Architectural Design

Reference and Reading recommendations

-Chapter 6, Architectural design -Software Engineering, I. Sommerville

-Chapter 6, System Design: Decomposing the System, OO Software Engineering Using UML, Patterns and Java, B.Bruegge and A.H. Dutoit

-Software Engineering: A Practitioner's Approach, 6/e R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

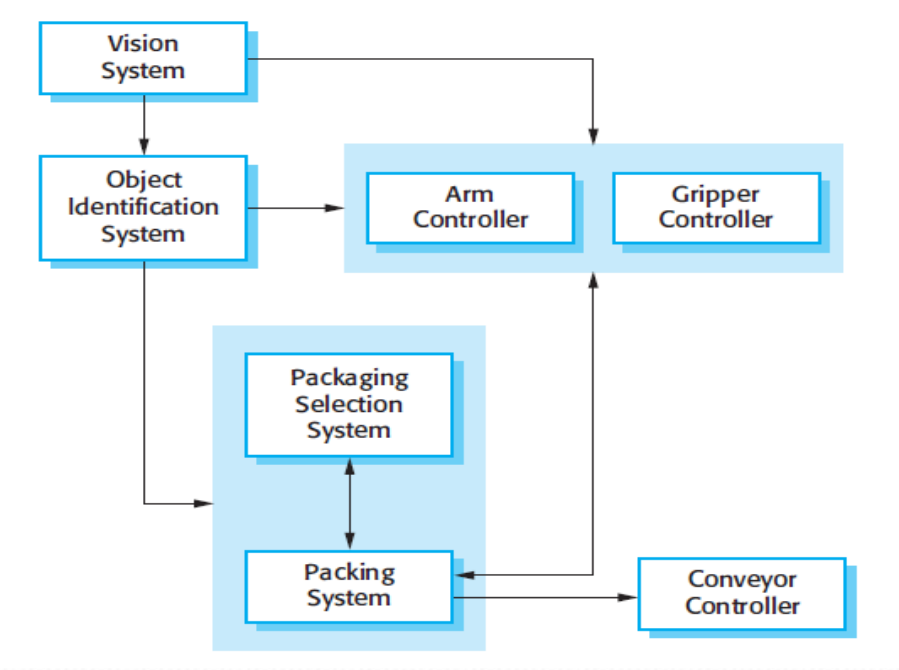
What is Architectural design

- Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system
- The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components – or software architecture.

Architectural abstraction levels

- **Architecture in the small** is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.
- **Architecture in the large** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

Example- The architecture of a packing robot control system



- Using block diagrams. Each block is a component. Boxes within boxes indicate that the component has been decomposed into sub-components
- Arrows: data and/or control signals are passed from component to component in the direction of the arrows

(I. Sommerville, 2011)

Advantages of explicit architecture

- To enable everyone to better understand the system
- To allow people to work on individual pieces of the system in isolation
- To prepare for extension of the system
- To facilitate reuse and reusability

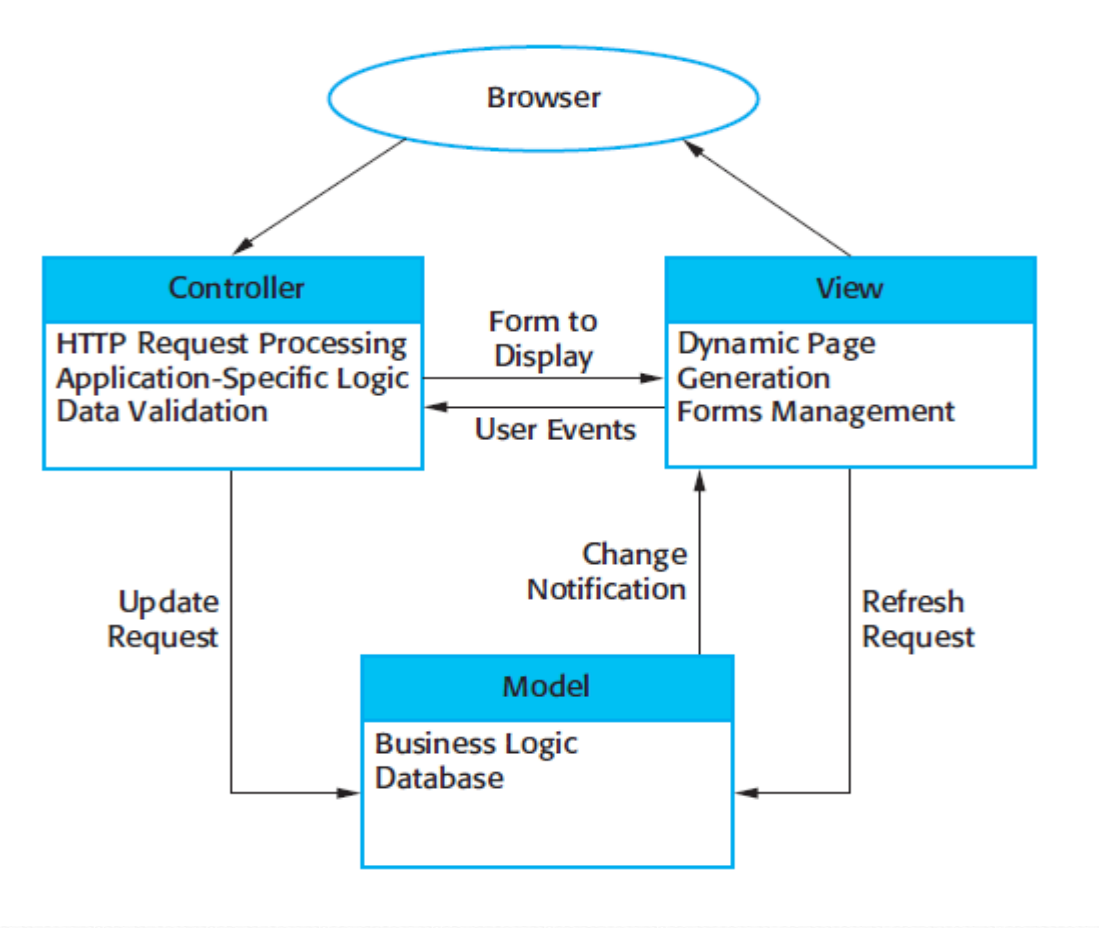
Architectural patterns/styles

- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments
- Different patterns:
  - MVC
  - Layered architecture
  - Repository/Data-Centered
  - Data Flow architecture (Pipe and Filter)
  - Client-Server architecture

The Model-View-Controller (MVC) pattern

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.
Example	A web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

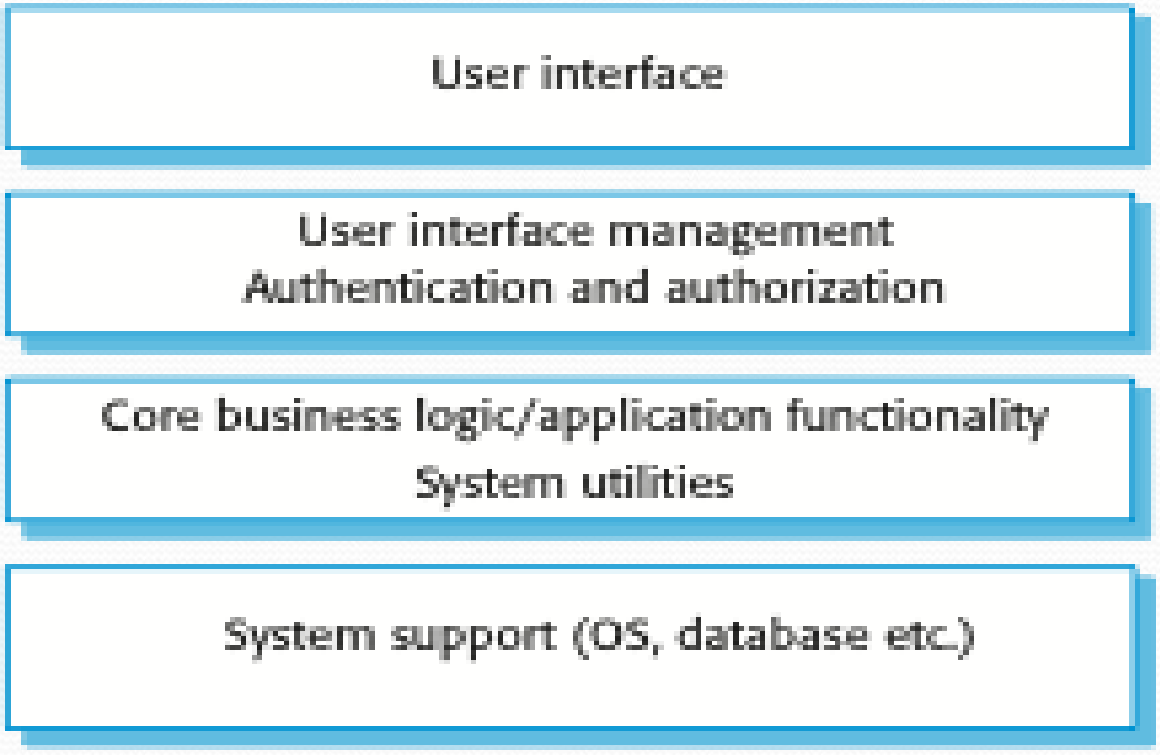
Web application architecture using the MVC pattern



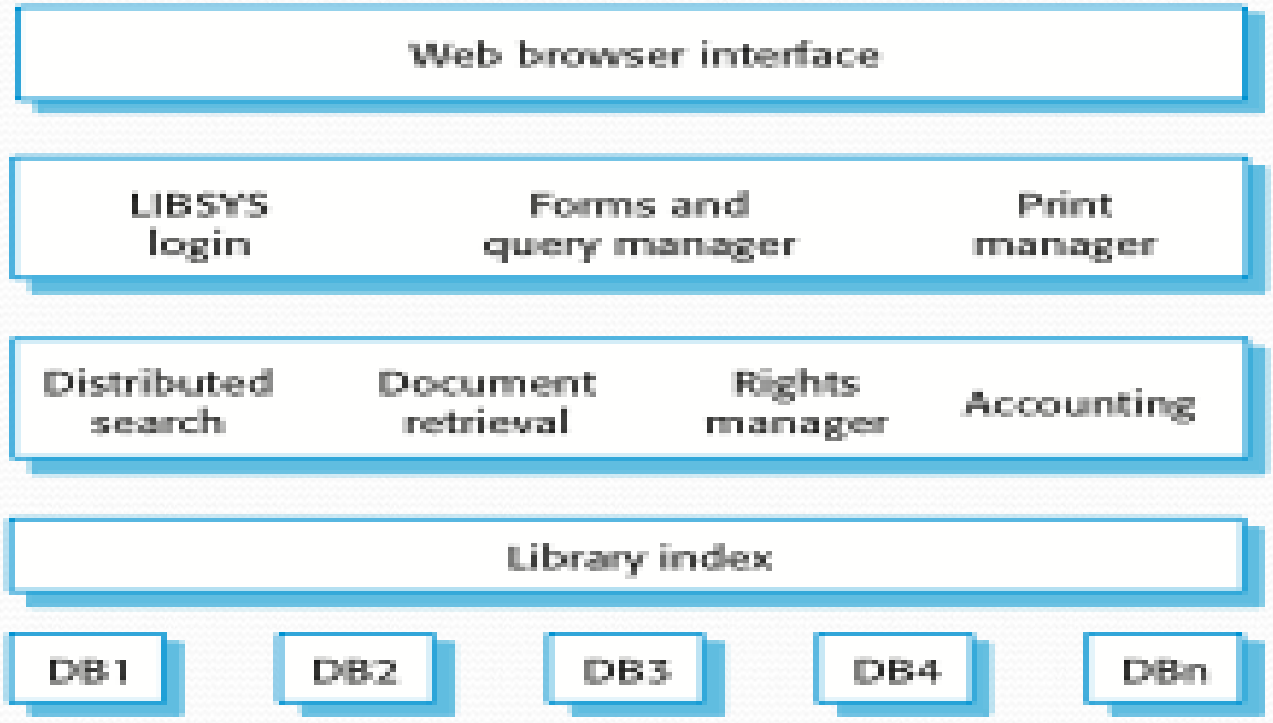
Layered architecture pattern

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system
Example	A system for sharing copyright documents held in different libraries
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g. authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer

