Chapter 10

Architectural Design

Establishing the overall structure of a software system

Topics covered

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

Software architecture

- Architectural design is a software process in which the constituent sub-systems and the framework for sub-system control and communication are identified.
- The output of this design process is a description of the *software architecture.*

Advantages of explicit architecture

It facilitates

- Stakeholder communication
- System analysis
- Large-scale reuse

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 among the subsystems is established.
- Modular decomposition

  The subsystems are further decomposed into modules.

Subsystems vs. modules

- A subsystem is a system in its own right whose operation is independent of the services provided by other subsystems.
- A module is a system component that provides services to other components, and would not normally be considered as a separate system.

Architectural models

- Static structural model that shows the major system components
- Dynamic process model that shows the interaction among the subsystems
- Interface model that defines subsystem interfaces
- Relationships model such as a 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.
- Most large systems, however, are heterogeneous and do not follow a single architectural style.

Architecture attributes

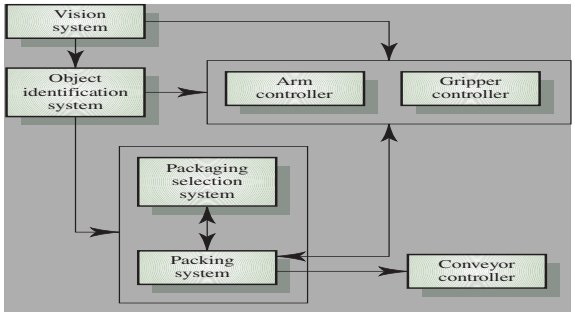
- Performance
  Localize operations to minimize sub-system communication
- Security
  Use a layered architecture with critical assets in inner layers
- Safety
  Isolate safety-critical components
- Availability
  Include redundant components in the architecture
- Maintainability
  Use fine-grain, self-contained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

System structuring

is the first phase of architectural design in which

- the system is decomposed into interacting subsystems,
- a block diagram is used to present an overview of the system structure, and
- more specific models may be developed to show how subsystems share data, are distributed, and interface with each other.

Example: Packing robot control system

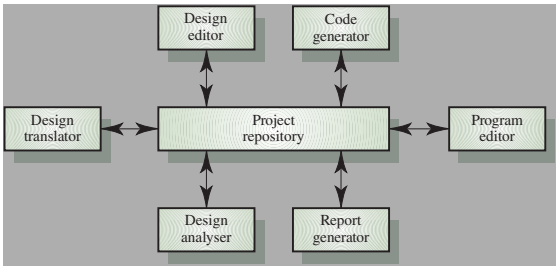


System structuring:

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

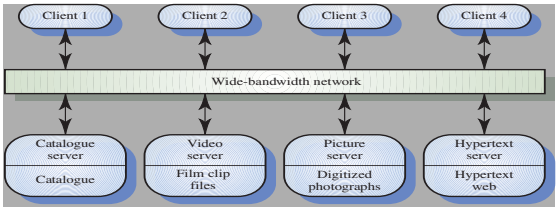
- Advantages
  - Efficient way to share large amounts of data
  - Sub-systems need not be concerned with how data is produced
  - Operations such backup, security, etc., can be centralized
  - The model of sharing is visible through the repository schema
- Disadvantages
  - Sub-systems must agree on a repository data model.
  - Data evolution is difficult and expensive
  - All subsystems are forced to adopt same security and recovery policies
  - Difficult to distribute the repository efficiently

System structuring:

Client-server architecture

- A distributed system model in which data and processing is distributed among components.
- A set of stand-alone servers, each of which provide specific services, such as printing, data management, etc.
- A set of clients that makes use these services.
- A network that allows clients to access servers.

Film and picture library



Client-server characteristics

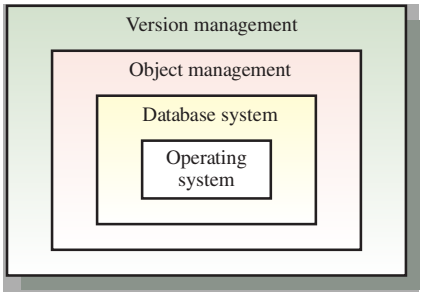
- Advantages
  - Distribution of data is straightforward.
  - Makes effective use of networked systems. May require cheaper hardware.
  - Easy to add new servers or upgrade existing servers.
- Disadvantages
  - No shared data model so sub-systems use different data organization.
  - Data interchange 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

System structuring:

Abstract machine model

- It is used to model the interfacing of sub-systems.
- It organizes the system into a set of layers (or abstract machines), each of which provide a set of services.
- It supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- Often it is difficult to structure systems in this way.

Version management system



Control models

After the system is decomposed into subsystem, a control model is established to describe how they interact.

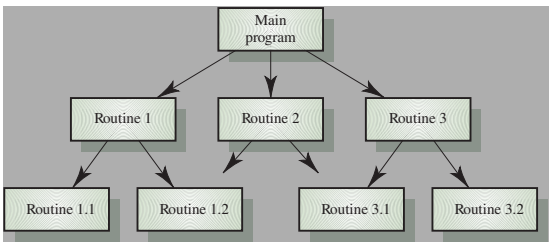
- Centralized control
  One sub-system has overall responsibility for control and starts and stops other sub-systems.
- Event-driven systems
  Each sub-system can respond to external events.

