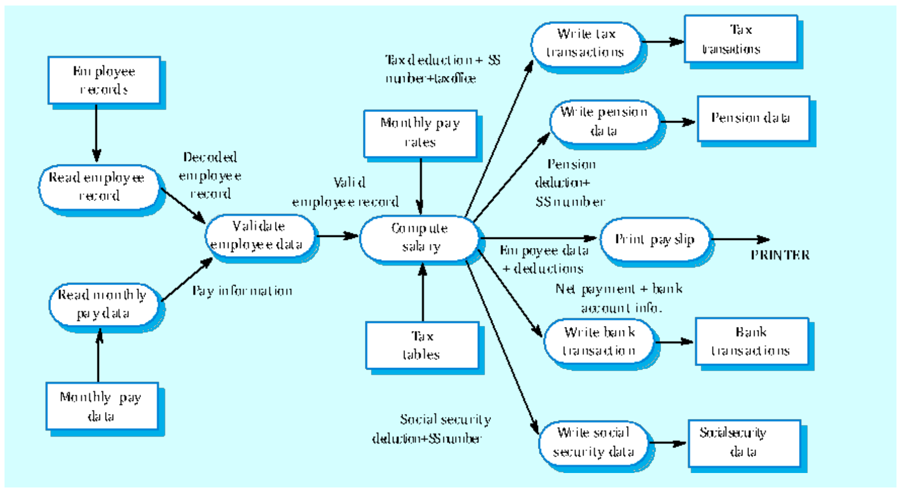
ELEC3609 Week 03
Architecture for Web Applications

- **Data processing applications**
  - Data driven applications that process data in batches without explicit user intervention during the processing.
  - Billing systems;
  - Payroll systems.
- **Transaction processing applications**
  - Data-centred applications that process user requests and update information in a system database.
  - E-commerce systems;
  - Reservation systems.

Use Data Flow Diagrams or Activity Diagrams:

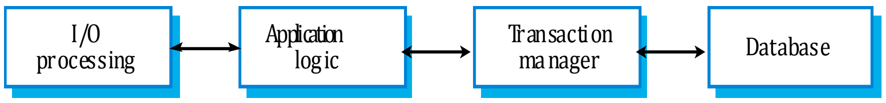
- Show how data is processed as it moves through a system. In DFD transformations are represented as round-edged rectangles, data-flows as arrows between them and files/data stores as rectangles.

Salary payment DFD

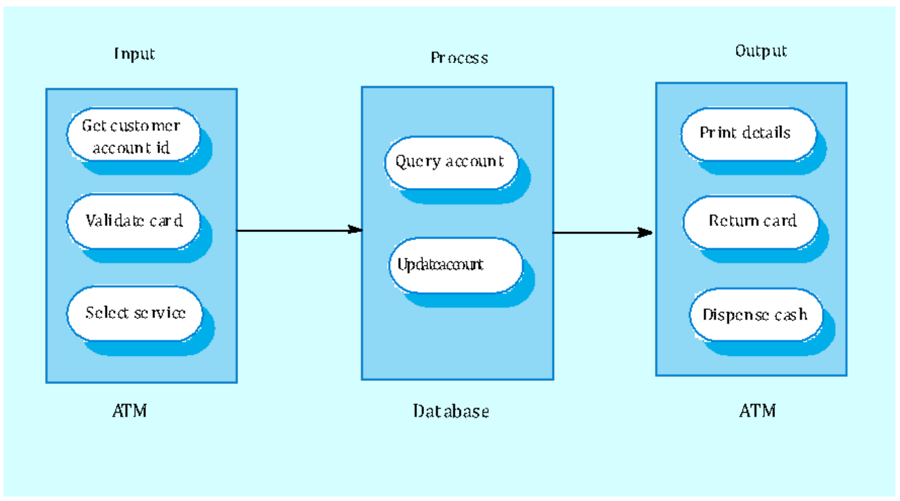


Transaction processing systems

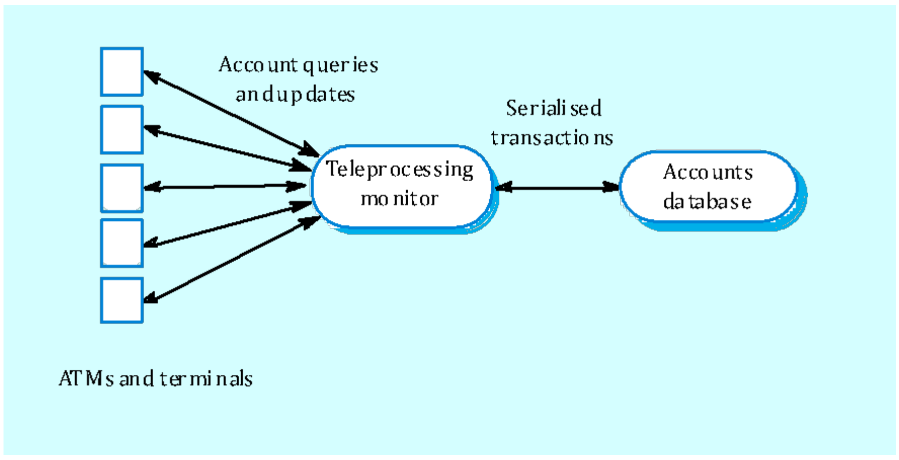
- Process user requests for information from a database or requests to update the database.
- From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- Users make asynchronous requests for service which are then processed by a transaction manager.



