Name	Description	Type	Date
has-labels	1:N relation between entities of type Node or Link and entities of type Label.	Relation	5.10.1998
Label	Holds structured or unstructured information about nodes or links. Labels are represented by an icon (which can be a transparent box) and associated text.	Entity	8.12.1998
Link	A 1:1 relation between design entities represented as nodes. Links are typed and may be named.	Relation	8.12.1998
name (label)	Each label has a name which identifies the type of label. The name must be unique within the set of label types used in a design.	Attribute	8.12.1998
name (node)	Each node has a name which must be unique within a design. The name may be up to 64 characters long.	Attribute	15.11.1998

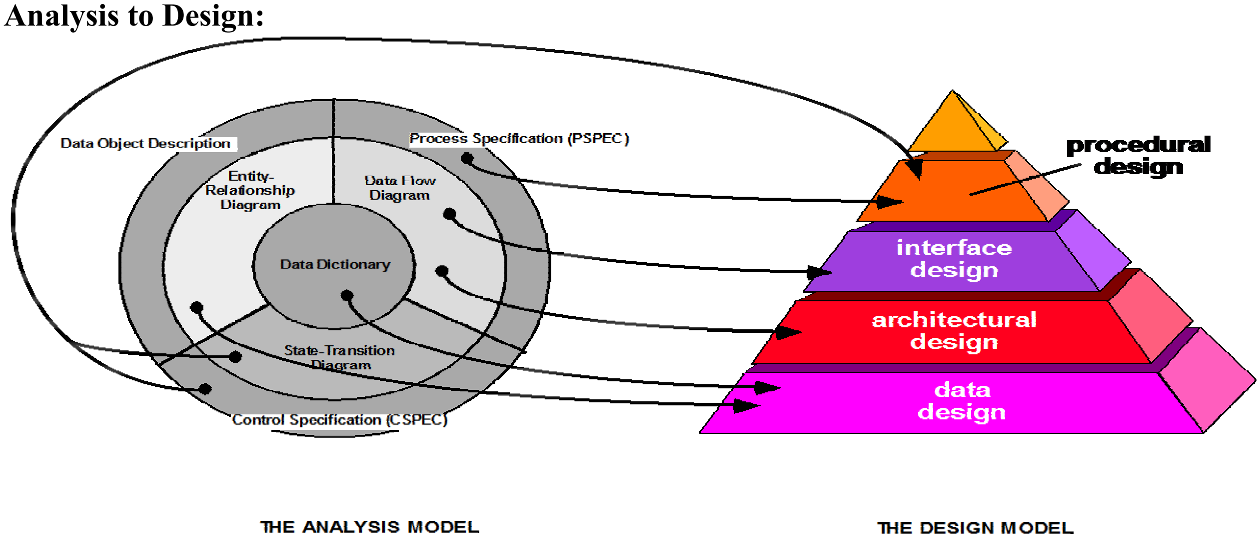
UNIT III

ANALYSIS, DESIGN CONCEPTS AND PRINCIPLES

Design Concepts and Principles:

- Map the information from the analysis model to the design representations - data design, architectural design, interface design, procedural design

Analysis to Design:



Design Models – 1:

- **Data Design**
  - created by transforming the data dictionary and ERD into implementation data structures
  - requires as much attention as algorithm design
- **Architectural Design**
  - derived from the analysis model and the subsystem interactions defined in the DFD
- **Interface Design**
  - derived from DFD and CFD
  - describes software elements communication with
    - other software elements
    - other systems
    - human users

Design Models – 2 :

- Procedure-level design
  - created by transforming the structural elements defined by the software architecture into procedural descriptions of software components
  - Derived from information in the PSPEC, CSPEC, and STD

Design Principles – 1:

- Process should not suffer from tunnel vision – consider alternative approaches
- Design should be traceable to analysis model
- Do not try to reinvent the wheel

- use design patterns ie reusable components

- Design should exhibit both uniformity and integration
- Should be structured to accommodate changes

Design Principles – 2 :

- Design is not coding and coding is not design
- Should be structured to degrade gently, when bad data, events, or operating conditions are encountered
- Needs to be assessed for quality as it is being created
- Needs to be reviewed to minimize conceptual (semantic) errors

Design Concepts -1 :

- Abstraction
  - allows designers to focus on solving a problem without being concerned about irrelevant lower level details

Procedural abstraction is a named sequence of instructions that has a specific and limited function

e.g open a door

Open implies a long sequence of procedural steps

data abstraction is collection of data that describes a data object

e.g door type, opening mech, weight,dimen

Design Concepts -2 :

- Design Patterns
  - description of a design structure that solves a particular design problem within a specific context and its impact when applied

Design Concepts -3 :

- Software Architecture
  - overall structure of the software components and the ways in which that structure
  - provides conceptual integrity for a system

Design Concepts -4 :

- Information Hiding
  - information (data and procedure) contained within a module is inaccessible to modules that have no need for such information
- Functional Independence
  - achieved by developing modules with single-minded purpose and an aversion to excessive interaction with other models

Refactoring – Design concepts :

- Fowler [FOW99] defines refactoring in the following manner:
  - "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.‖
- When software is refectories, the existing design is examined for
  - redundancy
  - unused design elements
  - inefficient or unnecessary algorithms
  - poorly constructed or inappropriate data structures
  - or any other design failure that can be corrected to yield a better design.

Design Concepts – 4 :

- Objects
  - encapsulate both data and data manipulation procedures needed to describe the content and behavior of a real world entity
- Class
  - generalized description (template or pattern) that describes a collection of similar objects
- Inheritance
  - provides a means for allowing subclasses to reuse existing superclass data and procedures; also provides mechanism for propagating changes

Design Concepts – 5:

- Messages
  - the means by which objects exchange information with one another
- Polymorphism
  - a mechanism that allows several objects in an class hierarchy to have different methods with the same name
  - instances of each subclass will be free to respond to messages by calling their own version of the method

Modular Design Methodology Evaluation – 1:

Modularity

- the degree to which software can be understood by examining its components independently of one another

- Modular decomposability
  - provides systematic means for breaking problem into sub problems

- Modular compos ability
  - supports reuse of existing modules in new systems
- Modular understandability
  - module can be understood as a stand-alone unit

Modular Design Methodology Evaluation – 2:

- Modular continuity
  - module change side-effects minimized
- Modular protection
  - processing error side-effects minimized

Effective Modular Design:

- Functional independence
  - modules have high cohesion and low coupling
- Cohesion
  - qualitative indication of the degree to which a module focuses on just one thing
- Coupling
  - qualitative indication of the degree to which a module is connected to other modules and to the outside world

Architectural Design:

Why Architecture?

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

(1) analyze the effectiveness of the design in meeting its stated requirements,

(2) consider architectural alternatives at a stage when making design changes is still relatively easy, and

(3) reduce the risks associated with the construction of the software.

