UNIVERSITY OF Cincinnati

Distribution Patterns
(Patterns of Enterprise Application Architecture - Chapter 15)

Mike Helmick
University of Cincinnati
Large Scale Software Engineering
Spring 2014

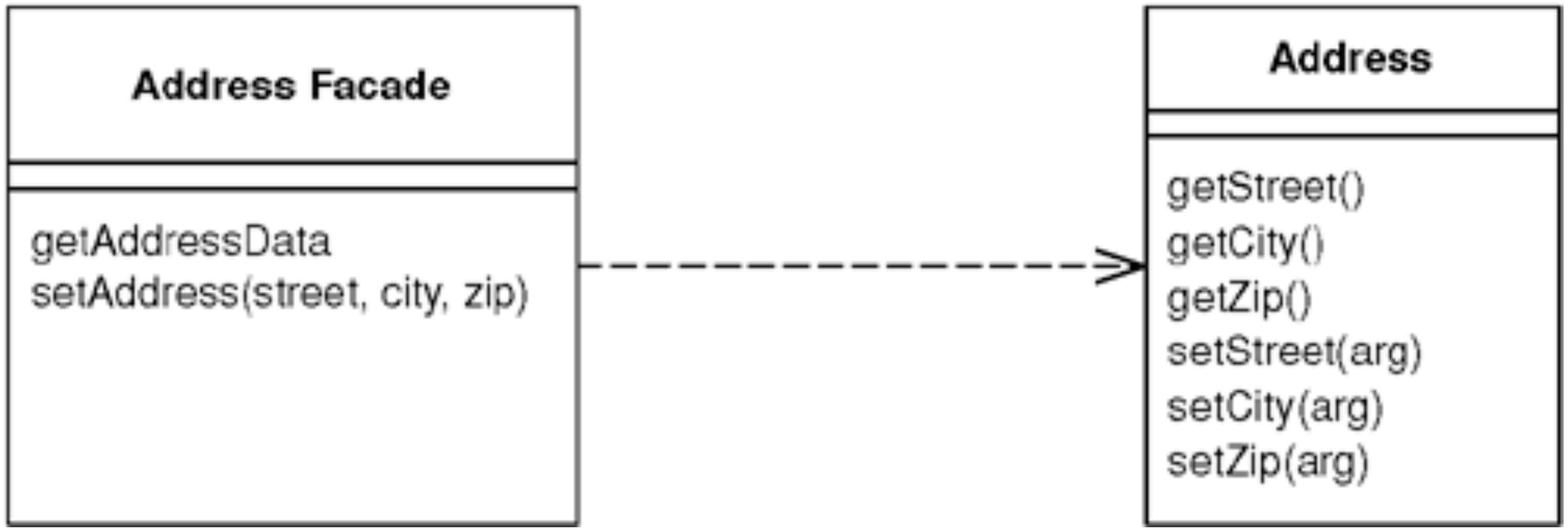
Chapter 15

Chapter 15

- Remote Facade
- Data Transfer Object

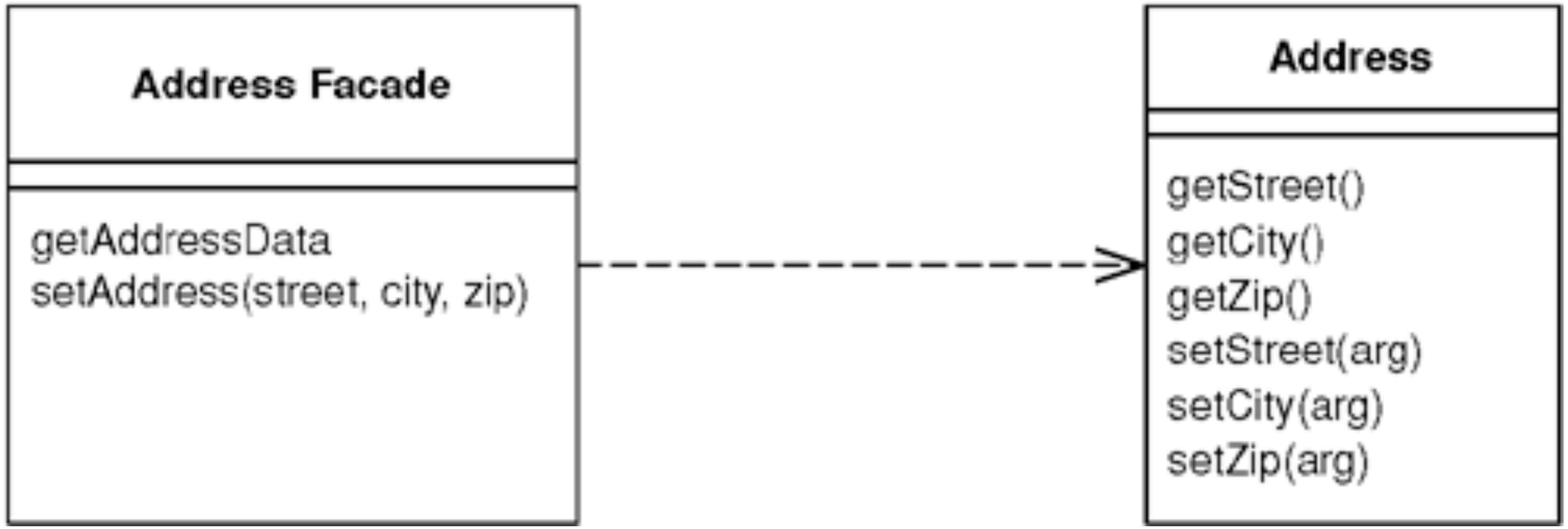
Remote Facade

Remote Facade



Remote Facade

- “Provides a coarse-grained facade on fine-grained objects to improve efficiency over a network.”



Remote Facade

Remote Facade

- OO Model
  - small objects
  - lots of getters / setters
  - lots of interaction between objects

Remote Facade

Remote Facade

- This works well if everything is on the same machine / same JVM
- Remote calls are more expensive
  - convert in memory to wire / marshal
  - make remote call / establish connection
  - convert wire to in memory / unmarshal

Remote Facade

Remote Facade

- So
  - remote interfaces are bundled into coarse grained objects

How It Works

How It Works

- Converts OO interaction into a more procedural paradigm
    - Allowing for easier distribution
- The facade object
    - coarse-grained interface
    - actually works on fine-grained objects

How It Works

How It Works

- Reading data
  - replaces multiple calls
    - get street, get city, get state
  - With one call
    - get address
      - that does the other calls remotely

How It Works

How It Works

- The problem then is
  - how to we represent that data on both sides of the network?

Serializable

Serializable

- If the class definition is available on both sides
    - then the objects can be serialized
    - but they have to be serializable

Duplication

Duplication

- This requires you to duplicate your domain objects in both the clients and the servers
- May not be a possibility if you don’t want to give clients your objects
  - Can leverage the use of Data Transfer Object
  - or something like XML, JSON, protocol buffers, thrift, or whatever the new hotness is

Granularity

Granularity

- Need to balance the amount of activity done in each call
    - against the number of calls available

Design

Design

- These interfaces basically get design based on the needs of the service consumers
- Might lead to several APIs for working remotely with the system
  - Duplication is somewhat acceptable in this case
  - You want to make life easier for clients

State

State

- Can be stateful or stateless
- Stateless
  - can be more efficient
  - poolable
  - reusable

State

State

- Most likely --
  - we still need to maintain state
    - Client Session State
    - Database Session State
    - Server Session State

What goes in it

What goes in it

- Application Logic
  - not Domain Logic
    - this is what you do not want to duplicate

Service Layer

Service Layer

- Remote Facades are similar to Service Layers
- Service Layer
  - not always remote
  - not necessarily coarse-grained
  - usually doesn’t use data transfer object