ATM system organisation



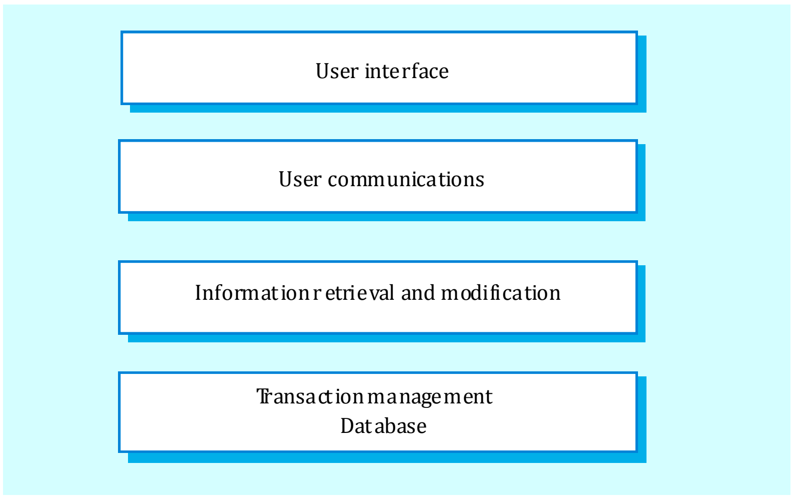
Transaction management



Information systems architecture

- Information systems have a generic architecture that can be organised as a layered architecture.
- Layers include:
  - The user interface
  - User communications
  - Information retrieval
  - System database
    - Information retrieval and System database are often handled as one layer

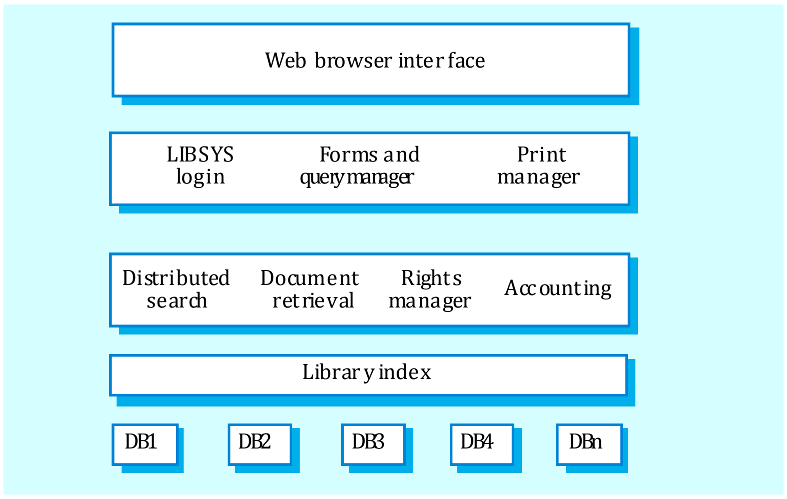
Information system structure



LIBSYS architecture

- The library system LIBSYS is an example of an information system.
- User communications layer:
  - LIBSYS login component;
  - Form and query manager;
  - Print manager;
- Information retrieval layer
  - Distributed search;
  - Document retrieval;
  - Rights manager;
  - Accounting.

LIBSYS organisation



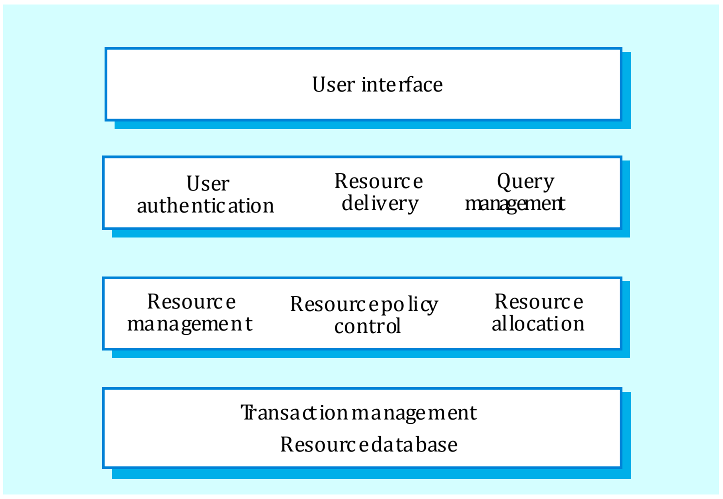
Resource allocation systems

- Systems that manage a fixed amount of some resource (football game tickets, books in a bookshop, etc.) and allocate this to users.
- Examples of resource allocation systems:
  - Timetabling systems where the resource being allocated is a time period;
  - Library systems where the resource being managed is books and other items for loan;
  - Air traffic control systems where the resource being managed is the airspace.

