System Models

Minsoo Ryu

Hanyang University

Contents

1. Context Models
2. Structural Model
3. Behavioural Models
4. Object Models

Building a System Model

- User requirements should be written in natural language because they have to be understood by people who are not technical experts
- System requirements may be expressed in a more technical way
  - One widely use technique is to document the system specification as a set of system models
  - These models are usually graphical representations which are often more understandable than detailed natural language descriptions

System Models

- System models are also referred to as analysis models
  - A system or analysis model is the first technical representation of a system
- System models are also an important bridge between the requirements analysis and system design processes

Analysis Model vs. Design Model

- **Analysis model**
  - The analysis process focuses on aspects of the problem that are visible to users or customers
  - The level of abstraction of an analysis model is relatively high

- **Design model**
  - The design process refines the analysis model by providing detail that will enable the model to be implemented, and creates a new set of model elements that need not be visible to users and customers (visible only to developers)
  - The level of abstraction of a design model is relatively low

System Modeling Perspectives

- Different models present the system from different perspectives
  - External perspective showing the system's context or environment;
  - Structural perspective showing the system or data architecture.
  - Behavioural perspective showing the behaviour of the system;

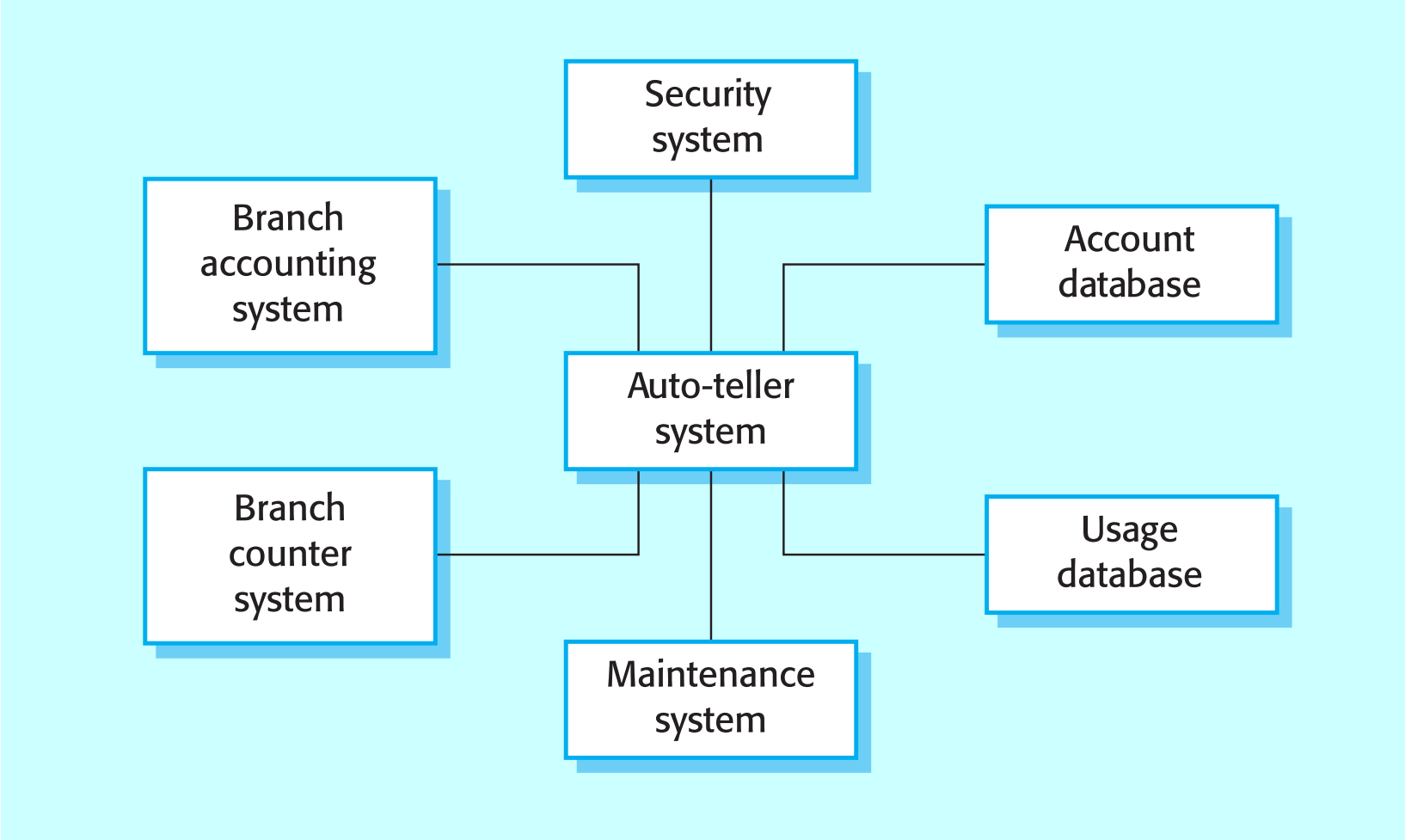
Contents

1. Context Models
2. Structural Models
3. Behavioural Models
4. Object Models

Context Models

- At an early stage in the requirements elicitation and analysis process, you should decide on the boundaries of the system
  - This involves working with system stakeholders to distinguish what is the system and what is the system's environment
- In some cases, the boundary is relatively clear
- In other cases, there is more flexibility, and you decide what constitutes the boundary during the requirements engineering process

