The abstraction levels with respect to multimedia data and their relations among each other shown in figure below.

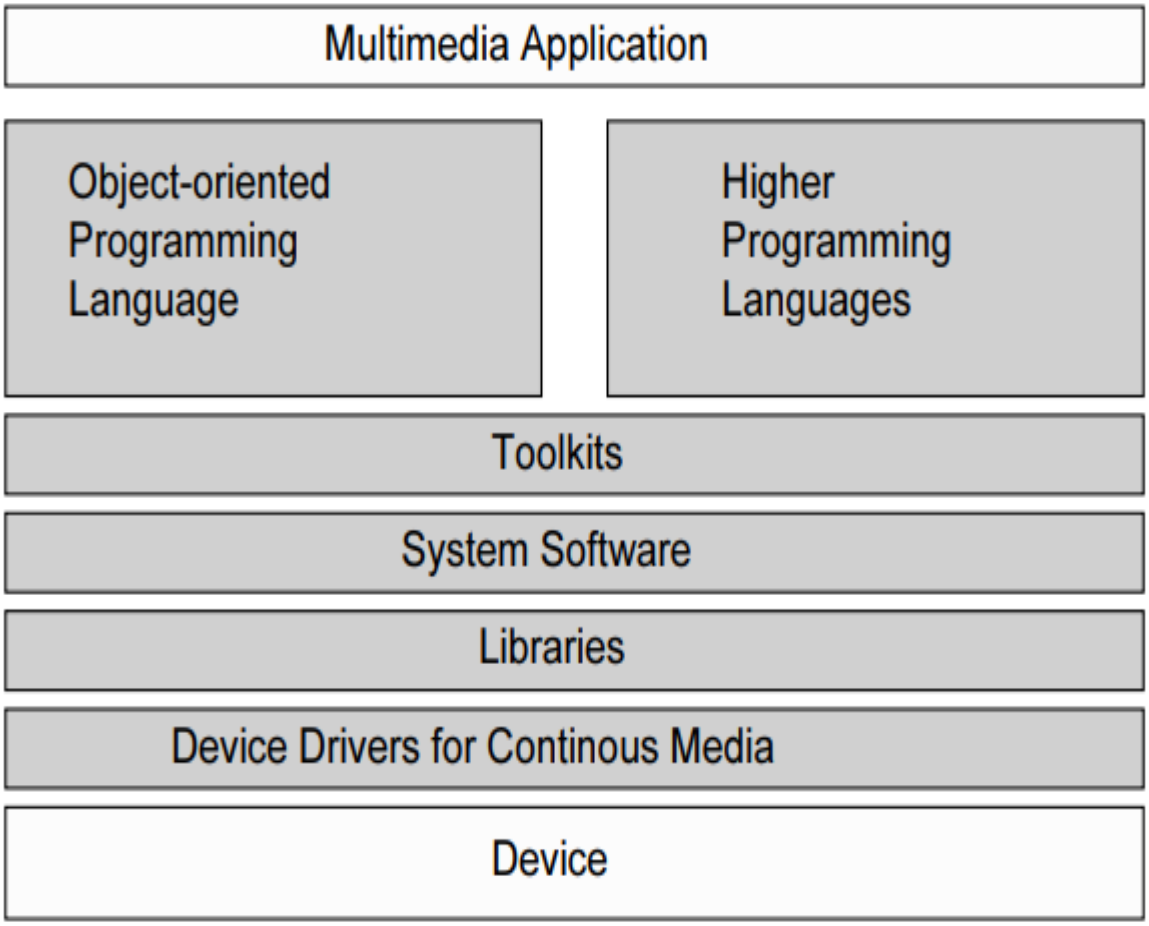

Figure 1: Abstraction Levels of the Programming of Multimedia Systems

A ***device*** for processing continuous media can exit as a separate component in a computer. In this case, a device is not part of the OS, but is directly accessible to every component and application.

A ***library***, the simplest abstraction level, includes the necessary for controlling the corresponding hardware with specific device access operations. Libraries are very useful at the OS level, but there is no agreement over which function are best for different drivers

As with any device, multimedia devices can be bound through a ***device driver***, respectively with the OS.

The processing of the continuous data become part of the ***system software***. For the continuous data processing requires appropriate schedulers, such as rate monotonic scheduler or earliest deadline first scheduler. Multimedia device driver embedded in OS simply considerably the implementation of device access and scheduling.

A simpler approach in a programming environment than the system software interface for the control of the audio and video data processing can be taken by using ***toolkits.*** Toolkits can also hide process structures.

Language used to implement multimedia application contains abstractions of multimedia data known as ***higher procedural programming language***.

The ***object-oriented approach*** was the first introduce as a method for the reduction of complexity in the software development and it is used mainly with this today. Provides the application with a class hierarchy for the manipulation of multimedia.

