Lecture 9: Chapter 9

- **Architectural Design**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

Why Architecture?

- Architecture of a system describes the components and how they fit together.
- 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].

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.

Genre Examples for Software Systems

- Artificial Intelligence
- Devices
- Sports
- Financial
- Games
- Medical
- Scientific
- Transportation
- Government
- Etc.

Architectural Styles

- There's a pattern or type of architecture at the back of each artist.
  - Differentiate a house from other styles
- Software also exhibits some styles
- Each style describes a system category that encompasses:

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

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

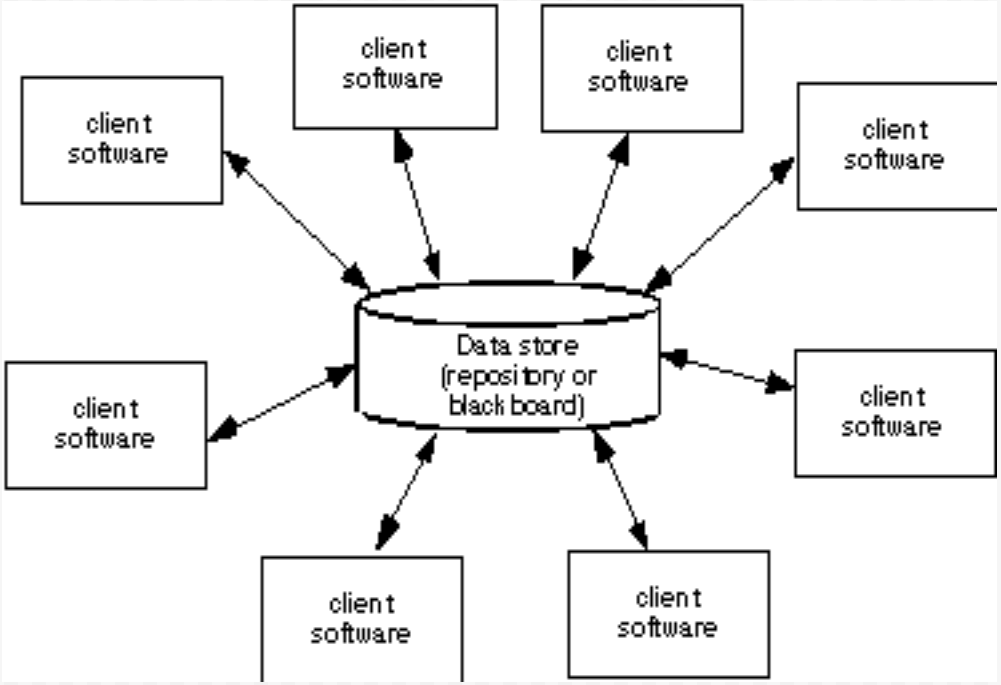
(3) constraints that define how components can be integrated to form the system, and

