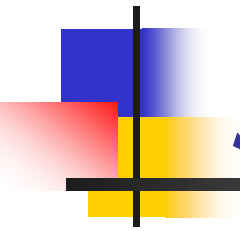
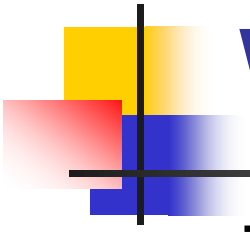


COMP 354: Introduction to Software Engineering

Architectural Design

Based on Chapter 10 of the textbook

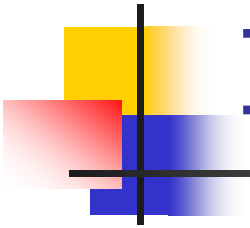


What is Software Architecture?

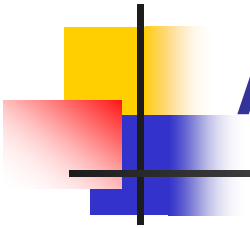
The architecture is not the operational software, it is a representation that enables a software engineer to:

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 Software Architecture Important?



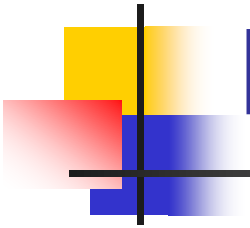
- Software architecture provides a representation that facilitates communication among all stakeholders interested in the development of a computer-based system.
- Architecture highlights early design decisions that will have a profound impact on all software engineering work that follows.
- Architecture constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together.



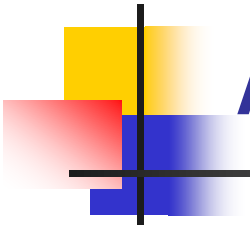
Architectural Descriptions

- The I E E E Computer Society has proposed I E E E-Std-42010-2011, Systems and Software Engineering – Architecture Description.
  - Describes the use of architecture viewpoints, architecture frameworks, and architecture description languages as a means of codifying the conventions and common practices for architectural description.
- The IEEE Standard defines an architectural description (A D) as a "a collection of products to document an architecture."
  - An architecture description shall identify the system stakeholders having concerns considered fundamental to the architecture of the system-of-interest.
  - These concerns shall be considered when applicable and identified in the architecture description: system purpose, suitability of the architecture, feasibility of constructing and deploying the system, risks and impacts of the system, and the maintainability and evolvability of the system.

Architecture Decision Documentation

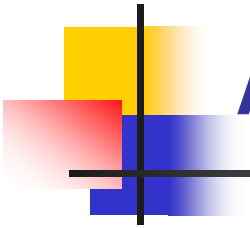


1. Determine information items needed for each decision.
2. Define links between each decision and appropriate requirements.
3. Provide mechanisms to change status when alternative decisions need to be evaluated.
4. Define prerequisite relationships among decisions to support traceability.
5. Link significant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



Agility and Architecture

- To avoid rework, user stories are used to create and evolve an architectural model (walking skeleton) before beginning any coding.
- Use models which allow software architects to add user stories to the evolving storyboard and works with the product owner to prioritize the architectural stories as "sprints" (work units) are planned.
- Well run agile projects include delivery of architectural documentation during each sprint.
- After the sprint is completed, the architect reviews the working prototype for quality before the team presents it to the stakeholders in a formal sprint review.