Importance :

- Software architecture representations enable communications among stakeholders
- Architecture highlights early design decisions that will have a profound impact on the ultimate success of the system as an operational entity
- The architecture constitutes an intellectually graspable model of how the system is structured and how its components work together

Architectural Styles – 1:

- Data centered
  - file or database lies at the center of this architecture and is accessed frequently by other components that modify data

Architectural Styles – 2:

- Data flow
  - input data is transformed by a series of computational components into output data
  - Pipe and filter pattern has a set of components called filters, connected by pipes that transmit data from one component to the next.
  - If the data flow degenerates into a single line of transforms, it is termed batch sequential
- Object-oriented
  - components of system encapsulate data and operations, communication between components is by message passing

- Layered
    - several layers are defined
    - each layer performs operations that become closer to the machine instruction set in the lower layers

Architectural Styles – 3:

Call and return

- program structure decomposes function into control hierarchy with main program invoking several subprograms

Software Architecture Design – 1:

- Software to be developed must be put into context
    - model external entities and define interfaces
- Identify architectural archetypes
    - collection of abstractions that must be modeled if the system is to be constructed

Object oriented Architecture :

- The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components is accomplished via message passing

Software Architecture Design – 2:

- Specify structure of the system
    - define and refine the software components needed to implement each archetype
- Continue the process iteratively until a complete architectural structure has been derived

Layered Architecture:

- Number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set
- At the outer layer –components service user interface operations.
- At the inner layer – components perform operating system interfacing.
- Intermediate layers provide utility services and application software function

Architecture Tradeoff Analysis – 1:

1. Collect scenarios
2. Elicit requirements, constraints, and environmental description
3. Describe architectural styles/patterns chosen to address scenarios and requirements
    - module view
    - process view
    - data flow view

Architecture Tradeoff Analysis – 2:

4. Evaluate quality attributes independently (e.g. reliability, performance, security, maintainability, flexibility, testability, portability, reusability, interoperability)
5. Identify sensitivity points for architecture
    - any attributes significantly affected by changing in the architecture

Refining Architectural Design:

- Processing narrative developed for each module
- Interface description provided for each module
- Local and global data structures are defined
- Design restrictions/limitations noted
- Design reviews conducted

- Refinement considered if required and justified

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.

Architecture and system characteristics

- Performance
  - Localise critical operations and minimise communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localise safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

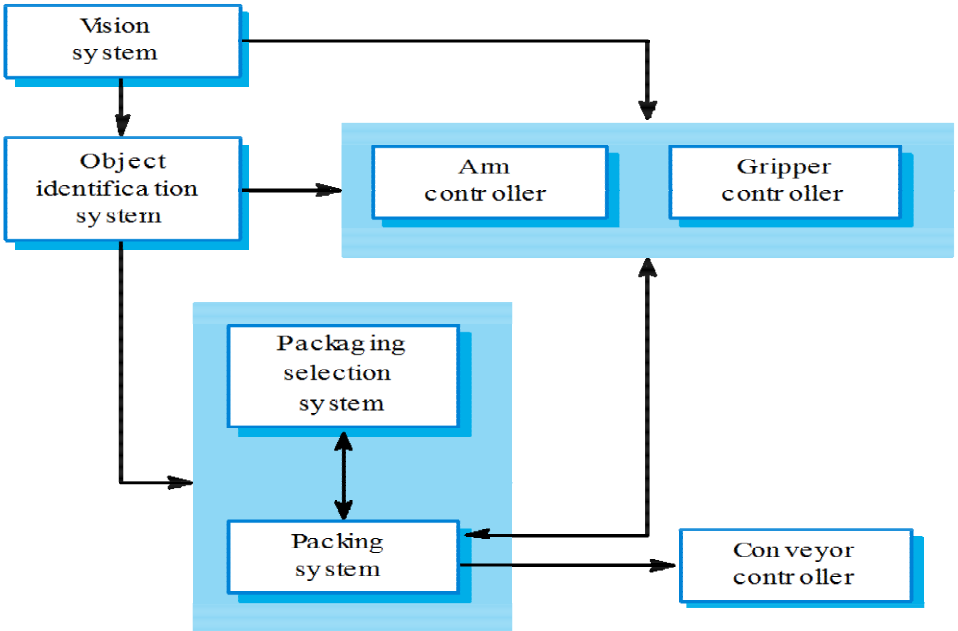
Architectural conflicts

- Using large-grain components improves performance but reduces maintainability.
- Introducing redundant data improves availability but makes security more difficult.
- Localising safety-related features usually means more communication so degraded performance.

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



Box and line diagrams

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

Architectural design decisions

- Architectural design is a creative process so the process differs depending on the type of system being developed.
- However, a number of common decisions span all design processes.
- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

Architecture reuse

- Systems in the same domain often have similar architectures that reflect domain concepts.
- Application product lines are built around a core architecture with variants that satisfy particular customer requirements.

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.

Architectural models

- Used to document an architectural design.

- Static structural model that shows the major system components.
- Dynamic process model that shows the process structure of the system.
- Interface model that defines sub-system interfaces.
- Relationships model such as a data-flow model that shows sub-system relationships.
- Distribution model that shows how sub-systems are distributed across computers.

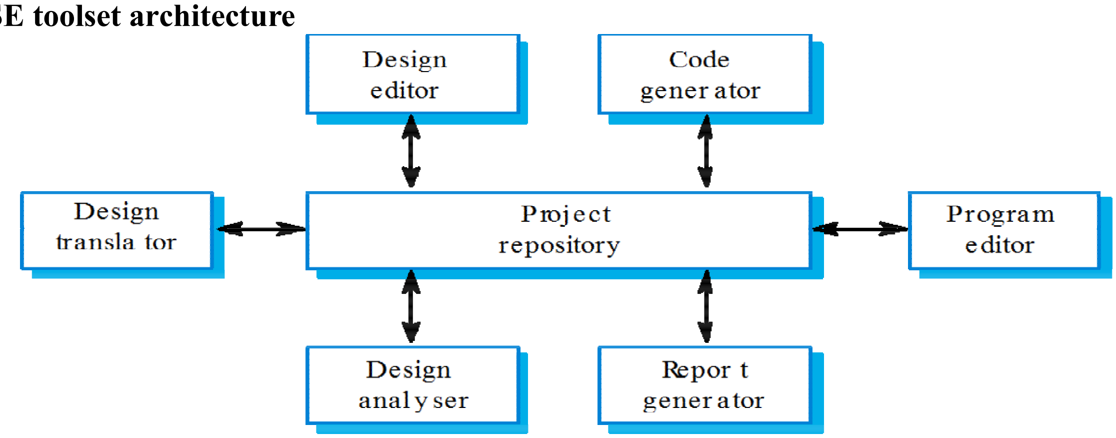
System organisation

- 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



Repository model characteristics

Advantages

- Efficient way to share large amounts of data;
- Sub-systems need not be concerned with how data is produced Centralised 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 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.

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 organisation. 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 (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.

Modular decomposition styles

- Styles of decomposing sub-systems into modules.
- No rigid distinction between system organisation and modular decomposition.