Resource allocation architecture

- Resource allocation systems are also layered systems that include:
  - A resource database;
  - A rule set describing how resources are allocated;
  - A resource manager;
  - A resource allocator;
  - User authentication;
  - Query management;
  - Resource delivery component;
  - User interface.

Layered resource allocation



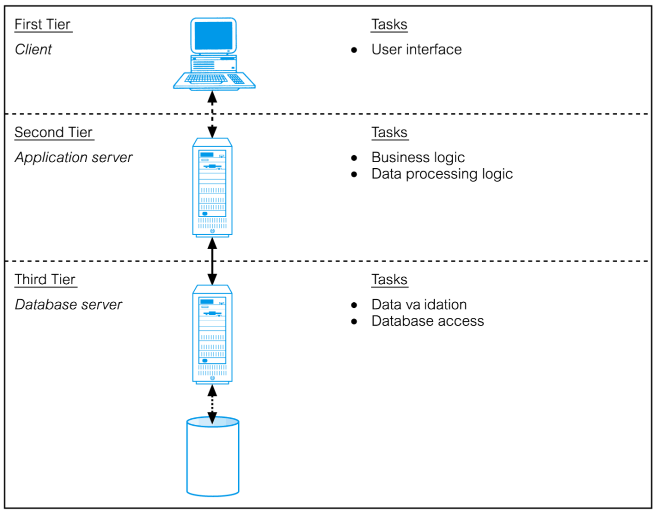
Layered system implementation

- Each layer can be implemented as a large scale component running on a separate server. This is the most commonly used architectural model for web-based systems.
- On a single machine, the middle layers are implemented as a separate program that communicates with the database through its API.
- Fine-grain components within layers can be implemented as web services.

Three-Tier Client-Server

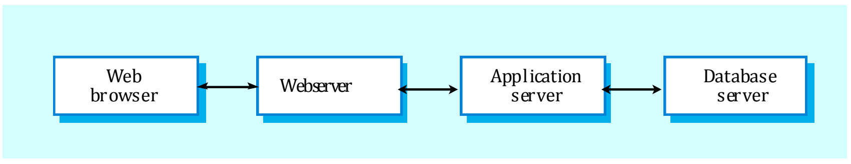
- **Advantages:**
  - **'Thin' client, requiring less expensive hardware.**
  - **Application maintenance centralized.**
  - **Easier to modify or replace one tier without affecting others.**
  - **Separating business logic from database functions makes it easier to implement load balancing.**
  - **Maps quite naturally to Web environment.**

