UML Implementation Diagrams

Object-Oriented Analysis and Design

CSCI 6448 - Fall 1998

Kenneth M. Anderson

Goals of the Lecture

- Present UML Diagrams useful for implementation
- Provide examples
- Next Lecture
  - A variety of topics on mapping from design into implementation
    - translating associations into operations, scalability considerations, etc.

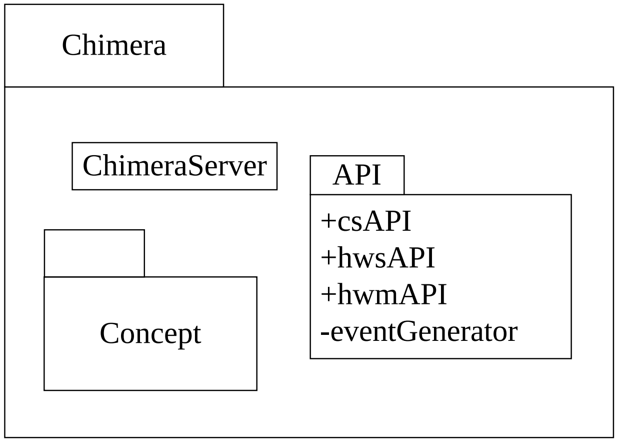
Overview

- Package Diagrams
  - Systems and Subsystems
- Component Diagrams
- Deployment Diagrams
- Collaborations

Packages

- A package is a general-purpose mechanism for organizing elements into groups
  - Packages are rendered as tabbed folders
  - The visibility of elements can be specified
  - Elements are semantically related for a single purpose
- Packages are thus conducive to providing groups which are highly cohesive, loosely coupled, and access tightly controlled

Package Example



Chimera

ChimeraServer

Concept

API

+csAPI
+hwsAPI
+hwmAPI
-eventGenerator

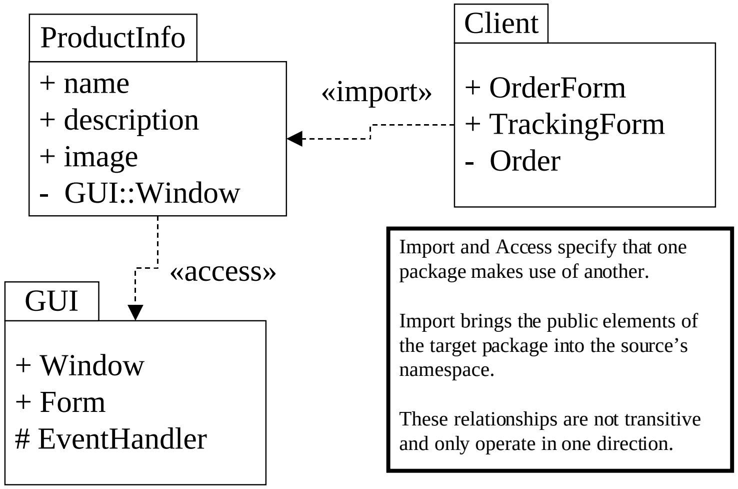
Information on Packages

- Packages can contain
  - packages, classes, interfaces, components, nodes, collaborations, use cases, etc.
- Containment is a composite relationship
  - The element belongs to the package
    - Delete the package, you delete its elements

More Information on Packages

- Each package defines a namespace
  - Names of elements (of a particular type) must be unique within a package
    - e.g. you can have a class named Timer and an interface named Timer but you can’t have two classes both named Timer
      - its recommended that you keep all names unique regardless of type, however
  - Nested names use the following notation
    - Chimera::API::csAPI

Accessing Package Elements



ProductInfo

+ name
+ description
+ image
- GUI::Window

«import»

Client

+ OrderForm
+ TrackingForm
- Order

«access»

GUI

+ Window
+ Form
# EventHandler

Import and Access specify that one package makes use of another.

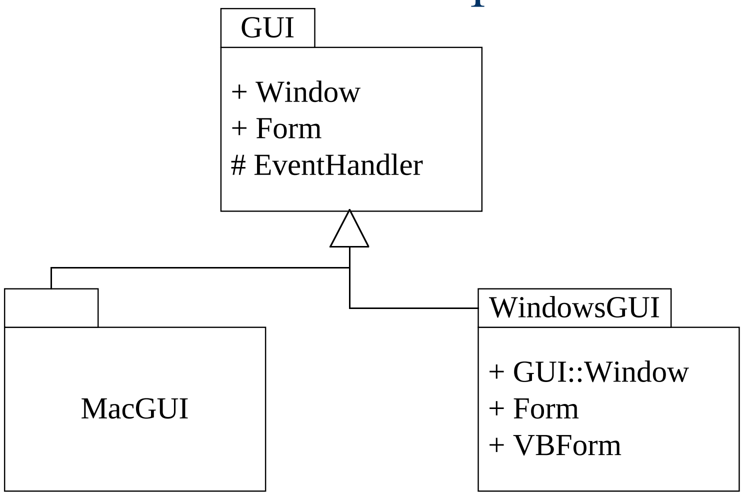
Import brings the public elements of the target package into the source’s namespace.