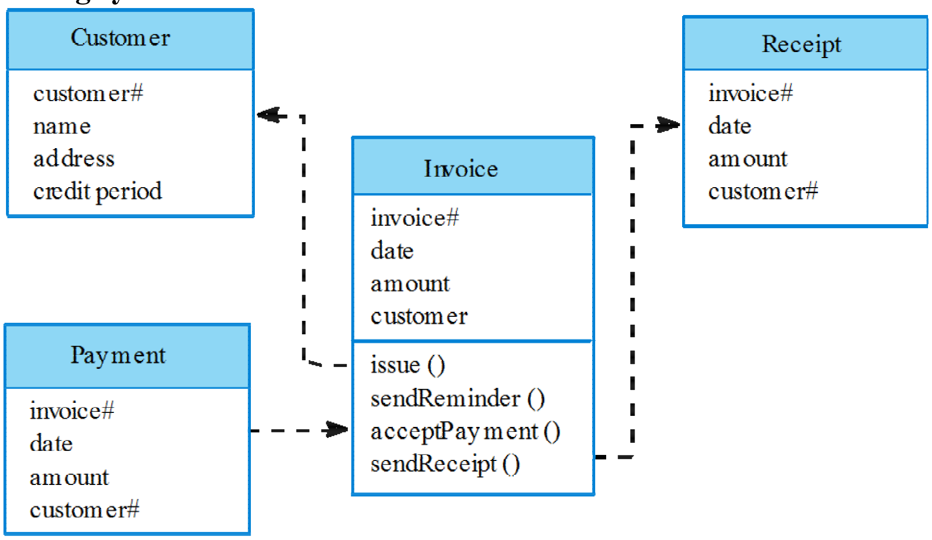
Sub-systems and modules

- 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.
- 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.
- 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



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.

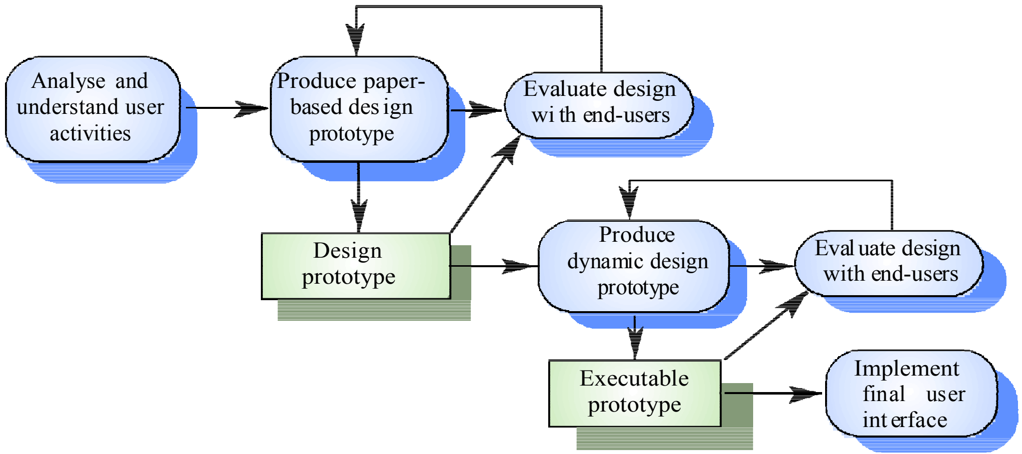
Function-oriented pipelining

- 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 systems.

User interface design

- Designing effective interfaces for software systems
- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used
- Most users of business systems interact with these systems through graphical user interfaces (GUIs)
- In some cases, legacy text-based interfaces are still used

User interface design process



UI design principles

- User familiarity
  - The interface should be based on user-oriented terms and concepts rather than computer concepts
  - E.g., an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.
- Consistency
  - The system should display an appropriate level of consistency
  - Commands and menus should have the same format, command punctuation should be similar, etc.
- Minimal surprise
  - If a command operates in a known way, the user should be able to predict the operation of comparable commands
- Recoverability
  - The system should provide some interface to user errors and allow the user to recover from errors
- User guidance
  - Some user guidance such as help systems, on-line manuals, etc. should be supplied
- User diversity
  - Interaction facilities for different types of user should be supported
  - E.g., some users have seeing difficulties and so larger text should be available

User-system interaction

- Two problems must be addressed in interactive systems design
  - How should information from the user be provided to the computer system?
  - How should information from the computer system be presented to the user?

Interaction styles

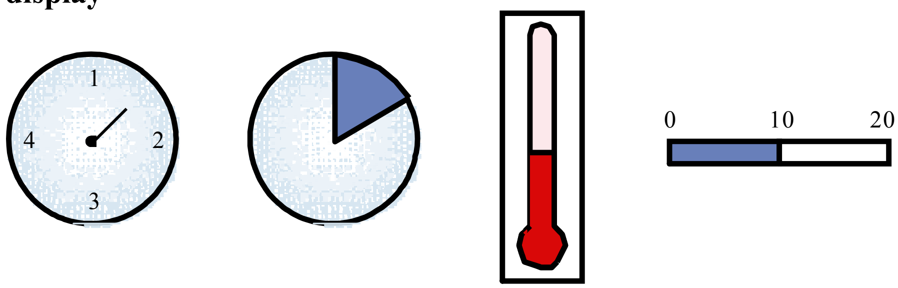
- Direct manipulation
  - Easiest to grasp with immediate feedback
  - Difficult to program
- Menu selection
  - User effort and errors minimized
  - Large numbers and combinations of choices a problem

- Form fill-in
  - Ease of use, simple data entry
  - Tedious, takes a lot of screen space
- Natural language
  - Great for casual users
  - Tedious for expert users

Information presentation

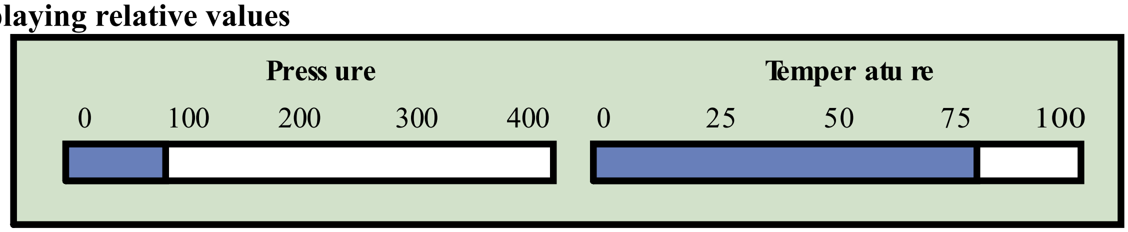
- Information presentation is concerned with presenting system information to system users
- The information may be presented directly or may be transformed in some way for presentation
- The Model-View-Controller approach is a way of supporting multiple presentations of data

Information display

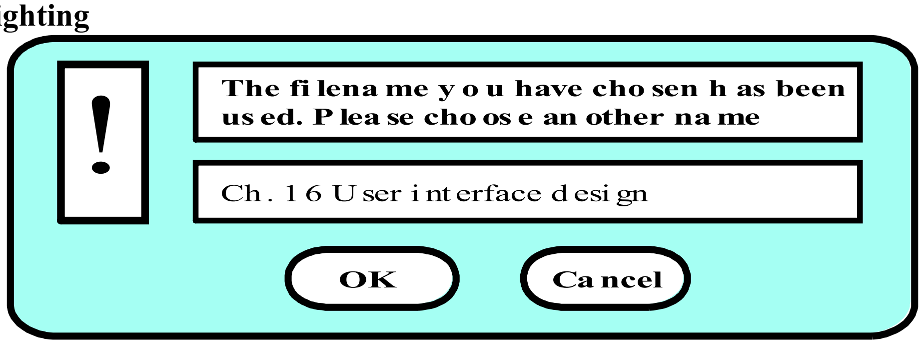


Dial with needle   Pie chart   Thermometer   Horizontal bar

Displaying relative values



Textual highlighting



Data visualisation

- Concerned with techniques for displaying large amounts of information

- Visualisation can reveal relationships between entities and trends in the data
- Possible data visualisations are:
    - Weather information
    - State of a telephone network
    - Chemical plant pressures and temperatures
    - A model of a molecule

