Unit 7. Design and Implementation

Architectural Design {Review}

- ✧ Need of Architecture
- ✧ Architectural Patterns
  - Box and Line Diagram
  - MVC
  - Layered Architecture
  - Repository Architecture
  - Client Server Architecture
  - P2P architecture
  - Multiprocessor Architecture
- ✧ Application Architecture

Unit 7: Design and Implementation (5 Hrs.)

Introduction; Object-Oriented Design using UML; Design Patterns; Implementation Issues; Open-Source Development

Design and implementation

- ✧ Software design is the stage at which the conceptual/logical model is converted to physical model
- ✧ Software design is the process by which an agent creates a specification of a software artifact intended to accomplish goals, using a set of primitive components and subject to constraints.
- ✧ Software design may refer to either "all the activity involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems"
- ✧ the activity following requirements specification and before programming

Design and implementation

- A product software implementation method is a systematically structured approach to transform the design model into a working product

- Software design and implementation activities are invariably inter-leaved.
  - Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
  - Implementation is the process of realizing the design as a program.

Build or buy

- In a wide range of domains, it is now possible to buy off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.
  - For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals.
  - It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.
- Re-use oriented software engineering
  - Application system reuse
  - Component reuse
  - Object and function reuse

Software reuse landscape

Approach	Description
Architectural patterns	Standard software architectures that support common types of application systems are used as the basis of applications.
Design patterns	Generic abstractions that occur across applications are represented as design patterns showing abstract and concrete objects and interactions.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards
Application frameworks	Collections of abstract and concrete classes are adapted and extended to create application systems.
Legacy system wrapping	Legacy systems are ‘wrapped’ by defining a set of interfaces and providing access to these legacy systems through these interfaces.

An object-oriented design process

Process stages

- There are a variety

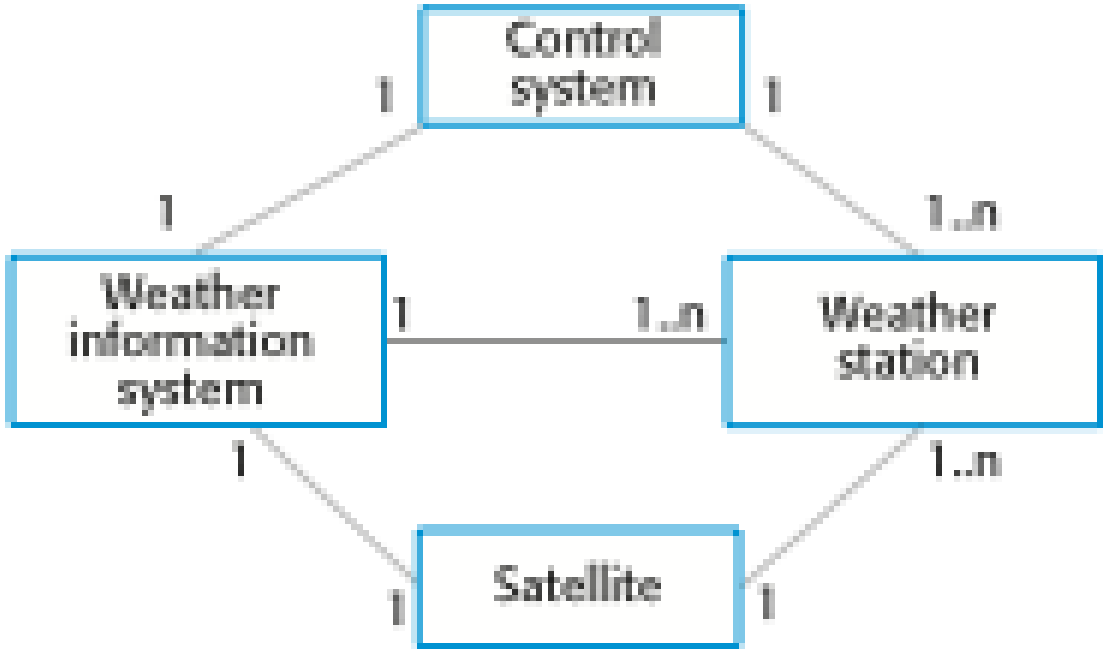
System context and interactions

- Understanding the relationships between the software that is being designed and its external environment is essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment.
- Understanding of the context also lets you establish the boundaries of the system.
- Setting the system boundaries helps you decide what features are implemented in the system being designed and what features are in other associated systems.

