POAD Book:
Chapter 4: Design Patterns as Components
Chapter 5: Visual Design Models

Instructor: Dr. Hany H. Ammar

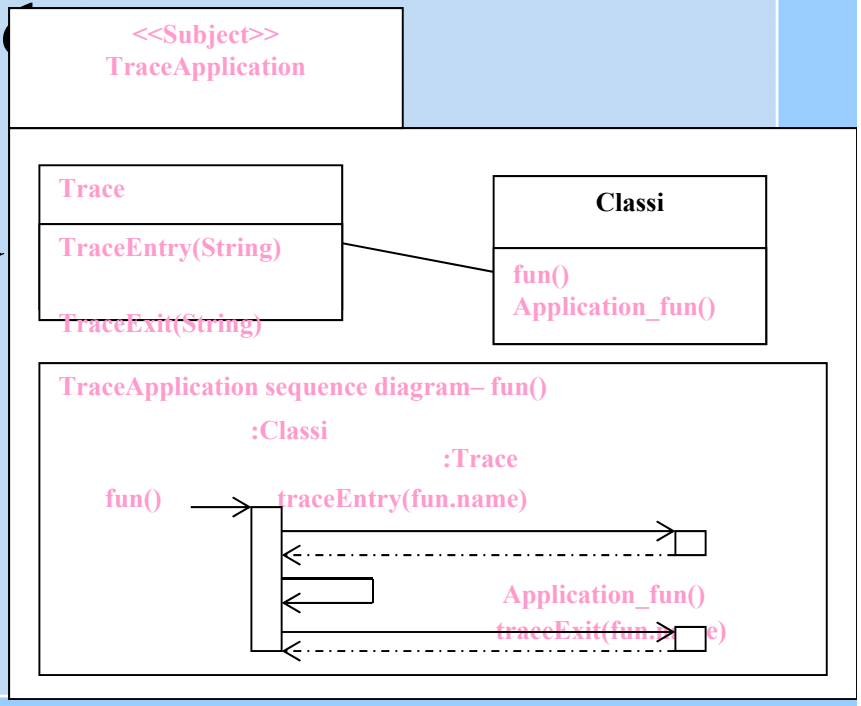
Dept. of Computer Science and
Electrical Engineering, WVU

Outline

- Chapter 4: Design Patterns as Components
  - Constructional Design Patterns
  - Software Components
  - Design Component Properties
  - Patterns as Components
  - Pattern Interfaces
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

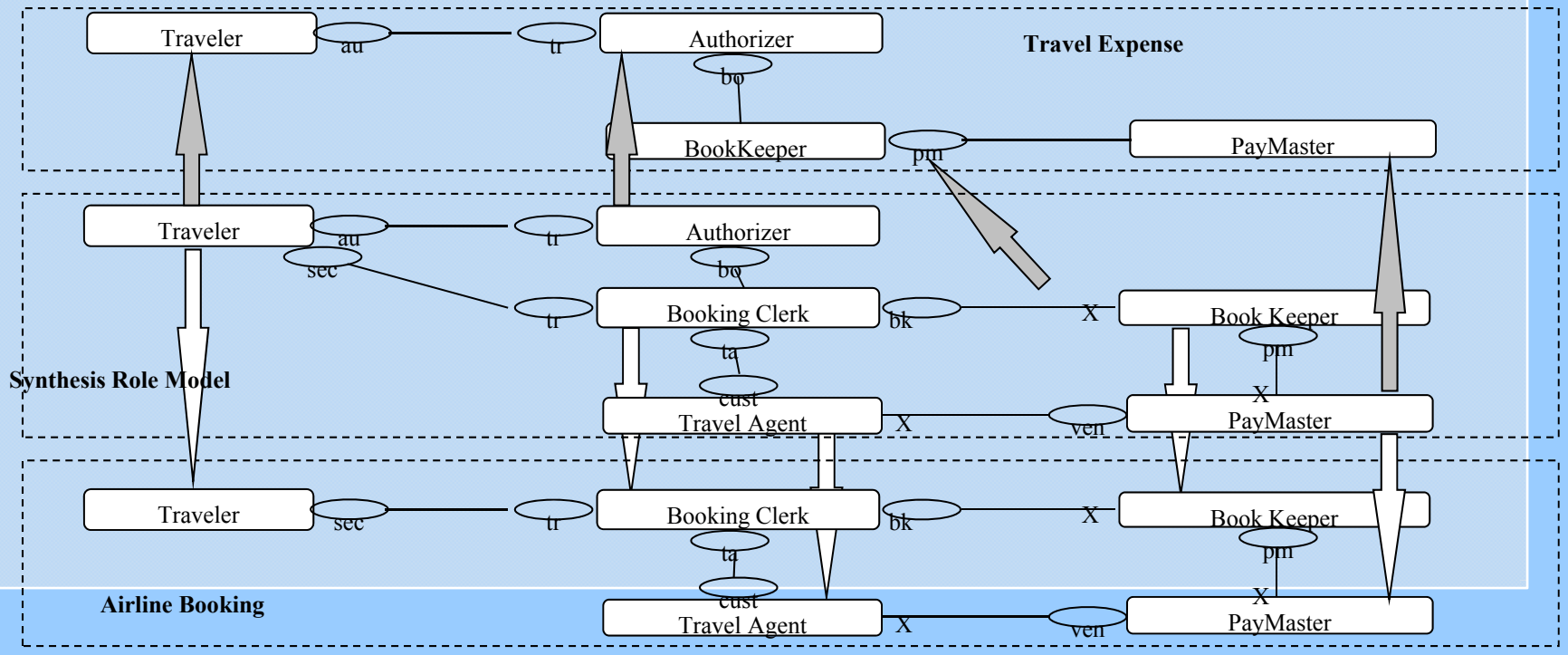
Structural vs Behavioral Pattern Composition

- Pattern Oriented Analysis and Design, POAD. Provides a structural approach for pattern composition (Not a behavioral approach such as role based or Aspect-Oriente
  methods)
  - Example of an aspect-oriented
    is the trace pattern
    applied to a class to trace
    Functions entries and exits



Role-Based Behavioral Pattern composition (Not to be discussed in details)

- A role model is a collaboration of objects that the analyst chooses to regard as a unit, separated from the rest of the application during some period of consideration. A Synthesis Role Model is obtained from different role models



Outline

- Chapter 4: Design Patterns as Components
  - Constructional Design Patterns
  - Software Components
  - Design Component Properties
  - Patterns as Components
  - Pattern Interfaces
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Constructional Design Patterns (structural pattern composition)

- In POAD, the category of Patterns used is called *constructional design patterns.*
- The term Constructional indicates that the patterns:
  - Are design components used in constructing Application Design.
  - Patterns become the core building blocks of the design

What Constructional Design patterns can Offer

- Encapsulation is a core concept of Object Oriented Analysis and Design, OOAD.
- It provides a means to access an interface
- Constructional design patterns encapsulate information
  - They encapsulate solutions to a common design problems
- They are analogous to classes in OOD
  - Apps are built by gluing these patterns together

Outline

- Chapter 4: Design Patterns as Components
  - Constructional Design Patterns
  - Software Components
  - Design Component Properties
  - Patterns as Components
  - Pattern Interfaces
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Software Components