Colour displays

- Colour adds an extra dimension to an interface and can help the user understand complex information structures
- Can be used to highlight exceptional events
    - The use of colour to communicate meaning

Error messages

- Error message design is critically important. Poor error messages can mean that a user rejects rather than accepts a system
- Messages should be polite, concise, consistent and constructive
- The background and experience of users should be the determining factor in message design

User interface evaluation

- Some evaluation of a user interface design should be carried out to assess its suitability
- Full scale evaluation is very expensive and impractical for most systems
- Ideally, an interface should be evaluated against req
- However, it is rare for such specifications to be produced

Real Time Software Design

- Systems which monitor and control their environment
- Inevitably associated with hardware devices
    - Sensors: Collect data from the system environment
    - Actuators: Change (in some way) the system's environment
- Time is critical. Real-time systems MUST respond within specified times
- A real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced
- A ‗soft' real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements
- A ‗hard' real-time system is a system whose operation is incorrect if results are not produced according to the timing specification

Stimulus/Response Systems

- Given a stimulus, the system must produce a response within a specified time
- 2 classes
- Periodic stimuli. Stimuli which occur at predictable time intervals
    - For example, a temperature sensor may be polled 10 times per second
- Aperiodic stimuli. Stimuli which occur at unpredictable times
    - For example, a system power failure may trigger an interrupt which must be processed by the system

Architectural considerations

- Because of the need to respond to timing demands made by different stimuli / responses, the system architecture must allow for fast switching between stimulus handlers
- Timing demands of different stimuli are different so a simple sequential loop is not usually adequate

Real –Time Software Design:

- Designing embedded software systems whose behaviour is subject to timing constraints
- To explain the concept of a real-time system and why these systems are usually implemented as concurrent processes
- To describe a design process for real-time systems
- To explain the role of a real-time executive
- To introduce generic architectures for monitoring and control and data acquisition systems

Real-time systems:

- Systems which monitor and control their environment
- Inevitably associated with hardware devices
  - Sensors: Collect data from the system environment
  - Actuators: Change (in some way) the system's environment
- Time is critical. Real-time systems MUST respond within specified times

Definition:

- A real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced
- A ‗soft‘ real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements
- A ‗hard‘ real-time system is a system whose operation is incorrect if results are not produced according to the timing specification

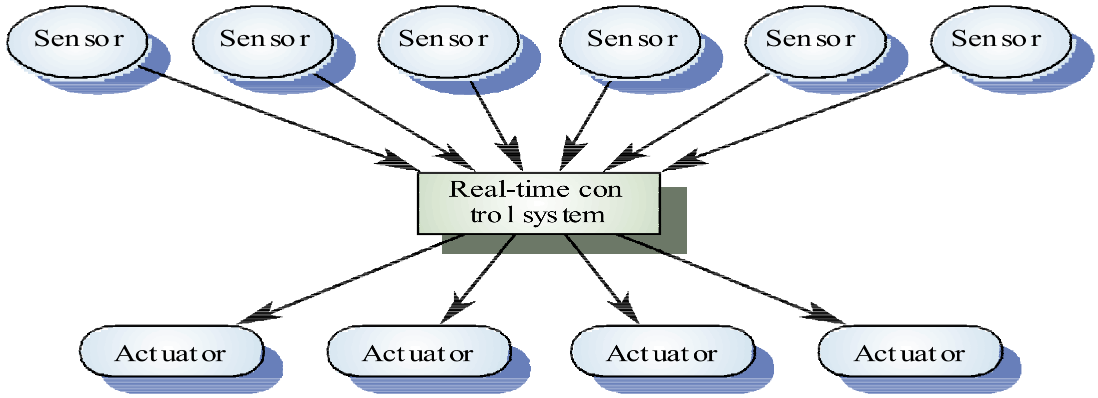
Stimulus/Response Systems:

- Given a stimulus, the system must produce a esponse within a specified time
- Periodic stimuli. Stimuli which occur at predictable time intervals
  - For example, a temperature sensor may be polled 10 times per second
- Aperiodic stimuli. Stimuli which occur at unpredictable times
  - For example, a system power failure may trigger an interrupt which must be processed by the system

Architectural considerations:

- Because of the need to respond to timing demands made by different stimuli/responses, the system architecture must allow for fast switching between stimulus handlers
- Timing demands of different stimuli are different so a simple sequential loop is not usually adequate
- Real-time systems are usually designed as cooperating processes with a real-time executive controlling these processes

A real-time system model:



System elements:

- Sensors control processes
  - Collect information from sensors. May buffer information collected in response to a sensor stimulus
- Data processor
  - Carries out processing of collected information and computes the system response
- Actuator control
  - Generates control signals for the actuator

R-T systems design process:

- Identify the stimuli to be processed and the required responses to these stimuli
- For each stimulus and response, identify the timing constraints
- Aggregate the stimulus and response processing into concurrent processes. A process may be associated with each class of stimulus and response
- Design algorithms to process each class of stimulus and response. These must meet the given timing requirements
- Design a scheduling system which will ensure that processes are started in time to meet their deadlines
- Integrate using a real-time executive or operating system

Timing constraints:

- May require extensive simulation and experiment to ensure that these are met by the system
- May mean that certain design strategies such as object-oriented design cannot be used because of the additional overhead involved
- May mean that low-level programming language features have to be used for performance reasons

Real-time programming:

- Hard-real time systems may have to programmed in assembly language to ensure that deadlines are met
- Languages such as C allow efficient programs to be written but do not have constructs to support concurrency or shared resource management
- Ada as a language designed to support real-time systems design so includes a general purpose concurrency mechanism

Non-stop system components:

- Configuration manager
  - Responsible for the dynamic reconfiguration of the system software and hardware. Hardware modules may be replaced and software upgraded without stopping the systems
- Fault manager
  - Responsible for detecting software and hardware faults and taking appropriate actions (e.g. switching to backup disks) to ensure that the system continues in operation

Burglar alarm system e.g

- A system is required to monitor sensors on doors and windows to detect the presence of intruders in a building
- When a sensor indicates a break-in, the system switches on lights around the area and calls police automatically
- The system should include provision for operation without a mains power supply
- Sensors
  - Movement detectors, window sensors, door sensors.
  - 50 window sensors, 30 door sensors and 200 movement detectors
  - Voltage drop sensor
- Actions
  - When an intruder is detected, police are called automatically.
  - Lights are switched on in rooms with active sensors.
  - An audible alarm is switched on.
  - The system switches automatically to backup power when a voltage drop is detected.