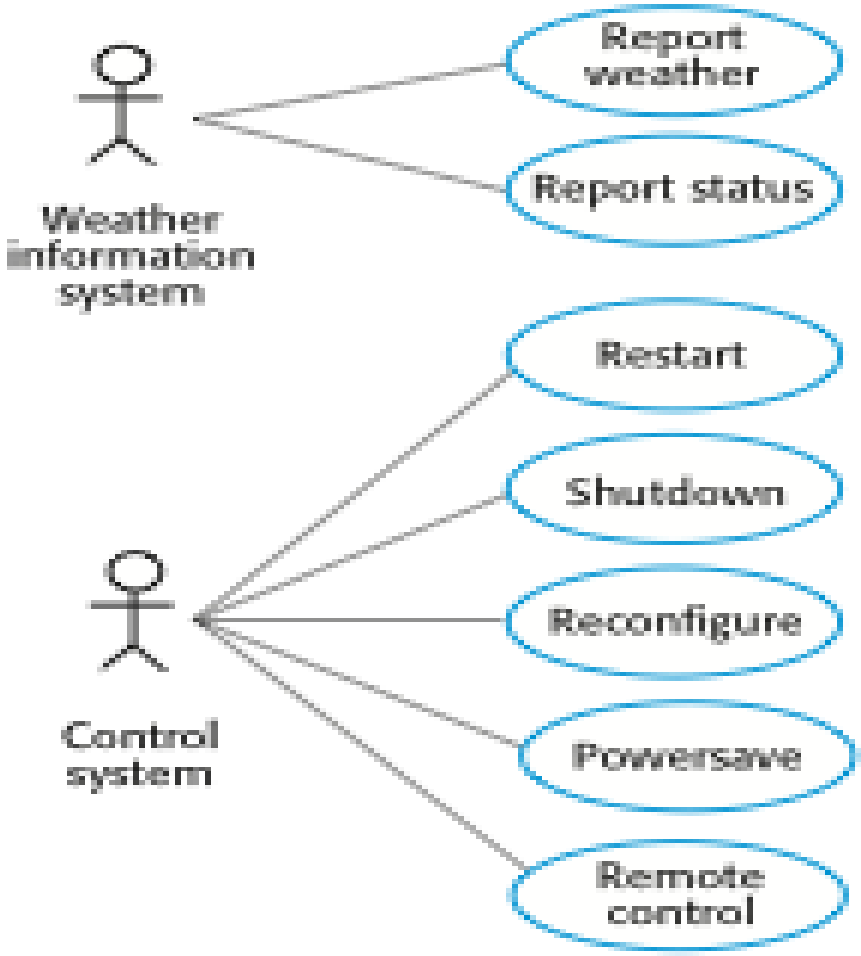
Context and interaction models

- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

System context for the weather station



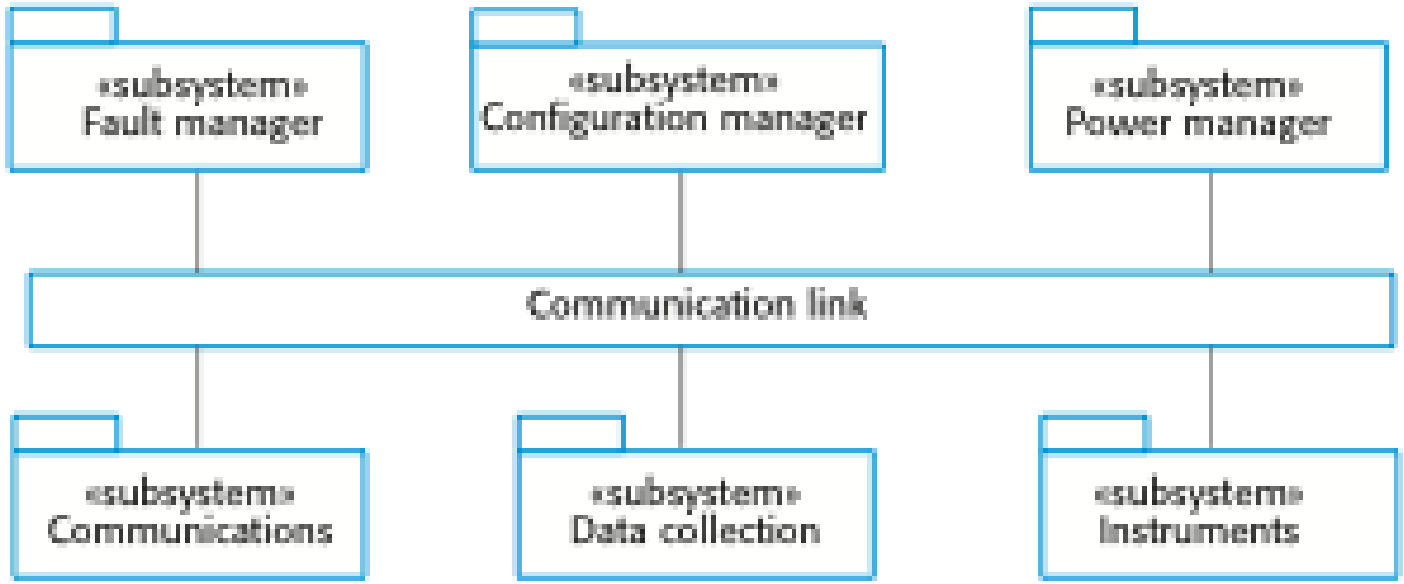
Weather station use cases



