Designing the system

creative process of transforming
the problem into a solution

Topics

- Architecture design
- Distributed System Architecture
- Application Architecture
- Object-oriented Design

Software architecture

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design.
- The output of this design process is a description of the software architecture.

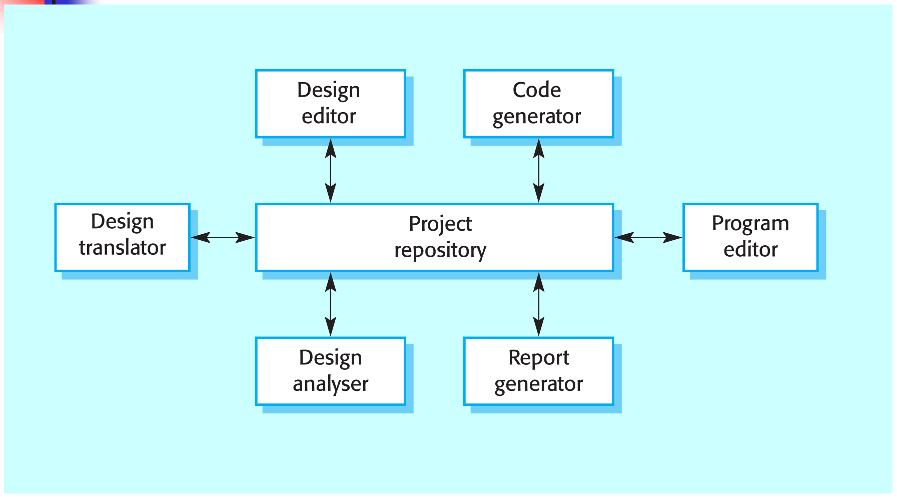
System organisation

- Reflects the basic strategy that is used to structure a system.
- Three organisational styles are widely used:
  - A shared data repository style;
  - A shared services and servers style;
  - An abstract machine or layered style.

The repository model

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

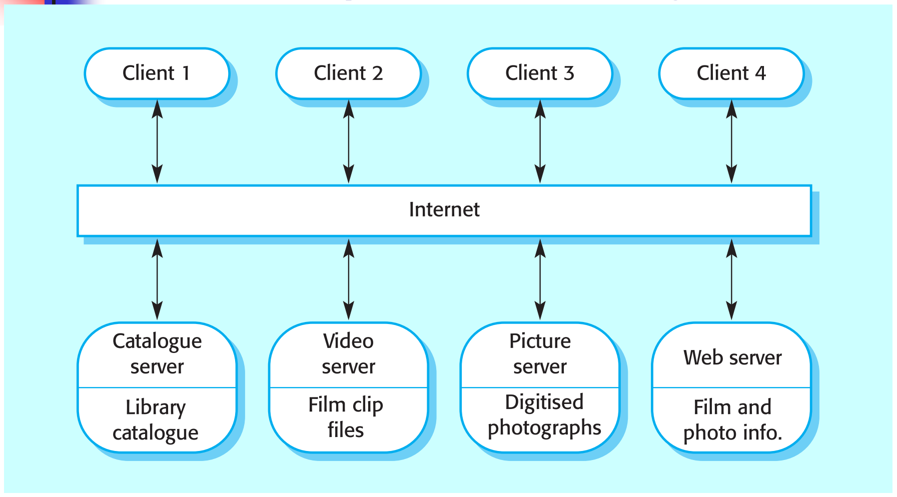
CASE toolset architecture



Client-server model

- Distributed system model which shows how data and processing is distributed across a range of components.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

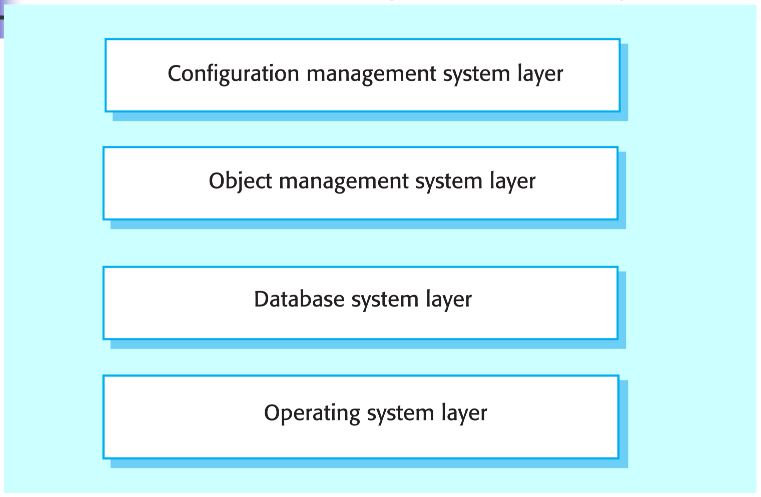
Film and picture library



Abstract machine (layered) model

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

Version management system


Configuration management system layer
Object management system layer
Database system layer
Operating system layer