The R-T system design process:

- Identify stimuli and associated responses
- Define the timing constraints associated with each stimulus and response
- Allocate system functions to concurrent processes
- Design algorithms for stimulus processing and response generation
- Design a scheduling system which ensures that processes will always be scheduled to meet their deadlines

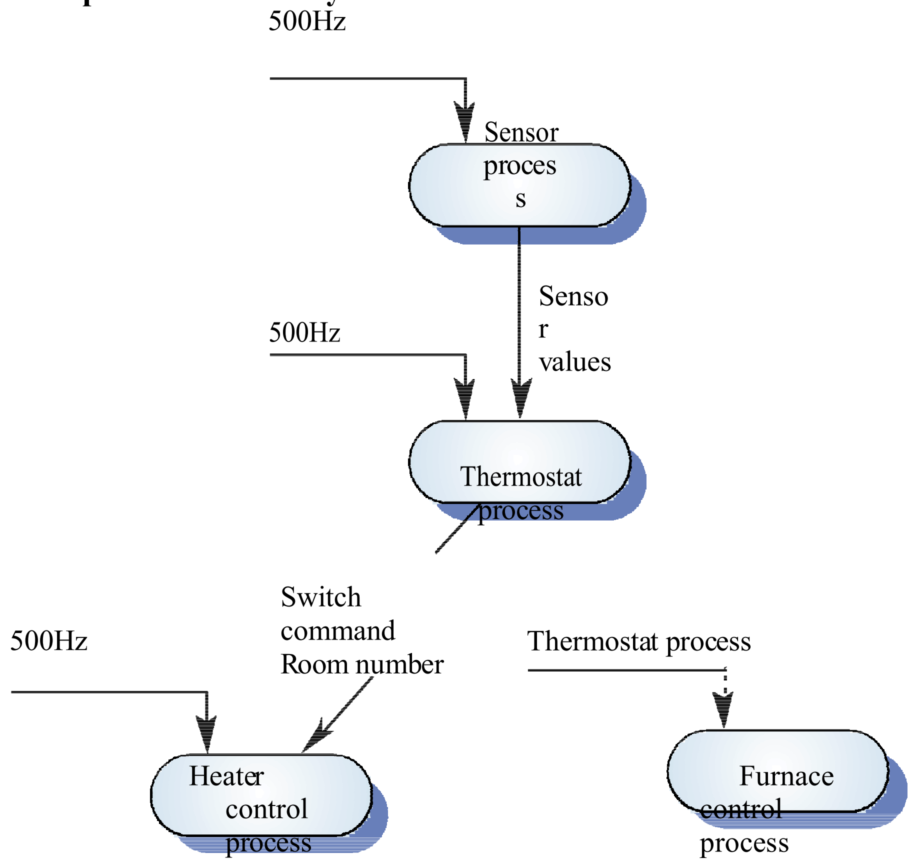
Control systems:

- A burglar alarm system is primarily a monitoring system. It collects data from sensors but no real-time actuator control
- Control systems are similar but, in response to sensor values, the system sends control signals to actuators
- An example of a monitoring and control system is a system which monitors temperature and switches heaters on and off

Data acquisition systems:

- Collect data from sensors for subsequent processing and analysis.
- Data collection processes and processing processes may have different periods and deadlines.
- Data collection may be faster than processing e.g. collecting information about an explosion.
- Circular or ring buffers are a mechanism for smoothing speed differences.

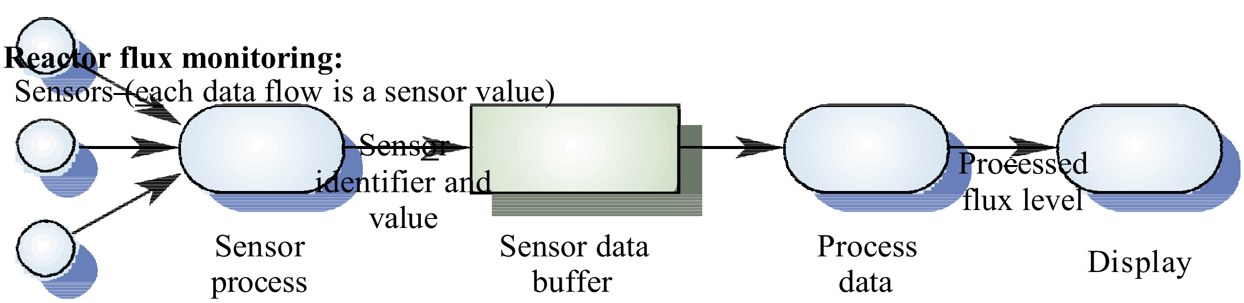
A temperature control system:



Reactor data collection:

- A system collects data from a set of sensors monitoring the neutron flux from a nuclear reactor.
- Flux data is placed in a ring buffer for later processing.
- The ring buffer is itself implemented as a concurrent process so that the collection and processing processes may be synchronized.

Reactor flux monitoring:



Mutual exclusion:

- Producer processes collect data and add it to the buffer. Consumer processes take data from the buffer and make elements available

- Producer and consumer processes must be mutually excluded from accessing the same element.

The buffer must stop producer processes adding information to a full buffer and consumer processes trying to take information from an empty buffer

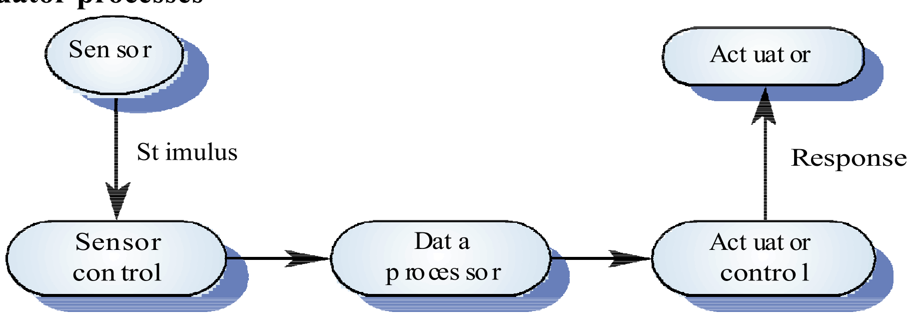
System Design

- Design both the hardware and the software associated with system. Partition functions to either hardware or software
- Design decisions should be made on the basis on non-functional system requirements
- Hardware delivers better performance but potentially longer development and less scope for change

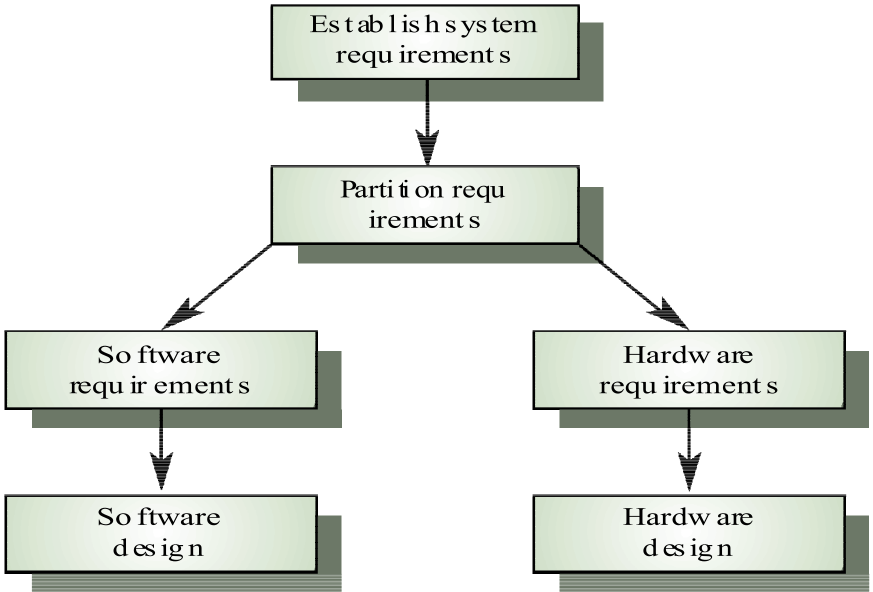
System elements

- Sensors control processes
  - Collect information from sensors. May buffer information collected in response t o a sensor stimulus
- Data processor
  - Carries out processing of collected information and computes the system response
- Actuator control
  - Generates control signals for the actuator

