DESIGN ENGINEERING

1. Commences after 1st iteration of Requirements Analysis
2. GOAL : To create design model that will implement all customer requirements correctly to his satisfaction.
3. INTENT: To develop High Quality Software by applying a set of principles, concepts and practices
4. COMPONENTS:
   - Data Structures Design
   - Architectural Design
   - Interface Design
   - Component Level Design

DESIGN MODELING PRINCIPLES

Traceability to analysis model

- Consider architecture of the system
- Data design as important as processing
- Interfaces (Int and Ext) must be designed
- HCI as per needs of end user
- Functionally Independent component design
- Low coupling and high cohesion
- KIS – easily understandable
- Design iterations for greater simplicity

Design Principles

The design process should not suffer from ‘tunnel vision.’
The design should be traceable to the analysis model.
The design should not reinvent the wheel.
The design should “minimize the intellectual distance” [DAV95] between the software and the problem as it exists in the real world.
The design should exhibit uniformity and integration.
The design should be structured to accommodate change.
The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.
Design is not coding, coding is not design.
The design should be assessed for quality as it is being created, not after the fact.
The design should be reviewed to minimize conceptual (semantic) errors.

Design and Quality

- **the design must implement all of the explicit requirements** contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- **the design must be a readable, understandable guide** for those who generate code and for those who test and subsequently support the software.
- **the design should provide a complete picture of the software**, addressing the data, functional, and behavioral domains from an implementation perspective.

Quality Guidelines

- **A design should exhibit an architecture** that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion
  - For smaller systems, design can sometimes be developed linearly.
- **A design should be modular;** that is, the software should be logically partitioned into elements or subsystems
- **A design should contain distinct representations** of data, architecture, interfaces, and components.
- **A design should lead to data structures that are appropriate** for the classes to be implemented and are drawn from recognizable data patterns.

Quality Guidelines – contd.

- **A design should lead to components that exhibit independent functional characteristics.**

- **A design should lead to interfaces that reduce the complexity** of connections between components and with the external environment.

- **A design should be derived using a repeatable method** that is driven by information obtained during software requirements analysis.

- **A design should be represented using a notation that effectively communicates its meaning**.

Conclusion

- Satisfy all customer requirements using standard methods, must be completer for next stage

Quality Attributes (FURPS)

- **Functionality** : Feature Sets, Security
- **Usability** : easy to use, overall aesthetics, consistency, documentation
- **Reliability** : frequency & severity of failure
- **Performance** : time and space complexity
- **Supportability** : extensibility, adaptability, serviceability, maintainability, compatibility, configurability

Design Concepts

- **abstraction** : data, procedure, control
- **architecture** : the overall structure of the software
- **patterns** : "conveys the essence" of a proven design solution
- **modularity** : compartmentalization of data and function
- **information hiding :** controlled interfaces
- **functional independence :** high cohesion and low coupling
- **refinement** : elaboration of detail for all abstractions
- **refactoring** : improve design without effecting behavior

Patterns

Design Pattern Template

Pattern name—describes the essence of the pattern in a short but expressive name

Intent—describes the pattern and what it does

Also-known-as—lists any synonyms for the pattern

Motivation—provides an example of the problem

Applicability—notes specific design situations in which the pattern is applicable

Structure—describes the classes that are required to implement the pattern

Participants—describes the responsibilities of the classes that are required to implement the pattern

Collaborations—describes how the participants collaborate to carry out their responsibilities

Consequences—describes the "design forces" that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented

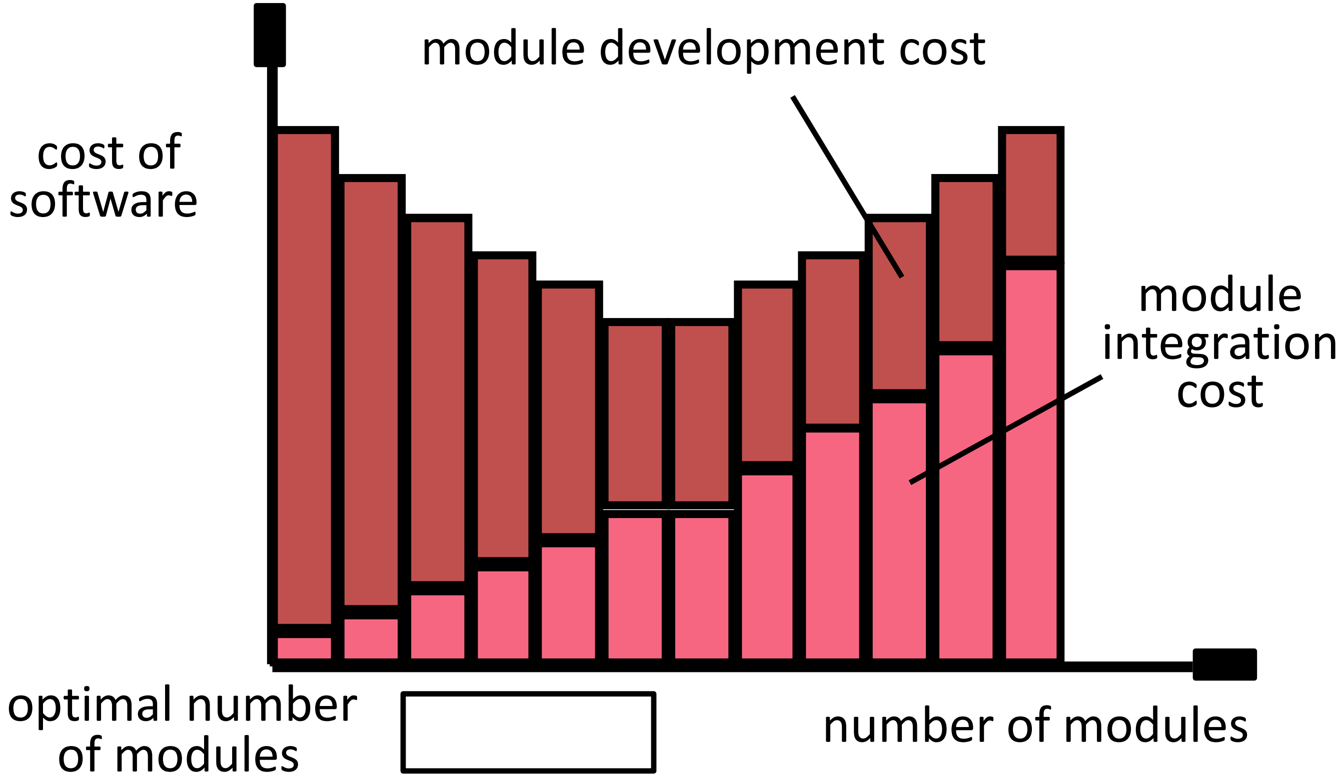
***Related patterns*—cross-references** related design patterns

Roadmap for Design Principles

- Modularization
- Cohesion and Coupling
- KISS
- DRY and WET ##

Modularity: Trade-offs

What is the "right" number of modules for a specific software design?



Cohesion and Coupling

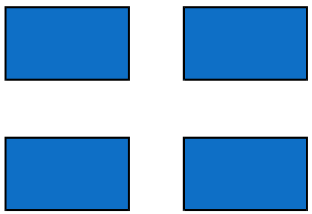
- Cohesion is a measure of:
  - functional strength of a module.
  - A cohesive module performs a single task or function.

- Coupling between two modules:
  - a measure of the degree of interdependence or interaction between the two modules. ##

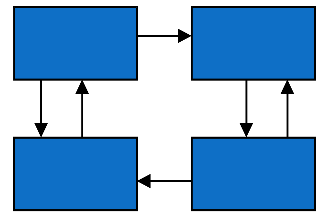
Cohesion

- Definition: The degree to which all elements of a component are directed towards a single task and all elements directed towards that task are contained in a single component.
- Internal glue with which component is constructed
- All elements of component are directed toward and essential for performing the same task
- High is good ##

Coupling: Degree of dependence among components



No dependencies



Loosely coupled-some dependencies

Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Characteristics of Good Design

- Component independence
  - High Cohesion
  - Low Coupling

K.I.S.S. Design Principle

- Keep It Simple Stupid
- Keep It Short and Simple
- Keep It Simple and Straight Forward
  ##

DRY

Do not Repeat Yourself

Root cause: Needless Repetition

- The design contains repeating structures that could be unified under a single abstraction
- The problem is due to developer's abuse of cut and paste.
- It is really hard to maintain and understand the system with duplicated code.

