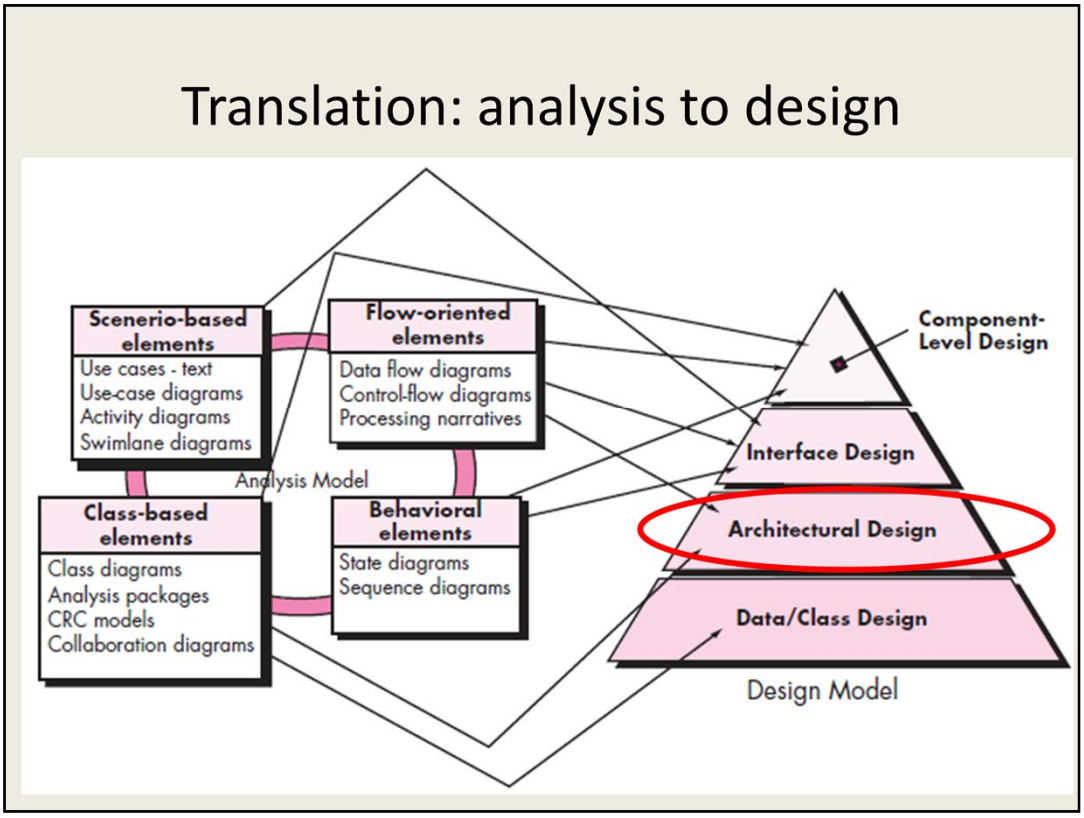
Architectural Design
(I)



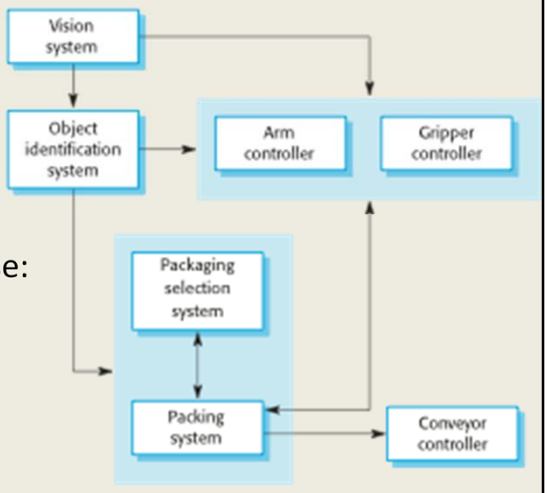
Now you see why we need different elements in requirements modeling?
Each of the elements of the requirements model (Chapters 6 and 7) provides information that is necessary to create the four design models required for a complete specification of design.