The Context of an ATM System



Contents

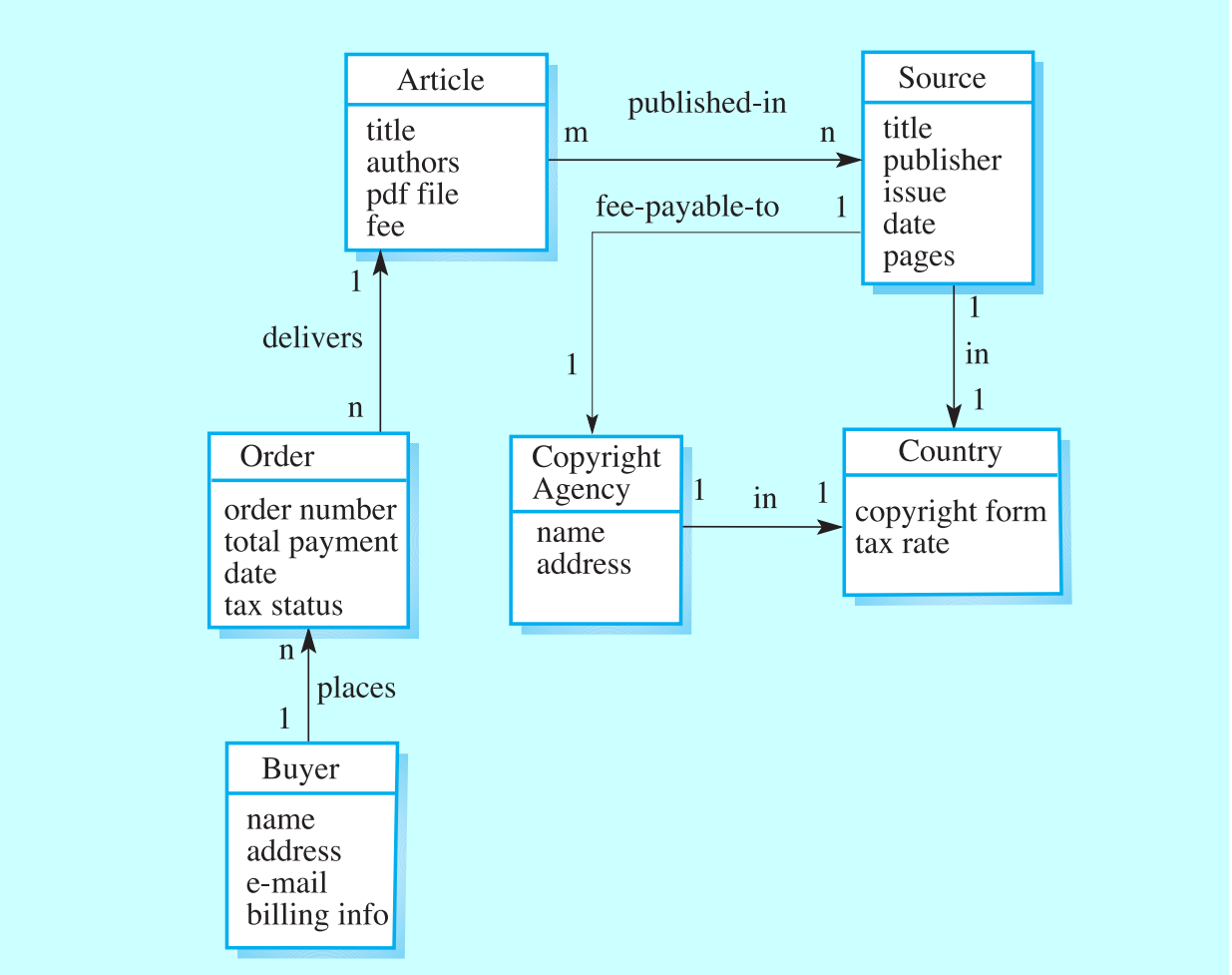
1. Context Models
2. Structural Models
   - Data Models
3. Behavioural Models
4. Object Models

Data Models

- System modelling with data models is to define the logical form of the data processed by the system
  - Sometimes called semantic data model

- An entity-relation-attribute (ERA) modelling is the most widely used technique
  - ERA modelling shows the data entities, their associated attributes, and the relations between these entities

Library Data Model



Data Dictionaries

- Like all graphical models, data models lack detail, and you should maintain more detailed descriptions of the entities, relationships, and attributes that are included in the model
  - You can collect these more detailed descriptions in a repository or data dictionary
- A data dictionary is an alphabetical list of the names used in the system models
  - Descriptions of the entities, relationships and attributes are also included

Data Dictionary Entries

Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a co py of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

Contents

1. Context Models
2. Structural Models
3. Behavioural Models
4. Object Models

Behavioural Models

- Behavioural models are used to describe the overall behaviour of a system.

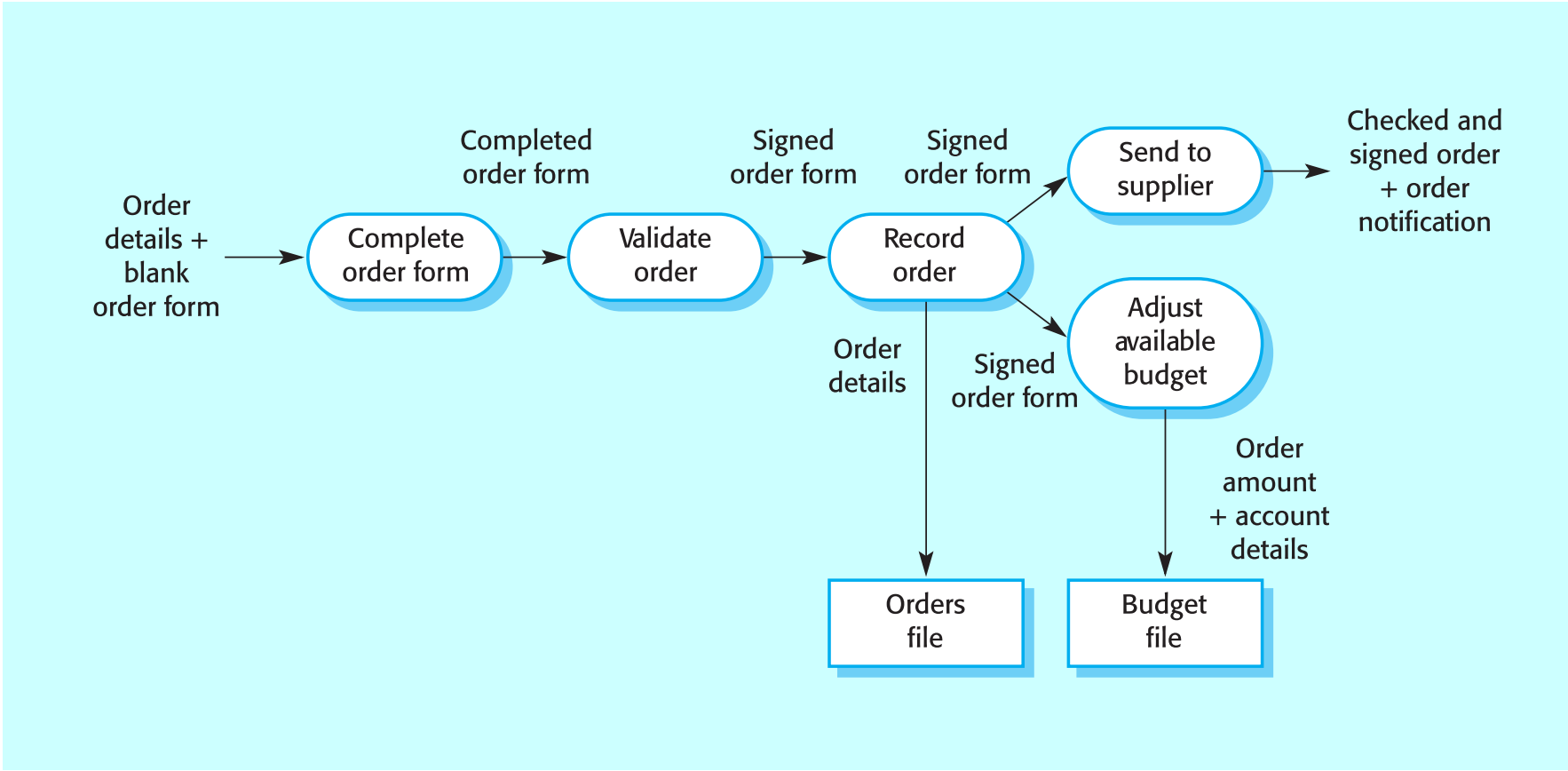
- A widely used behavioural models are
  - Data-flow model that shows how data is processed by a system
  - State machine model that shows the systems response to events