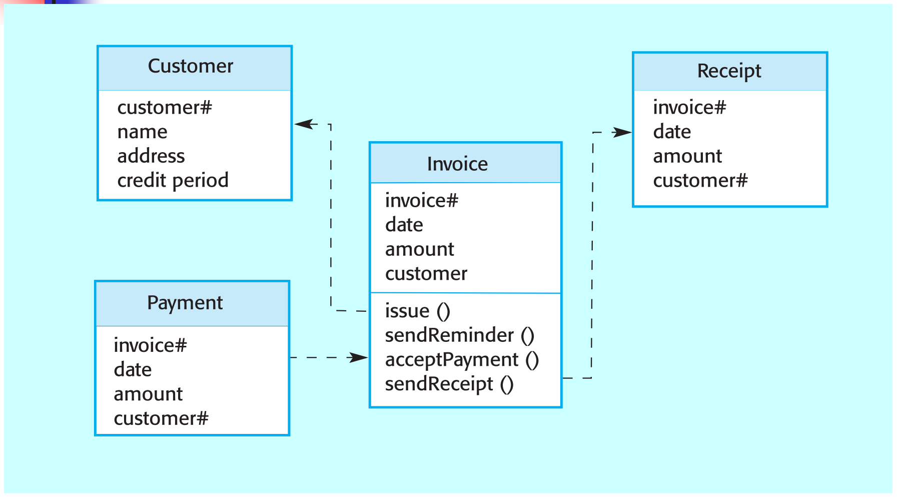
Modular decomposition

- Another structural level where sub-systems are decomposed into modules.
- Two modular decomposition models covered
  - An object model where the system is decomposed into interacting object;
  - A pipeline or data-flow model where the system is decomposed into functional modules which transform inputs to outputs.

Object models

- Structure the system into a set of loosely coupled objects with well-defined interfaces.
- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
- When implemented, objects are created from these classes and some control model used to coordinate object

Invoice processing system



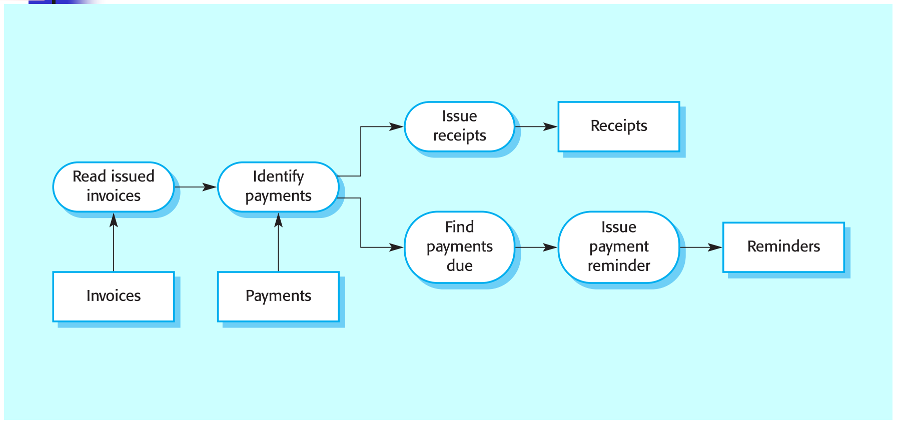
Object model advantages

- Objects are loosely coupled so their implementation can be modified without affecting other objects.
- The objects may reflect real-world entities.
- OO implementation languages are widely used.
- However, object interface changes may cause problems and complex entities may be hard to represent as objects.

Function-oriented pipelining

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model (as in UNIX shell).
- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
- Not really suitable for interactive

Invoice processing system



Pipeline model advantages

- Supports transformation reuse.
- Intuitive organisation for stakeholder communication.
- Easy to add new transformations.
- Relatively simple to implement as either a concurrent or sequential system.
- However, requires a common format for data transfer along the pipeline and difficult to support event-based interaction.

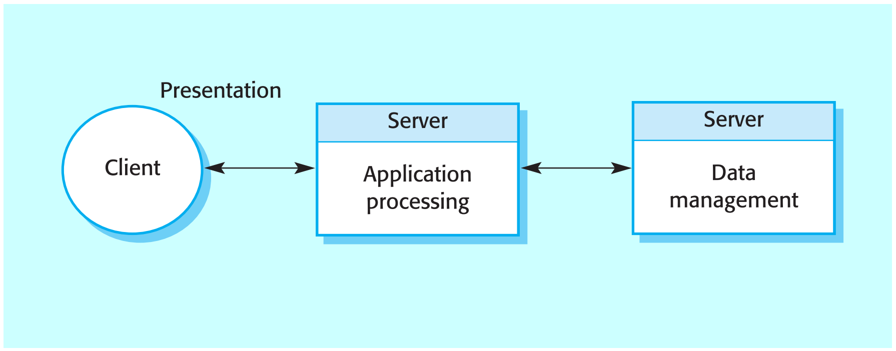
Topics

- Architecture design
- <u>Distributed System Architecture</u>
- Application Architecture
- Object-oriented Design

Thin and fat clients

- Thin-client model
  - In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.
- Fat-client model
  - In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with

