Module 1

CS 404 Embedded Systems

Contents

- ***Fundamentals of Embedded Systems***
- ***Complex systems and microprocessors***
- ***Embedded system design process***
  - *Specifications*
  - *Architecture design of embedded system*
  - *Design of hardware and software components*
- ***Structural and behavioural description***

Fundamentals of Embedded Systems

- Definition of a system
- Embedded system definition
- Comparison between computer and embedded system
- Characteristics of Embedded System
- Constraints of Embedded System Design

System Definition

- A *way of working*, *organizing* or *performing one or many tasks* according to a *fixed* ***set of rules***, *program or plan*.
- Also an *arrangement* in which all *units assemble* and *work together* according to a program or plan.
- Examples of Systems
  - Time display system – A watch
  - Automatic cloth washing system – A washing machine

Embedded System Definitions

1. *"An embedded system is a system that has software embedded into computer-hardware, which makes a system dedicated for an application(s) or specific part of an application or product or part of a larger system."*
2. *"An embedded system is one that has a dedicated purpose software embedded in a computer hardware."*
3. *"It is a dedicated computer based system for an application(s) or product. It may be an independent system or a part of large system. Its software usually embeds into a ROM (Read Only Memory) or flash."*

Embedded System Definitions

4. "*It is any device that includes a programmable computer but is not itself intended to be a general purpose computer*." – ***Wayne Wolf***
5. "*Embedded Systems are the electronic systems that contain a microprocessor or a microcontroller, but we do not think of them as computers– the computer is hidden or embedded in the system*." – ***Todd D. Morton***

Complex systems and microprocessors

Characteristics of Embedded Computing Application

- Embedded computing systems have to provide sophisticated functionality:
  - ***Complex algorithms:*** The operations performed by the microprocessor may be very sophisticated.
  - ***User interface:*** sophisticated user interfaces
  - ***Real time :*** have to perform in real time. If the data is not ready by a certain deadline, the system breaks.
  - ***Multirate:*** They may simultaneously control some operations that run at slow rates and others that run at high rates. Eg: Multimedia application.
- Costs of various sorts :
  - ***Manufacturing cost***
  - ***Power and energy :*** Power consumption directly affects the cost of the hardware. Energy consumption affects battery life.

Consider a computer

- A computer is a system that has the following or more components.
  - A microprocessor
  - A large memory comprising the following two kinds:
    - Primary memory (*semiconductor* memories - RAM, ROM and fast accessible caches)
    - Secondary memory using which different user programs can load into the primary memory and can be run.
  - I/O units such as touch screen, modem, fax cum modem etc.

Consider an Embedded System

- **Three main embedded components**

1. **Embeds hardware** to give computer like functionalities
2. **Embeds main application software** generally into flash or ROM and the application software performs concurrently the number of tasks.
3. **Embeds a real time operating system (RTOS),** which supervises the application software tasks running on the hardware and organizes the accesses to system resources according to priorities and timing constraints of tasks in the system.

Criteria	General Purpose Computer	Embedded system
Contents	It is combination of generic hardware and a general purpose OS for executing a variety of applications.	It is combination of special purpose hardware and embedded OS for executing specific set of applications
Operating System	It contains general purpose operating system	It may or may not contain operating system.
Alterations	Applications are alterable by the user.	Applications are non-alterable by the user.
Key factor	Performance" is key factor.	Application specific requirements are key factors.
Power Consumption	More	Less
Response Time	Not Critical	Critical for some applications

Sophisticated Embedded System Characteristics

1. Dedicated functions
2. Dedicated complex algorithms
3. Dedicated GUIs and other user interfaces for the application
4. Real time operations
   - Defines the ways in which the system works, reacts to the events and interrupts, schedules the system functioning in real time and executes by following a plan to control the latencies and to meet the deadlines.
5. Multi-rate operations
   - Different operations may take place at distinct rates.
     - For example, the audio, video, network data or stream and events have the different rates and time constraints to finish associated processes.

Constraints of an Embedded System Design

- Available system-memory
- Available processor speed
- Limited power dissipation when running the system continuously in cycles of the system start, wait for event, wake-up and run, sleep and stop.
- Performance
- Size
- Non-recurring design cost, and
- Manufacturing costs.

Real Time operation

- Define the way in which the system works, reacts to events, interrupts and schedules the system's functioning in real time.
  - Must finish the operations by deadlines
    - **HARD REAL TIME**: Missing deadline causes failure
    - **SOFT REAL TIME**: Missing deadline results in degraded performance
  - Must handle multi-rate operations

Challenges in Embedded System Design

- Amount and type of hardware needed
  - Taking into account the design metrics [cost vs. performance]
- Optimizing the Power Dissipation
  - Disable use of certain structural units of the processor to reduce power dissipation
  - Clock Rate reduction
  - Voltage reduction
  - Wait, stop and cache disable instructions