Data-Flow Models

- Data-flow models are an intuitive way of showing how data is processed by a system
  - They are used to show how data flows through a sequence of processing steps

- Data-flow diagrams (DFDs)
  - Functional processing (transformation): rounded rectangles
  - Data stores: rectangles
  - Data movement: labelled arrows

Order Processing DFD



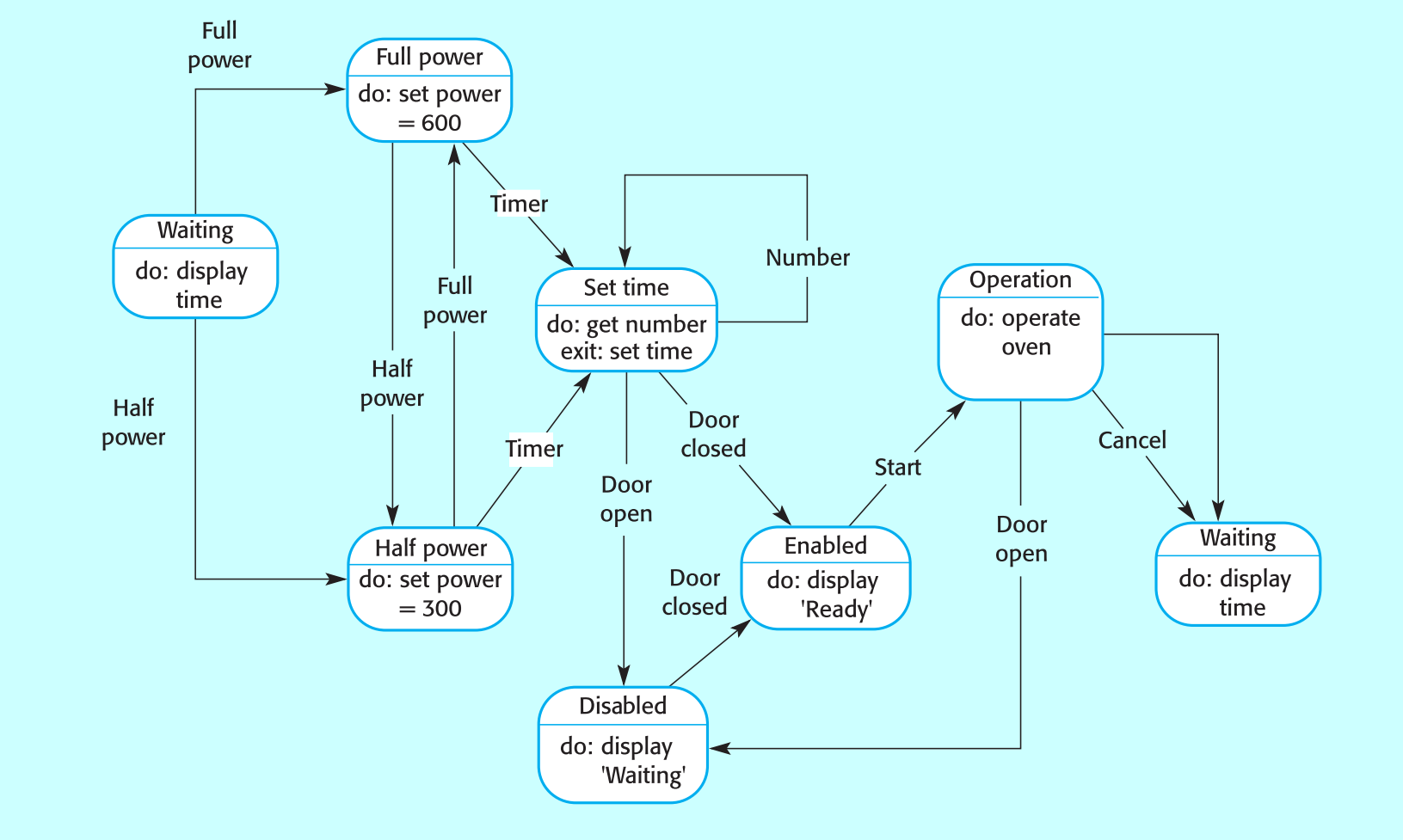
Advantages of Data-Flow Models

- Data-flow models are simple and intuitive
  - It is usually possible to explain them to potential system users who can then participate in validating the analysis
- Data-flow models show end-to-end processing in a system
  - They show the entire sequence of actions that take place from an input being processed to the corresponding output that is the system's response

State Machine Models

- A state machine model describes how a system responds to internal or external events
  - They show system states and events that cause transitions from one state to another
  - They do not show the flow of data within the system
  - They are often used for modelling real-time systems because these systems are often driven by stimuli from the system's environment

Microwave Oven Model



Problems with State Machine Models

- The number of possible states increases rapidly
  - For large system models, one way to manage a large number of states is by using the notion of a superstate that encapsulates a number of separate states
  - This superstate looks like a single state on a high-level model but is then expanded in more detail on a sepearate diagram

Contents

1. Context Models
2. Structural Models
3. Behavioural Models
4. Object Models

Object Models

- Object models may be used to represent both system data and its processing
  - In this respect, they combine some of the uses of data-flow model and semantic data models
- Object models are natural ways of reflecting the real-world entities that are manipulated by the system
  - This is particularly true when the system possesses information about tangible entities, such as cars, aircraft, or books
- More abstract, higher-level entities, are harder to model as object classes
  - E.g., a library, a medical record system, or a word processor

Object Models

- Developing object models during requirements analysis usually simplifies the transition to object-oriented design and programming

- However, end-users of a system often find object models unnatural and difficult to understand
  - They may prefer to adopt a more functional, data-processing view
  - Therefore, it is sometimes helpful to supplement object models with data-flow models that show the end-to-end data processing in the system

Objects and Object Classes

- An object class is an abstraction over a set of entities that identifies common attributes and the services or operations that are provided by each object
  - Objects are instantiations of object classes

