LECTURE 8: SUBSYSTEMS

Wirfs Brock et al., *Designing Object-Oriented Software,* Prentice Hall, 1990. (Chapter 7)

Outline

- Subsystems: what and why?
- Subsystem documentation
  - Subsystems cards
  - Collaboration graphs
- Guidelines for defining subsystems

Tuesday's Quote

3

- Contracts submission is due on Sunday 03/11 @ 11:59
  - BY E-MAIL AND ON SMARTCLOUD
- GUI Evaluation is scheduled for class time on Thursday 03/22

Outline

4

- Subsystems: what and why?
- Subsystem documentation
  - Subsystems cards
  - Collaboration graphs
- Guidelines for defining subsystems

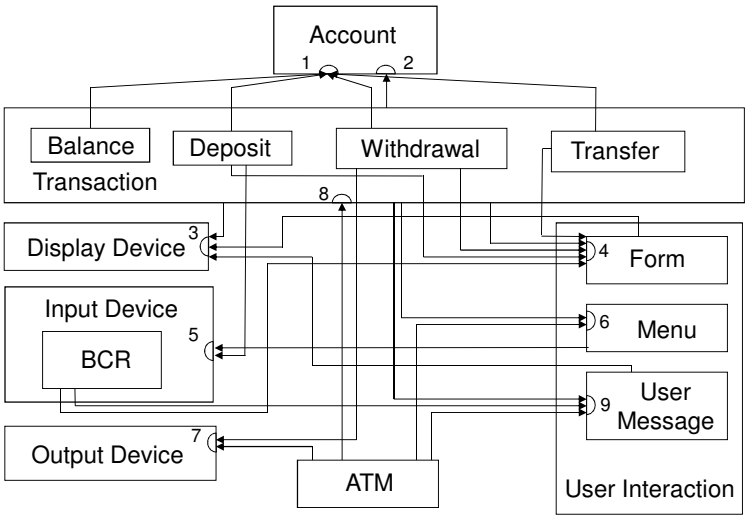
Steps for Producing Initial Designs

5

- Identify
  - Classes
  - Responsibilities
  - Collaborations
- Analyze
  - Class hierarchies
  - Contracts
  - Collaboration graphs

Motivation: Collaboration Graph

6



Problem

7

- Collaboration graphs get very complicated.
  - Too many lines (i.e., interactions or communications)
- Designs become incomprehensible.
- How to simplify the design, esp. patterns of communications?

Solution

8

- Need an abstraction tool to provide macro views
- Collect classes together to form *subsystems* that
  - Behave like a class.
  - Support services to outside via contracts.

What Are Subsystems?

9

- Groups of classes that collaborate among themselves to support a set of contracts
- Goal: To simplify patterns of communications

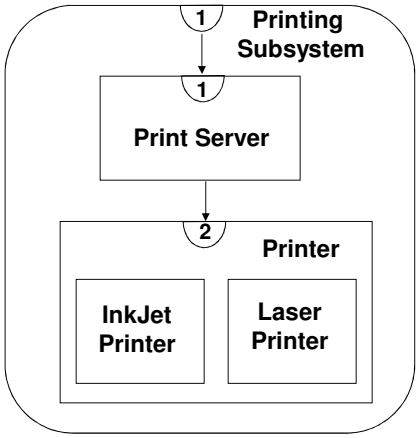
What Are Subsystems? (Cont.)

10

- Subsystems ***are not*** super classes.
- Subsystems are not "just a bunch of classes."
- Subsystems should provide a good abstraction.
- Subsystems should provide a clearly defined interface, called *subsystem contracts*.

Subsystem Contracts

11



- All contracts supported
  - **by** objects inside a subsystem
  - **for** objects outside the subsystem.
- Delegation of contracts

Subsystem Cards for Documenting Subsystems

12

- Write the subsystem name at the top
- List all classes in the subsystem
- Include ref. to the subsystem's position in the collaborations graphs
- Describe the purpose of the subsystem
- List contracts for which it is a server
- For each contract, identify the class or subsystem to which the contract is delegated

Subsystem Cards (Cont.)

13

Subsystem: Drawing Subsystem
Classes: Control Point, Drawing, Drawing Element
Collaborations Graphs: see Figure 4-6
Description: Responsible for displaying and maintaining the contents of the drawing
Contracts
1. Display itself
   Server: Drawing
2. Access and modify the contents of a drawing
   Server: Drawing
3. Modify the attributes of a Drawing Element
   Server: Control Point

How to Identify Subsystems?