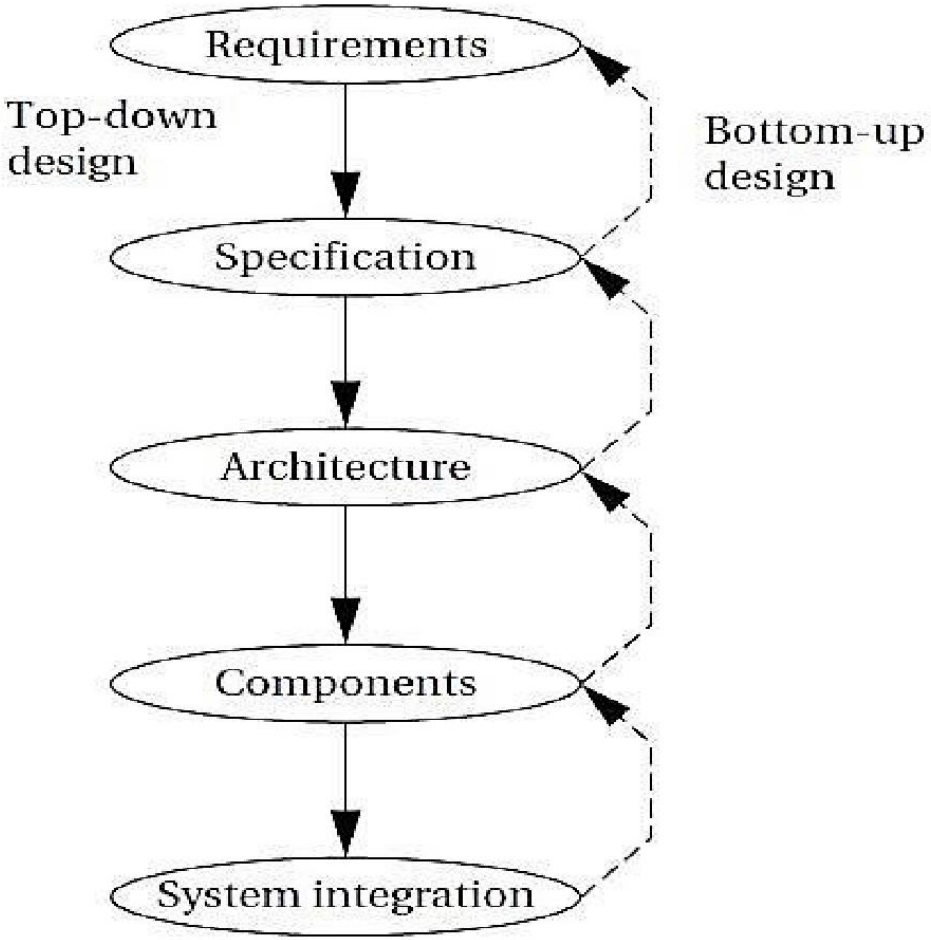
Challenges in Embedded System Design

- Process Deadlines
- Flexibility and Upgradeability
- Reliability
- Testing, Verification and Validation

Challenges in Embedded System Design

- *How much hardware do we need?*
- *How do we meet deadlines?*
- *How do we minimize power consumption?*
- *How do we design for upgradability?*
- *Does it really work?*

Embedded system design process



Requirements

- In first step we gather informal description from customers known as requirements, we then refine that requirement into a specification that contain enough information to start design.
- Requirements are of 2 types :

1. Functional Requirements

   - focused on function of the system

2. Non functional Requirements

   - performance
   - cost
   - physical size and weight
   - Power consumption

- Requirement analysis of a large system is complex and time consuming. The following figure shows a sample requirement analysis form that need to filled during the start of a project.

Sample requirements form.

Name

Purpose

Inputs

Outputs

Functions

Performance

Manufacturing cost

Power

Physical size and weight

- **Name :** Selection of name that depicts the theme of project
- **Purpose :** One or two line description of what the system is supposed to do
- **Inputs and Outputs**: Type of data, analog digital or mechanical input or output
- **Functions**: Is a description of what the system does
- **Performance:** Measure of the system work precisely?
- **Manufacturing cost**: Cost of hardware, application software including all the cost involved in the manufacturing
- **Power**: Measure of how much power the system consume
- **Physical size and weight**: It guided to certain architectural design

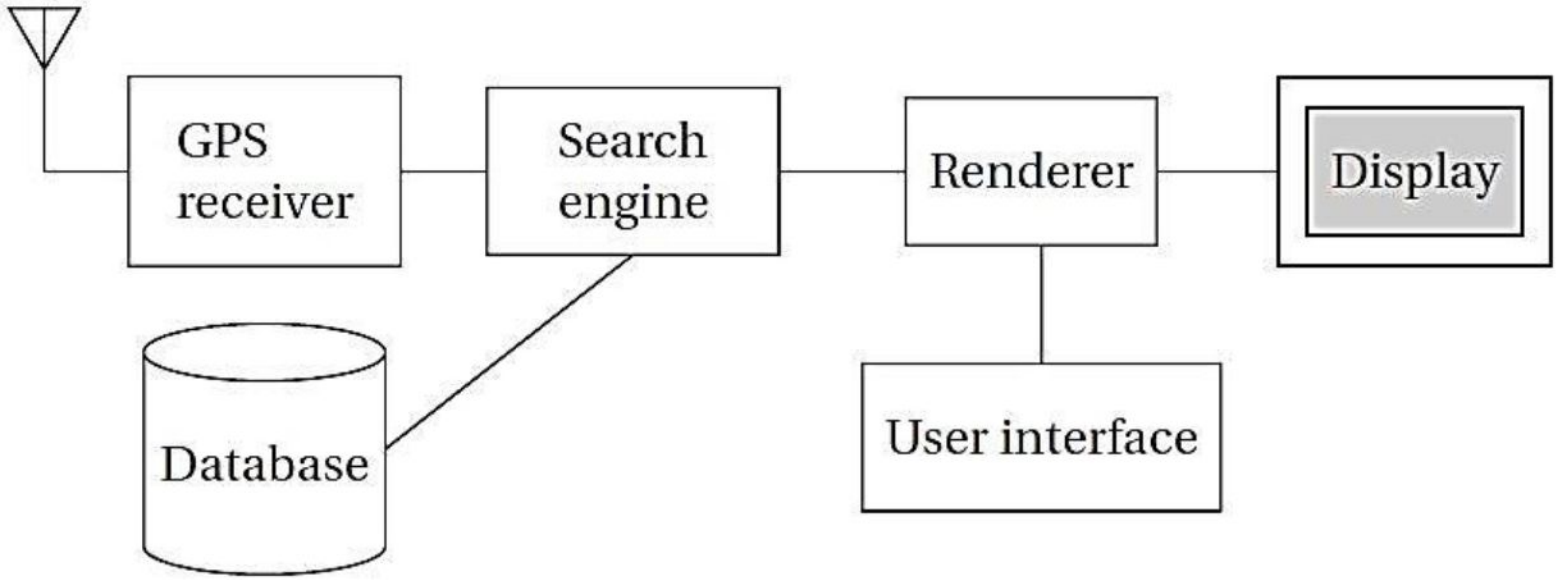
Specifications

- The specification is more precise—it serves as the **contract between the customer and the architects**.
- The specification must be carefully written so that it accurately reflects the customer's requirements and does so in a way that can be clearly followed during design.
- The specification should be **understandable** enough so that someone can verify that it meets system requirements and overall expectations of the customer.
- It should also be **unambiguous** enough that designers know what they need to build.
- If global characteristics of the specification are wrong or incomplete, the overall system architecture derived from the specification may be inadequate to meet the needs of implementation.