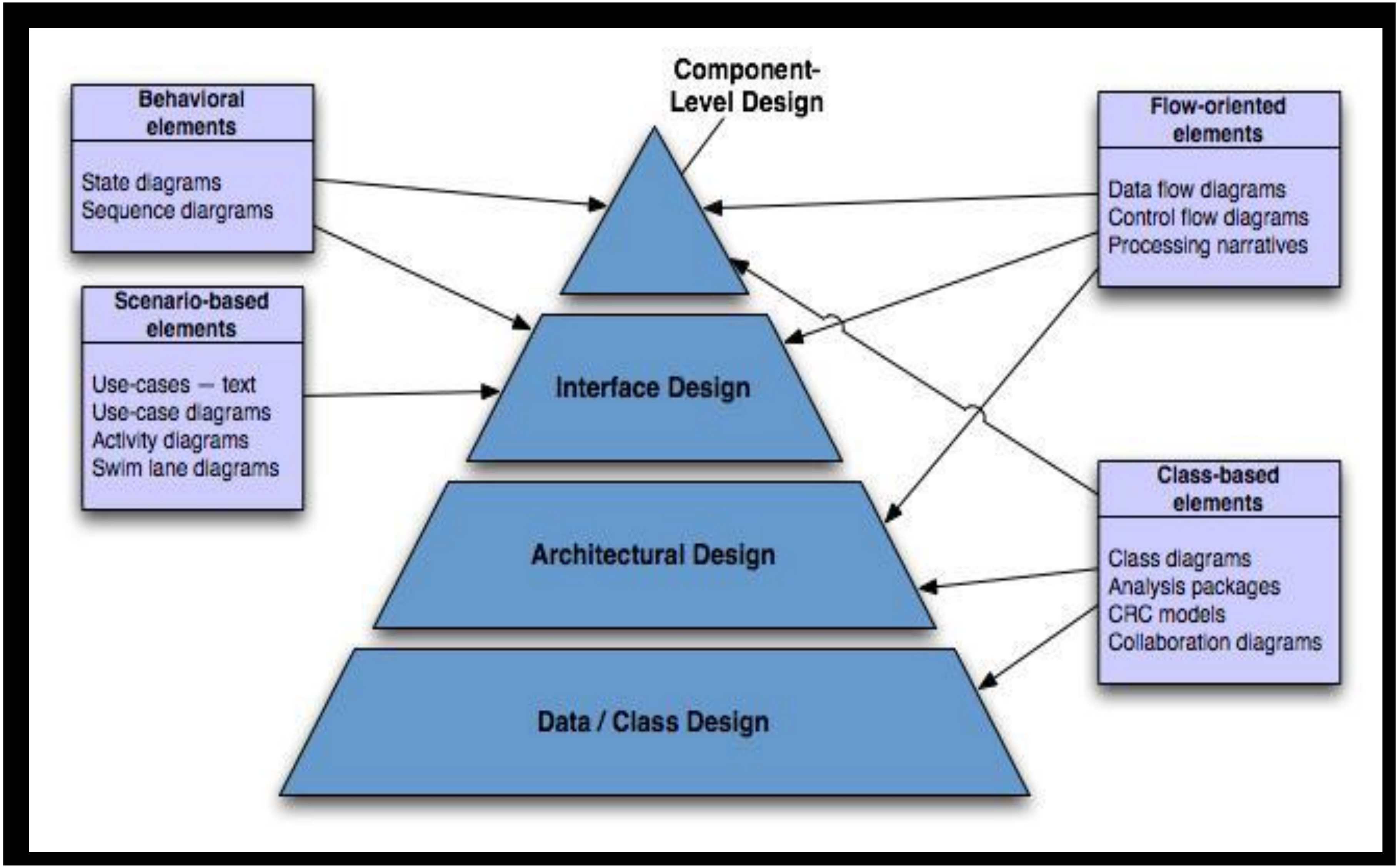
DRY Design Principle

- **Don't Repeat Yourself** (**DRY**) is a principle of software development aimed at reducing repetition of information of all kinds, especially useful in multi-tier architectures
- The principle has been formulated by Andy Hunt and Dave Thomas in their book “The Pragmatic Programmer”
- When the DRY principle is applied successfully, a modification of any single element of a system does not require a change in other logically unrelated elements.

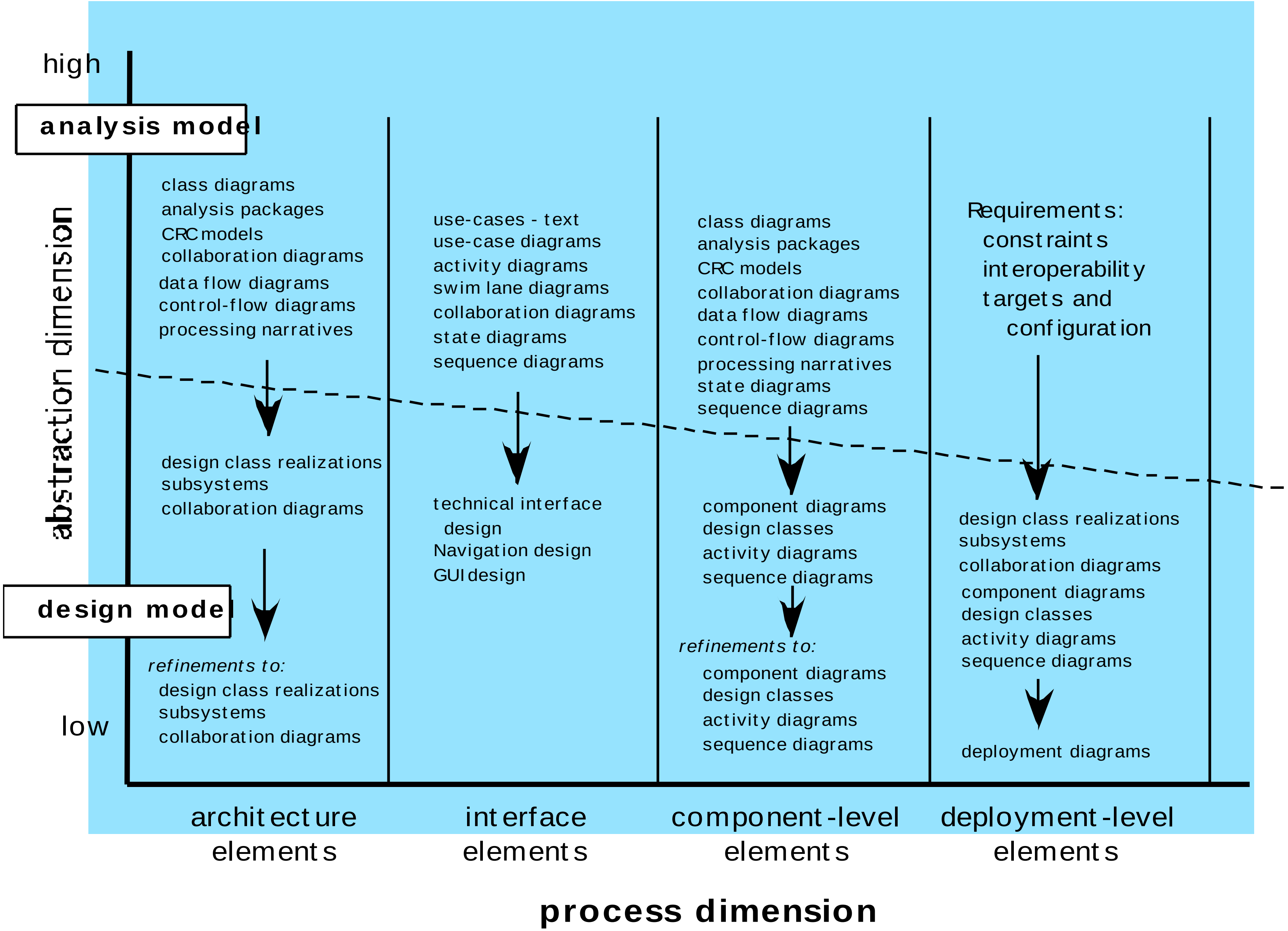
- Violations of DRY are typically referred to as **WET** solutions, which stands for "**Write Everything Twice**".

- Design Model is viewed in 2 dimension
- A)Abstraction – level of detail
- B)Process-a task in Process model

Analysis → Design



The Design Model



Design Model Elements

- **Data elements**
  - –Architectural level → databases and files
  - –Component level → data structures
- **Architectural elements**
  - –An architectural model is derived from:
    - Application domain
    - Analysis model
    - Available styles and patterns
- **Interface elements**
  - –There are three parts to the interface design element:
  - –The user interface (UI)
  - –Interfaces to external systems
  - –Interfaces to components within the application
- **Component elements**
- **Deployment elements**