A generic layered architecture



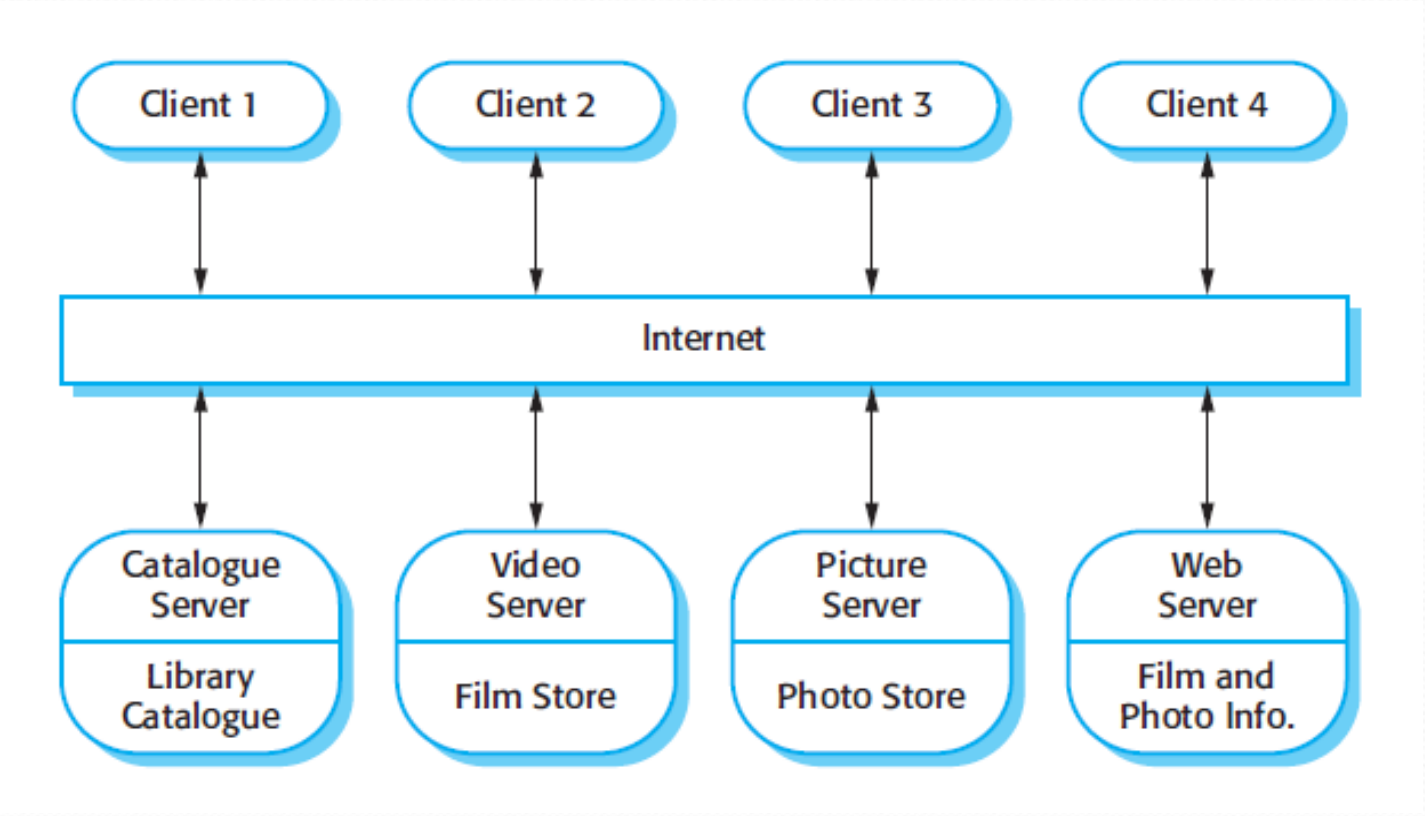
Example- The architecture of the LIBSYS system



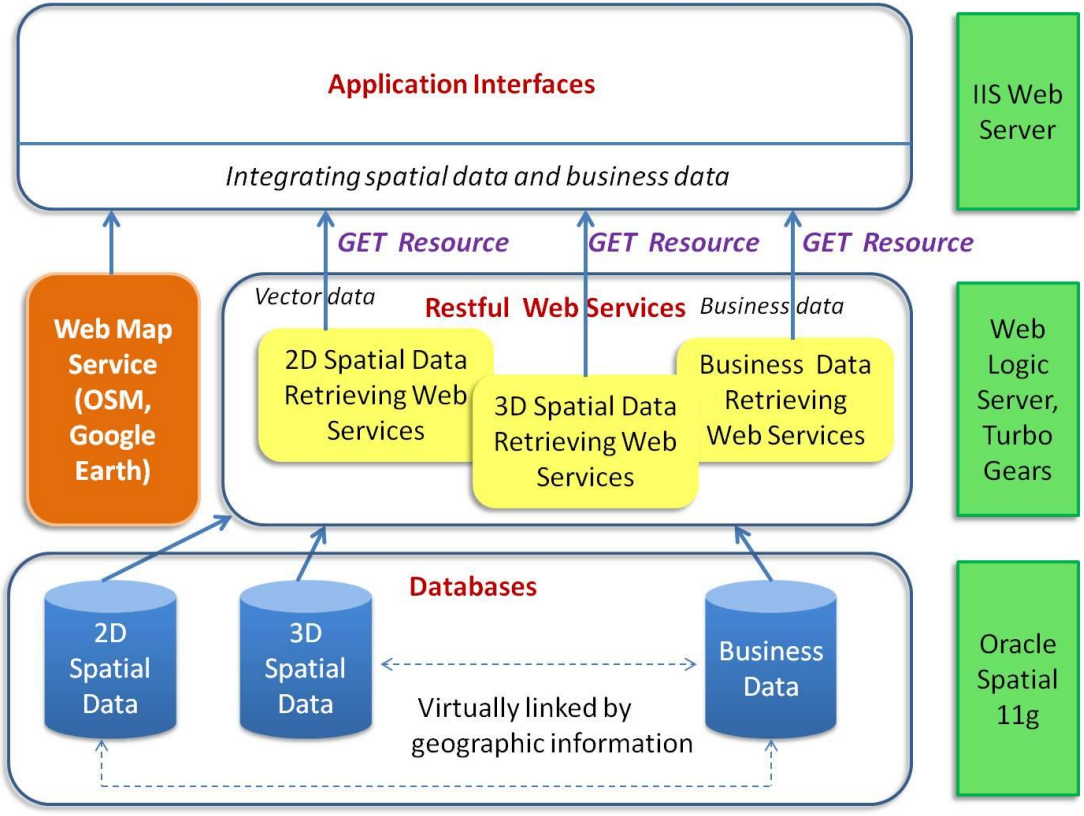
Client-Server pattern (2-tier)

Name	Client –server architecture
Description	In a client-server architecture, the functionality of the system is organised into services, with each services delivered from a separate server. Clients are users of these services and access servers to make use of them
Example	Film library application
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g. a printer service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organisations.

Example- A client–server architecture for a film library



Example- architecture of eCampus application – 3 tier

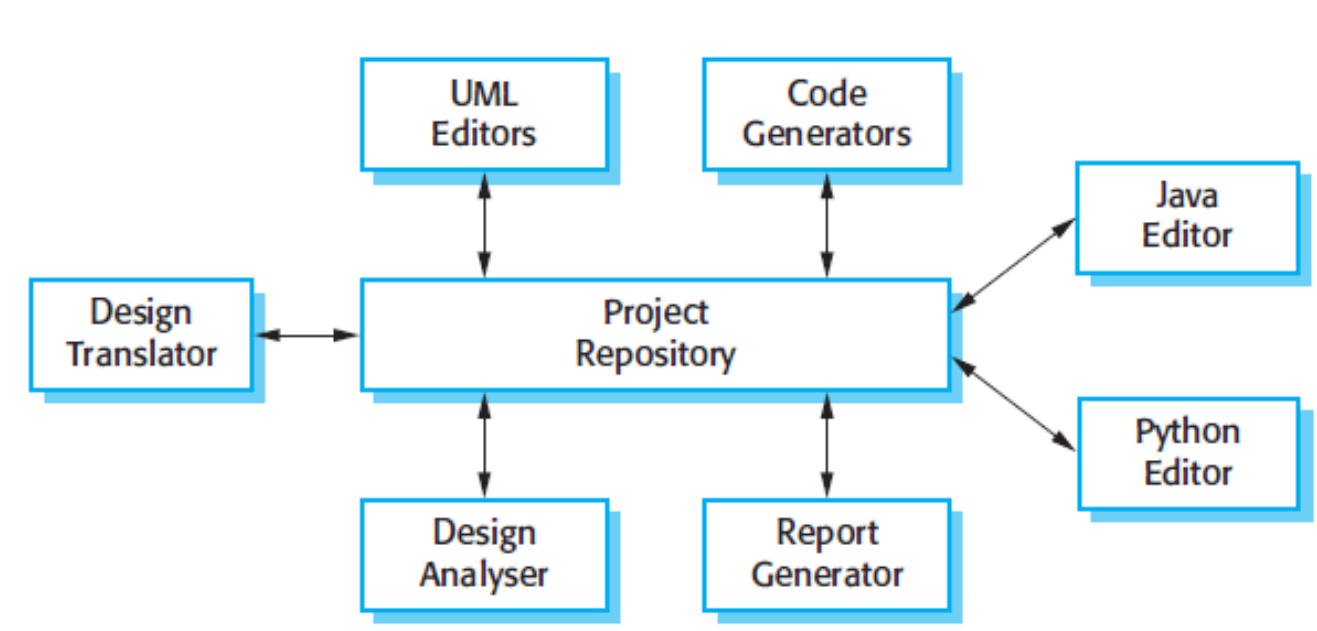


(Pham et al., 2013)

