Chapter 10

- **Architectural Design**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 8/e

by Roger S. Pressman and Bruce R. Maxim

Software Design

Design is an activity concerned with making major decisions, often of a structural nature.

It shares with programming a concern for abstracting information representation and processing sequences, but the level of detail is quite different at the extremes.

Design builds coherent, well-planned representations of programs that concentrate on the interrelationships of parts at the higher level and the logical operations involved at the lower levels.

What is Architecture?

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

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.

Why is Architecture Important?

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- 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.
- Architecture "constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together" [BAS03].

Formal Descriptions of Architectural

- The IEEE Computer Society has proposed IEEE-Std-1471-2000, *Recommended Practice for Architectural Description of Software-Intensive System*, [IEE00]
  - to establish a conceptual framework and vocabulary for use during the design of software architecture,
  - to provide detailed guidelines for representing an architectural description, and
  - to encourage sound architectural design practices.
- The IEEE Standard defines *an architectural description* (AD) as a "a collection of products to document an architecture."
  - The description itself is represented using multiple views, where each *view* is "a representation of a whole system from the perspective of a related set of [stakeholder] concerns."

Architectural Decisions

- Each view developed as part of an architectural description addresses a specific stakeholder concern.
- To develop each view and the architectural description as a whole, the system architect considers a variety of alternatives and ultimately decides on the specific architectural features that best meet the concern.
- Architectural decisions themselves can be considered to be one view of the architecture. The reasons that decisions were made provide insight into the structure of a system and its conformance to stakeholder concerns.

Architectural Decisions

INFO

Architecture Decision Description Template

Each major architectural decision can be documented for later review by stakeholders who want to understand the architecture description that has been proposed. The template presented in this sidebar is an adapted and abbreviated version of a template proposed by Tyree and Ackerman [Tyr05].

Design issue: Describe the architectural design issues that are to be addressed.

Resolution: State the approach you've chosen to address the design issue.

Category: Specify the design category that the issue and resolution address (e.g., data design, content structure, component structure, integration, presentation).

Assumptions: Indicate any assumptions that helped shape the decision.

Constraints: Specify any environmental constraints that helped shape the decision (e.g., technology standards, available patterns, project-related issues).

Alternatives: Briefly describe the architectural design alternatives that were considered and why they were rejected.

Argument: State why you chose the resolution over other alternatives.

Implications: Indicate the design consequences of making the decision. How will the resolution affect other architectural design issues? Will the resolution constrain the design in any way?

Related decisions: What other documented decisions are related to this decision?

Related concerns: What other requirements are related to this decision?

Work products: Indicate where this decision will be reflected in the architecture description.

Notes: Reference any team notes or other documentation that was used to make the decision.

Architectural Genres

- *Genre* implies a specific category within the overall software domain.
- Within each category, you encounter a number of subcategories.
  - 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.
  - Within each general style, more specific styles might apply. Each style would have a structure that can be described using a set of predictable patterns.

Architectural Styles

Each style describes a system category that encompasses:

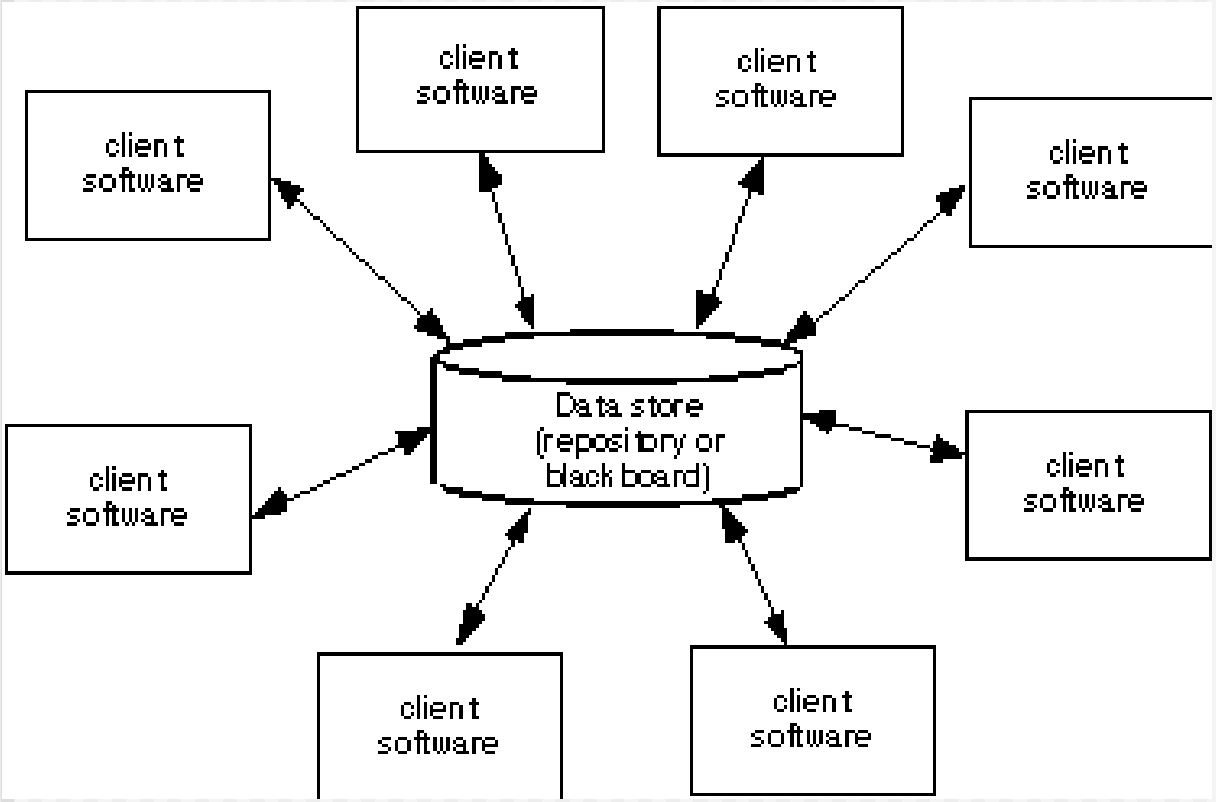
(1) a **set of components** (e.g., a database, computational modules) that perform a function required by a system,

(2) a **set of connectors** that enable "communication, coordination and cooperation" among components,

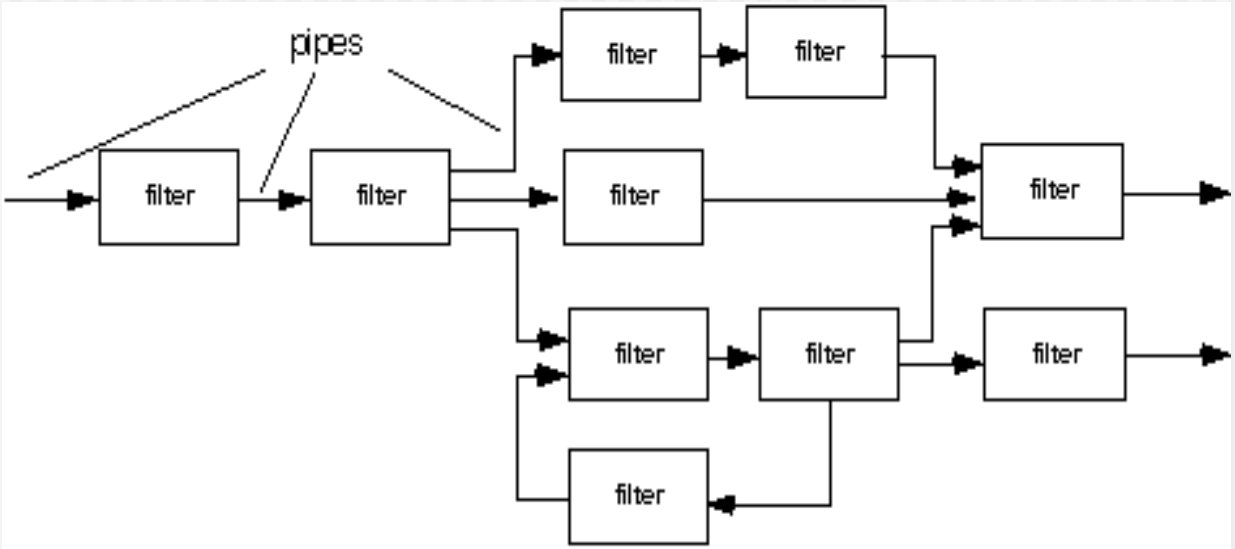
(3) **constraints** that define how components can be integrated to form the system.