- Component–Based software development is a corner stone in software engineering (development processes and tools support component-based development)
- Software components are:
  - Self contained
  - Fairly independent
  - Require little or no customization (true only at the code level)
  - Provide well-defined services for the whole application

Software Components: **JavaBeans**

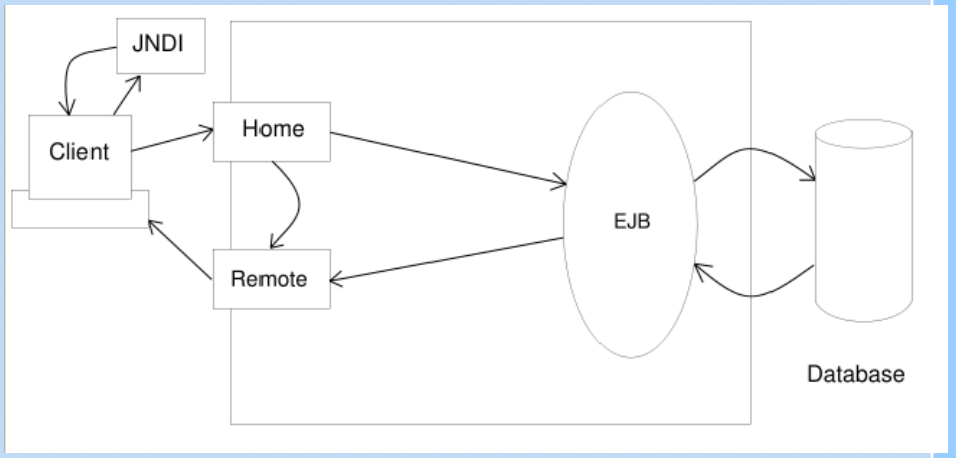
- **JavaBeans** are reusable software components for Java that can be manipulated visually in a builder tool. Practically, they are classes written in the Java programming language conforming to a particular convention.
- They are used to encapsulate many objects into a single object (the bean), so that they can be passed around as a single bean object instead of as multiple individual objects.

Software Components: **JavaBeans**

- AWT, Swing, and SWT, the major Java GUI toolkits, use JavaBeans conventions for their components.
- This allows GUI editors like the Eclipse Visual Editor or the NetBeans GUI Editor to maintain a hierarchy of components and to provide access to their properties via uniformly-named accessors and mutators

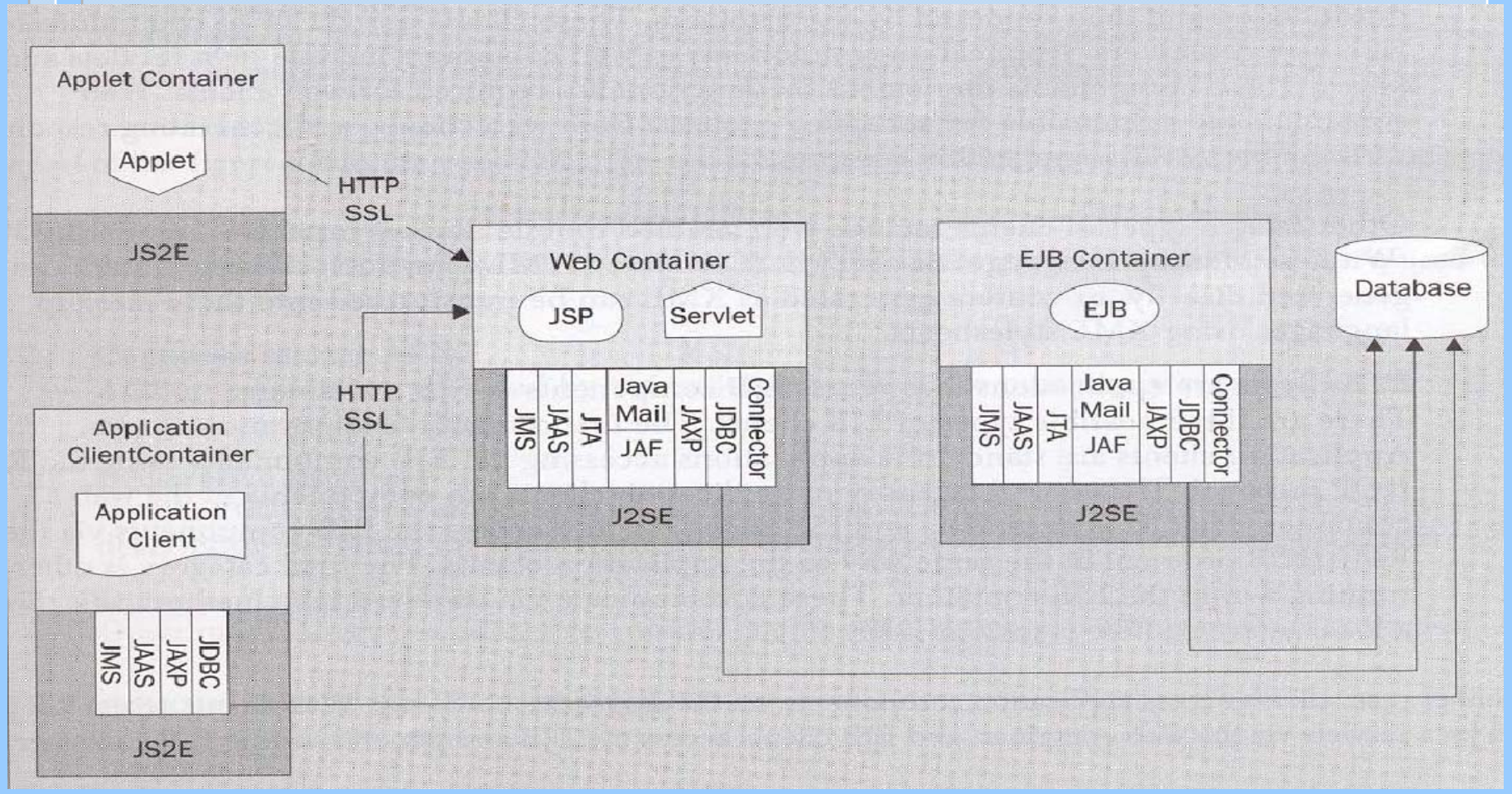
Software Components: **Enterprise JavaBeans**

- **Enterprise JavaBeans (EJB)** is a managed, server-side component architecture for modular construction of enterprise applications.
- EJB encapsulates the business logic of an application, with Concurrency control, Java Naming and directory services (JNDI), Security (Java Cryptography Extension (JCE) and JAAS), and Exposing business methods as Web Services Events using Java Message Service, Remote procedure calls using RMI-IIOP.