Repository pattern

- All data in the system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through repository
- Use this pattern when having a system in which large volumes of information are generated that has to be stored for a long time

Example- A repository architecture for an IDE (Integrated Development Environemtn)



Architectural views

- A system's architecture will often be expressed in terms of several different *views*
  - The logical breakdown into subsystems
  - The interfaces among the subsystems
  - The dynamics of the interaction among components at run time
  - The data that will be shared among the subsystems
  - The components that will exist at run time, and the machines or devices on which they will be located
- A system architecture can be described with block diagram, package diagram, component diagram (UML) and deployment diagram (UML)

Developing an architectural model

- Start by sketching an outline of the architecture
  - Based on the principal requirements and use cases
  - Determine the main components that will be needed
  - Choose among the various architectural patterns

Developing an architectural model

- Refine the architecture
  - Identify the main ways in which the components will interact and the interfaces between them
  - Decide how each piece of data and functionality will be distributed among the various components
  - Determine if you can re-use an existing style, if you can build a style
- Consider each use case and adjust the architecture to make it realizable
- Mature the architecture

Heuristics for grouping objects into subsytems

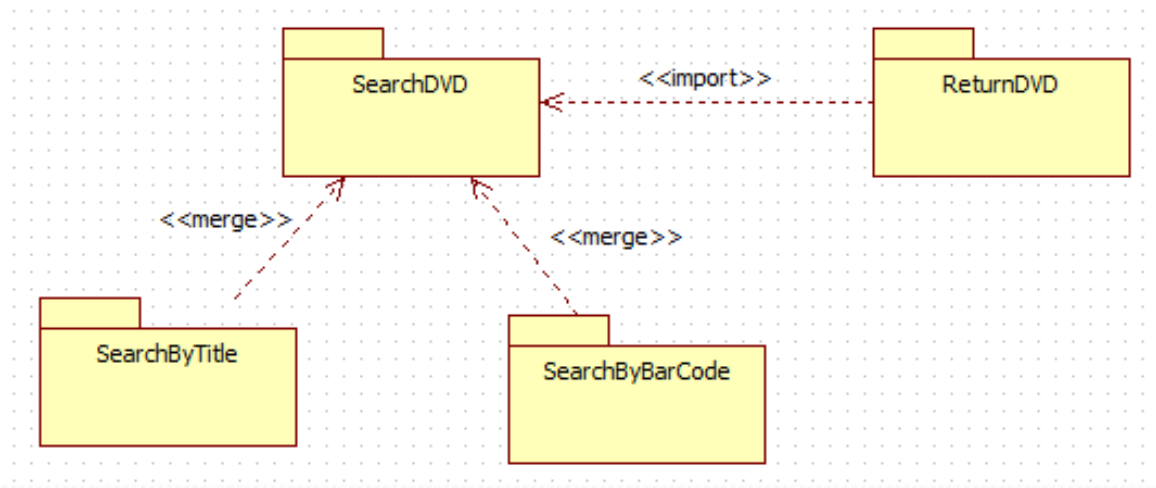
- Assign objects identified in one use case into the same subsystem
- Create a dedicated subsystem for objects used for moving data among subsystems
- Minimize the number of association crossing subsystem boundaries
- All objects in the same subsystem should be functionally related

Describing an architecture using UML

- Package diagrams:

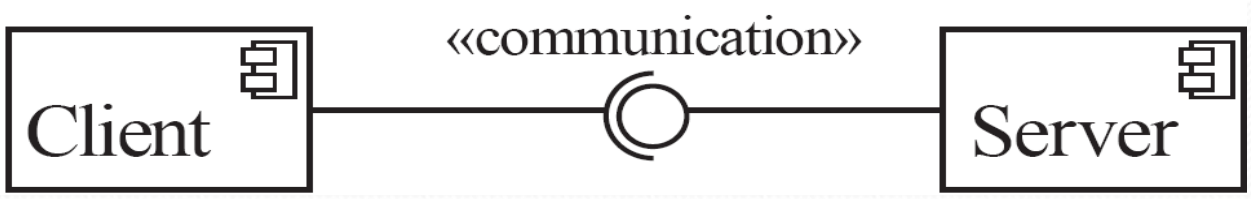
A package represents a physical partitioning of the system.

It can also represent layers of system architecture.



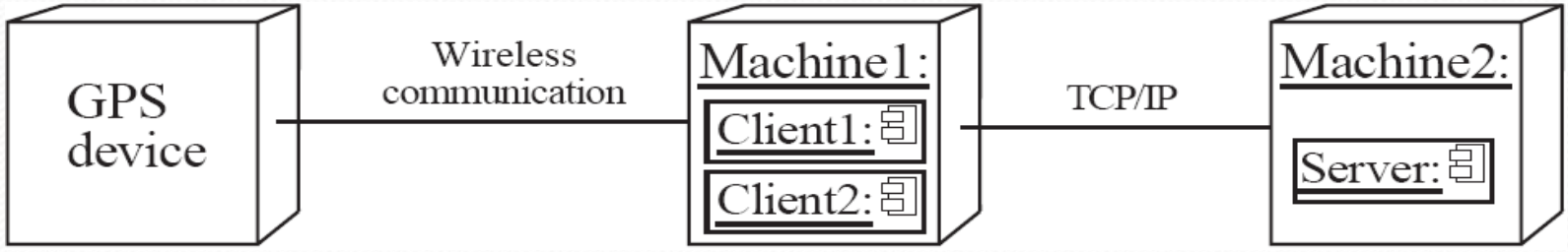
Component diagrams

Depict components and communication between them in term of request services and provide services via interfaces. Components are self-contained entities that provides services to other components or actors. E.g. Weather forecast service (server) provides services to a user (client site via a web browser).



Deployment diagrams

Depict the relationship among run-time components and nodes. A node is a physical device or an execution environment in which components are executed.



Use case based architecture design

- Identify main use cases
- Each main use case may correspond to a component (at the first time)
- Locate these components to suitable blocks/Layers/sub system in the architecture
- Identify the services of each component (interface-input-output)
- Review the granularity of components and redesign the architecture if necessary

Example of Architectural Design MyTrip system

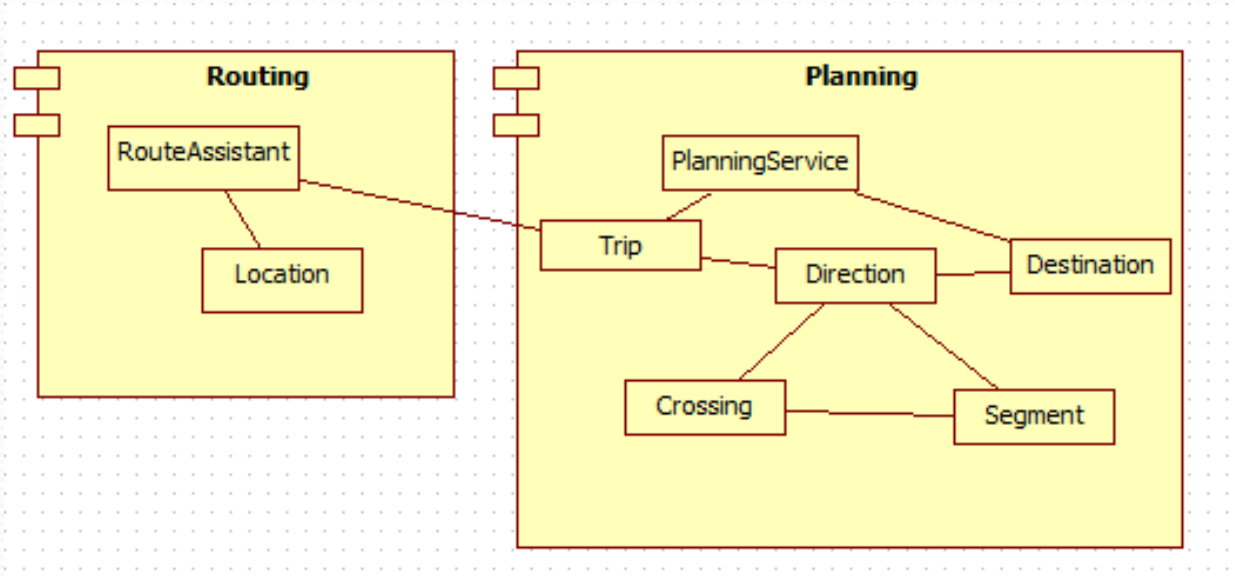
- Analysis models: use cases (and sequence diagram)
- Use case PlanTrip

Flow of events:

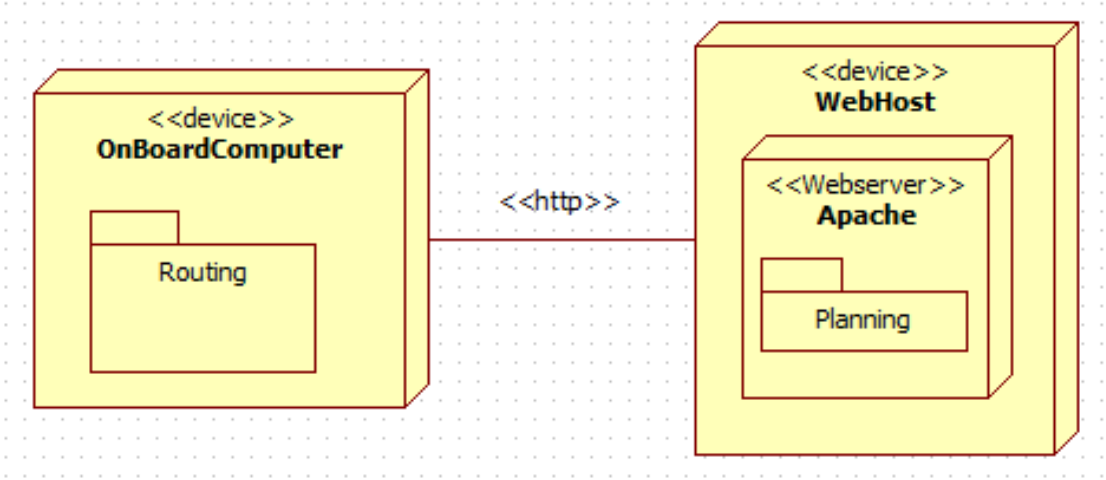
1. The Driver activates her computer and logs into the tripplanning web service
2. The Driver enters constraints for a trip as a sequence of destinations
3. Based on the database of maps, the planning service computes the shortest way of visiting the destinations. The result is a sequence of segments binding a series of crossings and a list of directions.
4. The Driver can revise the trip by adding or removing destinations
5. The Driver saves the planned trip by name in the planning service database for later retrieval

- Use case ExecuteTrip
- Flow of events
- 1. The Driver starts her car and logs into the onboard route assistant
- 2.Upon sucessful login, the Driver specifies the planning service and the name of the trip to be executed
- 3.The onboard route assistant obtains the list of destinations, directions, segments, and crossing from the planning service
- 4.Given the current position, the route assistant provides the driver with the next set of directions
- 5.The driver arrives to destination and shuts down the route of assistant.