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

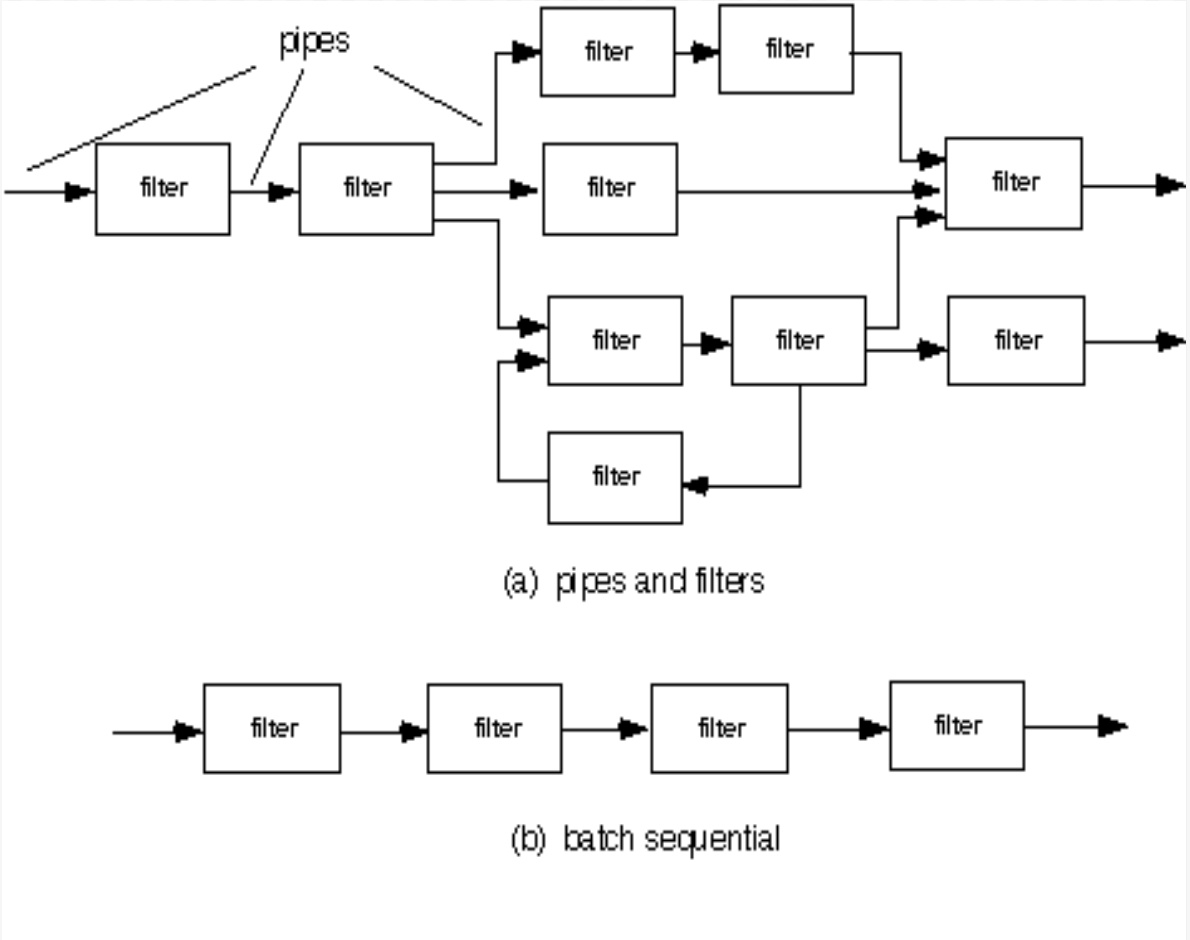
Taxonomy of Styles in Software

- Data-centered architectures
  - There is a central data server which is accessed by clients
- Data flow architectures
  - Data travels through a series of components
- Call and return architectures
  - Classical
- Object-oriented architectures
  - Modern style. Components pass messages
- Layered architectures
  - High-level to machine level

