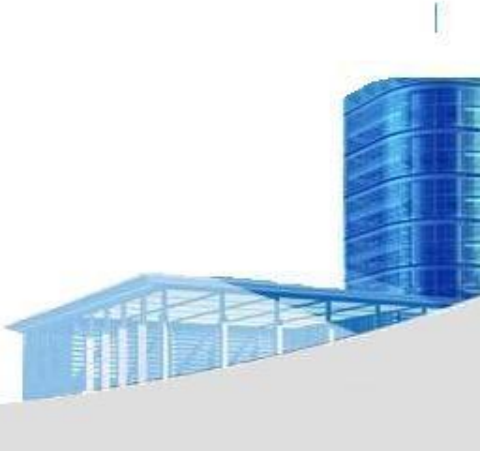
Ch.13 Architectural Design



- **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.

- **Why is Architecture Important?**

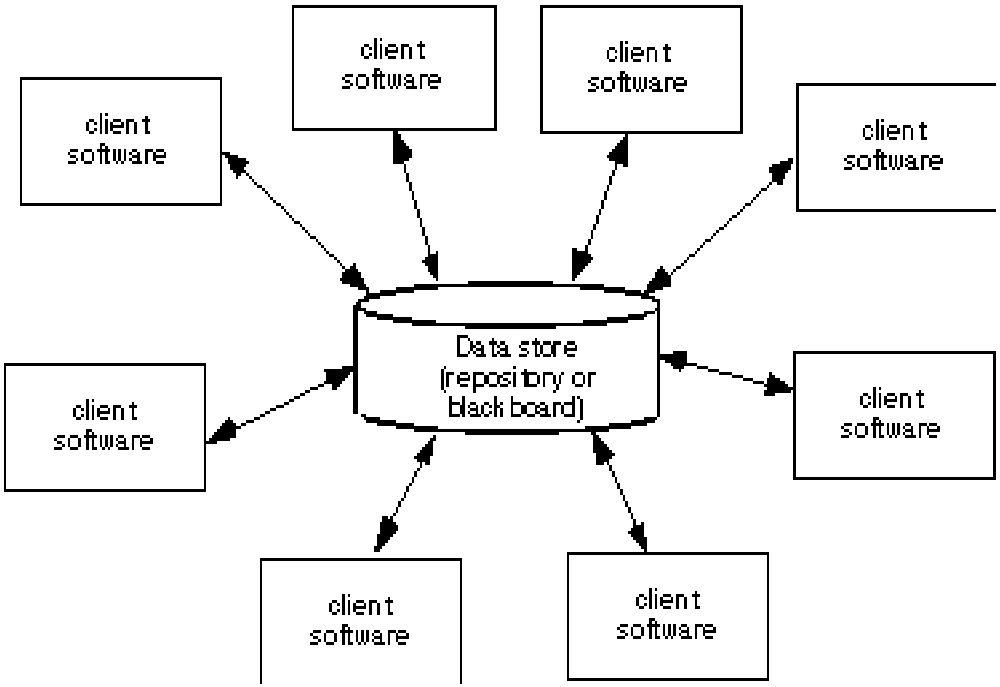
  - *Representations of software architecture are an enabler* for communication between all parties (stakeholders) interested in the development of a computer-based system.
  - *The architecture highlights early design decisions* that will have a profound impact on all software engineering work that follows 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 Descriptions**
  - 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."

