Architectural Design

- Establishing the overall structure of a software system

Objectives

- To introduce architectural design and to discuss its importance
- To explain why multiple models are required to document a software architecture
- To describe types of architectural models that may be used
- To discuss how domain-specific reference models may be used as a basis for product-lines and to compare software architectures

Topics covered

- System structuring
- Control models
- Modular decomposition
- Domain-specific architectures

What is software architecture?

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

Architectural design

- An early stage of the system design process
- Represents the link between specification and design processes
- Often carried out in parallel with some specification activities
- It involves identifying major system components and their communications

Advantages of explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible
- Large-scale reuse
  - The architecture may be reusable across a range of systems

Architectural design process

- System structuring
  - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified
- Control modelling
  - A model of the control relationships between the different parts of the system is established
- Modular decomposition
  - The identified sub-systems are decomposed into modules

Sub-systems and modules

- A *sub-system* is a system in its own right whose operation is independent of the services provided by other sub-systems
- A *module* is a system component that provides services to other components but would not normally be considered as a separate system

Architectural models

- Different architectural models may be produced during the design process
- Each model presents different perspectives on the architecture

Architectural models

- Static structural model
  - shows the major system components
- Dynamic process model
  - shows the process structure of the system
- Interface model
  - defines sub-system interfaces
- Relationships model
  - E.g., data-flow model

Architectural styles

- The architectural model of a system may conform to a generic architectural model or style
- An awareness of these styles can simplify the problem of defining system architectures
- However, most large systems are heterogeneous and do not follow a single architectural style

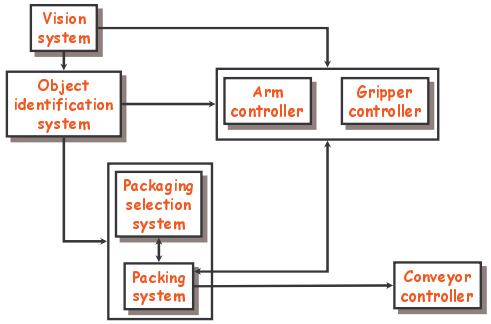
Architecture attributes

- Performance
  - Localize operations to minimize sub-system communication
- Security
  - Use a layered architecture with critical assets in inner layers
- Safety
  - Isolate safety-critical components
- Availability
  - Include redundant components in the architecture
- Maintainability
  - Use fine-grain, self-contained components

System structuring

- Concerned with decomposing the system into interacting sub-systems
- The architectural design is normally expressed as a block diagram presenting an overview of the system structure
- More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed

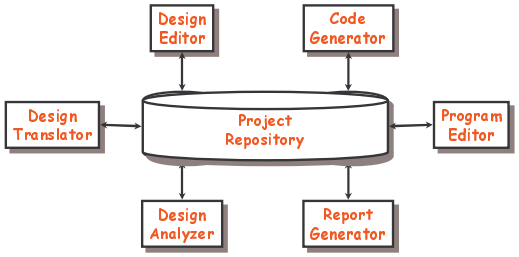
Packing robot control system



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



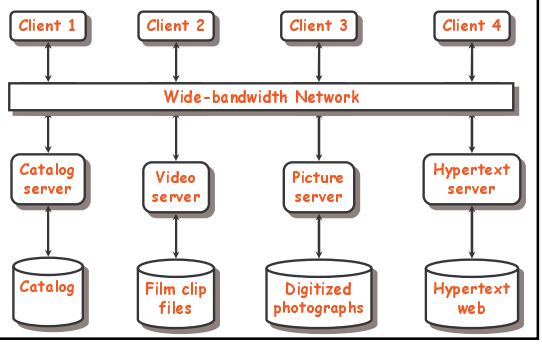
Repository model characteristics

- Advantages
  - Efficient way to share large amounts of data
  - Sub-systems need not be concerned with how data is managed
  - Centralized management e.g. backup, security, etc.
- Disadvantages
  - Sub-systems must agree on a repository data model. Inevitably a compromise
  - Data evolution is difficult and expensive
  - Difficult to distribute efficiently