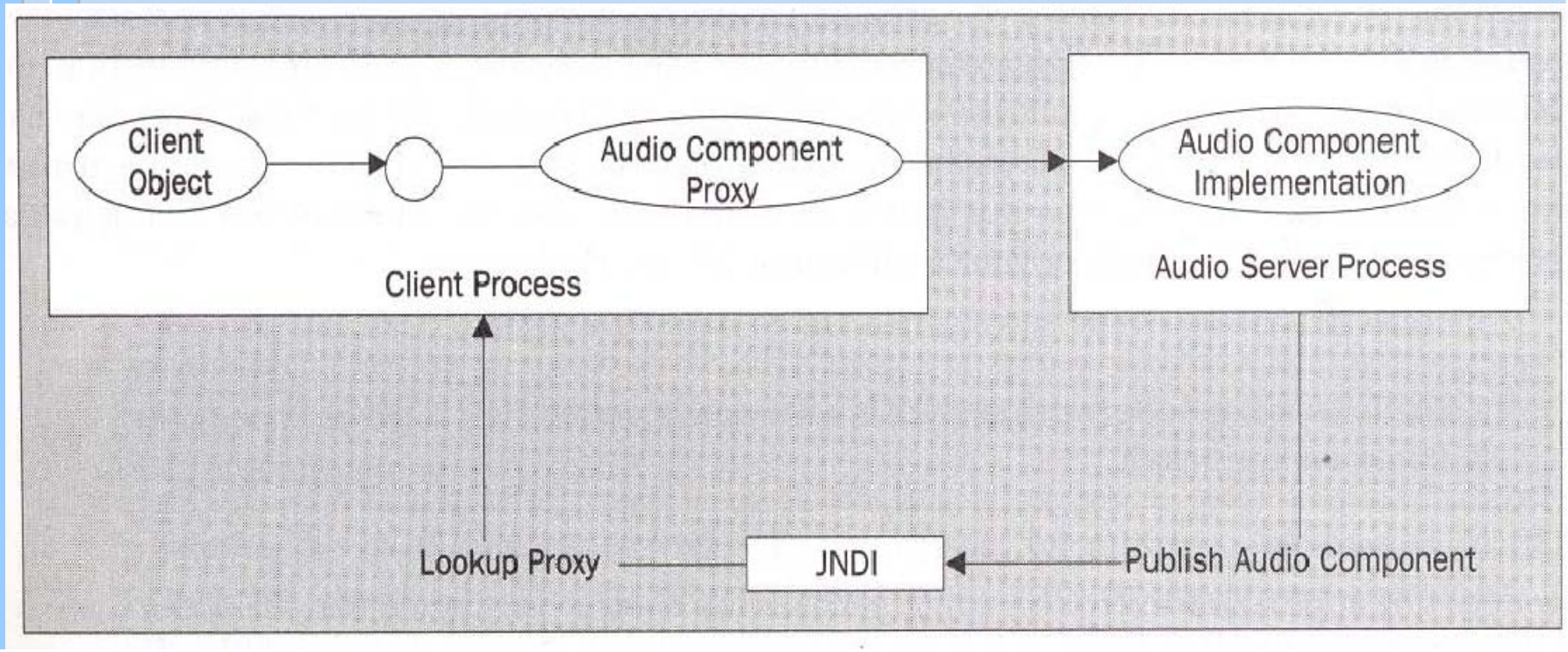
The Bigger Picture: Java 2 Enterprise Edition (J2EE) :

- J2EE Architecture



Software Components: Container Service Application Programming Interfaces (APIs)

- Example: create audio component, publish its name in a naming service (JNDI) available to your application. This provides a simple method to access the service APIs



Software Components: Component Object Model (COM) Technologies

- Microsoft COM technology in the Microsoft Windows-family of Operating Systems enables software components to communicate.
- COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services.
- The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX Controls.
- Microsoft recommends that developers use the .NET Framework rather than COM for new development.

Software Components: .NET

- The Microsoft .NET Framework is a managed code programming model for building applications on Windows clients, servers, and mobile or embedded devices.
- Developers use .NET to build applications of many types: Web applications, server applications, smart client applications, console applications, or database applications
- Windows Communication Foundation is a set of .NET technologies for building and running connected systems. It is a new breed of communications infrastructure built around the Web services architecture.

Outline

- Chapter 4: Design Patterns as Components
  - Constructional Design Patterns
  - Software Components
  - Design Component Properties
  - Patterns as Components
  - Pattern Interfaces
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Design Components

- Characteristics of design components
  - Defines a software design fragment
  - Represented using design notation and delivered as a design model
  - It is deployable at design time
  - White box component (well defined design structure and behavior)
  - Well defined Interface, to glue and integrate with other design components.

Design Component Properties

- Composable
  - The Internals are defined in terms of the internal structure and behavior models.
  - The interfaces by which it is glued together with other design components
- Customizable: Can be customized to allow selection between tradeoffs at lower design levels
- Persistent: Internals are preserved after instantiation and are traceable

Outline

- Chapter 4: Design Patterns as Components
  - Constructional Design Patterns
  - Software Components
  - Design Component Properties
  - Patterns as Components
  - Pattern Interfaces
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Specifying Patterns as Components

- A pattern can be described and specified in a variety of forms
- We classify techniques to describe a pattern into three categories
  1. Recipe – an informal description of the pattern that helps application designers to understand the pattern
     - Essential elements are
       - Context in which problem is incurred
       - The problem solved by the pattern
       - Forces influencing the selection of the pattern
       - Solution to be used for problem at hand
       - Consequences of applying the pattern
     - When it comes to composition with other patterns, the recipe is not sufficient to guide the integration process

Patterns as Components

2. Formal Specification (use notation based on scripting design languages or design scripts)
   - Helps designers compare and contrast several solution issues
   - Improves understandability of patterns
   - Patterns encapsulate mental reasoning decisions beside their technical solutions that are difficult to capture using formal techniques.
3. Interface Specification
   - It is necessary to see how the patterns (thr' interfaces) glue together and to other design artifacts

Component Interfaces

- Examples of Interface Specifications
  - Module Interconnection Languages (MILSs)
  - Interface Definition Language (IDL) (found in CORBA, now an ISO standard, see http://www.omg.org/gettingstarted/omg_idl.htm)
  - Web Services Description Language (WSDL)

(found in SOA, see http://www.w3.org/TR/2007/REC-wsdl20-20070626/#component_model)