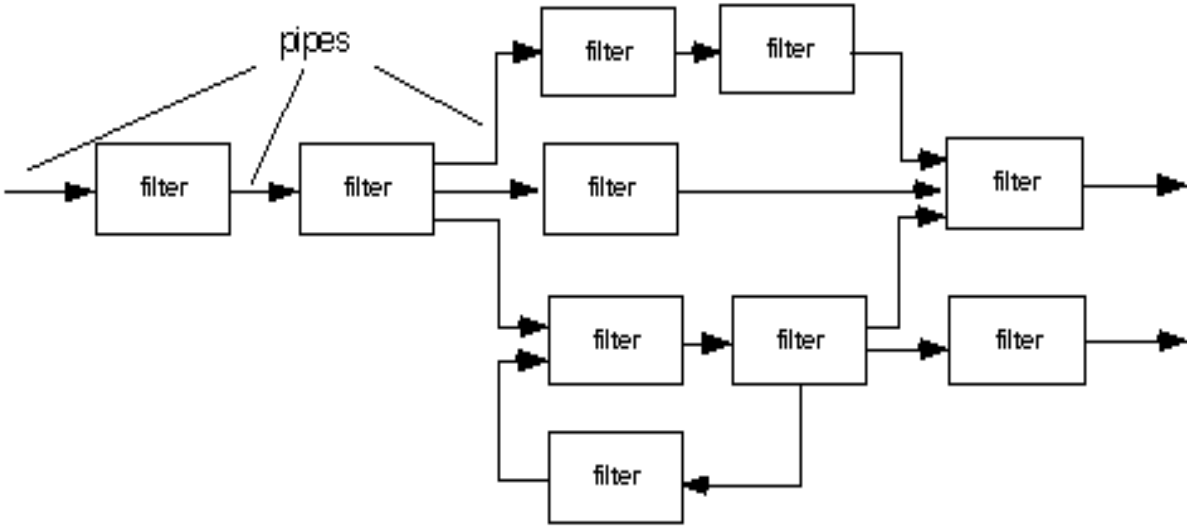
- **Architectural Genres**
  - *Genre* implies a specific category within the overall software domain.
  - Within each category, you encounter a number of subcategories.
    - For example, within the genre of buildings, you would encounter the following general ***styles***: houses, condos, apartment 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, 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.
    - Data-centered architectures
    - Data flow architectures
    - Call and return architectures
    - Object-oriented architectures
    - Layered architectures