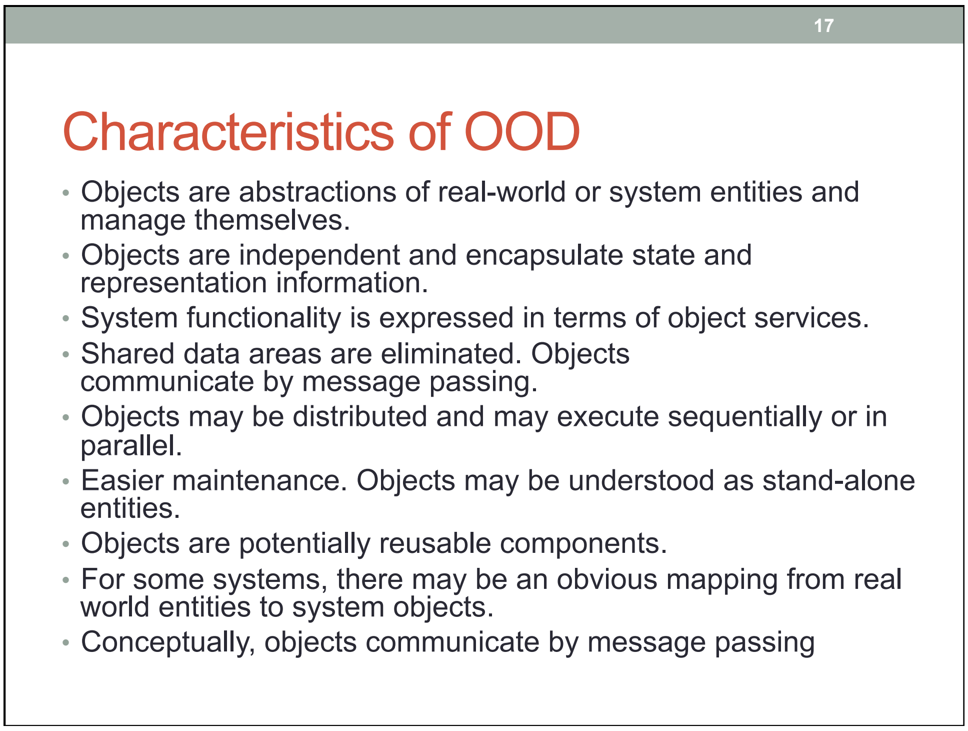
Three-Tier Client-Server



E-commerce system architecture

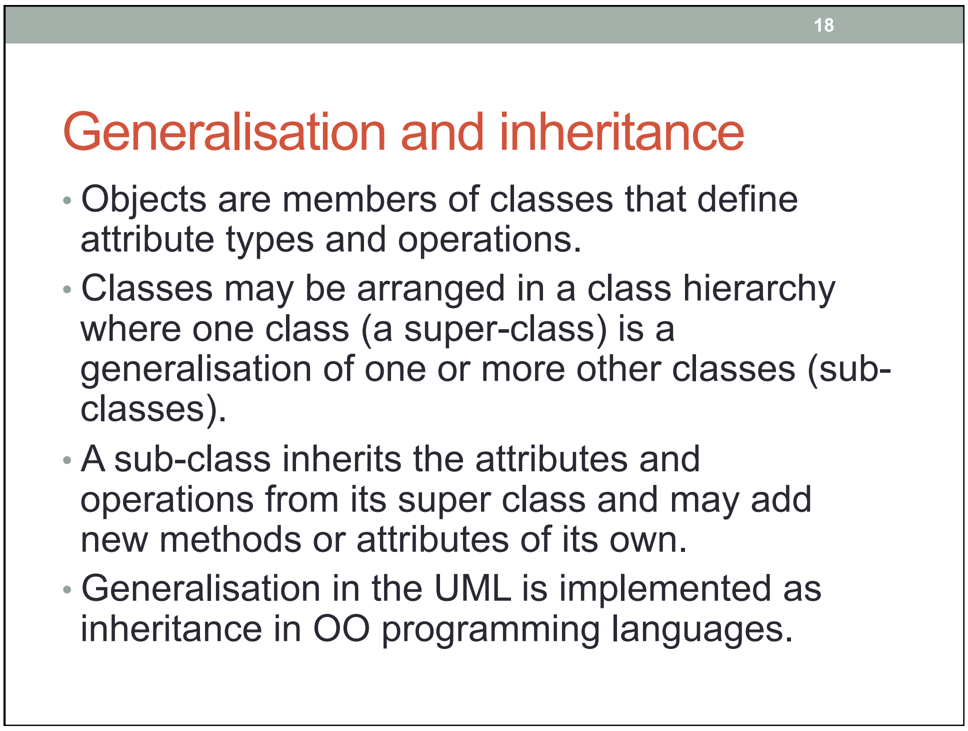
- E-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services.
- They are usually organised using a multi-tier architecture with application layers associated with each tier.





Characteristics of OOD

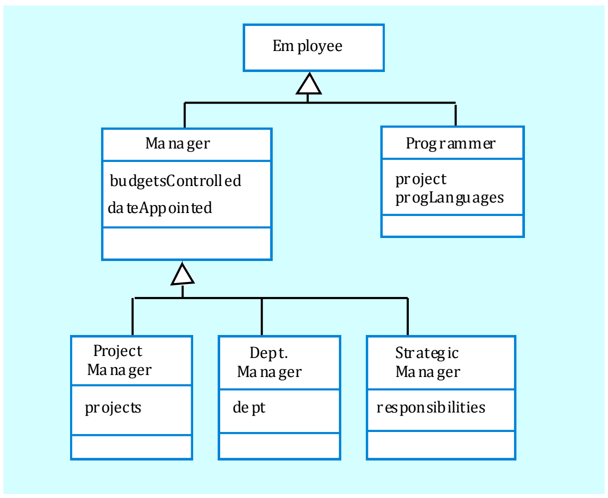
- Objects are abstractions of real-world or system entities and manage themselves.
- Objects are independent and encapsulate state and representation information.
- System functionality is expressed in terms of object services.
- Shared data areas are eliminated. Objects communicate by message passing.
- Objects may be distributed and may execute sequentially or in parallel.
- Easier maintenance. Objects may be understood as stand-alone entities.
- Objects are potentially reusable components.
- For some systems, there may be an obvious mapping from real world entities to system objects.
- Conceptually, objects communicate by message passing



Generalisation and inheritance

- Objects are members of classes that define attribute types and operations.
- Classes may be arranged in a class hierarchy where one class (a super-class) is a generalisation of one or more other classes (sub-classes).
- A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own.
- Generalisation in the UML is implemented as inheritance in OO programming languages.

A generalisation hierarchy



Advantages of inheritance

- It is an abstraction mechanism which may be used to classify entities.
- It is a reuse mechanism at both the design and the programming level.
- The inheritance graph is a source of organisational knowledge about domains and systems.
- PROBLEMS:
- Object classes are not self-contained. they cannot be understood without reference to their super-classes.
- Designers have a tendency to reuse the inheritance graph created during analysis. Can lead to significant inefficiency.
- The inheritance graphs of analysis, design and implementation have different functions and should be separately maintained.