(4) **semantic models** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

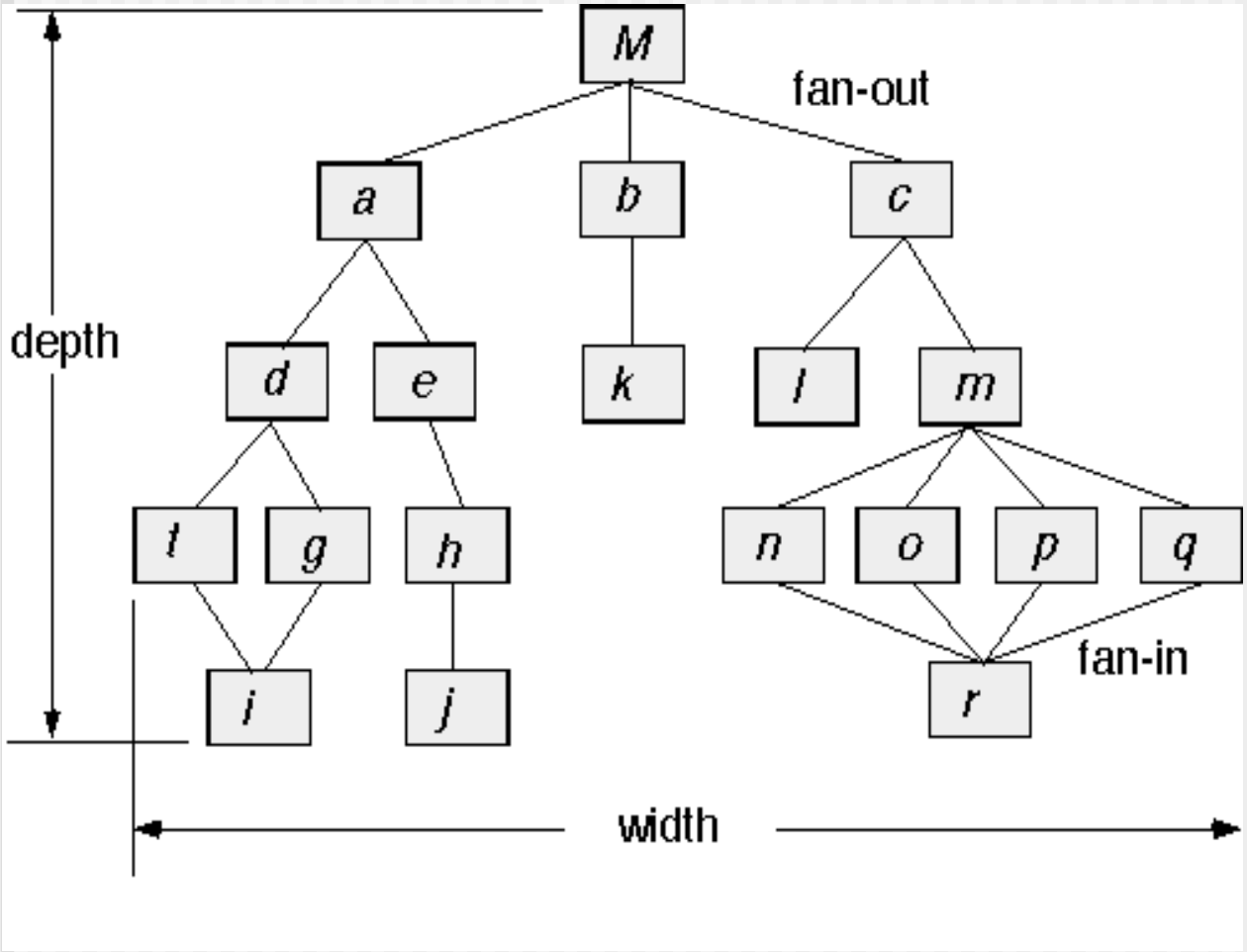
Data-Centered Architecture



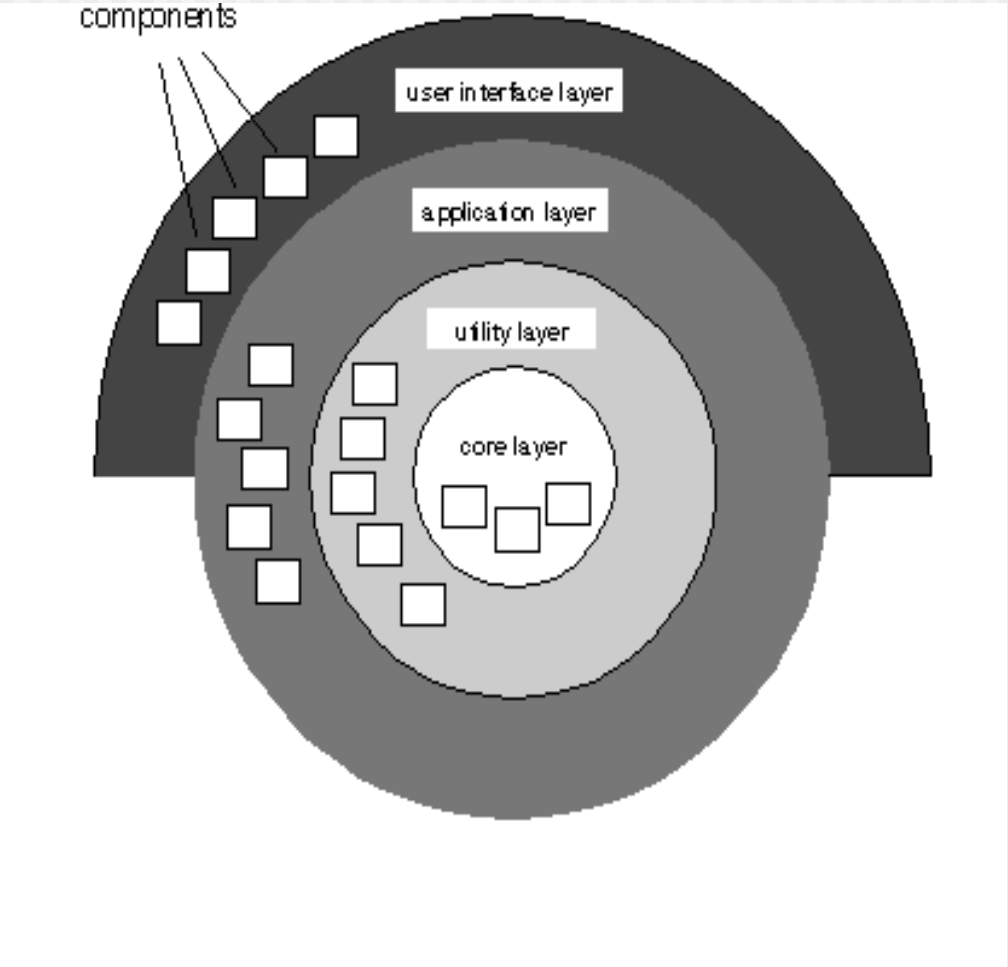
Data Flow Architecture



Call and Return Architecture



Layered Architecture



Architectural Patterns

- Concurrency—applications must handle multiple tasks in a manner that simulates parallelism
  - *operating system process management* pattern.
  - *task scheduler* pattern.
- Persistence—Data persists if it survives past the execution of the process that created it. Two patterns are common:
  - a *database management system* pattern that applies the storage and retrieval capability of a DBMS to the application architecture.
  - an *application level persistence* pattern that builds persistence features into the application architecture.
- Distribution— the manner in which systems or components within systems communicate with one another in a distributed environment
  - A *broker* acts as a 'middle-man' between the client component and a server component.