Architecture design of embedded system

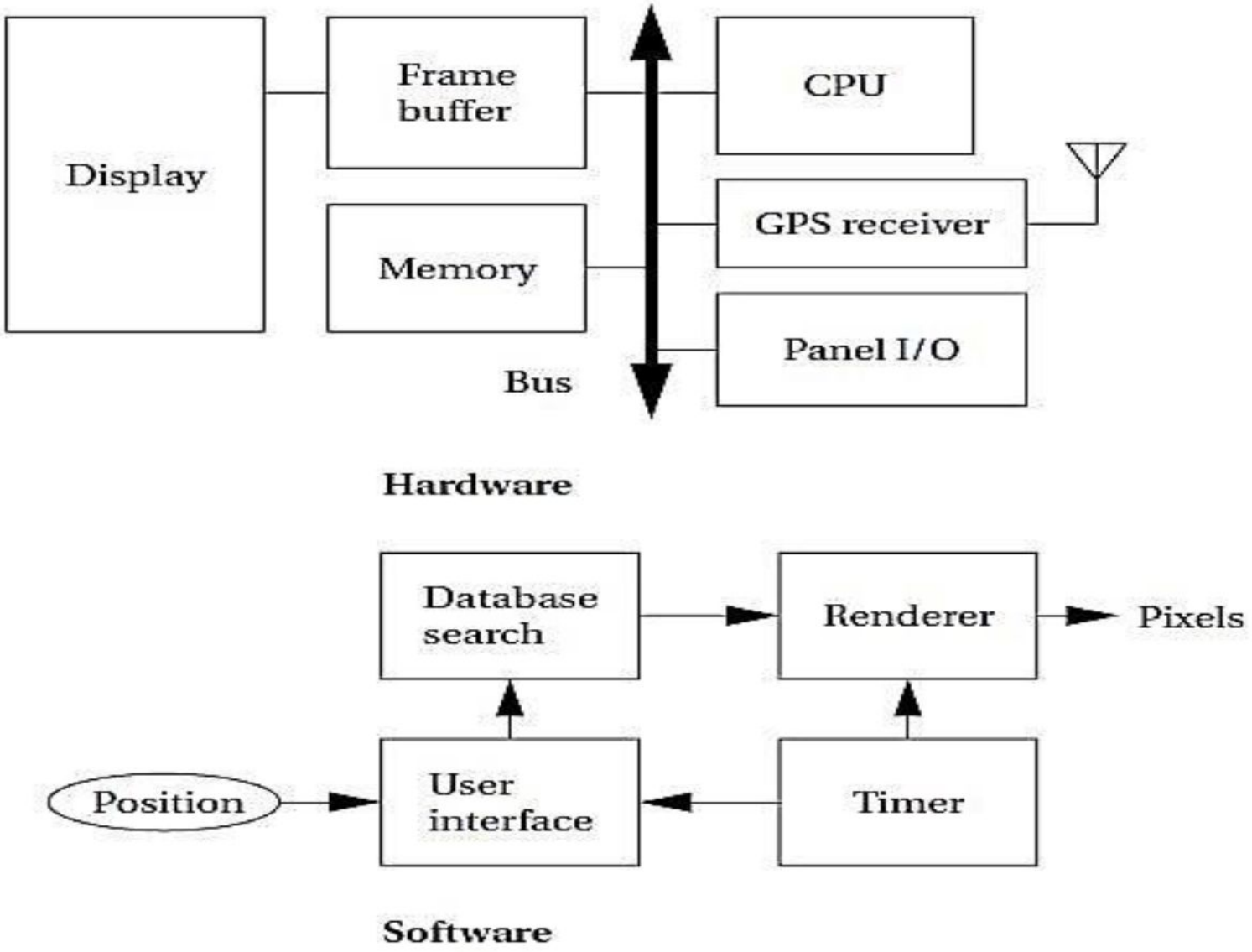
- The specification does not say how the system does things, only what the system does.
- Describing how the system implements those functions is the purpose of the architecture.
- The **architecture is a plan for the overall structure of the system** that will be used later to design the components that make up the architecture.
- The creation of the architecture is the first phase of what many designers think of as design.

Block diagram for the moving map



- This block diagram is still quite abstract—we have not yet specified which operations will be performed by software running on a CPU, what will be done by special-purpose hardware, and so on.

Hardware and software architectures for the moving map.



Design of hardware and software components

- The architectural description tells us what components we need.
- The components will in general include both hardware—FPGAs, boards, and so on—and software modules.
- Some of the components will be **ready-made**. The CPU, for example, will be a standard component in almost all cases, as will memory chips and many other components.
- In the moving map, the **GPS receiver** is a good example of a **specialized component** that will nonetheless be a predesigned, standard component. We can also make use of standard software modules.

Unified Modeling Language (UML)

- UML was designed to be useful at many levels of abstraction in the design process.
- UML is useful because it encourages design by successive refinement and progressively adding detail to the design, rather than rethinking the design at each new level of abstraction.
- UML is an object-oriented modeling language.

Structural Description

- By structural description, we mean the basic components of the system.
- The principal component of an object-oriented design is the object.
- An object includes a set of attributes that define its internal state.
- When implemented in a programming language, these attributes usually become variables or constants held in a data structure.
- In some cases, we will add the type of the attribute after A class is a form of type definition—all objects derived from the same class have the same characteristics, although their attributes may have different values.
- A class defines the attributes that an object may have. It also defines the operations that determine how the object interacts with the rest of the world.
- In a programming language, the operations would become pieces of code used to manipulate the object