A 3-tier C/S architecture

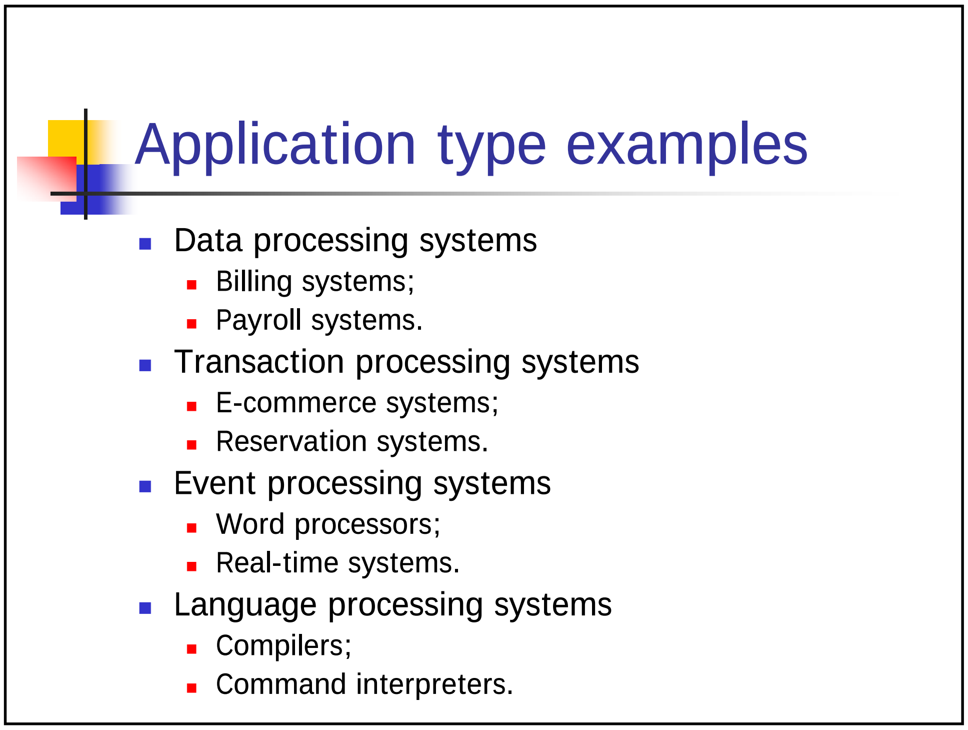


Topics

- Architecture design
- Distributed System Architecture
- Application Architecture
- Object-oriented Design

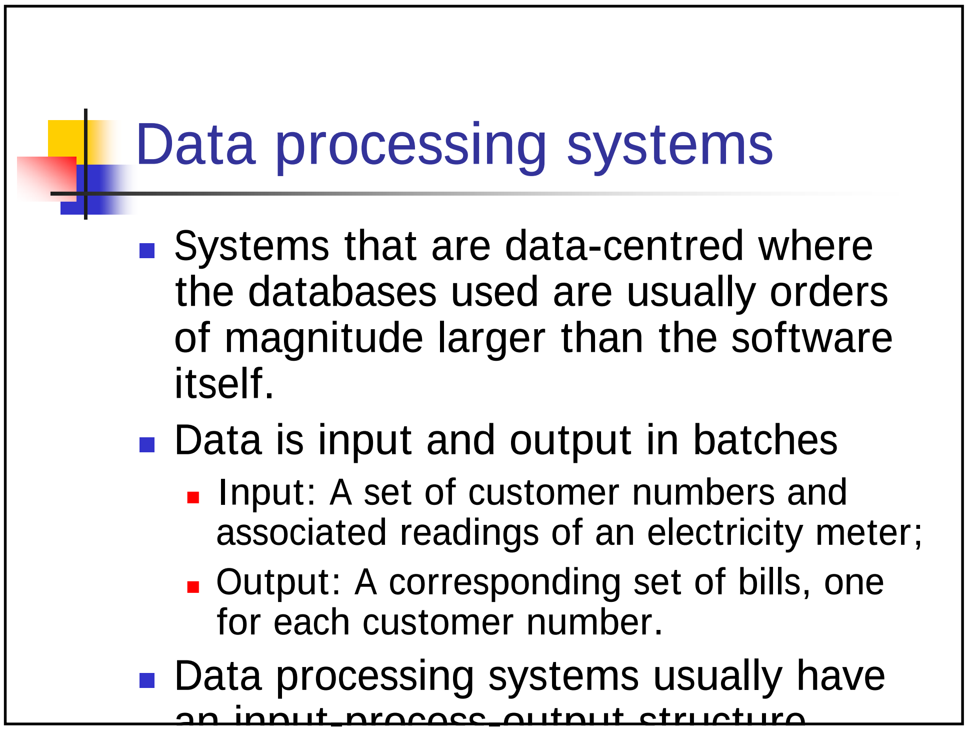
Generic application architectures

- Application systems are designed to meet an organisational need.
- As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.
- A generic architecture is configured and adapted to create a system that meets specific requirements.



Application type examples

- Data processing systems
  - Billing systems;
  - Payroll systems.
- Transaction processing systems
  - E-commerce systems;
  - Reservation systems.
- Event processing systems
  - Word processors;
  - Real-time systems.
- Language processing systems
  - Compilers;
  - Command interpreters.



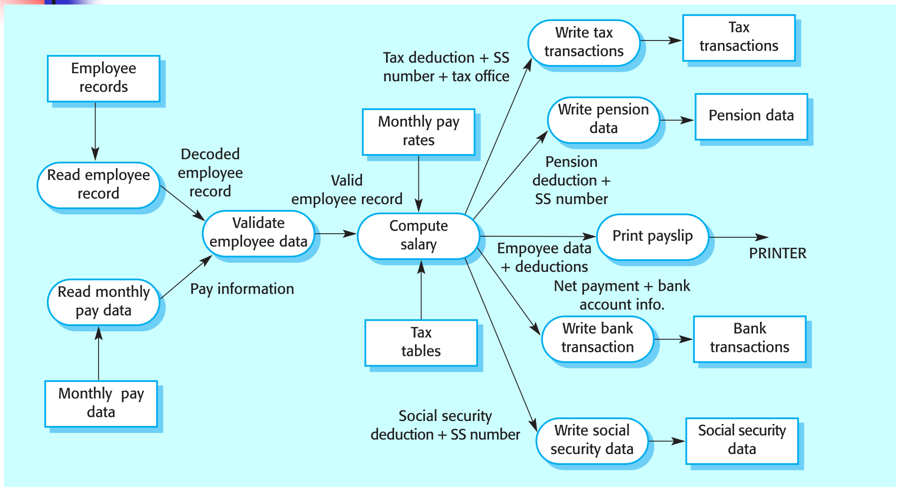
Data processing systems

- Systems that are data-centred where the databases used are usually orders of magnitude larger than the software itself.
- Data is input and output in batches
  - Input: A set of customer numbers and associated readings of an electricity meter;
  - Output: A corresponding set of bills, one for each customer number.
- Data processing systems usually have an input-process-output structure