Client-server architecture

- 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



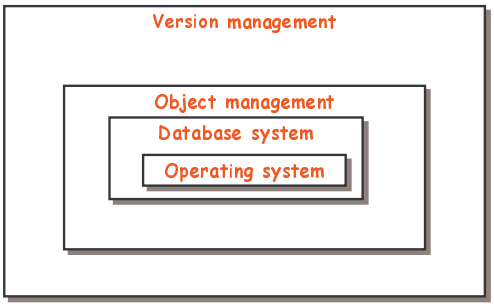
Client-server characteristics

- Advantages
  - Distribution of data is straightforward
  - Makes effective use of networked systems. May require cheaper hardware
  - Easy to add new servers or upgrade existing servers
- Disadvantages
  - No shared data model so sub-systems use different data organization
  - Data interchange may be inefficient
  - Redundant management in each server
  - No central register of names and services - it may be difficult to find out what servers and services are available

Abstract machine model

- Used to model the interfacing of sub-systems
- Organizes 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 difficult to structure systems in this way

Version management system



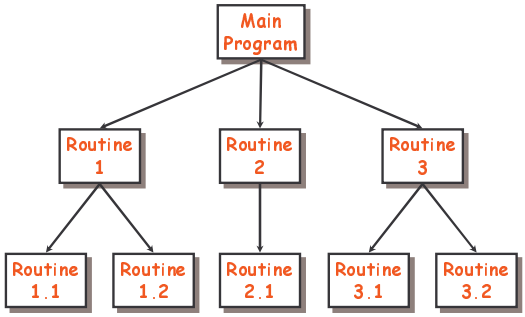
Control models

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model
- Centralized control
  - One sub-system has overall responsibility for control and starts and stops other sub-systems
- Event-based control
  - Each sub-system can respond to externally generated events from other sub-systems or the system's environment

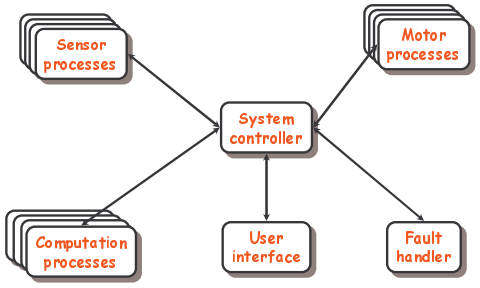
Centralized control

- A control sub-system takes responsibility for managing the execution of other sub-systems
- Call-return model
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems
- Manager model
  - One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement. Applicable to concurrent systems.

Call-return model



Real-time system control



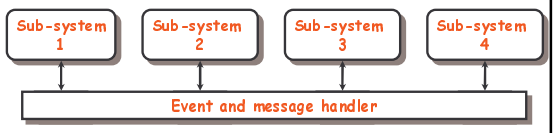
Event-driven systems

- Driven by externally generated events
- Two principal event-driven models
  - Broadcast models. An event is broadcast to all sub-systems. Any sub-system that can handle the event may do so
  - Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing
- Other event driven models include spreadsheets and production systems

Broadcast model

- Effective in integrating sub-systems on different computers in a network
- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system that can handle the event
- Control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them
- However, sub-systems don't know if or when an event will be handled

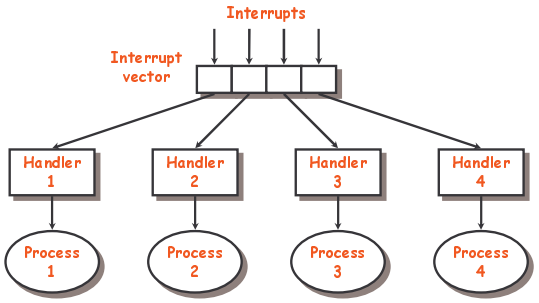
Selective broadcasting



Interrupt-driven systems

- Used in real-time systems where fast response to an event is essential
- There are known interrupt types with a handler defined for each type
- Each type is associated with a memory location and a hardware switch causes transfer to its handler
- Allows fast response but complex to program and difficult to validate

