



Objectives

- To introduce architectural design and its role in the software process
- To describe a number of different types of architectural model
- To show how the architecture of a system may be modeled in different ways
- To discuss how domain-specific reference models may be used to compare software architectures

CS-422 Software Engineering Principles
From Software Engineering by I. Sommerville, 1996.

Chapter 13
Slide 2

Topics covered

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

CS-422 Software Engineering Principles
From Software Engineering by I. Sommerville, 1996.

Chapter 13
Slide 3

Architectural parallels

Architects are the technical interface between the customer and the contractor building the system

A bad architectural design for a building cannot be rescued by good construction; the same is true for software

There are specialist types of building and software architects

There are schools or styles of building and software architecture

Architectural design process

System structuring
The system is decomposed into several principal sub-systems and communications between these sub-systems are identified

Control modeling
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

Structure, control and modular decomposition may be based on a particular model or architectural style

However, most systems are heterogeneous in that different parts of the system are based on different models and, in some cases, the system may follow a composite model

The architectural model used affects the performance, robustness, distributability and maintainability of the system

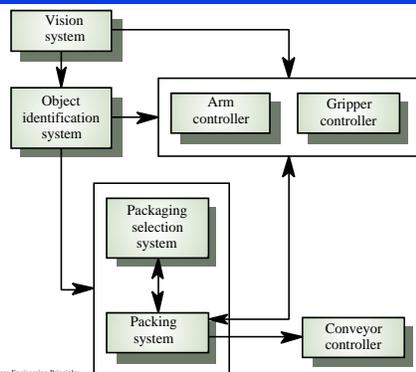
System structuring

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

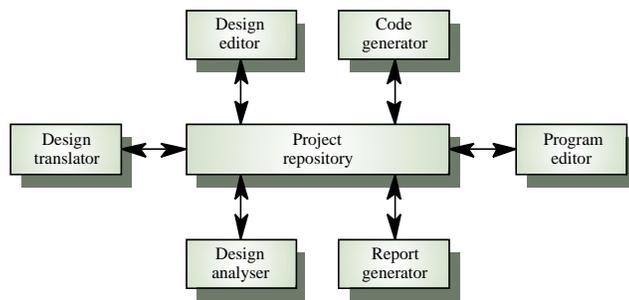
Sub-systems must exchange data. This may be done in two ways:

- 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 produced
- Centralized management e.g. backup, security, etc.
- Sharing model is published as the repository schema

Disadvantages

- Sub-systems must agree on a repository data model. Inevitably a compromise
- Data evolution is difficult and expensive
- No scope for specific management policies
- Difficult to distribute efficiently

Client-server architecture

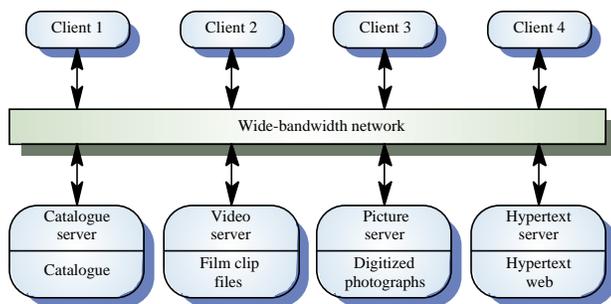
Distributed system model which shows how data and processing is distributed across a range of components

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

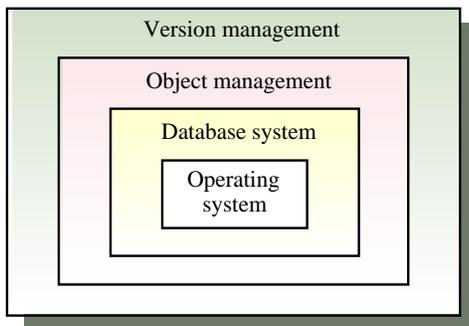
Disadvantages

- 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 hard 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

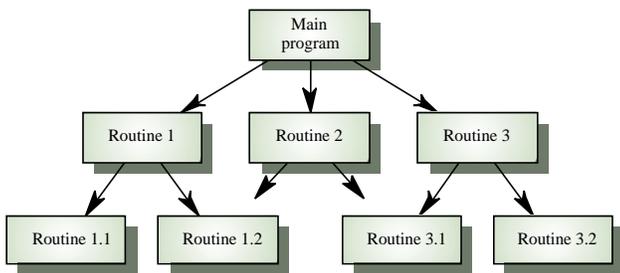
Call-return model

Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems

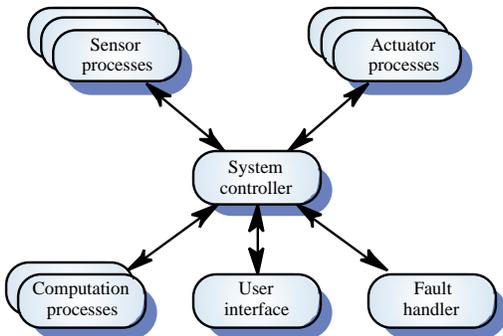
Manager model

Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement

Call-return model



Real-time system control



Event-driven systems

Driven by externally generated events where the timing of the event is outside the control of the sub-systems which process the event