Data-flow diagrams

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

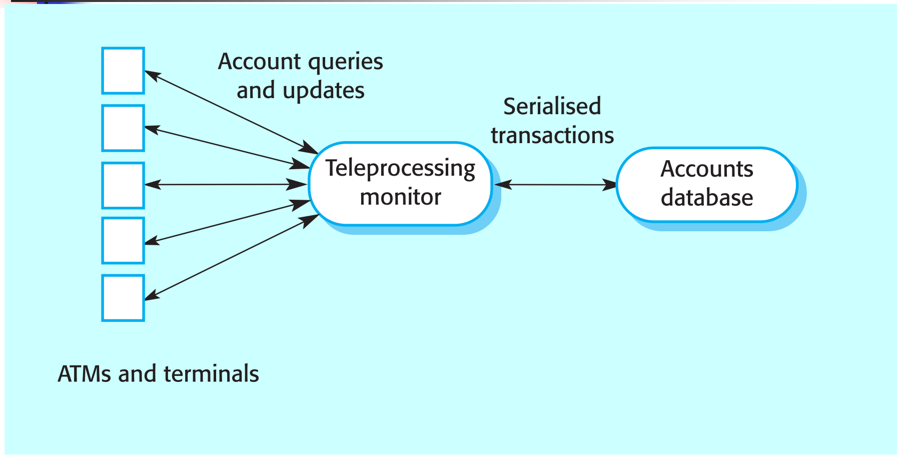
Salary payment DFD



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

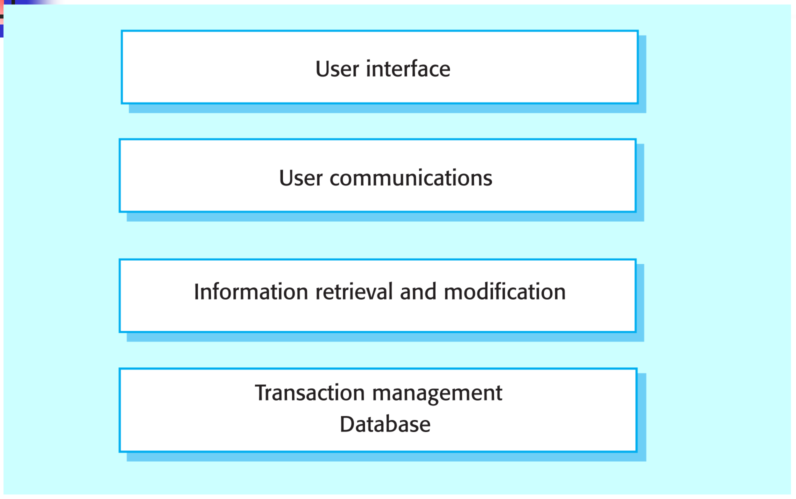
Transaction management



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

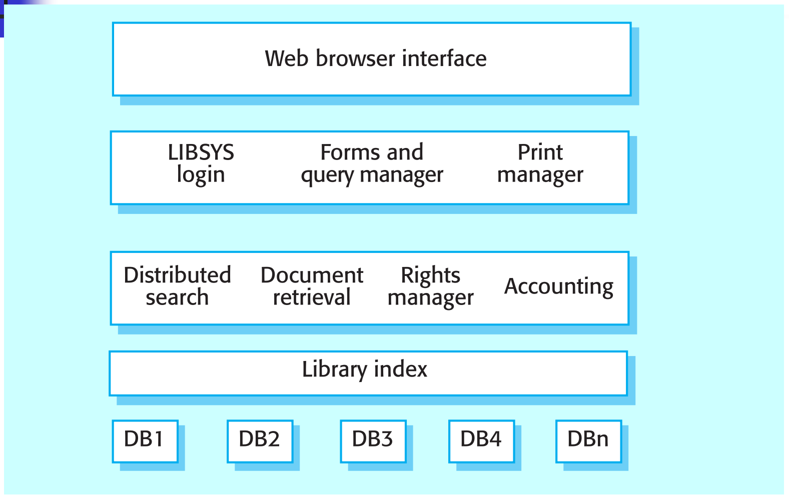
Information system structure



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

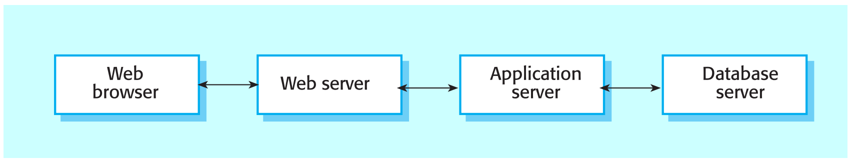


Layered system implementation

- 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

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.



Event processing systems

- These systems respond to events in the system’s environment.
- Their key characteristic is that event timing is unpredictable so the architecture has to be organised to handle this.
- Many common systems such as word processors, games, etc. are event processing systems.

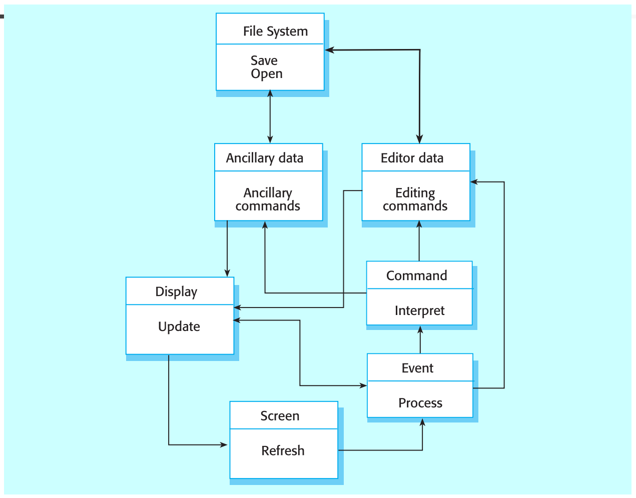
Editing systems

- Real-time systems and editing systems are the most common types of event processing system.
- Editing system characteristics:
  - Single user systems;
  - Must provide rapid feedback to user actions;
  - Organised around long transactions so may include recovery facilities.