Interrupt-driven control



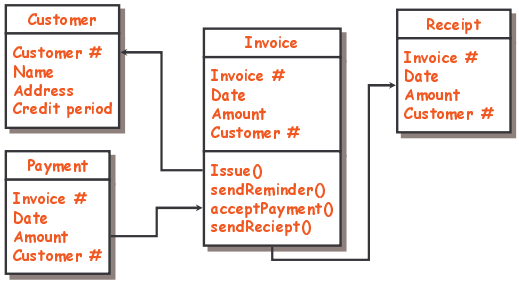
Modular decomposition

- Another structural level where sub-systems are decomposed into modules
- Two modular decomposition models
  - An object model where the system is decomposed into interacting objects
  - A data-flow model where the system is decomposed into functional modules that transform inputs to outputs. Also known as the pipeline model
- If possible, decisions about concurrency should be delayed until modules are implemented

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 operations

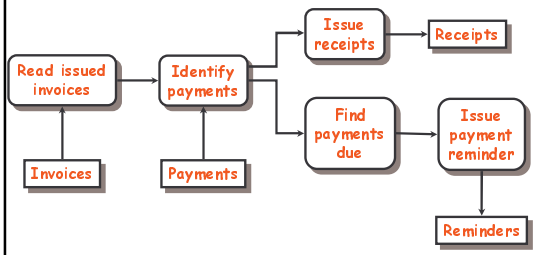
Invoice processing system



Data-flow models

- 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 that is extensively used in data processing systems
- Not really suitable for interactive systems

Invoice processing system



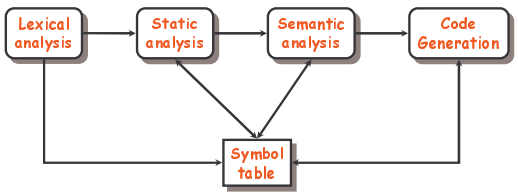
Domain-specific architectures

- Architectural models that are specific to some application domain
- Two types of domain-specific models
  - Generic models that are abstractions of a number of real systems and that encapsulate the principal characteristics of these systems
  - Reference models that are more abstract, idealized models. Provide a means of information about that class of system and of comparing different architectures
- Generic models are usually bottom-up models; Reference models are top-down models

Generic models

- Compiler model is a well-known example although other models exist in more specialized application domains
  - Lexical analyser
  - Symbol table
  - Syntax analyser
  - Syntax tree
  - Semantic analyser
  - Code generator
- Generic compiler model may be organized according to different architectural models

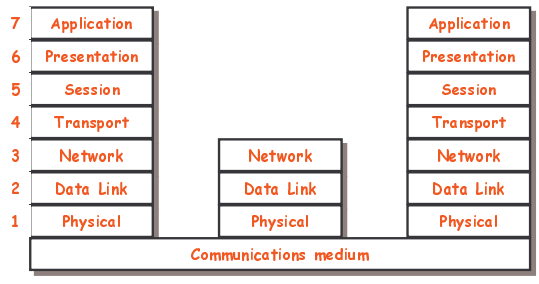
Compiler model



Reference architectures

- Reference models are derived from a study of the application domain rather than from existing systems
- May be used as a basis for system implementation or to compare different systems. It acts as a standard against which systems can be evaluated
- OSI model is a layered model for communication systems

OSI reference model



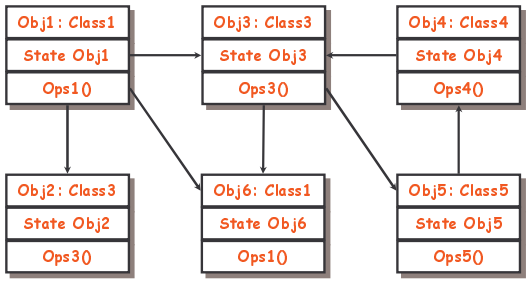
Object-oriented Design

Designing systems using self-contained objects and object classes

Characteristics of OOD

- Objects are abstractions of real-world entities
- 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