Data/Class design: transforms analysis classes into implementation classes and data structures
Architectural design: defines relationships among the major structural elements of the software, the architectural ***styles and patterns***
Interface design–defines how software elements, hardware elements, and end-users communicate. Usage scenarios and behavioral models are used
Component-level design–transforms structural elements into procedural descriptions of software components. Class-based models and behavioral models server as the basis

Only scenario-based elements serve just one design model

Software Architecture

- What is it?
  - Structure of software
    - Components (sub-systems)
    - Their external properties
    - Their relationships
  - Process of identifying these:
    - Architecture design
      - Output: architecture



The architecture of a packing robot control system

Let's consider the architecture of a building

Simply saying, the architecture is the overall shape of the physical structure

The architecture is the manner in which the various components of the building are integrated to form a cohesive whole

This is the way in which the building fits into its environment and meshes with other building in its vicinity

It is the degree to which the building meets its stated purpose and satisfies the needs of its owner

It is the aesthetic feel of the structure and the way textures, colors, and materials are combined

It is small details –the design of lighting fixtures, the type of flooring, the placement of wall hangings

Finally It is art

It is thousands of decisions, both big and small

Software Architecture

- What does it do for us?
  - Examine effectiveness of design
  - Consider design alternatives
  - Reduce risk
- Why is it important?
  - Enable communication
  - Highlight early yet impactful design decisions
  - Offer a graspable mode on structure and workings
  - Provide a holistic (gestalt) view

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 alternative sat a stage when making design changes is still relatively easy, and

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

Why important?

1.Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.

2.The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.

3.Architecture “constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together”

4.The architectural model provides a Gestalt view of the system, allowing the software engineer to examine it as a whole

Software Architecture

- How do we describe it?
  a set of work products that reflect different views of the system
  - Blueprint metaphor
  - Language metaphor
  - Decision metaphor
  - Literature metaphor

Blueprint metaphor – developers regard architecture descriptions as a means of transferring explicit information from architects to designers to software engineers charged with producing the system components

Language metaphor - Views architecture as a facilitator of communication across stakeholder groups

Decision metaphor - Represents architecture as the product of decisions involving trade-offs among properties such as cost, usability, maintainability, and performance; Stakeholders (project managers) view architectural decisions as the basis for allocating project resources and work tasks

Literature metaphor

It is used to document architectural solutions constructed in the past

This view supports the construction of artifacts and the transfer of knowledge between designers and software maintenance staff

It also supports stakeholders whose concern is reuse of components and designs

An architectural description must exhibit characteristics that combine these metaphors

Architecture: genre and style

- Genre
  - a specific category within the overall software domain
  - Examples: *AI, communication, content authoring, financial, game, medical, military, …*
- Style
  - Subcategories
  - Components, connectors, constraints, semantic models
  - Taxonomy:
    - Data-centered architectures
    - Data flow architectures
    - Call and return architectures
    - Object-oriented architectures
    - Layered architectures

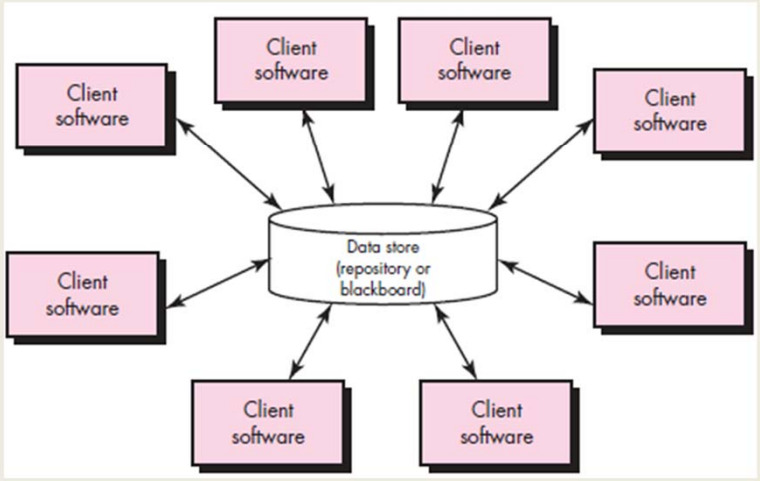
For example, within the genre of buildings, you would encounter the following general styles: houses, condos, apartment buildings, office buildings, industrial building, warehouses, and so on

Each style describes a system category that encompasses: (1) a set of components (e.g., a database, computational modules) that perform a function required by a system, (2) a set of connectors that enable "communication, coordination and cooperation" among components, (3) constraints that define how components can be integrated to form the system, and (4) semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

the architectural style is also a template for construction.