Weather system description

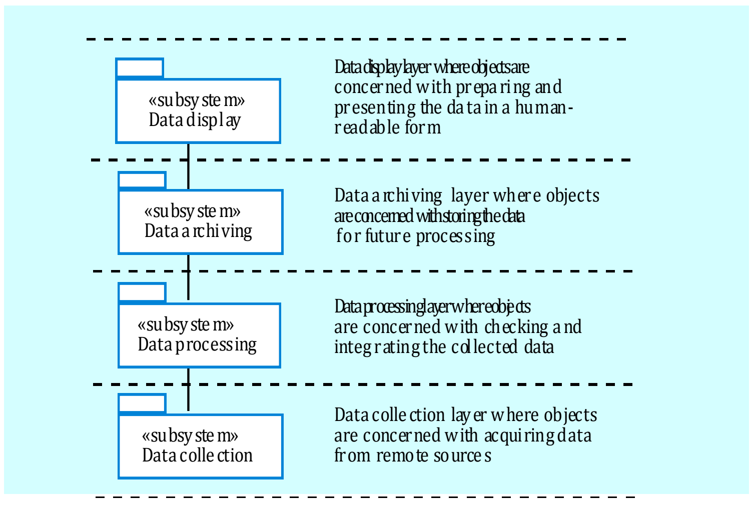
A weather mapping system is required to generate weather maps on a regular basis using data collected from remote, unattended weather stations and other data sources such as weather observers, balloons and satellites. Weather stations transmit their data to the area computer in response to a request from that machine.

The area computer system validates the collected data and integrates it with the data from different sources. The integrated data is archived and, using data from this archive and a digitised map database a set of local weather maps is created. Maps may be printed for distribution on a special-purpose map printer or may be displayed in a number of different formats.

System context and models of use

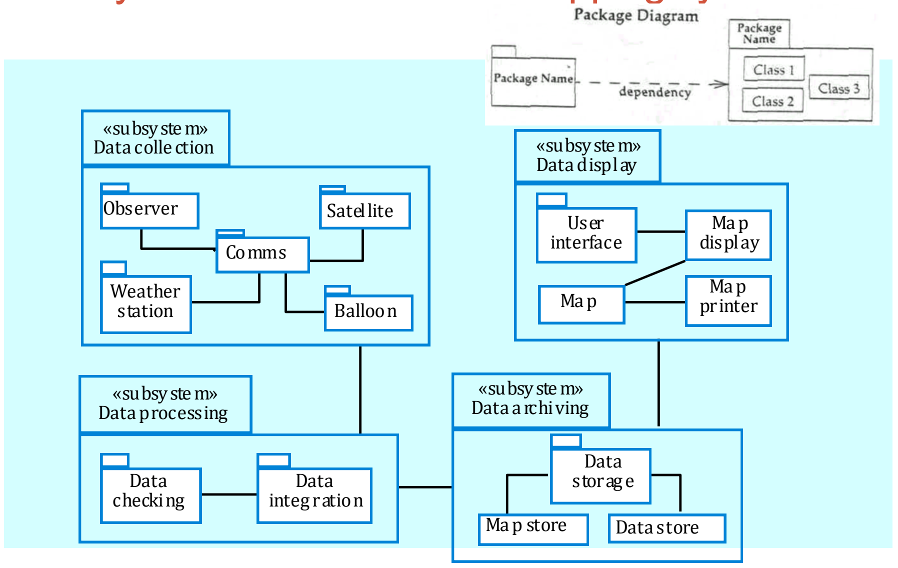
- Develop an understanding of the relationships between the software being designed and its external environment
- System context
  - A static model that describes other systems in the environment. Use a subsystem model to show other systems. Following slide shows the systems around the weather station system.
- Model of system use
  - A dynamic model that describes how the system interacts with its environment. Use use-cases to show interactions

Layered architecture



«subsystem» Data display
Data display layer where objects are concerned with preparing and presenting the data in a human-readable form
«subsystem» Data archiving
Data archiving layer where objects are concerned with storing the data for future processing
«subsystem» Data processing
Data processing layer where objects are concerned with checking and integrating the collected data
«subsystem» Data collection
Data collection layer where objects are concerned with acquiring data from remote sources

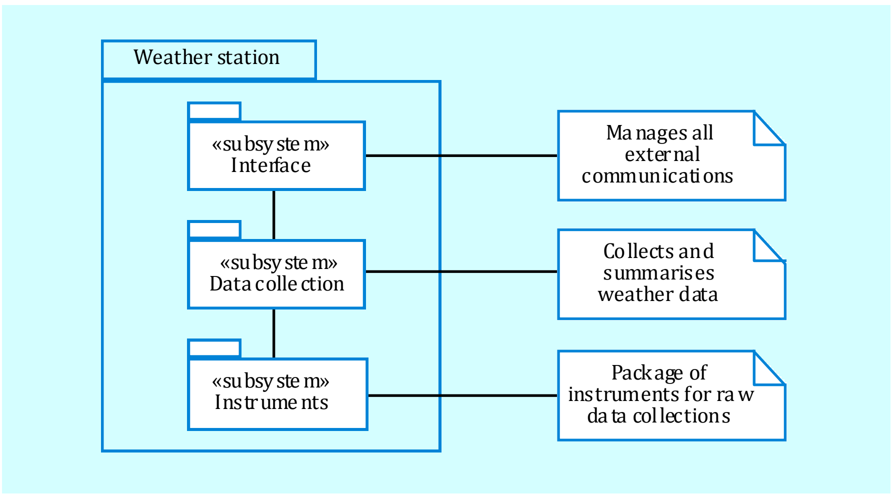
Subsystems in the weather mapping system



