Week 6

CS2212

Introduction to Software Engineering

Architectural Design

Some images and content from Software Engineering: A Practitioner's Approach 9th Edition By Roger Pressman and Bruce Maxim

What is Architecture?

Let’s say you want to build a building….

Iowa State University Student Innovation Center

Before we start engineering and specifying exactly how each room will be built and the materials used we need an overall plan and a unified vision.

Software Architecture is No Different

Before we start designing the individual components of our software we need an overall plan or blueprint to make sure it all comes together correctly.

What is Software Architecture?

- The architecture of a software system is a **comprehensive framework** that describes its form and structure – its **components** and how they fit together.

- Representation that allow developers to:

  1. Analyze the effectiveness of the design.
  2. Consider architectural alternatives.
  3. Reduce risk.

What is a Component?

Software Component:

- *“A **modular**, **deployable**, and **replaceable** part of the system that **encapsulates implementation** and exposes a set of **interfaces**.”*
- Parts of a system that **break the complexity into manageable parts**.
- **Hides (encapsulates) implementation details** behind an **interface**.
- Components can be **swapped in and out** so long as they share a common **interface**.

What is a Component?

Software Component Views:

- **Object-Oriented View:** A set of one or more collaborating classes.
- **Traditional View:** A functional element of a program (aka a module).
- **Process-Related View:** A pre-existing prepackaged component or design pattern.

Why is Architecture Important?

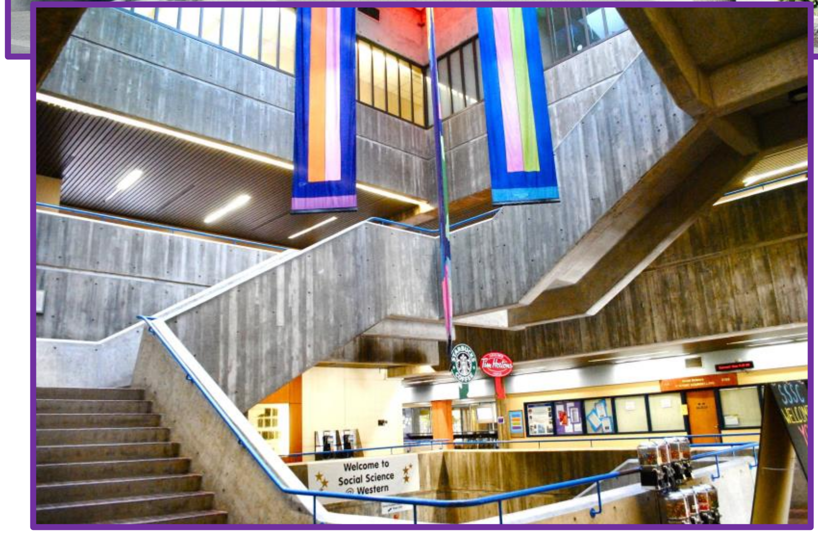
1. Provides **representations** that **facilitate communication** with stakeholders.

2. Highlights early **design decisions with profound impact** on all software engineering work that follows.

3. Gives an ***"intellectually graspable"*** model of the system and how its components work together.

Architectural Styles

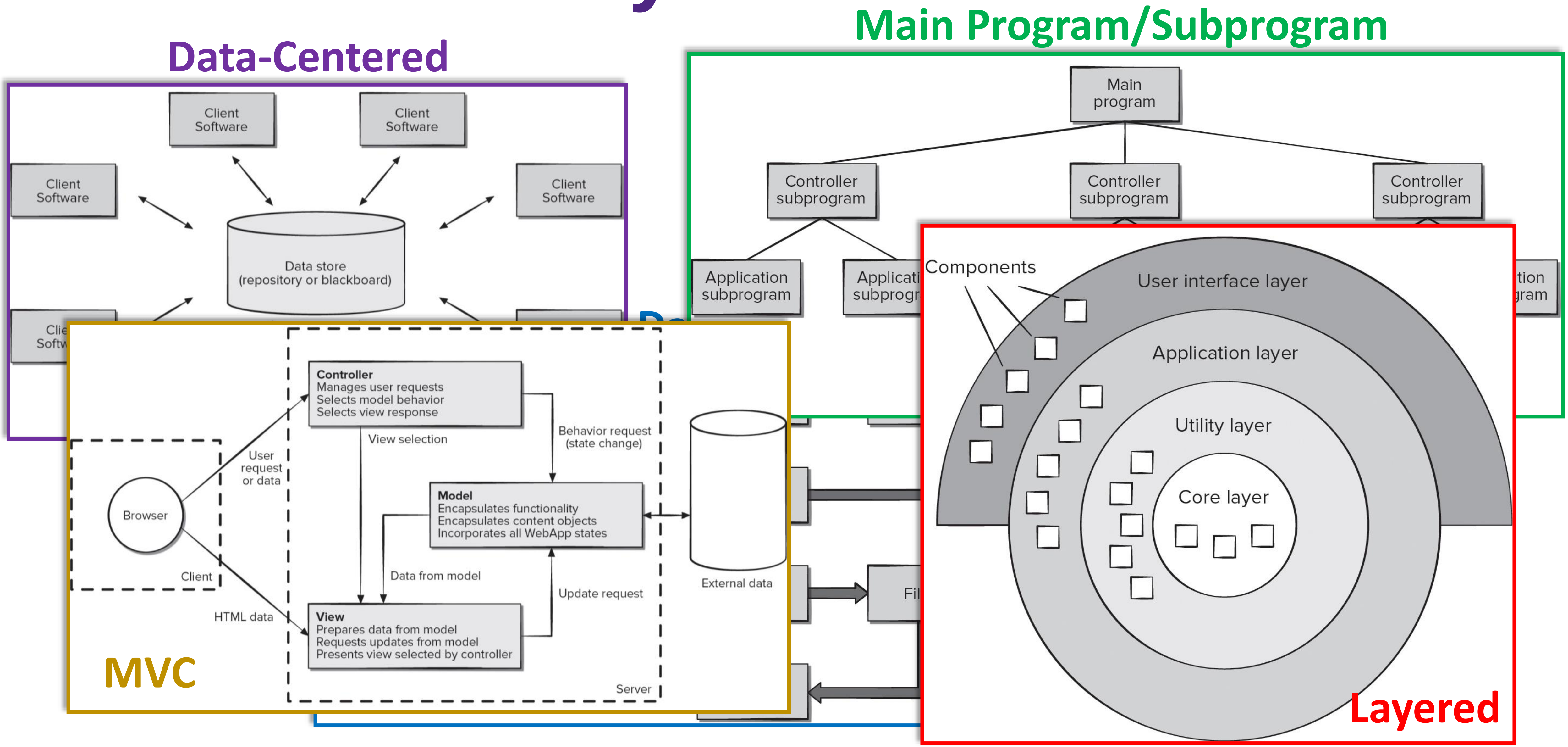
Brutalism



Modern

Collegiate Gothic

Architectural Styles



Architectural Styles

Each style describes a system category that encompasses:

1. **Set of Components:** 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.
4. **Semantic Models:** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

Architectural Styles

Common Styles:

- Data-Centered
- Data-Flow
- Call-and-Return
- Object-Oriented
- Layered
- Model-View-Controller

Architectural Styles

Common Styles:

- **Data-Centered**

Data-Centered

A data store resides at the center of the architecture, accessed frequently by other components that update, add, delete, or otherwise modify data in the store.

- Model-View-Controller

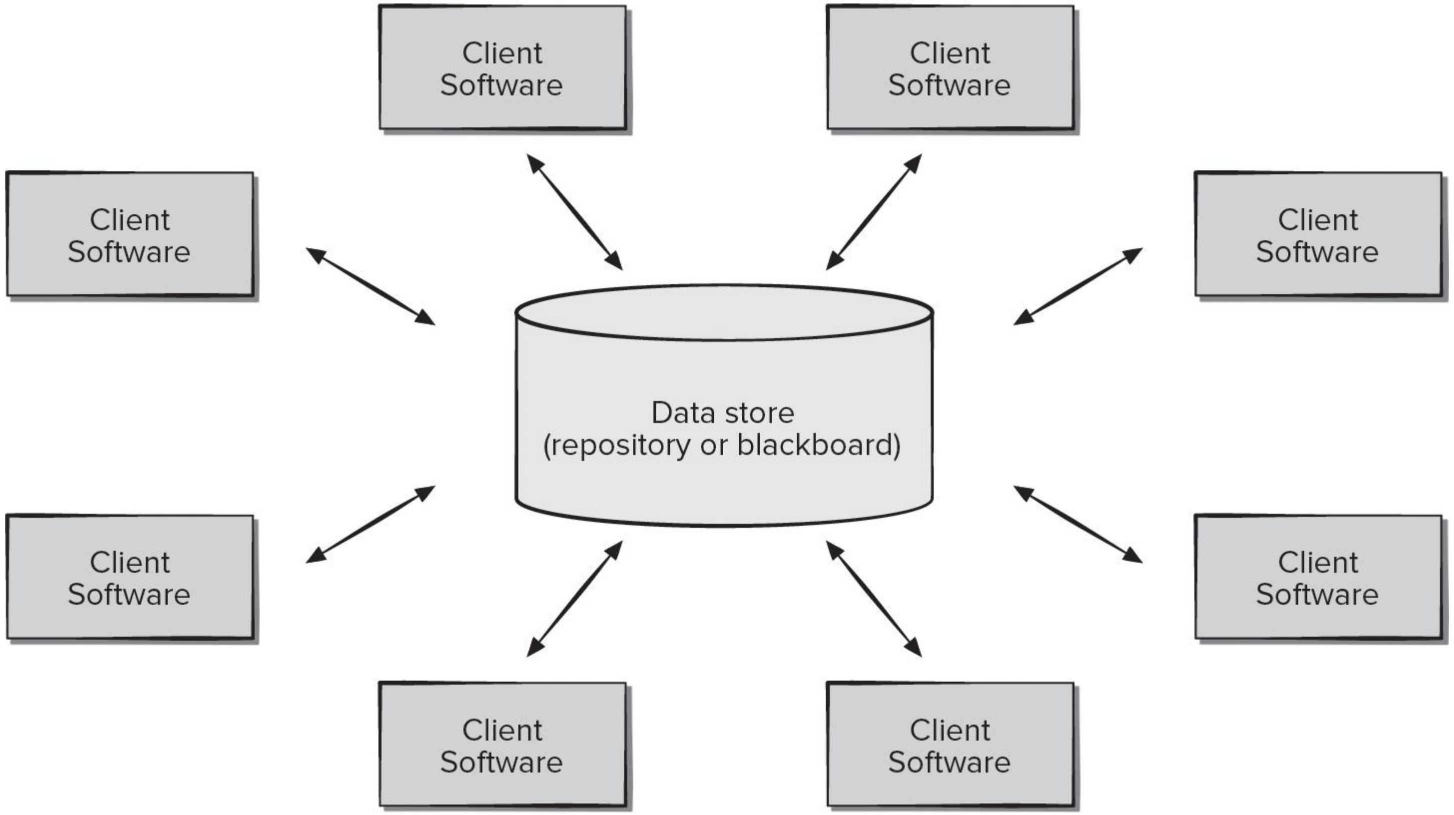


Fig. 10.1 from textbook

Architectural Styles

Common Styles:

- Data-Centered
- **Data-Flow**

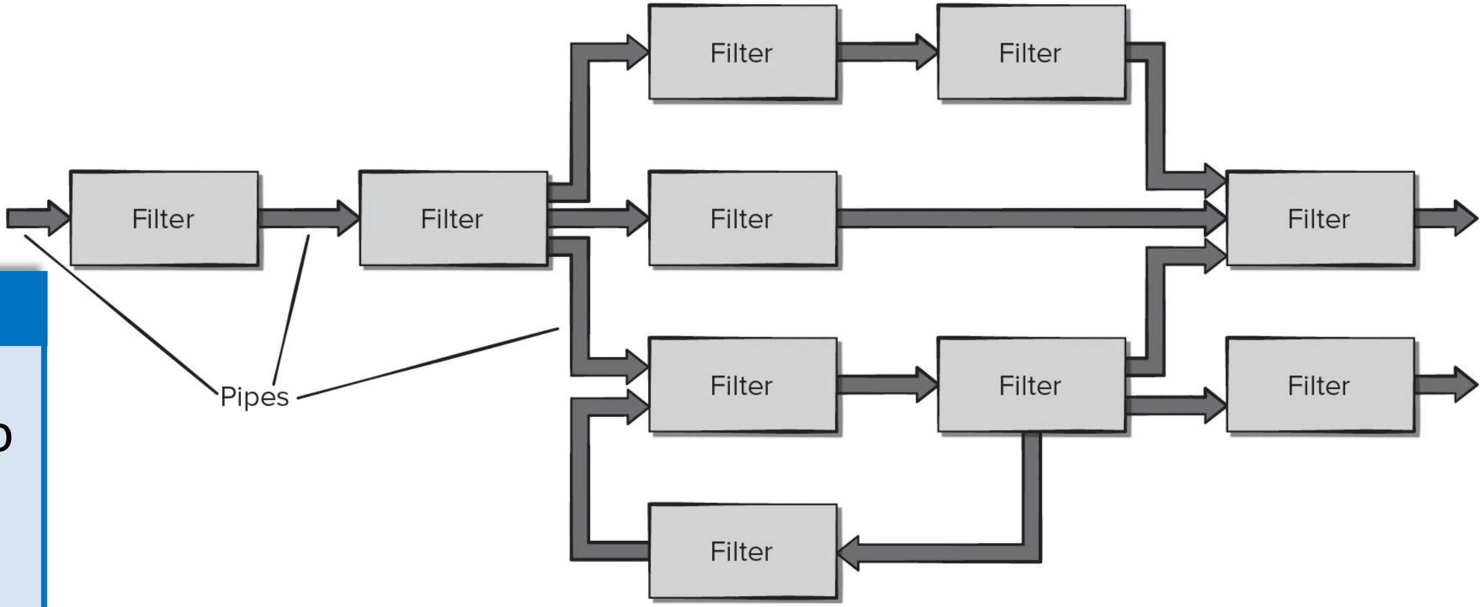


Fig. 10.2 from textbook

Data-Flow

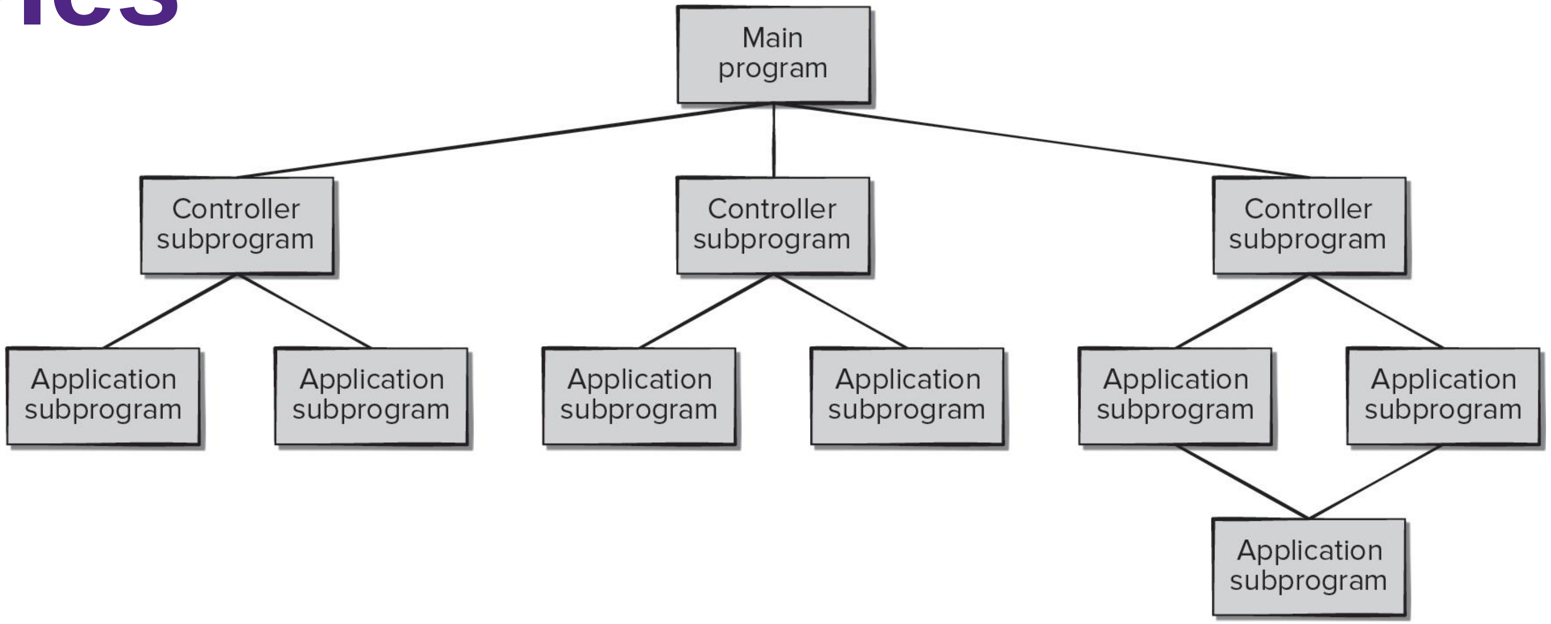
- Input data is transformed through a series of computational components into output data.
- Pipe-and-filter pattern uses filters (computational components) connected by pipes that transmit the data between components.
- Filters do not require knowledge of each other.

Architectural Styles

Common Styles:

- Data-Centered
- Data-Flow
- **Call-and-Return**
- Object-Oriented
- Layered
- Model-View-Controller

Fig. 10.3 from textbook



Call-and-Return

- **Two substyles:**
  - **Main Program/Subprogram:** a main program invokes a number of program components, which in turn may invoke still other components.
  - **Remote Procedure Call:** Components are distributed across multiple networked computers.

Architectural Styles

Common Styles:

- Data-Centered
- Data-Flow
- Call-and-Return
- **Object-Oriented**
- Layered
- Model-View-Controller

Object-Oriented

- 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 (method invocation).

Architectural Styles

Layered

- Layers are defined, each providing services to layers above through operations completed within the layer or by leveraging services from lower layers.
- Outer layers interface more directly with the user, inner-most layers interface with the operating system and/or underlying hardware.
- Intermediate layers provide utility services and other application functions.

- **Layered**
- Model-View-Controller

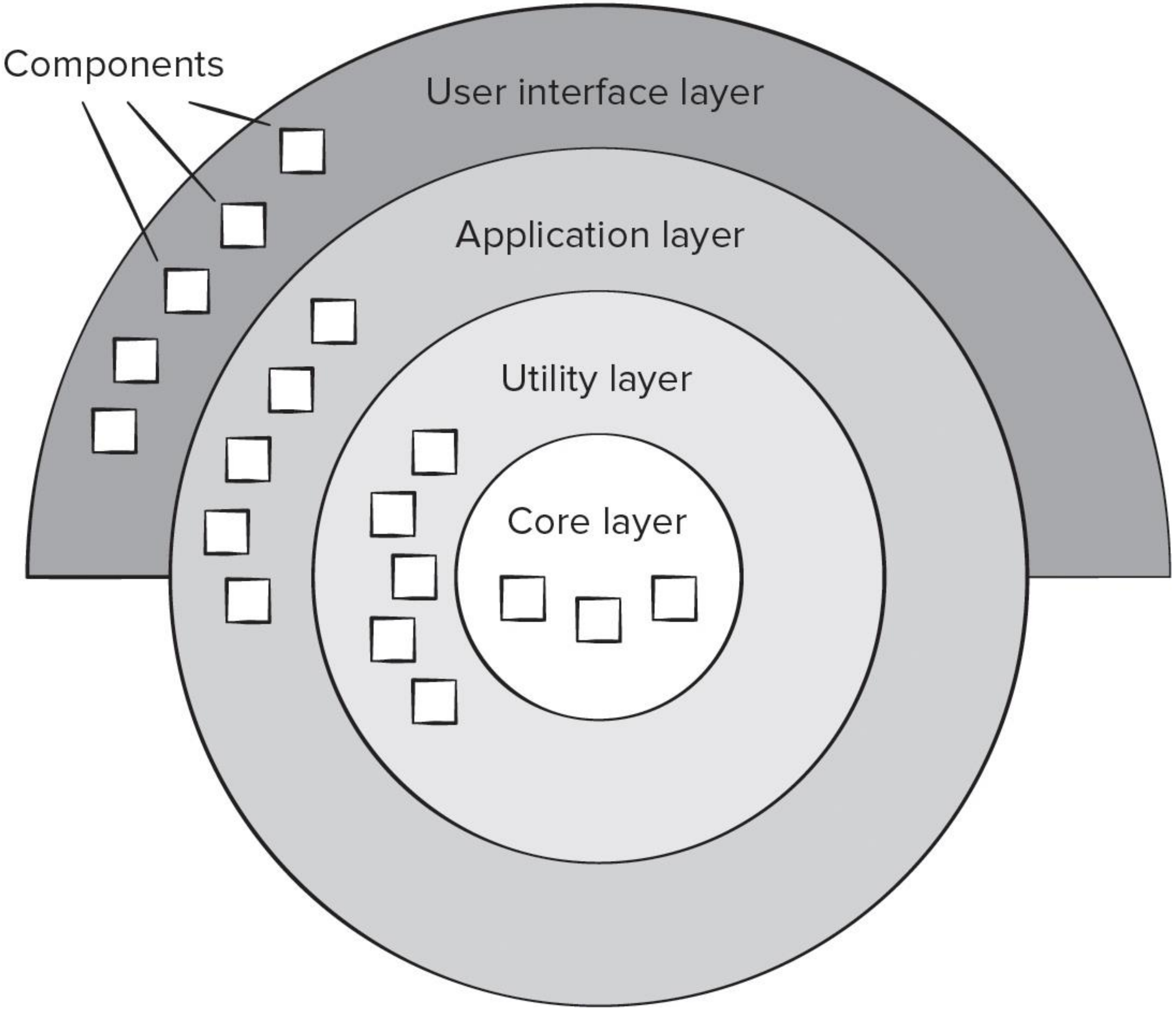


Fig. 10.5 from textbook

Architectural Styles

Common Styles:

Model-View-Controller (MVC)

- Often used in web and mobile development.
- **Comprised of three kinds of components:**
  - **Model:** contains all application-specific content and processing logic.
  - **View:** contains all the interface-specific functions and handles presentation of content to the end user.
  - **Controller:** Manages access to the model and view, coordinates flow of data.