Interacting objects



Advantages of OOD

- Easier maintenance. Objects may be understood as stand-alone entities
- Objects are appropriate reusable components
- For some systems, there may be an obvious mapping from real world entities to system objects

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 realizing an OOD using an OO programming language such as Java or C++

Objects and object classes

- Objects are entities in a software system that represent instances of real-world and system entities
- Object classes are templates for objects. They may be used to create objects
- Object classes may inherit attributes and services from other object classes

Objects

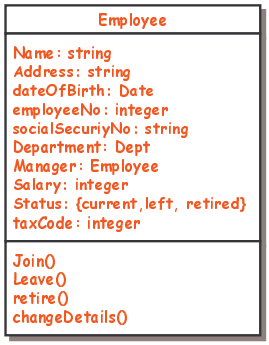
An **object** is an entity that has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is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

The Unified Modelling Language

- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s
- The Unified Modelling 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

Employee object class (UML)



Object communication

- Conceptually, objects communicate using messages
- Messages
  - The name of the service requested by the calling object
  - Copies of the information required to execute the service and the name of a holder for the result of the service
- In practice, messages are often implemented by procedure calls
  - Name = procedure name.
  - Information = parameter list.

Message examples

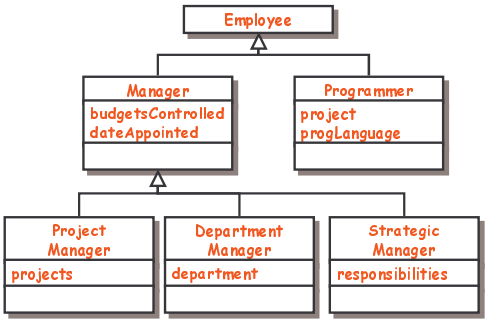
```
//Call a method associated with a buffer
//object that returns the next value
//in the buffer
        v = circularBuffer.Get() ;

//Call the method associated with a
//thermostat object that sets the
//temperature to be maintained
        thermostat.setTemp (20) ;
```

Generalization 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 generalization 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
- Generalization in the UML is implemented as inheritance in OO programming languages

A generalization hierarchy



Advantages of inheritance

- It is an abstraction mechanism that 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 organizational knowledge about domains and systems

Problems with inheritance

- 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

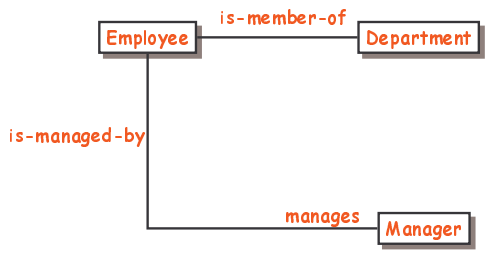
Inheritance and OOD

- There are differing views as to whether inheritance is fundamental to OOD.
  - View 1. Identifying the inheritance hierarchy or network is a fundamental part of object-oriented design. Obviously this can only be implemented using an OOPL.
  - View 2. Inheritance is a useful implementation concept which allows reuse of attribute and operation definitions. Identifying an inheritance hierarchy at the design stage places unnecessary restrictions on the implementation
- Inheritance introduces complexity that is undesirable, especially in critical systems

UML associations

- Objects and object classes participate in relationships with other objects and object classes
- In the UML, a generalized relationship is indicated by an association
- Associations may be annotated with information that describes the association
- Associations are general but may indicate that an attribute of an object is an associated object or that a method relies on an associated object

An association model



Concurrent objects

- The nature of objects as self-contained entities make them suitable for concurrent implementation
- The message-passing model of object communication can be implemented directly if objects are executing on separate processors in a distributed system

Servers and active objects

- Servers
  - The object is implemented as a parallel process (server) with entry points corresponding to object operations. If no calls are made to it, the object suspends itself and waits for further requests for service
- Active objects
  - Objects are implemented as parallel processes and the internal object state may be changed by
    - the object itself, and
    - external calls

Active objects

- Active objects may have their attributes modified by operations but may also update them autonomously using internal operations
- Example
  - Transponder object broadcasts an aircraft's position. The position may be updated using a satellite positioning system. The object periodically update the position by triangulation from satellites