  - Interfaces for OO Components

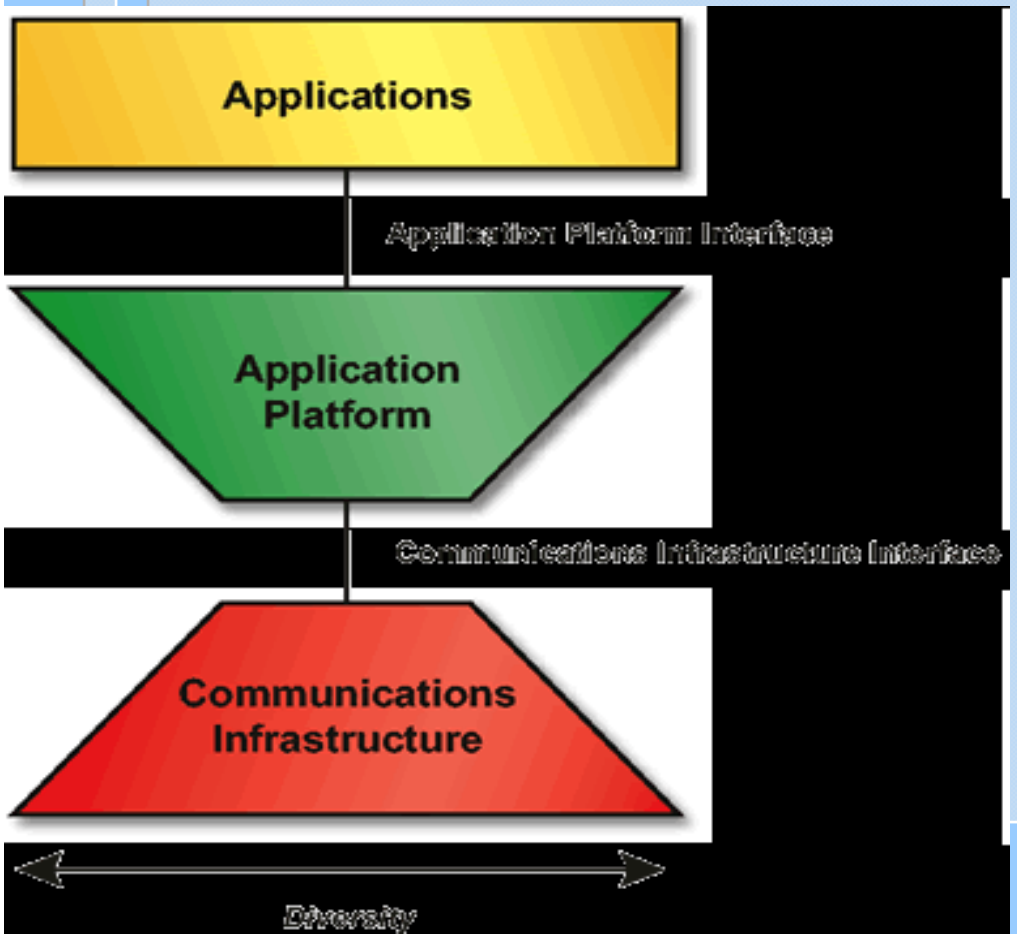
(see http://portal.acm.org/citation.cfm?id=566171.566212)

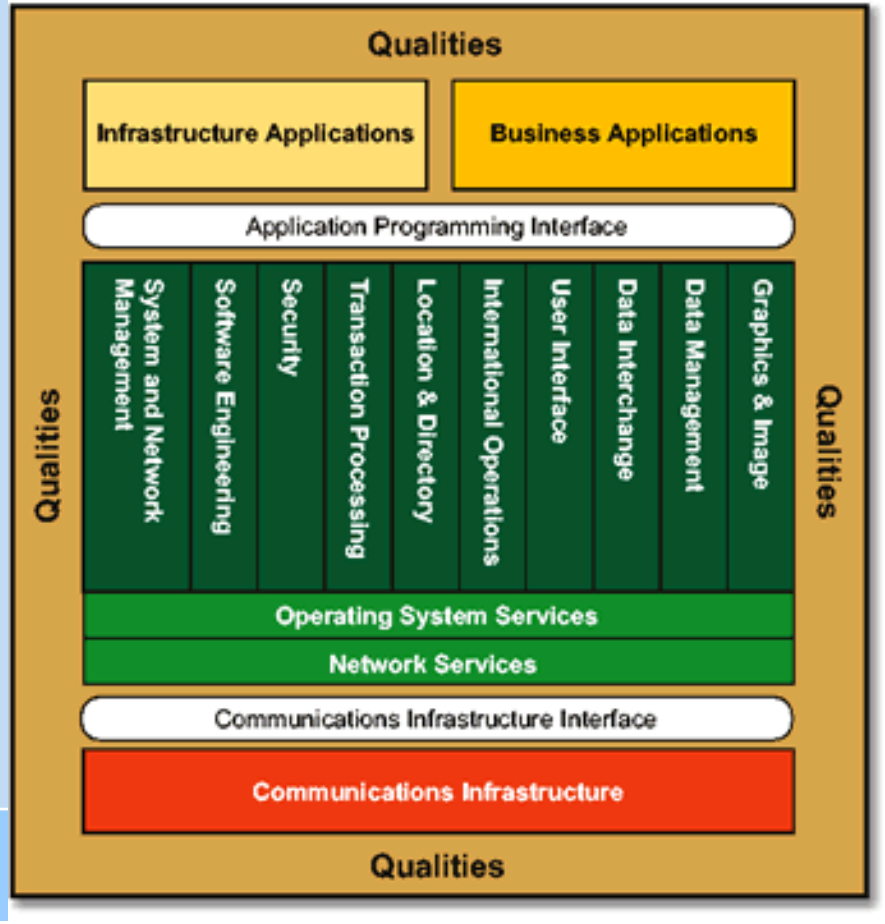
One approach uses the idea of contracts to define interfaces of objects

Component Interfaces: APIs

http://www.opengroup.org/architecture/togaf8-doc/arch/chap19.html

- Application/Platform Interfaces (APIs) support portability





Component Interfaces

- Interface Properties
  - Type
    - Referential – (Class reference or Pattern reference)
      - A a client or a requestor component has a reference to the provider component with no details about the usage relationship
      - Useful for building the design structural views
    - Functional – (Services and Actions)
      - Specify the services provided by the design component and the services required from other components.
      - Useful for building the design behavioral views

Component Interfaces

- Interface Properties (cont.)
  - Role: distinguishes the role that is played by a design component
    - Emphasizes the C/S relationship and explicitly defines provided and required interfaces.
  - Nature
    - Abstract (most cases, e.g. abs classes, or Java Interfaces)

      vs. concrete (provides default implementation for the interface operations.)
  - Dynamism
    - Static (e.g. CORBA IDLs or Java interfaces)

      vs. Dynamic Interfaces (multiple objects that wait for multiple events))
      - Dynamic interfaces are not specified for users at design-time instead they are interrogated (inquired) by the calling component at run-time (for example, CORBA Dynamic Method Invocation).

Component Interfaces

- Description
  - Description characterizes the interface
  - Signature
    - Names and parameters
  - Behavioral
    - How the component reacts when it is called
- Multiplicity
  - A component can have multiple interfaces, all of which are valid interfaces to the same component
  - According to the application context using the component one of those interfaces will be used

Outline

- Chapter 4: Design Patterns as Components
  - Constructional Design Patterns
  - Software Components
  - Design Component Properties
  - Patterns as Components
  - Pattern Interfaces
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