- Model-View-Controller

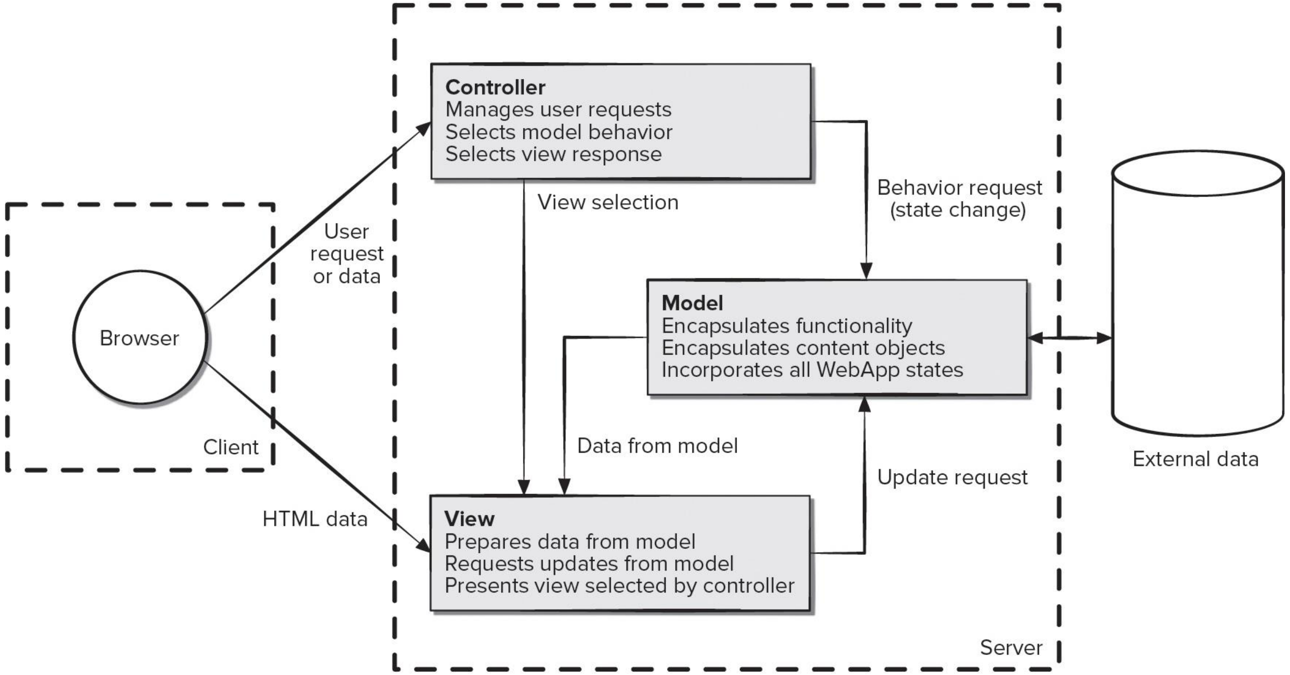


Fig. 10.6 from textbook

Architectural Styles

- **Many** more styles exist, this is just a small selection of some common ones.
- You can also combine multiple styles together.

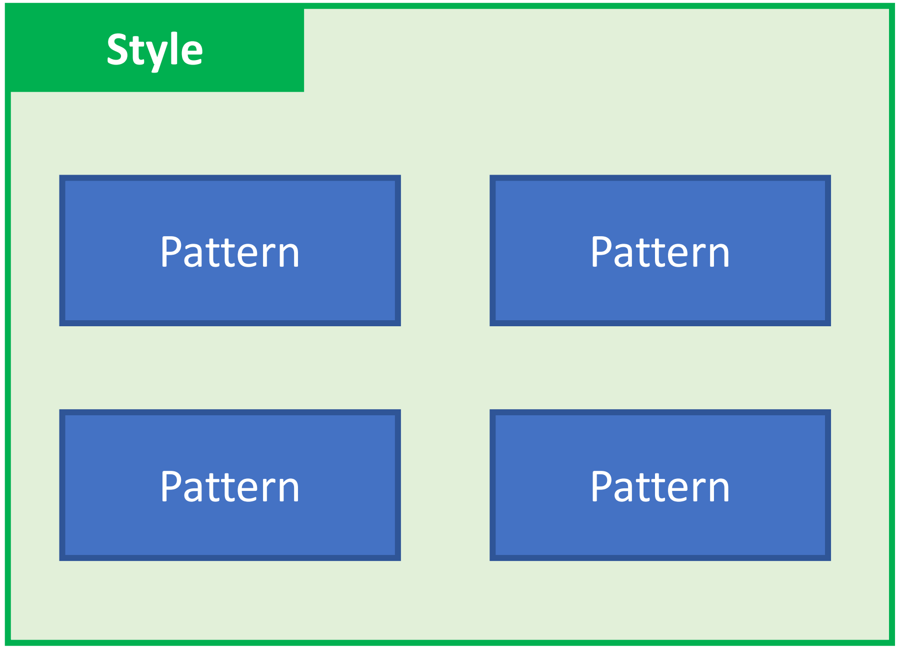
Architectural Patterns

- **Architectural patterns** address an **application-specific problem** within a specific context and under a set of limitations and constraints.

- The **pattern** **proposes an architectural solution** that can serve as the basis for the architectural design.

- **Patterns** can be **used in conjunction with an architectural style** to shape the overall structure of a system.

- The **style** typically **influences the architecture in its entirety**, while **patterns** tend to **focus on one aspect** of the architecture at a time



Architectural Considerations

Several considerations when choosing an architecture:

- **Economy:** Software is **uncluttered** and **relies on abstraction** to reduce unnecessary detail.
- **Visibility**: Architectural decisions and their justifications **should be obvious** to software engineers who review.
- **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 behaviour and control** are key to creating scalable, efficient, and economic software architectures.

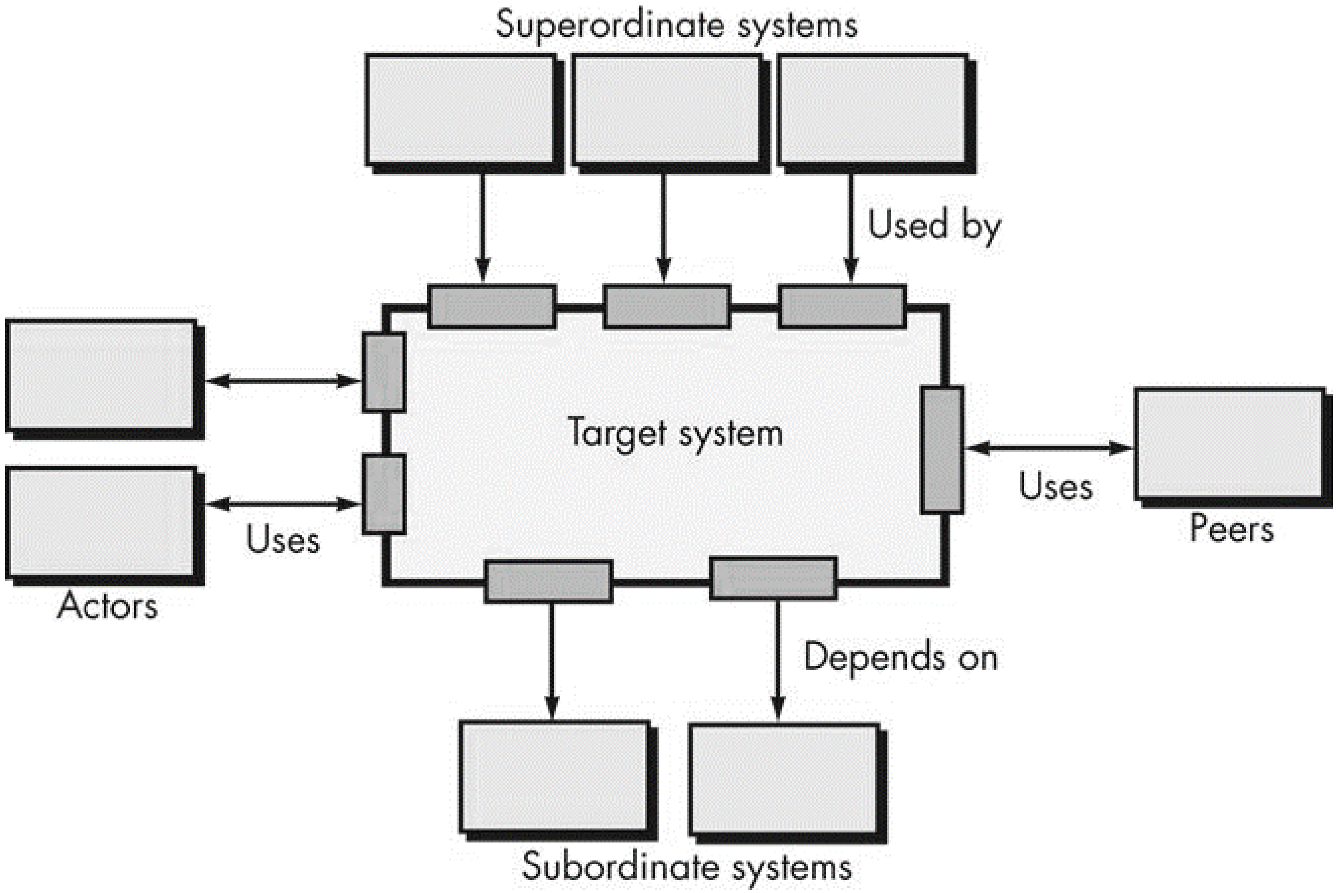
Architectural Design

- As **architectural design** begins, 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 their interactions**.
- A set of **architectural archetypes** should be identified.
    - An **archetype** is an **abstraction** (similar to a class) that represents **one element of system behaviour**.
- The designer specifies the **structure of the system** by defining and refining software components that implement each archetype.

Architectural Context

How do we represent architectural context?

- UML does not contain specific diagrams for system context.
  - Can use combination of use cases, class, component, activity, sequence, and collaboration diagrams.
- Or can make use of **Architectural Context Diagram (ACD)**.



Architectural Context

How do we represent architectural context?

- UML does not contain specific diagrams for system context.
  - Can use combination of use cases, class, component, activity, sequence, and collaboration diagrams.
- Or can make use of **Architectural Context Diagram (ACD)**.