- Generally, the object models focus on object classes and their relationships

Architectural Design

Minsoo Ryu

Hanyang University

Contents

1. Architecture
2. Architectural Styles
3. Architectural Design

1. Architecture

- Architectural design is the initial design process of identifying these sub-systems and establish a structural framework for sub-system control and communications

- "The architecture of a system is a comprehensive framework that describes its form and structure—its components and how they fit together," by Jerrold Grochow

Software 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 between them

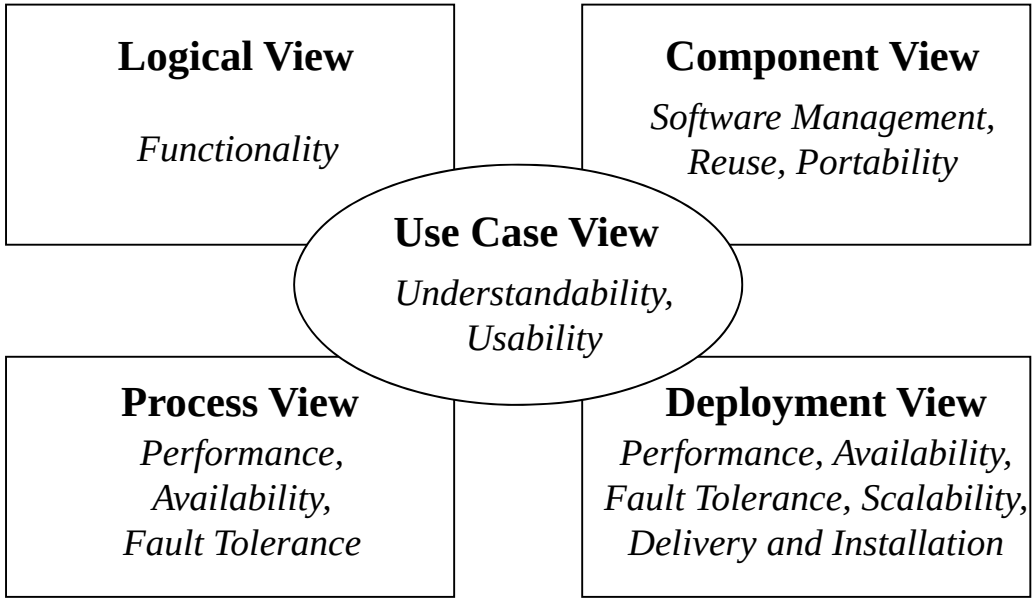
- To date there is still no agreement on the precise definition of the term “software architecture”

“4+1” Software Architecture

- “4+1” architecture designed by Philippe Kruchten
  - Logical view: the functional requirements of the system captured in class diagrams
  - Component view: the actual software module organization within the development environment physical partitioning of the system
  - Process View: the run-time implementation structure of the system
  - Deployment view: mapping software to processing nodes

“4+1” Software Architecture

- Logical view
  - class diagram level
- Component view
  - source code level
  - one-to-one mapping between classes and components
- Process view
  - executable file level
- Deployment view
  - the configuration of processing elements and the software processes



Architecture Description Languages

- Architecture description languages (ADLs) are used to describe a Software Architecture
  - Common elements of an ADL are component, connector and configuration

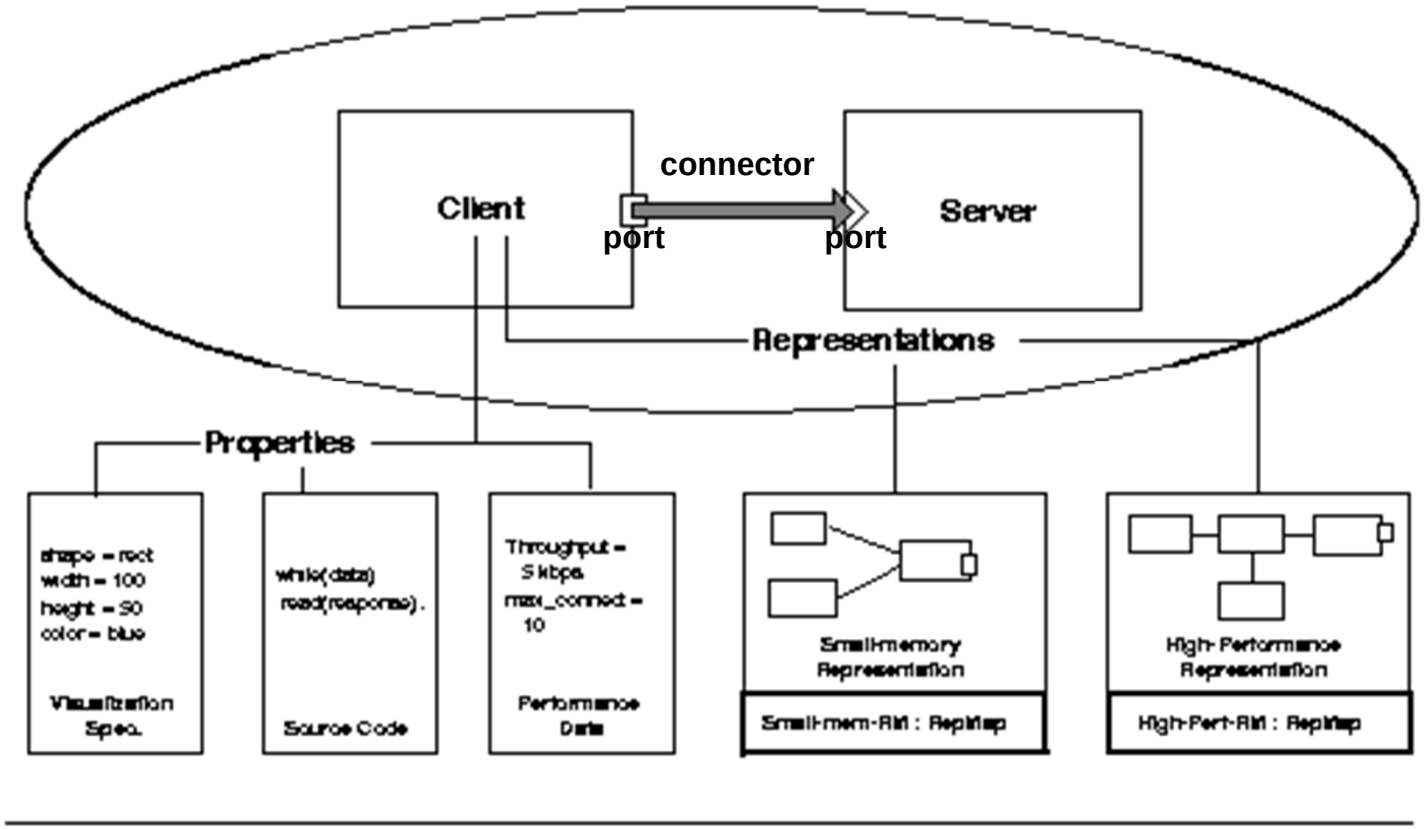
- Several different ADLs have been developed by different organizations, including;
  - Wright (developed by Carnegie Mellon)
  - Acme (developed by Carnegie Mellon)
  - xADL (developed by UCI)
  - Darwin (developed by Imperial College London)
  - DAOP-ADL (developed by University of Málaga)