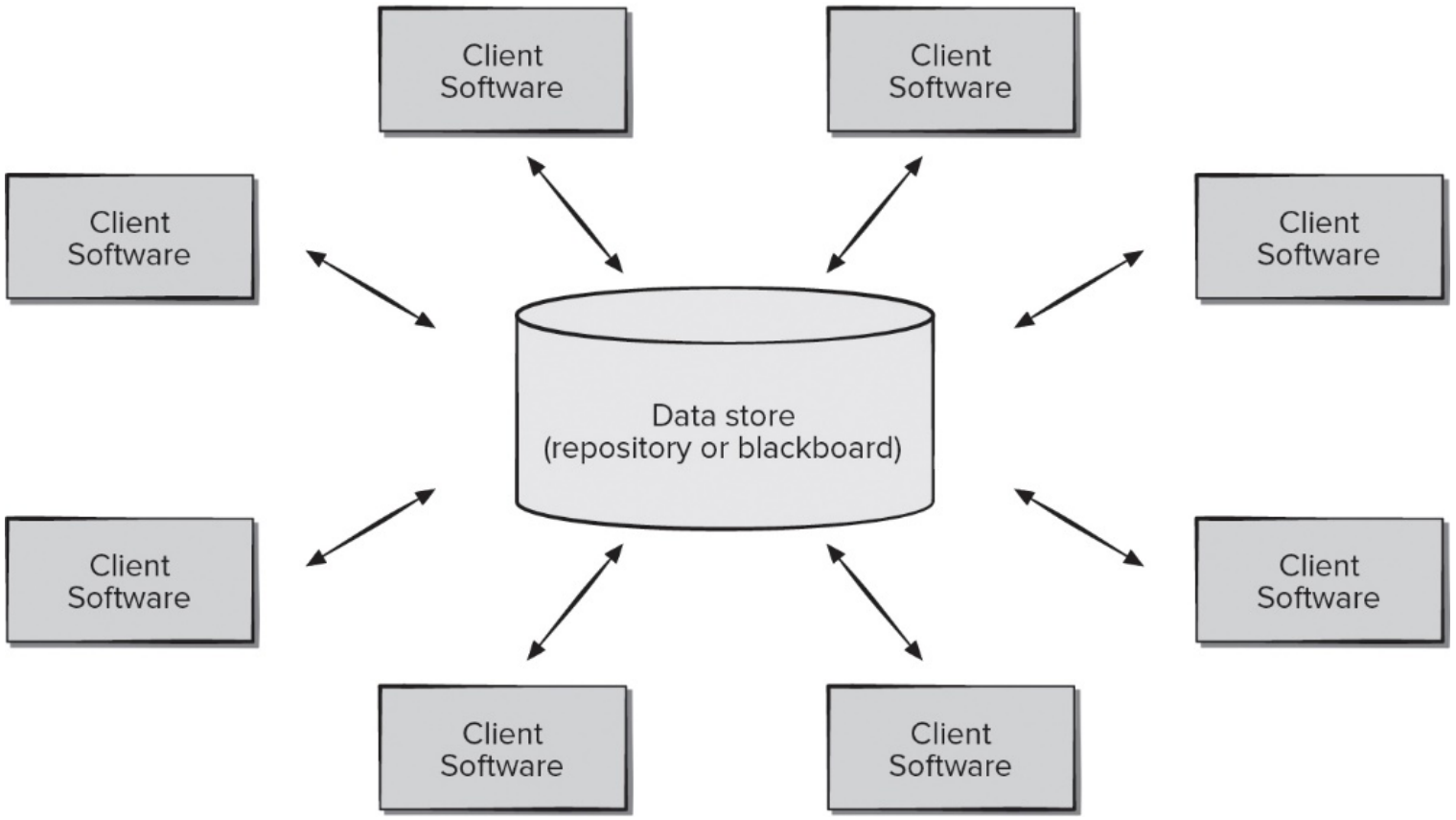


Architectural Styles

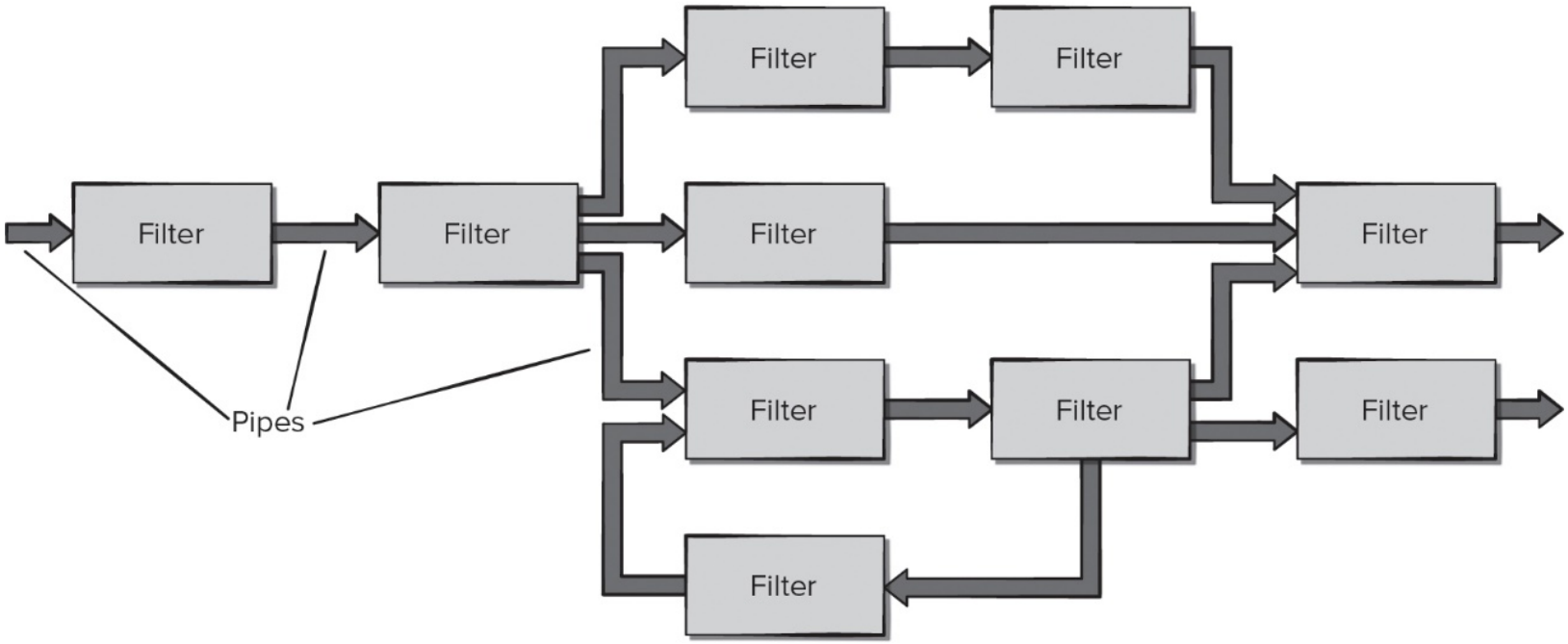
Each style describes a system category that encompasses:

1. set of components (for example: 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.
4. semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

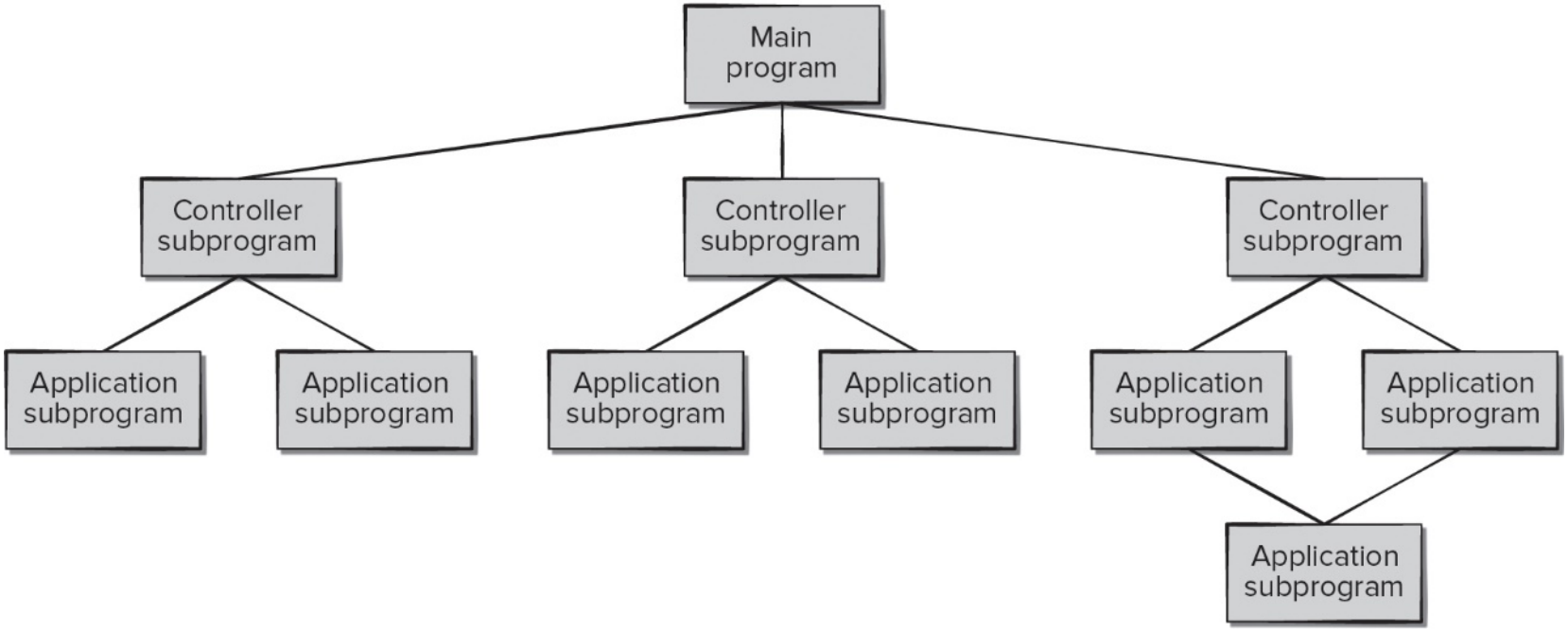
Data Centered Architecture



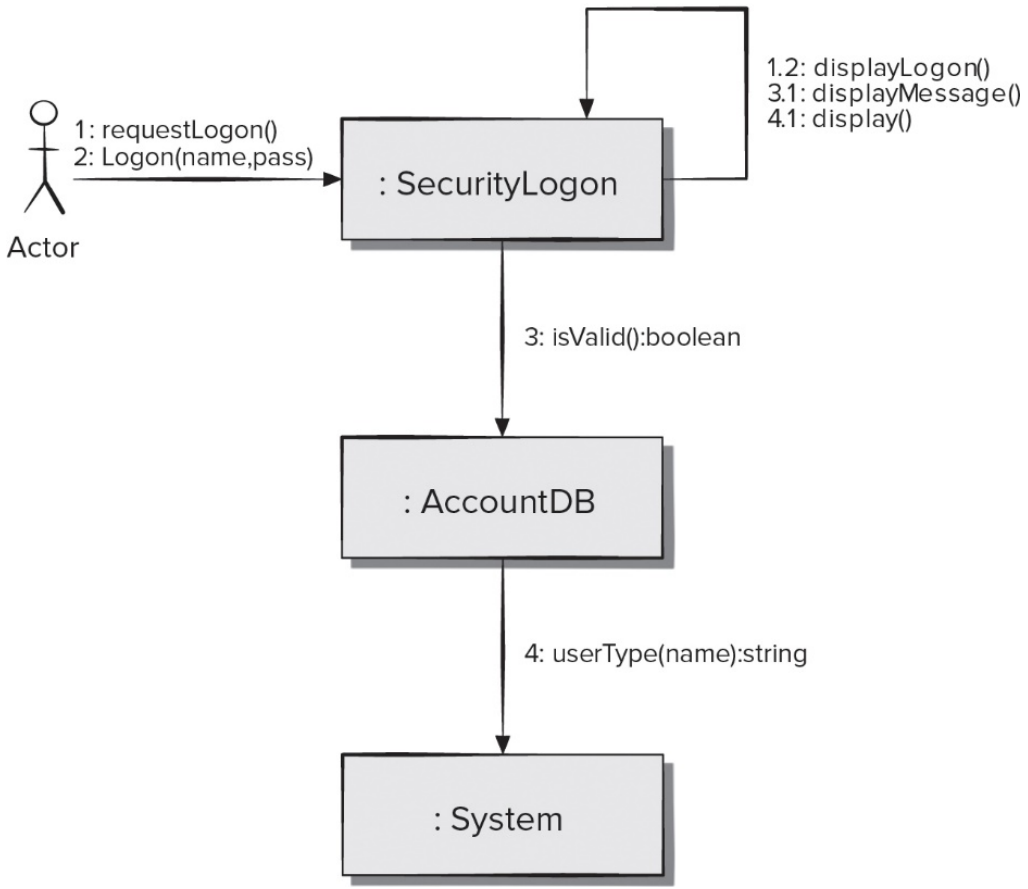
Data Flow Architecture



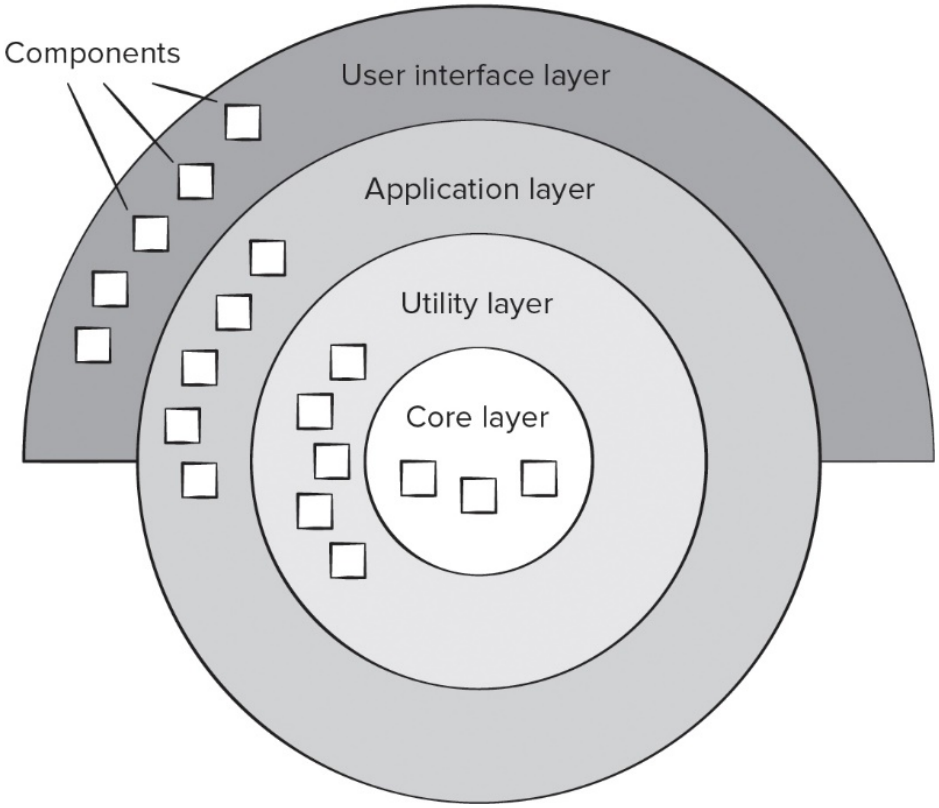
Call Return Architecture



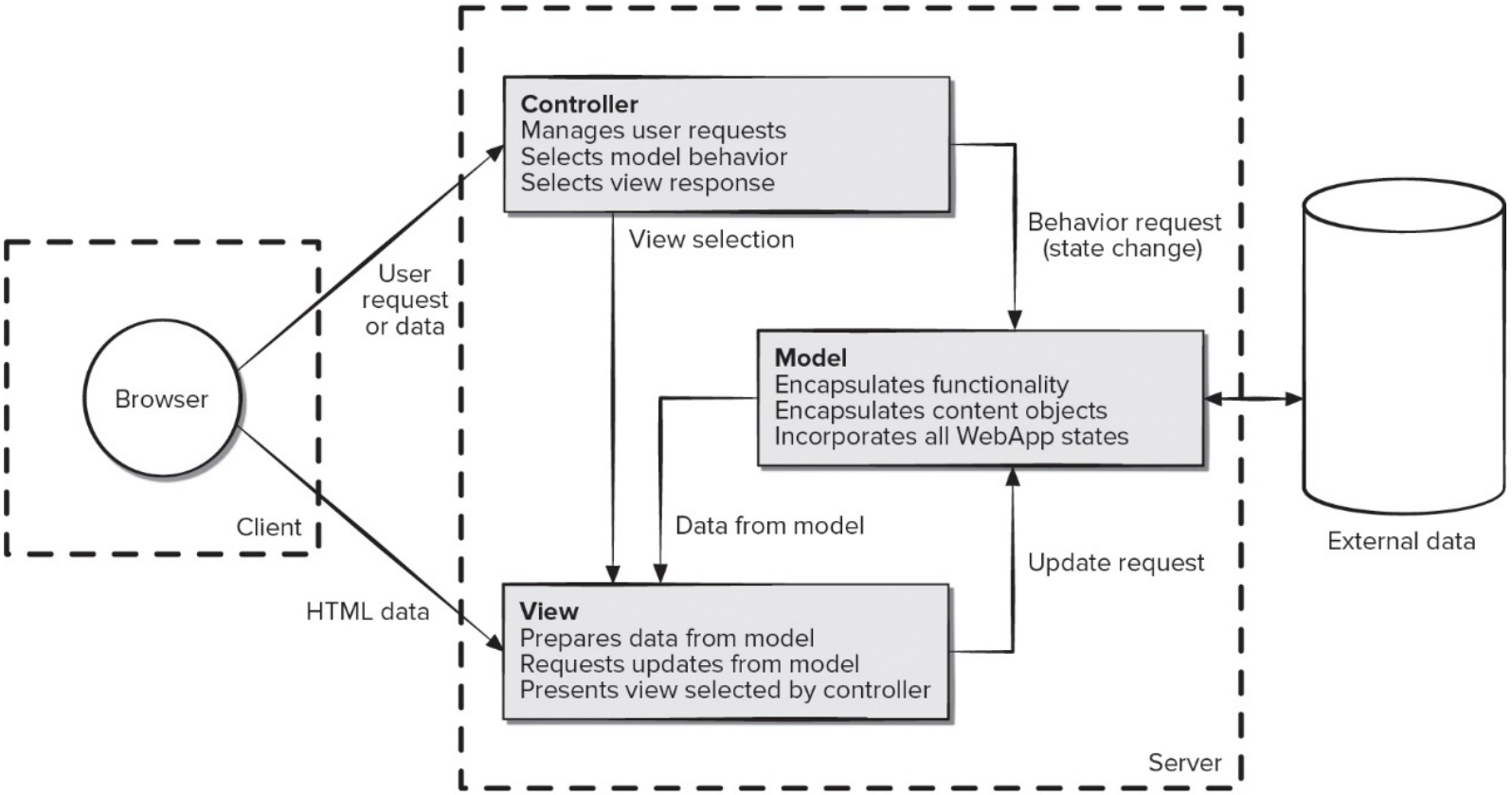
Object-Oriented Architecture



Layered Architecture

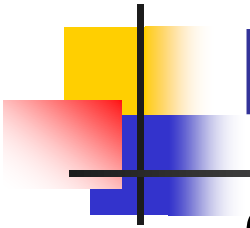


Model View Controller Architecture