2 LIBRARIES

Libraries contain the set of functions used for processing the continuous media. Libraries are provided together with the corresponding hardware. Some libraries can be considered as extensions of the GUI, whereas other libraries consist of control instructions passed as control blocks to the corresponding drivers.

Libraries are very useful at the operating system level. Since, there isn't any sufficient support of OS for continuous data and no integration into the programming environment exists, so there will always be a variety of interfaces and hence, a set of different libraries.

Libraries differ in their degree of abstraction.

3 SYSTEM SOFTWARE

Instead of implementing access to multimedia devices through individual libraries, the device access can become parts of the OS. E.g. Nemo system.

The nemo system consists of the ***nemo trusted supervisor call*** (NTSC) running in the supervisor mode and 3 domains running in user mode: system, device drivers and application.

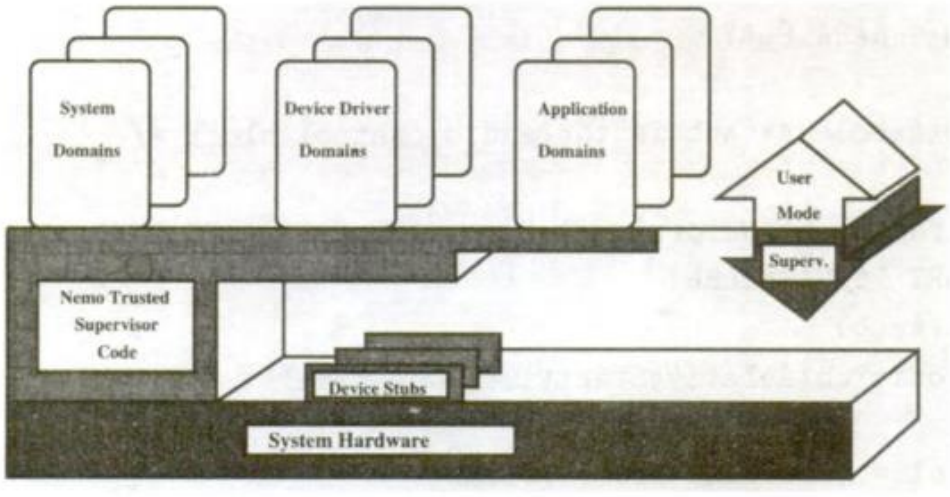

Figure 16.2: *Structure of the Nemo system.*

The NTSC code implements those functions which are required by user mode processes. It provides support for three types of processes. System processes implement the majority of the services provided by the OS. Devices processes are similar to system process, but are attached to device interrupt stubs which execute in supervisor mode.

The NTSC calls are separated into two classes, one containing calls which may only be executed by a suitable privileged system process such as kernel, the other containing calls which may be executed by any processes.

NTSC is responsible for providing a interface between a multimedia hardware device and its associated driver process. This device driver implementation ensure that if a device han only a low-level hardware interface to the system software. Application processes contains user programs. Processes interact with each other via the system abstraction – IPC (InterProcess Communication). IPC is implemented using low-level system abstractions events and, if required, shared memory.

Data as time capsule:

Time capsules are the special abstraction related to the file systems. These files extensions serve as storage, modification and access for continuous media. Each logical data unit (LDU) carries in its time capsule, in addition to its data types and actual value, its valid life span. This concept is used widely in video than in audio.

Data as streams:

A stream denotes the continuous flow of audio and video data. A stream is established between source and sink before the flow. Operation on a stream can be performed 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 the access to continuous media streams and their corresponding devices.

4 TOOLKITS

Toolkits are used for controlling the audio and video data processing in a programming environment. Toolkit hides the process structures. It represents interfaces at the system software level. 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

Toolkits can also hide process structures. It would be of great value for the development of multimedia application software to have the same toolkit on different system platforms, but according to current experiences, this remains to be a wish, and it would cause a decrease in performance.

Toolkit should represent interface at the system software level. In this case, it is possible to embed them into the programming languages or object-oriented environment. Hance, the available abstraction in the subsequent section on programming language and object-oriented approaches.

5 HIGHER PROGRAMMING LANGUAGES

In the higher programming languages, the processing of continuous media data is influenced by a group of similar constructed functions. These calls are mostly hardware and driver independent. The programs in high level language (HLL) either directly access multimedia data structures, or communicate directly with, the active processes in the real-time environment. The processing devices are controlled through corresponding device drivers.

Media can be considered differently inside a programming language.