The UML description of the Display class

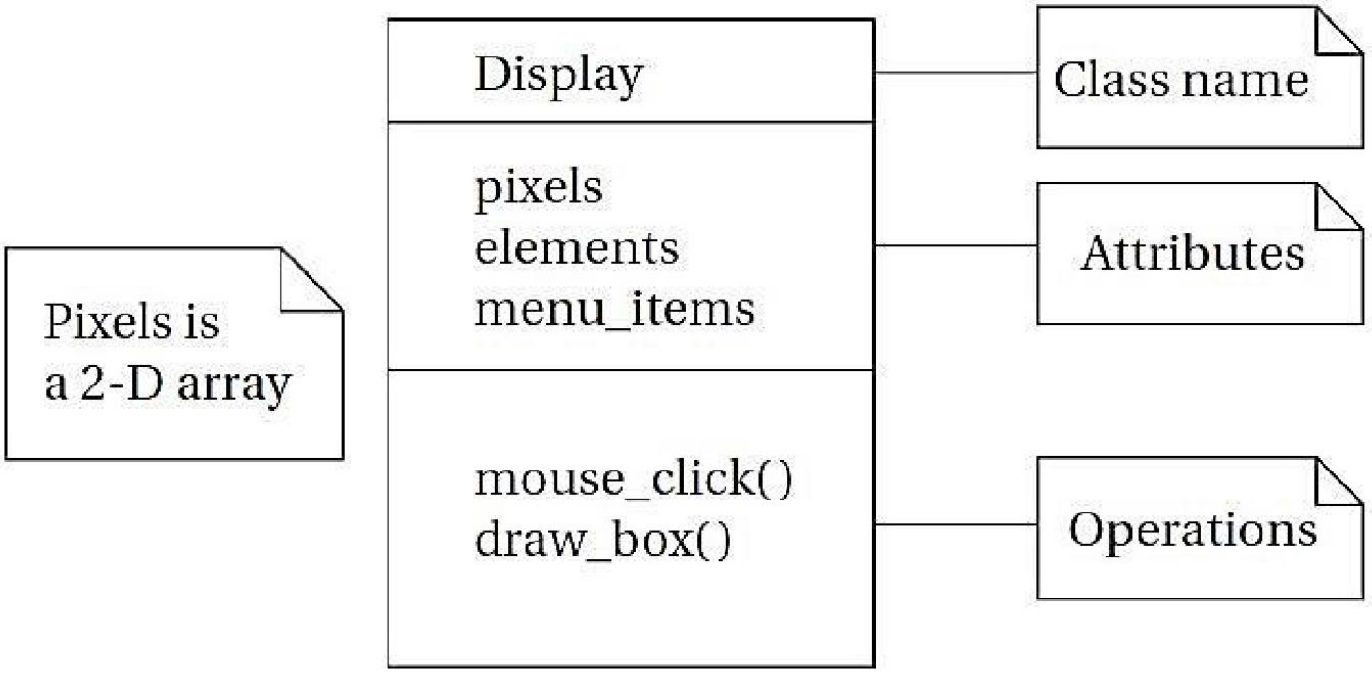
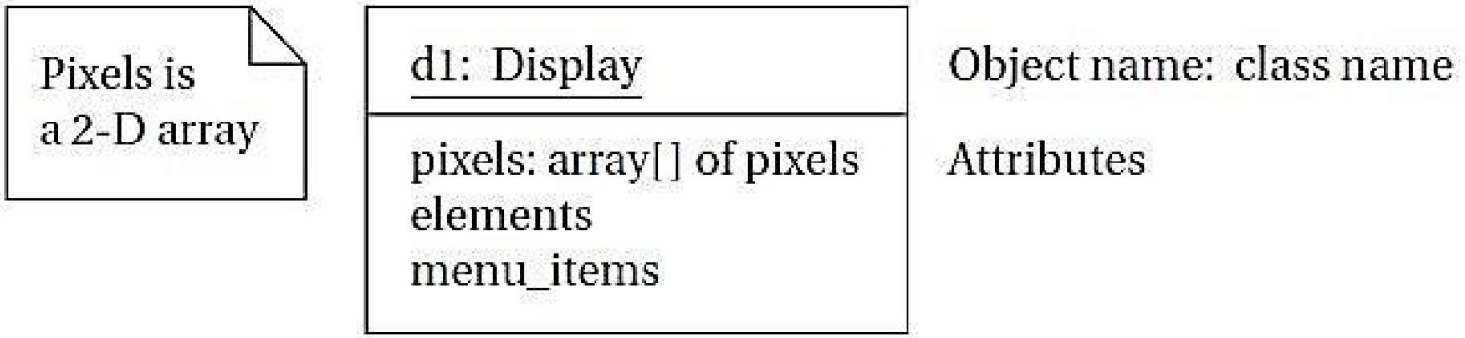


Fig: A class in UML notation.



An object in UML notation

- The class has the name that we saw used in the d1 object since d1 is an instance of class **Display**.
- The **Display class** defines the pixels attribute seen in the object; remember that when we instantiate the class an object, that object will have its own memory so that different objects of the same class have their own values for the attributes.
- Other classes can examine and modify class attributes; if we have to do something more complex than use the attribute directly, we define a behavior to perform that function.
- A class defines both the **interface** for a particular type of object and that object's **implementation.**
- When we use an object, we do not directly manipulate its attributes—we can only read or modify the object's state through the operations that define the interface to the object

Relationships that can exist between objects and classes:

- **Association** occurs between objects that communicate with each other but have no ownership relationship between them.
- **Aggregation** describes a complex object made of smaller objects.
- **Composition** is a type of aggregation in which the owner does not allow access to the component objects.
- **Generalization** allows us to define one class in terms of another.