Identifying objects and subsytems/components



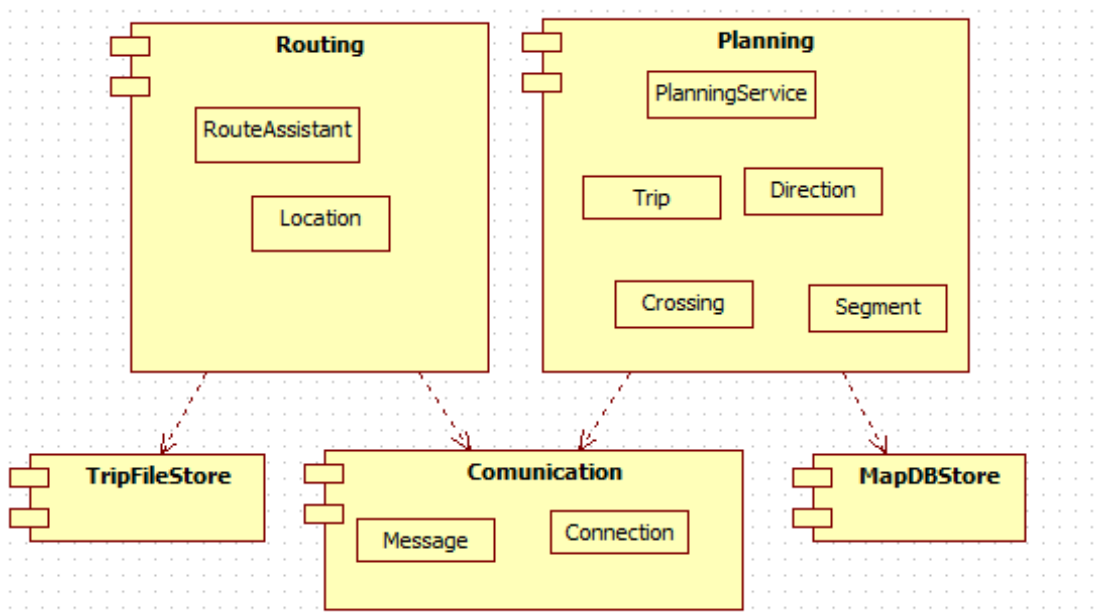
Mapping Subsystems to hardware/software platform



Revise design

- Because both RoutingSubsystem and PlanningSubsystem share the objects Trip, Destinatioin, Crossing, Segment and Direction. Object instances need to communicate via a wireless modem.
- Create a new subsystem Communication who is responsible for transporting objects from the Planning to RoutingSubsystem

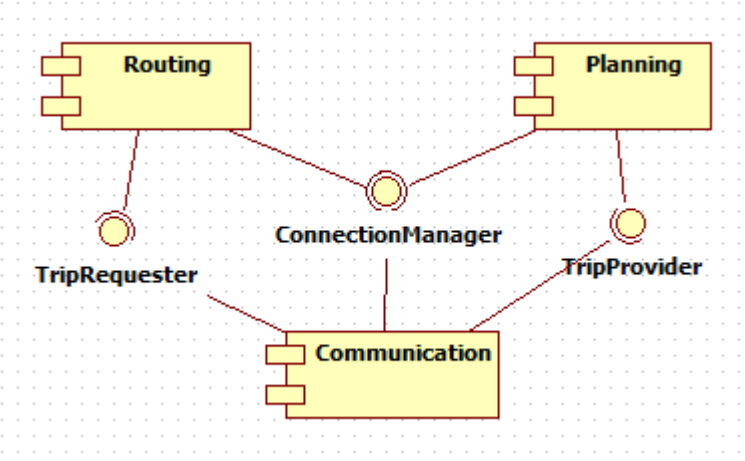
Re-design



- Connection: A connection represents an active link between the Planning subsystem and Routing sub systems. A connection object handles exceptional cases associated with loss of network services
- Message: A message represents a Trip and its related Destinations, Segments, Crossings, and Directions, encoded for transport

Refining the subsystem decomposition

- The Communication sub system provides three services for managing connections, uploading trips, and downloading trips
- Identifying 3 corresponding interfaces



Architecture and system characteristics

- Individual components of software architecture implement the functional system requirements

- The non-functional requirements depend on the system architecture- the way in which these components are organised and communicate

Architecture and system characteristics

- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localize 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.

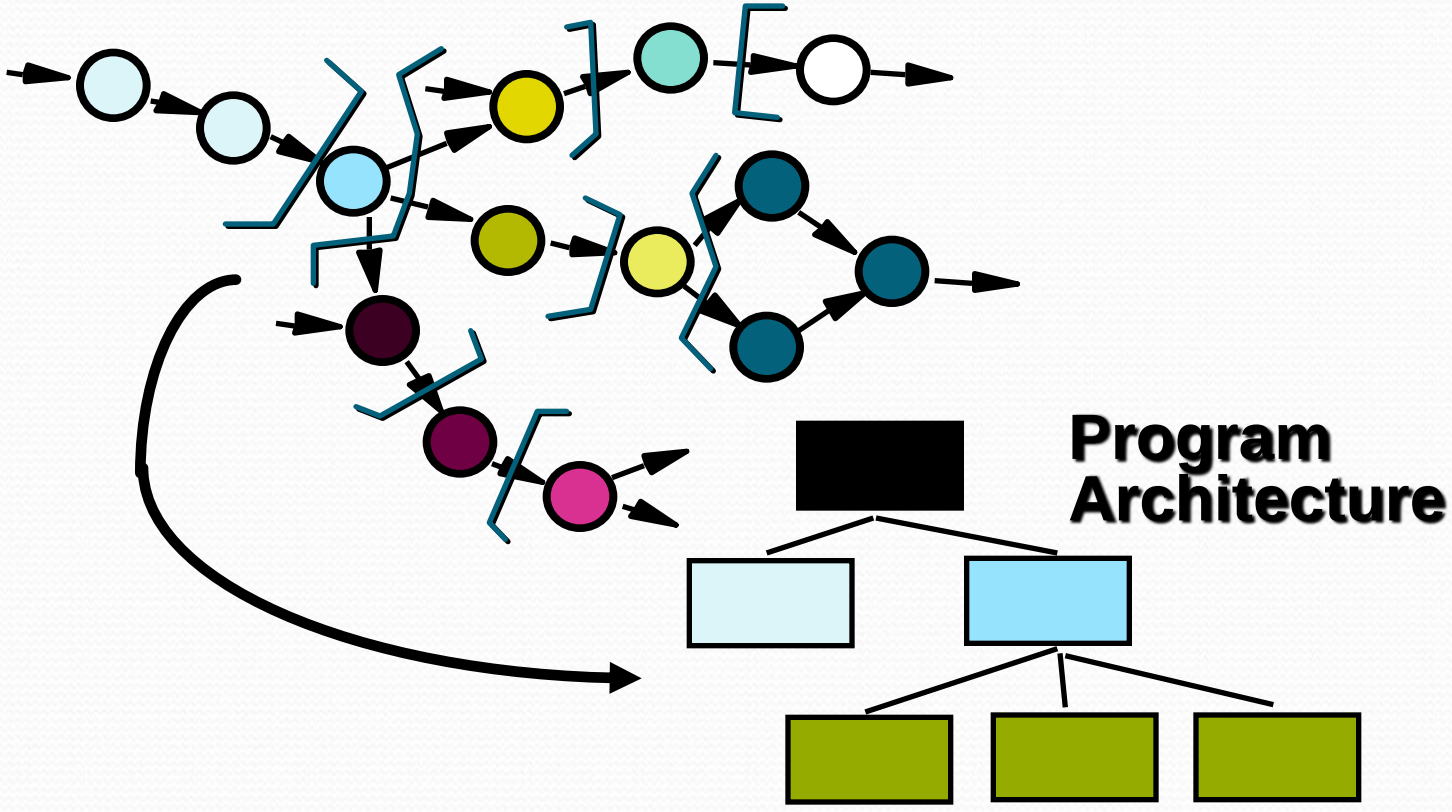
Architectural conflicts

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

Architectural design based on DFD

- Objective: to derive a software architecture that is partitioned
- Approach:
  - the DFD is mapped into a program architecture
  - provide a description to indicate the content of each module
- Notation: structure chart