Sensor/actuator processes



Hardware and software design



R-T systems design process

- Identify the stimuli to be processed and the required responses to these stimuli
- For each stimulus and response, identify the timing constraints
- Aggregate the stimulus and response processing into concurrent processes. A process may be associated with each class of stimulus and response
- Design algorithms to process each class of stimulus and response. These must meet the given timing requirements
- Design a scheduling system which will ensure that processes are started in time to meet their deadlines
- Integrate using a real-time executive or operating system

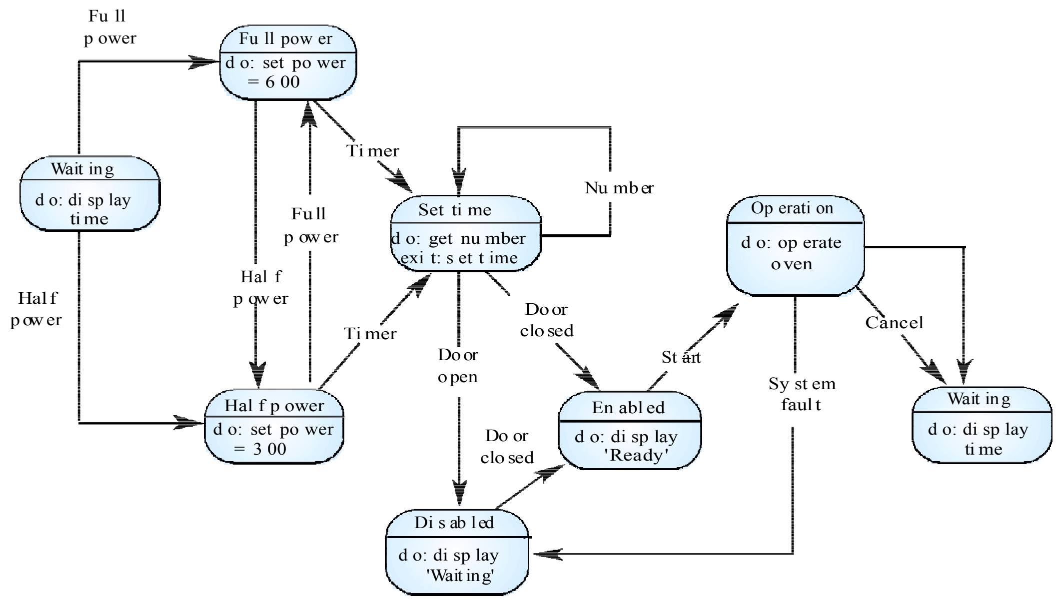
Timing constraints

- For aperiodic stimuli, designers make assumptions about probability of occurrence of stimuli.
- May mean that certain design strategies such as object-oriented design cannot be used because of the additional overhead involved

State machine modelling

- The effect of a stimulus in a real-time system may trigger a transition from one state to another.
- Finite state machines can be used for modelling real-time systems.
- However, FSM models lack structure. Even simple systems can have a complex model.
- The UML includes notations for defining state machine models

Microwave oven state machine



Real-time programming

- Hard-real time systems may have to programmed in assembly language to ensure that deadlines are met
- Languages such as C allow efficient programs to be written but do not have constructs to support concurrency or shared resource management
- Ada as a language designed to support real-time systems design so includes a general purpose concurrency mechanism

Java as a real-time language

- Java supports lightweight concurrency (threads and synchonized methods) and can be used for some soft real-time systems
- Java 2.0 is not suitable for hard RT programming or programming where precise control of timing is required
    - Not possible to specify thread execution time
    - Uncontrollable garbage collection
    - Not possible to discover queue sizes for shared resources
    - Variable virtual machine implementation
    - Not possible to do space or timing analysis

Real Time Executives

- Real-time executives are specialised operating systems which manage processes in the RTS
- Responsible for process management and resource (processor and memory) allocation
- Storage management, fault management.
- Components depend on complexity of system

Executive components

- Real-time clock
    - Provides information for process scheduling.
- Interrupt handler

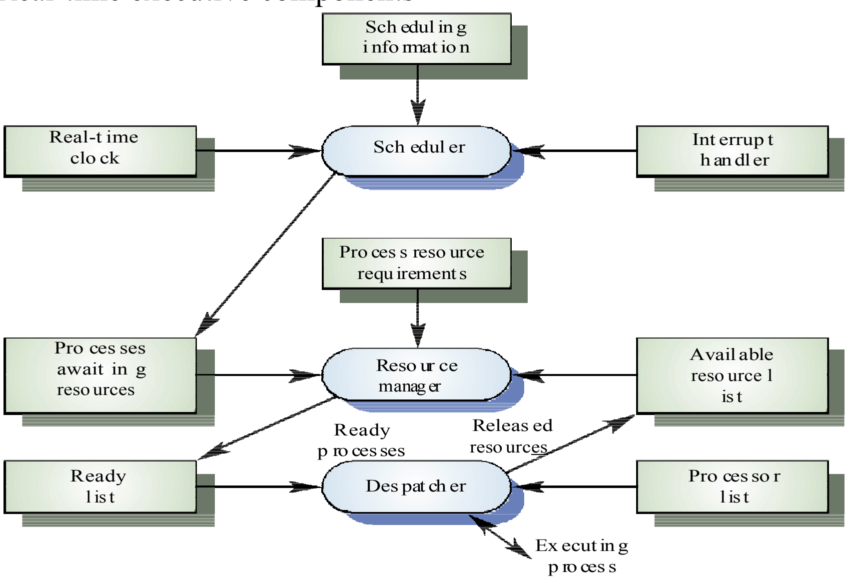
- Manages aperiodic requests for service.

- Scheduler
  - Chooses the next process to be run.
- Resource manager
  - Allocates memory and processor resources.
- Dispatchers
  - Starts process execution.