Editing system components

- Editing systems are naturally object-oriented:
  - Screen - monitors screen memory and detects events;
  - Event - recognises events and passes them for processing;
  - Command - executes a user command;
  - Editor data - manages the editor data structure;
  - Ancillary data - manages other data such as styles and preferences;
  - File system - manages file I/O;
  - Display - updates the screen display.

Editing system architecture



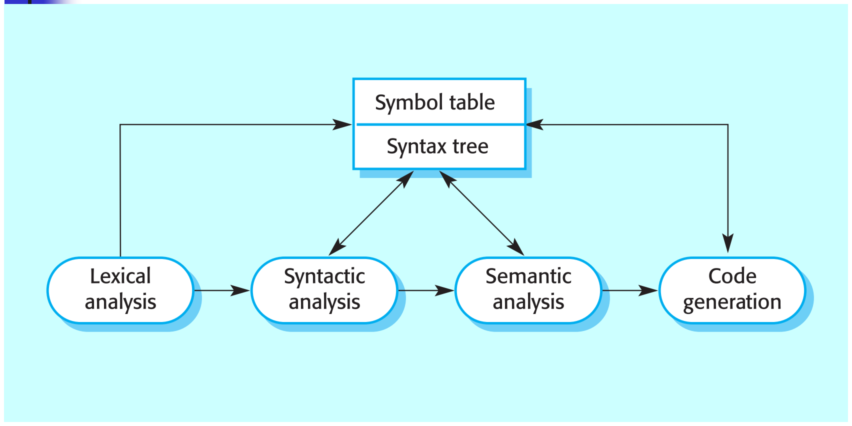
Language processing systems

- Accept a natural or artificial language as input and generate some other representation of that language.
- May include an interpreter to act on the instructions in the language that is being processed.
- Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data
  - Meta-case tools process tool descriptions, method rules, etc and generate tools.

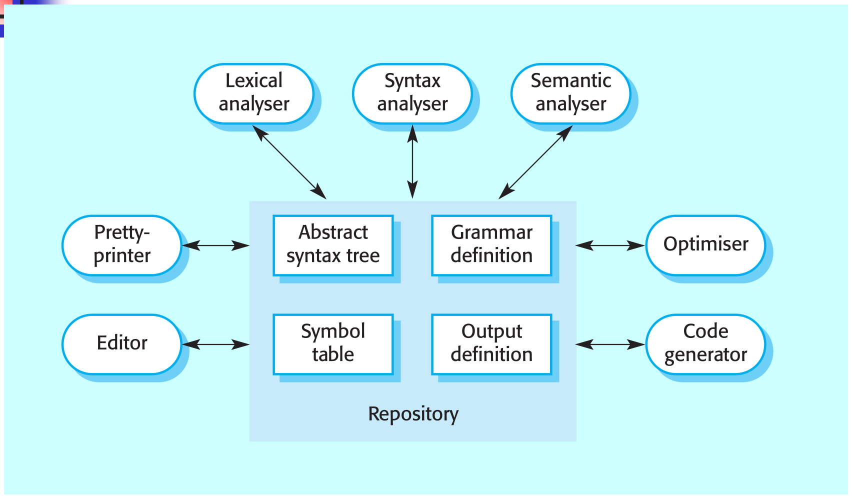
Language processing components

- Lexical analyser
- Symbol table
- Syntax analyser
- Syntax tree
- Semantic analyser
- Code generator

Data-flow model of a compiler



Repository model of a compiler



Topics

- Architecture design
- Distributed System Architecture
- Application Architecture
- <u>Object-oriented Design</u>

Object-oriented development

- Object-oriented analysis, design and programming are related but distinct.
- OOA is concerned with developing an object model of the application domain.
- OOD is concerned with developing an object-oriented system model to implement requirements.
- OOP is concerned with realising an OOD using an OO programming language

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.

Advantages of OOD

- 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.

Objects and object classes

*An **object** is an entity that has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.*

*Objects are created according to some **object class** definition. An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.*

Object identification

- Identifying objects (or object classes) is the most difficult part of object oriented design.
- There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.

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.