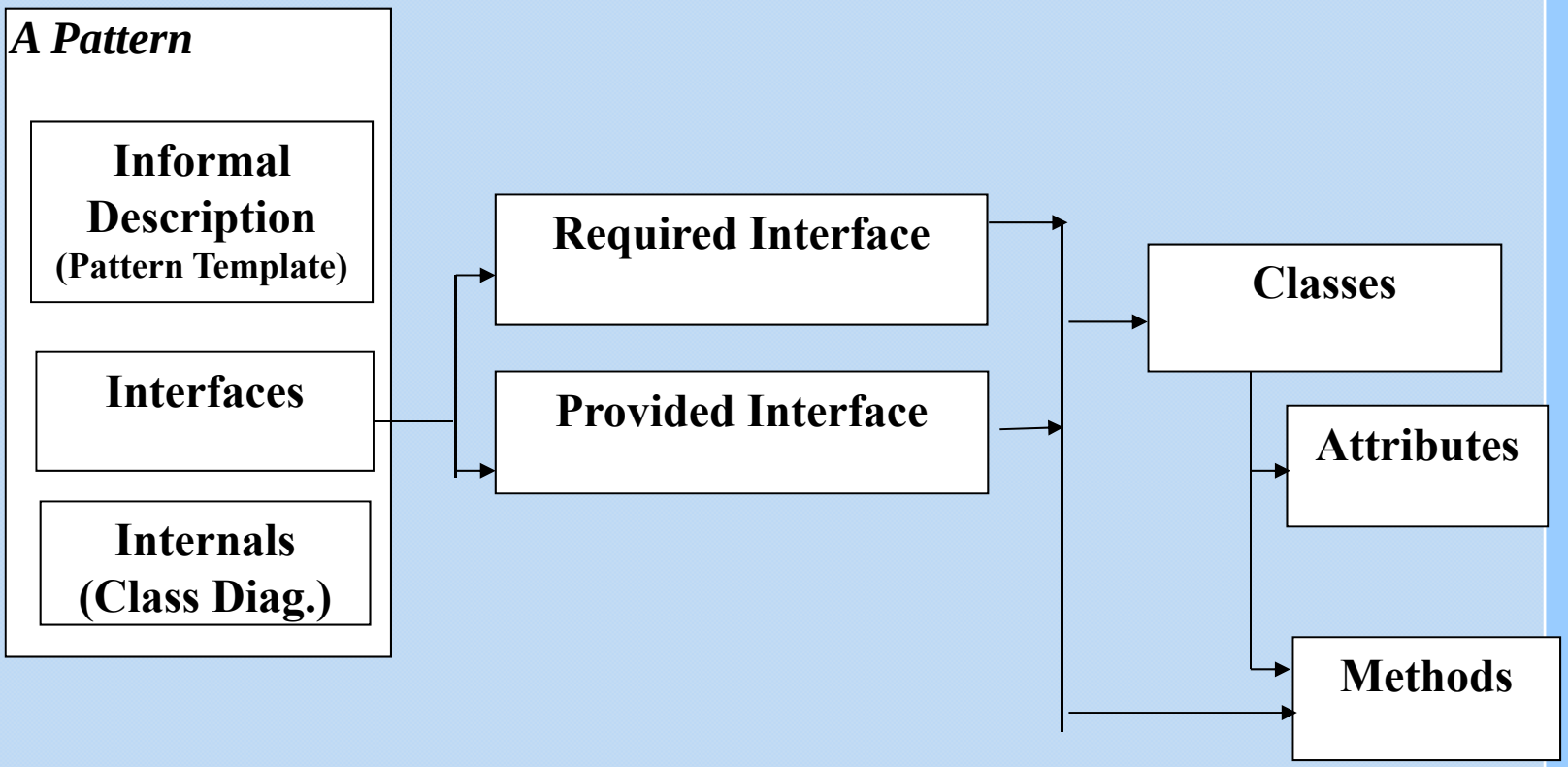
Pattern Interfaces

- Pattern interfaces must conform with the OO paradigm
- Interfaces are important for 3 reasons
  - Hide details
  - Distinguish parts crucial for integration
  - Provide flexibility
- Pattern Interfaces are application interfaces

Constructional Design Patterns (CDPs)

- Definition: A CDP has additional constraints that allow for composition and integration.
- CDPs are OO design patterns
- Have interfaces for composition and integration
- Their structural solution is based on well-defined class diagrams

A pattern template with emphasis on interfaces



Examples of CDPs

- All object behavioral patterns

  Iterator (257)
  Mediator (273)
  Memento (283)
  Flyweight (195)
  Observer (293)
  State (305)
  Strategy (315)
  Visitor (331)

  Chain of Responsibility (223)
  Command (233)

Outline

- Chapter 4: Design Patterns as Components
  - Constructional Design Patterns
  - Software Components
  - Design Component Properties
  - Patterns as Components
  - Pattern Interfaces
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Chapter 5: Visual Design Models

Outline

- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Pattern Composition Models

- Pattern Visualization
  - Models that are used to capture the internal design
  - UML based
- Pattern Composition Visualization
  - Models used to capture integration and composition of a set of patterns
  - Some use UML others use other tools
- POAD is based on structural composition

Pattern Composition Models

- POAD requires models to have varying granularity
  - Course grained
    - Used for integrating patterns
  - Fine grain => Class diagrams
    - Capture the internals of a Pattern
- POAD uses Hierarchical OOD
  - Objects may contain other Objects in a Hierarchical fashion
  - Connectivity between interface objects and internal objects

Guidelines in defining POAD Models

1. Model elements must serve a purpose
   - Close to mental building blocks
2. Models tend to be Hierarchical
   - Capture the design at various levels of abstraction
3. Exchangeable
   - Should integrate with and use UML.
4. Models serve other models
   - Models used in one phase should be used in others

Outline

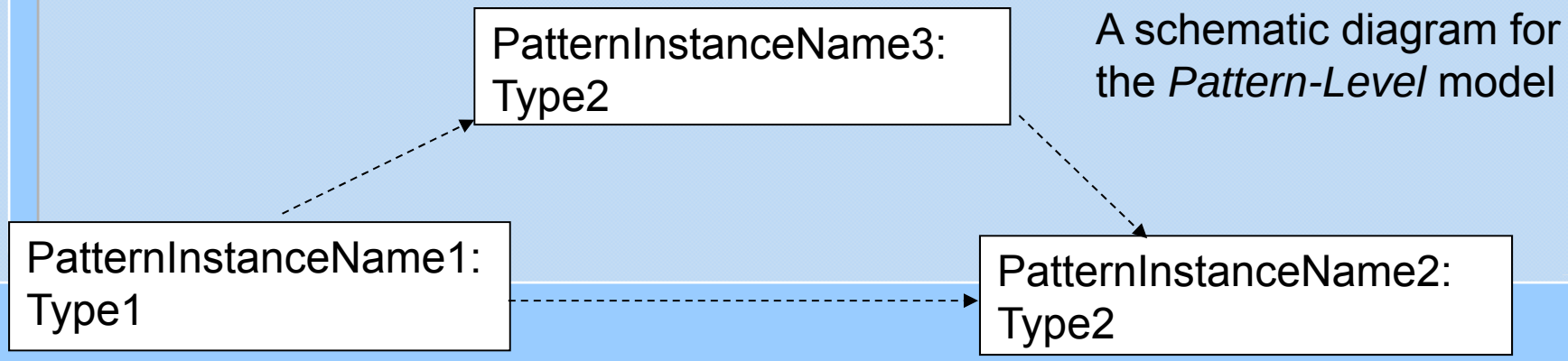
- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

POAD's design models

1. Pattern Level – pattern interfaces and dependencies
2. Pattern Level with interfaces – Explicitly defines relationships between interfaces
3. Detailed Pattern level – connectivity between internals and interfaces is defined.

Pattern Level

- Schematic- represents the patterns and the relationships between them
  - Pattern instance
    - Type – Observer, Factory, Strategy etc.
    - Name –Application specific as given by designer
      - Used to differentiate when there exist 2 instances of same type.