14

- Bottom-up and top-down approaches
- Bottom up
  - Start with classes and responsibilities
  - Identify collaborations
  - Partition the classes based on patterns of collaborations
  - This approach is useful when managing the complexity as a system grows.

Subsystem Identification

15

- Draw collaboration graph (use white board).
- Look for strongly coupled classes.
- Look for ways to simplify your description of the system.
- Look for clean separations.
- Look for good abstractions.

How to Identify Subsystems?

16

- Top down
  - Look at high level functions of system
  - Look at data sources and uses
  - Look at supporting technologies
  - Partition to manage complexity and reduce coupling
  - This approach may be useful when managing the complexity imposed by initial specification.

Guidelines for Simplifying Interactions

17

- Minimize the number of collaborations a class has with other classes or subsystems.
- Minimize number of classes and subsystems to which a subsystem delegates.
- Minimize number of contracts supported by a class or subsystem.

G-1: Minimize Number of Collaborations

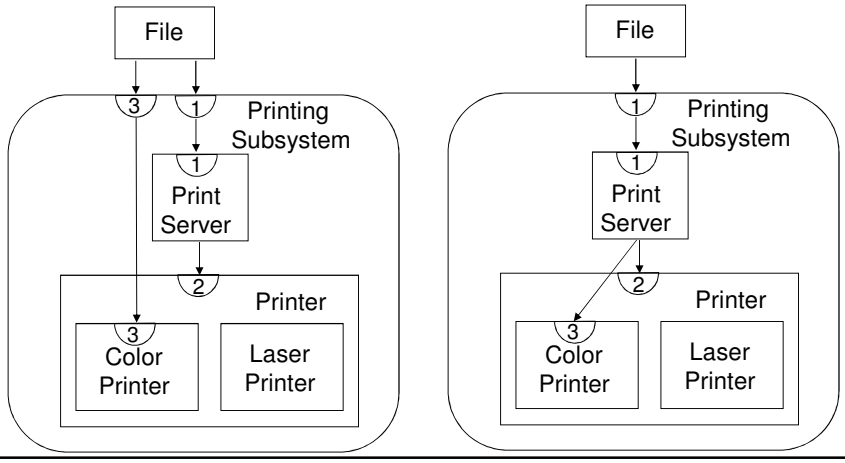
18

- Class should collaborate with as few other classes and subsystems as possible. (Why?)
- Heuristic: Centralize communications

Example: Minimize Number of Collaborations

19

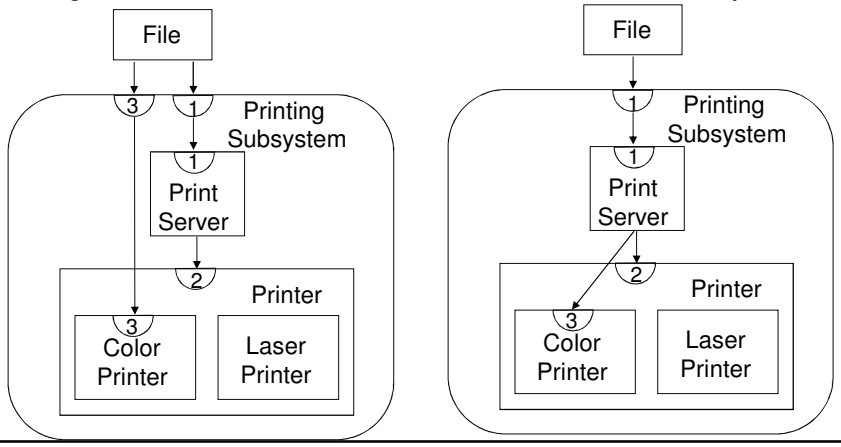
Minimize the number of collaborations a class has with other classes or subsystems.



G-2: Minimize Delegations of Subsystem Contracts

20

- Keep the number of classes inside the subsystem that support subsystem contracts to a minimum
- Again, centralize communications into and out of the subsystem



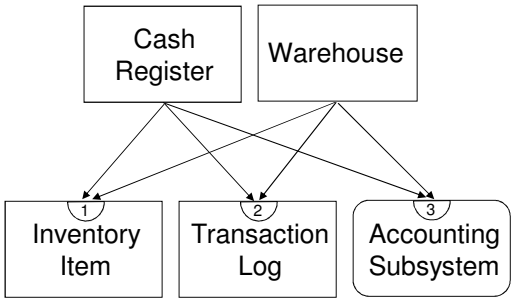
G-3: Minimize Number of Contracts

21

- Too many contracts in one class/subsystem indicate:
  - Too much intelligence concentrated in one place
  - Split the functionality between two or more classes.
- Re-examine the collaboration patterns