- Media as types
- Media as files
- Media as processes
- Programming language requirements
    - Inter-process communication mechanism
    - Language

5.1 MEDIA AS TYPES:

E.g. Programming expression used in OCCAM-2 which was derived from Communication Sequential Processes. This language is used for programming of transputers. This notation was choosed because of it's simplicity and embedded expressions of parallel behaviour.

a, b REAL;

ldu.left1, ldu.left2, ldu.left-mixed AUDIO_LDU;

……

WHILE

COBEGIN

PROCESS_1

Input (micro1, ldu.left1)

PROCESS_2

Input(micro2,ldu.left2)

ldu.left_mixed:=a*ldu.left1+b*ldu.left2;

……

END WHILE

……

One of the alternatives to programming is an HLL with libraries is the concept of media as types. In this example, there are 2 ldus from microphones that are read and mixed. Here, the data types for video and audio are defined. In the case of text, character is the type (the smallest addressable element). 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. Further, they can be displayed, be part of other data structures, etc.

5.2 MEDIA AS FILES

Another possibility of programming continuous media data is the consideration of continuous media streams as files instead of data types.

file_h1= open (MICROPHONE 1,...)

file h2= open(MICROPHONE 2,...)

tile_h3= open(SPEAKER,...)

....

read (file_h1)

read (file_h2)

mix(file_3, file_h1,file_h2)

activate (file_h1 , file_h2, file_h3)

....

deactivate (file_h1 , file h2, file_h3)

rc1= close (file_h1)

rc2= close (file_h2)

rc3= close (file_h3)

The example describes the merging of two audio streams. The physical file is associated during the open process of a file with a corresponding file name. The program receives a file descriptor through which the file is accessed. In this case, a device unit, which creates or processes continuous data streams, can be associated with a file name.

Read and write functions are based on continuous data stream behaviour. Therefore, a new value is assigned continuously to a specific variable which is connected, for example with one read function. On other hand, the read and write functions of discrete data occur in separate steps. For each assignment of a new value from a file to the corresponding variable, the read function is called again.

5.3 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 between source and destination. A connection can exist locally, as well as remotely. Under this consideration, it is possible to map continuous media to processes and to integrate them in an HLL.

PROCESSS cont_Process_a;

……

On_message_do

Set_Volume…….

Set_loudness…….

……..

…….

[main]

pid=create(cont_process_a)

send(pid, set_volume,3)

send(pid, set_loudness)

…….

In the above example, the process cont_process_a implements a set of actions which apply to a continuous data stream, two of them are the modification of volume set_volume and the process of setting a volume, dependent from a band filter, set_loudness.

During the creation of the process, the identification and reservation of the used physical devices occur. The different actions of the continuous process are controlled through an IPC mechanism.

5.4 PROGRAMMING LANGUAGE REQUIREMENTS

HLL should support parallel processing. Number of processes must be known as compile time. Process should be defined dynamically at run-time.

- Interprocess communication mechanism:
  - The IPC mechanism must be able to transmit the audio and video in a timely fashion because these media have a limited life span. The IPC must be able to:-
  - Understand a prior and/or implicitly specified time requirements.
  - Transmit the continuous data according to the requirements.
  - Initiate the processing of the received continuous process on time.
- Language
  - A simple language should be developed for the purpose of simplicity.
  - An example of such language is OCCAM-2, ADA, parallel C-variant for transputer etc.

5.5 INTERPROCESS COMMUNICATION MECHANISM

Different processes must be able to communicate through an Inter-Processes communication mechanism. This IPC mechanism must be able to transmit audio and video in a timely fashion because these media have a limited life span. Therefore, The IPC must be able to:-

- Understand a prior and/or implicitly specified time requirements.
- Transmit the continuous data according to the requirements.
- Initiate the processing of the received continuous process on time.

5.6 LANGUAGE

The authors see no demand for the development of a new dedicated language. A partial language replacement is also quite difficult because co-operation 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. For the purpose of simplicity, a simple language should be developed which satisfies most of the above described requirements. An example of such language is OCCAM-2.

6 OBJECT –ORIENTED APPROACHES

The object-oriented approach was first introduced as a method for the reduction of complexity in the software development and it is used mainly with this goal today. Further, the reuse of software components is a main advantage of this paradigm. The basic ideas of object-oriented programming are: data encapsulation and inheritance, in connection with class and object definitions. The programs are implemented, instead of using functions and data structures, by using classes, objects, and methods.

6.1 CLASS

A class represents a collection of objects having same characteristic properties that exhibit common behaviour. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

Example:

Let us consider a simple class, *Circle,* that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows –

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows –

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

6.2 OBJECT

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has –

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behaviour that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

6.3 INHERITANCE

The concept allows us to inherit or acquire the properties of an existing class (parent class) into a newly created class (child class). It is known as inheritance. It provides code reusability.

The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an “is – a” relationship.

Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow “is – a” mammal.

6.4 POLYMORPHISM

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example

Let us consider two classes, Circle and Square, each with a method findArea(). Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of

calculating area is different for each class. When an object of class Circle invokes its findArea() method, the operation finds the area of the circle without any conflict with the findArea() method of the Square class.