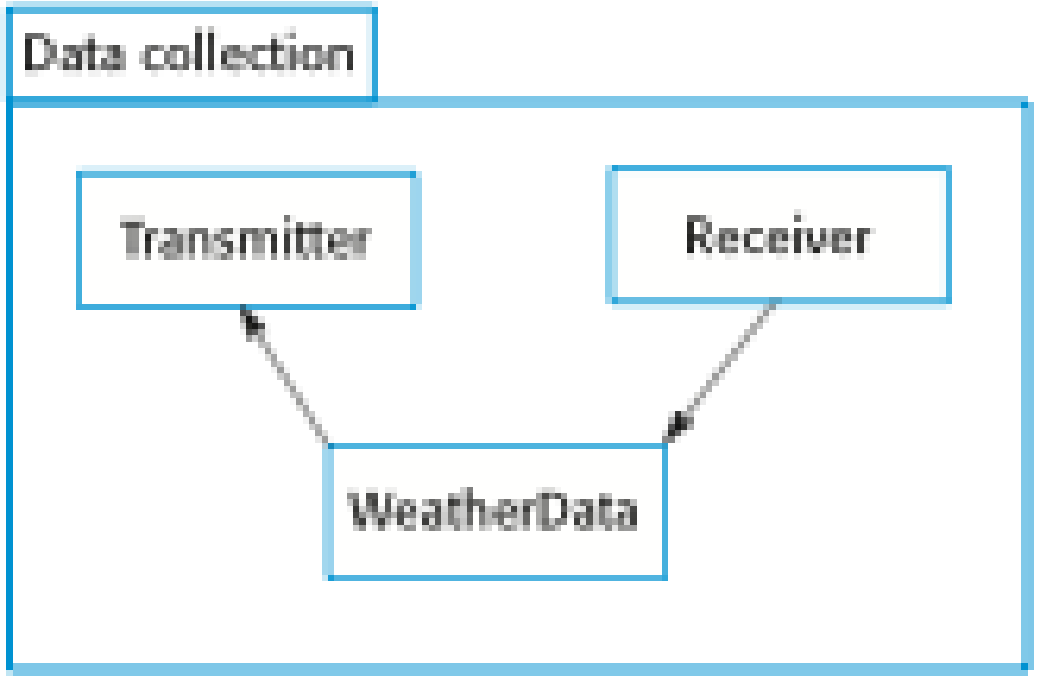
Architectural design

- Once interactions b

High-level architecture of the weather station



Architecture of data collection system



Identifying object

Object class identification