These relationships are not transitive and only operate in one direction.

Generalization

- Generalization relationships are used to specify families of packages
  - All rules of inheritance apply
    - Public and Protected members are available in children, and children can override members of their parents and add new elements
    - Substitutability can also be used; a child package can be used in the place of one of its parents

Generalization Example



Standard Package Stereotypes

- facade
  - A package that is a view onto some other package
- framework
  - A package consisting mainly of patterns
- stub
  - A package that serves as a proxy for the public contents of another package
- system, subsystem

Systems and Subsystems

- A system is the thing being modeled and developed to accomplish some task
  - It contains all its models such as classes, use cases, activity diagrams, etc.
- A subsystem is simply a part of a system
  - used to decompose a complex system into nearly independent parts
  - typically has high cohesion collecting everything needed to accomplish a particular subtask or goal

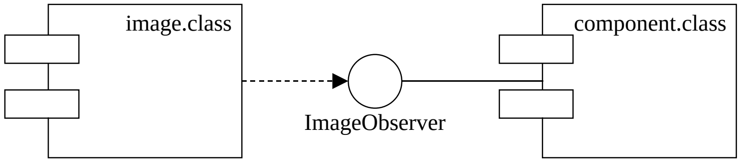
Components

- A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces
  - Rendered as a rectangle with tabs
- Example Components
  - dynamically-linked libraries, jar files, database tables, software components (JavaBeans, CORBA, and DCOM)

Components vs. Classes

- Classes are logical abstractions
  - Components are physical objects that can be deployed
- Classes may have attributes and operations
  - Components typically only have operations defined by their associated interfaces
- Components are at a different level of abstraction; they represent the physical packaging of classes

Components and Interfaces



Component.class exports the interface, image.class imports it.

A component can import and export many different interfaces.

The interface breaks the dependency between the two components; they can change independently as long as they maintain their obligations with respect to the interface.

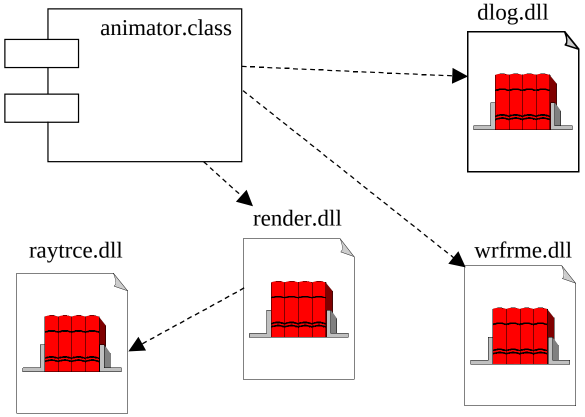
More on components

- Components are physical
- Components are replaceable
- Components are part of a system
  - They can be thought of as building blocks
- Components conform to and provide the realization of a set of interfaces

Standard Component Stereotypes

- executable
  - can be executed on a node
- library
  - static or dynamic object library
- table
  - database table
- file
  - source code or data
- document
  - catch-all for other types of documents

Executable Component Example



animator.class

dlog.dll

render.dll

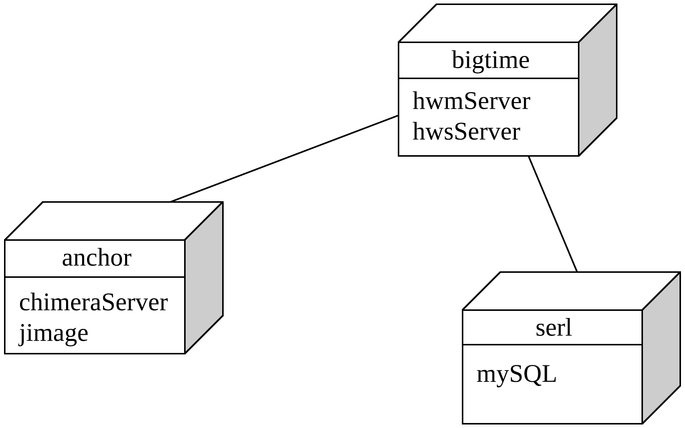
raytrce.dll

wrfrme.dll

Deployment Diagrams

- Document the nodes of a system
- A node is a physical element that represents a computational resource
  - typically having memory and processing capability
  - Nodes model the topology of the hardware used by a system
    - A node represents (typically) a processor or device (sensor, modem, etc.)