Non-stop system components

- Configuration manager
  - Responsible for the dynamic reconfiguration of the system software and hardware. Hardware modules may be replaced and software upgraded without stopping the systems
- Fault manager
  - Responsible for detecting software and hardware faults and taking appropriate actions (e.g. switching to backup disks) to ensure that the system continues in operation

Real-time executive components



Process priority

- The processing of some types of stimuli must sometimes take priority
- Interrupt level priority. Highest priority which is allocated to processes requiring a very fast response
- Clock level priority. Allocated to periodic processes
- Within these, further levels of priority may be assigned

Interrupt servicing

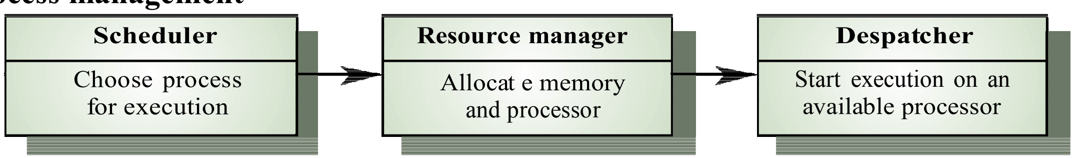
- Control is transferred automatically to a pre-determined memory location
- This location contains an instruction to jump to an interrupt service routine
- Further interrupts are disabled, the interrupt serviced and control returned to the interrupted process

- Interrupt service routines MUST be short, simple and fast

Periodic process servicing
- In most real-time systems, there will be several classes of periodic process, each with different periods (the time between executions), execution times and deadlines (the time by which processing must be completed)
- The real-time clock ticks periodically and each tick causes an interrupt which schedules the process manager for periodic processes
- The process manager selects a process which is ready for execution

Process management
- Concerned with managing the set of concurrent processes
- Periodic processes are executed at pre-specified time intervals
- The executive uses the real-time clock to determine when to execute a process
- Process period - time between executions
- Process deadline - the time by which processing must be complete

RTE process management



Process switching
- The scheduler chooses the next process to be executed by the processor. This depends on a scheduling strategy which may take the process priority into account
- The resource manager allocates memory and a processor for the process to be executed
- The despatcher takes the process from ready list, loads it onto a processor and starts execution

Scheduling strategies
- Non pre-emptive scheduling
  - Once a process has been scheduled for execution, it runs to completion or until it is blocked for some reason (e.g. waiting for I/O)
- Pre-emptive scheduling
  - The execution of an executing processes may be stopped if a higher priority process requires service
- Scheduling algorithms
  - Round-robin
  - Shortest deadline first

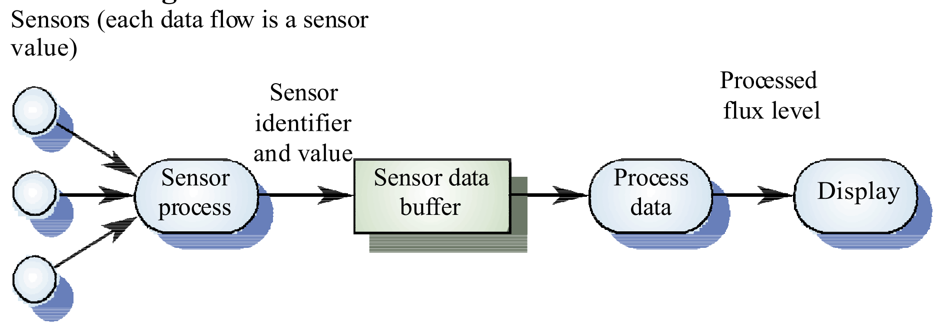
Data Acquisition System
- Collect data from sensors for subsequent processing and analysis.
- Data collection processes and processing processes may have different periods and deadlines.

- Data collection may be faster than processing
  e.g. collecting information about an explosion, scientific experiments
- Circular or ring buffers are a mechanism for smoothing speed differences.

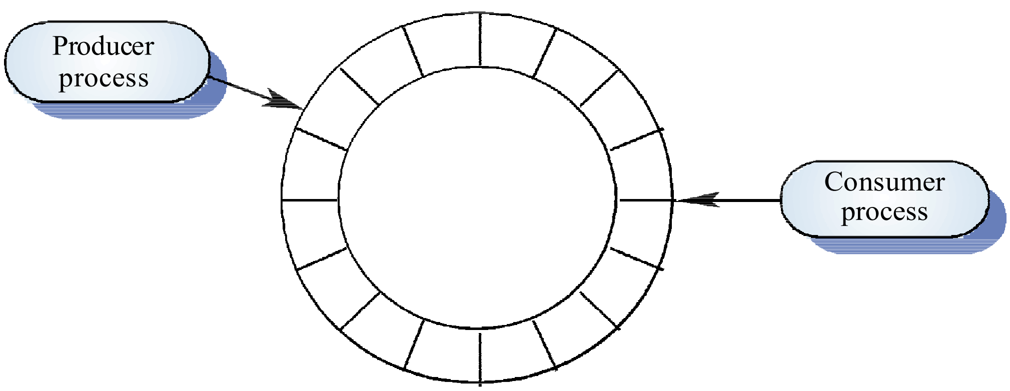
Reactor data collection

- A system collects data from a set of sensors monitoring the neutron flux from a nuclear reactor.
- Flux data is placed in a ring buffer for later processing.
- The ring buffer is itself implemented as a concurrent process so that the collection and processing processes may be synchronized.

Reactor flux monitoring



A ring buffer



Mutual exclusion

- Producer processes collect data and add it to the buffer. Consumer processes take data from the buffer and make elements available.
- Producer and consumer processes must be mutually excluded from accessing the same element.
- The buffer must stop producer processes adding information to a full buffer and consumer processes trying to take information from an empty buffer.

Java implementation of a ring buffer

```
class CircularBuffer
{
        int bufsize ;
        SensorRecord [] store ;
```

```
    int numberOfEntries = 0 ;
    int front = 0, back = 0 ;

    CircularBuffer (int n) {
        bufsize = n ;
        store = new SensorRecord [bufsize] ;
    } // CircularBuffer

    synchronized void put (SensorRecord rec ) throws InterruptedException
    {
        if ( numberOfEntries == bufsize)
            wait () ;
        store [back] = new SensorRecord (rec.sensorId, rec.sensorVal) ;
        back = back + 1 ;
        if (back == bufsize)
            back = 0 ;
        numberOfEntries = numberOfEntries + 1 ;
        notify () ;
    } // put

    synchronized SensorRecord get () throws InterruptedException
    {
        SensorRecord result = new SensorRecord (-1, -1) ;
        if (numberOfEntries == 0)
                wait () ;
        result = store [front] ;
        front = front + 1 ;
        if (front == bufsize)
            front = 0 ;
        numberOfEntries = numberOfEntries - 1 ;
        notify () ;
        return result ;
    } // get
} // CircularBuffer
```

Monitoring and Control System

- Important class of real-time systems
- Continuously check sensors and take actions depending on sensor values
- Monitoring systems examine sensors and report their results
- Control systems take sensor values and control hardware actuators
- Burglar alarm system e.g
- A system is required to monitor sensors on doors and windows to detect the presence of intruders in a building
- When a sensor indicates a break-in, the system switches on lights around the area and calls police automatically

- The system should include provision for operation without a mains power supply

Burglar alarm system

- Sensors
  - Movement detectors, window sensors, door sensors.
  - 50 window sensors, 30 door sensors and 200 movement detectors
  - Voltage drop sensor
- Actions
  - When an intruder is detected, police are called automatically.
  - Lights are switched on in rooms with active sensors.
  - An audible alarm is switched on.
  - The system switches automatically to backup power when a voltage drop is detected.