Approaches to identification

- Use a grammatical a

Weather station description

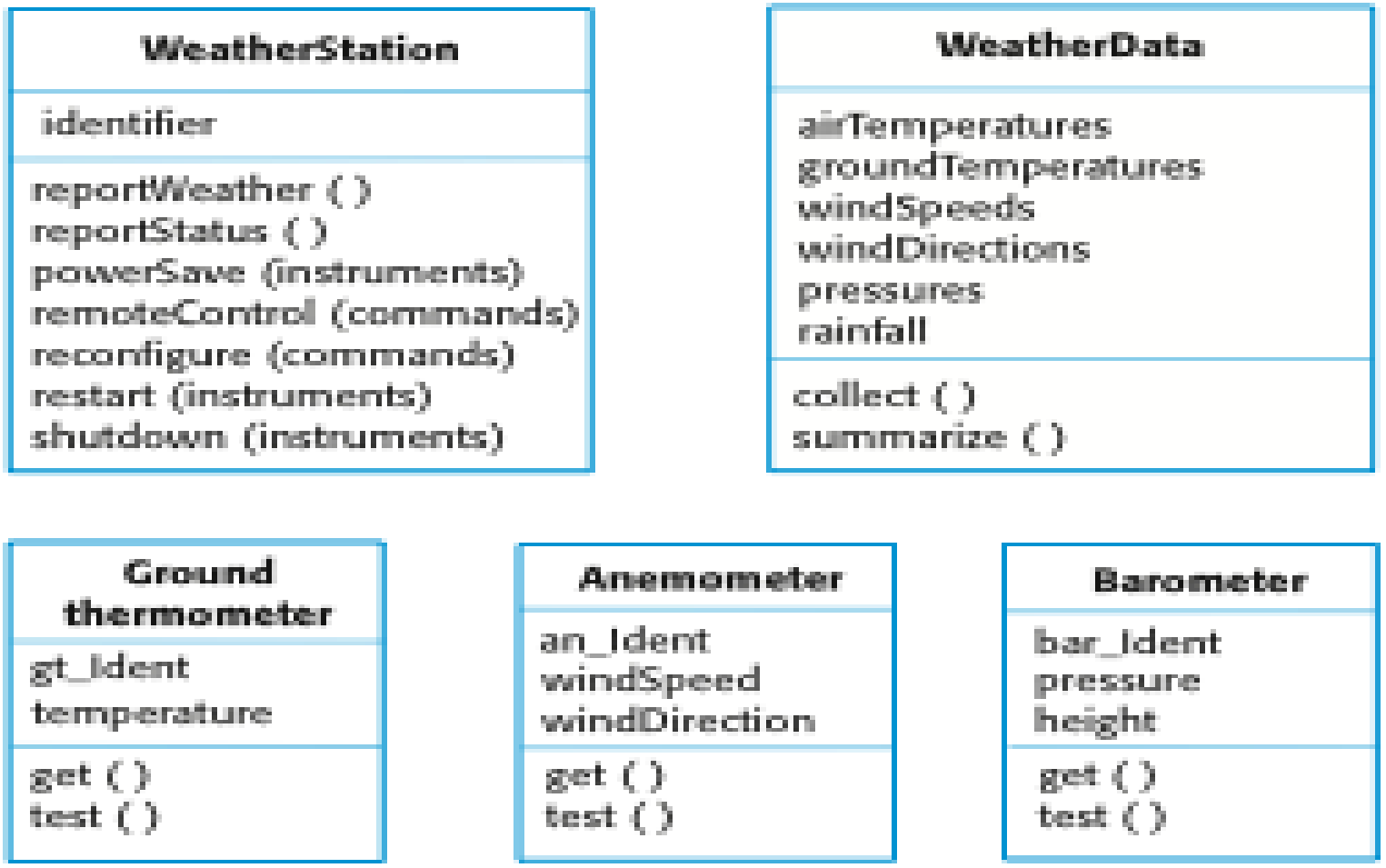
A weather station is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge. Data is collected periodically.