[Note: see book for more references]

6.5 APPLICATION-SPECIFIC METAPHORS AS CLASSES

Multimedia metaphor is a set of user interface visuals, actions and procedures that exploit specific knowledge that users already have of other domains. An application-specific class hierarchy introduces abstraction specifically designed for a particular application. Thus, it is not necessary to consider other class hierarchies. Using this approach, one very easily abandons the actual advantages of object-oriented programming, i.e. the reuse of existing code.

6.6 APPLICATION-GENERIC METAPHORS AS CLASSES

This approach is to combine similar functionalities of all applications. This properties or functions which occur repeatedly 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. The methods of these classes inherit the general methods through integration of application-specific subclasses.

6.7 DEVICES AS CLASSES

In this section we consider objects which reflect a physical view of the multimedia system. 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 considered methods use internally, for example, methods like start, stop and seek. Some units can manipulate several media together.

A computer-controlled VCR or a Laser Disc Player (LDP) are storage units which, by themselves, integrate (bind) video and audio. In a multimedia system, abstract device definitions can be provided, e.g., camera and monitor. We did not say anything until now about the actual implementation. The results show that defining a general and valid interface for several similar audio and video units, as well as input and output units, is quite a difficult design process.

6.8 PROCESSING UNITS AS CLASSES

This abstraction comprises 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 ofa dataflow path through a connection of objects. The outputs of objects are connected with inputs of other objects, either directly or through channel objects.

6.9 DISTRIBUTION OF BMOS AND CMOS

In a multimedia environment, the data elements are more complex, taking the form of video, voice, text, images and may be real time in nature or can be gathered from a stored environment. More importantly, the separate data objects may combined into more complex forms so that the users may want to create new objects by concatenating several simpler objects into a complex whole. Thus, we can conceive of a set of three objects composed of an image, a voice annotation and a pointer motion

annotating the voice annotation. The combination of all three of these can also be viewed as a single identifiable multimedia object.

We can consider a multimedia data object to be composed of several related multimedia data objects which are a voice segment, an image and a pointer movement (e.g. mouse movement). As we have just described, these can be combined into a more complex object. We call the initial objects Simple Multimedia Objects (SMOs) and the combination of several a Compound Multimedia Object (CMO). In general a multimedia communications process involves one or multiple SMOs and possibly several CMOs.

The SMO contains two headers that are to be defined and a long data string. The data string , we call a Basic Multimedia Object (BMO). There may be two types of BMOs. The first type we call a segmented BMO or SG:BMO. It has a definite length in data bits and may result from either a stored data record or from a generated record that has a natural data length such as a single image, screen or text record. We show the SMO. The second type of BMO is a streamed BMO, ST:BMO. This BMO has an a priori undetermined duration. Thus it may be a real time voice or video segment.

A CMO has two headers, the Orchestration header and the Concatenation header. The Orchestration header describes the temporal relationship between the SMOs and ensures that they are not only individually synchronized but also they are jointly orchestrated. The orchestration concept has also been introduced by Nicolaou. The concatenation function provides a description of the logical and spatial relationships amongst the SMOs.

We can also expand the concept of a CMO as a data construct that is created and managed by multiple users at multiple locations. In this construct we have demonstrated that N users can create a CMO by entering multiple SMOs into the overall CMO structure. The objectives of the communications system are thus focused on meeting the interaction between users who are communicating with CMOs. Specifically we must be able to perform the following tasks:

- Allow any user to create an SMO and a CMO.
- Allow any user or set of users to share, store, or modify a CMO.
- Ensure that the user to user communications preserves the temporal, logical and spatial
- relationships between all CMOs at all users at all times.
- Provide an environment to define, manage and monitor the overall activity.
- Provide for an environment to monitor, manage and restore all services in the event of system failures or degradation.

6.10 MEDIA AS CLASSES

The Media Class Hierarchy defines a hierarchical relation for different media. The following example shows such a class hierarchy. Thus, the individual methods of the class hierarchy will not be described. The class Pixel in the class hierarchy uses, for example, multiple inheritance.

Medium

AcousticMedium

Music

Opus

Note

Audio_Block

Sample Value

Speech

…….

…….

Optical_Medium

Video

Video_Scene

Image

Image_Segment

Pixel

Line

Pixel

Column

Pixel

Animation

……

Text

…….

other-continuous _medium

discrete_medium

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 activated.

6.11 COMMUNICATION-SPECIFIC METAPHORS AS CLASSES

Communication-oriented approaches often consider objects in a distributed environment through an explicit specification of classes and objects tied to a communication system. Blakowski specifies, information, presentation and transport classes.

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 which depends on whether the information should be presented, transmitted or stored.