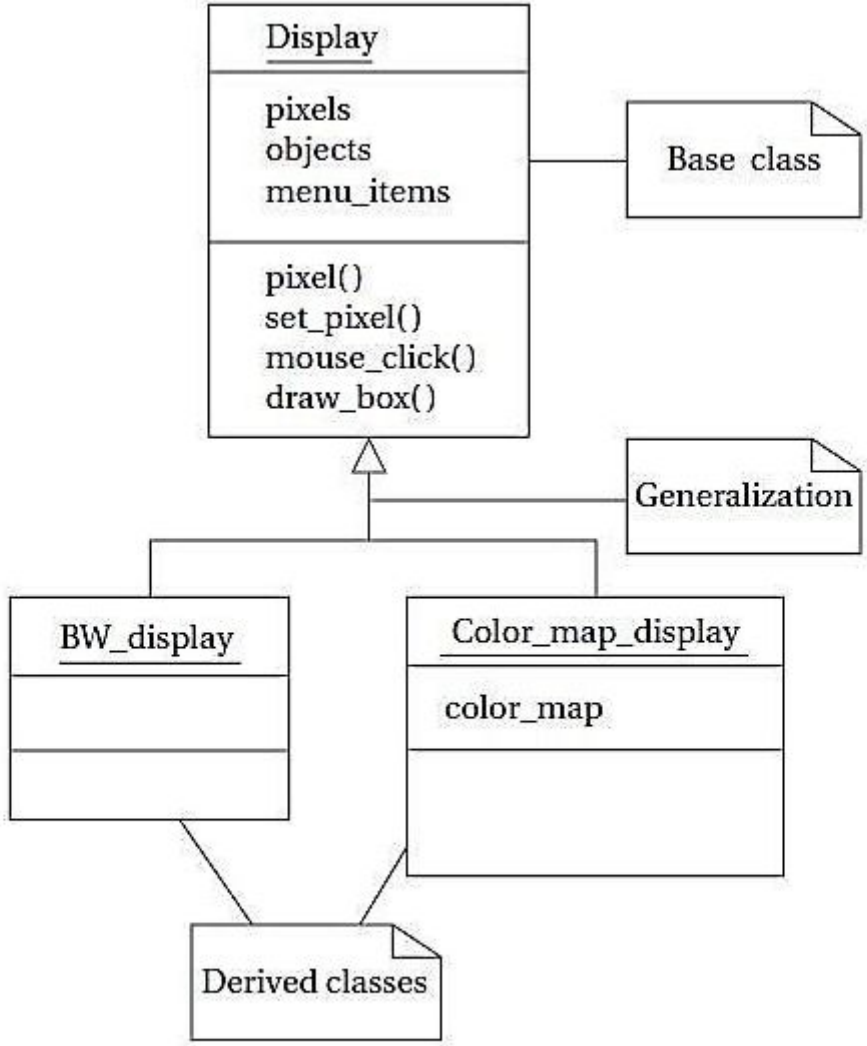


Fig: Derived classes as a form of generalization in UML.

- A ***derived class*** *inherits all the attributes and operations from its base class.*

- In this class, *Display is the base class for the two derived classes. A derived class is defined to include all the attributes of its base class.*

- This relation is transitive—if *Display were derived from another class, both* ***BW_display*** *and* ***Color_map_display*** *would inherit all the attributes and operations of Display's base class as well.*

- **Unified Modeling Language** considers inheritance to be one form of generalization. A generalization relationship is shown in a UML diagram as an arrow with an open (unfilled) arrowhead.

- UML also allows us to define ***multiple inheritances,*** *in which a class is derived from more than one base class.*

- In the figure below we have created a ***Multimedia_display*** *class by combining the* ***Display*** *class with a* ***Speaker*** *class for sound.*
- The derived class inherits all the attributes and operations of both its base classes, *Display and Speaker.*
- A ***link*** *describes a relationship between objects; association is to link as class is to object.*

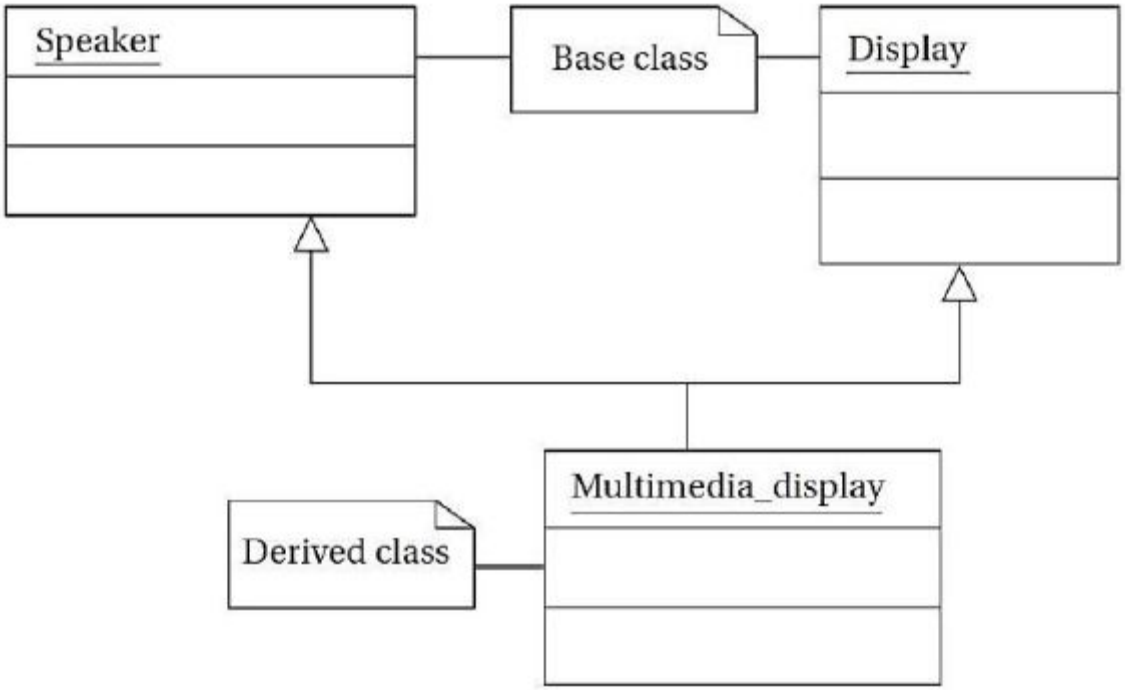
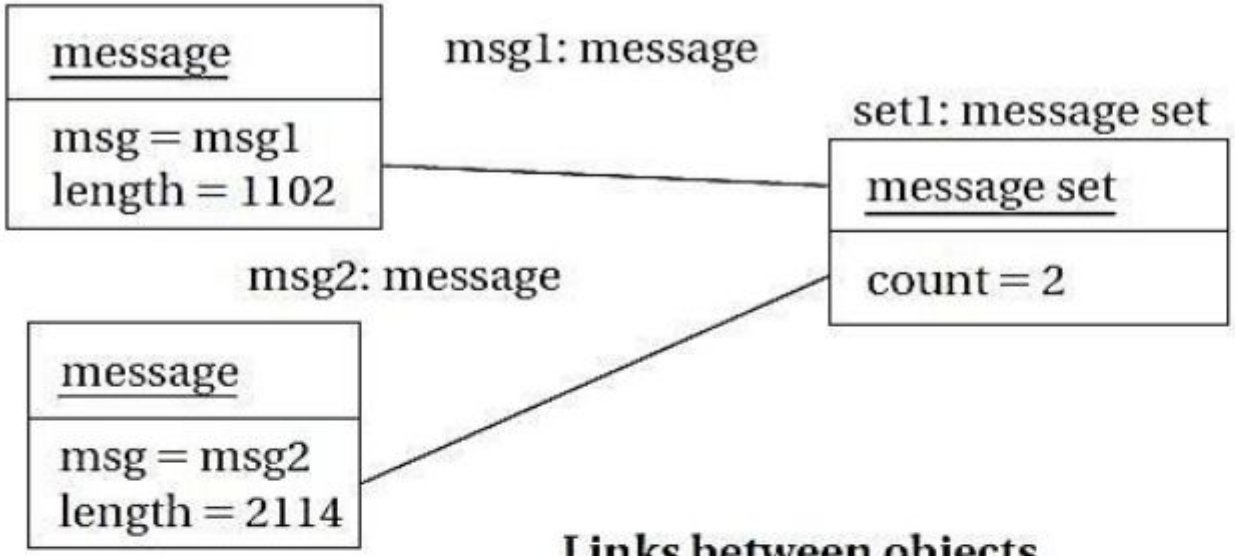
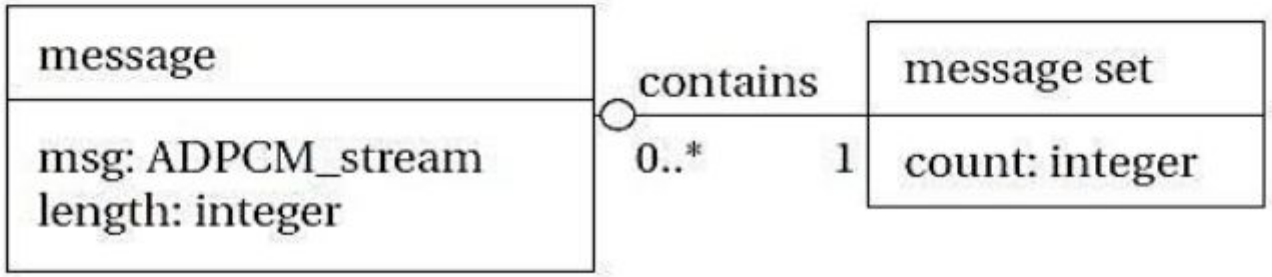