Source: Adapted from Jacyntho, Mark Douglas, Schwabe, Daniel and Rossi, Gustavo, "An Architecture for Structuring Complex Web Applications," 2002, available at http://www-di.inf.puc-rio.br/schwabe/papers/OOHDMJava2%20Report.pdf

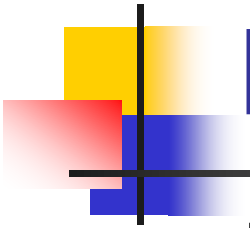
Architectural Organization and Refinement



Control.

- How is control managed within the architecture?
- Does a distinct control hierarchy exist, and if so, what is the role of components within this control hierarchy?
- How do components transfer control within the system?
- How is control shared among components?
- What is the control topology (that is, the geometric form that the control takes)?
- Is control synchronized, or do components operate asynchronously?

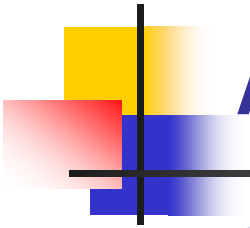
Architectural Organization and Refinement



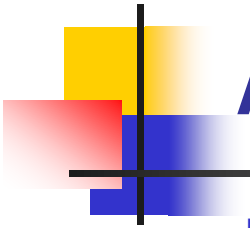
Data.

- How are data communicated between components?
- Is the flow of data continuous, or are data objects passed to the system sporadically?
- What is the mode of data transfer?
- Do data components exist, and if so, what is their role?
- How do functional components interact with data components?
- Are data components passive or active?
- How do data and control interact within the system?