Architectural Considerations

- Economy – The best software is uncluttered and relies on abstraction to reduce unnecessary detail.
- Visibility – Architectural decisions and the reasons for them should be obvious to software engineers who examine the model at a later time.
- Spacing – Separation of concerns in a design without introducing hidden dependencies.
- Symmetry – Architectural symmetry implies that a system is consistent and balanced in its attributes.
- Emergence – Emergent, self-organized behavior and control.

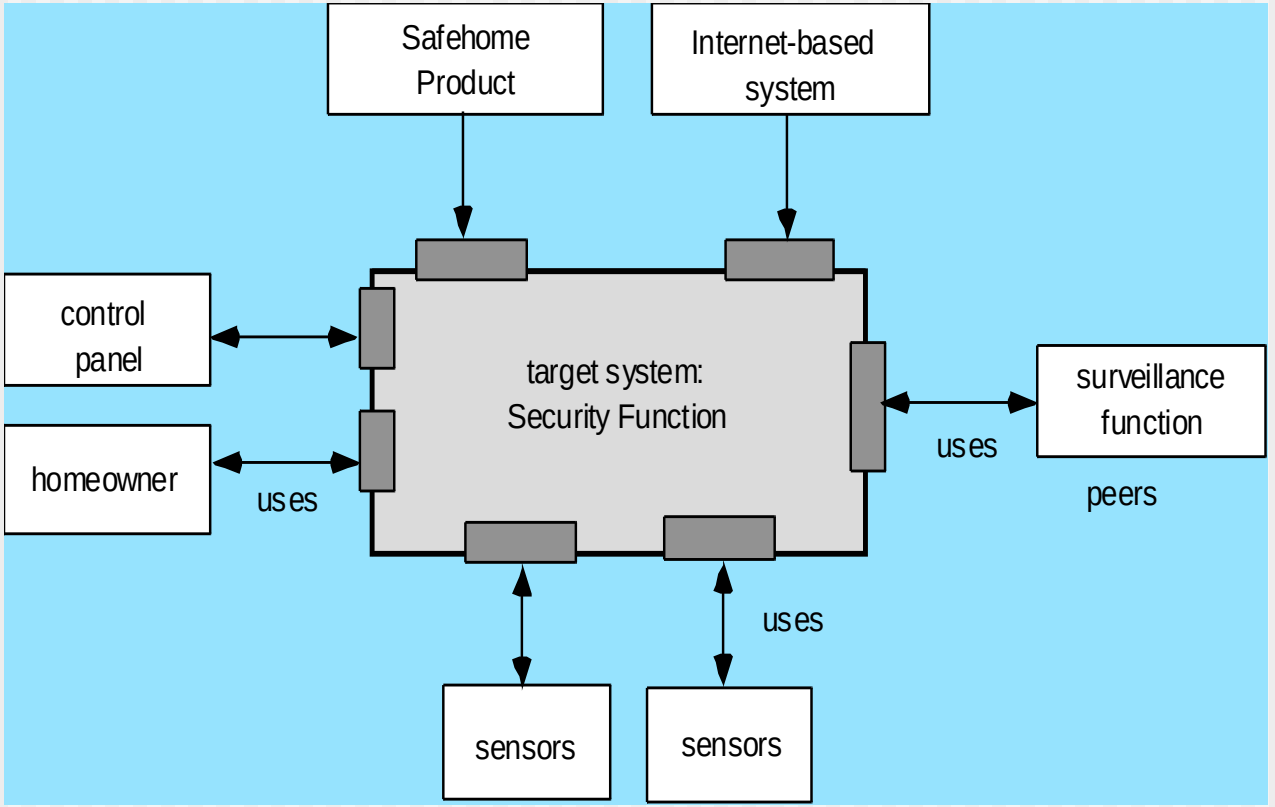
Architectural Decision Documentation

1. Determine which information items are needed for each decision.
2. Define links between each decision and appropriate requirements.
3. Provide mechanisms to change status when alternative decisions need to be evaluated.
4. Define prerequisite relationships among decisions to support traceability.
5. Link significant decisions to architectural views resulting from decisions.
6. Document and communicate all decisions as they are made.