Architecture: genre and style

- Data-centered architecture



Data-centered architectures promote *integrability* [Bas03].

That is, existing components can be changed and new client components added to the architecture without concern about other clients

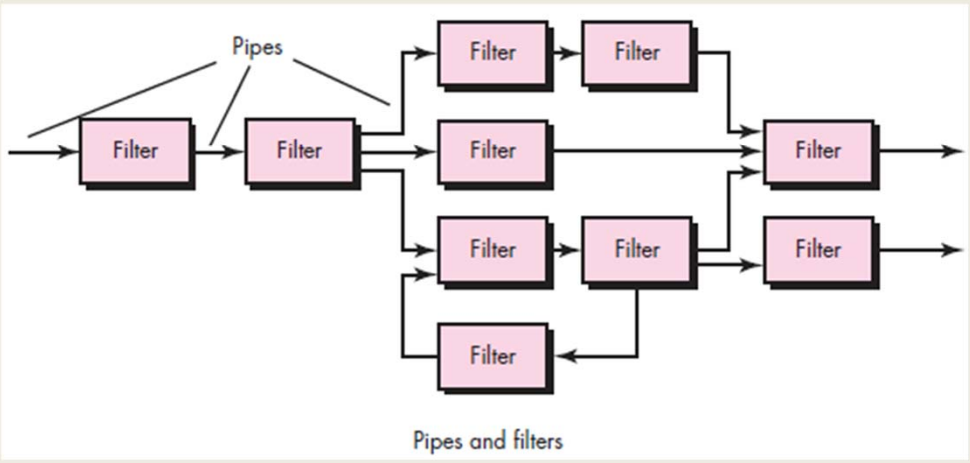
In addition, data can be passed among clients using the blackboard mechanism

The blackboard component serves to coordinate the transfer of information between clients

Client components independently execute processes.

Architecture: genre and style

- Data-flow architecture



Pipes and filters

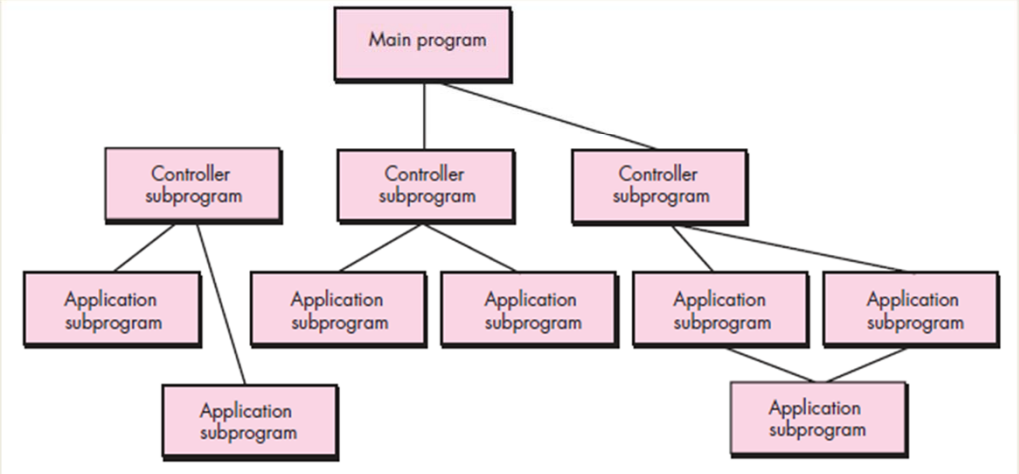
This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.

If the data flow degenerates into a single line of transforms, it is termed batch sequential.

This structure accepts a batch of data and then applies a series of sequential components (filters) to transform it.

Architecture: genre and style

- Call-return architecture



This architectural style enables you to achieve a program structure that is relatively easy to modify and scale.