Control models:

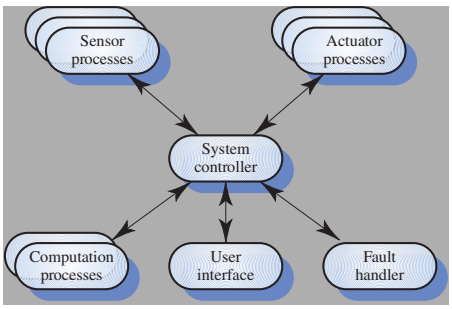
Centralized control

- A control subsystem 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



Control models:

Event-driven systems

- Driven by occurrences of external event, the occurrence of which is not under the control of the system.
- 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. Used in distributed or multiprocessor systems
  - Interrupt-driven models. The occurrence of an event is detected by an interrupt handler that invoke an appropriate handler. Used in single processor systems.

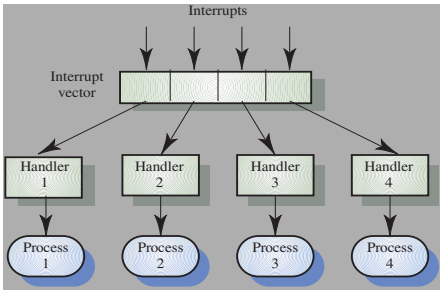
Broadcast model

- 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.

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

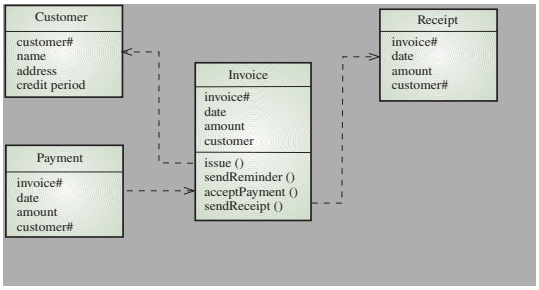
- Two modular decomposition models:
  - An object model where the system is decomposed into interacting objects
  - A data-flow (or pipe and filter) model where the system is decomposed into functional modules which transform inputs to outputs.

Module decomposition:

Object models

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

Object models (continued)



Object models (continued)

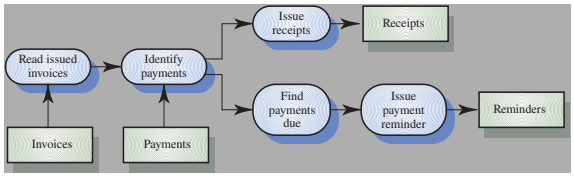
- Advantages:
  - Implementation can be modified without affecting other objects.
  - Structure of the system is readily understandable.
  - More amendable to reuse.
- Disadvantages:
  - If an interface change is required, the effect of that change on all users of the changed object must be evaluated.
  - While objects may map cleanly to small-scale real-world entities, more complex entities are sometimes difficult to be represented as objects.

Module decomposition:

Data-flow models

- Functional transformations are applied to their inputs to produce outputs.
- Also known as pipe and filter model in UNIX.
- Commonly used in batch data processing systems
- Not suitable for interactive systems

Invoice processing system



Domain-specific architectures

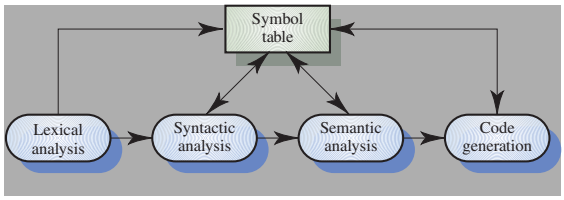
- Two types of domain-specific model
  - Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems
  - Reference models which are more abstract, idealized model. Provide a means of informing the designers about that class of system and of comparing different architectures

Domain specific architecture:

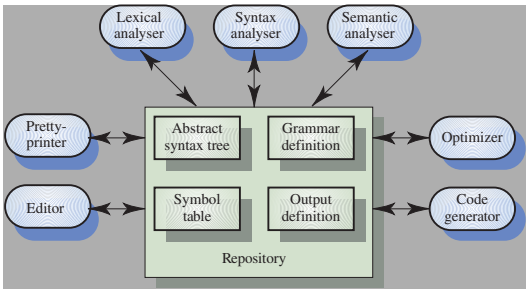
Generic models

- Compiler model is a well-known example
  - 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



Language processing system

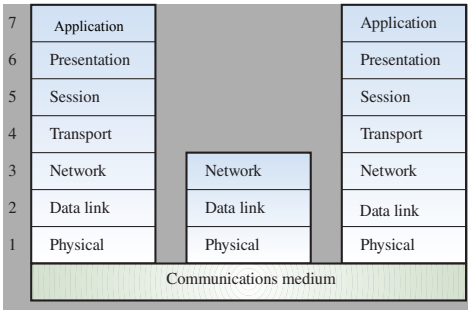


Domain specific architecture:

Reference architectures

- Reference models are derived from a study of the application domain rather than from existing systems.
- Often used as a standard for system implementation or comparison of different systems.
- Open system interconnection (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 (continued)

- Modular decomposition models include data-flow and object models.
- Domain specific architectural models are abstractions over an application domain.