Chimera Deployment



bigtime

hwmServer
hwsServer

anchor

chimeraServer
jimage

serl

mySQL

Nodes vs. Components

- Components participate in the execution of a system
  - Nodes are things that execute components
- Components physically package logical elements
  - Nodes represent the physical deployment of components

More information on Nodes

- Nodes (as well as components) can participate in dependency, generalization, and association relationships
- They can be nested, have instances, participate in interactions, etc.
- Nodes can also have attributes and operations

Common uses for Deployment Diagrams

- To model embedded systems
  - Nodes can represent physical devices and show how components access those devices
- To model client/server systems
  - See previous example
- To model fully distributed systems
  - Including adaptive systems
    - for example, agents that migrate from node to node

Collaborations

- A collaboration is a society of classes, interfaces, and other elements that work together to provide a cooperative behavior
  - A collaboration consists of a structural part and a behavioral part
    - Class diagrams specify the former
    - Interaction diagrams specify the latter
  - Rendered as an ellipse with dashed lines

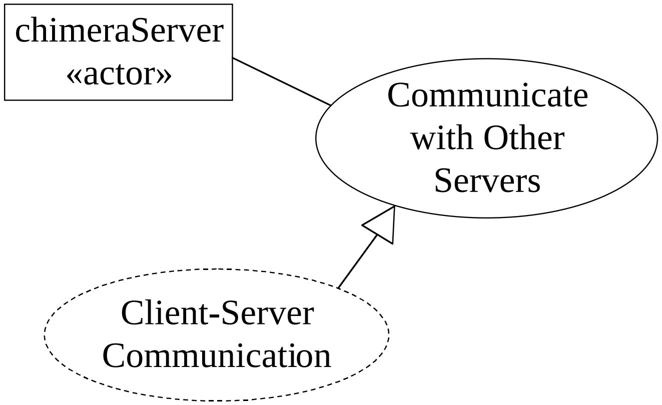
Uses for collaborations

- Modeling a mechanism
  - Show me all the classes and activities involved with client-server communication
- Modeling a use case
  - Show me all the classes and interactions to support the process order use case
- Modeling an operation
  - Show me how ray tracing is accomplished

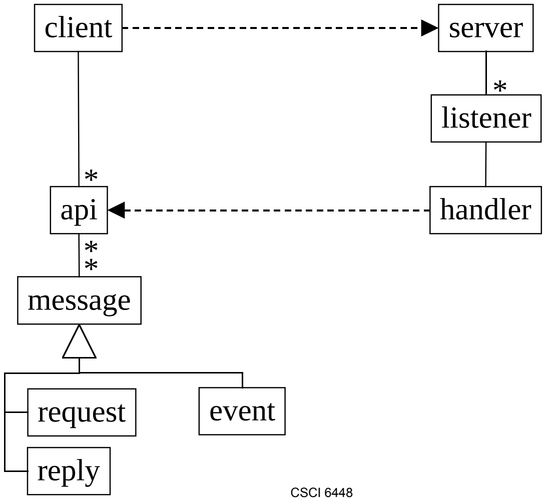
Collaborations, continued

- Elements can appear in multiple collaborations
  - A collaboration can be thought of as a set of pointers into a system's packages
  - Sort of like the perspective mechanism's notion of a virtual copy

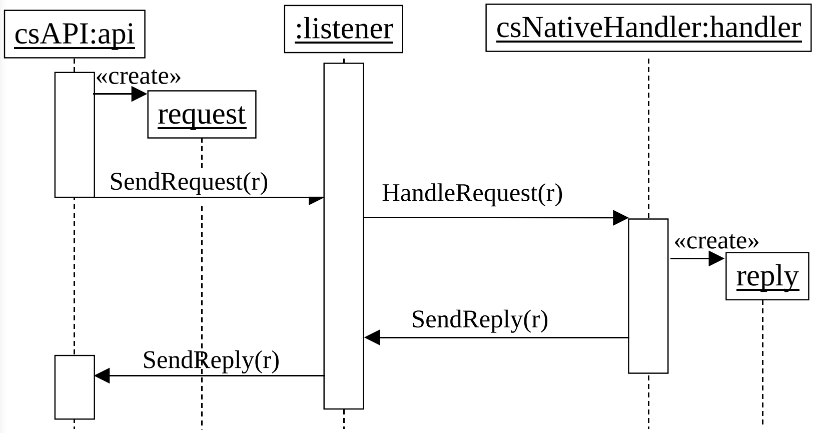
Example



Structural Aspect



Behavioral Aspect


csAPI:api
:listener
csNativeHandler:handler
«create»
request
SendRequest(r)
HandleRequest(r)
«create»
reply
SendReply(r)
SendReply(r)