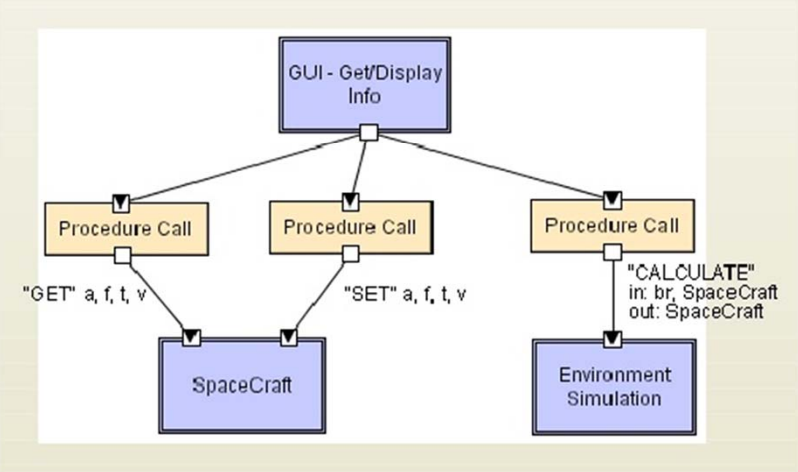
A number of substyles exist within this category:

(1) Main program/subprogram architectures. This classic program structure decomposes function into a control hierarchy where a "main" program invokes a number of program components that in turn may invoke still other components.

(2) Remote procedure call architectures. The components of a main program/subprogram architecture are distributed across multiple computers on a network.

Architecture: genre and style

- Object-oriented architecture

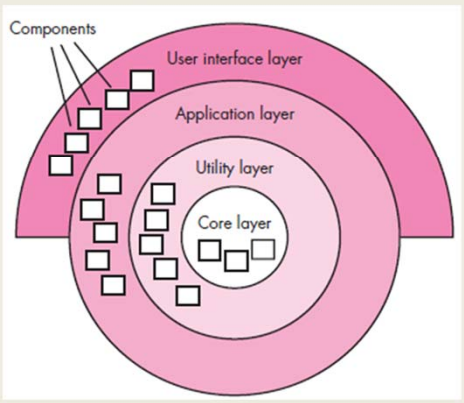


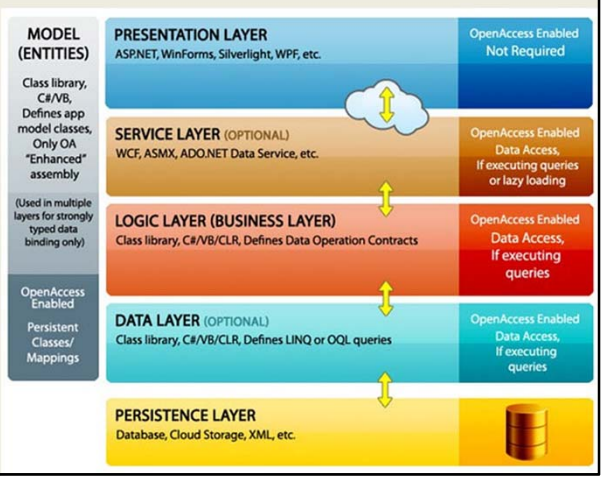
The components of a system encapsulate data and the operations that must be applied to manipulate the data.

Communication and coordination between components are accomplished via message passing.

Architecture: genre and style

- Layered architecture





A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.

Outer layer: components service user interface operations.

Inner layer: components perform operating system interfacing.

Intermediate layers: provide utility services and application software functions.

These architectural styles are only a small subset of those available

Architecture patterns

- What are they?
  - solutions for common issues in architecture design
  - a means of representing, sharing and reusing knowledge
  - a stylized description of good design practice, which has been tried and tested in different environments
    - Tabular / graphical

Patterns are a means of representing, sharing and reusing knowledge.

An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.

Patterns should include information about when they are and when the are not useful.

Patterns may be represented using tabular and graphical descriptions.

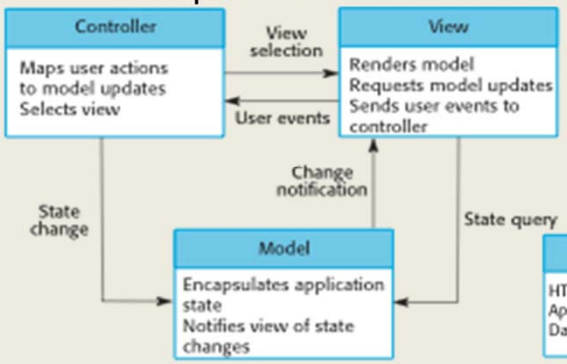
Architecture patterns

- Example: model-view-controller (MVC)

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. See Figure 6.3.
Example	Figure 6.4 shows the architecture of 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.