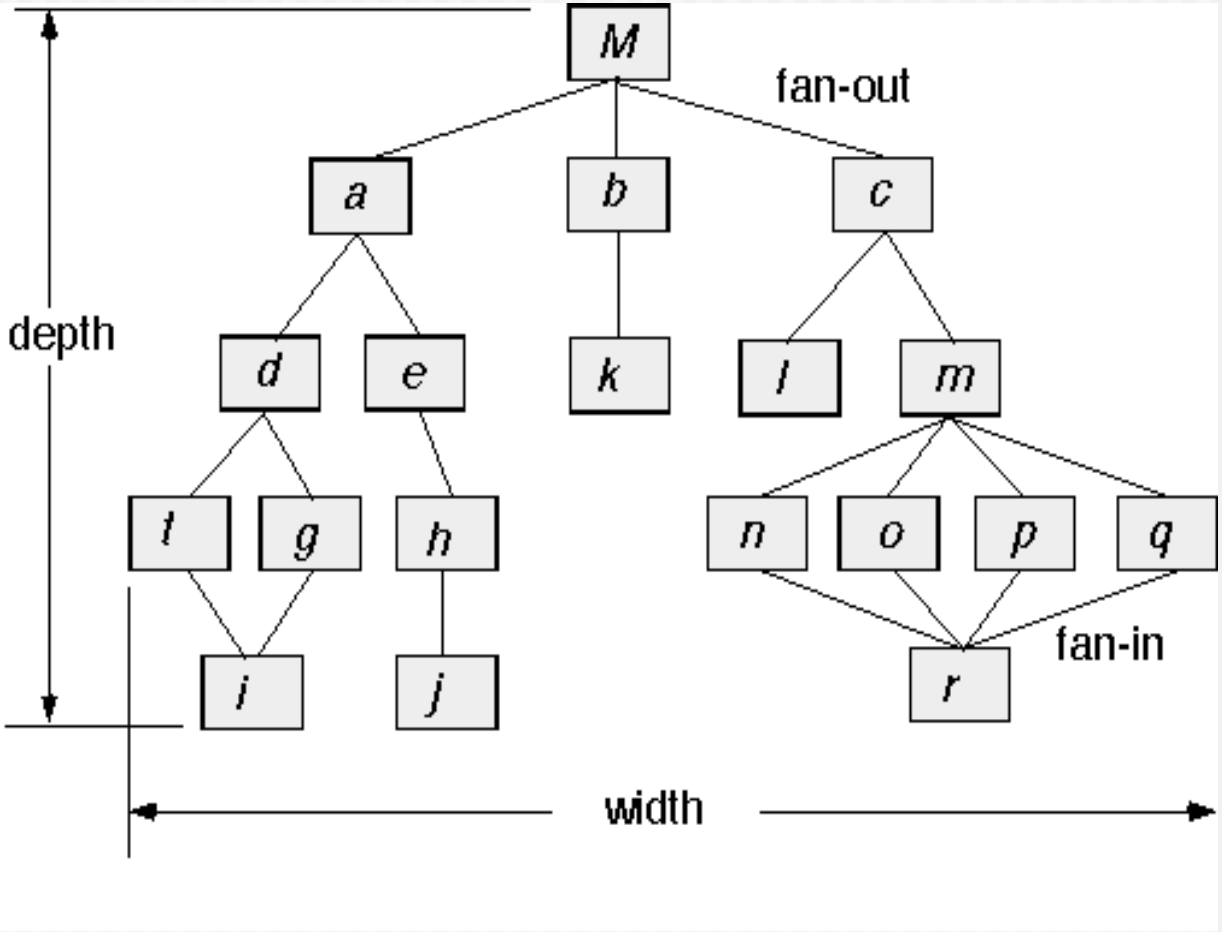
Data-Centered Architecture



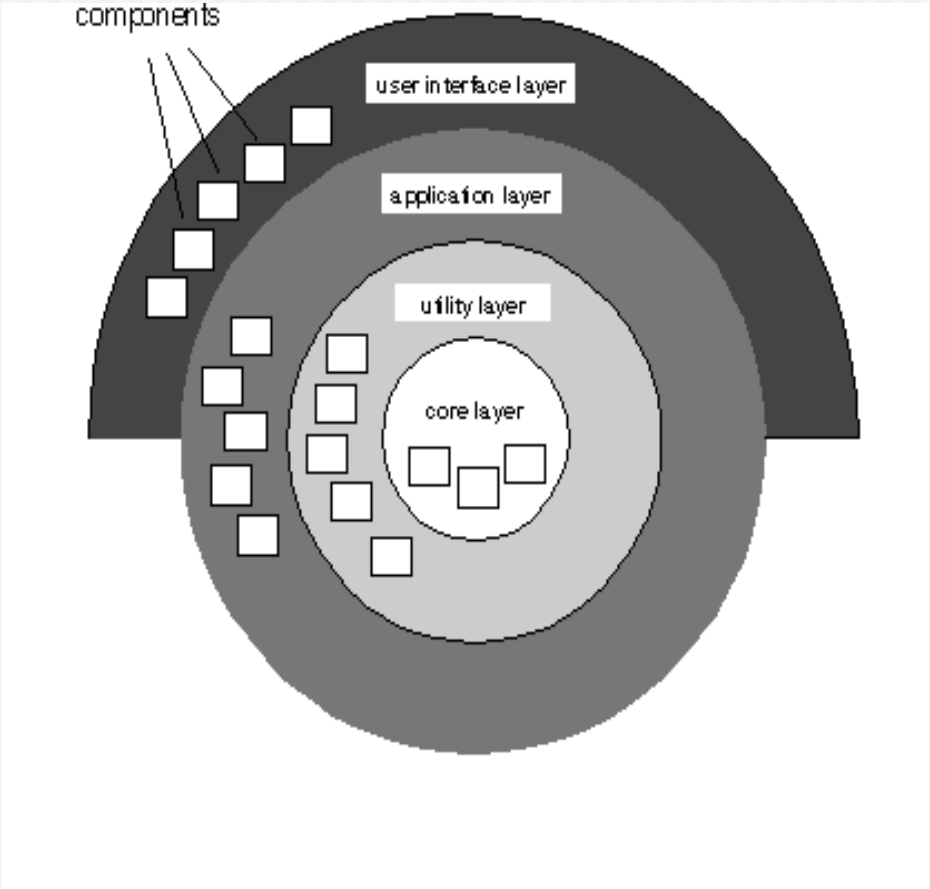
Data Flow Architecture



Call and Return Architecture