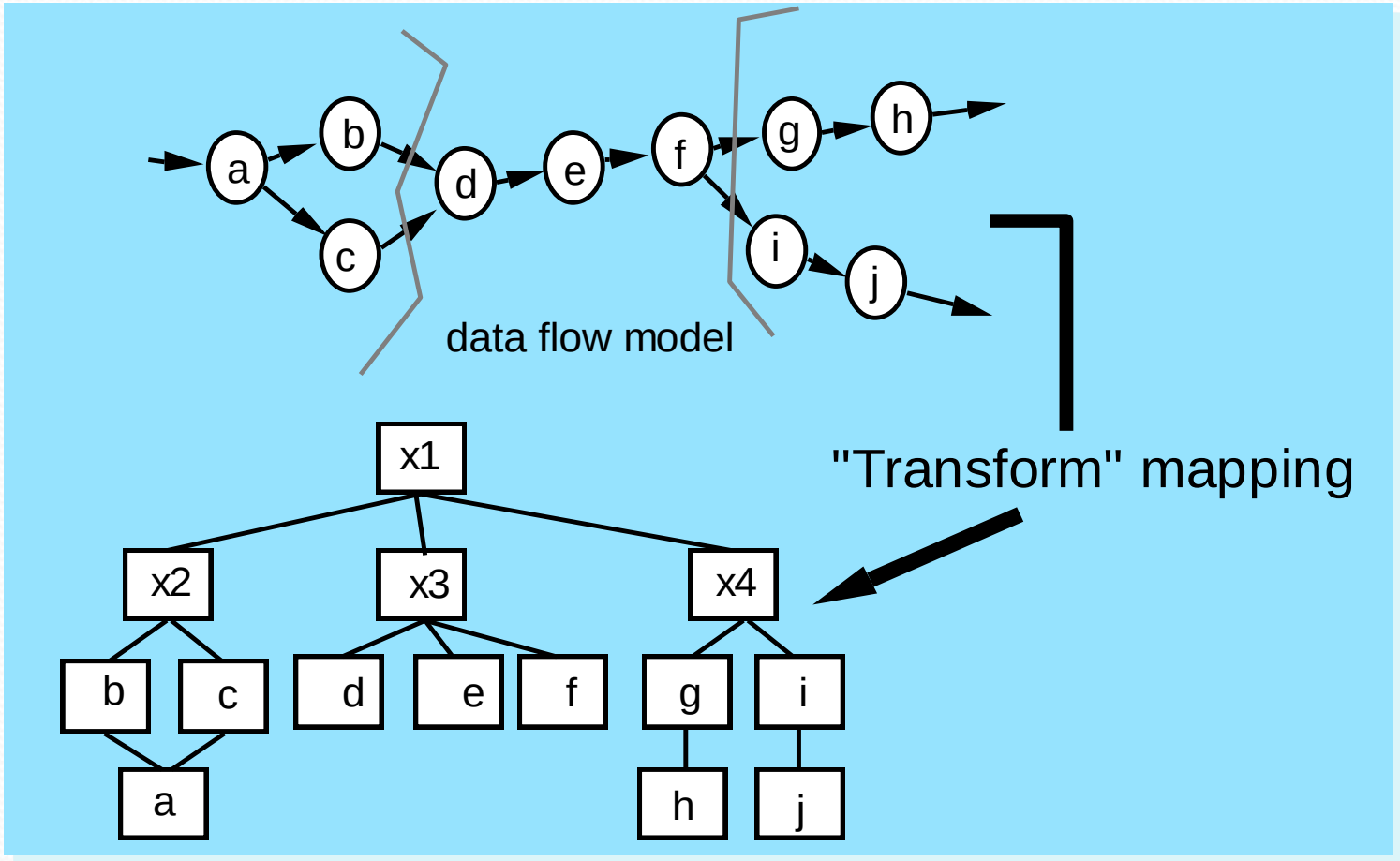
Deriving Program Architecture using DFD



General Mapping Approach

- Isolate incoming and outgoing flow boundaries
  - Incoming flow was described as a path in which information is converted from external to internal form;
  - Outgoing flow converts from internal to external form
- Working from the **boundary outward**, map DFD transforms into corresponding modules
- Add **control modules** as required
- Refine the resultant program structure using effective modularity concepts

Transform Mapping

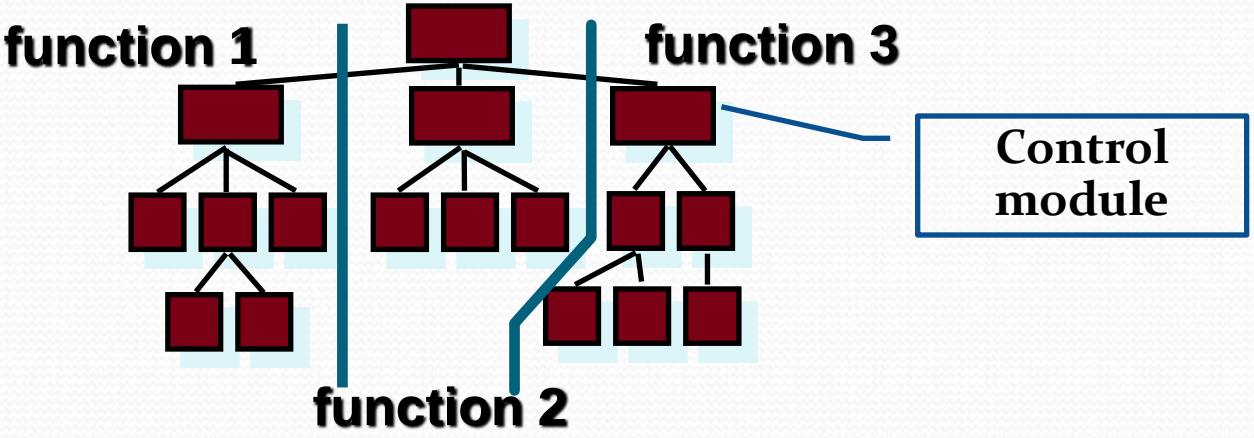


Refining the architecture design

- After the program structure has been developed and refined, the following tasks must be completed:
  - A processing narrative must be developed for each module
  - An interface description is provided for each module
  - Local and global data structures are defined
  - All design restrictions and limitations are noted
  - A set of design reviews are conducted

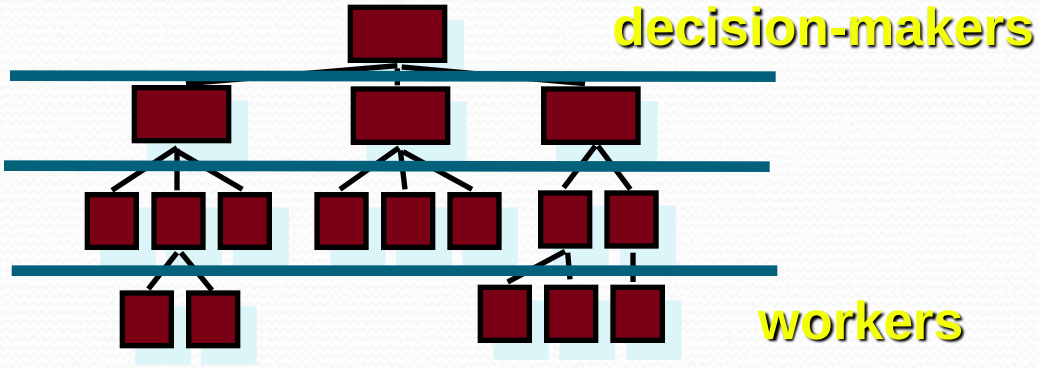
Horizontal Partitioning

- Define separate branches of the module hierarchy for each major function
- Use control modules to coordinate communication between functions
- The simplest approach to horizontal partitioning defines three partitions -input, data transformation (or processing) and output