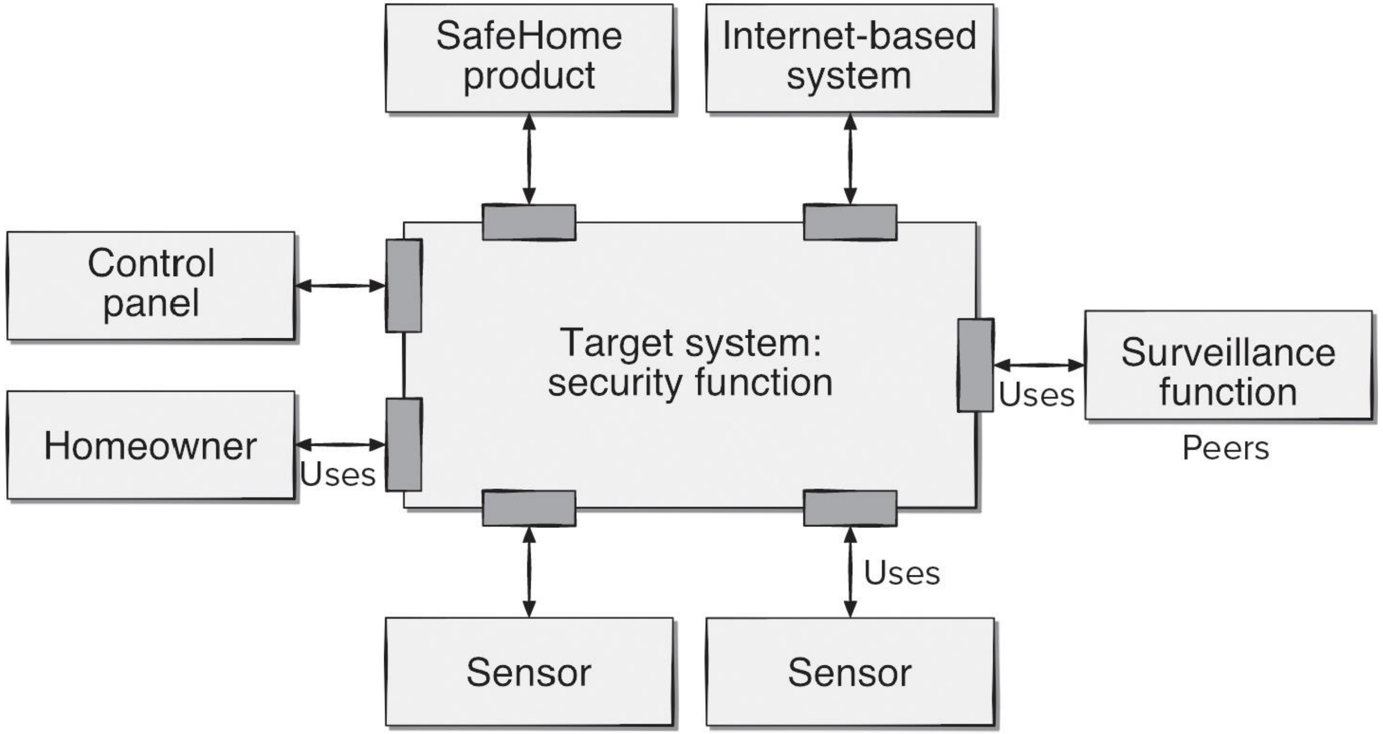


Fig. 10.7 from textbook
***ACD** for SafeHome security function*

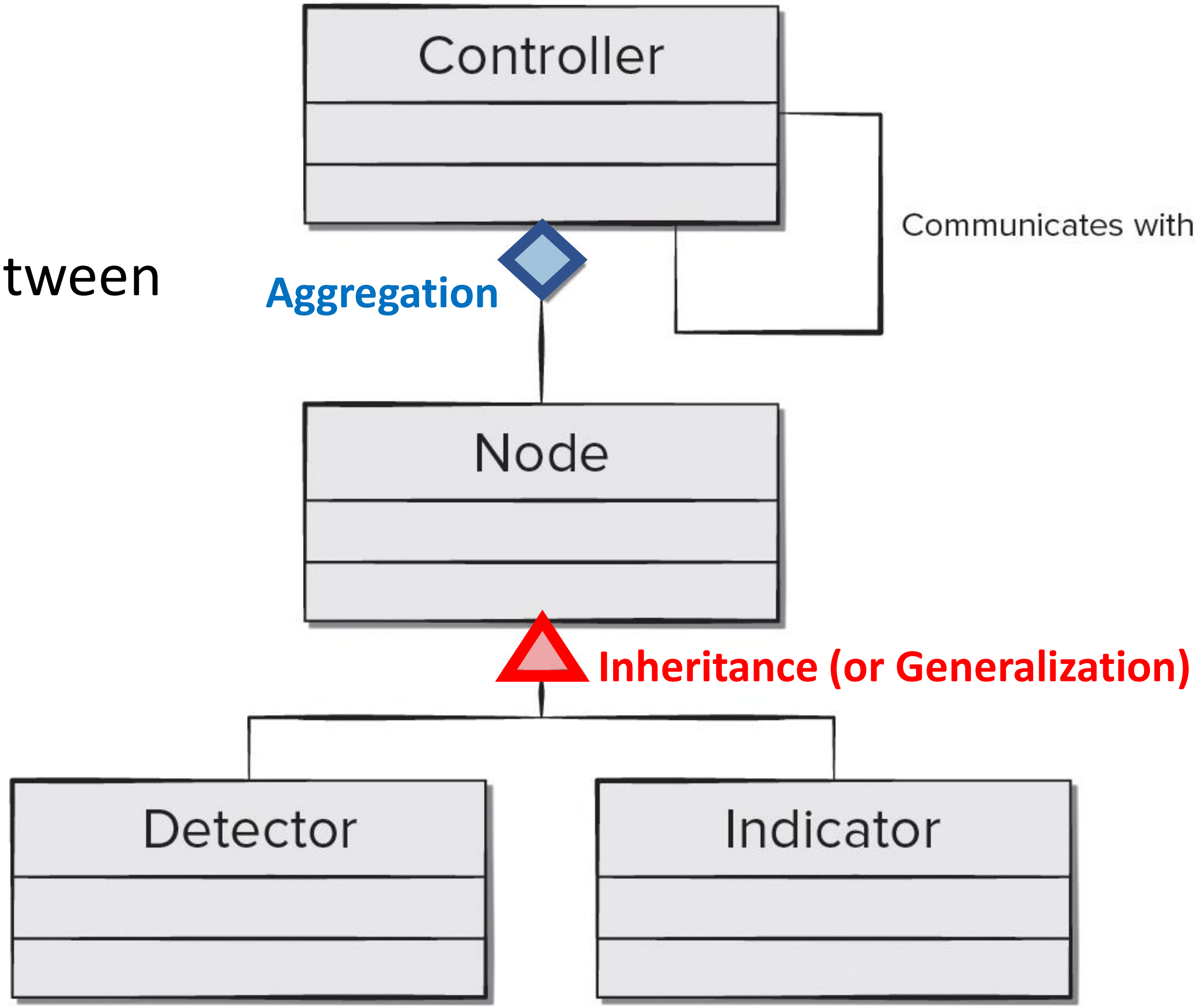
Defining Archetypes

- An archetype is a **class** or **pattern** that represents a **core abstraction** that is critical to the design of an architecture for the target system.
- Generally, a relatively **small set of** archetypes **is required** to design even relatively complex systems.
- The system architecture is **composed of** archetypes, which represent stable elements of the architecture. These may **be instantiated in different ways** based on the behaviour of the system.
- In many cases, archetypes **can be derived by examining the analysis classes** defined as part of the **requirements model**.

Archetype Example

Archetypes for the *SafeHome* Security Function

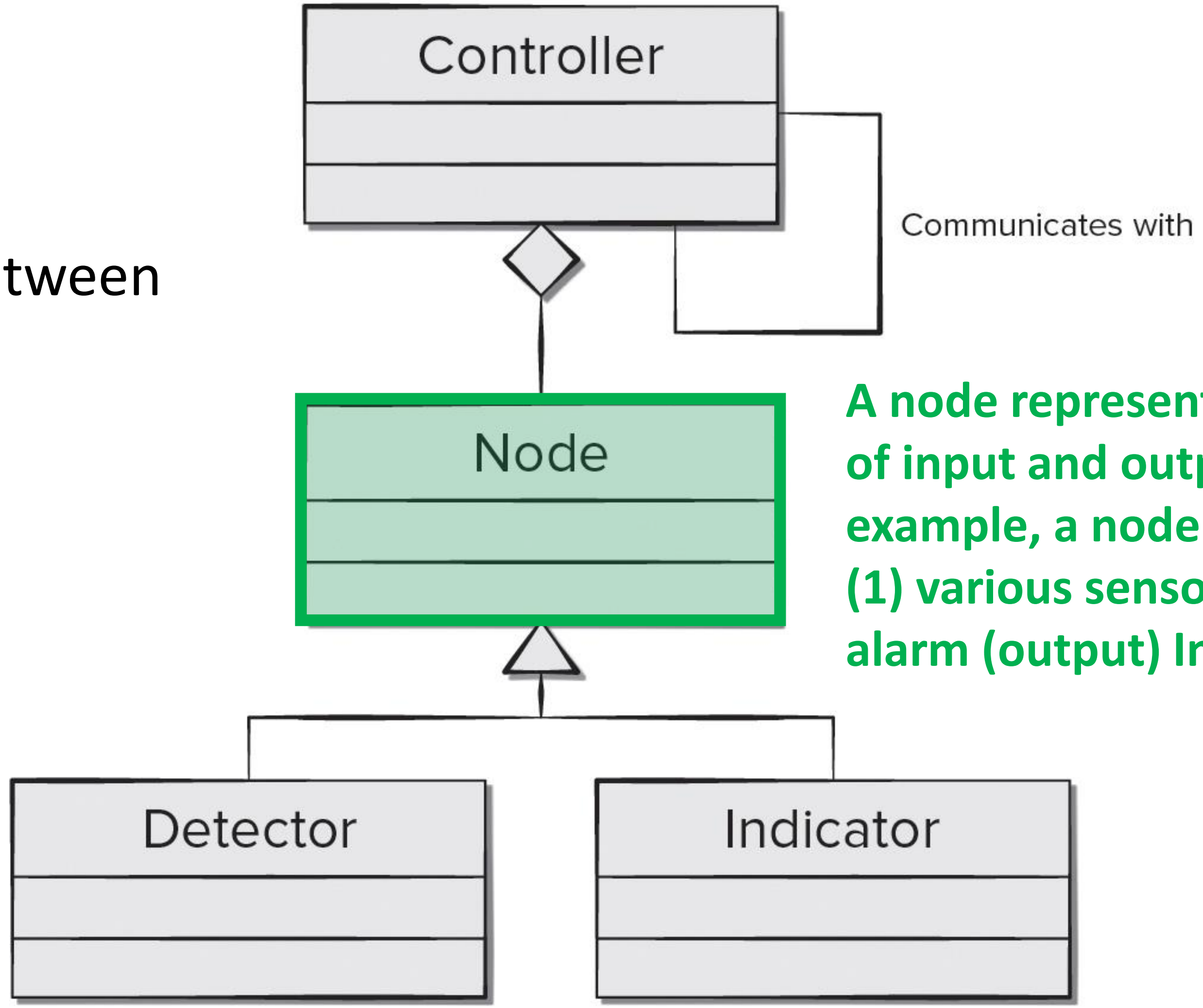
- UML notation.
- Type of class diagram.
- Shows relationships between *SafeHome* archetypes.



Archetype Example

Archetypes for the *SafeHome* Security Function

- UML notation.
- Type of class diagram.
- Shows relationships between *SafeHome* archetypes.

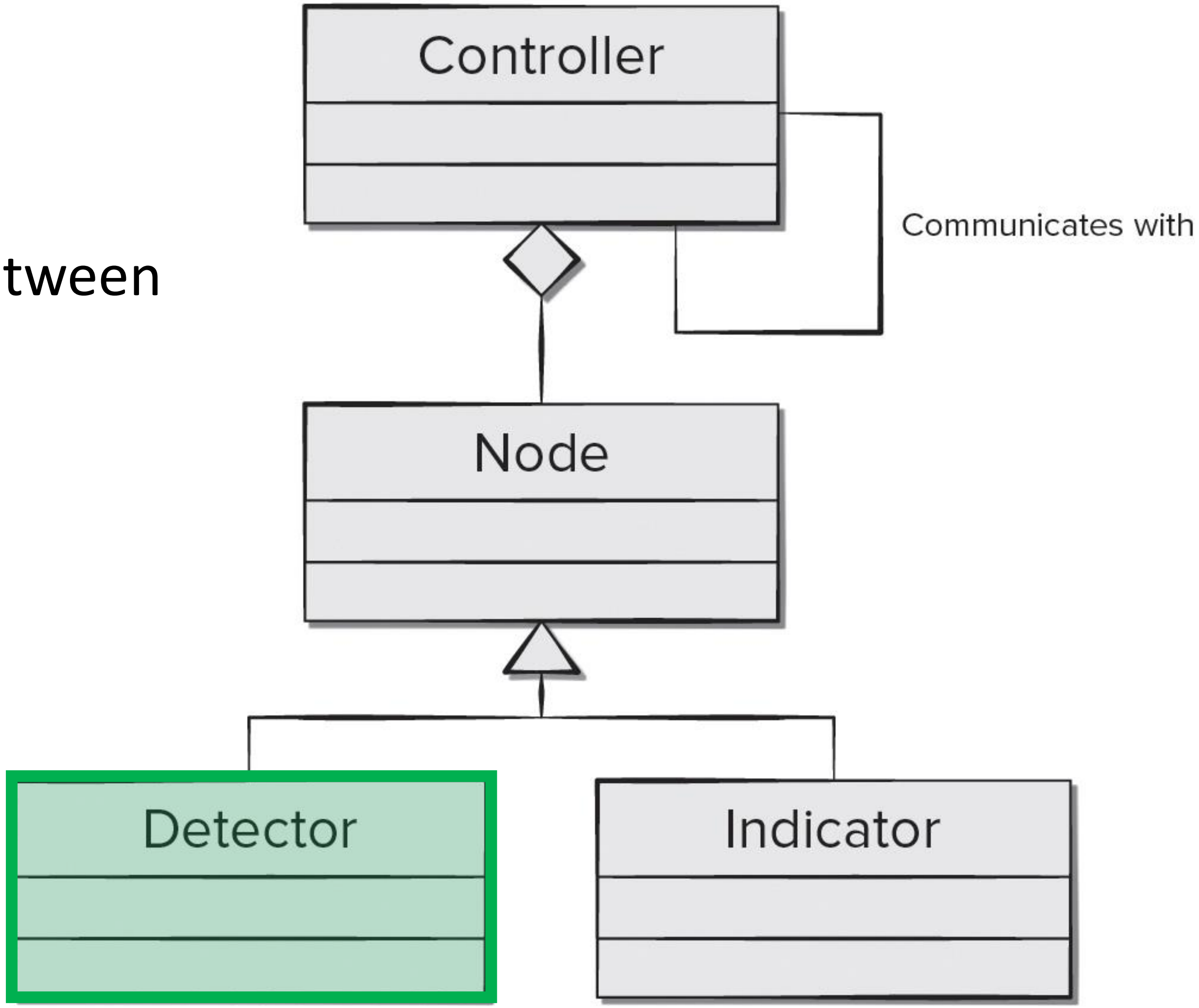


A node represents a cohesive collection of input and output elements; for example, a node might be composed of (1) various sensors and (2) a variety of alarm (output) Indicators.