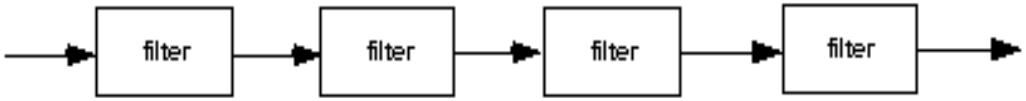
- **Data-Centered Architecture**



- **Data Flow Architecture**

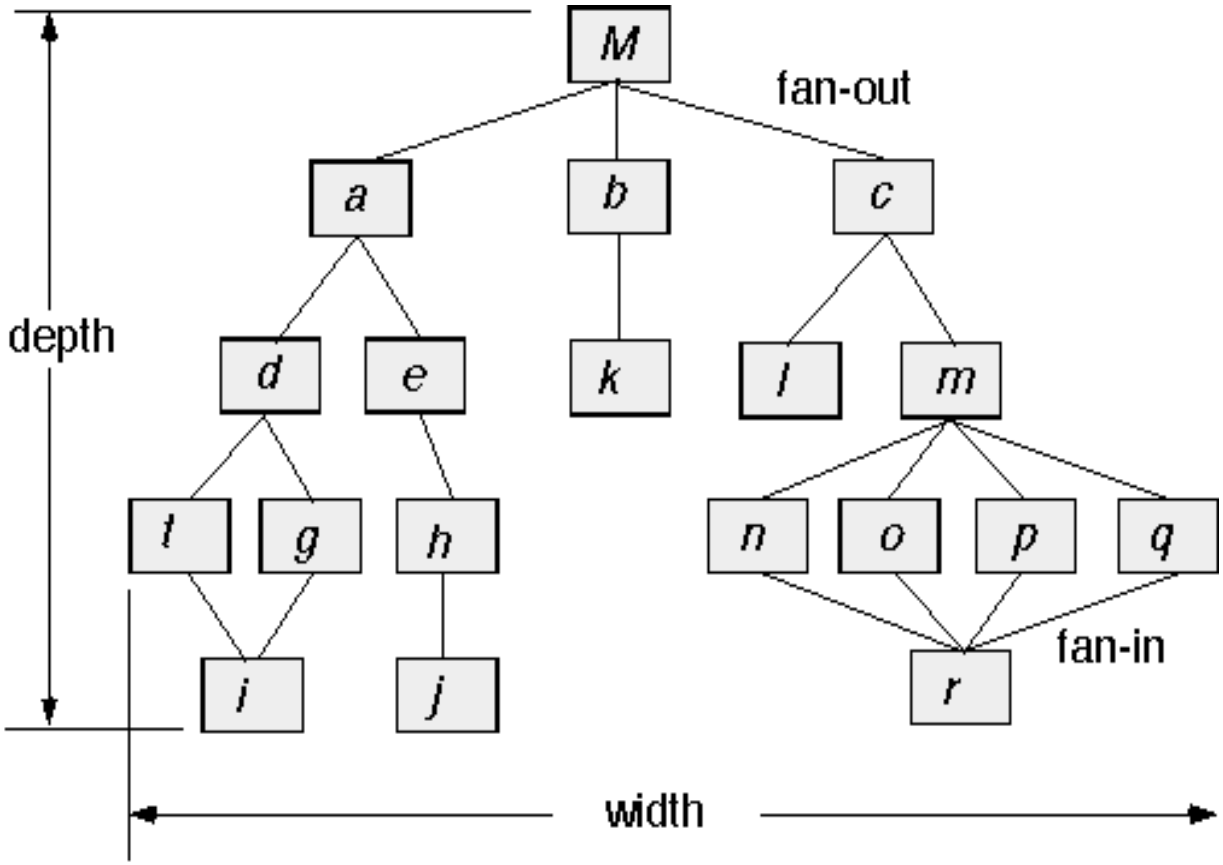


(a) pipes and filters

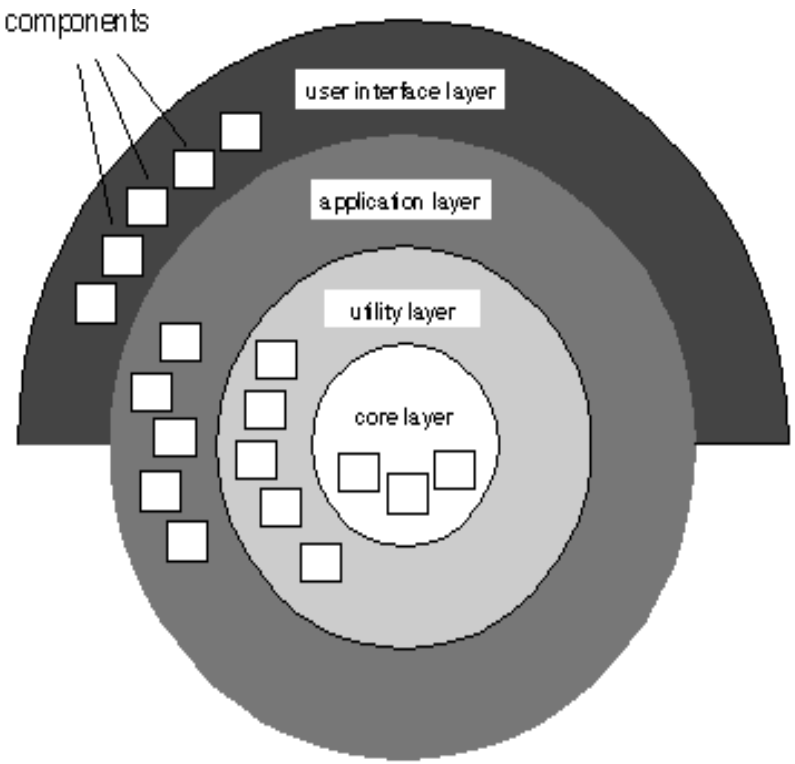


(b) batch sequential

- **Call and Return Architecture**



- **Layered Architecture**

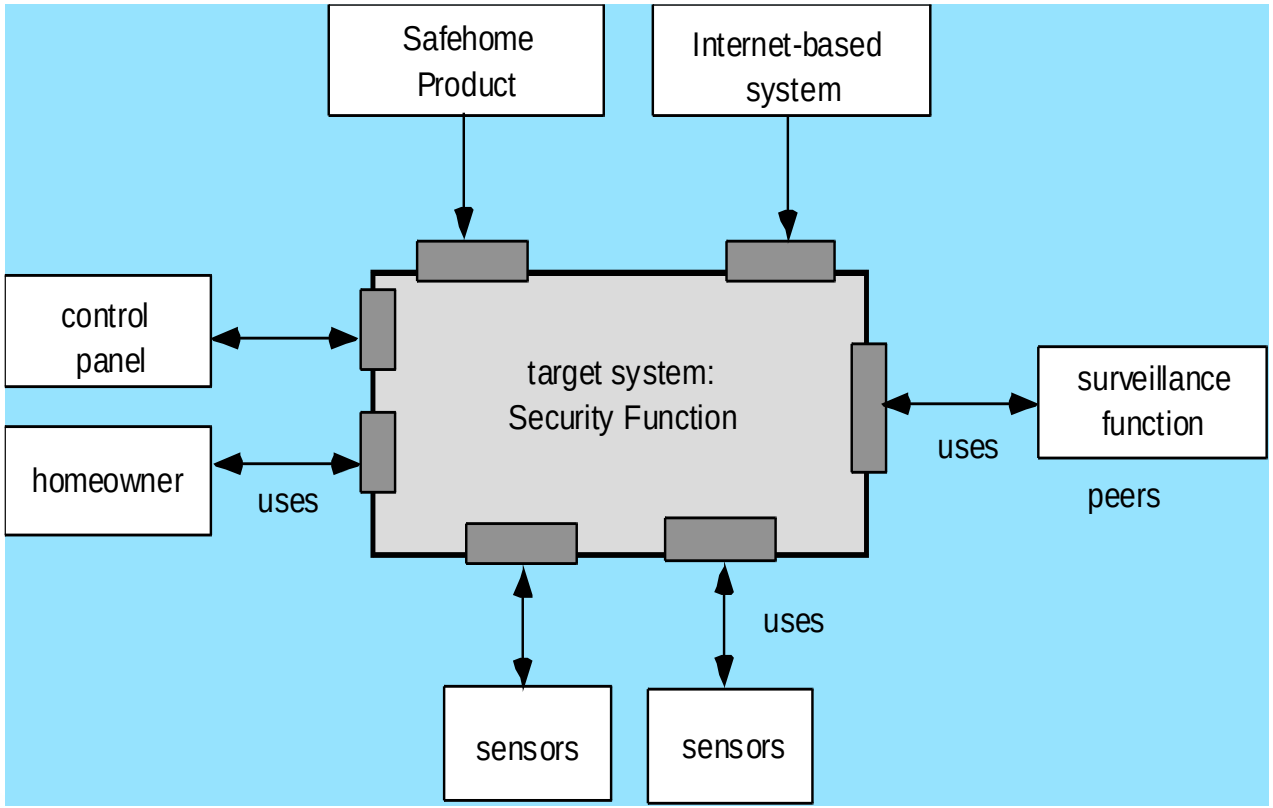


- **Architectural Patterns**
  - *Concurrency*—applications must handle multiple tasks in a manner that simulates parallelism
    - ***operating system process management*** pattern
    - ***task scheduler*** pattern
  - *Persistence*—Data persists if it survives past the execution of the process that 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 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**