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.

Class identification

Weather station Classes

Weather station object classes



Design models

- Design models show

Examples of design models

- Subsystem models
- th

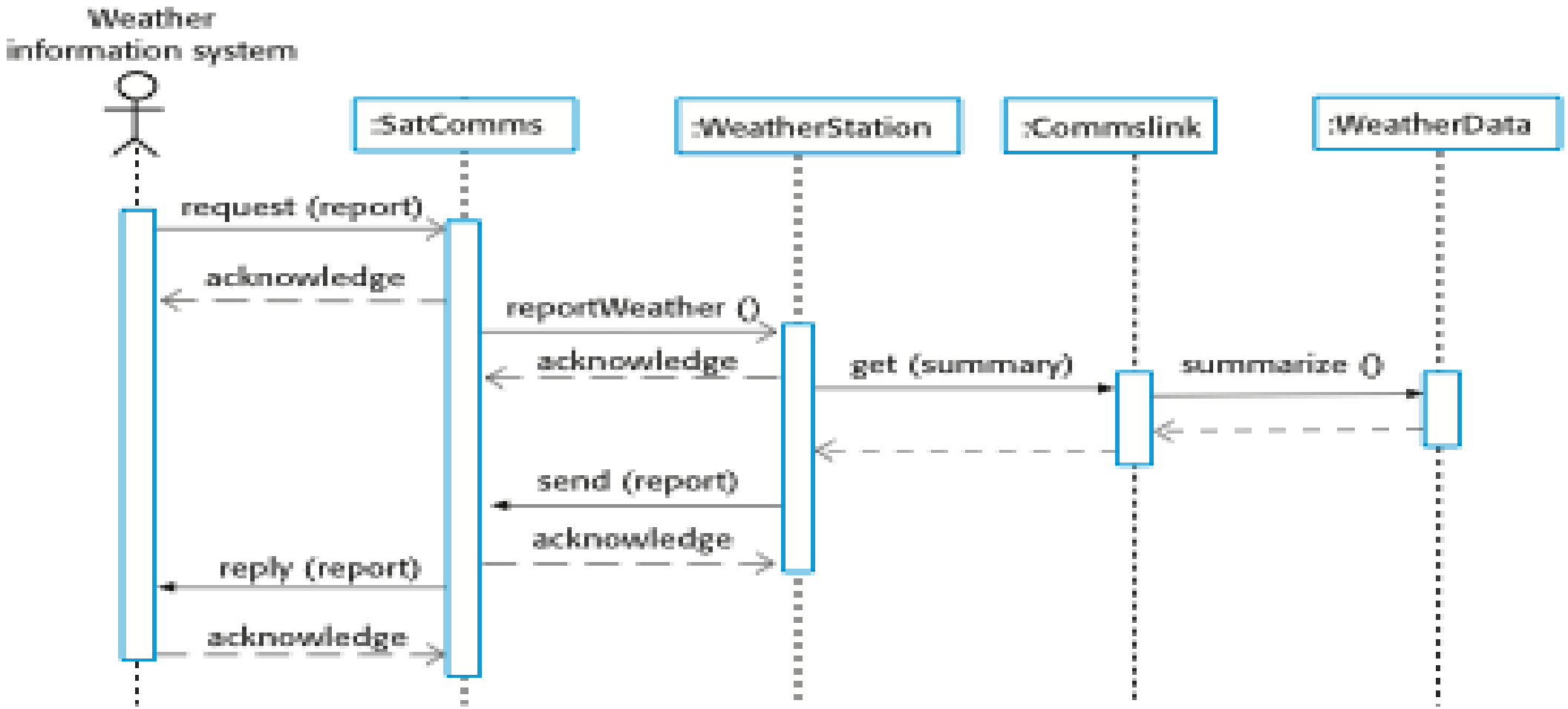
Subsystem models

- Shows how the design

Sequence models

✧ Sequence models show

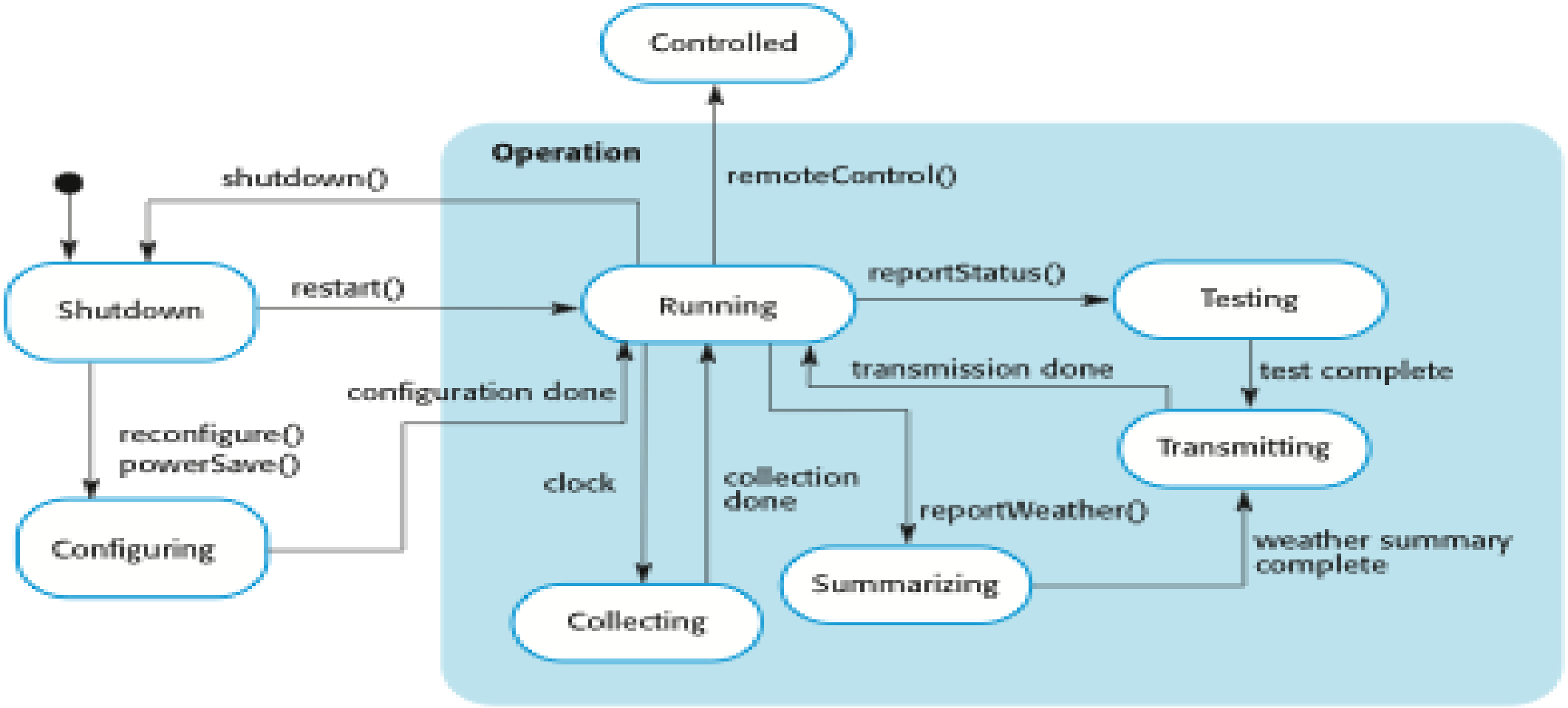
Sequence diagram describing data collection