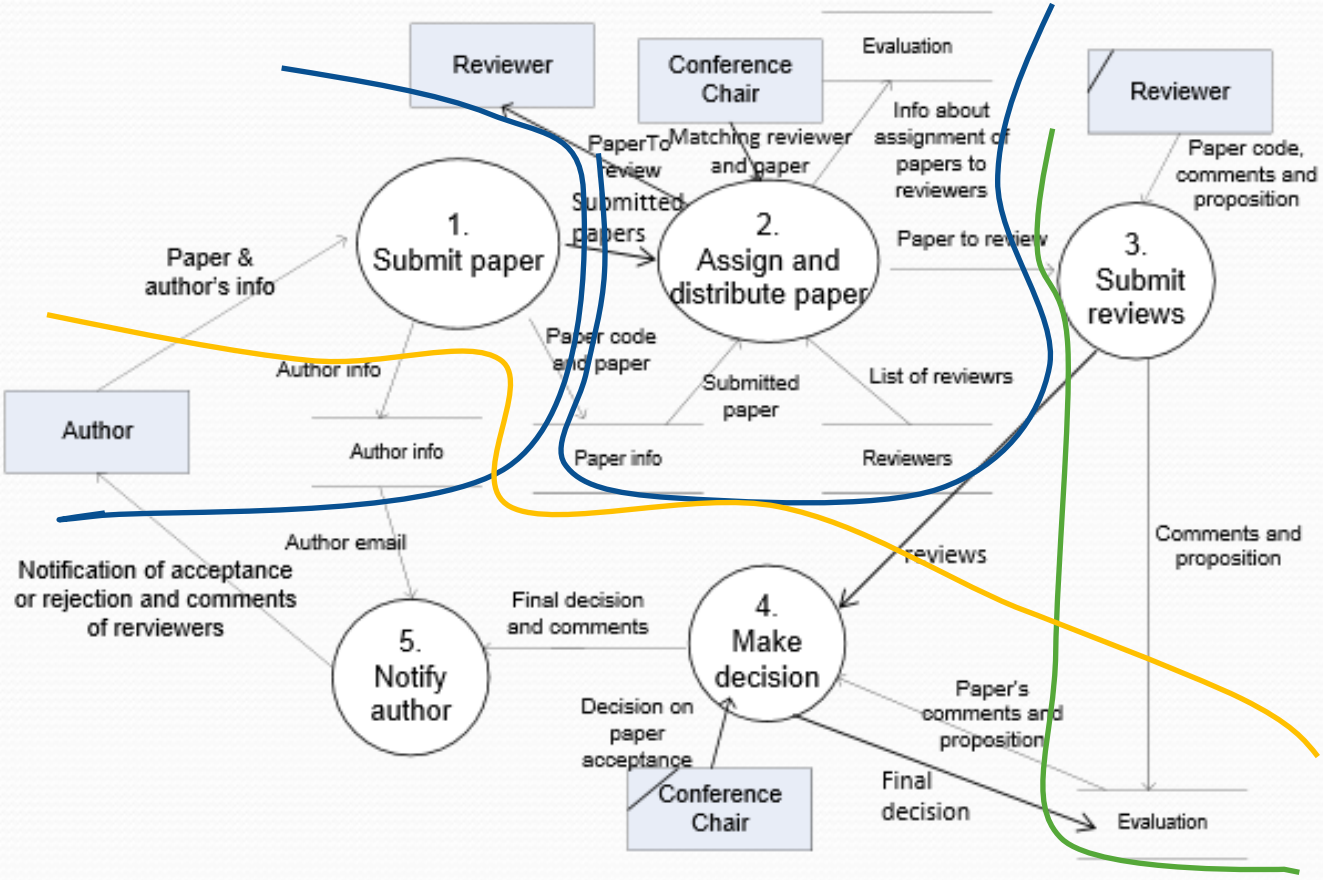


Vertical Partitioning: Factoring

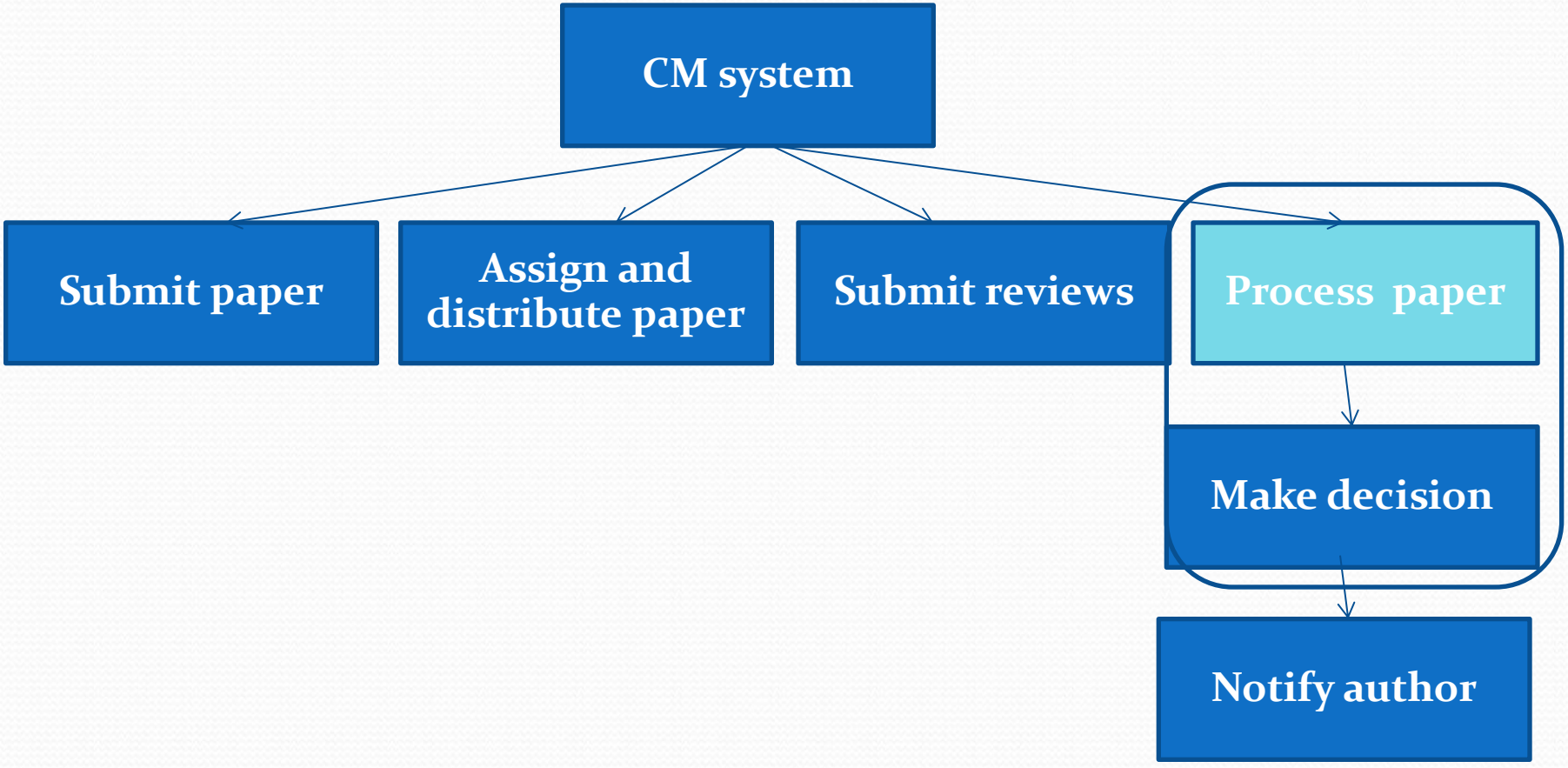
- Design so that decision making and work are stratified
- Decision making modules should reside at the top of the architecture



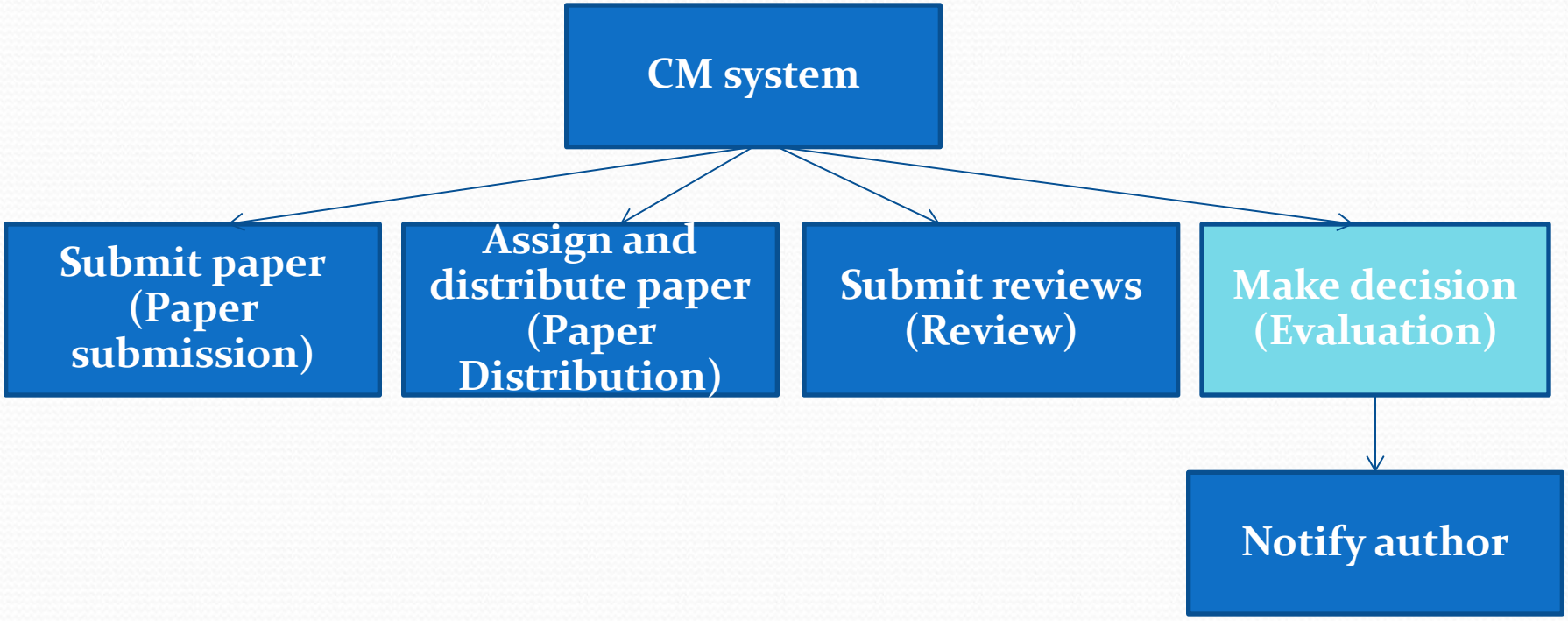
Example – Conference paper MS



First function hierarchy



Final function hierarchy

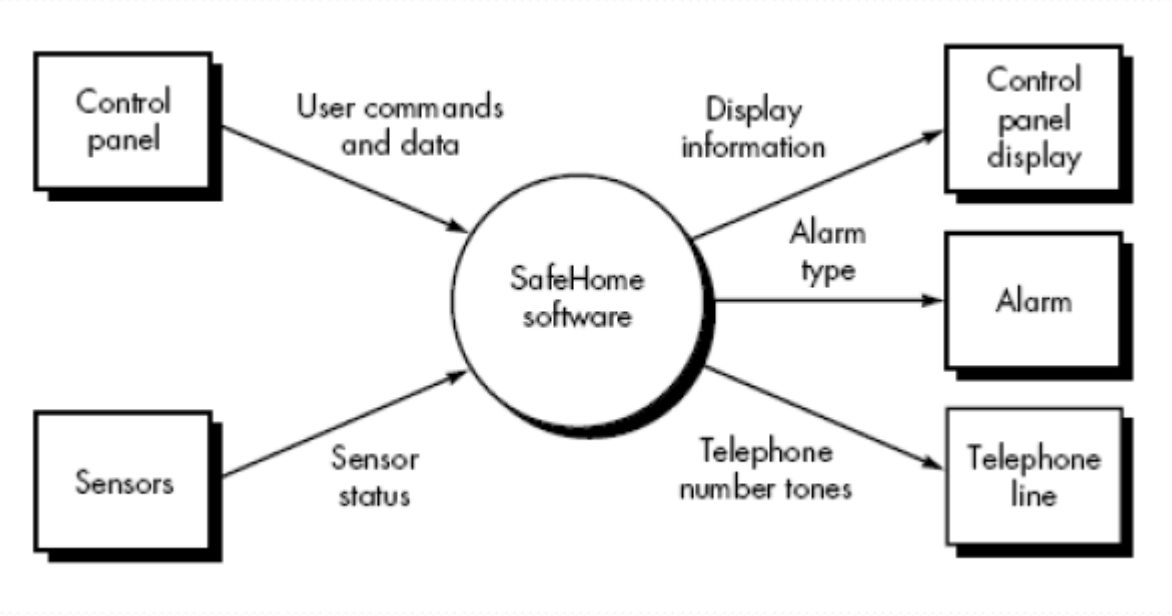


SafeHome example

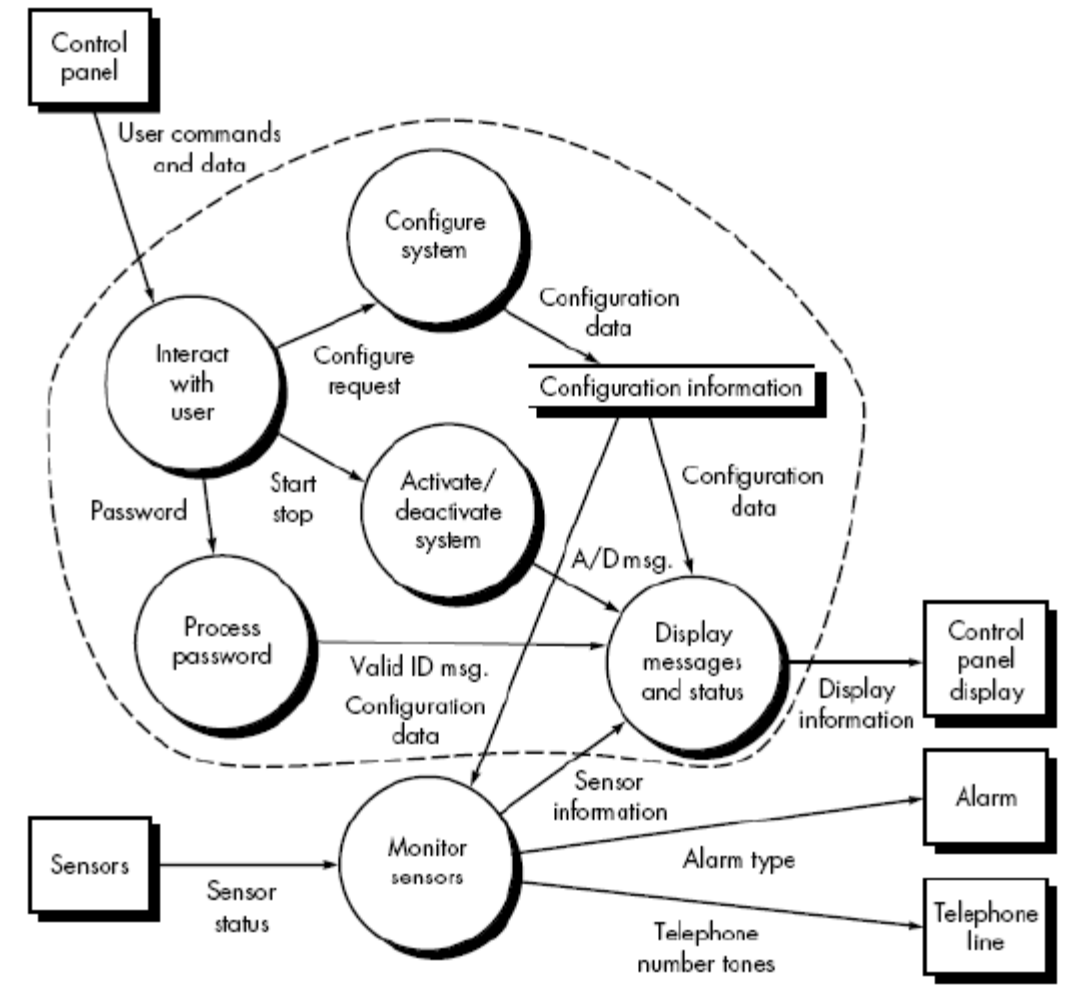
"**SafeHome** software enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through a keypad and function keys contained in the SafeHome control panel .