Two principal event-driven models

Broadcast models. An event is broadcast to all sub-systems. Any sub-system which 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

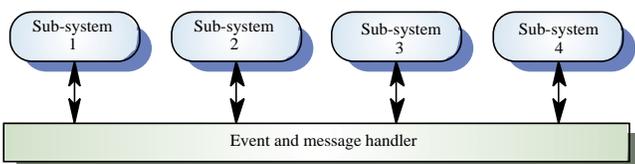
Effective in integrating sub-systems on different computers in a network

Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which 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

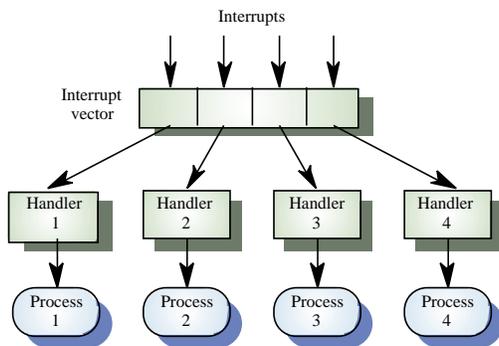
Used in real-time systems where fast response to an event is essential

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 covered

An object model where the system is decomposed into interacting objects

A data-flow model where the system is decomposed into functional modules which transform inputs to outputs. Also known as the pipeline model

If possible, decisions about concurrency should be delayed until modules are implemented

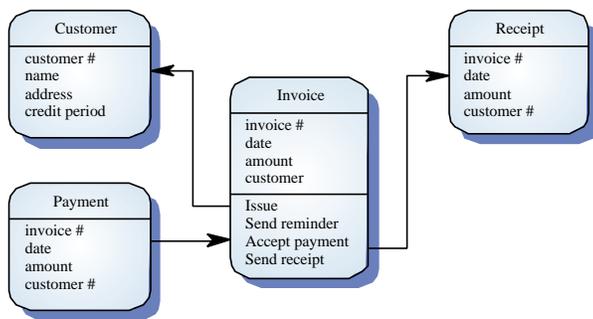
Object models

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

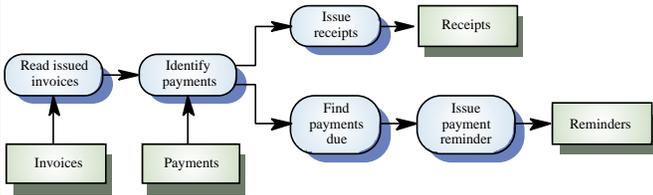
Functional transformations process their inputs to produce outputs

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 which is extensively used in data processing systems

Not really suitable for interactive systems

Invoice processing system



Domain-specific architectures

Architectural models which are specific to some application domain

Two types of domain-specific model

Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems
Reference models which are more abstract, idealized model. 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

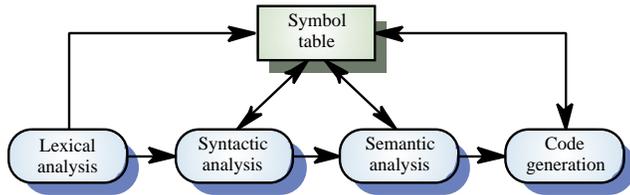
Generic models

Compiler model is a well-known example although other models exist in more specialized application domains

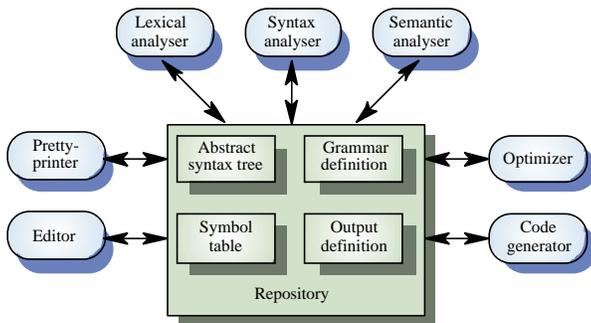
- Lexical analyzer
- Symbol table
- Syntax analyzer
- Syntax tree
- Semantic analyzer
- Code generator

Generic compiler model may be organized according to different architectural models

Compiler model



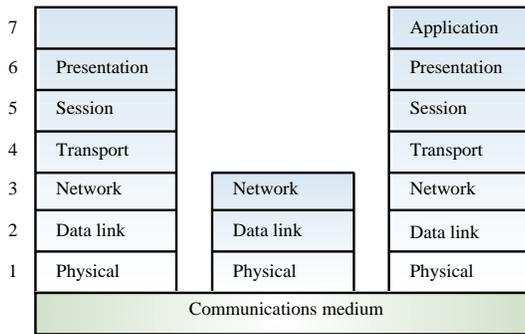
Language processing system



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



Key points

The software architect is responsible for deriving a *structural system model*, a *control model* and a *sub-system decomposition model*

Large systems rarely conform to a single architectural model

System decomposition models include repository models, client-server models and abstract machine models

Control models include centralized control and event-driven models

Key points

Modular decomposition models include data-flow and object models

Domain specific architectural models are abstractions over an application domain. They may be constructed by abstracting from existing systems or may be idealized reference models