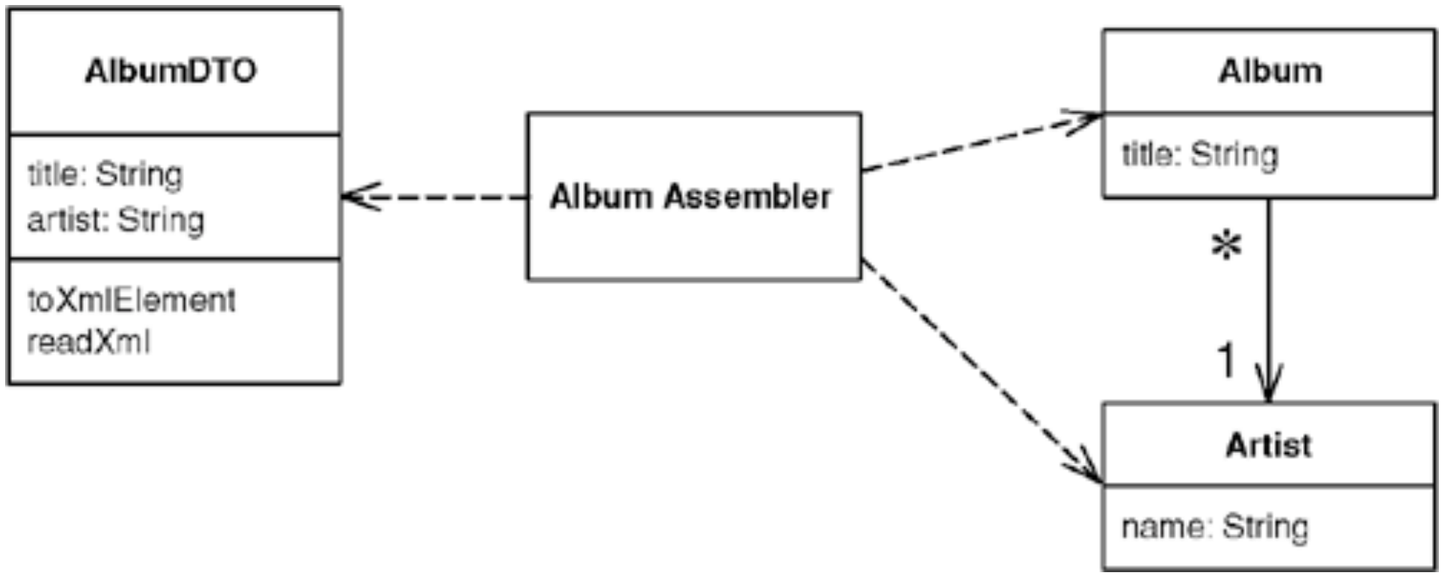
When to Use It

When to Use It

- When you need remote access
    - and the domain layer is composed of fine-grained objects
- When 2 processes need to communicate (even on the same machine)

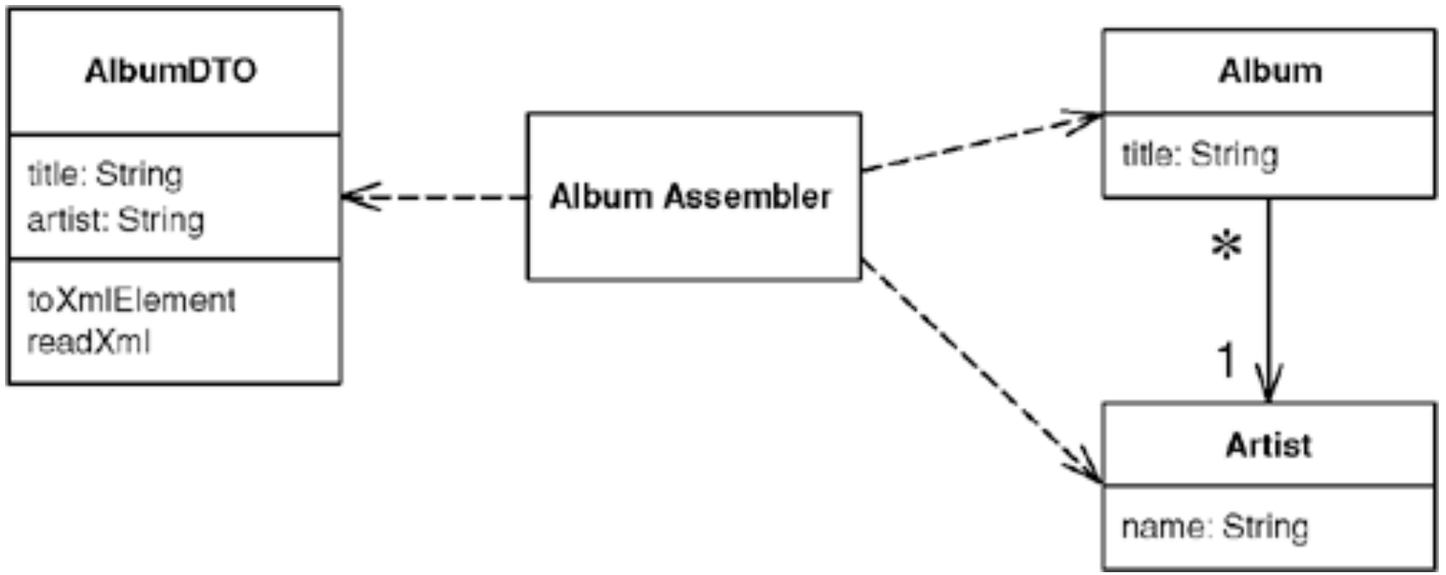
Data Transfer Object

Data Transfer Object



Data Transfer Object

- “An object that carries data between processes in order to reduce the number of method calls.”



How It Works

How It Works

- DTOs are basic objects
    - fields with getters and setters
    - contain no domain logic
        - These would be structs in c/c++

How It Works

How It Works

- These objects often contain more than just the simple data requested
    - but dependent data as well
    - done in order to delay (or eliminate) the necessity for another remote call

How It Works

How It Works

- Typically complex relationships will not be represented in the domain transfer object
    - just the data itself
- Usually can’t transfer domain model directly
    - things won’t always be serializable

How It Works

How It Works

- As the Remote Facade is usually geared towards a specific client's needs
    - corresponding domain transfer objects are the same way

How It Works

How It Works

- This leads to a lot of data transfer objects
- Can be avoided by just using a map to store everything
- Depends on the semantics you want to uphold