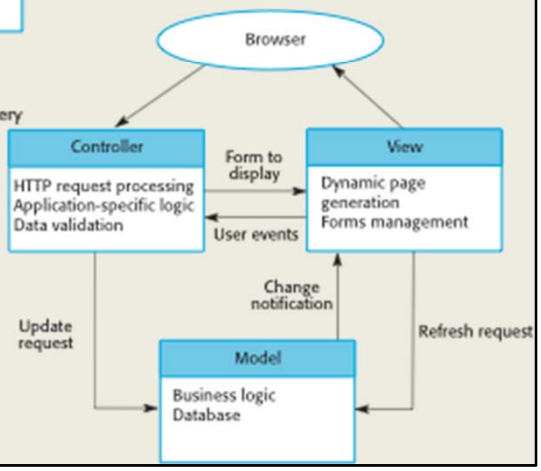
Architecture patterns

- Example: model-view-controller (MVC)



Pattern

Used in web app



Architectural design

- What do we do?
  - define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction
  - Identify architectural archetypes
  - Define/refine system components implementing archetypes -> structure
- Archetype
  - an abstraction (similar to a class) that represents one element of system behavior

The software must be placed into context

An *archetype* is an abstraction (similar to a class) that represents one element of system behavior

The designer specifies the structure of the system by defining and refining software components that implement each archetype

Architectural design: step 1

- Representing the system in Context
  - Consider architectural context
  - Define superordinate systems, subordinate systems, peer-level systems, actors

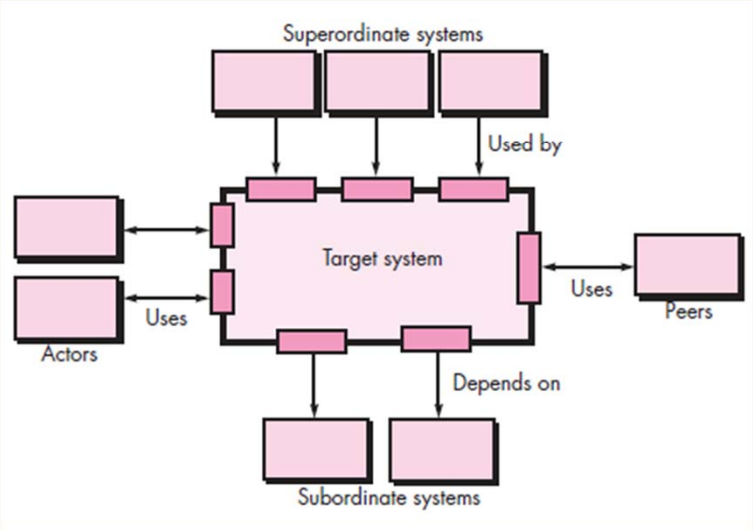
At the architectural design level, a software architect uses an *architectural context diagram*

(ACD) to model the manner in which software interacts with entities external

to its boundaries.

Architectural design: step 1

- Representation: architectural context diagram (ACD)



Systems that interoperate with the *target system*

Superordinate systems—those systems that use the target system as part of some higher-level processing scheme.

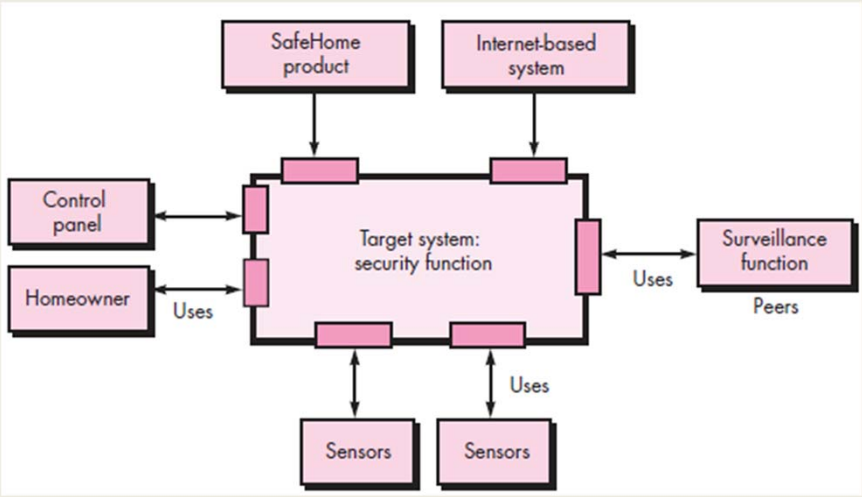
Subordinate systems—those systems that are used by the target system and provide data or processing that are necessary to complete target system functionality.