The Unified Modeling Language

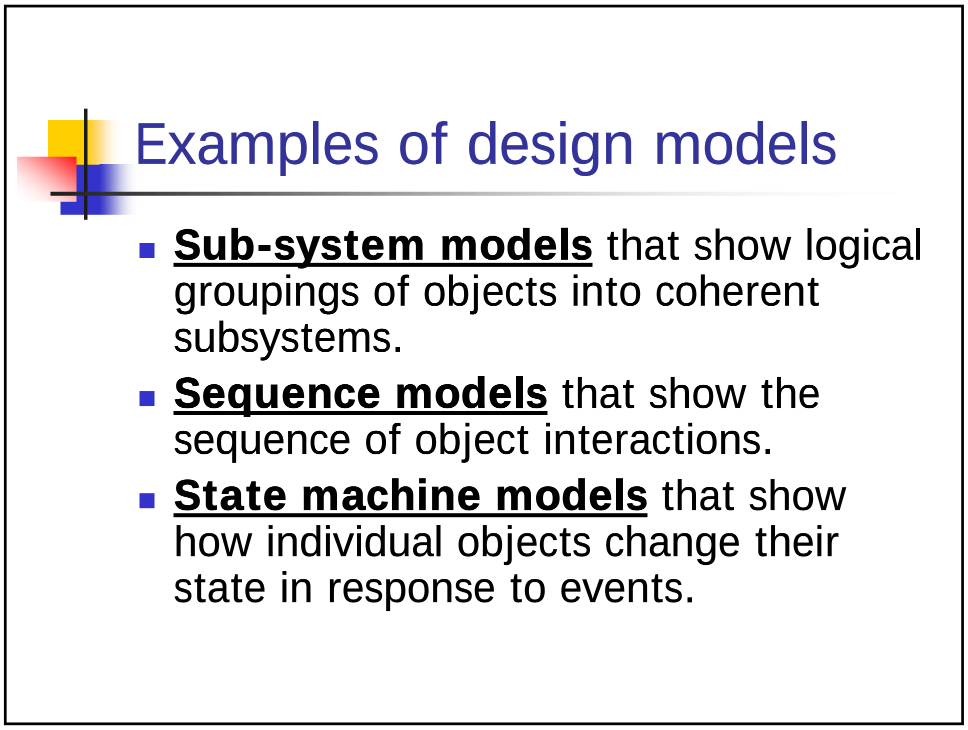
- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s.
- The Unified Modeling Language is an integration of these notations.
- It describes notations for a number of different models that may be produced during OO analysis and design.
- It is now a *de facto* standard for OO modelling.

What can you model with UML

- Structure Diagrams
  - Class diagram
  - Object diagram
  - Component diagram
  - Composite structure diagram
  - Package diagram
  - Deployment diagram
- Behavior Diagrams
  - Use case diagram
  - Activity diagram
  - State machine diagram
- Interaction diagrams
  - Sequence diagram
  - Communication Diagram
  - Timing Diagram
  - Interaction Overview Diagram

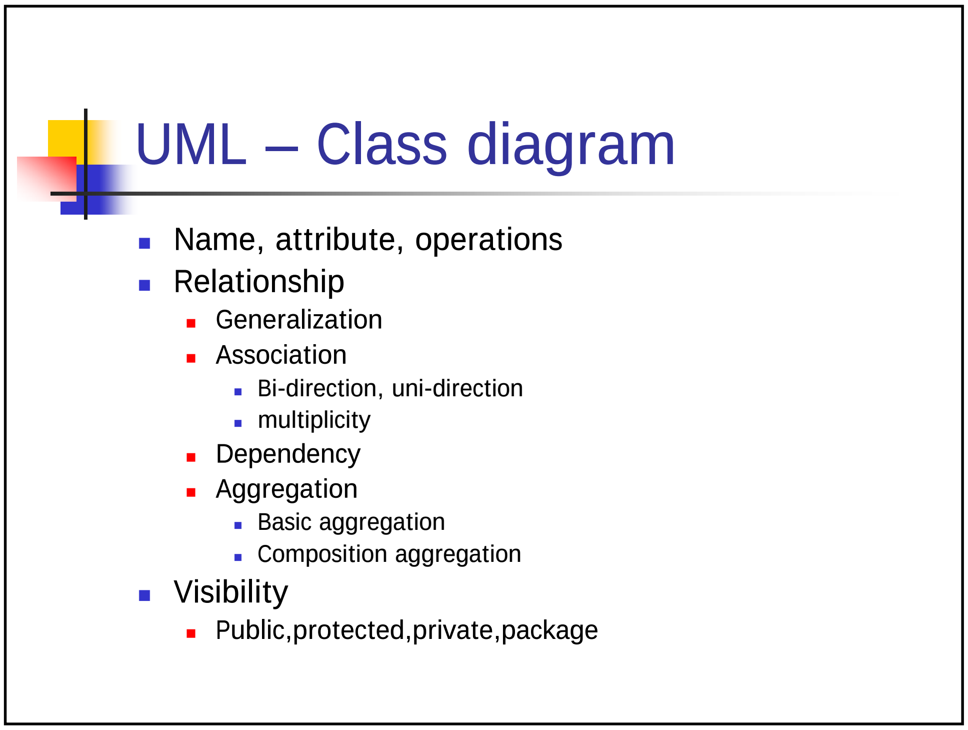
Design models

- 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.



Examples of design models

- **<u>Sub-system models</u>** that show logical groupings of objects into coherent subsystems.
- **<u>Sequence models</u>** that show the sequence of object interactions.
- **<u>State machine models</u>** that show how individual objects change their state in response to events.



UML – Class diagram

- Name, attribute, operations
- Relationship
  - Generalization
  - Association
    - Bi-direction, uni-direction
    - multiplicity
  - Dependency
  - Aggregation
    - Basic aggregation
    - Composition aggregation
- Visibility
  - Public,protected,private,package

UML – Sequence Model

- Objects
  - First placing the objects that participate in the interaction at the top of your diagram
    - Place the object that initiate the interaction at the left, and increasing more subordinate objects to the right
- Messages
  - Then Place the messages what these object send and receive along the Y axis, in order of increasing time from top to bottom
- Object lifeline
  - Vertical dashed line from top to the bottom
  - Objects may be created during the interaction
- Focus of control
  - Tall think rectangle shows the period of time during which an object is performing an action, either directly or through a subroutine procedure.