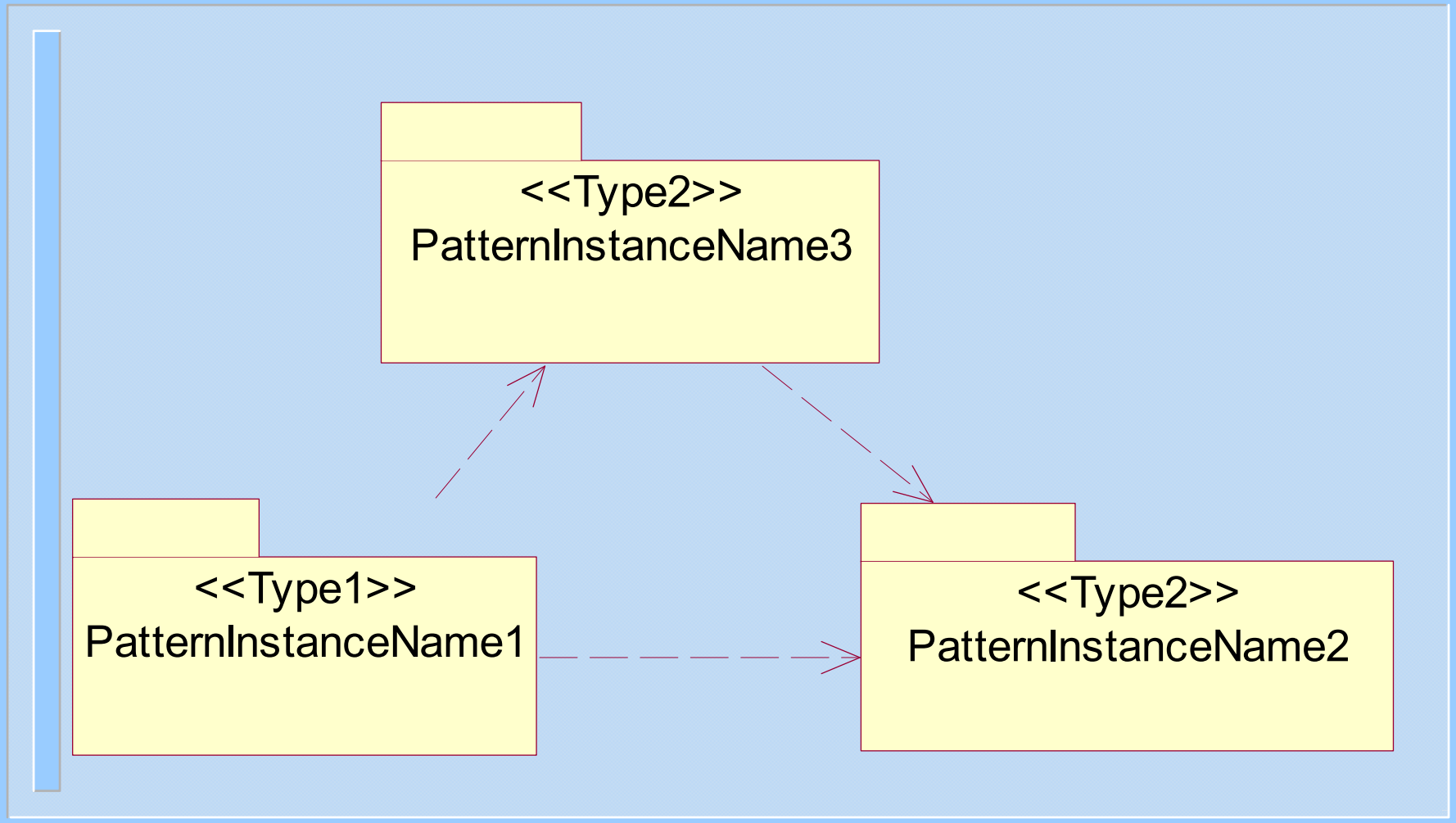


A schematic diagram for the *Pattern-Level* model

Pattern Level

- Relationships –only one relationship exists
  - Dependency –USES, further defined later and become associations between interface classes
- Design Decisions
  - Selecting the appropriate Patterns
  - Defining Dependencies –How one pattern uses another
- UML Syntax
  - Pattern level view resembles UML Package diagrams
  - Packages represent constructional Patterns
  - Name of Package is Instance Name
    - Relationships are defined as *dependency*

Pattern-Level model using UML Syntax



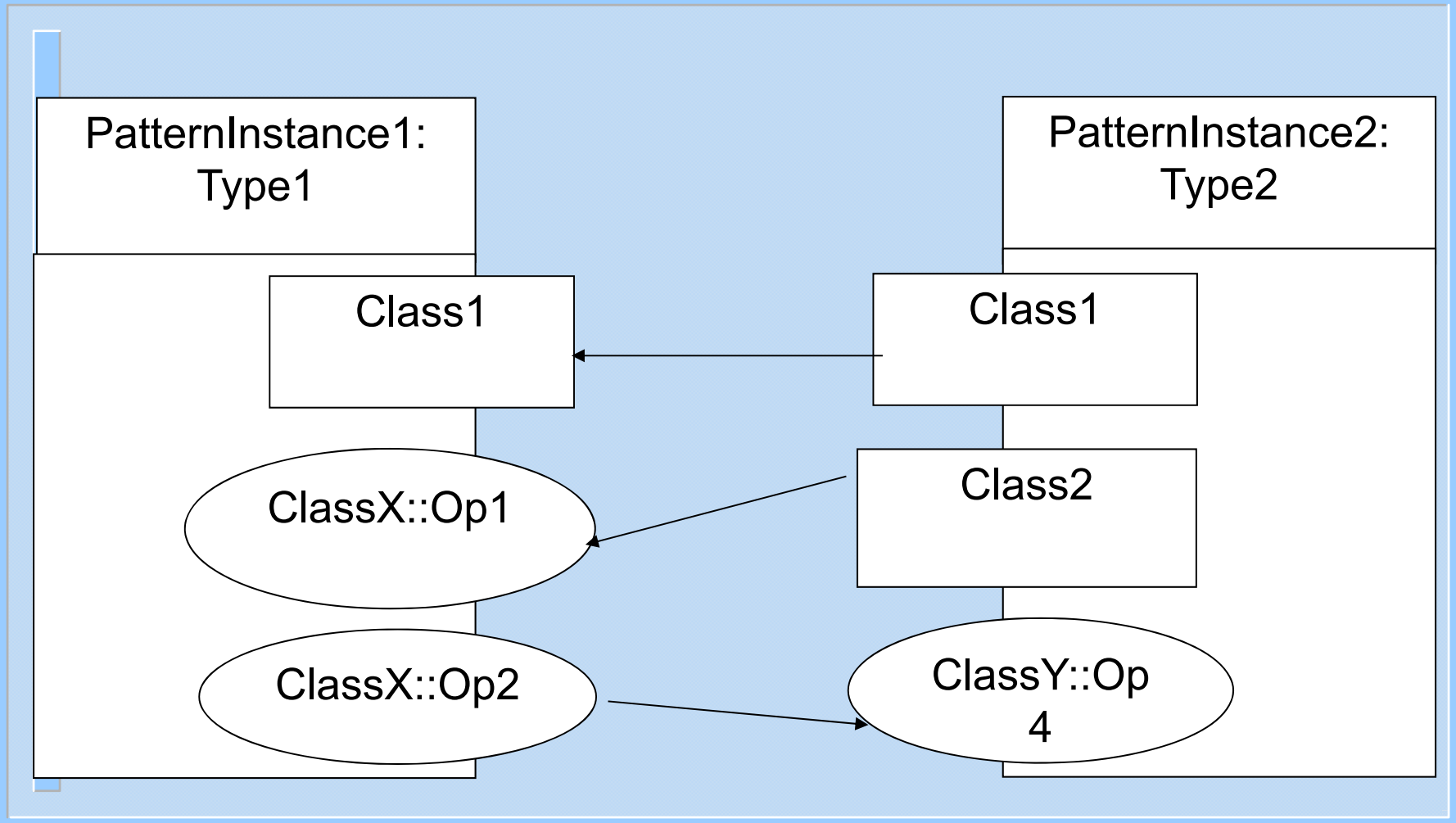
Outline

- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Pattern Level With Interfaces

- Schematic –shows interfaces and relationships between them, 2 types of interfaces
  - Interface Classes –One of the internal classes
  - Interface Operations –An Operation in one of the interface classes. One internal class can implement several operations.