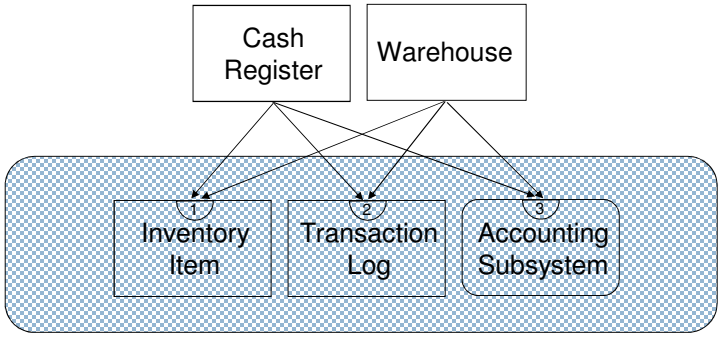
Example

22



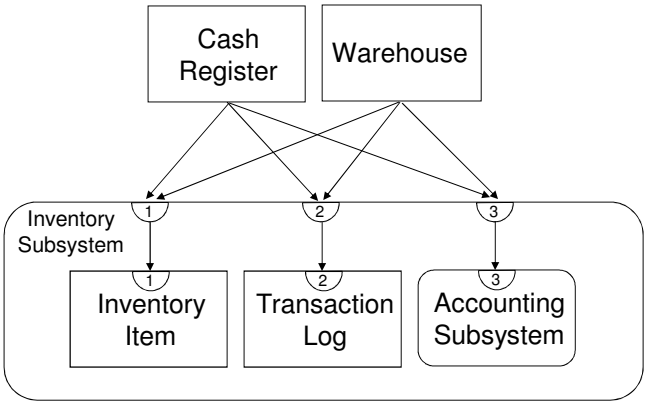
Example

23



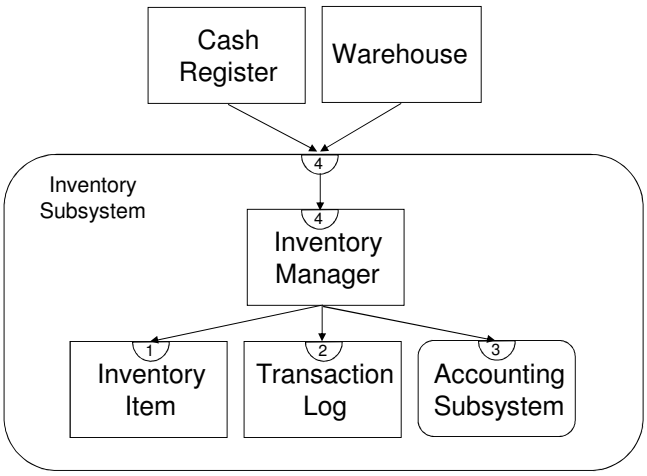
Example Refined

24



Example Refined Further

25



If You Have to Redesign ...

26

- Redraw the graphs
- Re-examine the collaboration patterns
- Walk through scenarios (all of them)
- Verify that things are simpler, have improved cohesion and reduced coupling

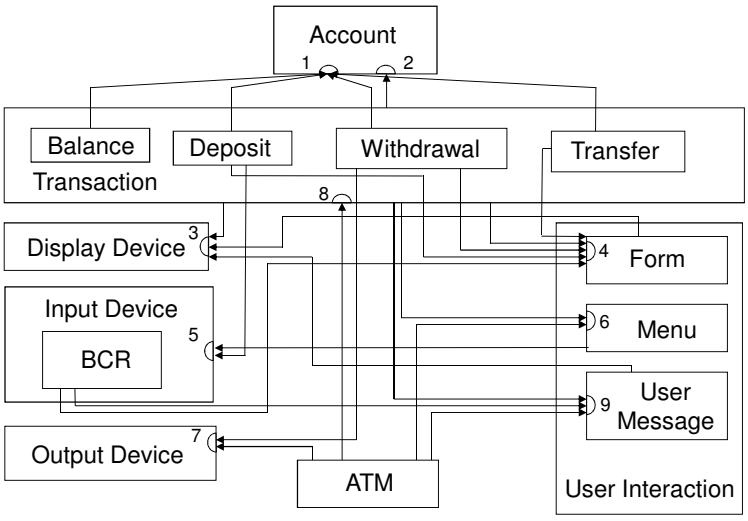
ATM Example: Contracts

27

1. Account: Access and modify balance
2. Account: commit result to database
3. Display: Display information
4. Form: Get numeric value from user
5. Input Device: accept user input
6. Menu: Get user selection
7. Output Device: output to the user
8. Transaction: execute financial transaction
9. User Message: display message and wait