Archetype Example

Archetypes for the *SafeHome* Security Function

- UML notation.
- Type of class diagram.
- Shows relationships between *SafeHome* archetypes.

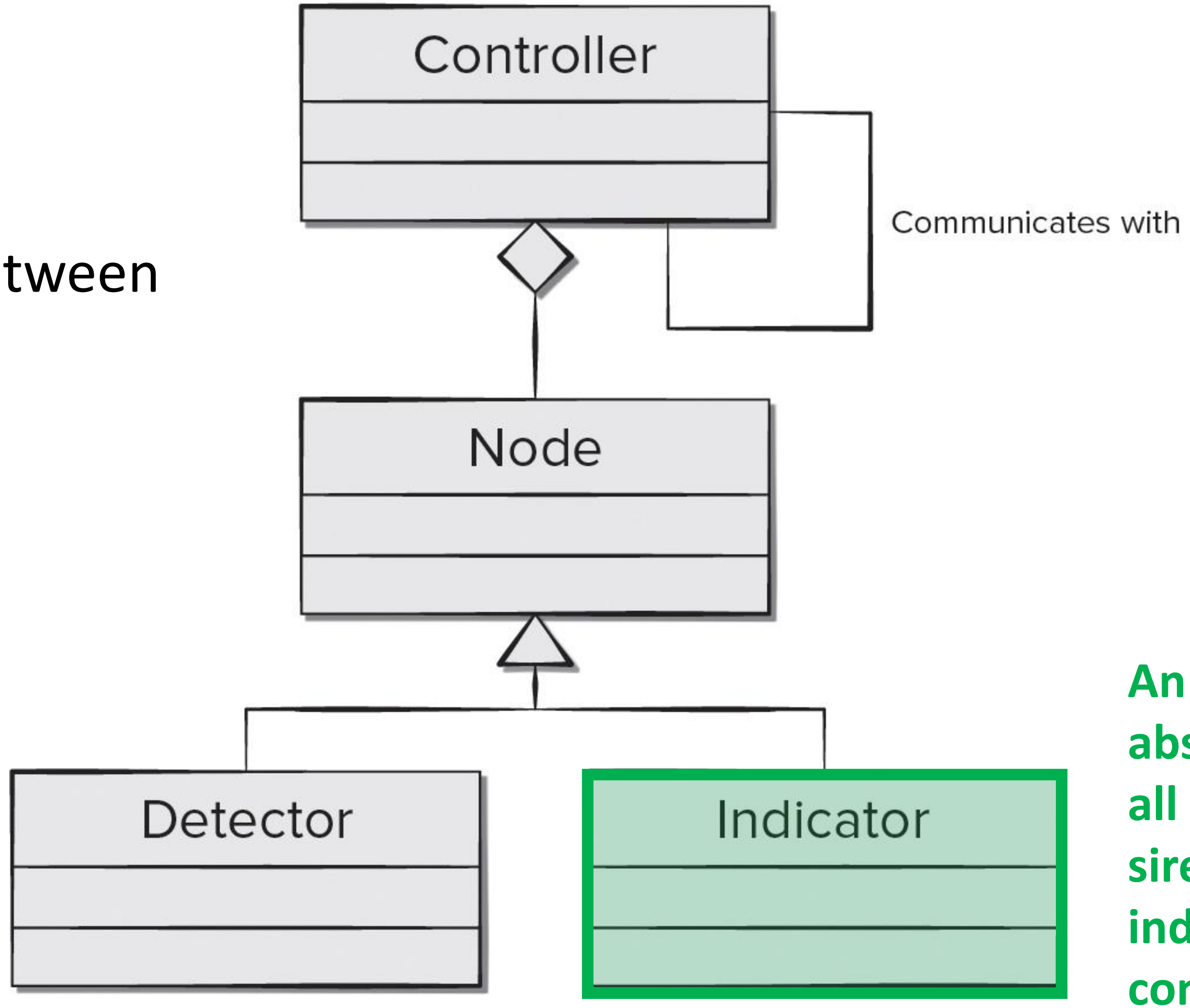


A detector is an abstraction that encompasses all sensing equipment feeding information into the target system.

Archetype Example

Archetypes for the *SafeHome* Security Function

- UML notation.
- Type of class diagram.
- Shows relationships between *SafeHome* archetypes.



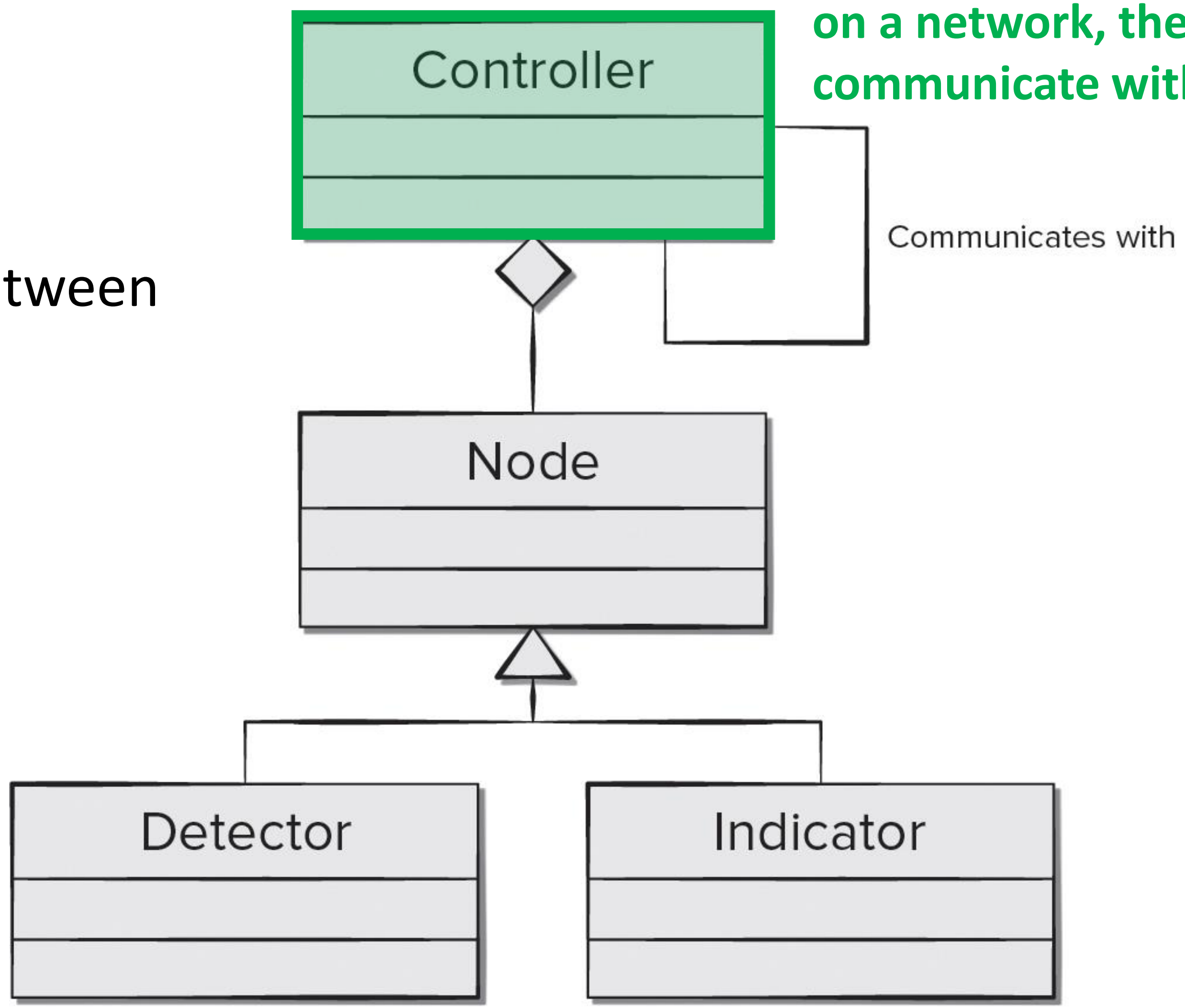
An indicator is an abstraction that represents all mechanisms (alarm sirens, lights, etc.) for indicating that an alarm condition is occurring.

Archetype Example

Archetypes for the *SafeHome* Security Function

- UML notation.
- Type of class diagram.
- Shows relationships between *SafeHome* archetypes.

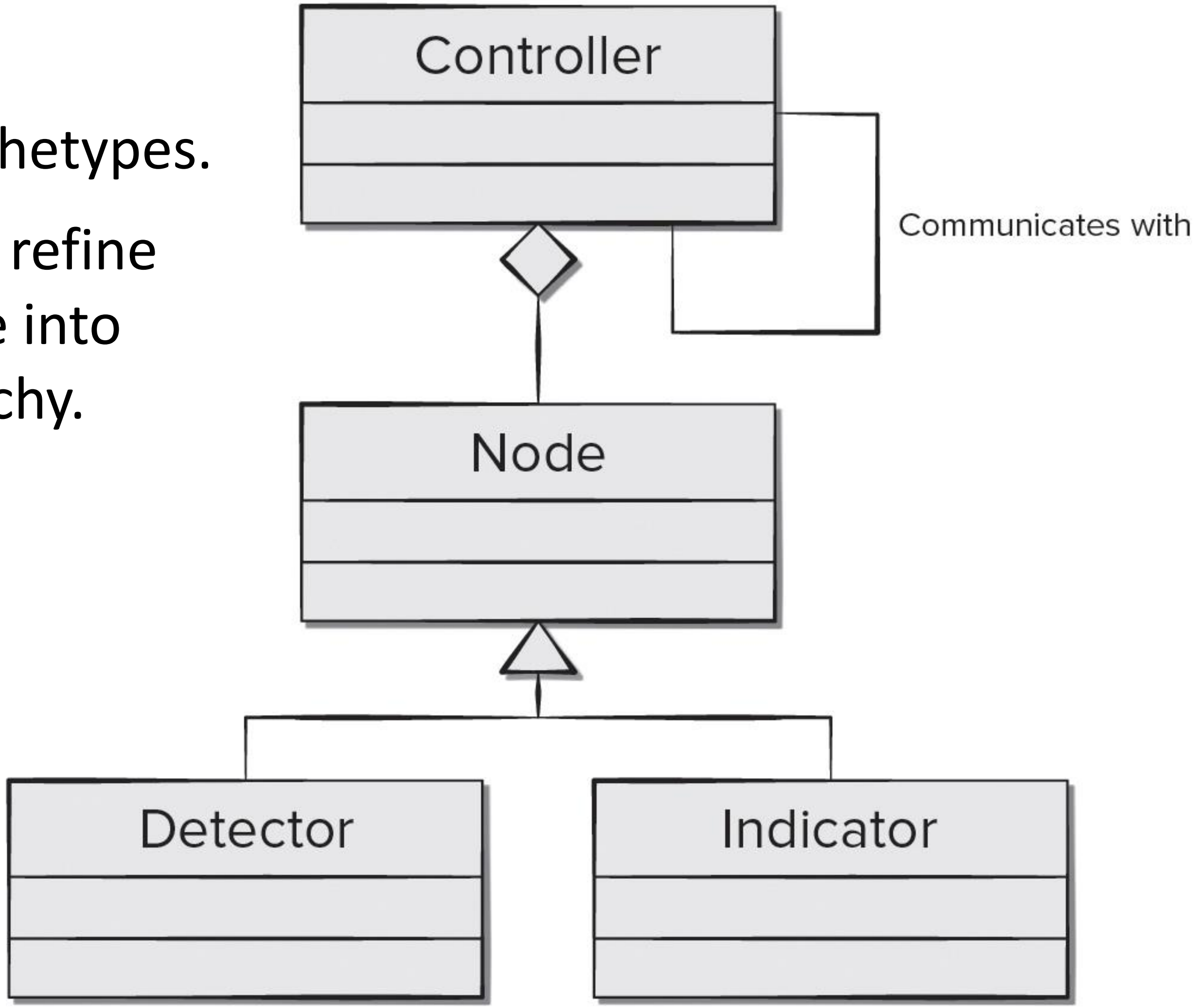
A controller is an abstraction that depicts the mechanism that allows the arming or disarming of a node; if controllers reside on a network, they have the ability to communicate with one another