Acme

- **Built on several types of architectural representation**
  - **Components represent the primary computational elements and data stores of a system**
    - Typical examples of components include such things as clients, servers, filters, objects, blackboards, and databases
  - **Connectors represent interactions among components**
    - Computationally speaking, connectors mediate the communication and coordination activities among components
    - Informally they provide the "glue" for architectural designs
    - Examples include simple forms of interaction, such as pipes, procedure call, and event broadcast
  - **Systems represent configurations of components and connectors**

Acme

❒ A simple client-server architecture (graphical)



Acme

❒ A simple client-server architecture (textual)

```
System simple_cs = {
  Component client = {
    Port send-request;
    Property Aesop-style : style-id = client-server;
    Property UniCon-style : style-id = client-server;
    Property source-code : external = "CODE-LIB/client.c";
  }
  Component server = {
    Port receive-request;
    Property idempotence : boolean = true;
    Property max-concurrent-clients : integer = 1;
    source-code : external = "CODE-LIB/server.c";
  }
  Connector rpc = {
    Role caller;
    Role callee;
    Property asynchronous : boolean = true;
    max-roles : integer = 2;
    protocol : Wright = " ... ";
  }
  Attachment client.send-request to rpc.caller;
  Attachment server.receive-request to rpc.callee;
}
```

Advantages of Explicit Architecture Design (Bass et al., 2003)

- **Stakeholder communication**
  - The architecture is a high-level presentation of the system
- **System analysis**
  - Making the system architecture explicit requires some analysis, in which architectural design decisions have a profound effect on whether the system can meet critical requirements such as performance, reliability, and maintainability
- **Large-scale reuse**
  - The architecture may be reusable across a range of related systems

Impact of System Architecture

❒ Performance

- If performance is a critical requirement, the architecture should be designed to localize critical operations within a small number of sub-systems, with as little communication as possible between these sub-systems
- This may mean using relatively large-grain than fine-grain components to reduce component communications

❒ Security

- If security is a critical requirement, a layered structure should be used, with the most critical assets protected in the innermost layers and with a high level of security validation applied to these layers

Impact of System Architecture

- **Safety**
  - Localize safety-critical features in a small number of sub-systems
  - This reduces the costs and problems of safety validation
- **Availability**
  - Include redundant components and mechanisms for fault tolerance
- **Maintainability**
  - Use fine-grain, replaceable components

Architectural Conflicts

- Using large-grain components improves performance but reduces maintainability
- Introducing redundant data improves availability but makes security more difficult
- Localizing safety-related features usually means more communication so degraded performance

Contents

1. Architecture
2. Architectural Styles
3. Architectural Design

2. Architectural Styles

- The architectural model of a system may conform to a generic architectural model or style
  - An awareness of these styles can simplify the problem of defining system architectures
  - However, most large systems are heterogeneous and do not follow a single architectural style

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

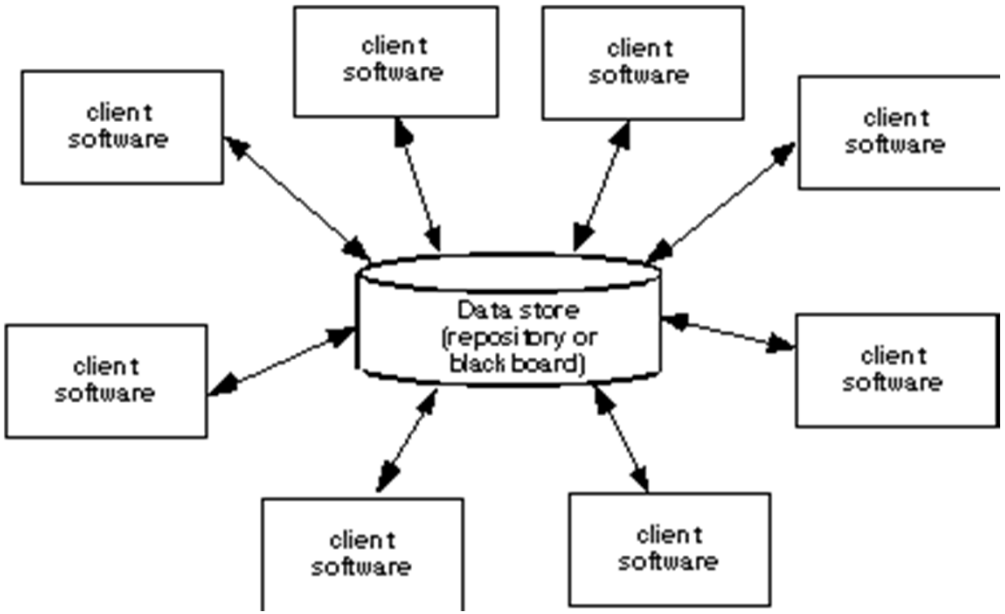
A Brief Taxonomy of Architectural Styles

- Although millions of computer-based systems have been created over the past 50 years, the vast majority can be categorized into one or a relatively small number of architectural styles

- Taxonomy
  - Data-centered architecture
  - Data-flow architecture
  - Call and return architecture
  - Object-oriented architecture
  - Layered architecture

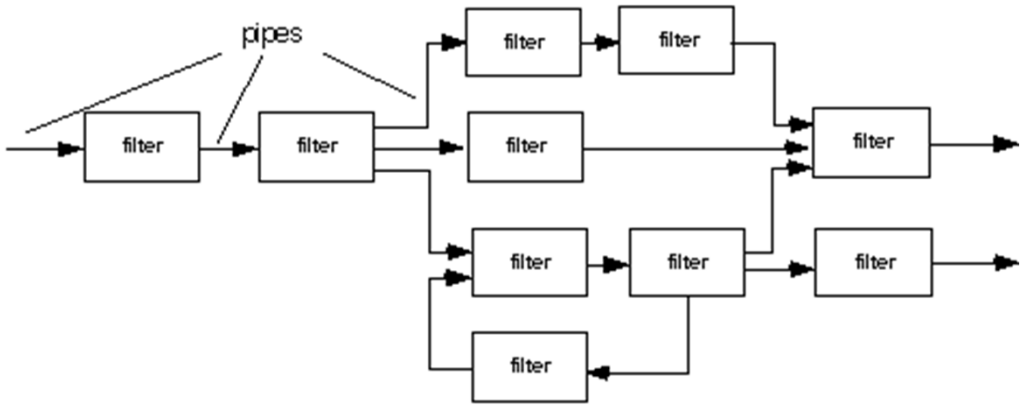
Data-Centered Architecture

- Also called repository architecture
- A data store (a file or database) resides at the center of this architecture and is accessed frequently by other components