Architectural Considerations



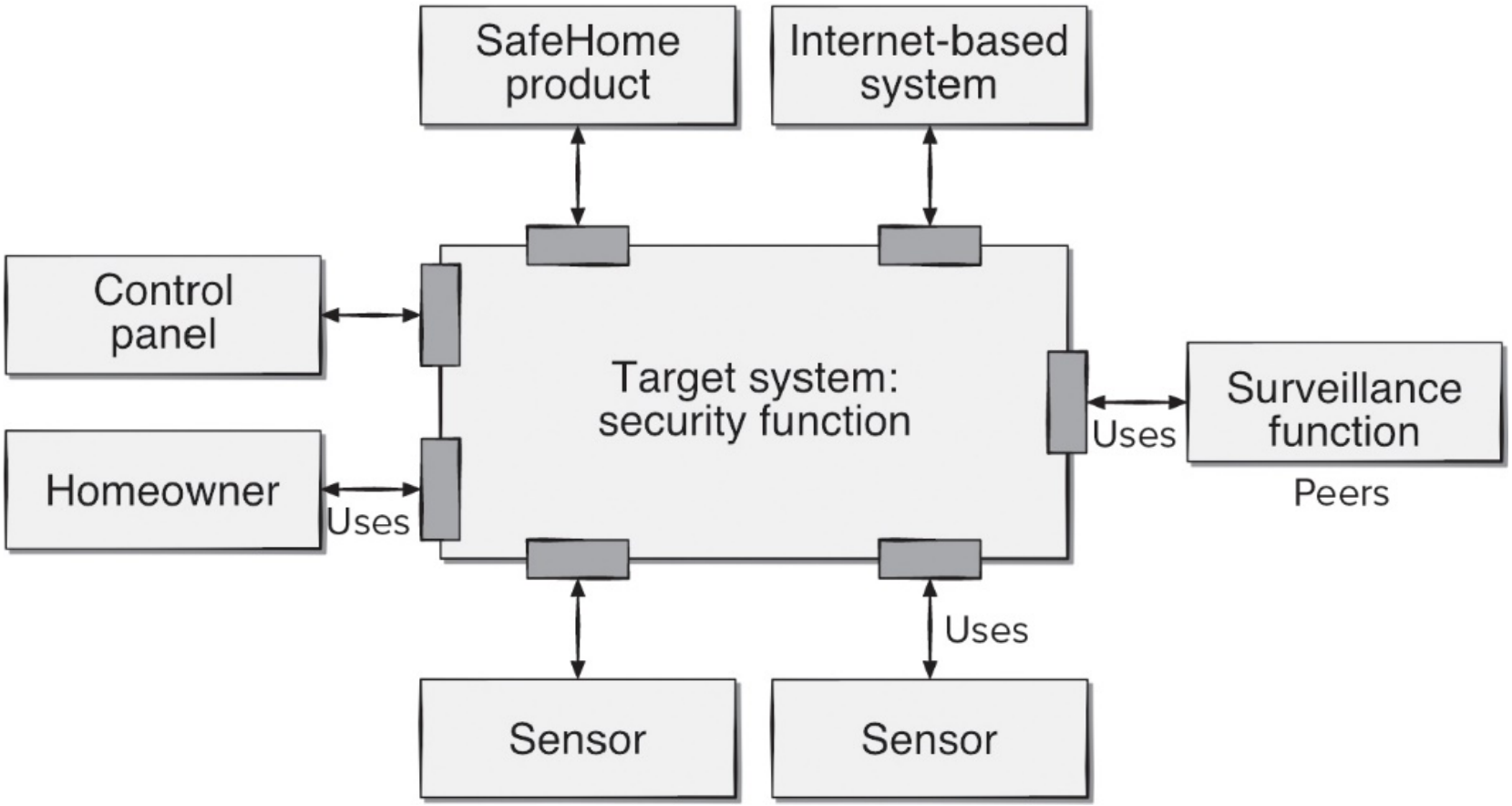
- Economy – software is uncluttered and relies on abstraction to reduce unnecessary detail.
- Visibility – Architectural decisions and their justifications should be obvious to software engineers who review.
- Spacing – Separation of concerns in a design without introducing hidden dependencies.
- Symmetry – Architectural symmetry implies that a system is consistent and balanced in its attributes.
- Emergence – Emergent, self-organized behavior and control are key to creating scalable, efficient, and economic software architectures.