Archetype Example

Archetypes for the *SafeHome* Security Function

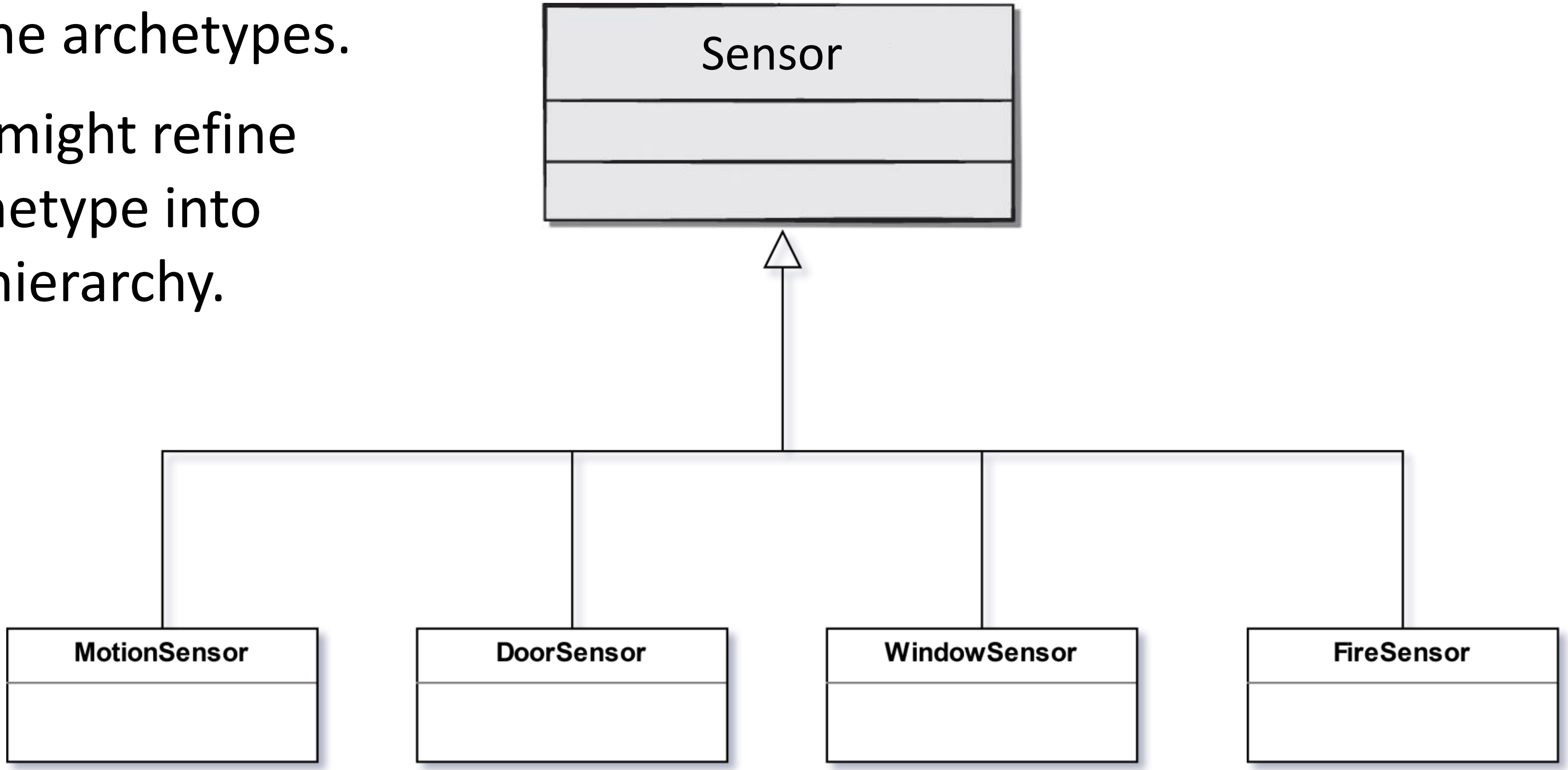
- As architectural design proceeds we refine archetypes.
- For example, we might refine the Detector archetype into the Sensor class hierarchy.



Archetype Example

Archetypes for the *SafeHome* Security Function

- As architectural design proceeds we refine archetypes.
- For example, we might refine the Detector archetype into the Sensor class hierarchy.



Refining into Components

- As the software architecture is **refined into components**, the structure of the system begins to emerge.

- **How are these components chosen?**

  - Begin with the **classes that were described as part of the requirements model.**

  - These classes **represent entities within the application domain** that must be addressed within the software architecture.

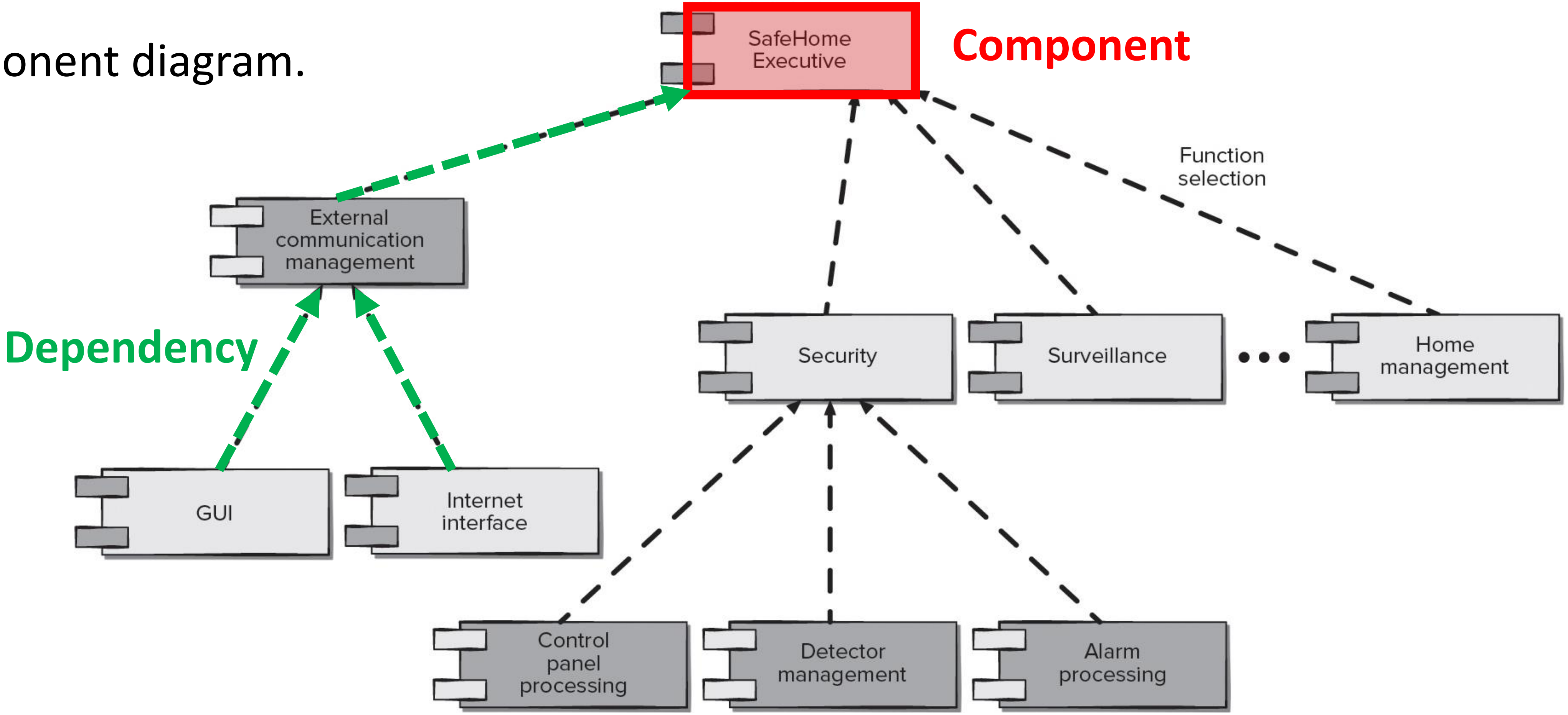
  - From there, **identify infrastructure components** that enable application components but are not part of the application domain *(e.g. database components, communication components, task management components).*

Overall *SafeHome* Architecture

UML Component Diagram

Fig. 10.9 from textbook

Simplified component diagram.



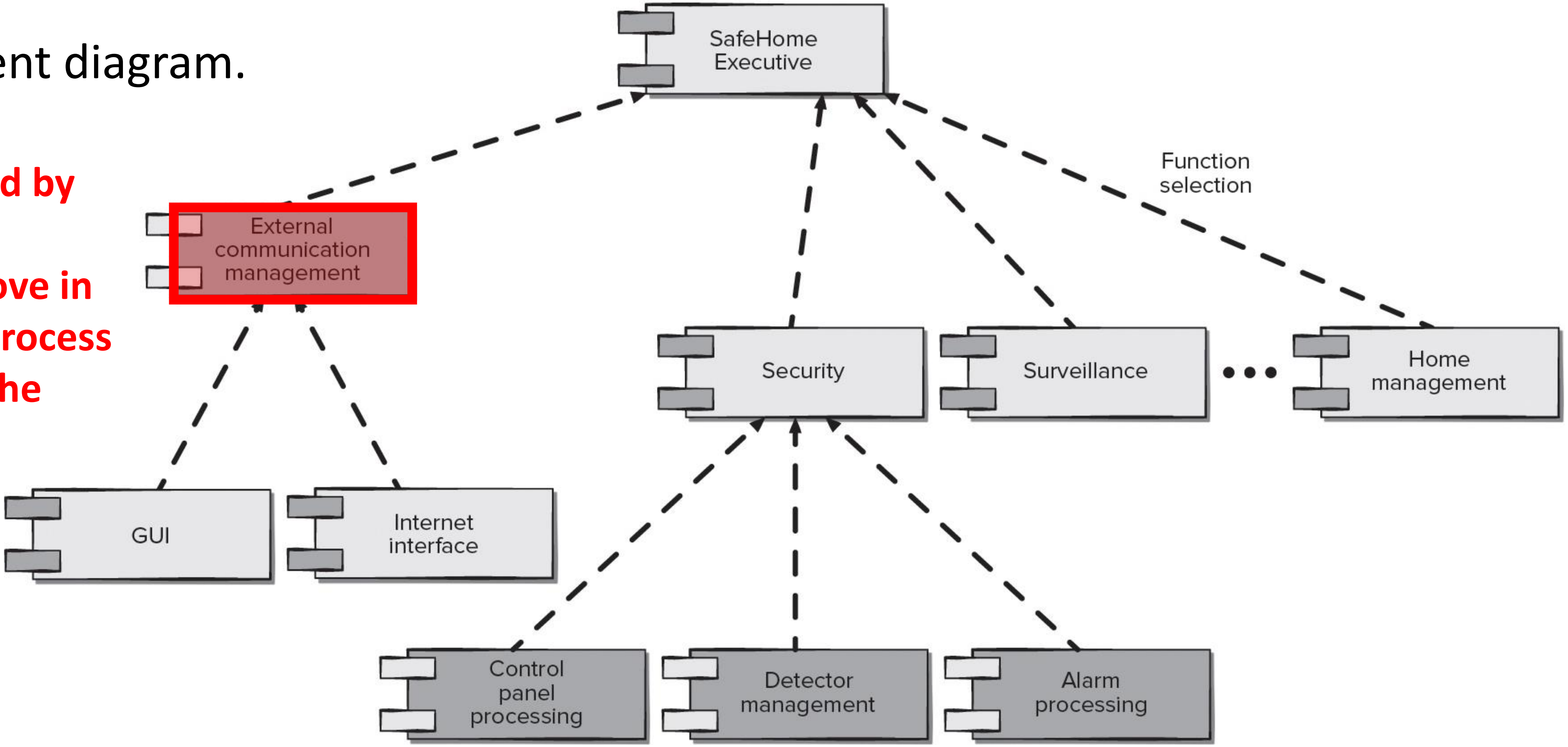
Overall *SafeHome* Architecture

UML Component Diagram

Fig. 10.9 from textbook

Simplified component diagram.