UML – Statechart Diagram

- Models the lifetime behavior of an individual object (a class, a use case, an entire system)
  - Focus on the event-ordered behavior of an object – modeling reactive systems

UML – Statechart Diagram

- States
  - A state is a condition or situation during which it satisfies some condition, performs some activity, or waits for some event. An object remains in a state for a finite amount of time
    - Name
      - Short nouns with the first latter capitalized
    - Entry/exit actions
    - Internal transitions
    - Substates
- Initial States
- Final states

UML – Statechart Diagram

- Transitions
  - Transition is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied.
- A transition have five parts
  - Source state
  - Event trigger
  - Guard condition
  - Action
  - Target state

UML – Statechart diagram

- Example: parsing a stream

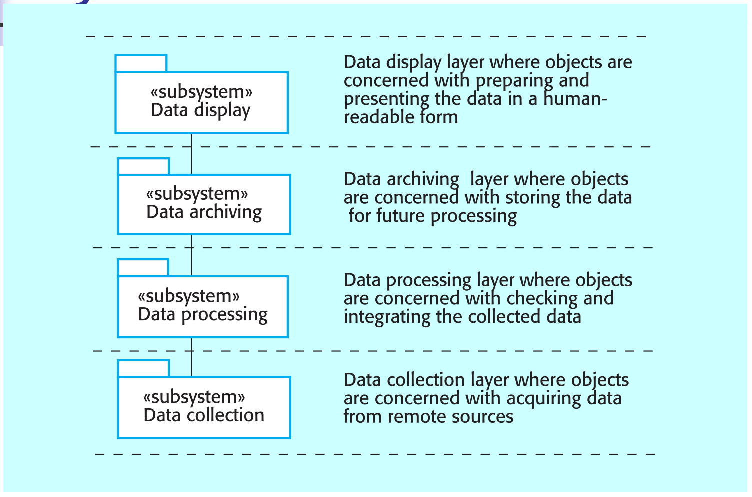
  ‘<‘ string ‘>’ body ‘;’

Weather system description

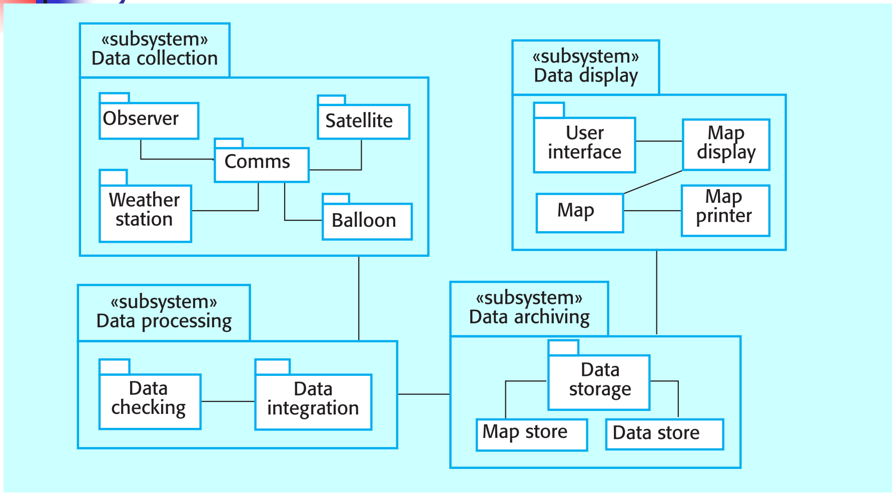
A weather mapping system is required to generate weather maps on a regular basis using data collected from remote, unattended weather stations and other data sources such as weather observers, balloons and satellites. Weather stations transmit their data to the area computer in response to a request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

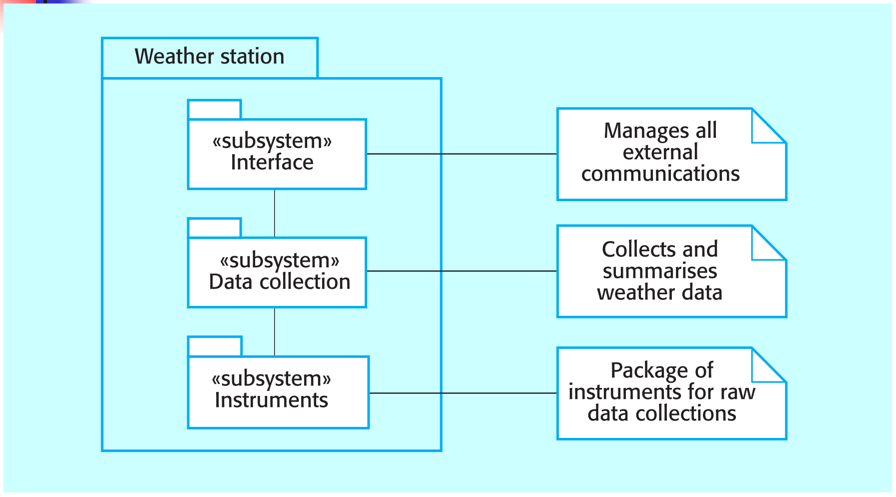
Layered architecture


«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



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

Weather station description

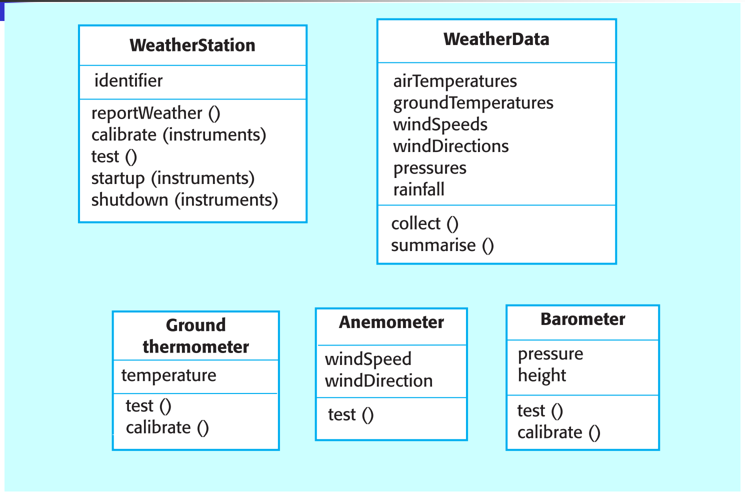
A weather station is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge. Data is collected periodically.