Fig: Multiple inheritances in UML.

- Following figure shows examples of links and an association



Links between objects



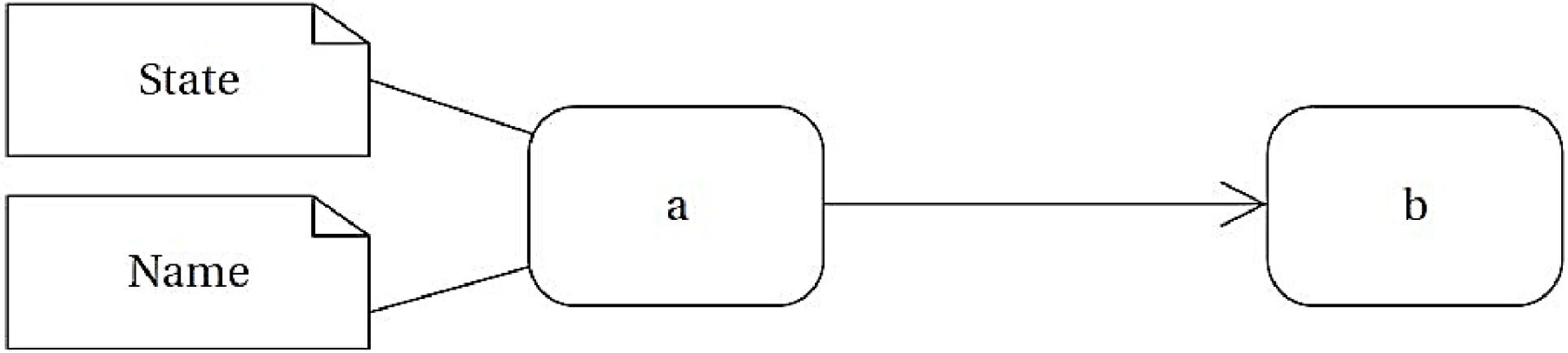
Association between classes

- When we consider the actual objects in the system, there is a set of messages that keeps track of the current number of active messages (two in the above example) and points to the active messages. In this case, the link defines the ***contains*** *relation****.***

- The association is drawn as a line between the two labeled with the name of the association, namely, ***contains.***

Behavioral Description

- We have to specify the behavior of the system as well as its structure. One way to specify the behavior of an operation is a **state machine**.



A state and transition in UML

- The state machines will not rely on the operation of a clock, as in hardware; rather, changes from one state to another are triggered by the occurrence of **events**.
- An event is some type of action. The event may originate outside the system, such as a user pressing a button. It may also originate inside, such as when one routine finishes its computation and passes the result on to another routine.