A schematic diagram for the *Pattern-Level with Interfaces* model



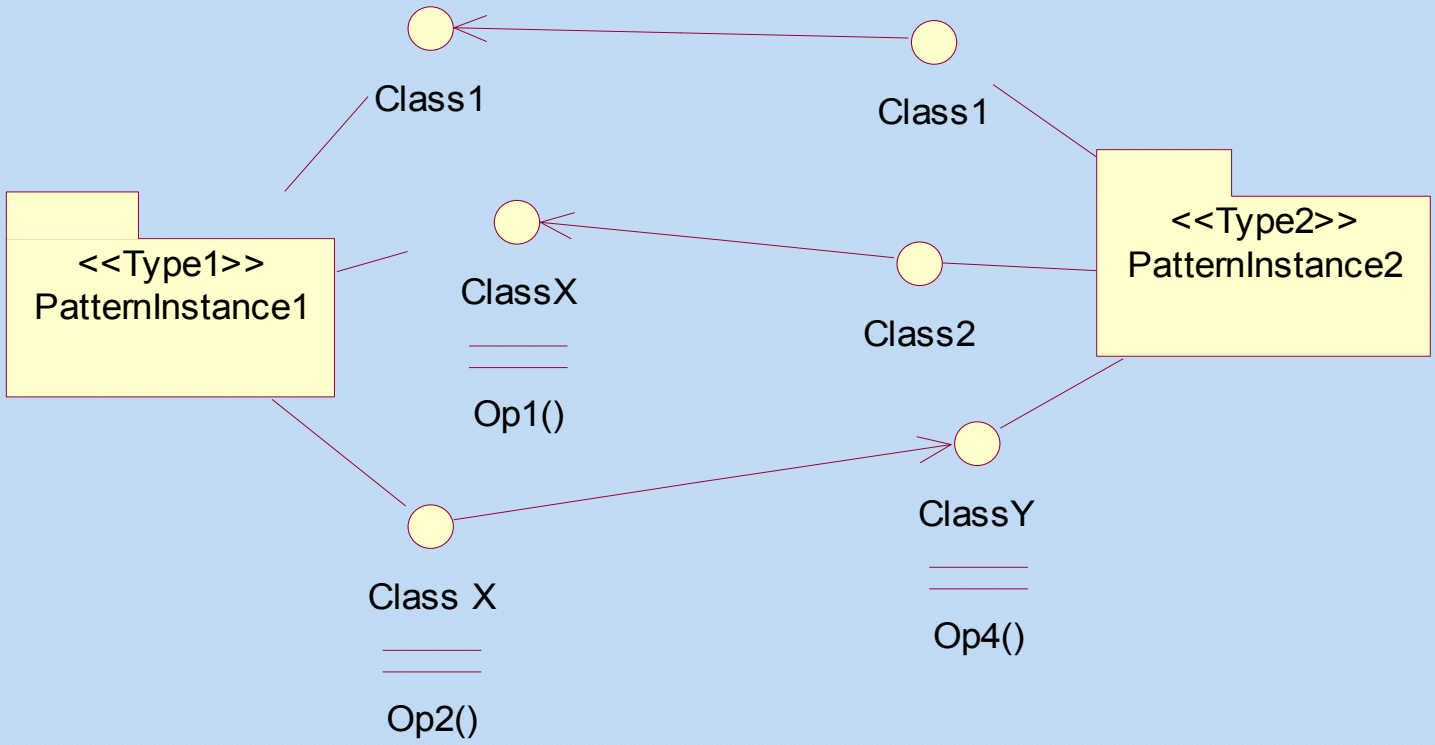
Pattern Level With Interfaces

- Relationships – *uses* between pattern interfaces further defined
  - Class/Class –Aggregation, association or dependency
  - Class/Operation –Interface class can Invoke an operation in another pattern.
  - Operation/Operation –Models interactions, show the designer's perceptions of lower level design details

Pattern Level With Interfaces

- Design Decisions –Selecting which interface to use for a given application.
- UML syntax –UML package and interface notation is used.
  - For interface classes and operations the UML syntax for interface is used (circle associated with a package).
  - A circle with only a class name underneath is a class interface
  - An operation interface is denoted by listing the operation name underneath the class name