An active transponder object

```
class Transponder extends Thread {
    Position currentPosition ;
    Coords c1, c2 ;
    Satellite sat1, sat2 ;
    Navigator theNavigator ;

    public Position givePosition ()
    {
        return currentPosition ;
    }

    public void run ()
    {
        while (true)
        {
            c1 = sat1.position () ;
            c2 = sat2.position () ;
            currentPosition = theNavigator.compute (c1, c2) ;
        }
    }
} //Transponder
```

Java threads

- Threads in Java are a simple construct for implementing concurrent objects
- Threads must include a method called run() and this is started up by the Java run-time system
- Active objects typically include an infinite loop so that they are always carrying out the computation

An object-oriented design process

- Define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop design models
- Specify object interfaces

EXAMPLE
Weather system description

A weather data collection 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.

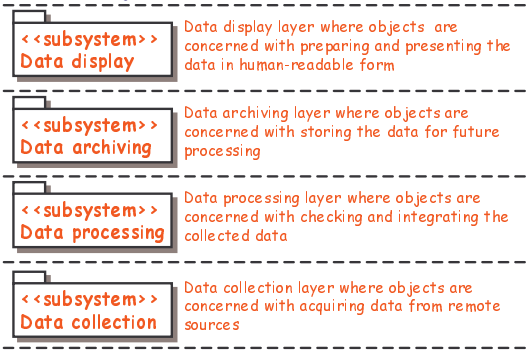
The area computer validates the collected data and integrates it with the data from different sources. The integrated data is archived and, using data from this archive and a digitized map database a set of local weather maps is created. Maps may be printed for distribution on a special-purpose map printer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

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 every five minutes.

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

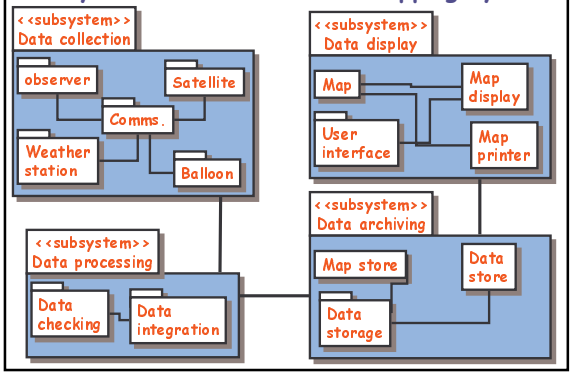
Layered architecture



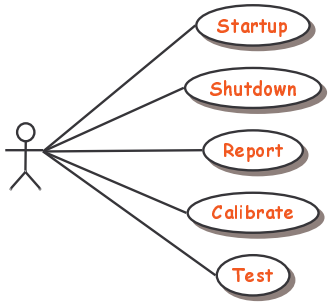
System context and models of use

- 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

Subsystems in the weather mapping system



Use-cases for the weather station



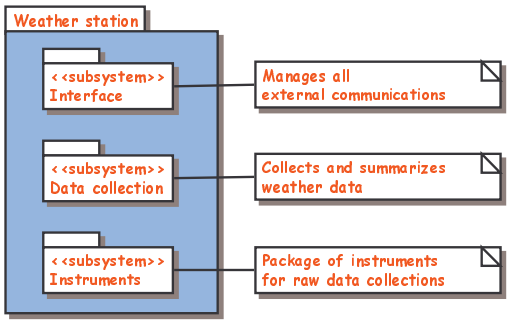
Use-case description

- **System**
  - **Weather station**
- **Use-case**
  - **Report**
- **Actors**
  - **Weather data collection system, Weather station**
- **Data**
  - **The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather data collection system. The data sent are the maximum minimum and average ground and air temperatures, the maximum, minimum and average air pressures, the maximum, minimum and average wind speeds, the total rainfall and the wind direction as sampled at 5 minute intervals.**
- **Stimulus**
  - **The weather data collection system establishes a modem link with the weather station and requests transmission of the data.**
- **Response**
  - **The summarized data is sent to the weather data collection system**
- **Comments**
  - **Weather stations are usually asked to report once per hour but this frequency may differ from one station to the other and may be modified in future.**