Data-Flow Architecture

- Input data are transformed through a series of computational or manipulative components into output data

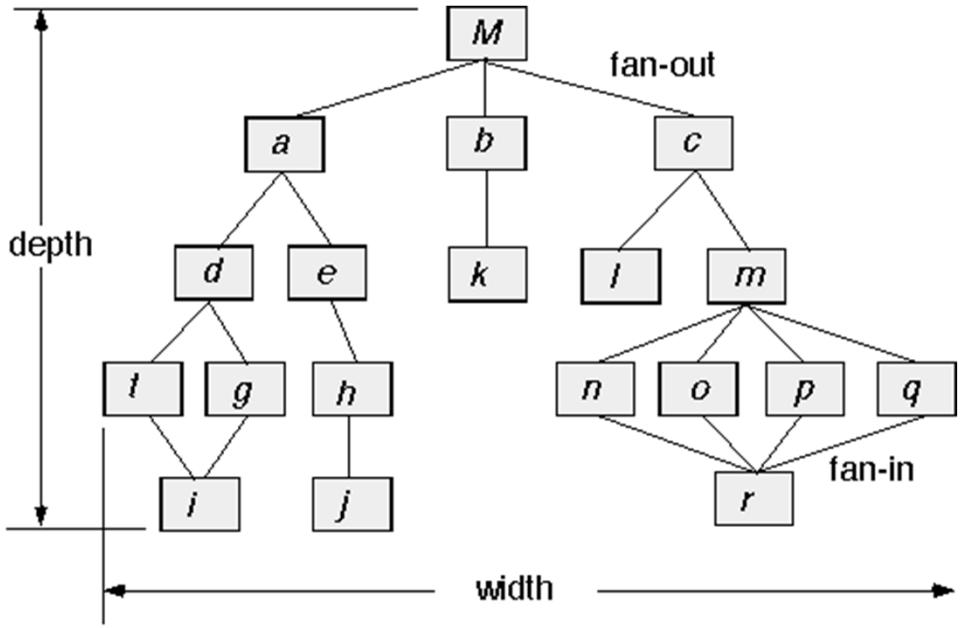


(a) pipes and filters

(b) batch sequential

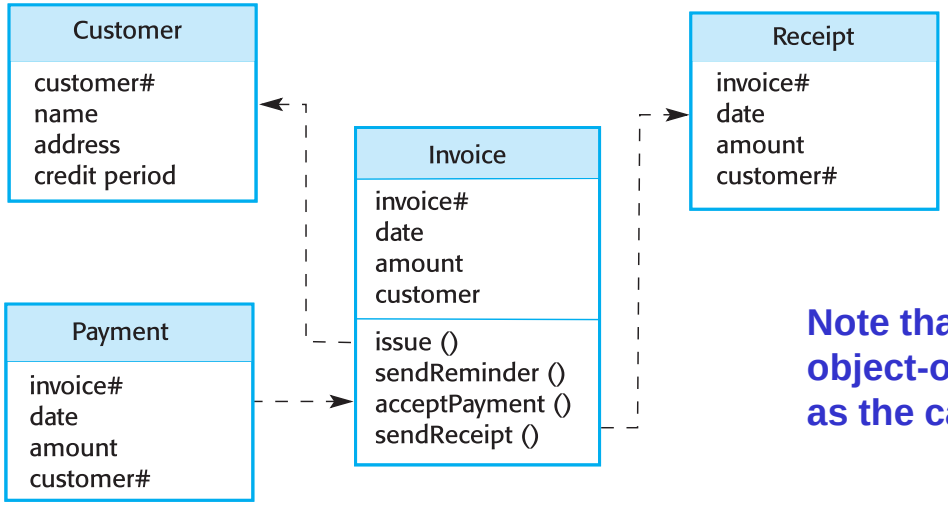
Call and Return Architecture

- This architecture decomposes function into a control hierarchy where a “main” program invokes a number of program components, which in turn may invoke still other components



Object-Oriented Architecture

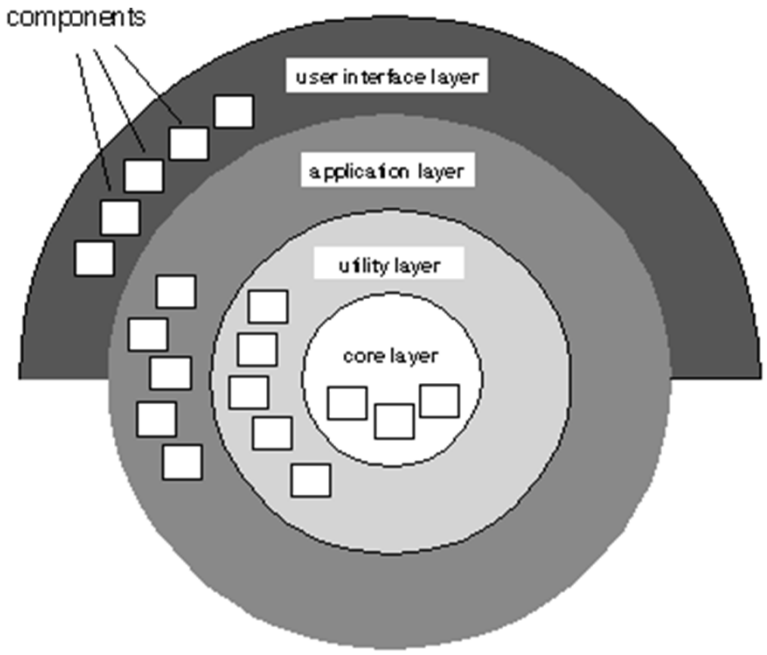
- The components of a system encapsulate data and the operations that must be applied to manipulate data
- Communication and coordination between components is accomplished via message passing