State diagrams

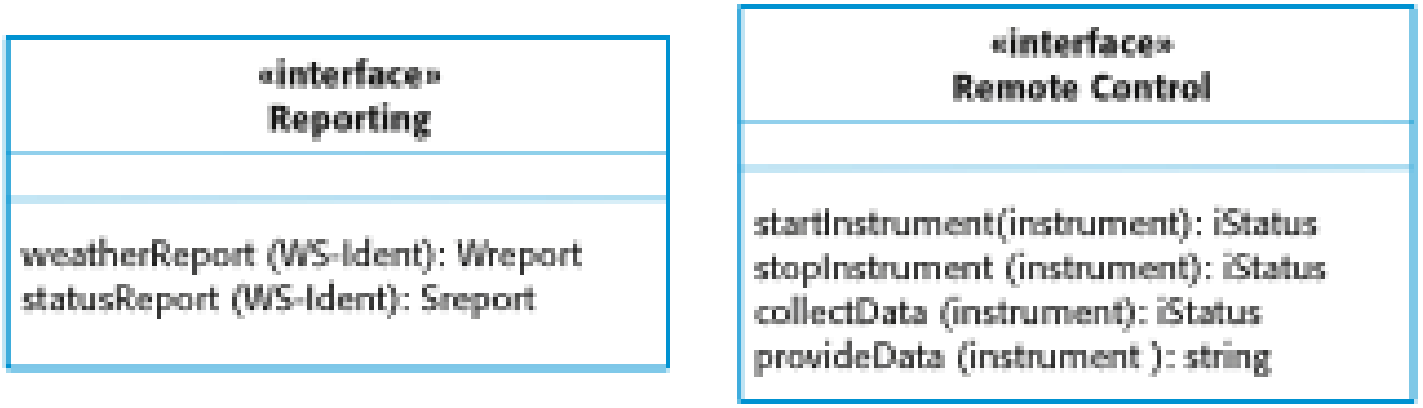
- State diagrams are used

Weather station state diagram



Interface specification

Weather station interfaces



Design patterns

✧ A design pattern is

Design patterns

✧ A design pattern can

Design patterns

Creational Design Patterns

Manage the way objects are created

- Singleton - Ensures that only one instance of a class is created and Provides a global access point to the object

- Factory - A software factory produces objects. And not just that — it does so without specifying the exact class of the object to be created. To accomplish this, objects are created by calling a factory method instead of calling a constructor.

Design patterns

Creational Design Patterns (contd..)

- Builder - Defines an instance for creating an object but letting subclasses decide which class to instantiate and Allows a finer control over the construction process.

- Prototype - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Design patterns

Structural Design Patterns

Define structures of objects and classes that can work together and define how the relations can be defined between entities.

- Adapter - Convert the interface of a class into another interface clients expect

- Bridge - Compose objects into tree structures to represent part-whole Hierarchies

Structural Design Patterns (contd..)

- Flyweight - use sharing to support a large number of objects that have part of their internal state in common where the other part of state can vary

- Memento - capture the internal state of an object without violating encapsulation and thus providing a mean for restoring the object into initial state when needed

Design patterns

Behavioral Design Patterns

Define the interactions and behaviors of classes

- Chain of Responsibility - The objects become parts of a chain and the request is sent from one object to another across the chain until one of the objects will handle it.
- Interpreter - Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language
- Iterator - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation
- Mediator - Define an object that encapsulates how a set of objects interact

Design patterns

Behavioral Design Patterns (contd..)