The *Pattern-Level with Interfaces* model using UML



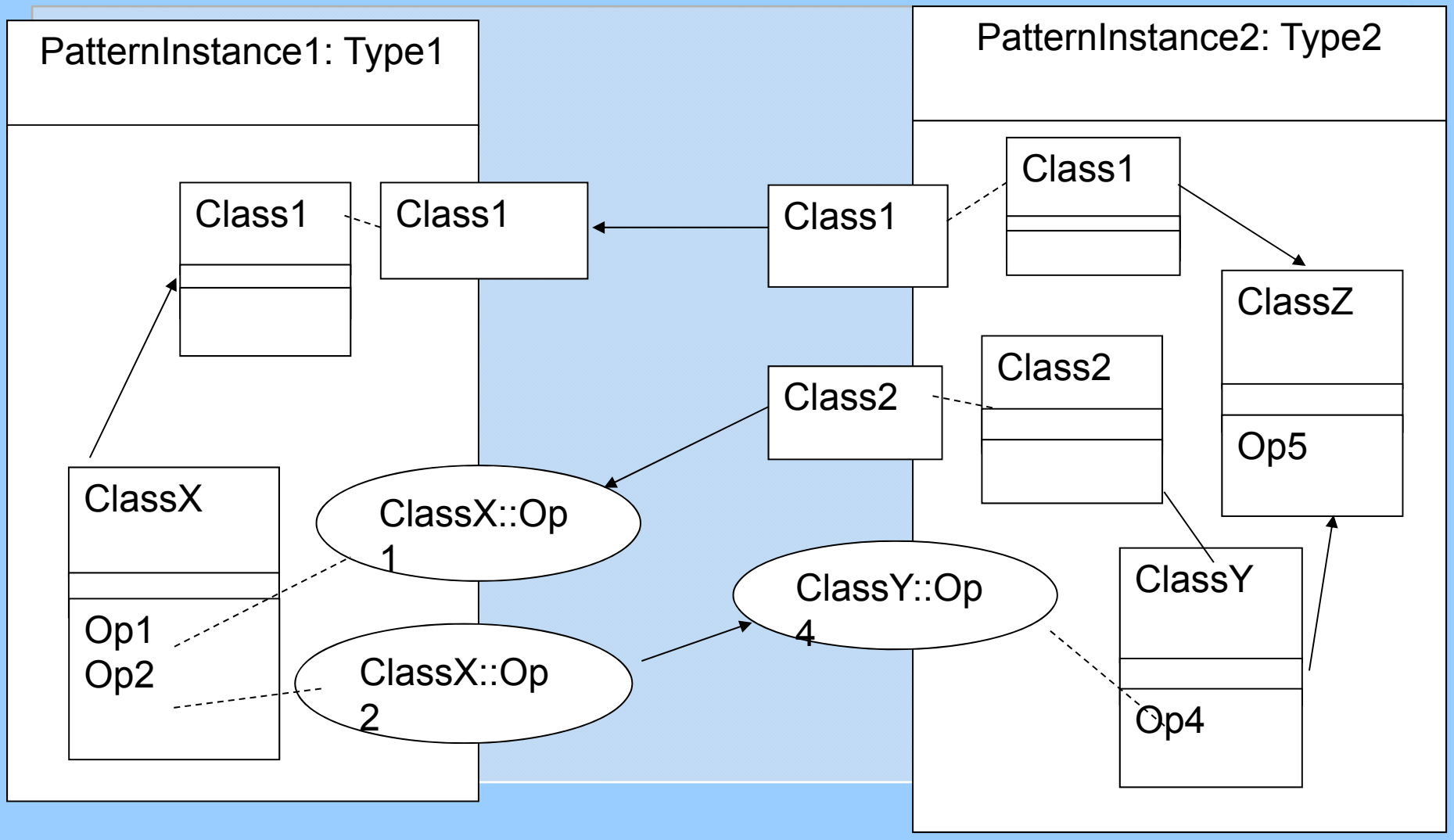
Outline

- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Detailed Pattern Level Model

- Purpose –To explore the internal details of each pattern and ID the internal classes that implement the interface.
- Schematic –Represents Patterns and their internal structure.
  - Pattern Instances and type
  - Interfaces
  - Internal class diagram
  - Relationship between interfaces and internals is established

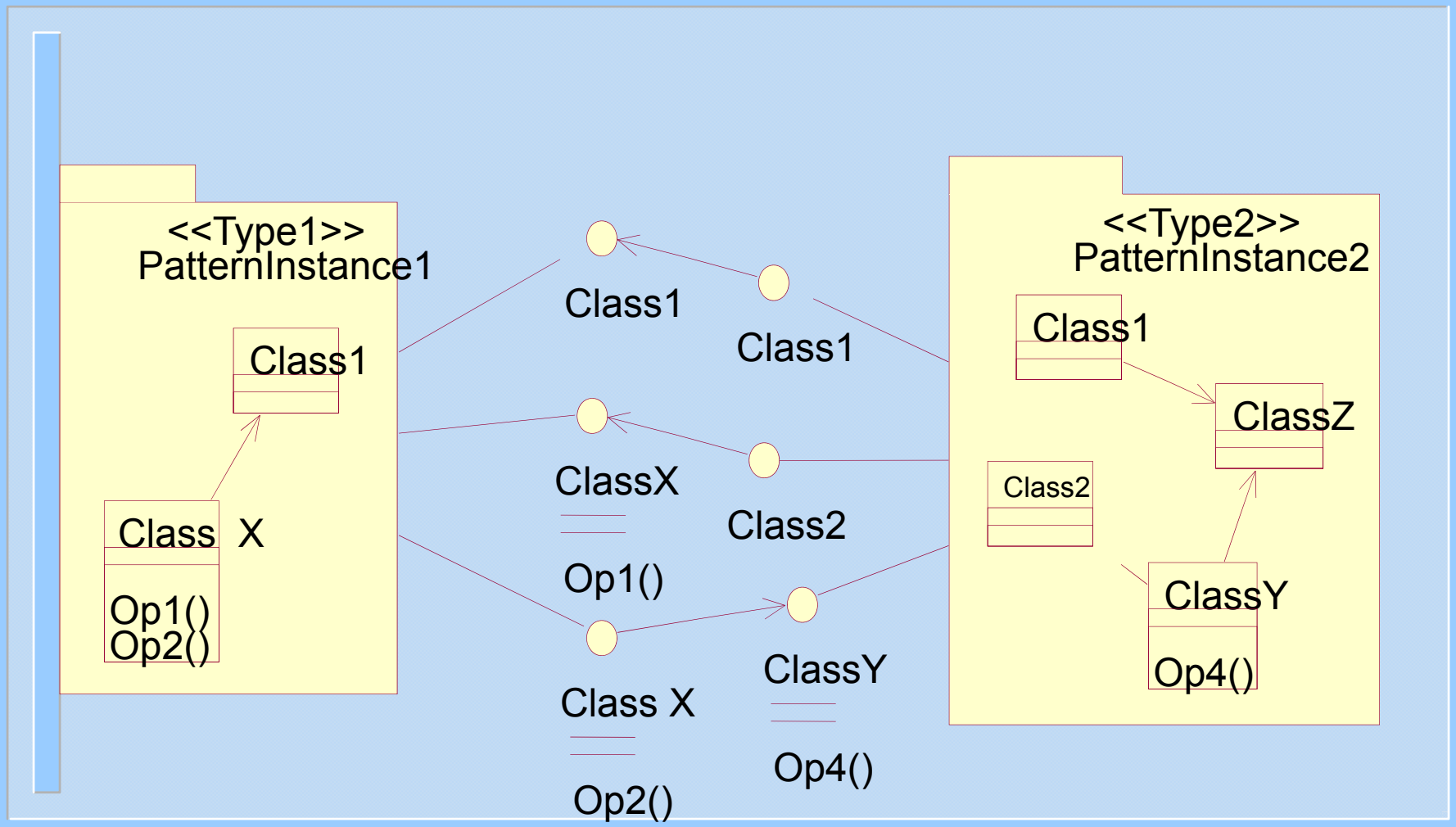
Schematic for The Detailed Pattern-Level view



Detailed Pattern Level Model

- Relationships –Connectivity is used to show which elements of internal are visible as interfaces.
- Design Decisions –None, This is a refinement stage
- UML syntax –The internal class diagram of each pattern is shown.
  - Interfaces from previous model are incorporated into this class diagram for each pattern

Detailed Pattern Level Model



Outline

- Chapter 5: Visual Design Models
  - Pattern Composition Models in general
  - POAD design models
    - Pattern Level
    - Pattern Level with Interfaces
    - Detailed Pattern Level with interfaces
  - Characteristics of POAD

Characteristics of POAD

- Hierarchy –The three models demonstrate 3 levels of abstraction. This allows the internals to be suppressed at one abstraction and then expressed at another.
- Traceability –Must be able to trace from high abstraction to lower.
  - Pattern dependencies in pattern-level are traceable to the relationships in pattern-level with interfaces view.

Characteristics of POAD

- Traceability enables designer to navigate up or down levels of abstraction
- Composability –enables model elements to be plugged together. Artifacts in each view are described as pluggable