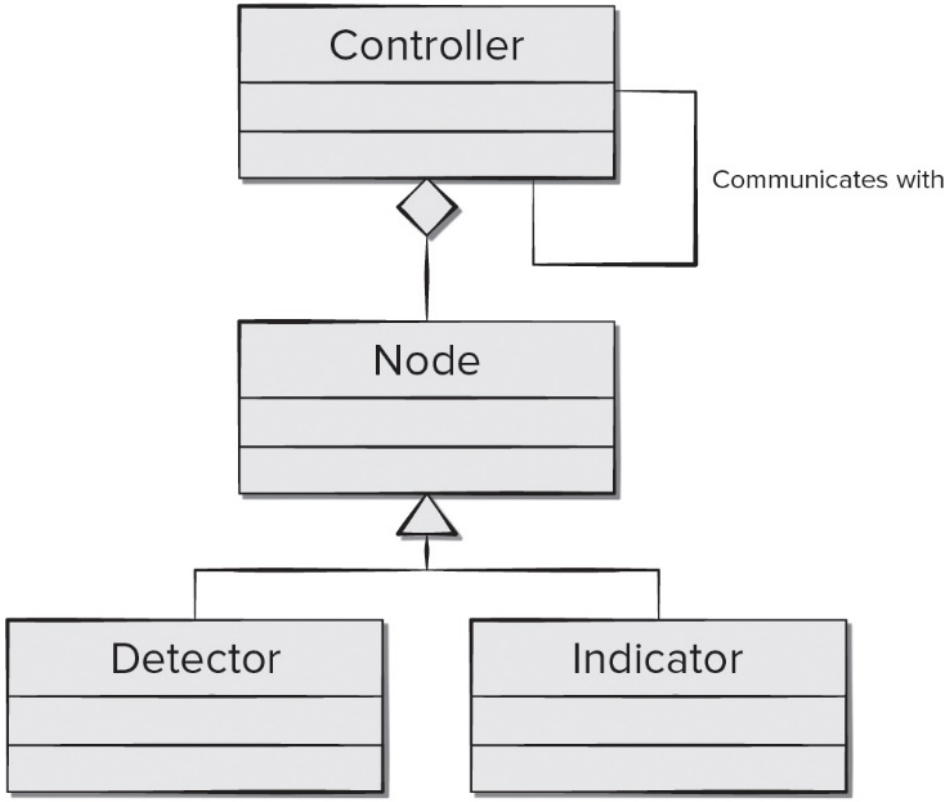
Architectural Design

- 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.