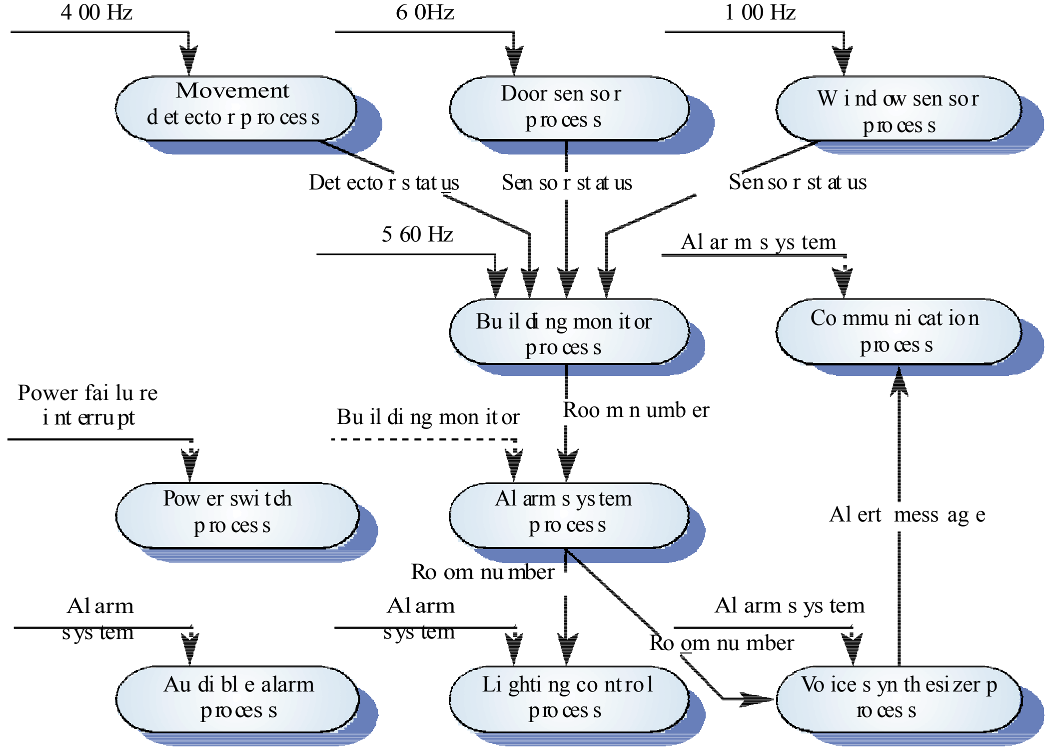
The R-T system design process

- Identify stimuli and associated responses
- Define the timing constraints associated with each stimulus and response
- Allocate system functions to concurrent processes
- Design algorithms for stimulus processing and response generation
- Design a scheduling system which ensures that processes will always be scheduled to meet their deadlines
- Stimuli to be processed
- Power failure
  - Generated by a circuit monitor. When received, the system must switch to backup power within 50 ms
- Intruder alarm
  - Stimulus generated by system sensors. Response is to call the police, switch on building lights and the audible alarm

Timing requirements

Stimulus/Response	Timing requirements
Power fail interrupt	The switch to backup power must be completed within a deadline of 50 ms.
Door alarm	Each door alarm should be polled twice per second.
Window alarm	Each window alarm should be polled twice per second.
Movement detector	Each movement detector should be polled twice per second.
Audible alarm	The audible alarm should be switched on within 1/2 second of an alarm being raised by a sensor.
Lights switch	The lights should be switched on within 1/2 second of an alarm being raised by a sensor.
Communications	The call to the police should be started within 2 seconds of an alarm being raised by a sensor.
Voice synthesiser	A synthesised message should be available within 4 seconds of an alarm being raised by a sensor.

Process architecture

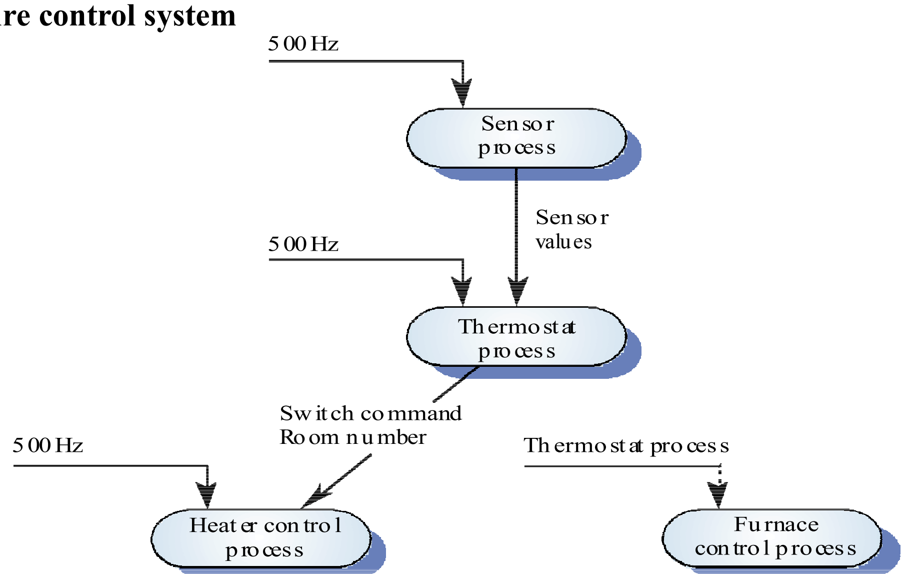


Building monitor process

```
class BuildingMonitor extends Thread {

        BuildingSensor win, door, move ;

        Siren   siren = new Siren () ;
        Lights  lights = new Lights () ;
        Synthesizer synthesizer = new Synthesizer () ;
        DoorSensors doors = new DoorSensors (30) ; WindowSensors
                                windows = new WindowSensors (50) ;
        MovementSensors movements = new MovementSensors (200) ;
        PowerMonitor pm = new PowerMonitor () ;

        BuildingMonitor()
        {
                // initialise all the sensors and start the processes
                siren.start () ; lights.start () ;
                synthesizer.start () ; windows.start () ;
                doors.start () ; movements.start () ; pm.start () ;
        }
```

```
    public void run ()
    {
        int room = 0 ;
        while (true)
        {
            // poll the movement sensors at least twice per second (400 Hz)
            move = movements.getVal () ;
            // poll the window sensors at least twice/second (100 Hz)
            win = windows.getVal () ;
            // poll the door sensors at least twice per second (60 Hz)
            door = doors.getVal () ;
            if (move.sensorVal == 1 | door.sensorVal == 1 | win.sensorVal == 1)
                {
                    // a sensor has indicated an intruder
                    if (move.sensorVal == 1)     room = move.room ;
                    if (door.sensorVal == 1)     room = door.room ;
                    if (win.sensorVal == 1 )          room = win.room ;

                    lights.on (room) ; siren.on () ; synthesizer.on (room) ;
                    break ;
                }
        }
        lights.shutdown () ; siren.shutdown () ; synthesizer.shutdown () ;
        windows.shutdown () ; doors.shutdown () ; movements.shutdown () ;

    } // run
} //BuildingMonitor
```

A temperature control system



Control systems