Architectural design

- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- A layered architecture is appropriate for the weather station
  - Interface layer for handling communications;
  - Data collection layer for managing instruments;
  - Instruments layer for collecting data.
- There should normally be no more than 7 entities in an architectural model.

Weather station architecture


Weather station
«subsystem» Interface
«subsystem» Data collection
«subsystem» Instruments
Manages all external communications
Collects and summarises weather data
Package of instruments for raw data collections

Approaches to identification

- Use a grammatical approach based on a natural language description of the system (used in Hood OOD method).
- Base the identification on tangible things in the application domain.
- Use a behavioural approach and identify objects based on what participates in what behaviour.
- Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.

Weather station description

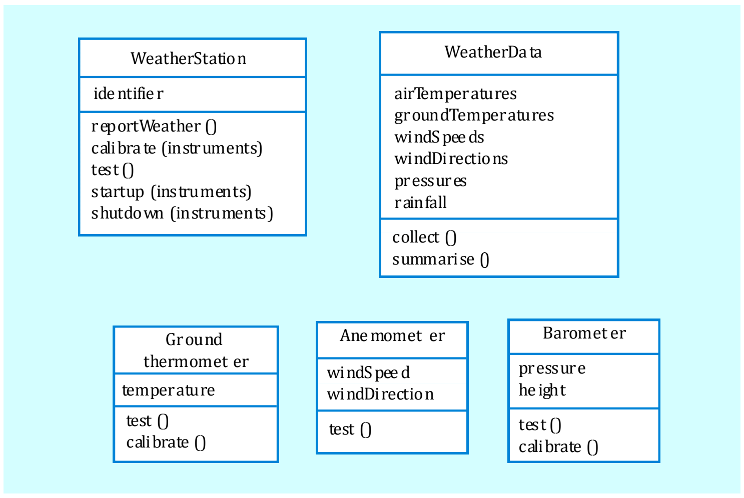
A weather station is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge. Data is collected periodically.

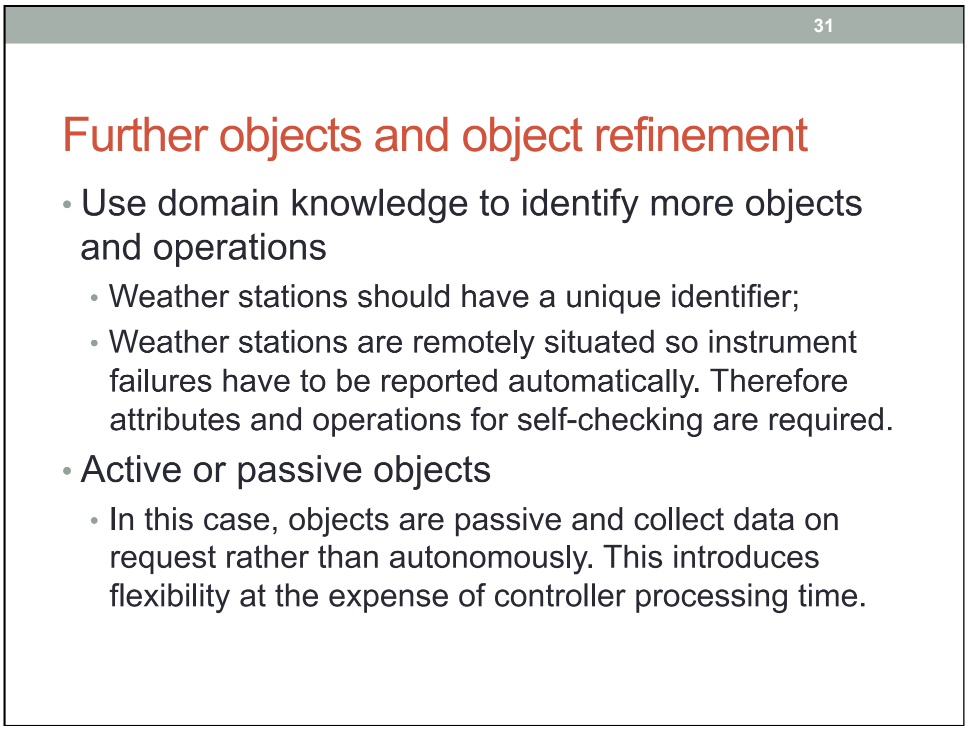
When a command is issued to transmit the weather data, the weather station processes and summarises the collected data. The summarised data is transmitted to the mapping computer when a request is received.