Interface Elements

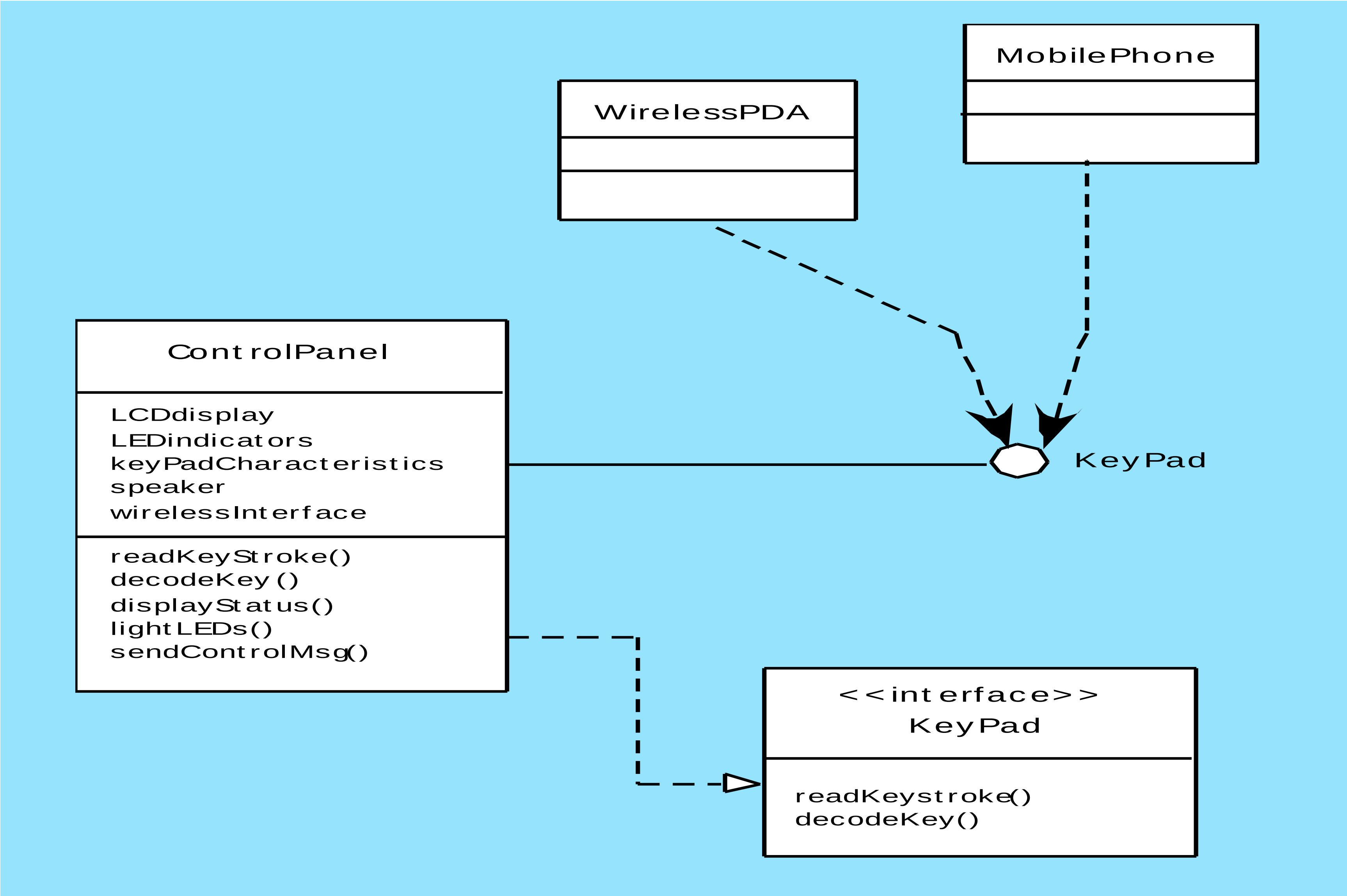
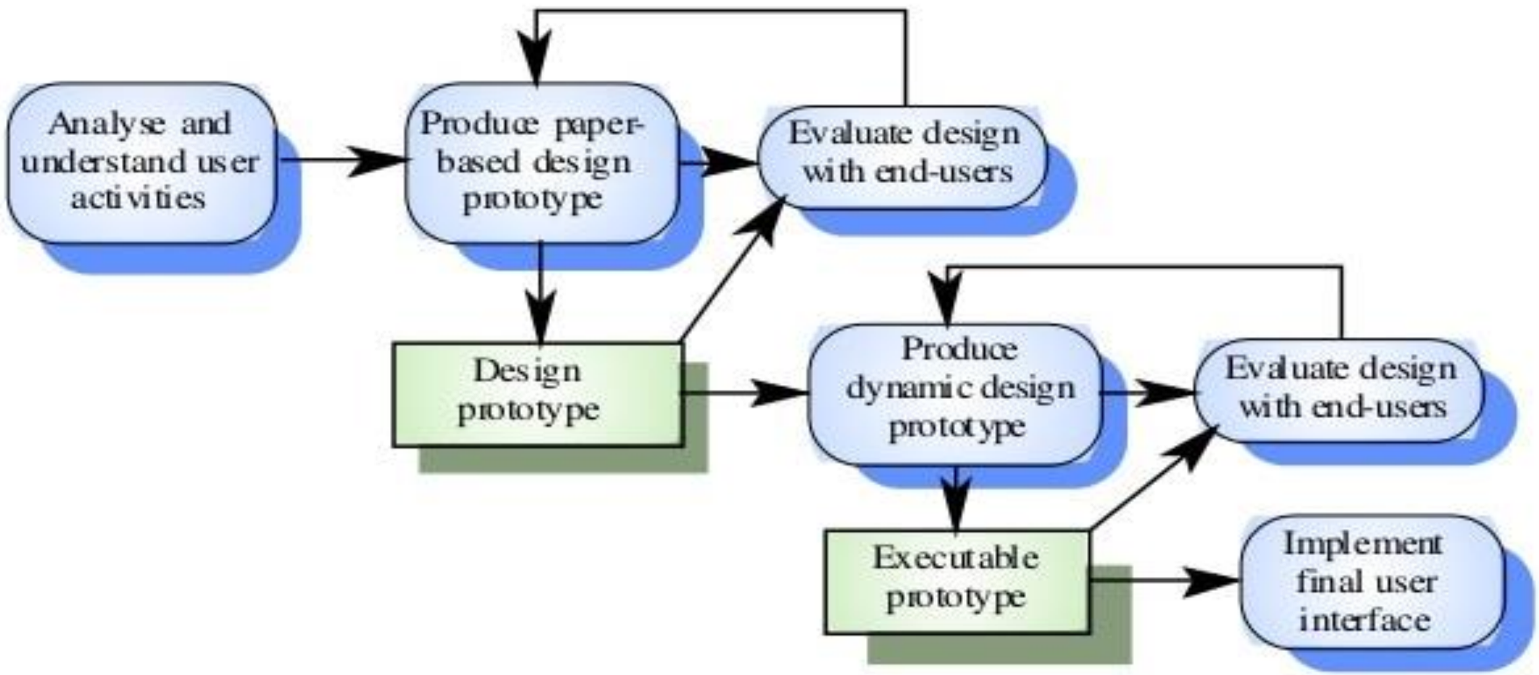


Figure 9.6 UML interface representation for **ControlPanel**

User interface design process



Importance of Interface Design

- Why ? Identify user, task, environmental requirements for better communication ,use
- Who ? Software Engineer , Designer
- Output : Prototype

Why Architecture?

- Architecture is a representation of a system that enables the 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.

Data Design

- Architectural level $\rightarrow$ Database design
  - data mining
  - data warehousing
- Component level $\rightarrow$ Data structure design

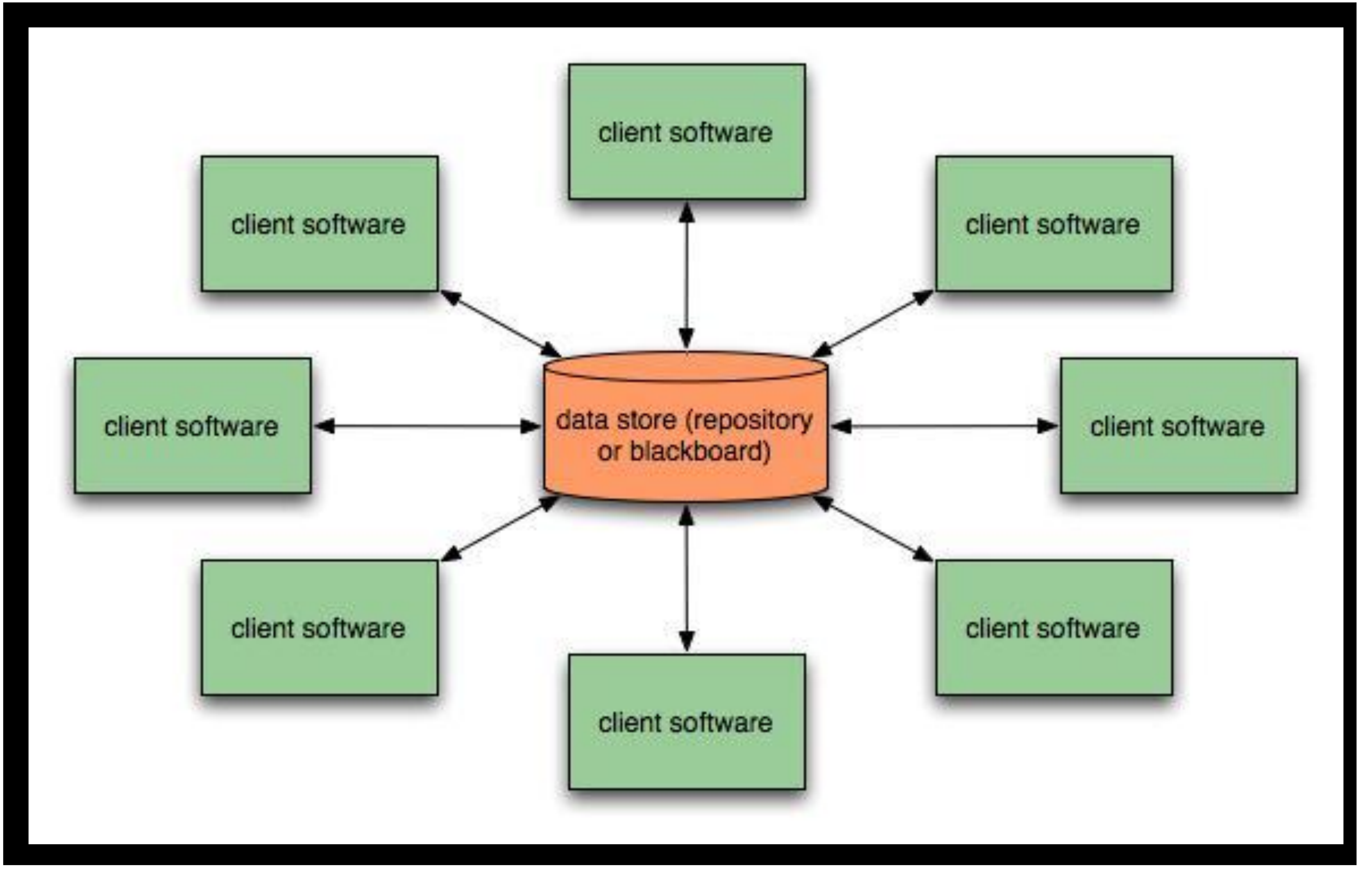
Architectural Styles

- **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.