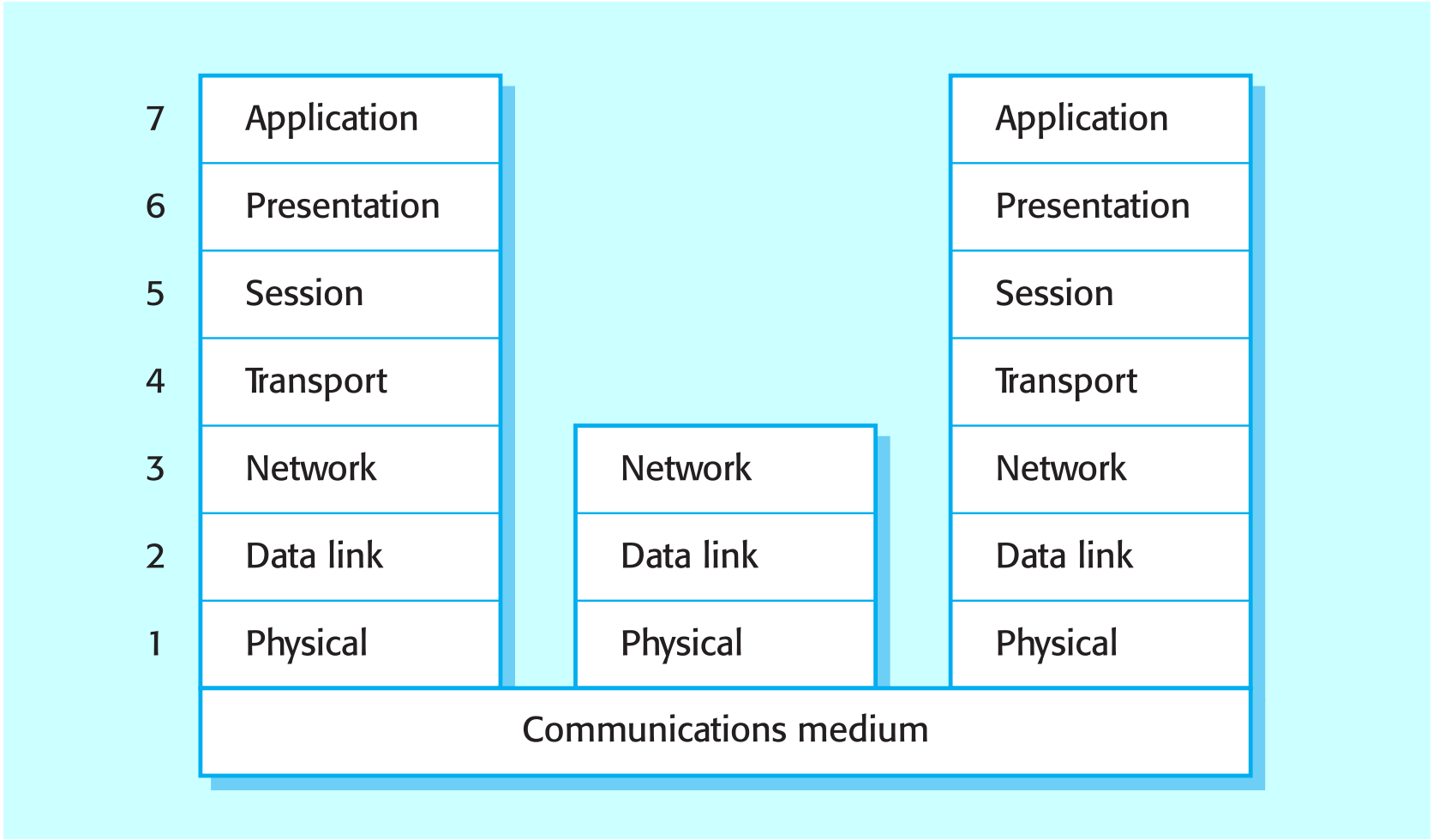
Note that the message passing in the object-oriented architecture can be thought of as the call and return relationship.

Layered Architecture

- A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set



OSI reference model



Contents

1. Architecture
2. Architectural Styles
3. Architectural Design

3. Architectural Design

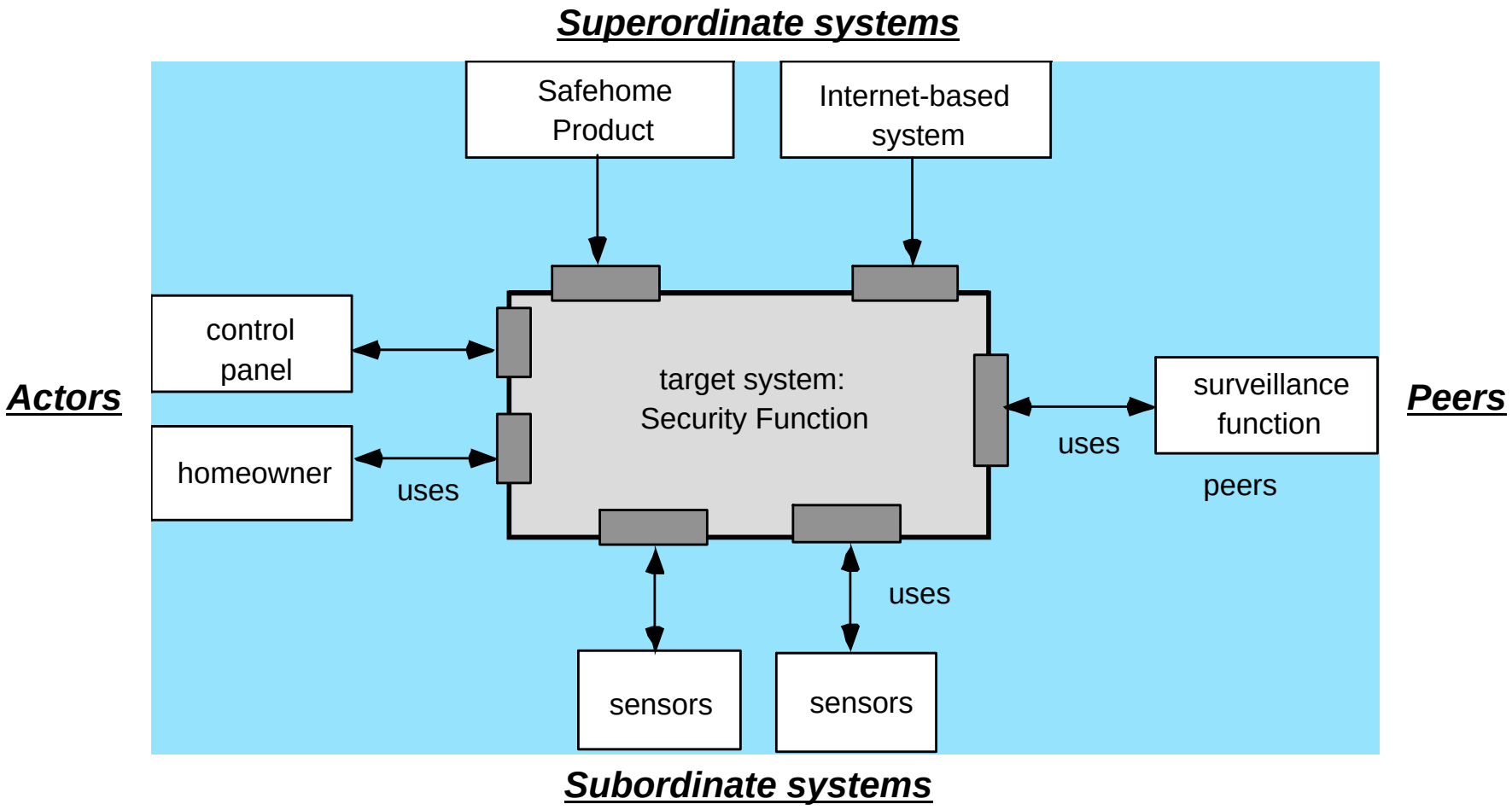
- **Architectural design is to identify sub-systems and establish a structural framework for sub-system control and communications**