Architectural Design

- The software must be placed into context
  - the design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction.
- A set of architectural archetypes should be identified
  - 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 Context



Archetypes

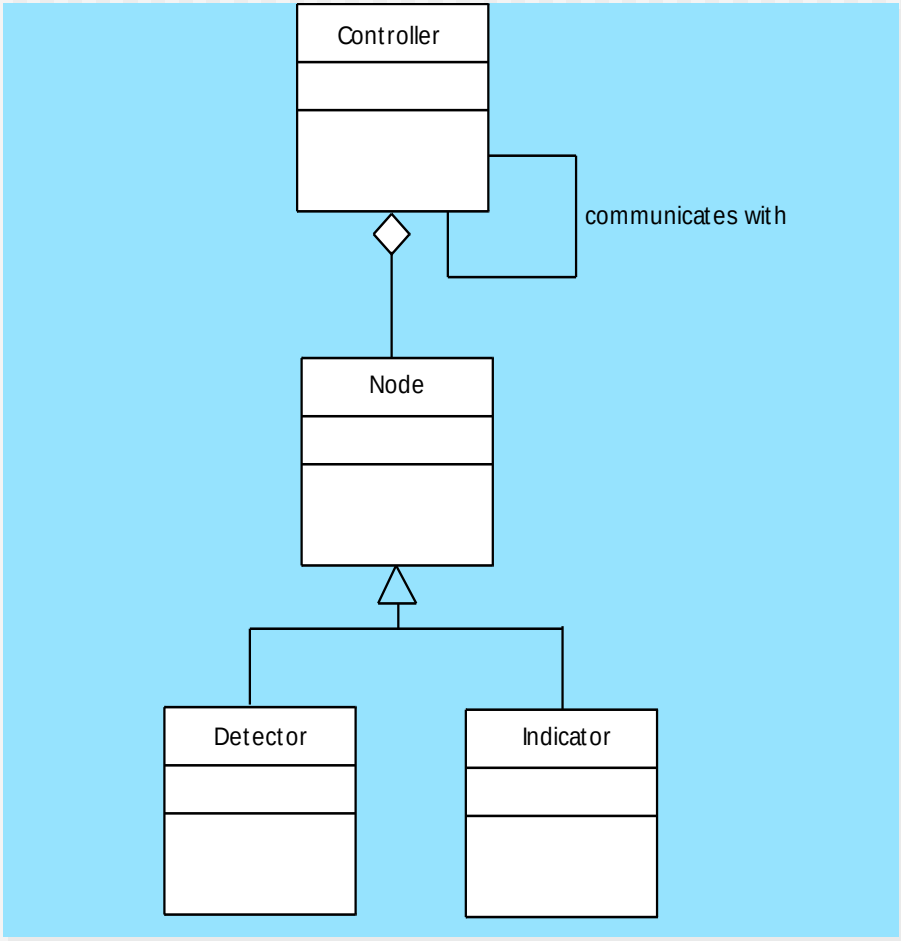
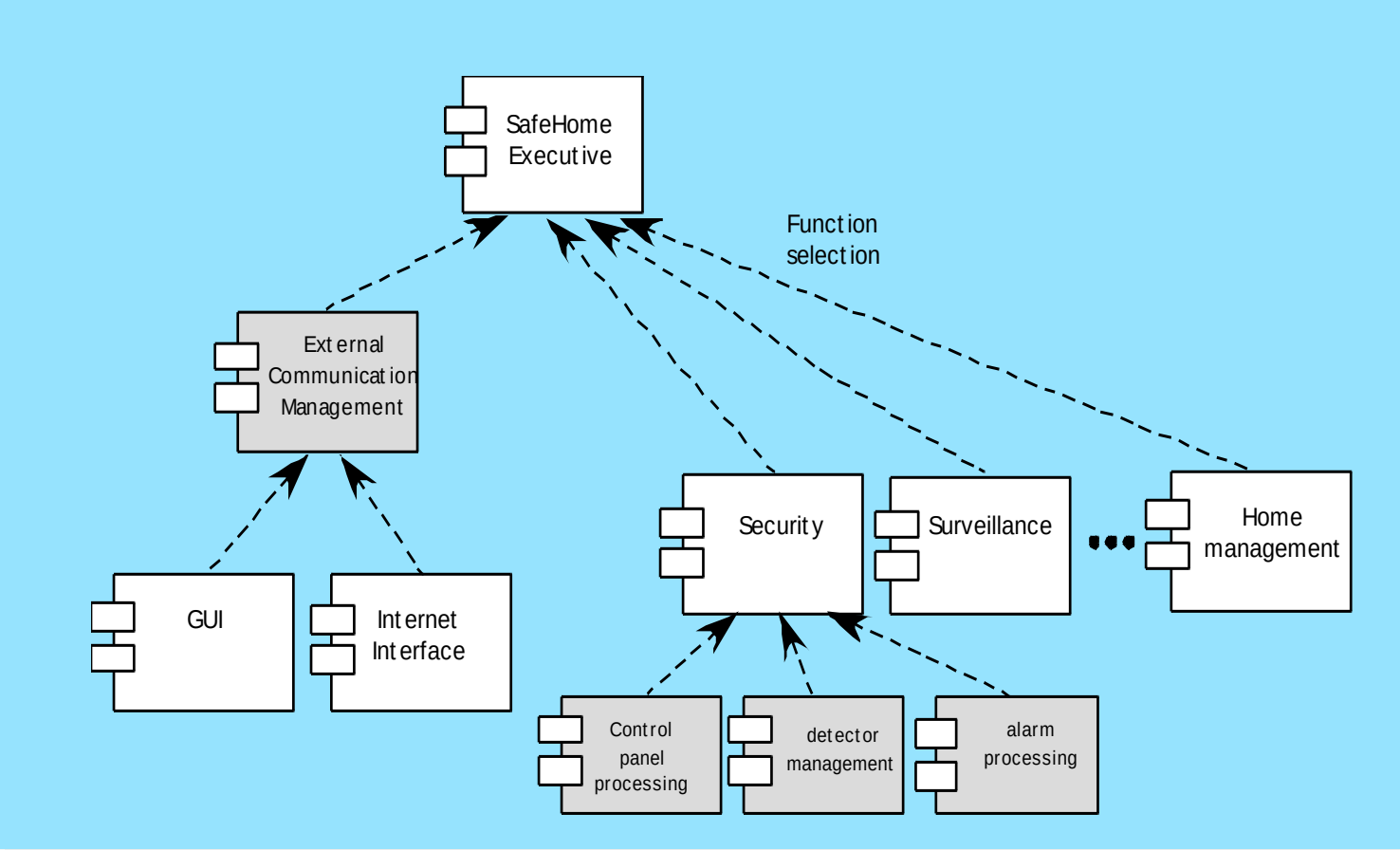
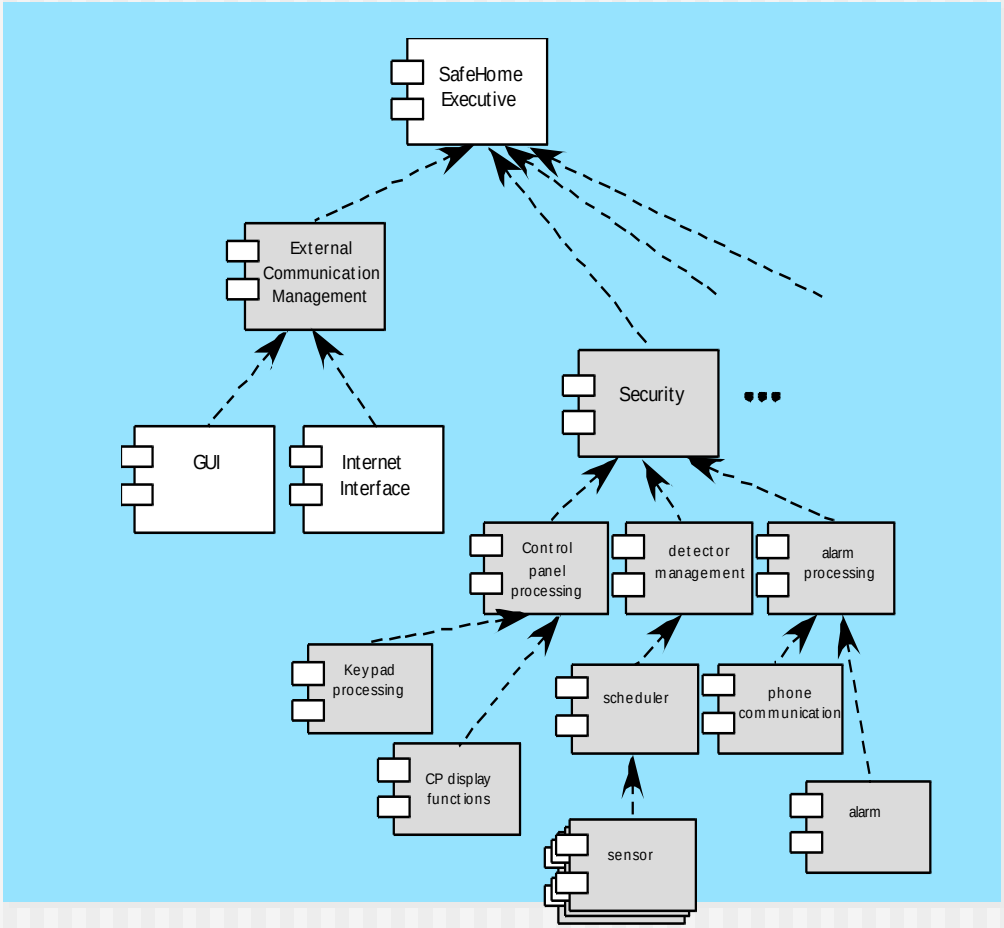