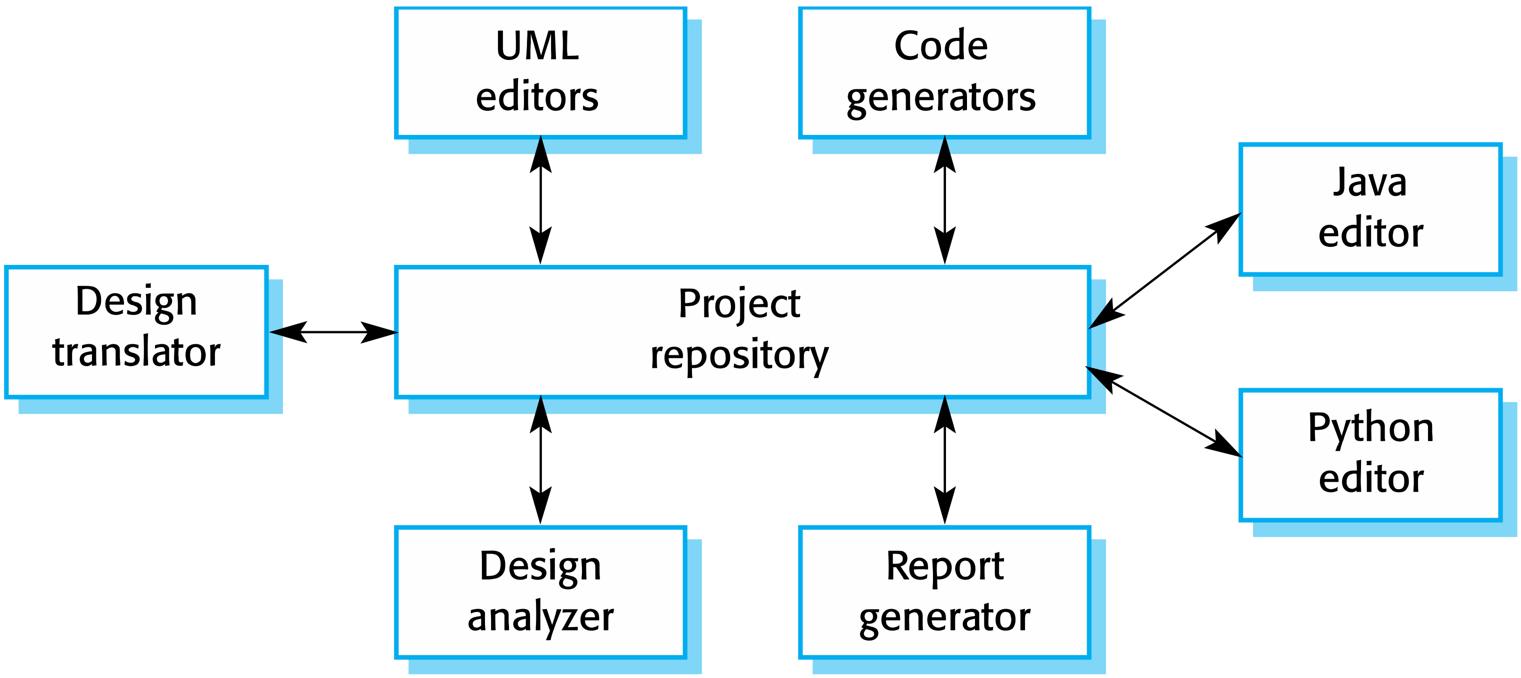
Specific Styles

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

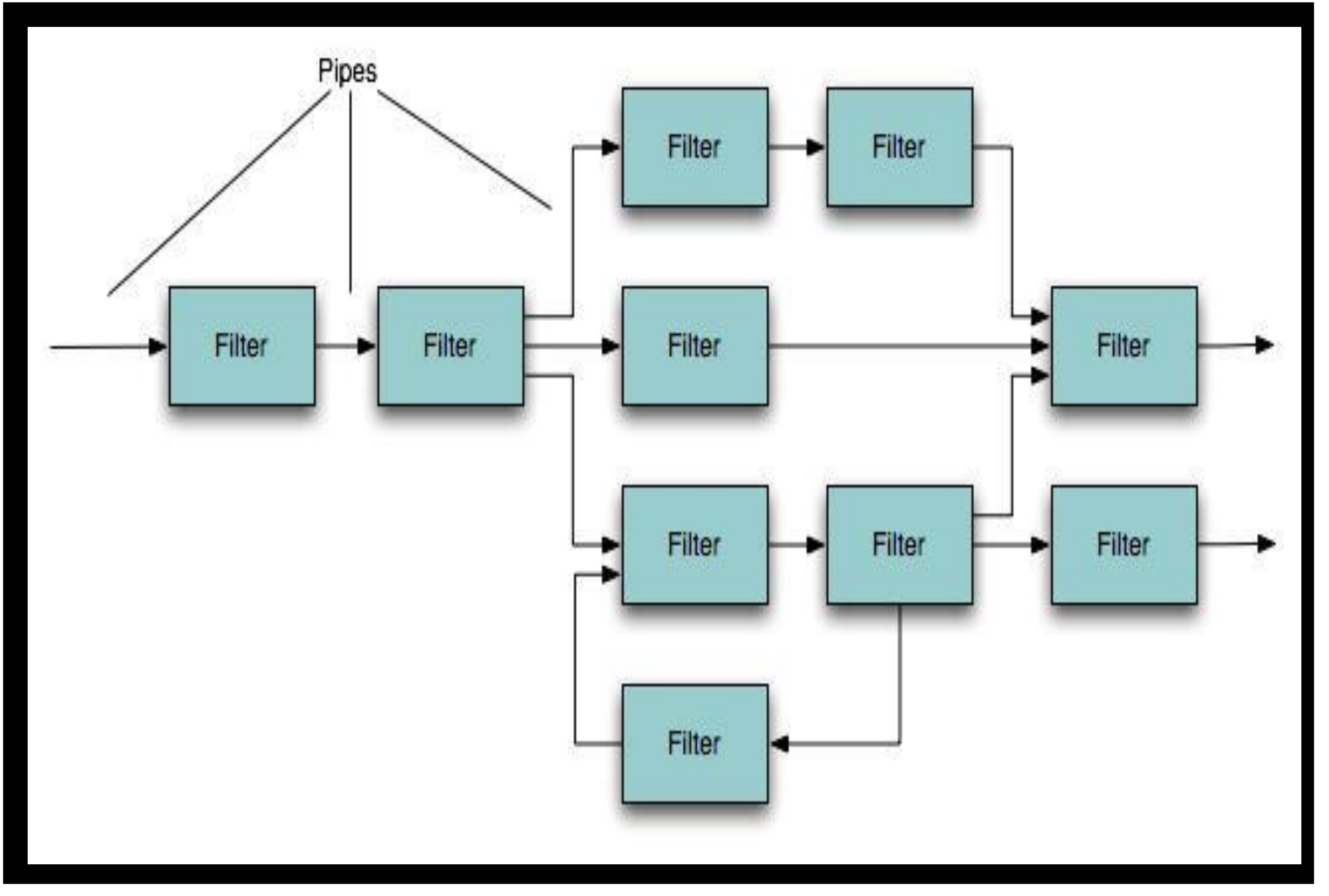
Data-Centered Architecture



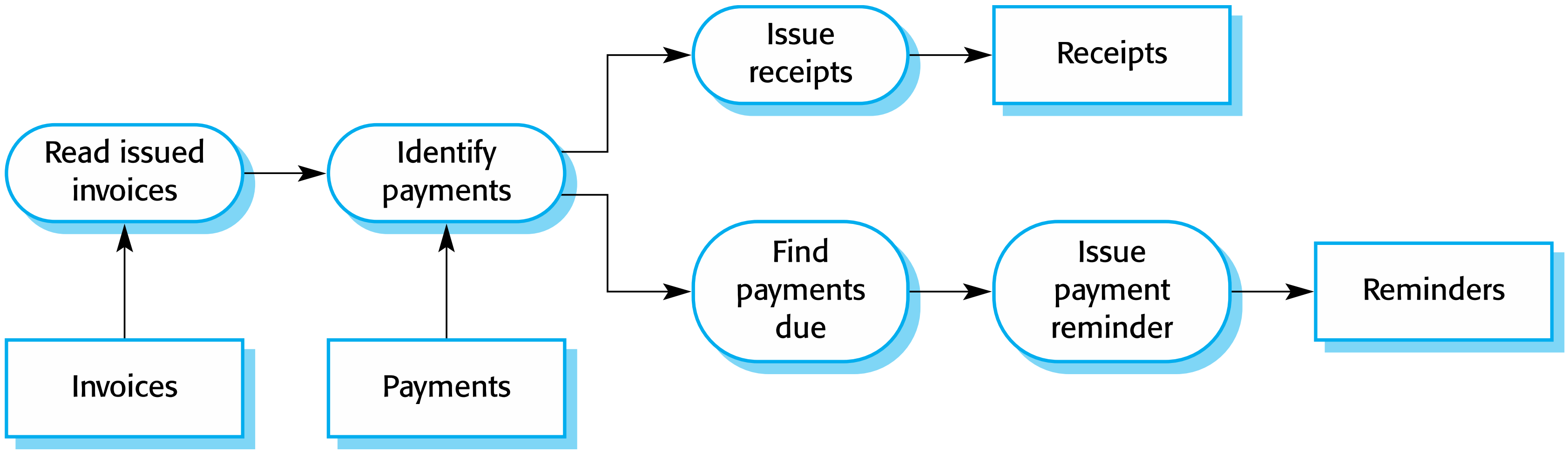