Weather station object classes

- Ground thermometer, Anemometer, Barometer
  - Application domain objects that are 'hardware' objects related to the instruments in the system.
- Weather station
  - The basic interface of the weather station to its environment. It therefore reflects the interactions identified in the use-case model.
- Weather data
  - Encapsulates the summarised data from the instruments.

Weather station object classes

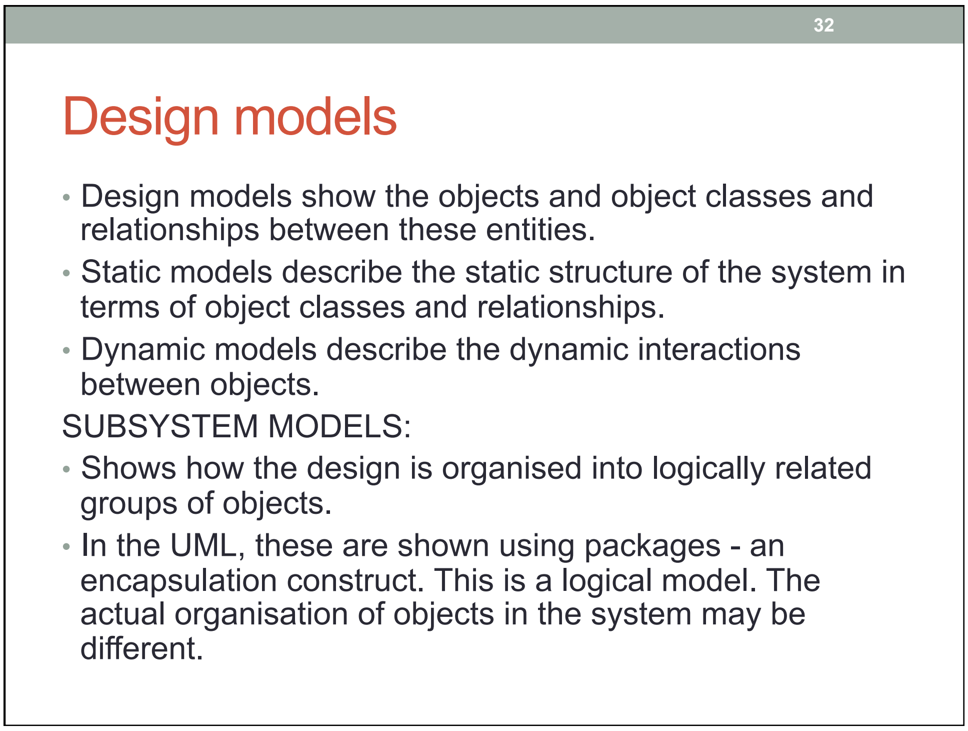




31

Further objects and object refinement

- Use domain knowledge to identify more objects and operations
  - Weather stations should have a unique identifier;
  - Weather stations are remotely situated so instrument failures have to be reported automatically. Therefore attributes and operations for self-checking are required.
- Active or passive objects
  - In this case, objects are passive and collect data on request rather than autonomously. This introduces flexibility at the expense of controller processing time.



32

Design models

- Design models show the objects and object classes and relationships between these entities.
- Static models describe the static structure of the system in terms of object classes and relationships.
- Dynamic models describe the dynamic interactions between objects.

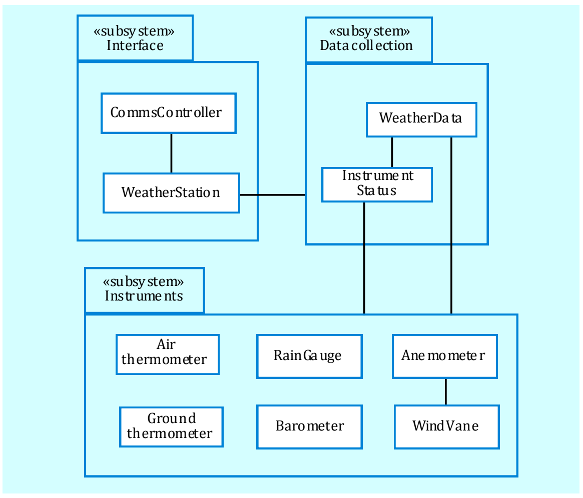
SUBSYSTEM MODELS:

- Shows how the design is organised into logically related groups of objects.
- In the UML, these are shown using packages - an encapsulation construct. This is a logical model. The actual organisation of objects in the system may be different.

Examples of design models

- Sub-system models that show logical groupings of objects into coherent subsystems.
- Sequence models that show the sequence of object interactions.
- State machine models that show how individual objects change their state in response to events.
- Other models include use-case models, aggregation models, generalisation models, etc.