Figure 10.7 UML relationships for SafeHome security function archetypes (adapted from [BOS00])

Component Structure



Refined Component Structure



Architectural Tradeoff Analysis

1. Collect scenarios.
2. Elicit requirements, constraints, and environment description.
3. Describe the architectural styles/patterns that have been chosen to address the scenarios and requirements:
    - module view
    - process view
    - data flow view
4. Evaluate quality attributes by considered each attribute in isolation.
5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style.
6. Critique candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

Architecture Reviews

- Assess the ability of the software architecture to meet the systems quality requirements and identify potential risks.
- Have the potential to reduce project costs by detecting design problems early.
- Often make use of experience-based reviews, prototype evaluation, and scenario reviews, and checklists.

Agility and Architecture

- To avoid rework, user stories are used to create and evolve an architectural model (walking skeleton) before coding.
- Hybrid models which allow software architects contributing users stories to the evolving storyboard.
- Well run agile projects include delivery of work products during each sprint.
- Reviewing code emerging from the sprint can be a useful form of architectural review.

Summary

- Architectural design represents the structure of the data and program components required to build a computer-based system. A number of architectural "styles" exist. Architectural design begins with data design and proceeds to the derivation of one or more representations of the architectural structure of the system. The resulting architectural model encompasses both the data architecture and the program structure. Alternative architectural patterns are analyzed to determine the structure that is best suited to the customer’s requirements. The architectural model is subjected to software quality review like all other design work products.

Assignment

- Preview

  《Software Engineering》(8th Edition)

  Chapter 14 Component-level Design

  by R.S.Pressman