Types

Types

- Named field
- Record Set (Result Set) style
- Map (dictionary) style

Serializing

Serializing

- Can be in different formats
  - Binary / Language specific / custom
  - XML / other text based
- Other formats using external tools
- Can roll your own

Binary Serialization Interfaces

- CORBA / Java RMI
- Google Protocol Buffers
- Apache Thrift

XML

XML

- Very popular
    - text format
    - easy to read parse
- Takes up more space during transfer (Bandwidth requirements)
- Takes longer to parse (performance penalty)

JSON

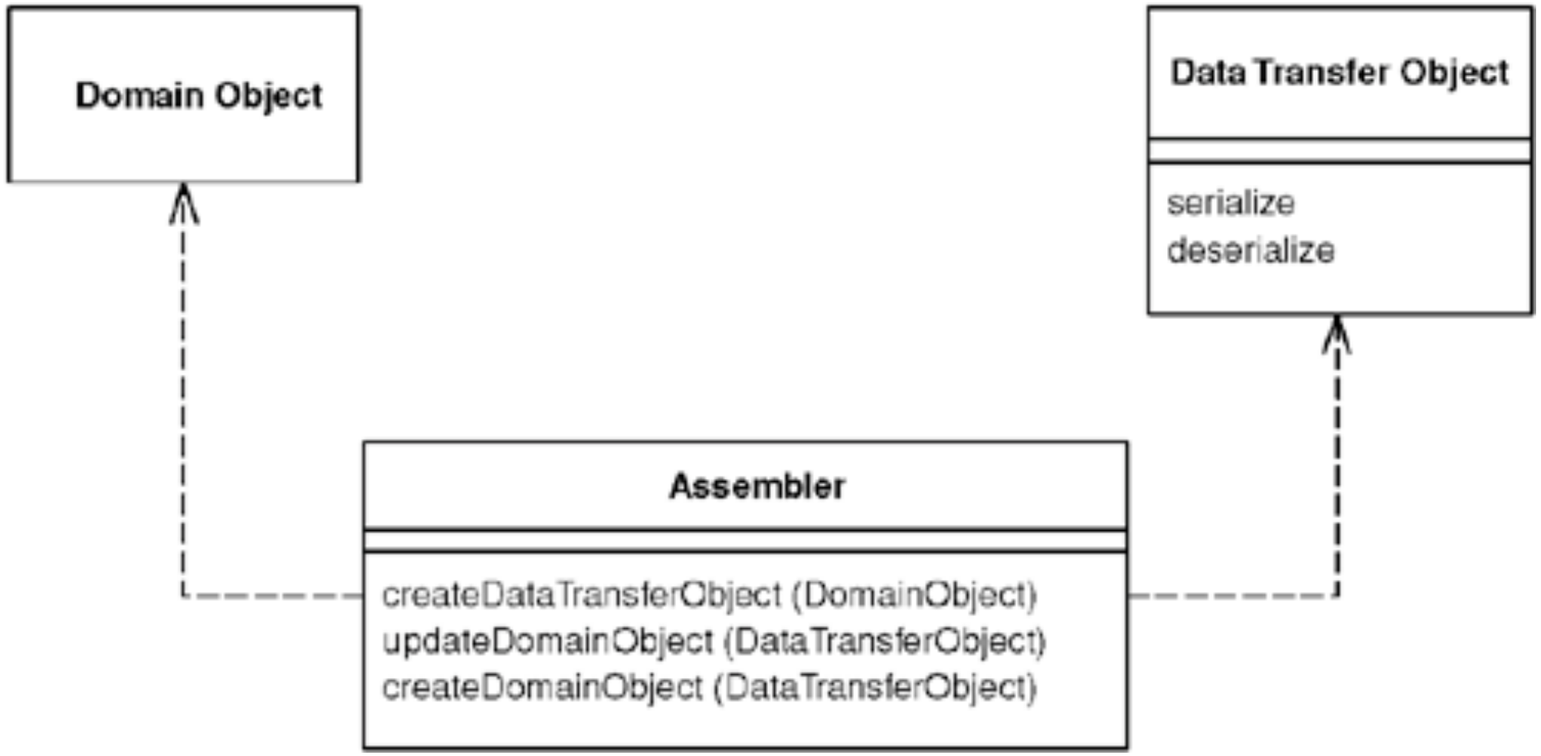
- Becoming more popular
- Used in AJAX and other RPC mechanisms
  - Human readable
  - Parsing can be slow (language dependent)

Assembly

Assembly

- DTOs are independent of the domain model objects
    - deployed on both sides of the wall
- Use an assembler object to map the domain object (and it's hierarchy / graph) to the data transfer object

Assembly



When to Use It

When to Use It

- When you need to transfer multiple object in a single method call
- When you don’t want to deal with the wire line types
    - we never want to do this...