Transactions are acquired by external communication management as they move in from components that process the *SafeHome* GUI and the Internet Interface.



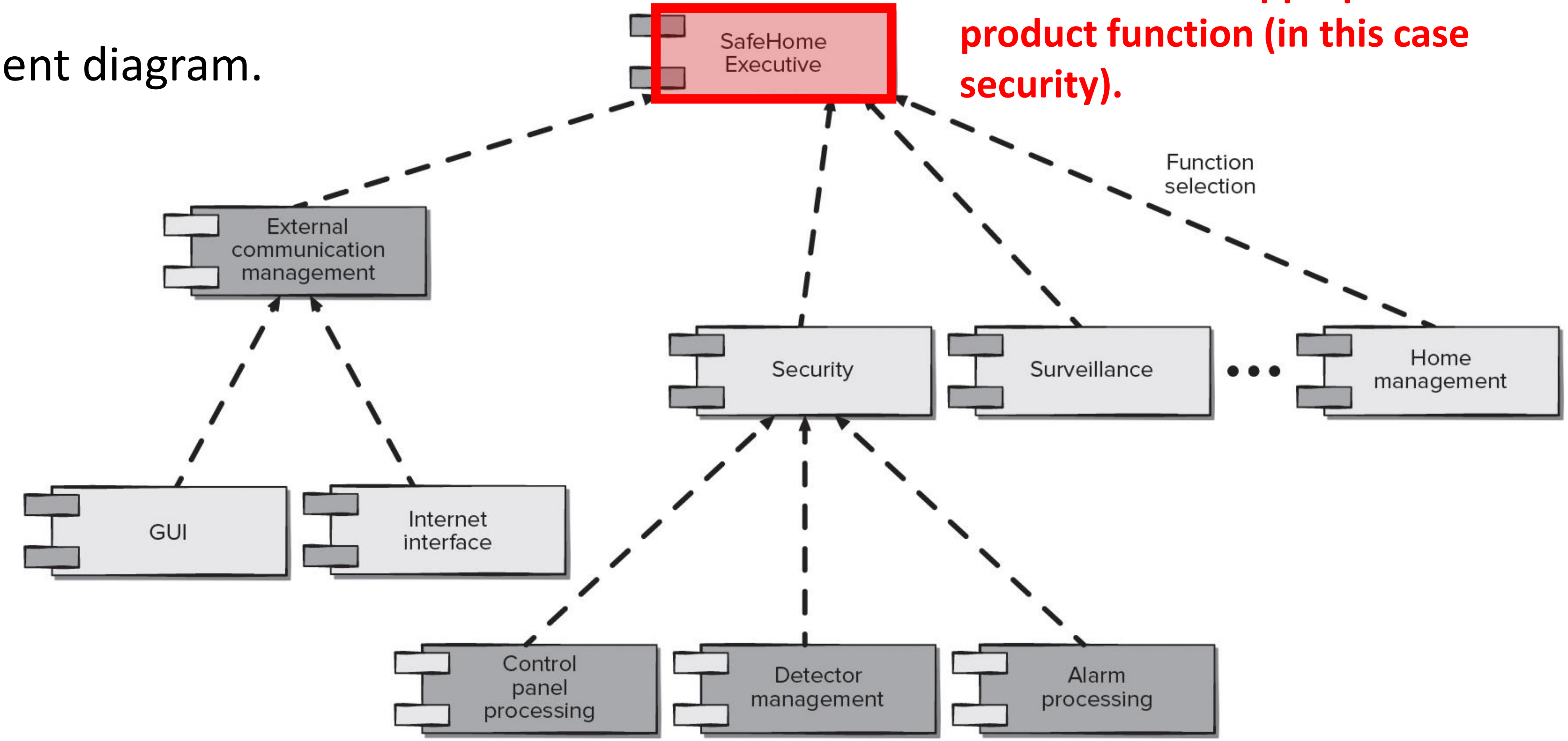
Overall *SafeHome* Architecture

UML Component Diagram

Fig. 10.9 from textbook

Simplified component diagram.

This information is managed by a *SafeHome* executive component that selects the appropriate product function (in this case security).

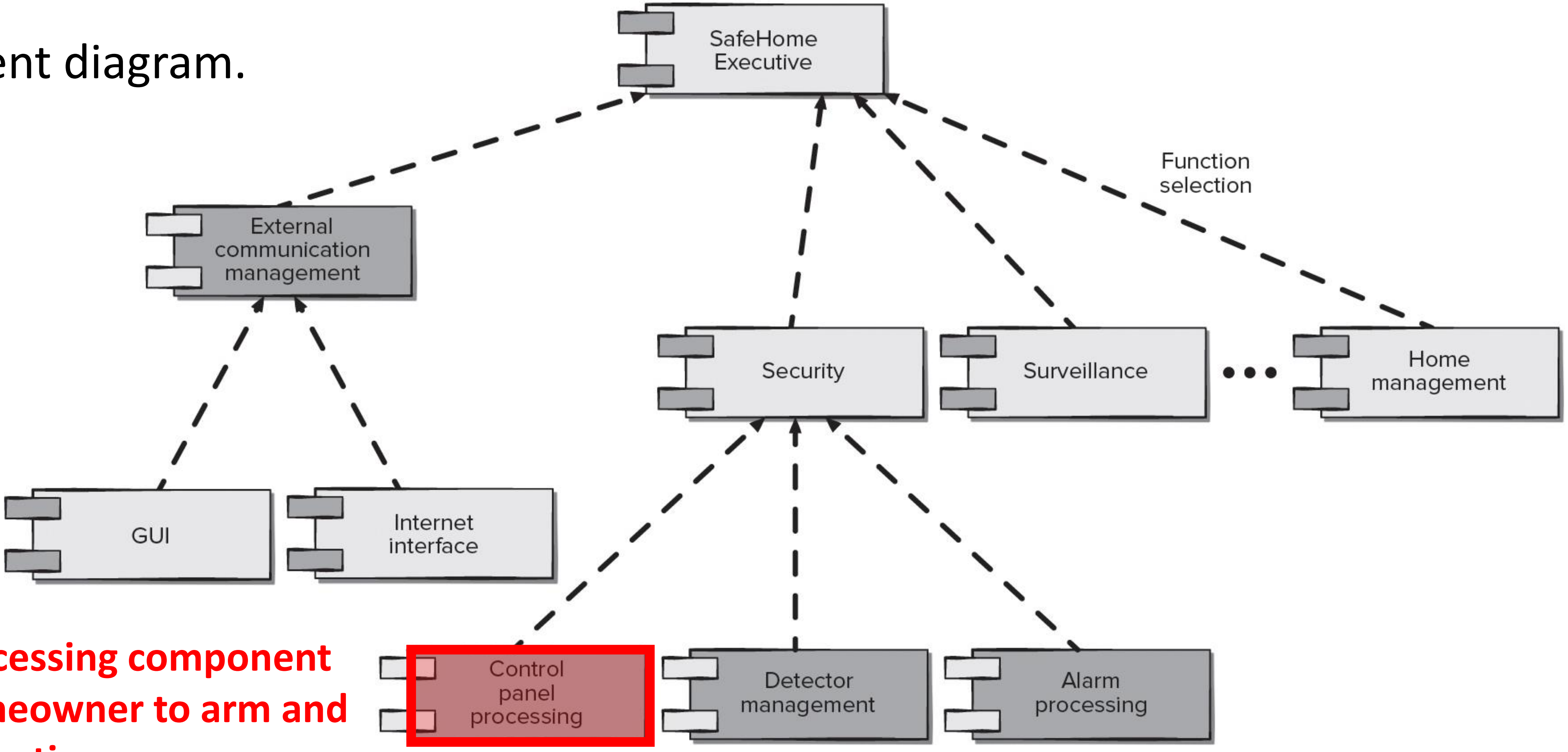


Overall *SafeHome* Architecture

UML Component Diagram

Fig. 10.9 from textbook

Simplified component diagram.



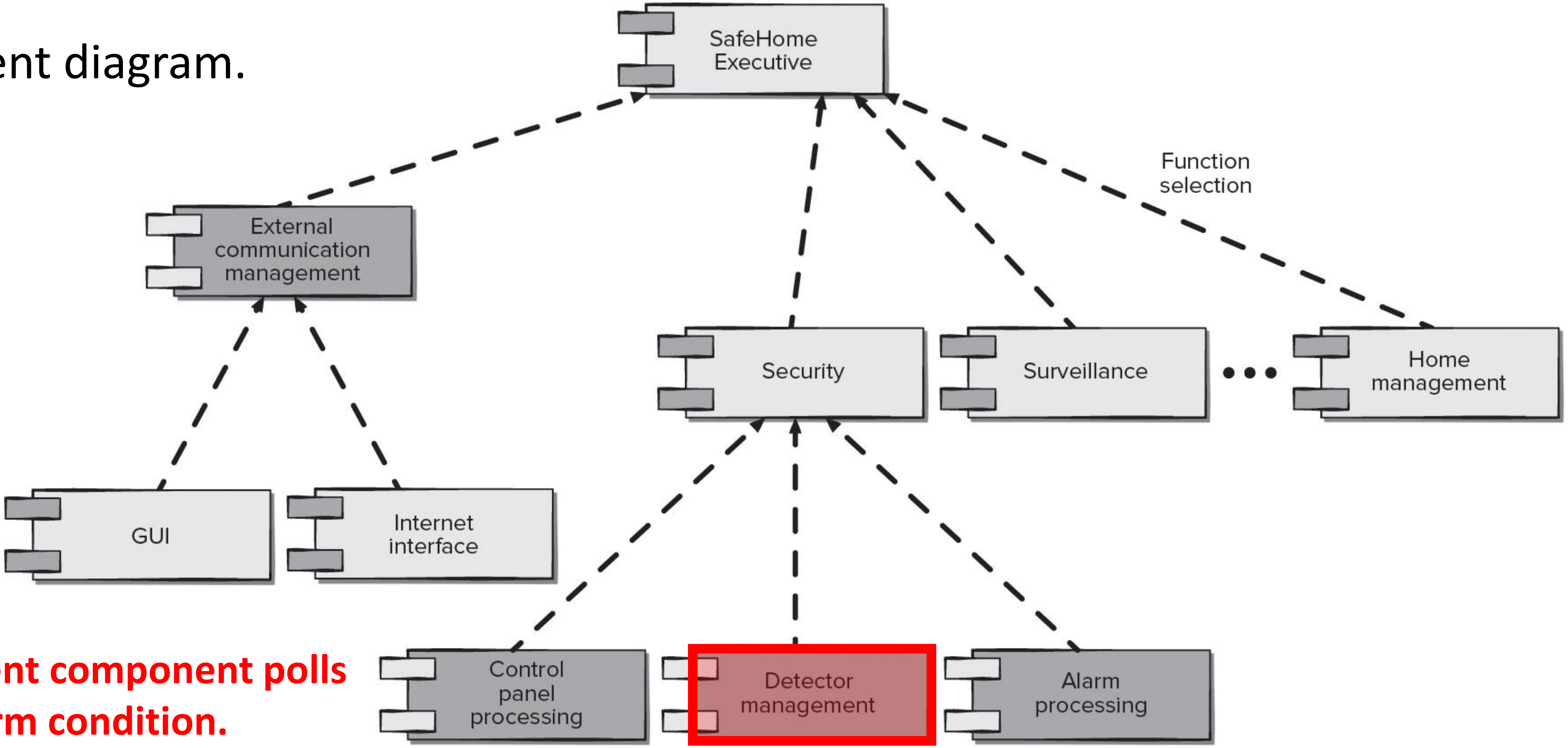
The Control panel processing component interacts with the homeowner to arm and disarm the security function.