The three types of events defined by UML

- A ***signal*** is an asynchronous occurrence. It is defined in UML by an object that is labeled as a **<<*signal*>>**. The object in the diagram serves as a declaration of the event's existence. Because it is an object, a signal may have parameters that are passed to the signal's receiver.
- A ***call event*** follows the model of a procedure call in a programming language.
- A ***time-out event*** causes the machine to leave a state after a certain amount of time. The label ***tm(time-value)*** on the edge gives the amount of time after which the transition occurs. A time-out is generally implemented with an external timer. This notation simplifies the specification and allows us to defer implementation details about the time-out mechanism

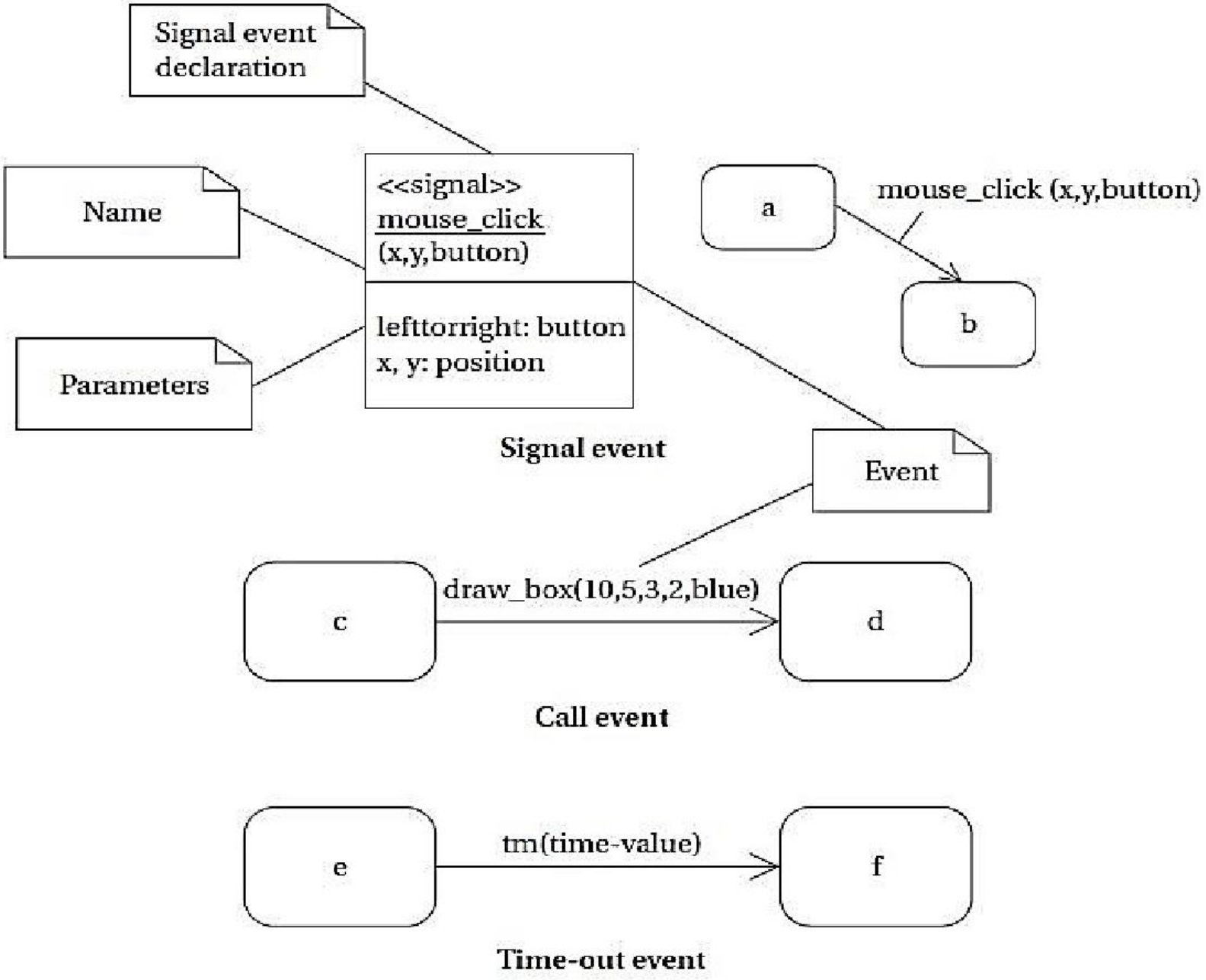
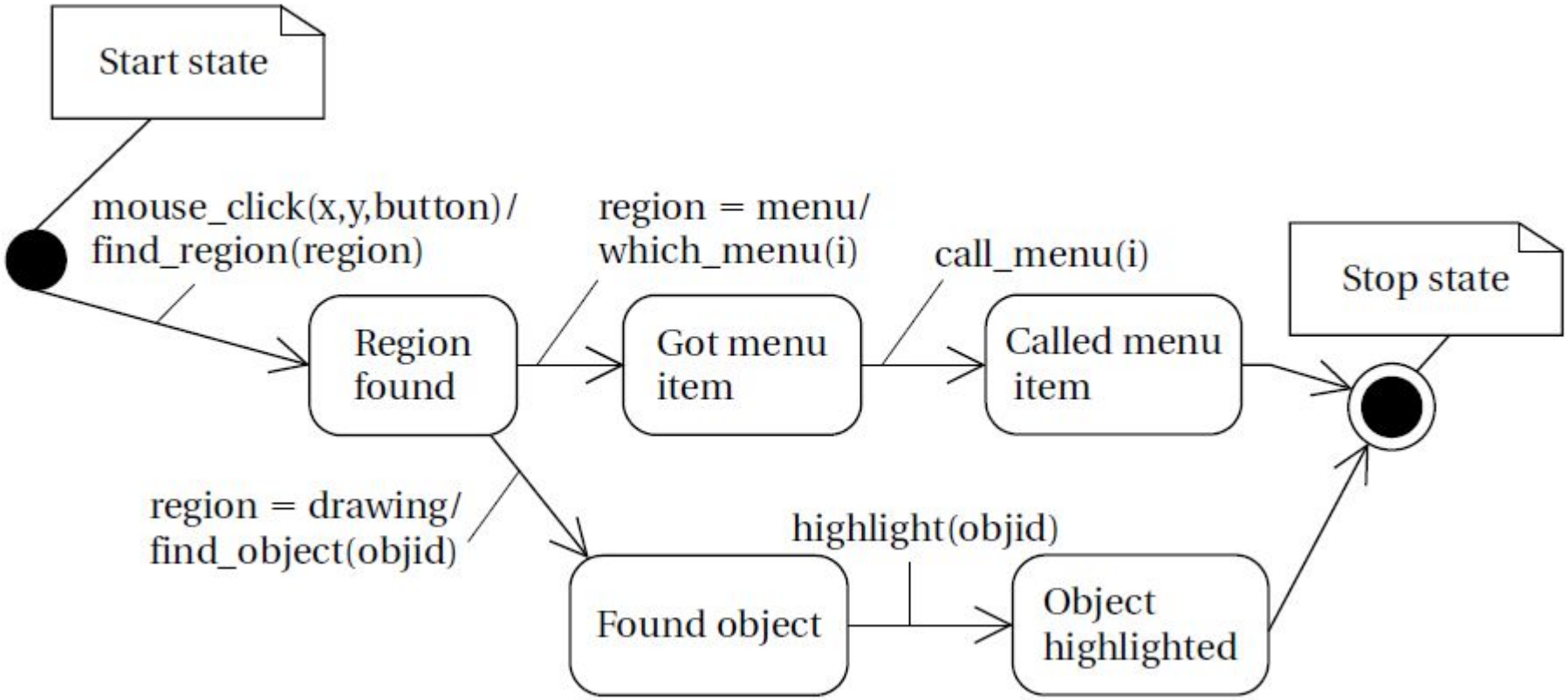