- **Architectural design steps**
  1. **Develop an architectural context model**
     - The context model should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction
  2. **Define a set of architectural archetypes**
     - An archetype is an abstraction (similar to a class) that represents one element of system behavior
  3. **Define and refine software components that implement each archetype**

Architectural Context

- **External entities include:**
  - **Superordinate systems**
    - Systems that use the target system as part of some higher level processing scheme
  - **Subordinate systems**
    - Systems that are used by the target system and provide data or processing that are necessary to complete target system functionality
  - **Peer-level systems**
    - Systems that interact on a peer-to-peer basis
  - **Actors**
    - Entities (people, devices) that interact with the target system by producing or consuming information that is necessary for requisite processing

Architecture Context Diagram



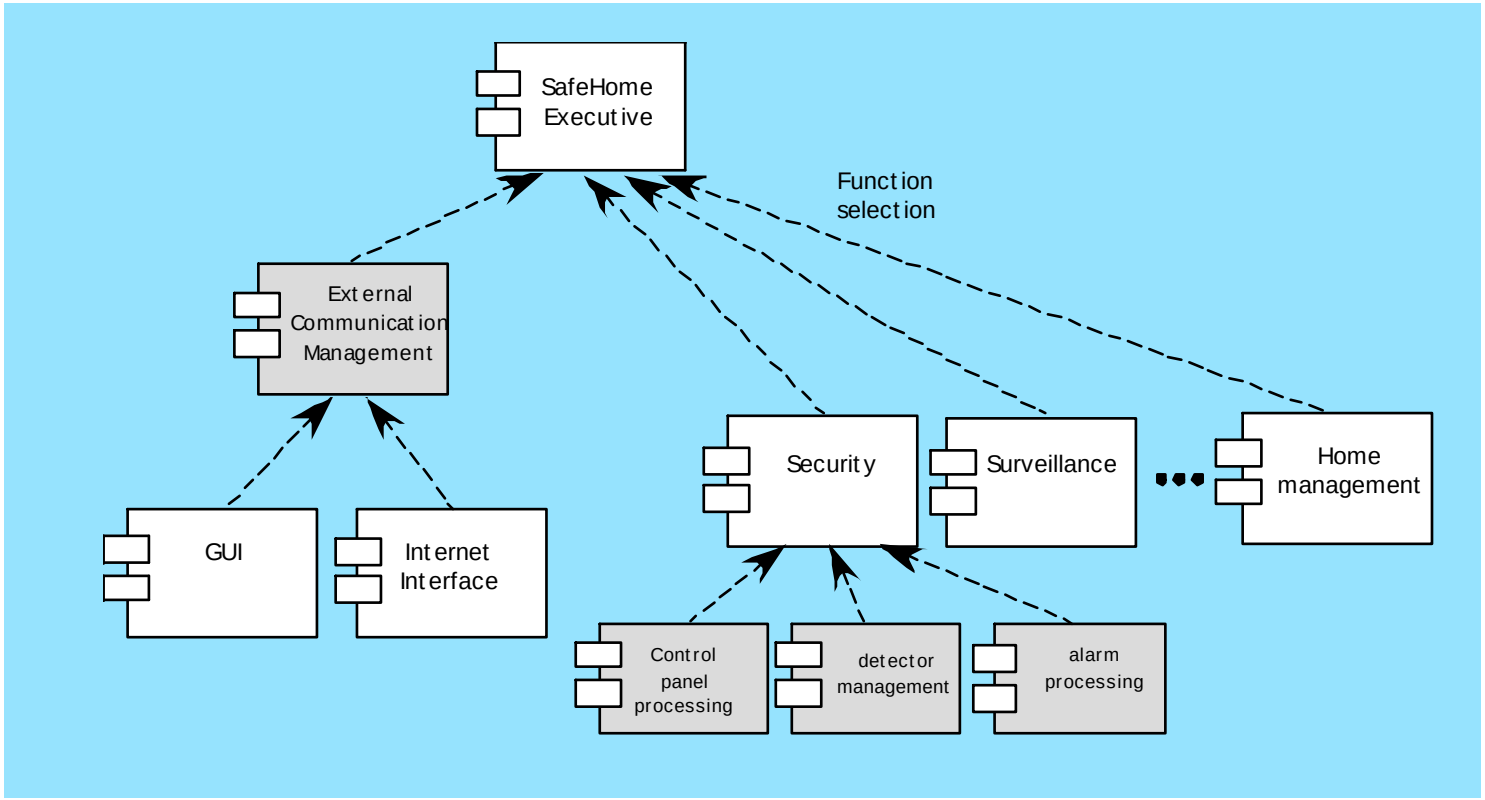
Archetypes

- 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
  - In general, a relatively small set of archetypes is required to design even relatively complex systems
  - In many cases, archetypes can be derived by examining the analysis classes defined as part of the analysis model developed at the requirements engineering phase
  - Note that it is straightforward to derive archetypes for object-oriented models that provides inheritance mechanisms
  - However, it may be difficult to do that for other models that have no inheritance mechanism

Refining the Architecture into Components

- **Components are derived from three sources**
  - **The application domain**
    - The analysis models represent entities within the application domain
  - **The infrastructure domain**
    - The architecture must accommodate many infrastructure components that enable application components but have no business connection to the application domain
    - E.g., memory management components, communication components, database components, …
  - **The interface domain**
    - The interfaces depicted in the architecture context diagram imply one or more specialized components that process data that flow across the interface
    - E.g., a graphical user interface

The Overall Architecture of SafeHome with Top-level Components



Refinement through Instantiations

- By an actual instantiation, we mean that the architecture is applied to a specific problem with the intent of demonstrating that the structure and components are appropriate
  - Therefore, instantiations can help refine the architecture
  - Elaboration can be performed for each of the components of the overall architecture

An Instantiation of the Security Function with Component Elaboration