Overall *SafeHome* Architecture

UML Component Diagram

Fig. 10.9 from textbook

Simplified component diagram.



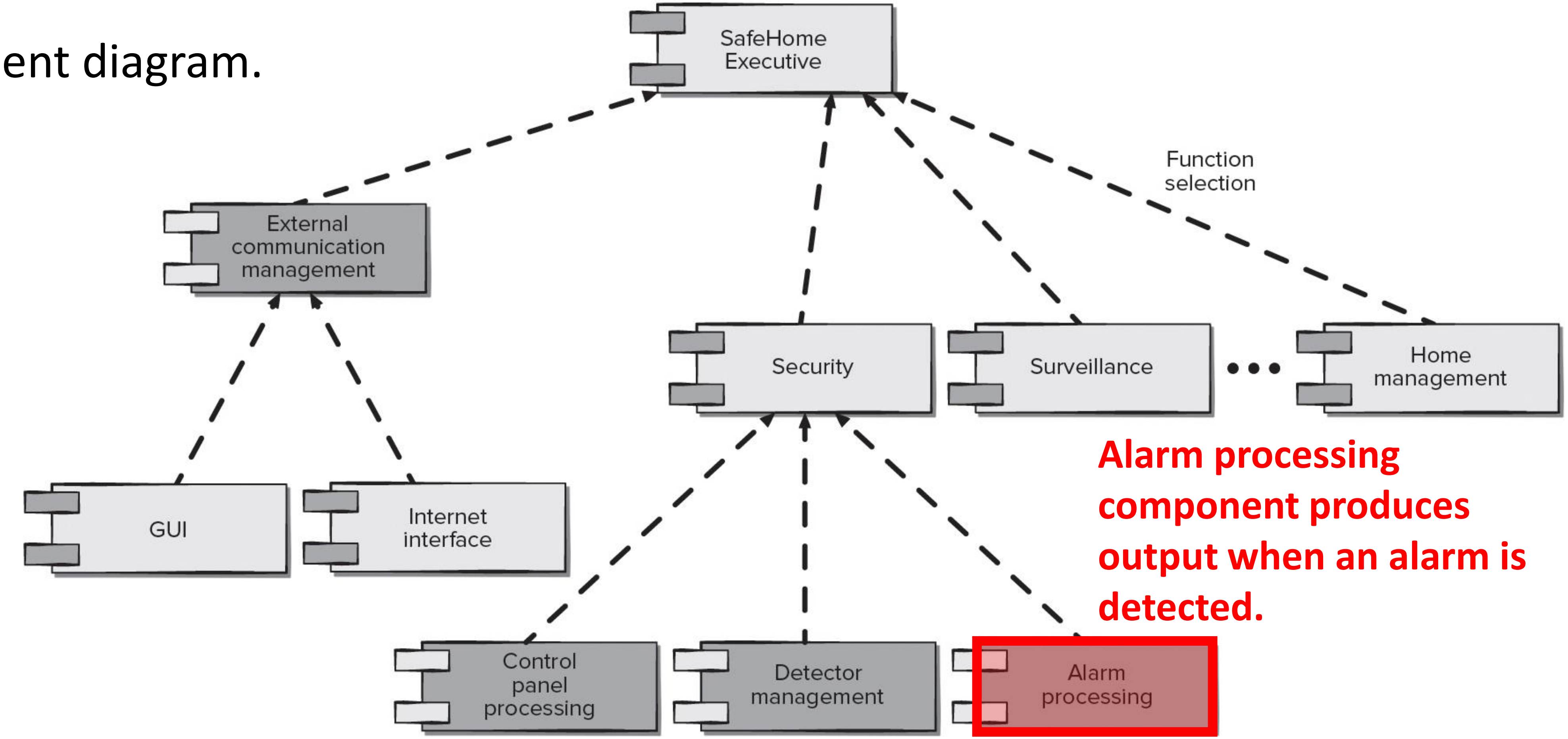
The detector management component polls sensors to detect an alarm condition.

Overall *SafeHome* Architecture

UML Component Diagram

Fig. 10.9 from textbook

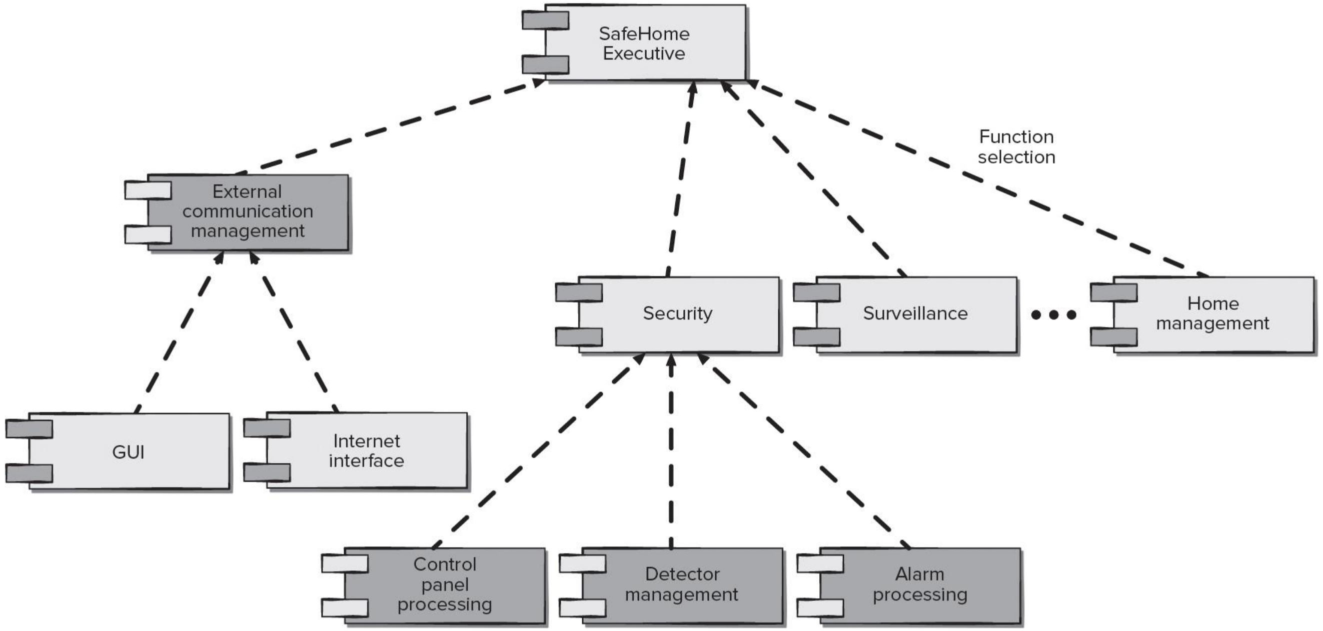
Simplified component diagram.



Alarm processing component produces output when an alarm is detected.

Overall *SafeHome* Architecture

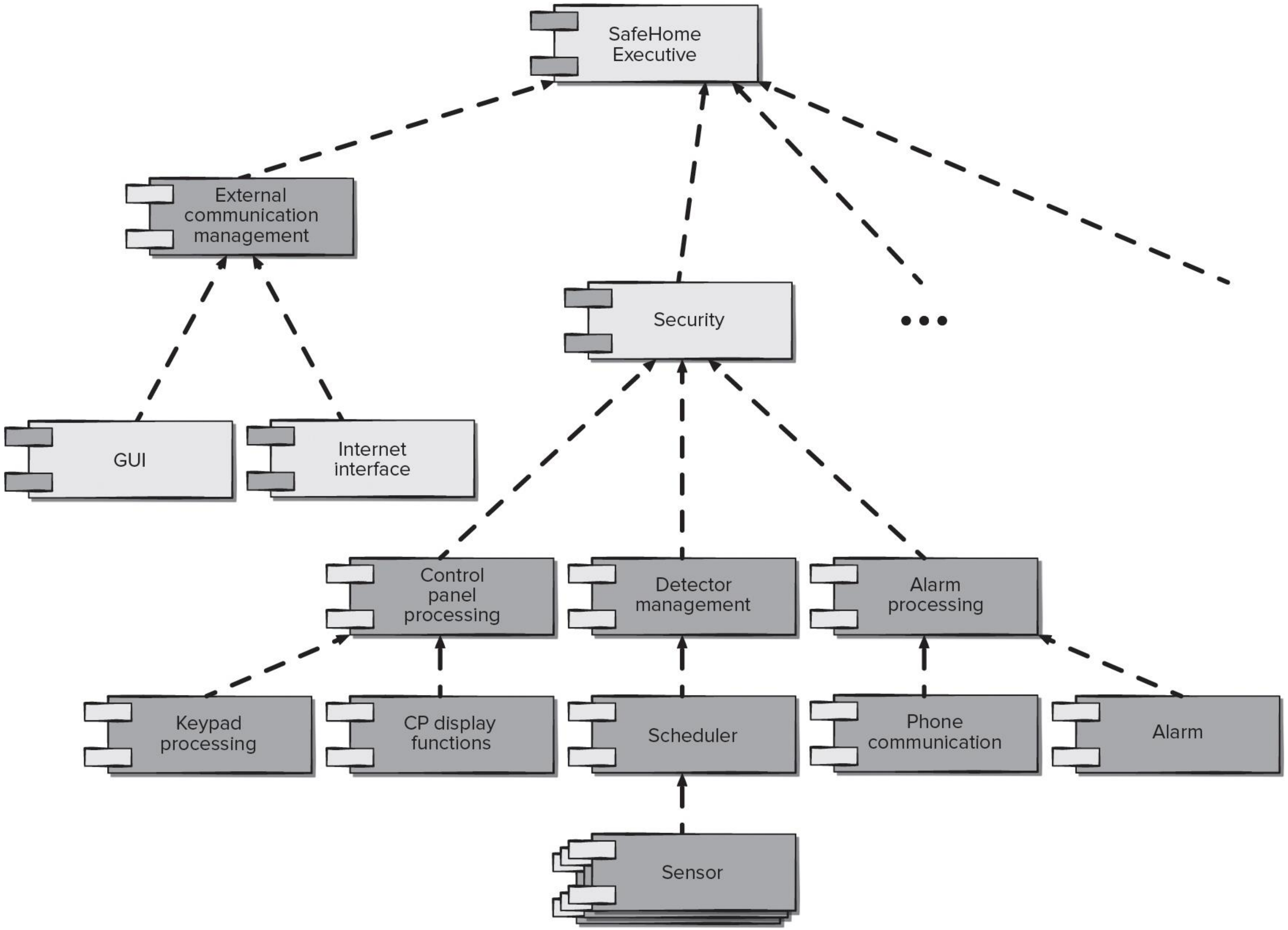
- Still at a very high level.
- Further refinement still necessary to enable construction.
- To accomplish this, architecture is applied to a specific problem.
- Intent is to uncover additional structure, components, and details required to appropriately address the problem.



Overall *SafeHome* Architecture

Elaboration on *SafeHome* security function.

- Still no design detail, that’s left for component-level design.



Architectural 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.
- Unlike requirements reviews that involve all stakeholders, architecture reviews **typically involve only developers and independent experts**.
- Often make use of **experience-based reviews**, **prototype evaluation**, **scenario reviews**, and **checklists**.