- Observer - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- Strategy - Define a family of algorithms
- Template Method - Define the skeleton of an algorithm
- Visitor - Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Design problems

- ✧ To use patterns in your design, you need to recognize that any design problem you are facing may have an associated pattern that can be applied.

Implementation issues

- Focus here is not on programming, although this is obviously important, but on other implementation issues that are often not covered in programming texts:
  - Reuse Most modern software is constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code.
  - Configuration management During the development process, you have to keep track of the many different versions of each software component in a configuration management system.
  - Host-target development Production software does not usually execute on the same computer as the software development environment. Rather, you develop it on one computer (the host system) and execute it on a separate computer (the target system).

Reuse

- From the 1960s to the 1990s, most new software was developed from scratch, by writing all code in a high-level programming language.
  - The only significant reuse or software was the reuse of functions and objects in programming language libraries.
- Costs and schedule pressure mean that this approach became increasingly unviable, especially for commercial and Internet-based systems.
- An approach to development based around the reuse of existing software emerged and is now generally used for business and scientific software.

Reuse levels

- ✧ The abstraction level
  - At this level, you don’t reuse software directly but use knowledge of successful abstractions in the design of your software.
- ✧ The object level
  - At this level, you directly reuse objects from a library rather than writing the code yourself.
- ✧ The component level
  - Components are collections of objects and object classes that you reuse in application systems.
- ✧ The system level
  - At this level, you reuse entire application systems.

Configuration management

- Configuration management is the name given to the general process of managing a changing software system.
- The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.

Configuration management activities

- Version management, where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers.
- System integration, where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.
- Problem tracking, where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.

Host-target development

- Most software is developed on one computer (the host), but runs on a separate machine (the target).
- More generally, we can talk about a development platform and an execution platform.
  - A platform is more than just hardware.
  - It includes the installed operating system plus other supporting software such as a database management system or, for development platforms, an interactive development environment.
- Development platform usually has different installed software than execution platform; these platforms may have different architectures.

Open source development

- ✧ Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process
- ✧ Its roots are in the Free Software Foundation (www.fsf.org), which advocates that source code should not be proprietary but rather should always be available for users to examine and modify as they wish.
- ✧ Open source software extended this idea by using the Internet to recruit a much larger population of volunteer developers. Many of them are also users of the code.

Open source issues

- ✧ Should the product that is being developed make use of open source components?
- ✧ Should an open source approach be used for the software's development?

Open source business

- More and more product companies are using an open source approach to development.
- Their business model is not reliant on selling a software product but on selling support for that product.
- They believe that involving the open source community will allow software to be developed more cheaply, more quickly and will create a community of users for the software.

Open source licensing

- A fundamental principle of open-source development is that source code should be freely available, this does not mean that anyone can do as they wish with that code.
  - Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.
  - Some open source developers believe that if an open source component is used to develop a new system, then that system should also be open source.
  - Others are willing to allow their code to be used without this restriction. The developed systems may be proprietary and sold as closed source systems.

License models

- The GNU General Public License (GPL). This is a so-called ‘reciprocal’ license that means that if you use open source software that is licensed under the GPL license, then you must make that software open source.
- The GNU Lesser General Public License (LGPL) is a variant of the GPL license where you can write components that link to open source code without having to publish the source of these components.
- The Berkley Standard Distribution (BSD) License. This is a non-reciprocal license, which means you are not obliged to re-publish any changes or modifications made to open source code. You can include the code in proprietary systems that are sold.

License management

- ✧ Establish a system for maintaining information about open-source components that are downloaded and used.
- ✧ Be aware of the different types of licenses and understand how a component is licensed before it is used.
- ✧ Be aware of evolution pathways for components.
- ✧ Educate people about open source.
- ✧ Have auditing systems in place.
- ✧ Participate in the open source community.