During installation, the SafeHome control panel is used to "program" and configure the system. Each sensor is assigned a number and type, a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs. When a sensor event is recognized, the software invokes an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, provides information about the location, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until telephone connection is obtained. All interaction with SafeHome is managed by a user-interaction subsystem that reads input provided through the keypad and function keys, displays prompting messages on the LCD display, displays system status information on the LCD display…"

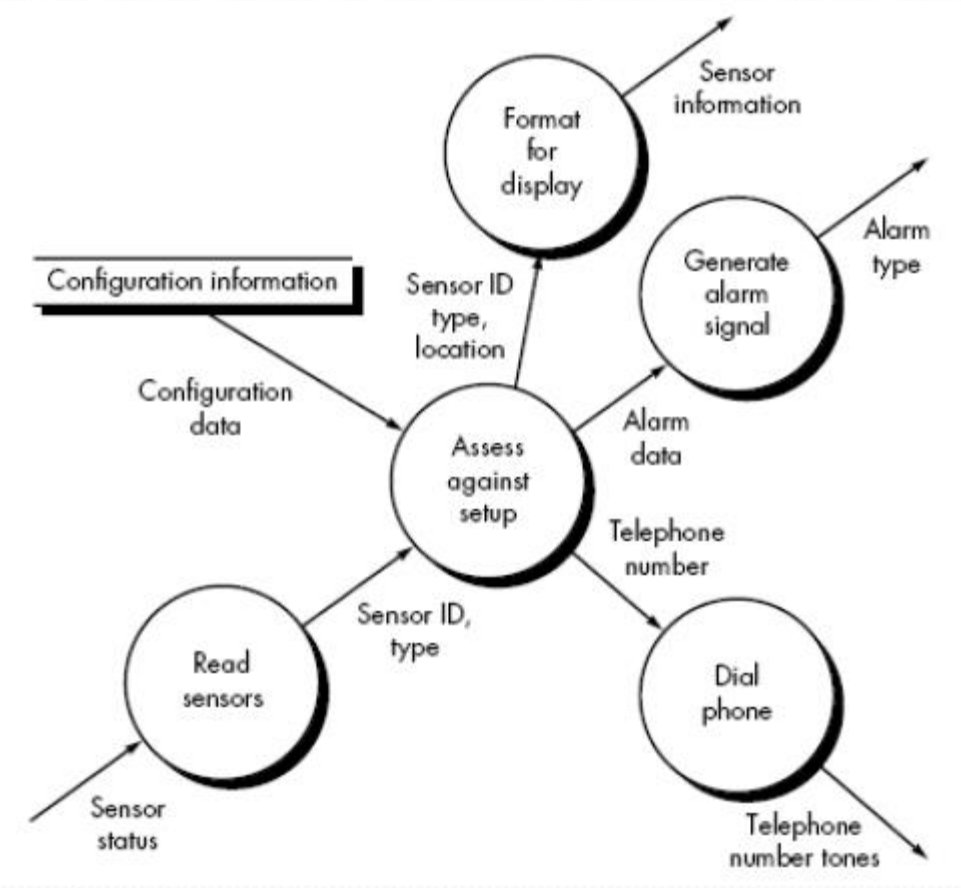
DFD- Level 0- Safe Home system



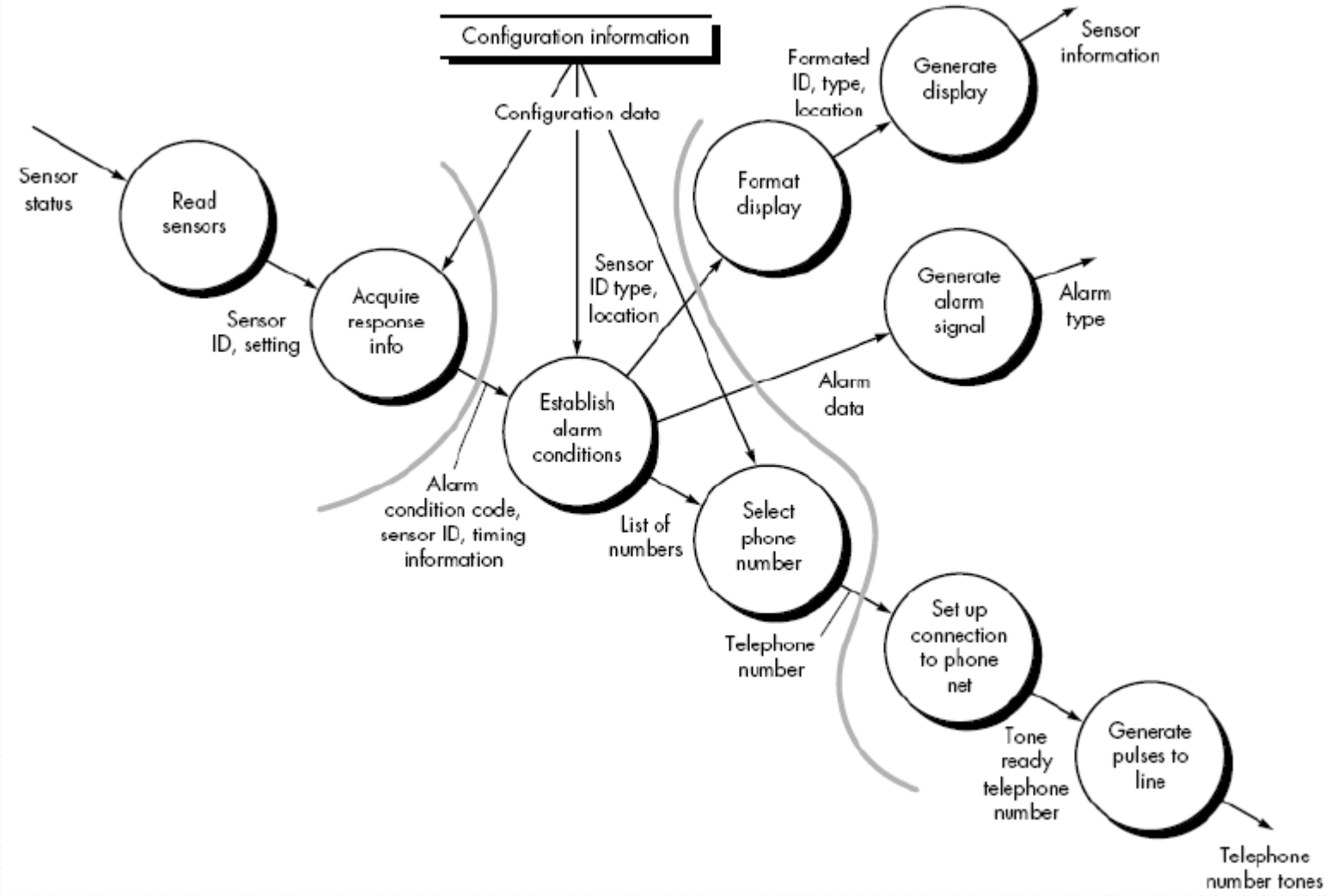
Level 1


Control panel
User commands and data
Configure system
Configuration data
Interact with user
Configure request
Configuration information
Start stop
Activate/ deactivate system
Configuration data
Password
A/D msg.
Process password
Valid ID msg.
Configuration data
Display messages and status
Control panel display
Display information
Sensor information
Alarm
Sensors
Monitor sensors
Alarm type
Sensor status
Telephone line
Telephone number tones