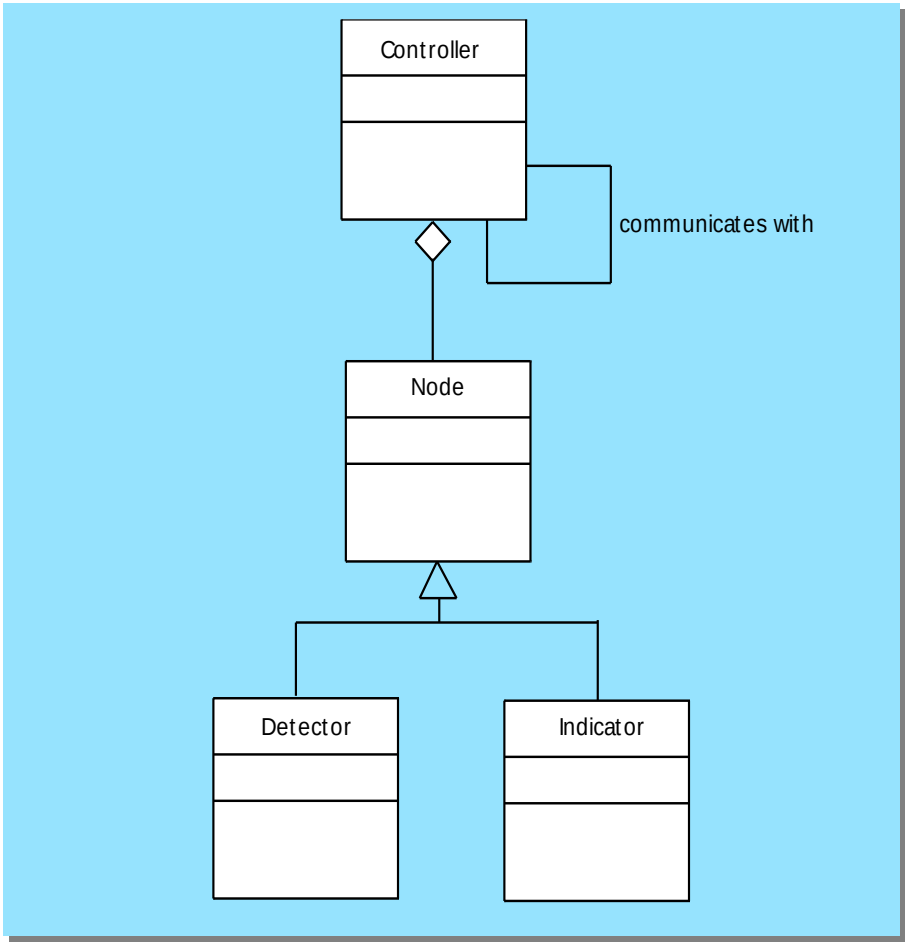
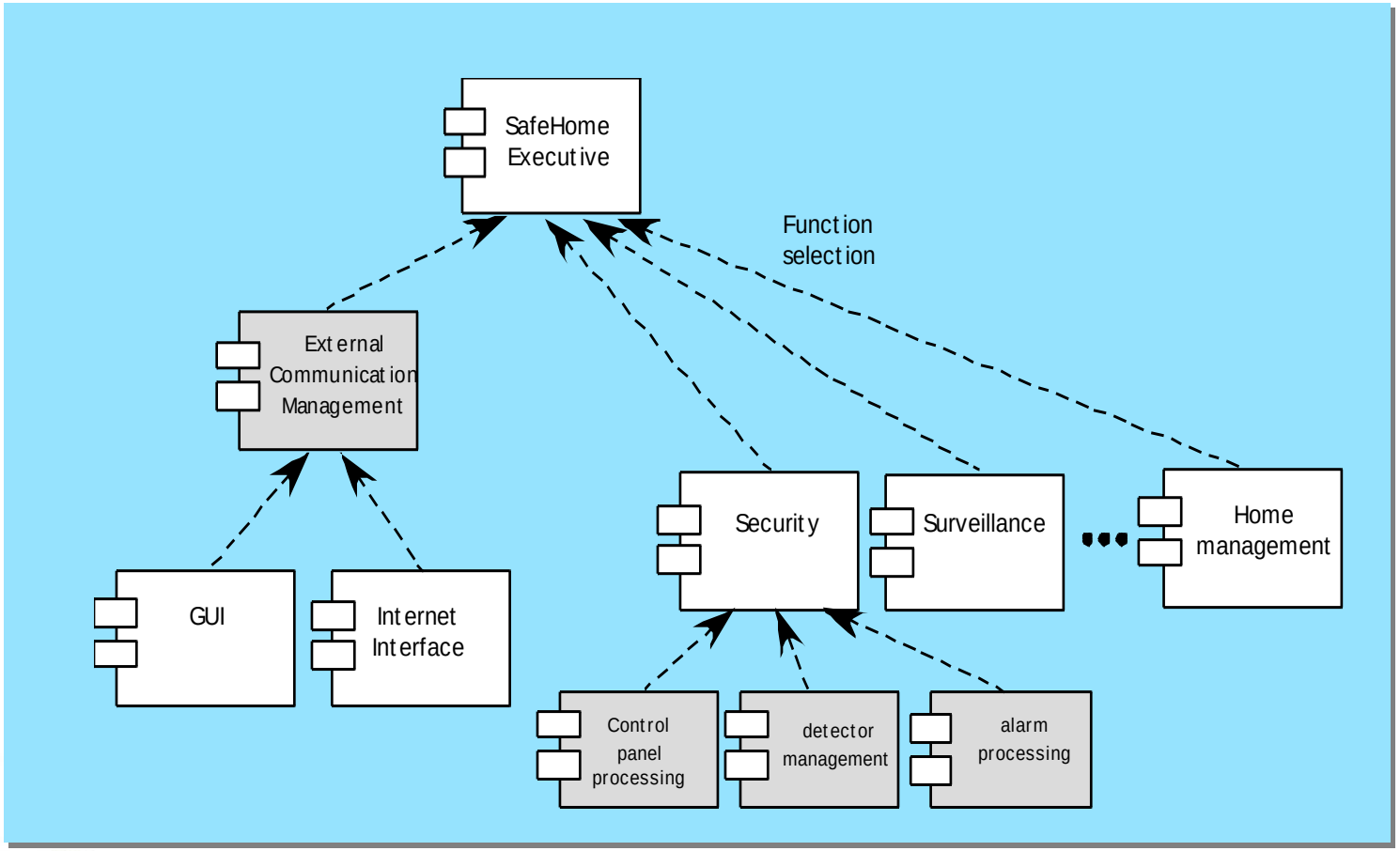
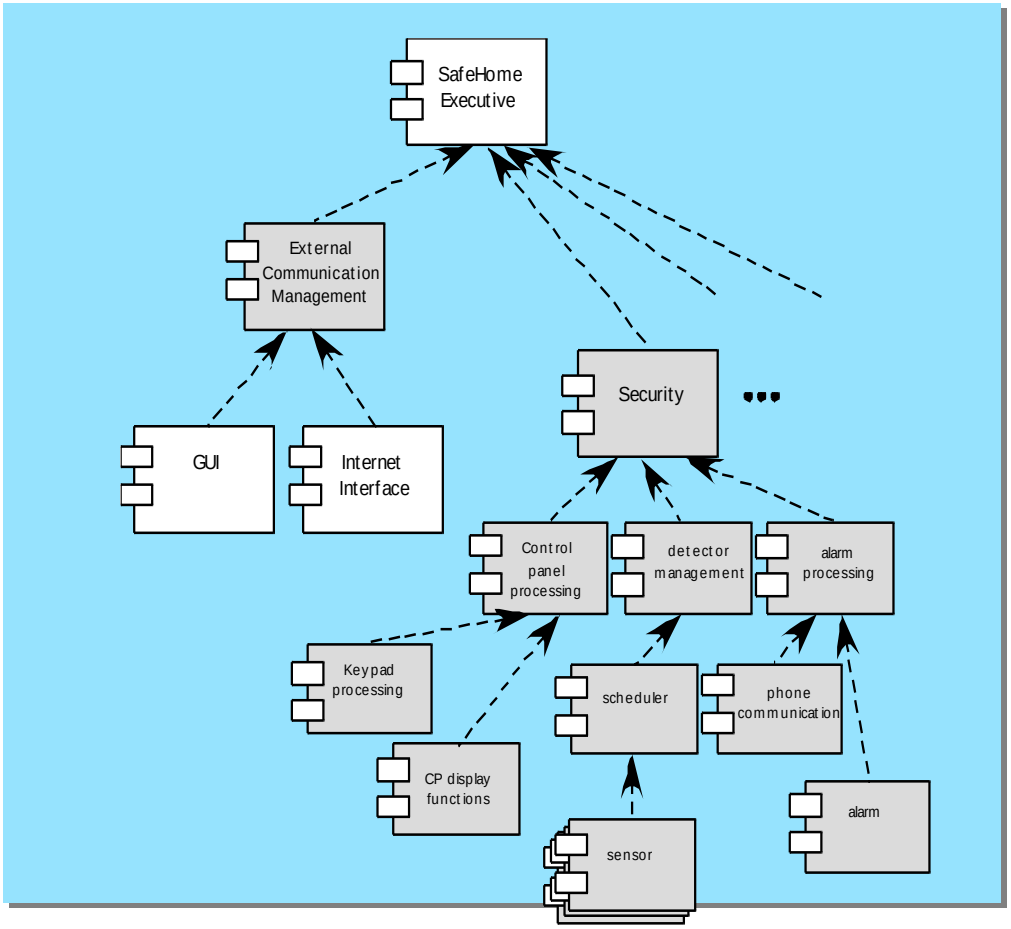


Figure 10.7 UML relationships for SafeHome security function archetypes (adapted from [BOS00])

- **Component Structure**



- **Refined Component Structure**



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

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

- **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.
    - ***Flow dependencies*** represent dependence relationships between producers and consumers of resources.
    - ***Constrained dependencies*** represent constraints on the relative flow of control among a set of activities.

- **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.

- **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

- **Patter-Based Architecture Review**
  - Identify and discuss the quality attributes by walking through the use cases.
  - Discuss a diagram of system's architecture in relation to its requirements.
  - Identify the architecture patterns used and match the system's structure to the patterns' structure.
  - Use existing documentation and use cases to determine each pattern's effect on quality attributes.
  - Identify all quality issues raised by architecture patterns used in the design.
  - Develop a short summary of issues uncovered during the meeting and make revisions to the walking skeleton.

- **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