A repository architecture for an IDE



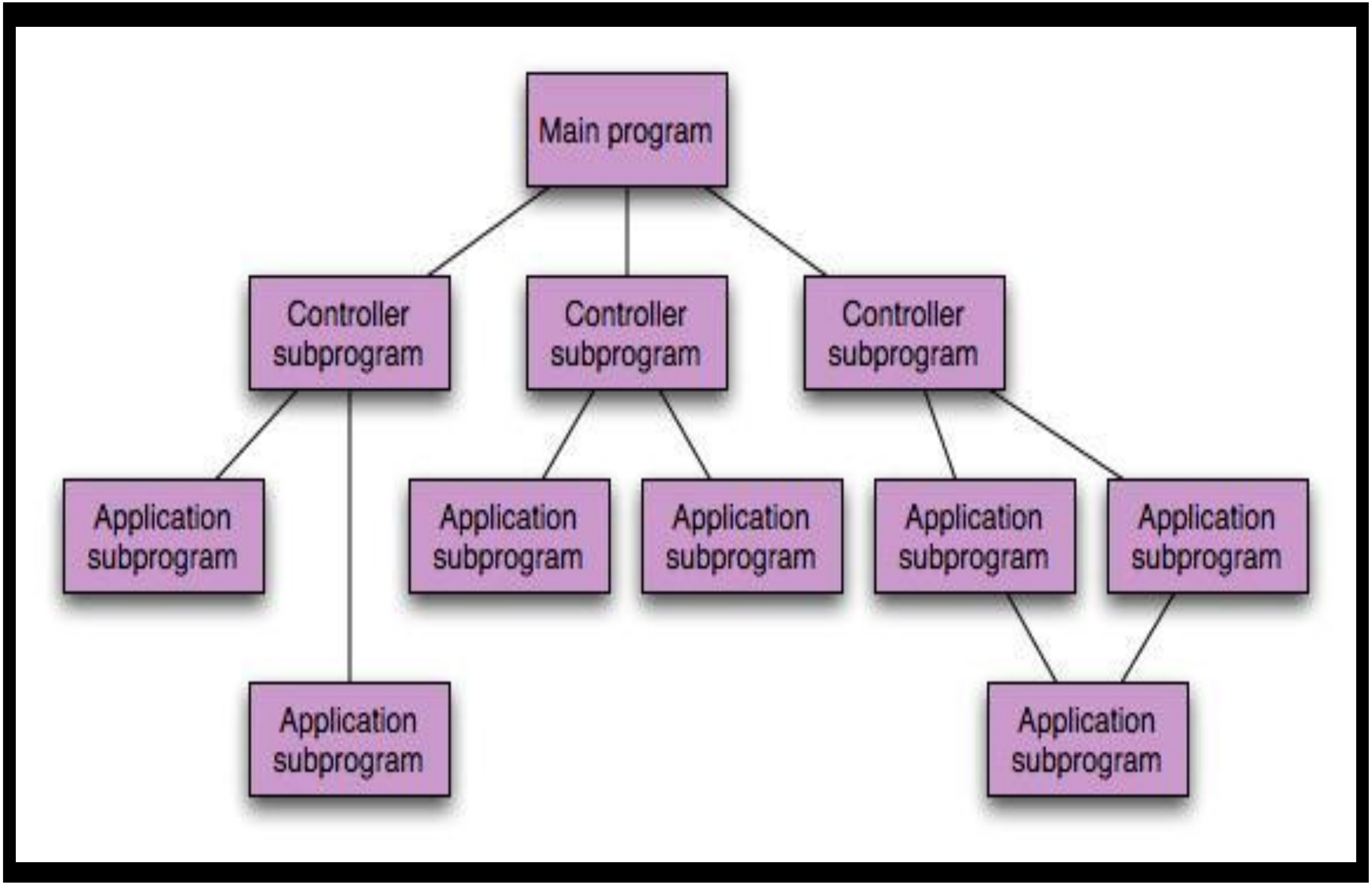
Data-Flow Architecture



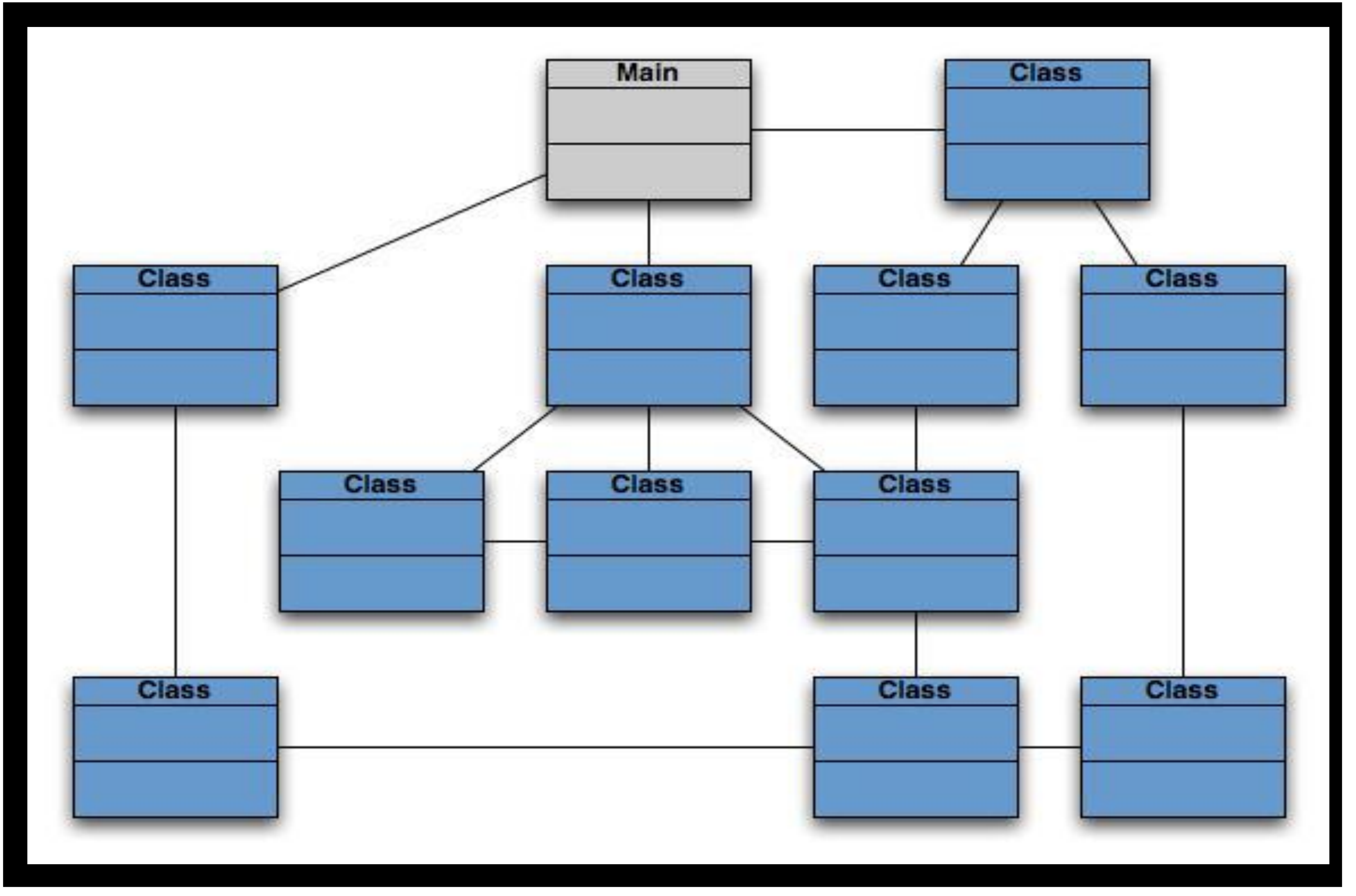
An example of the pipe and filter architecture used in a payments system