Peer-level systems—those systems that interact on a peer-to-peer basis (i.e., information is either produced or consumed by the peers and the target system.

Actors—entities (people, devices) that interact with the target system by producing or consuming information that is necessary for requisite processing

Architectural design: step 1

- Representation: architectural context diagram (ACD)



The overall *SafeHome* product controller and the Internet-based system are both superordinate to the security function

The surveillance function is a *peer system* and uses (is used by) the home security function in later versions of the product.

The homeowner and control panels are actors that are both producers and consumers of information used/produced by the security software.

Finally, sensors are used by the security software and are shown as subordinate to it.

Architectural design: step 2

- Defining archetypes
  - The target system architecture is composed of these archetypes, which represent stable elements of the architecture
  - may be instantiated many different ways based on the behavior of the system

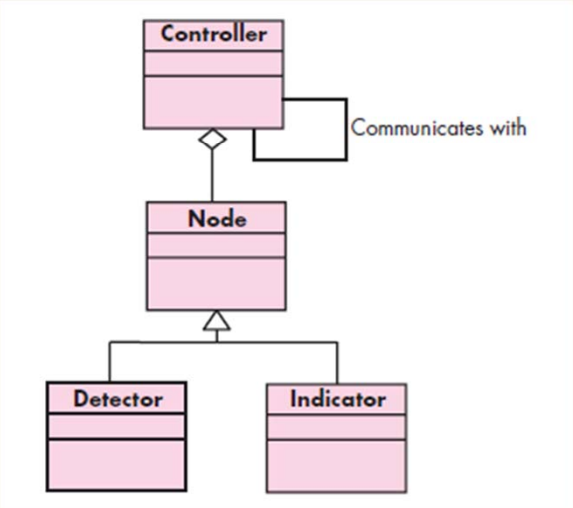
Archetype Is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system.

Archetypes are the abstract building blocks of an architectural design

In general, a relatively small set of archetypes is required to design even relatively complex systems

Architectural design: step 2

- Representation: class diagrams



Node. Represents a cohesive collection of input and output elements of the home security function.

For example a node might be comprised of (1) various sensors and (2) a variety of alarm (output) indicators

Detector. An abstraction that encompasses all sensing equipment that feeds information into the target system.

Indicator. An abstraction that represents all mechanisms (e.g., alarm siren, flashing lights, bell) for indicating that an alarm condition is occurring.

Controller. An abstraction that depicts the mechanism that allows the arming or disarming of a node. If controllers reside on a network, they have the ability to communicate with one another.

Architectural design: step 3

- Refining the Architecture into Components
  - Analysis classes -> Components
  - Consider domains for choosing components
    - Application domain
    - Infrastructure domain

How are these components chosen? We need to refer to two sources domains

1) The application domain

Describes application-relevant entities in analysis classes within the application (business) domain