Architectural design

- Once interactions between the system and its environment have been understood, we use this information for designing the system architecture
- 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 be no more than 7 entities in an architectural model

Weather station architecture



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. Unlikely to get it right first time

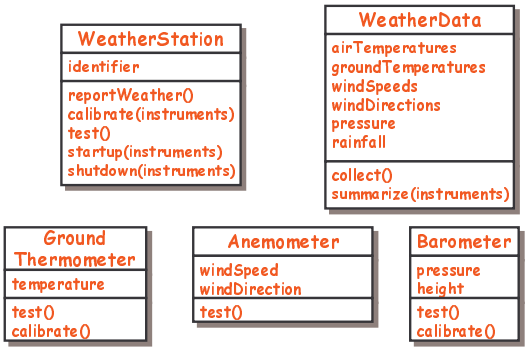
Approaches to identification

- 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 object classes

- Ground thermometer, 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 reflects the interactions identified in the use-case model
- Weather data
  - Encapsulates the summarized 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

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

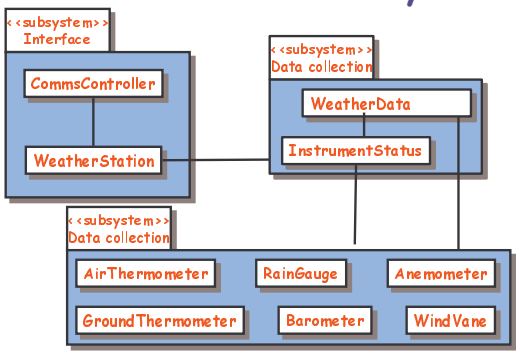
Examples of design models

- 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, generalization models,etc.

Subsystem models

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

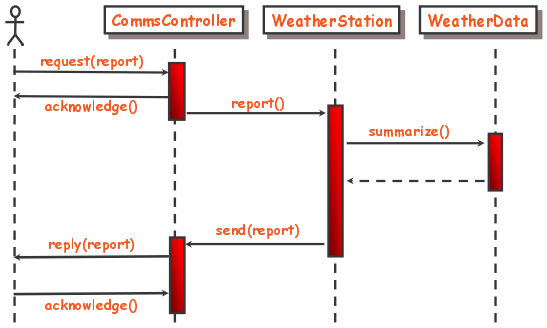
Weather station subsystems



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

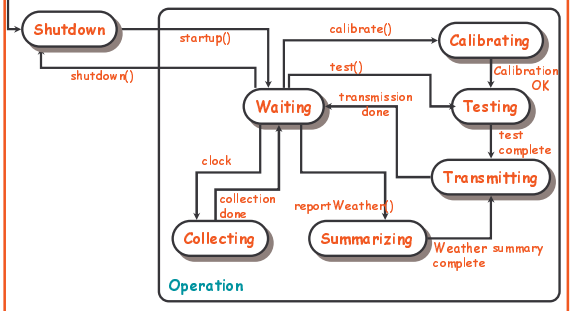
Data collection sequence



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 summarizing 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



Object interface specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel
- Designers should avoid designing the interface representation but should hide this in the object itself
- Objects may have several interfaces which are viewpoints on the methods provided
- The UML uses class diagrams for interface specification but Java may also be used

Weather station interface

```
interface WeatherStation {

    public void WeatherStation () ;

    public void startup () ;
    public void startup (Instrument i) ;

    public void shutdown () ;
    public void shutdown (Instrument i) ;

    public void reportWeather ( ) ;

    public void test () ;
    public void test ( Instrument i ) ;

    public void calibrate ( Instrument i) ;

    public int getID () ;

    } //WeatherStation
```

Design evolution

- Hiding information inside objects means that changes made to an object do not affect other objects in an unpredictable way
- Assume pollution monitoring facilities are to be added to weather stations. These sample the air and compute the amount of different pollutants in the atmosphere
- Pollution readings are transmitted with weather data

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