Call and Return Architecture



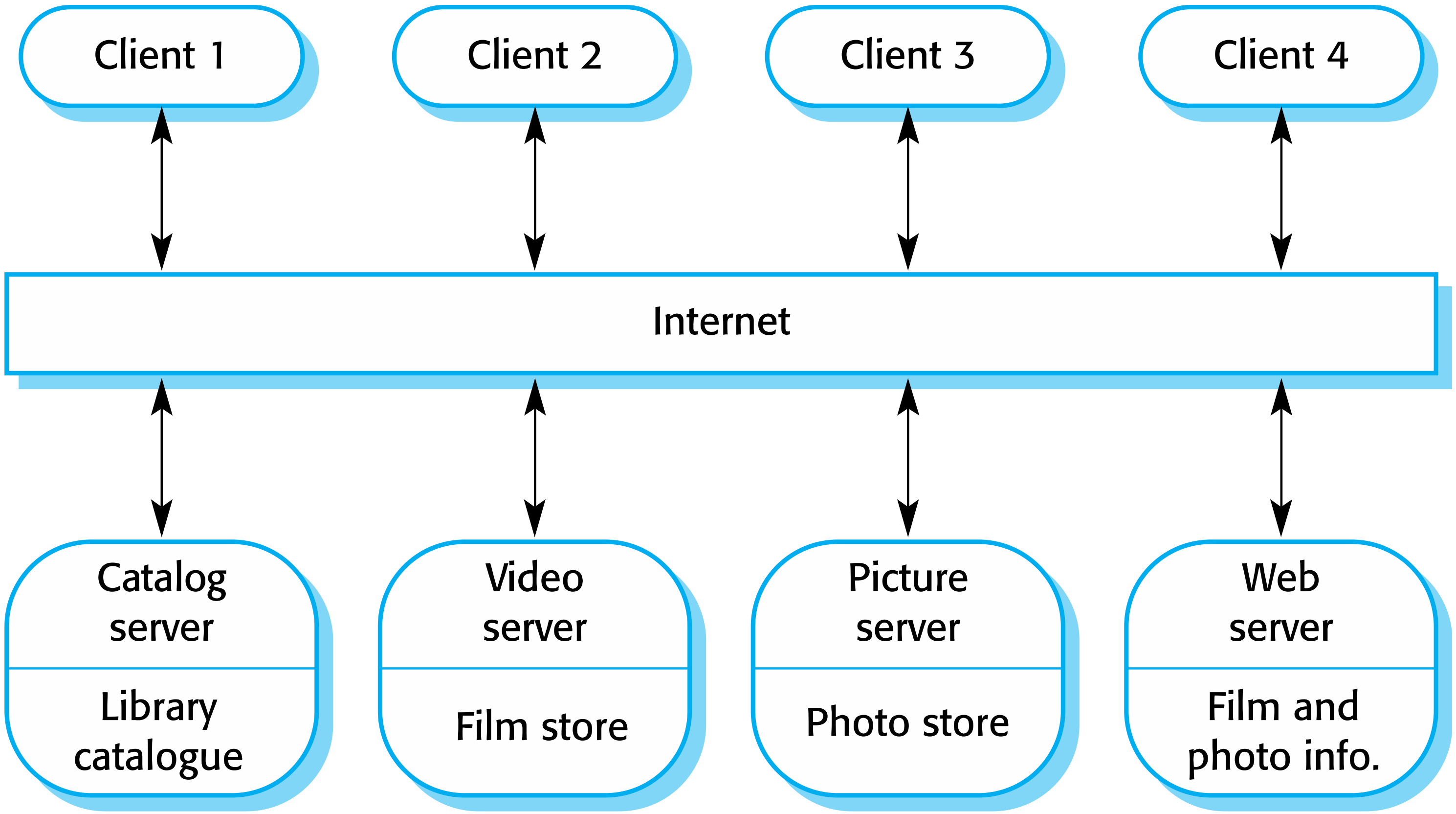
Object-Oriented Architecture



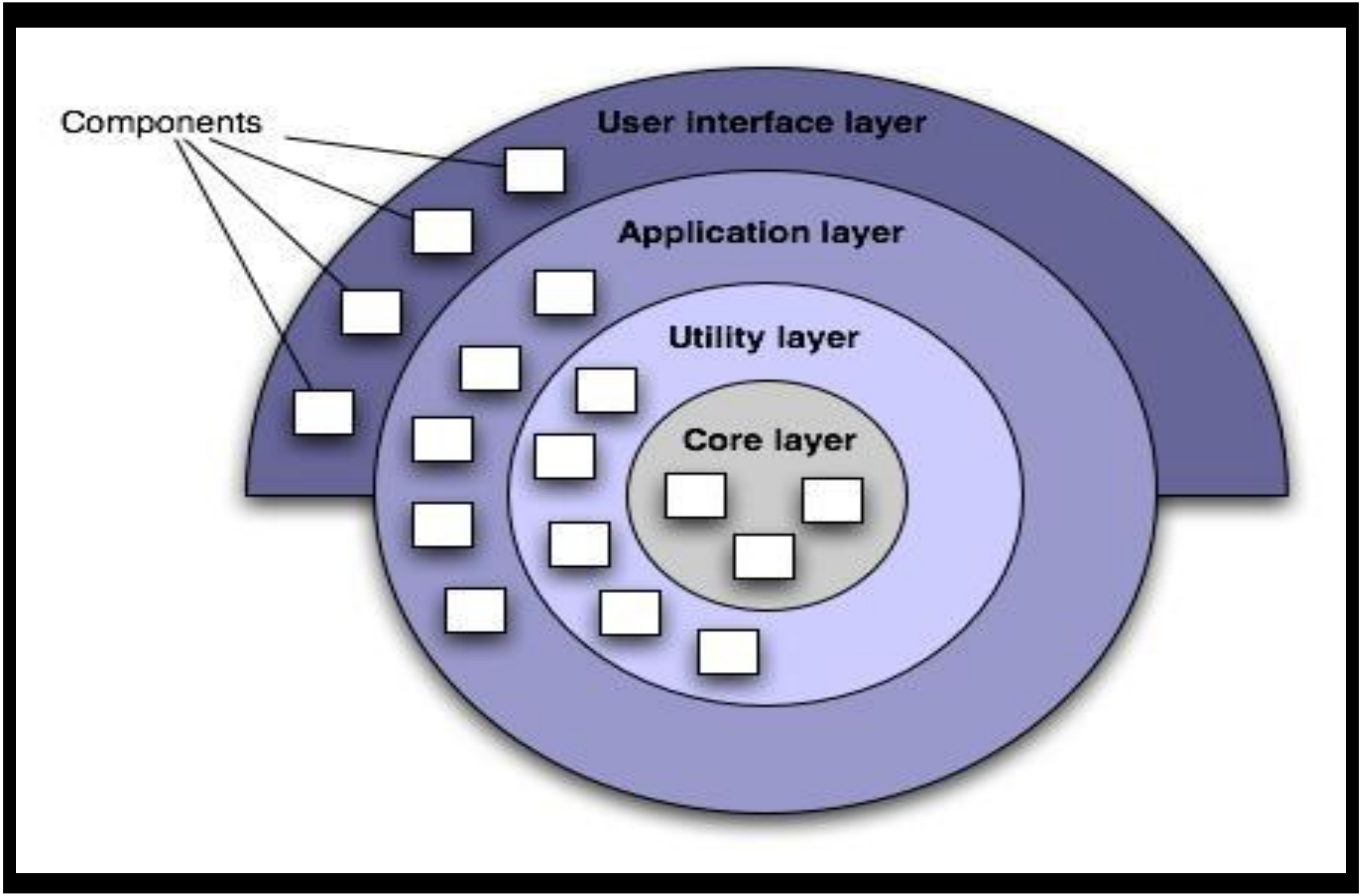
Client-server architecture

- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can be implemented on a single computer.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

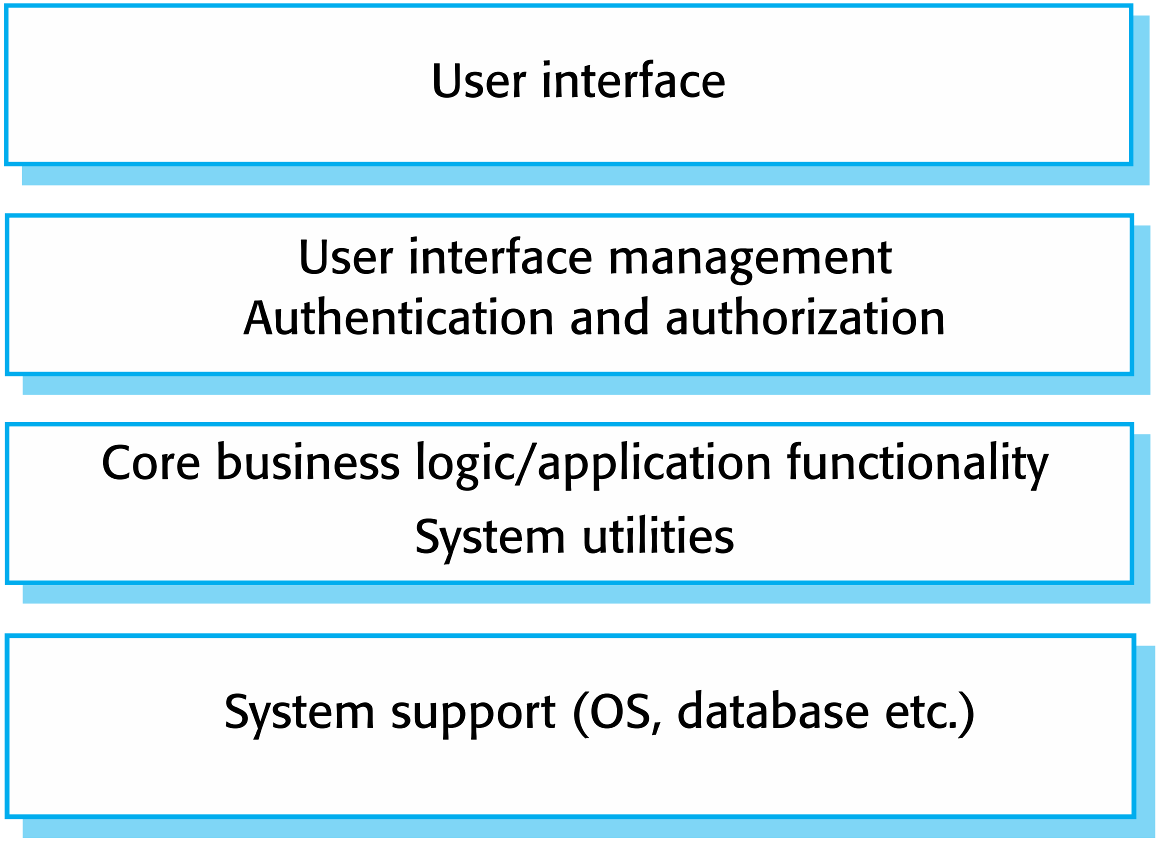
A client–server architecture for a film library