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



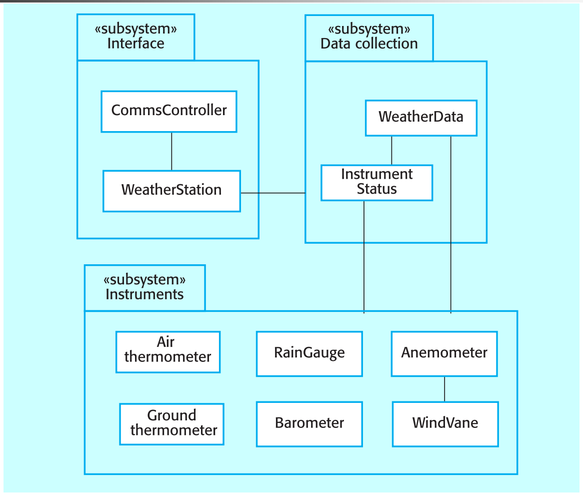
Further objects and object refinement

- 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.

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.

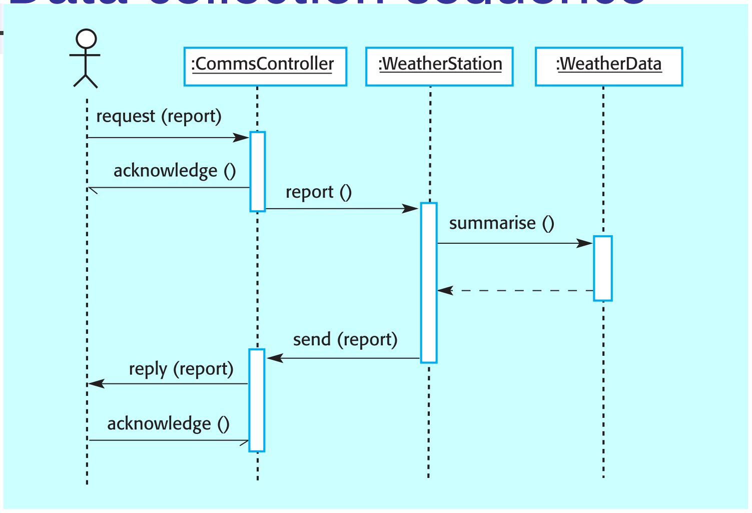
Weather station subsystems



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.

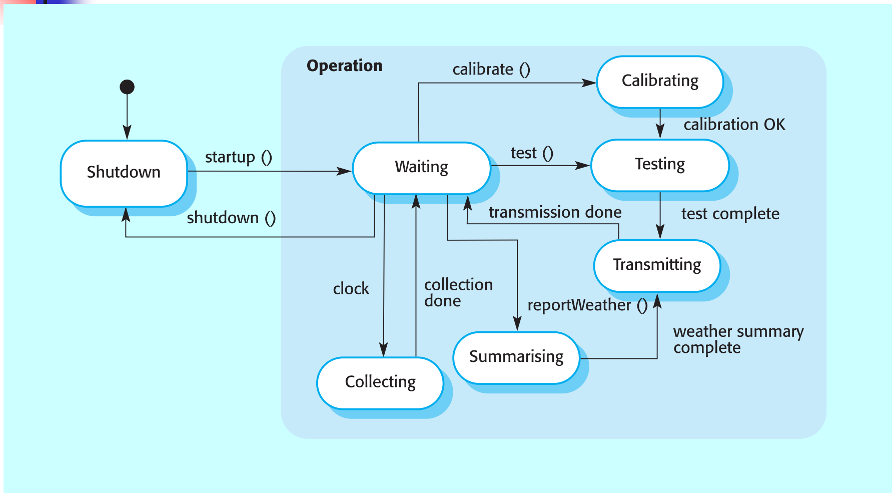
Data collection sequence



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.

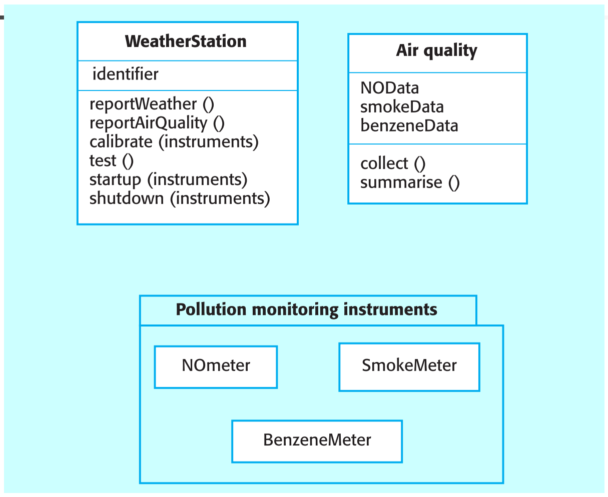
Weather station state diagram



Changes required

- Add an object class called Air quality as part of WeatherStation.
- Add an operation reportAirQuality to WeatherStation. Modify the control software to collect pollution readings.
- Add objects representing pollution monitoring instruments.

Pollution monitoring



Team Project Design

- Design artifacts
  - Architecture Design
  - Static models
    - Class diagram
    - Package diagram (optional)
  - Dynamic models (choose appropriate model for each use case)
    - Sequence diagram
    - Statechart diagram
    - Activity diagram