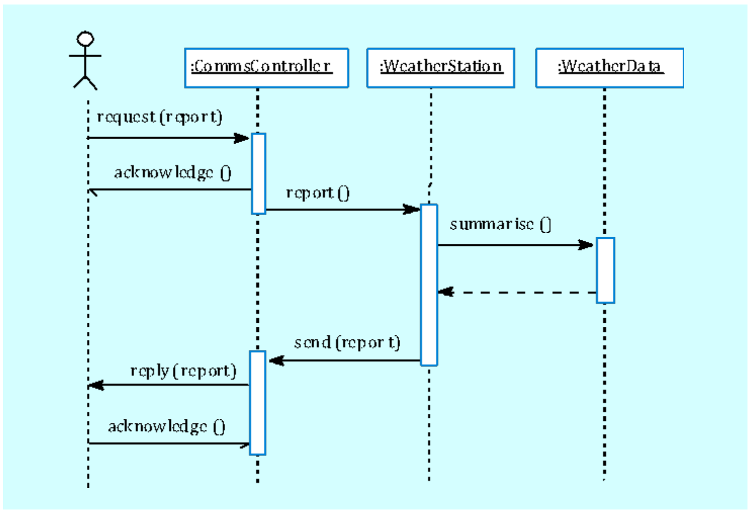
Weather station subsystems



Sequence models

- Sequence models show the sequence of object interactions that take place
  - Objects are arranged horizontally across the top;
  - Time is represented vertically so models are read top to bottom;
  - Interactions are represented by labelled arrows, Different styles of arrow represent different types of interaction;
  - A thin rectangle in an object lifeline represents the time when the object is the controlling object in the system.

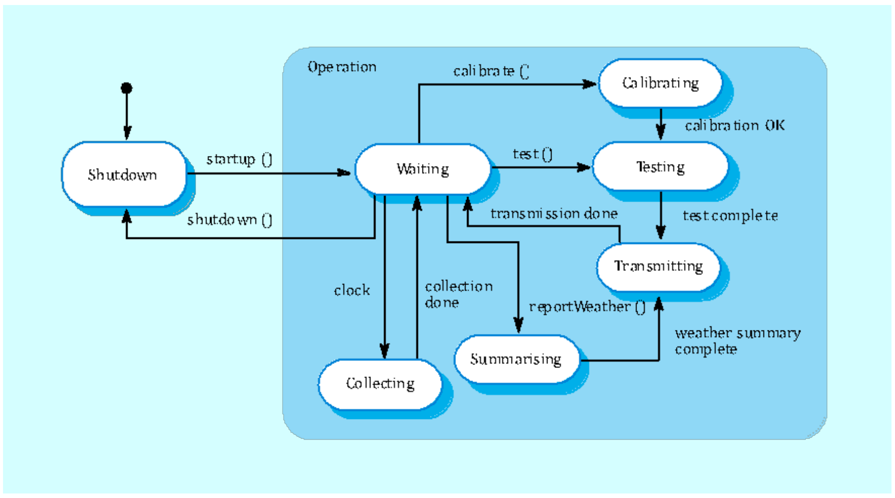
Data collection sequence



Statecharts

- Show how objects respond to different service requests and the state transitions triggered by these requests
  - If object state is Shutdown then it responds to a Startup() message;
  - In the waiting state the object is waiting for further messages;
  - If reportWeather () then system moves to summarising state;
  - If calibrate () the system moves to a calibrating state;
  - A collecting state is entered when a clock signal is received.

Weather station state diagram

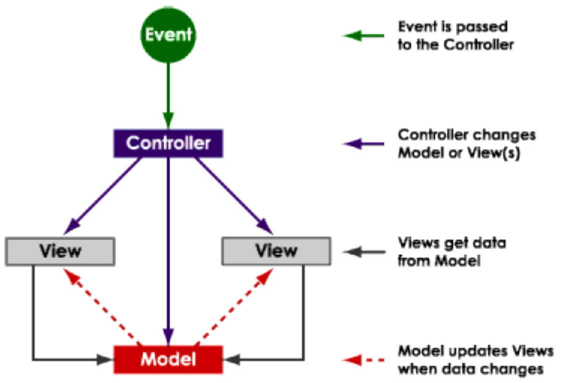


Model-View-Controller Pattern

- **Model-View-Controller (MVC)** is a classic design pattern often used by applications that need the ability to maintain multiple views of the same data. The MVC pattern hinges on a clean separation of objects into one of three categories — **models** for maintaining data, **views** for displaying all or a portion of the data, and **controllers** for handling events that affect the model or view(s).
- Because of this separation, multiple views and controllers can interface with the same model. Even new types of views and controllers that never existed before can interface with a model without forcing a change in the model design.

How It Works

- The MVC abstraction can be graphically represented as follows.
- Events typically cause a controller to change a model, or view, or both. Whenever a controller changes a model's data or properties, all dependent views are automatically updated. Similarly, whenever a controller changes a view, for example, by revealing areas that were previously hidden, the view gets data from the underlying model to refresh itself.


Event
Controller
View
View
Model
Event is passed to the Controller
Controller changes Model or View(s)
Views get data from Model
Model updates Views when data changes