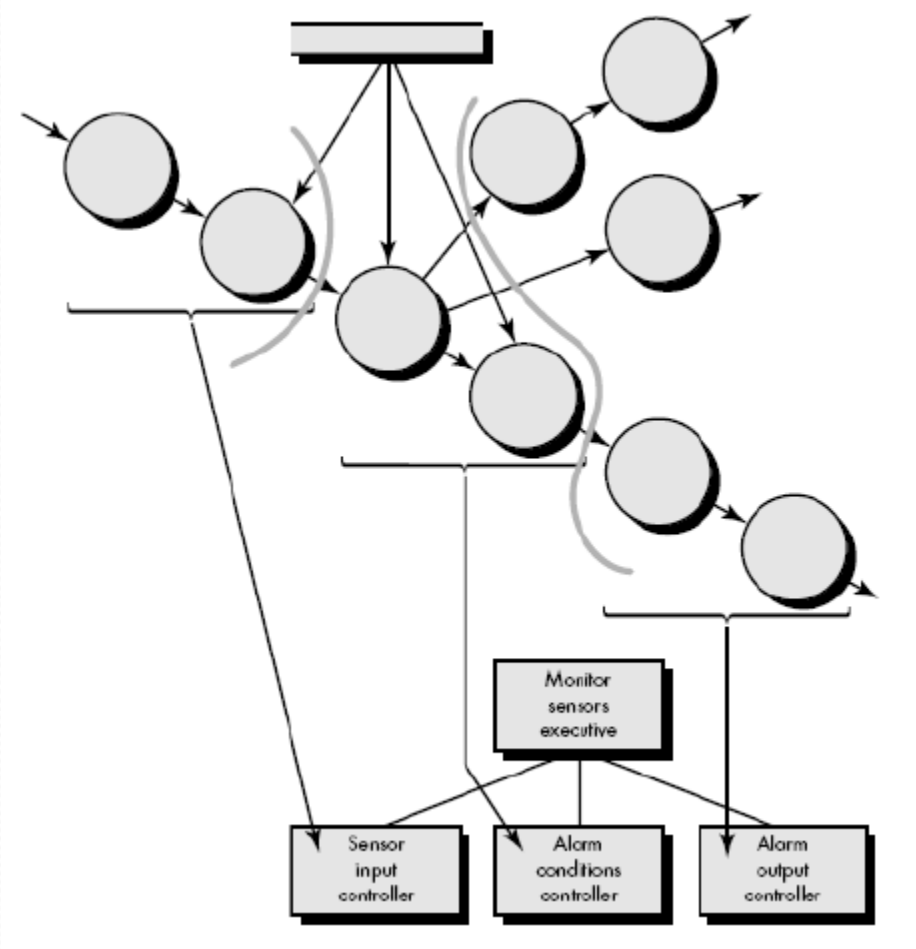
Level 2- Monitor sensors refinement



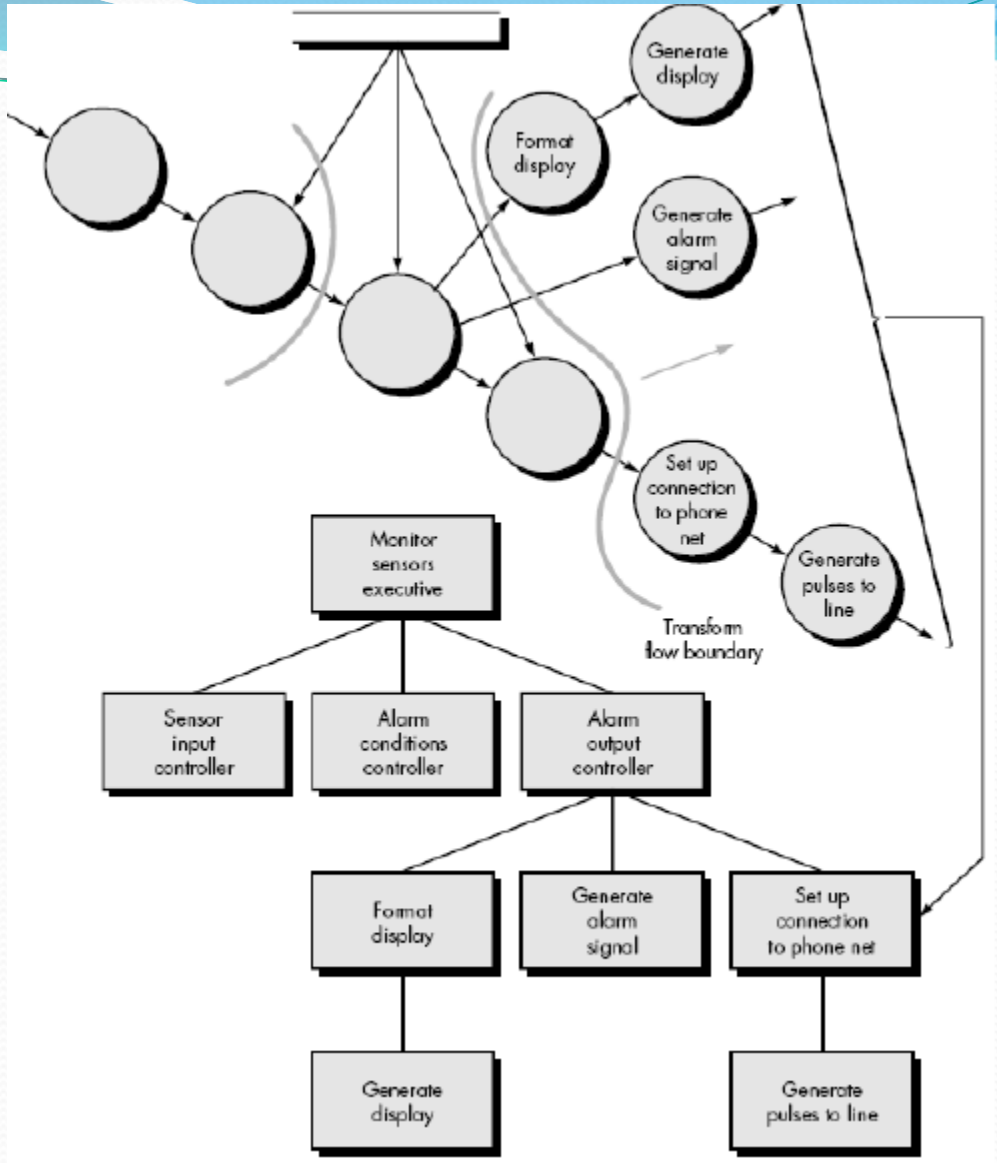
Level 3- Assess against set up



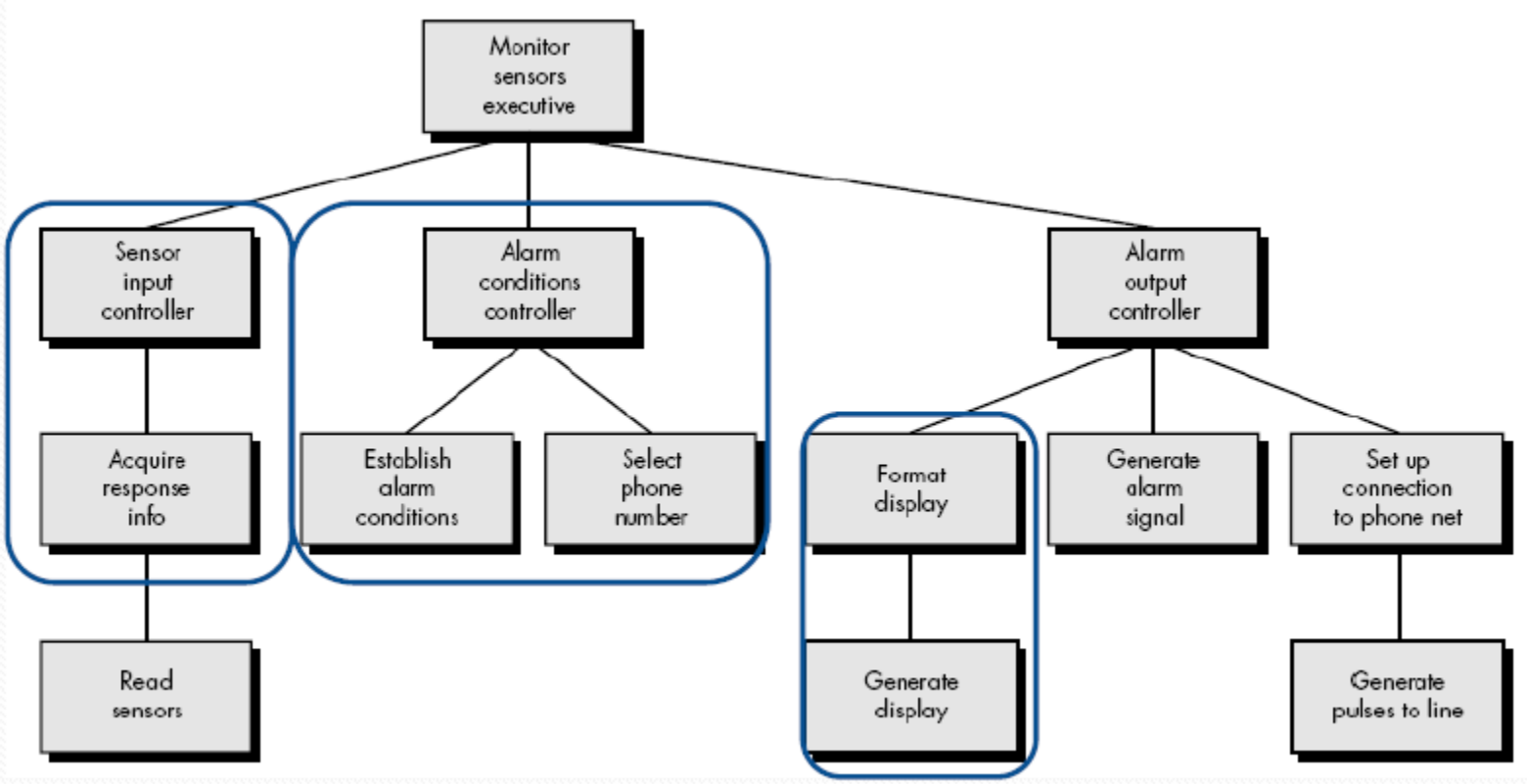
First level factoring



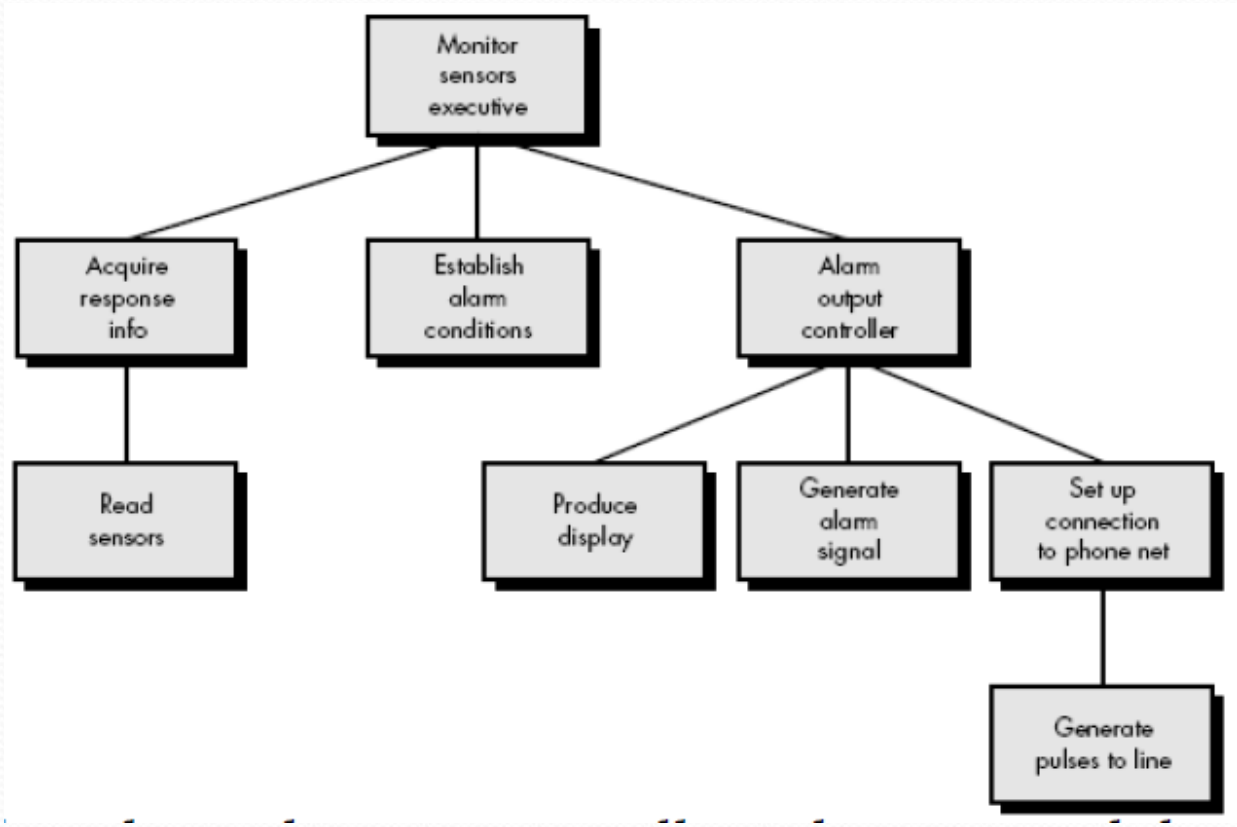
Second level



First program architecture



Final Program architecture



Key points

- A software architecture is a description of how a software system is organized. Properties of a system such as performance, security, and availability are influenced by the architecture used
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles /patterns to be used
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, when it may be used, and discuss advantages and disadvantages.