Fig: Signal, call, and time-out events in UML.

- Let’s consider a simple **state machine specification** to understand the semantics of UML state machines.
- A state machine for an operation of the display is shown in Fig.
- The start and stop states are special states that help us to organize the flow of the state machine.

- Fig: A state machine specification in UML



- The states in the state machine represent different conceptual operations.
- In some cases, we take conditional transitions out of states based on inputs or the results of some computation done in the state.
- In other cases, we make an unconditional transition to the next state.
- Both the unconditional and conditional transitions make use of the call event.
- Splitting a complex operation into several states helps document the required steps, much as subroutines can be used to structure code.

Sequence diagram

- To show the sequence of operations overtime, particularly when several objects are involved, we can create a **sequence diagram**.
- Sequence diagram for a mouse click scenario shown in Fig.
- A sequence diagram is some what similar to a hardware timing diagram.
- **Time flows vertically in a sequence diagram**, whereas time typically flows horizontally in a timing diagram.
- The sequence diagram is designed to show a particular scenario or choice of events — it is not convenient for showing a number of mutually exclusive possibilities.

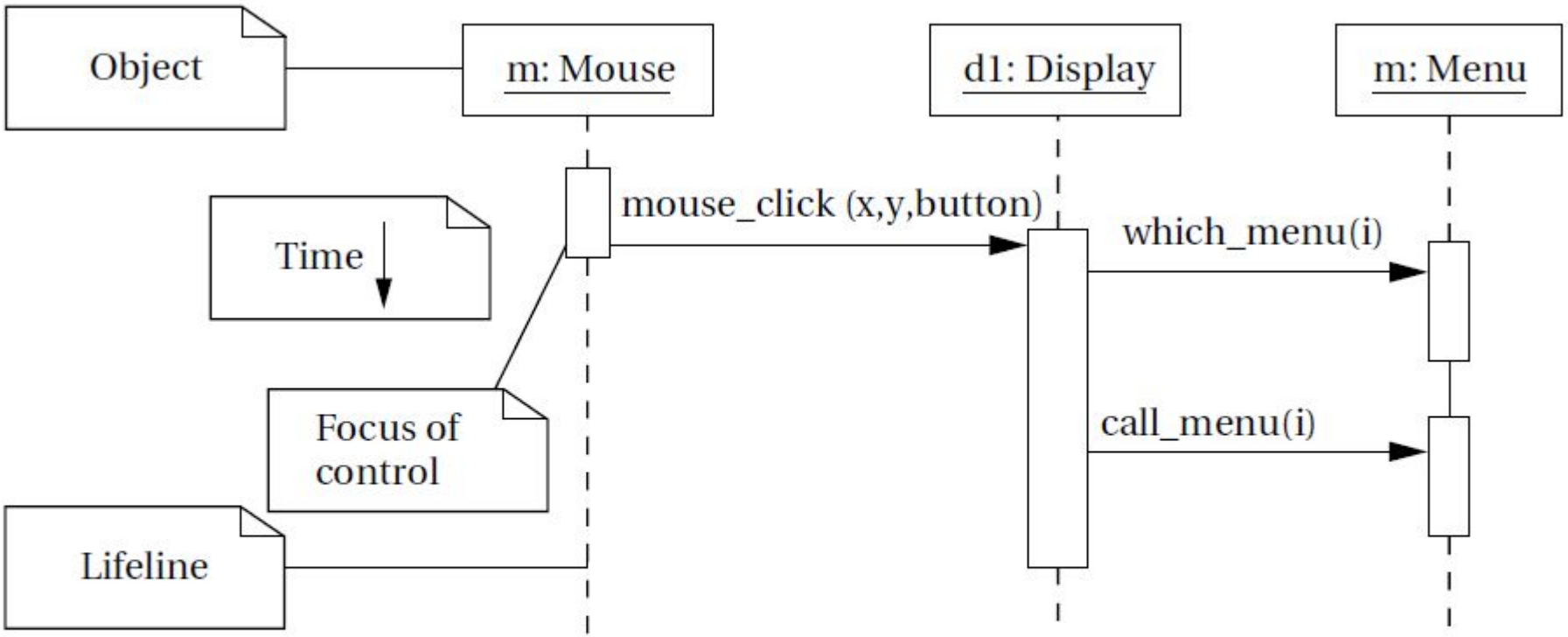


FIGURE 1.13

A sequence diagram in UML.

- In the fig, the sequence shows what happens when a mouse click is on the menu region. Processing includes three objects shown at the top of the diagram.
- Extending below each object is its **lifeline**, a dashed line that shows how long the object is alive.
- In this case, all the objects remain alive for the entire sequence, but in other cases objects may be created or destroyed during processing.

- The boxes along the life lines show the **focus of control** in the sequence, that is, when the object is actively processing.
- In this case, the mouse object is active only long enough to create the mouse_click event.
- The display object remains in play longer; it inturn uses call events to invoke the menu object twice: once to determine which menu item was selected and again to actually execute the menu call.
- The find_region() call is internal to the display object, so it does not appear as an event in the diagram.