2) The infrastructure domain

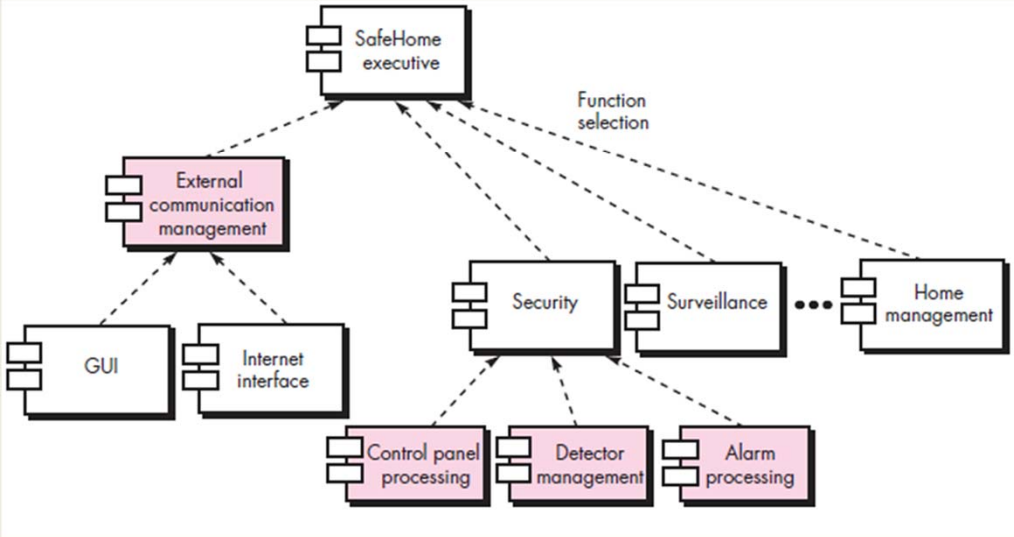
Many infrastructure components that enable application components

But have no business connection to the application domain.

•E.g.) memory management components, communication components, database components, and task management components

Architectural design: step 3

- Representation: component diagrams



External communication management—coordinates communication of the security function with external entities such as other Internet-based systems and external alarm notification.

Control panel processing—manages all control panel functionality.

Detector management—coordinates access to all detectors attached to the system.

Alarm processing—verifies and acts on all alarm conditions.

Each of these top-level components would have to be elaborated iteratively and then positioned within the overall *SafeHome* architecture.

Design classes (with appropriate attributes and operations) would be defined for each.

It is important to note, however, that the design details of all attributes and operations would not be specified until component-level design

Architectural design: step 4

- Describing Instantiations of the System
  - Further refine the architectural design
  - Develop an actual instantiation of the architecture
    - Apply it to specific problem

The architectural design that has been modeled to this point is still relatively high level.

Further refinement (recall that all design is iterative) is still necessary.

To accomplish this, an actual instantiation of the architecture is developed.

The architecture is **applied to a specific problem** with the intent of demonstrating that the structure and components are appropriate.

Architectural design: step 4

- Representation: (refined) component diagrams

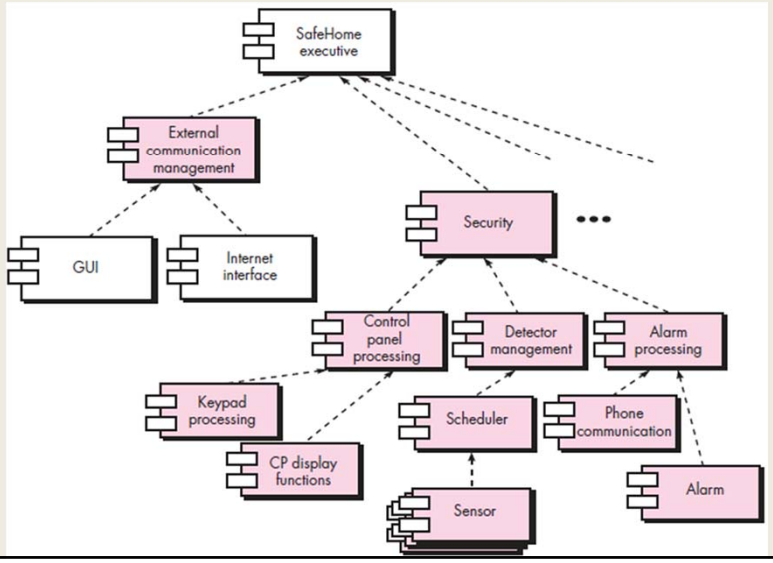


Figure 9.9 illustrates an instantiation of the *SafeHome* architecture for the security

system. Components shown in Figure 9.8 are elaborated to show additional detail.

For example, the *detector management* component interacts with a *scheduler* infrastructure

component that implements polling of each *sensor* object used by the security

system. Similar elaboration is performed for each of the components

represented in Figure 9.8.

Summary

- Architecture genres and styles
  - Many genres
  - Five common styles
- Architectural design pattern
  - MVC
- Design steps
  - Four steps of architectural design