Architecture Context Diagram

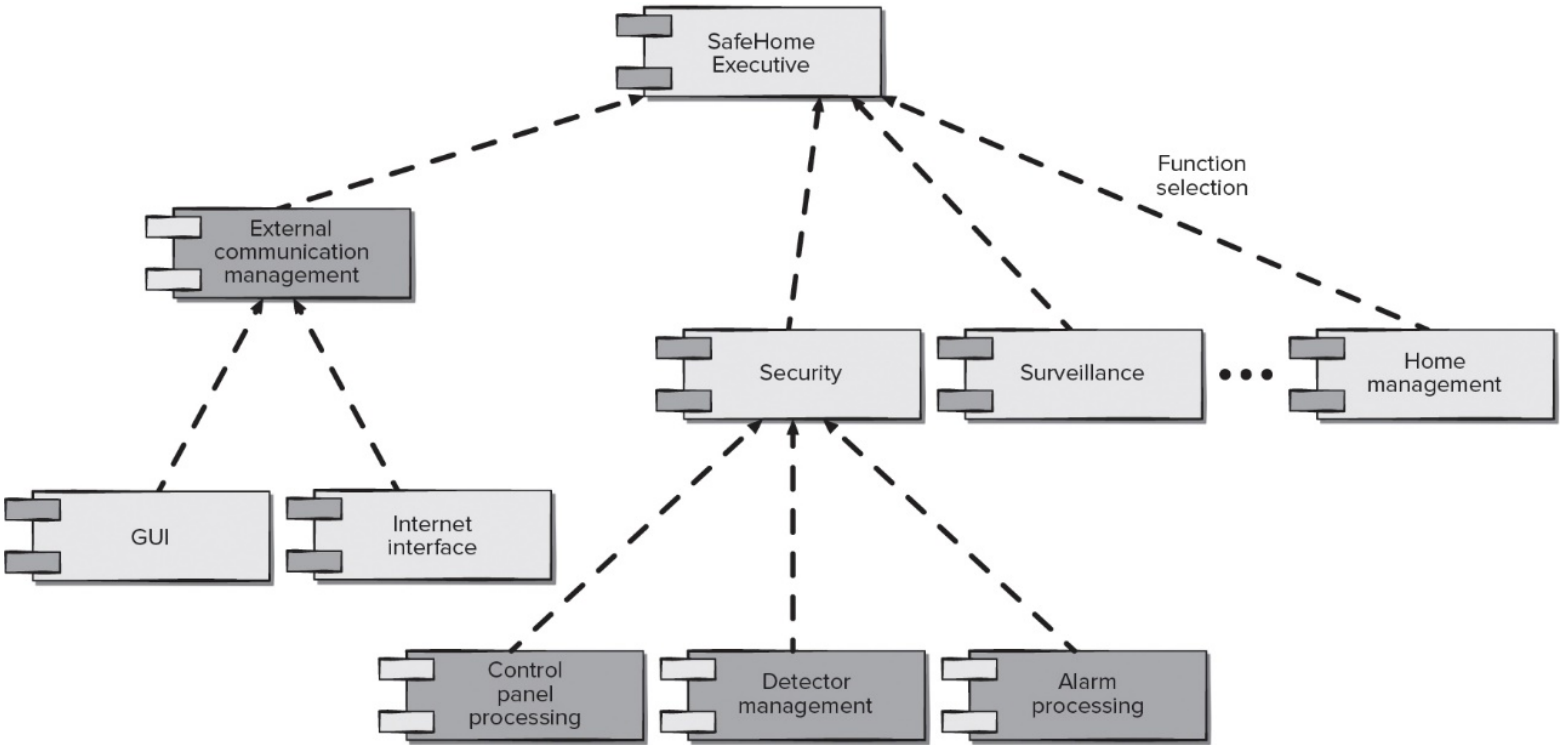


SafeHome Security Function Archetype

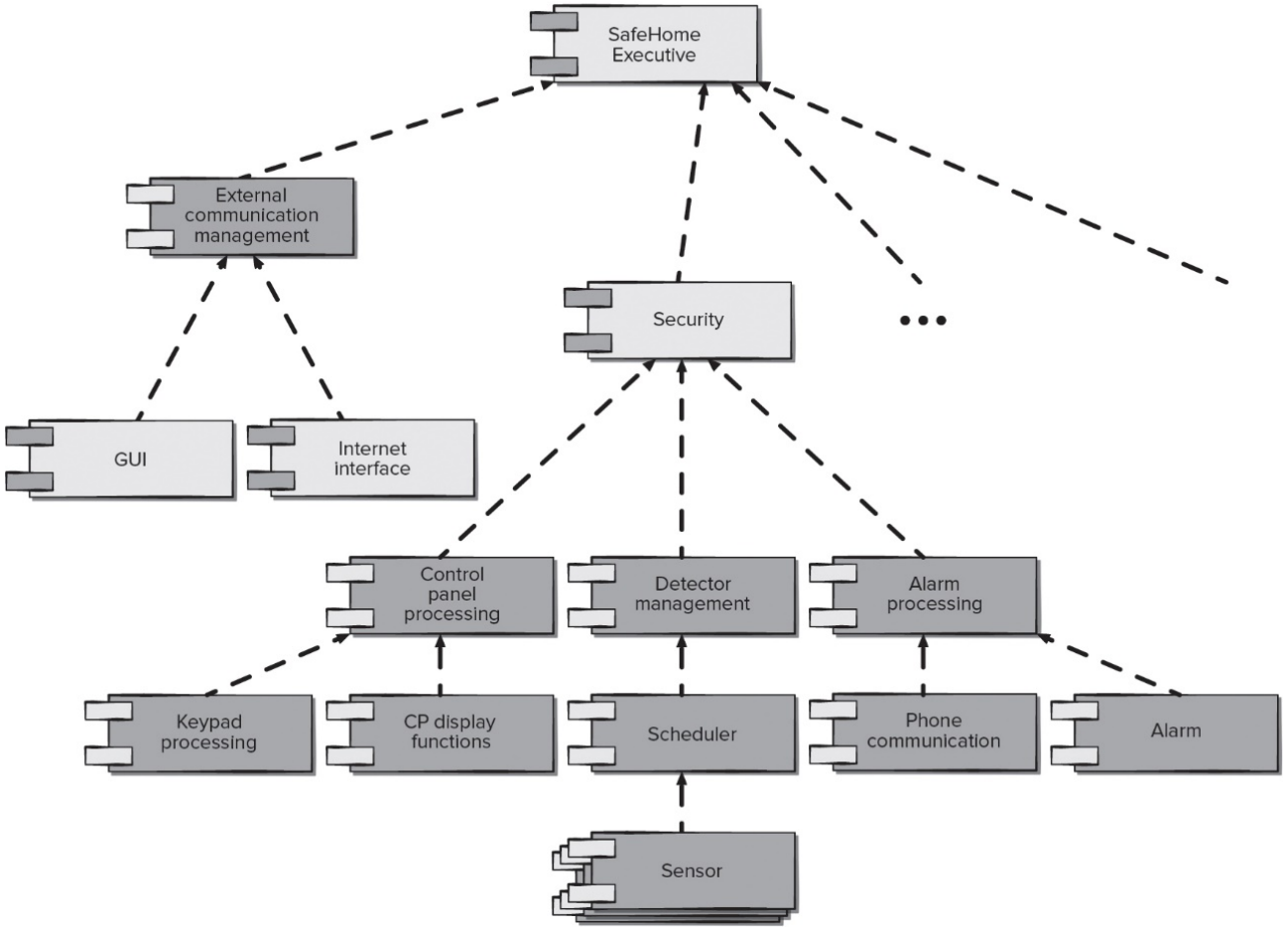


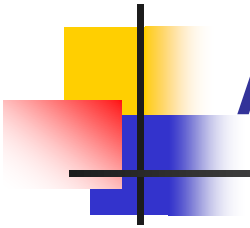
Source: Adapted from Bosch, Jan, Design & Use of Software Architectures. Pearson Education, 2000.

SafeHome Top-Level Component Architecture



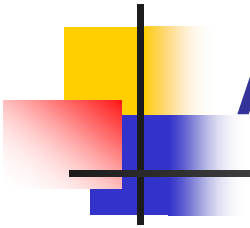
SafeHome Refined Component Architecture





Architectural Tradeoff Analysis

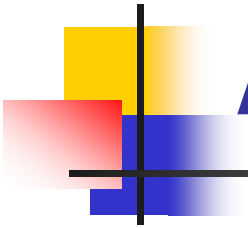
1. Collect scenarios.
2. Elicit requirements, constraints, environment description.
3. Describe the architectural styles/patterns that have been chosen to address the scenarios and requirements using one of these views: module, process, data flow.
4. Evaluate quality attributes by considering each one 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).



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.
- Often make use of experience-based reviews, prototype evaluation, and scenario reviews, and checklists.

Pattern-based Architectural Reviews



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