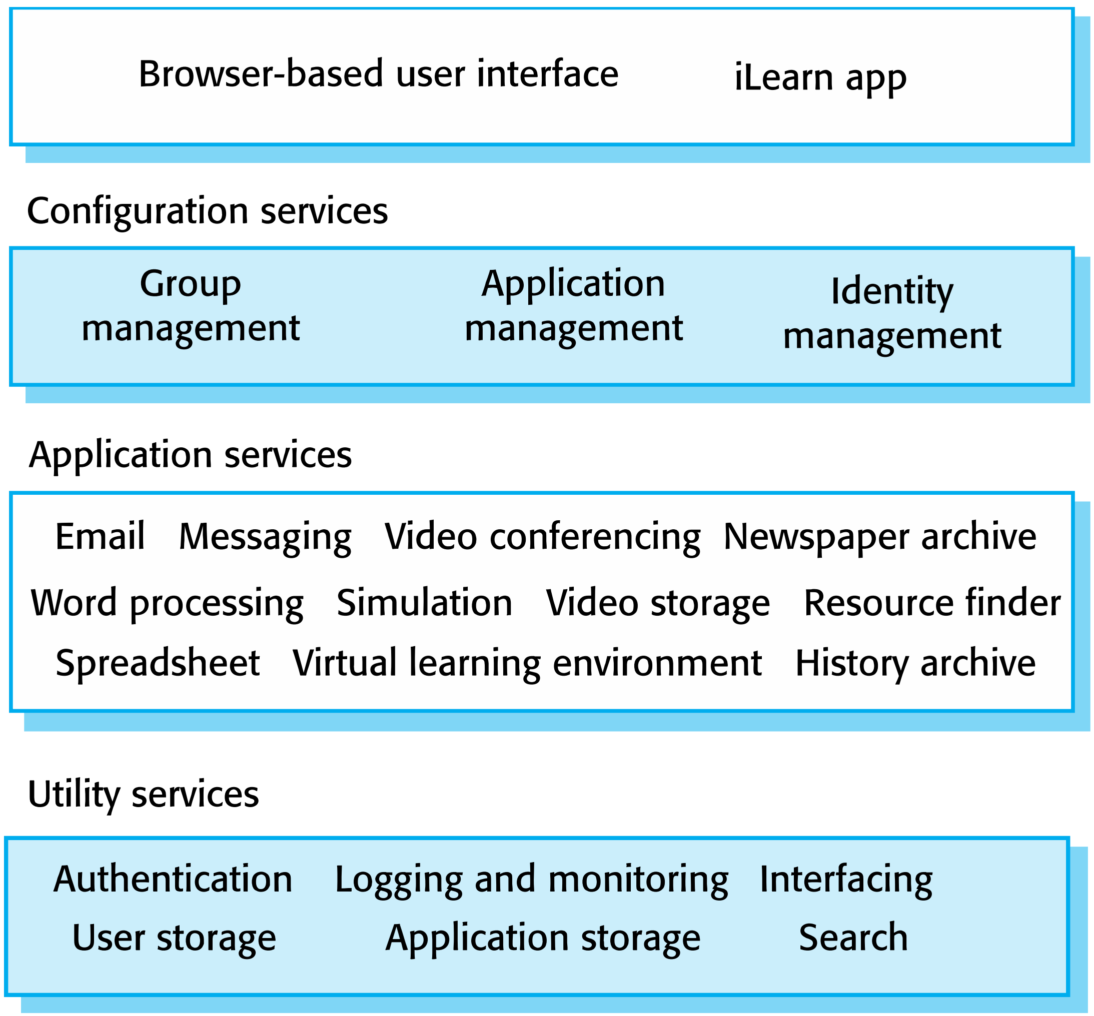
Layered Architecture



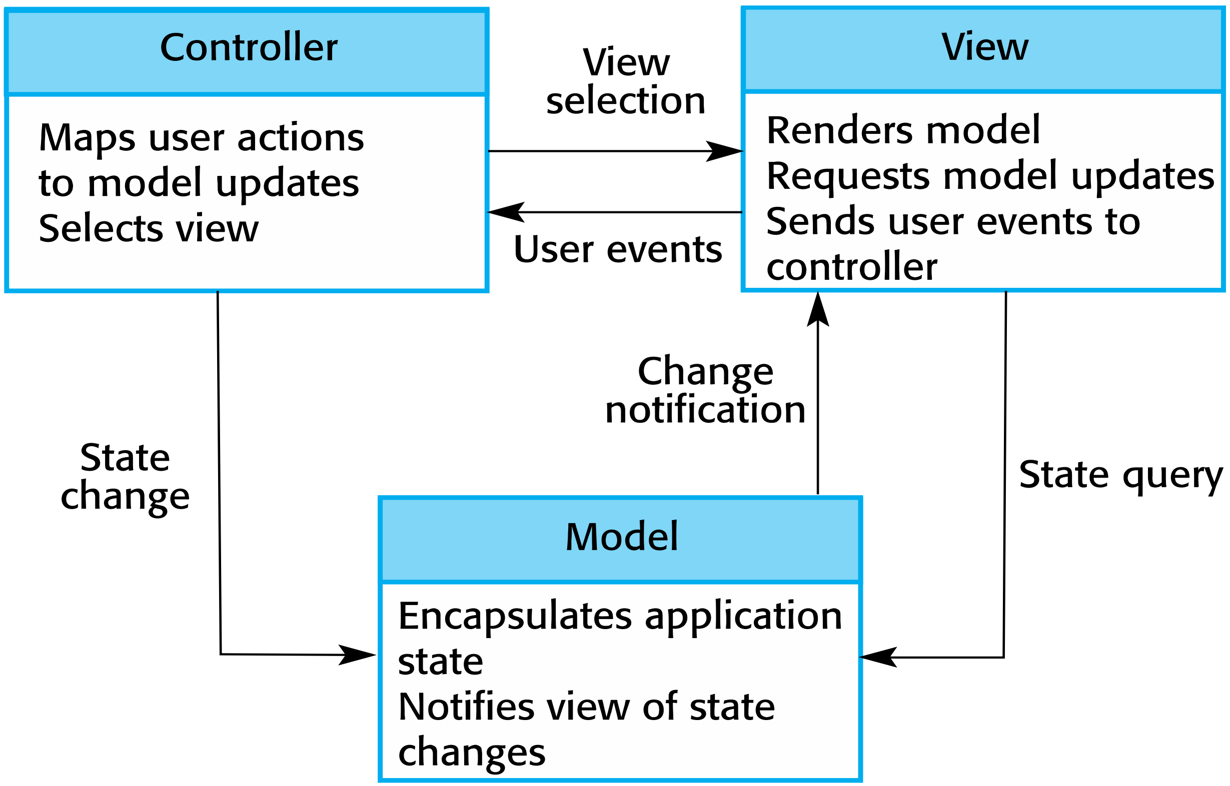
A generic layered architecture



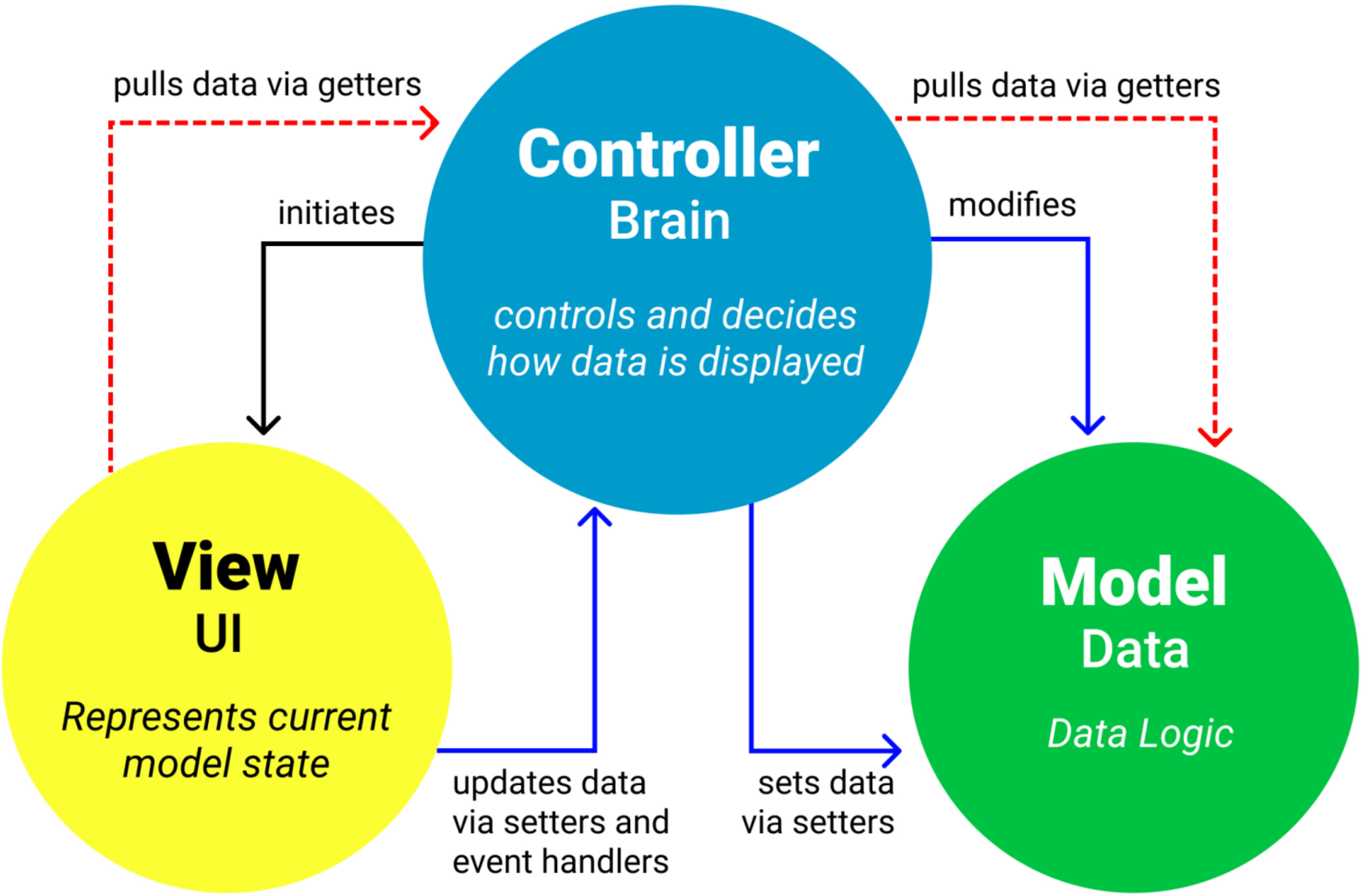
The architecture of the iLearn system



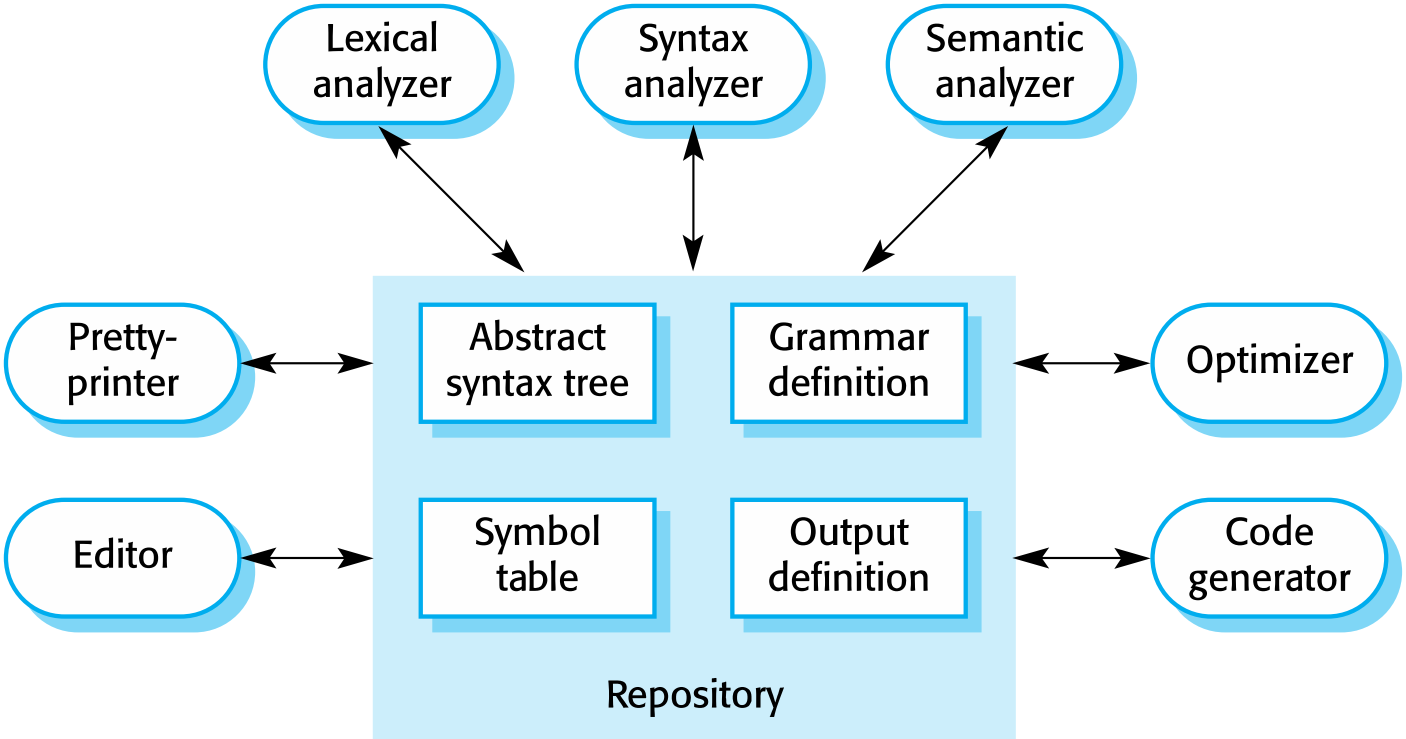