Layered Architecture



Architectural Patterns

- Design solutions to recurring problems

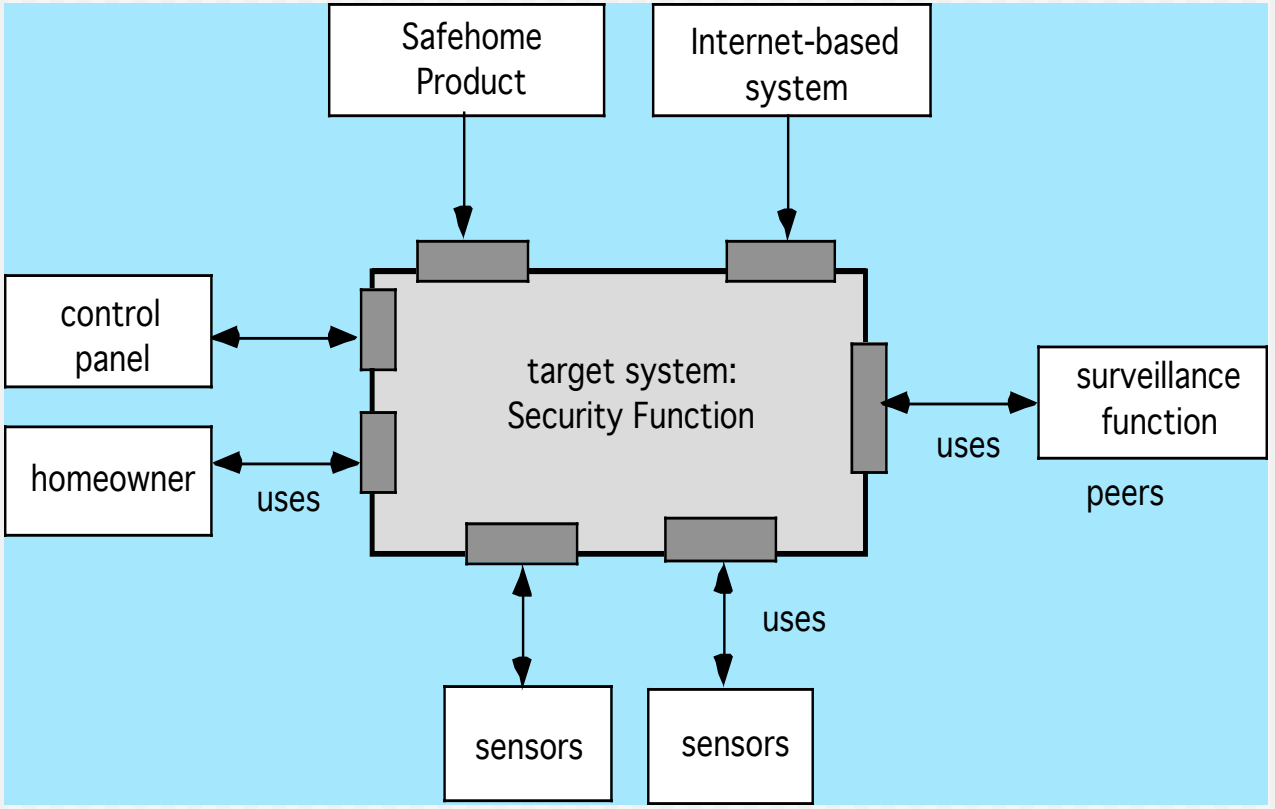
Examples:

- 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 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

Abstract Building Blocks

- Node
  - Input/output
- Controller
  - Arms-disarms a node
- Detector
  - Sensing
- Indicator
  - Alarms

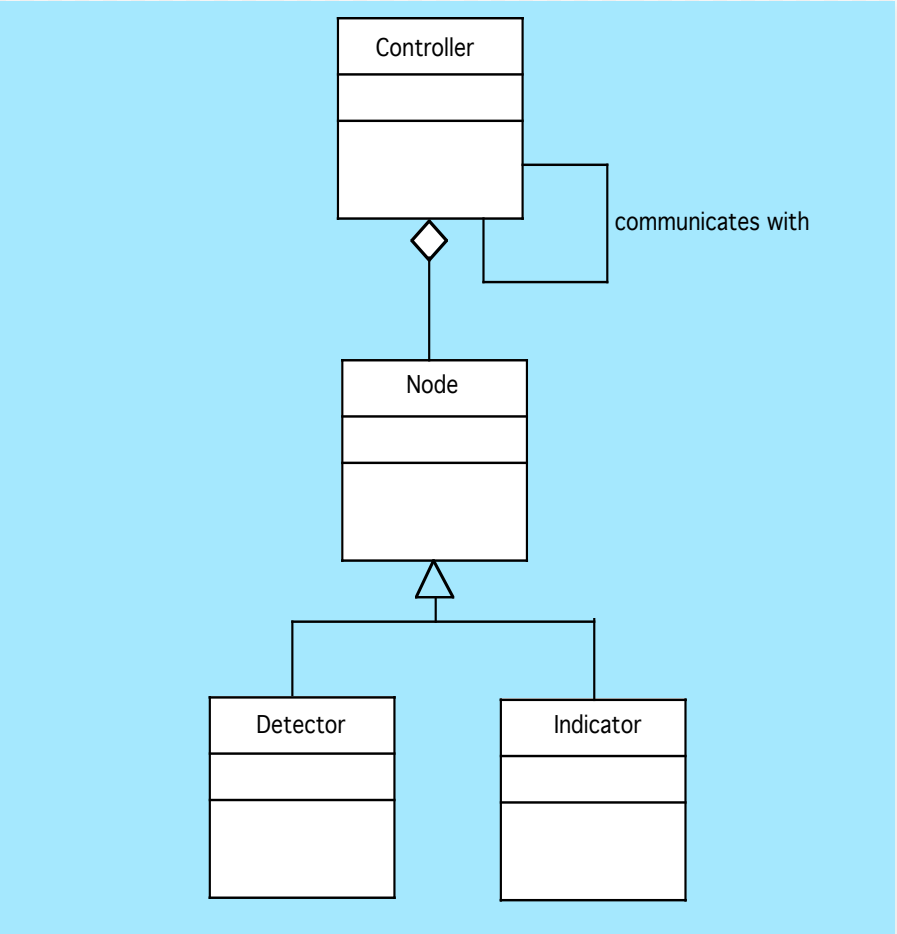
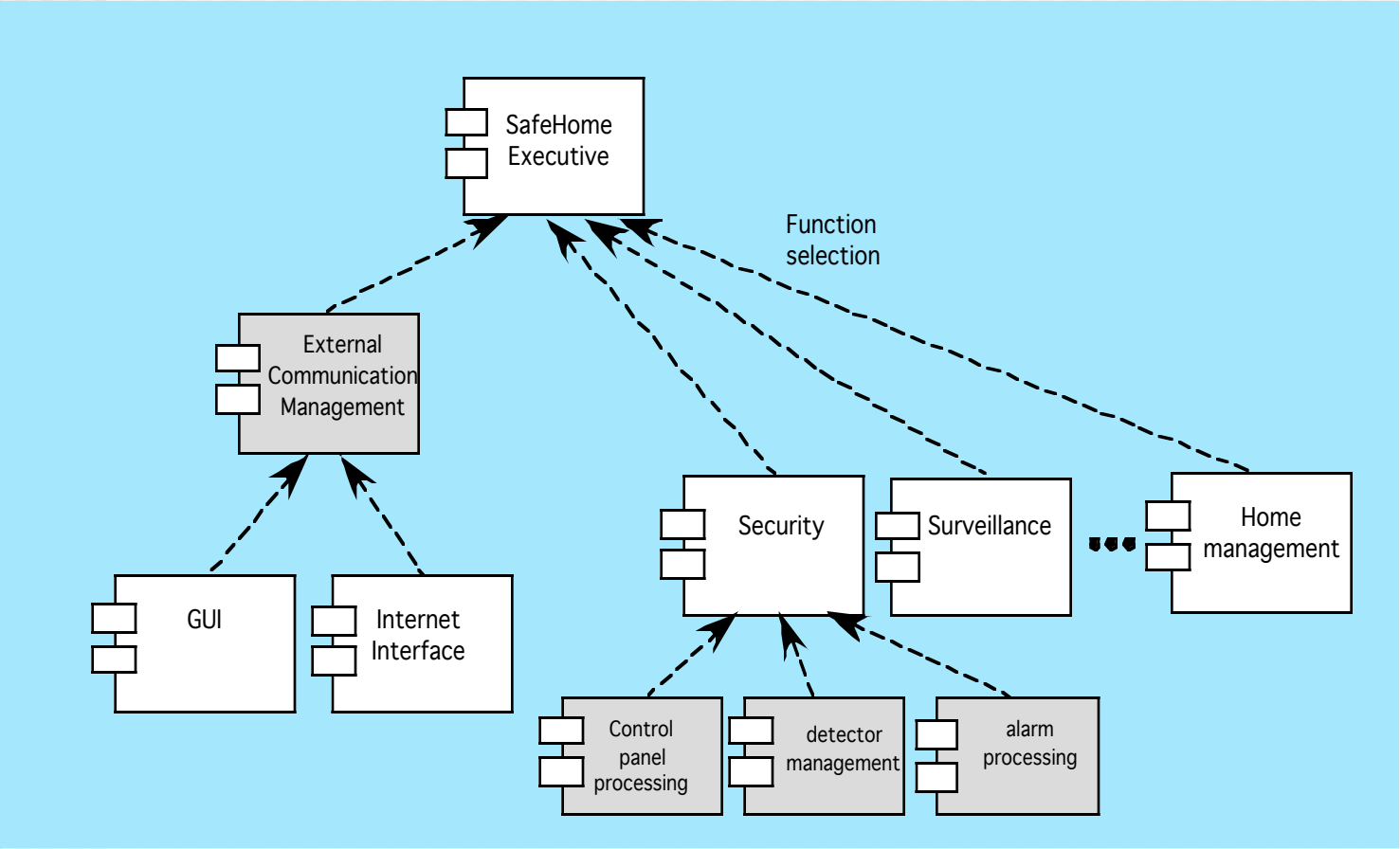
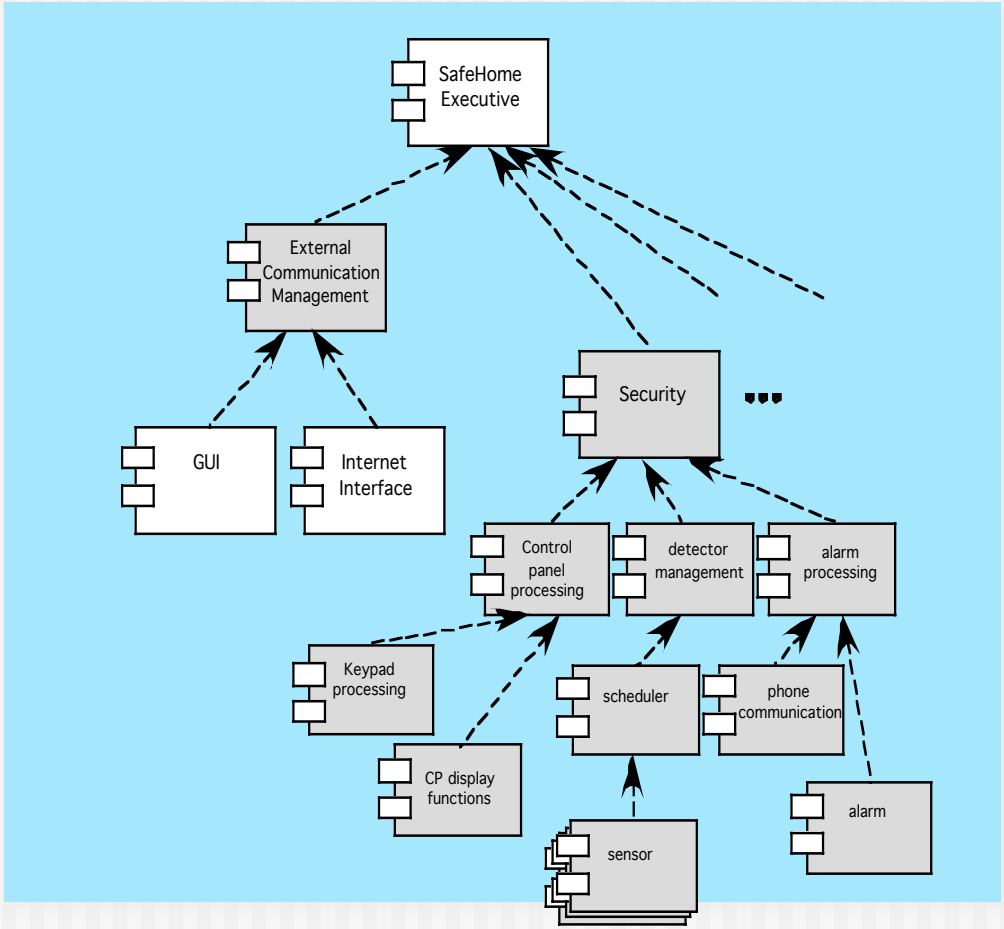


Figure 10.7 UML relationships for SafeHome security function archetypes (adapted from [BOS00])

Component Structure



Refined Component Structure



Analyzing Architectural Design

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 (reliability, performance, portability, etc.)
5. Identify the sensitivity of quality attributes to various architectural changes for a specific architectural style.
6. Critique candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

Based on the results of step 5 and 6, changes can be made.

Architectural Complexity

- The overall complexity of a proposed architecture is assessed by considering the dependencies between components within the architecture [Zha98]
  - *Sharing dependencies* represent dependence relationships among consumers who use the same resource or producers who produce for the same consumers (using same variables)
  - *Flow dependencies* represent dependence relationships between producers and consumers of resources (prerequisites)
  - *Constrained dependencies* represent constraints on the relative flow of control among a set of activities (mutual exclusion).

ADL

- *Architectural description language* (ADL) provides a semantics and syntax for describing a software architecture
- Provide the designer with the ability to:
  - decompose architectural components
  - compose individual components into larger architectural blocks and
  - represent interfaces (connection mechanisms) between components.