The organization of the Model-View-Controller



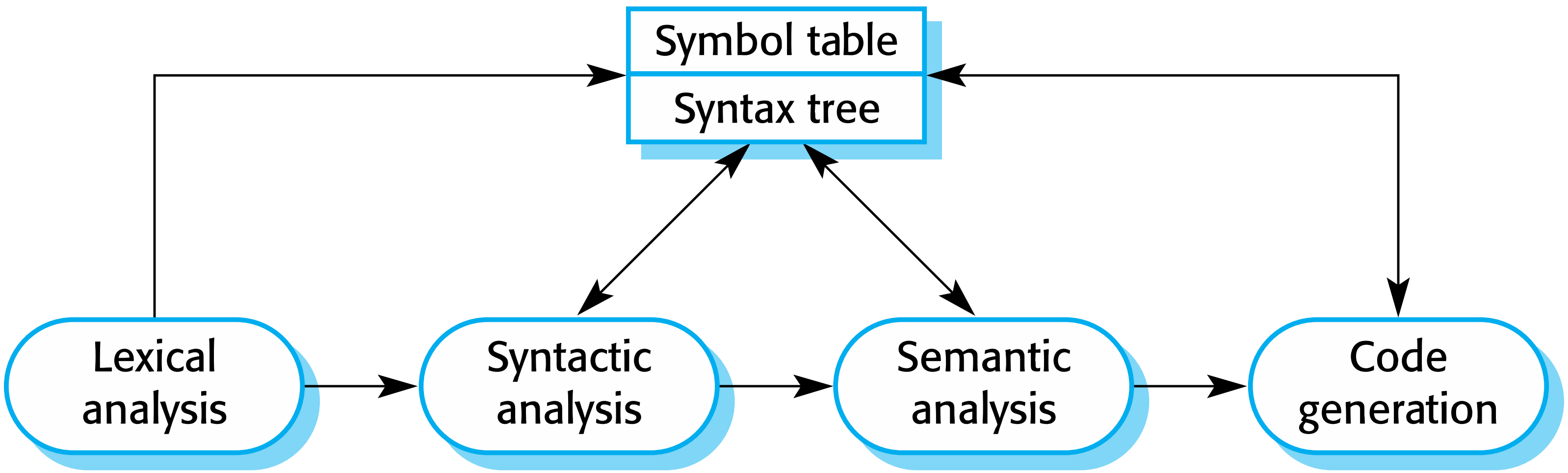
MVC Architecture Pattern



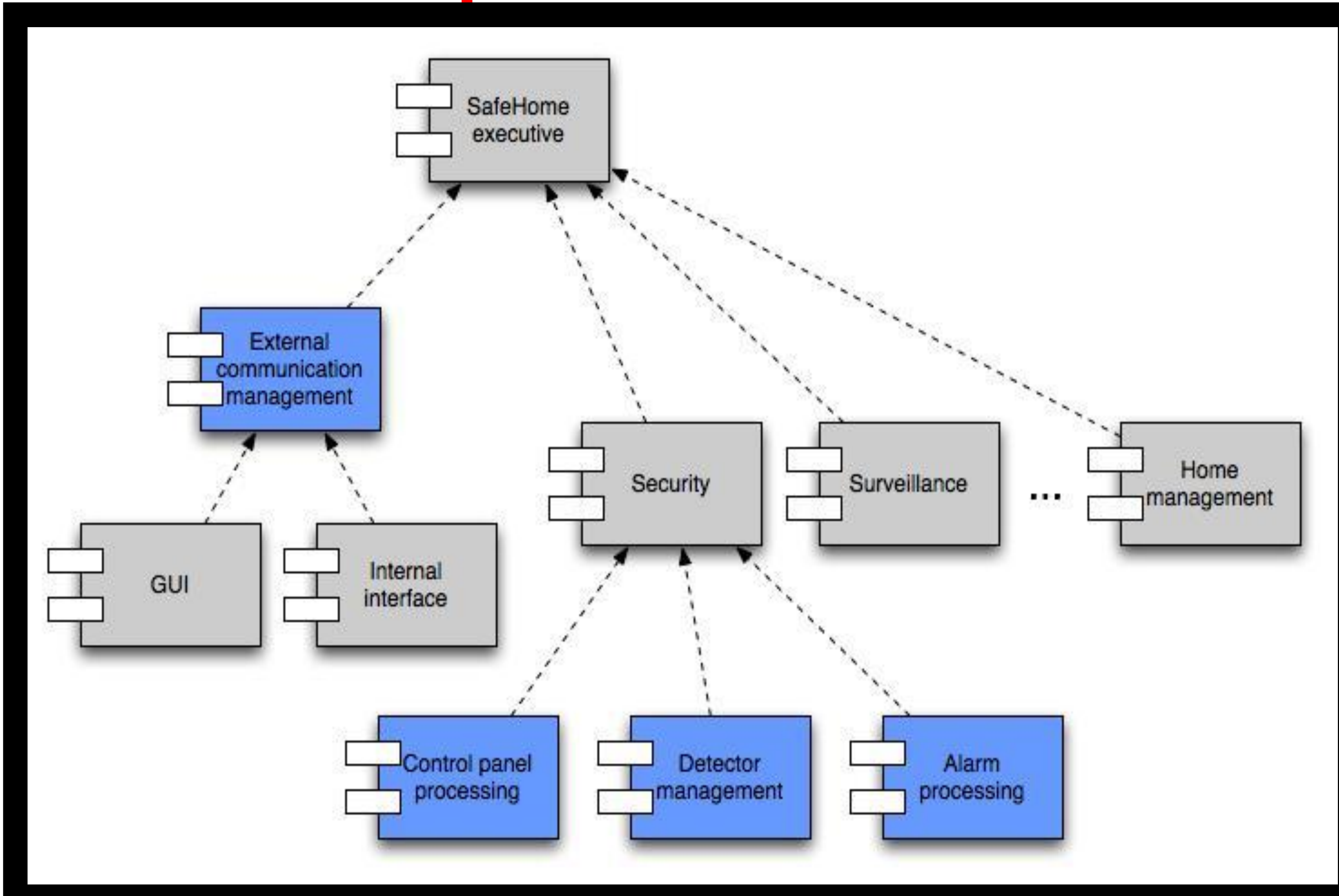
A repository architecture for a language processing system



A pipe and filter compiler architecture



Component Structure



Deployment Diagram