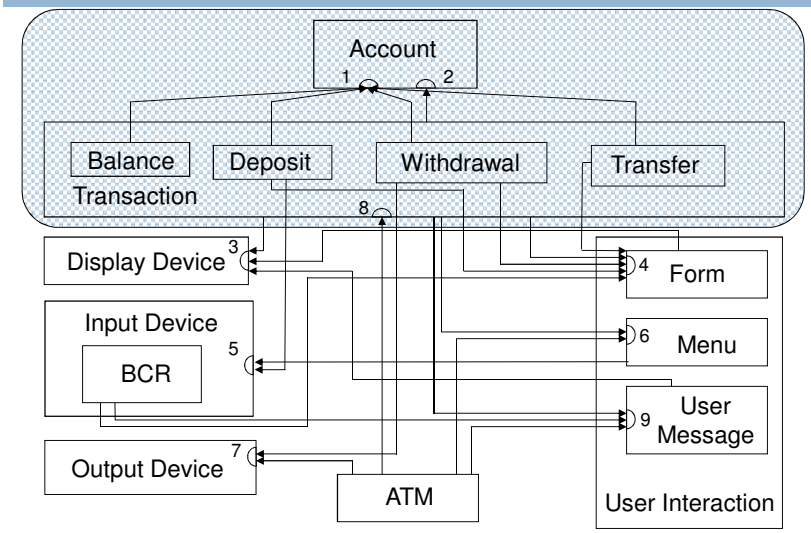
ATM: Collaboration Graph

28



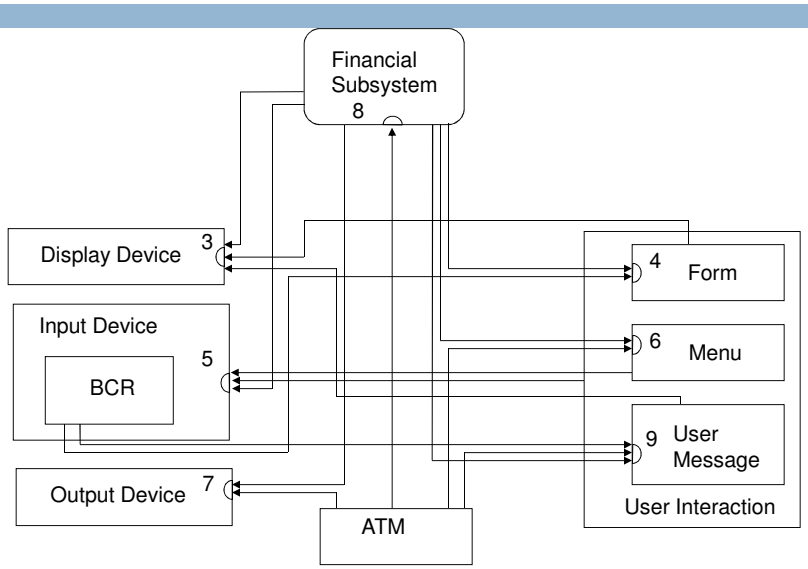
Refining

29



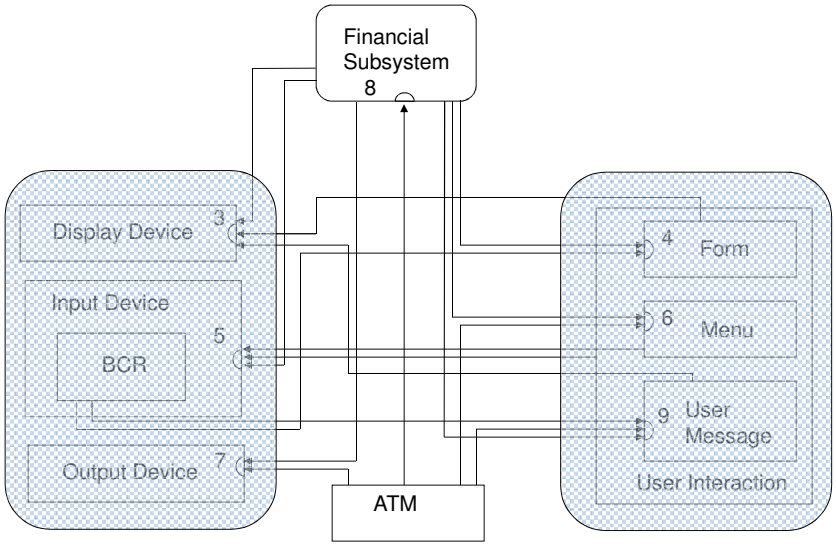
Simplified Graph 1

30



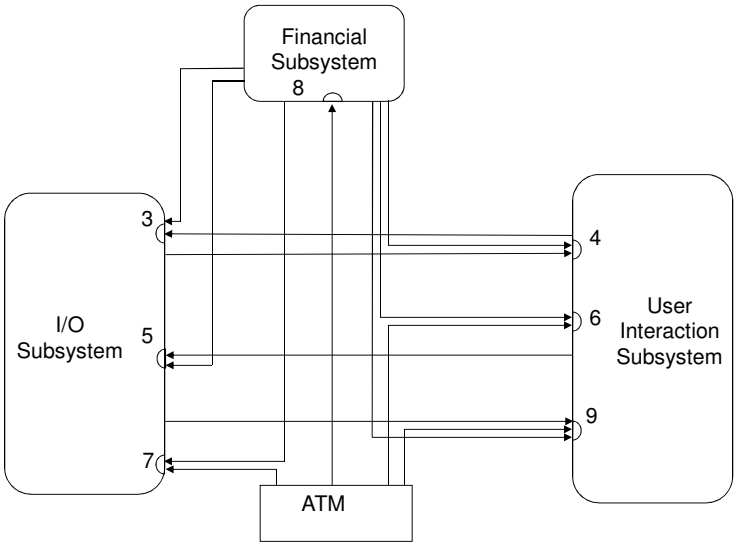
Refining

31



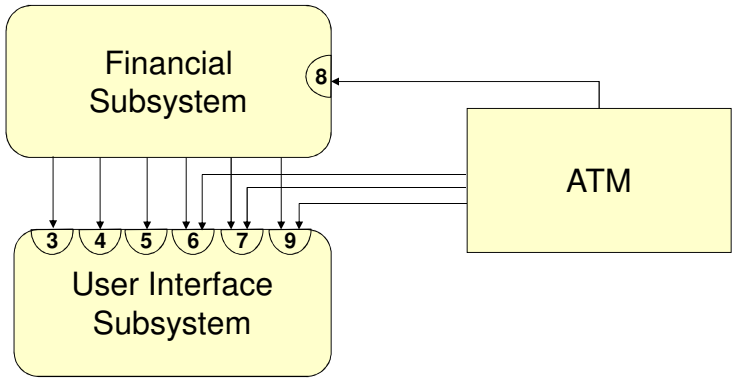
Simplified Graph 2

32



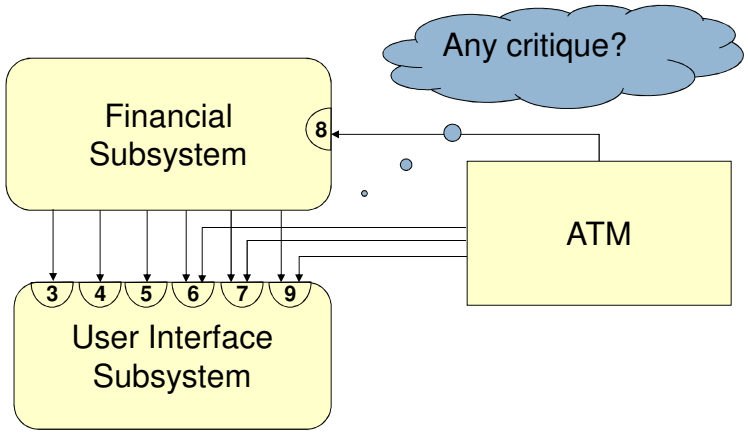
Even More Simplified Graph

33



Even More Simplified Graph

34



Rework

35

- Rework your design. If it isn't changing, you're not working.
- Do not change just to change, but change based on recognition of better design.
- Take pride in your work.

Group Work and Assignment

36

- Subsystems for the project
  - Complete the Subsystems document
    - Due on Sunday 03/25/2018
    - **Subsystems** by grouping classes and identifying subsystem contracts (subsystem cards)
    - **Collaboration graphs**
      - Subsystem Collaboration Graph
  - Leader: Architect

Recurring Problem

- Building User Interfaces
- How common is this?
- How often does it change?
- What do we do with something that changes a lot?
- Can you relate to this??

Motivation:

- Suppose we support both the command line and the GUI interfaces.
- What changes?
- What stays the same?

- This sounds familiar

Motivation:

- Suppose we support both the command line and the GUI interfaces.
- What changes?
- What stays the same?
- How do you design to handle this?

Model-View Separation

- Model: The domain layer of objects. (objects that contain data and operations).
- View: The presentation layer of objects (windows, applets, reports).

Model-View Context

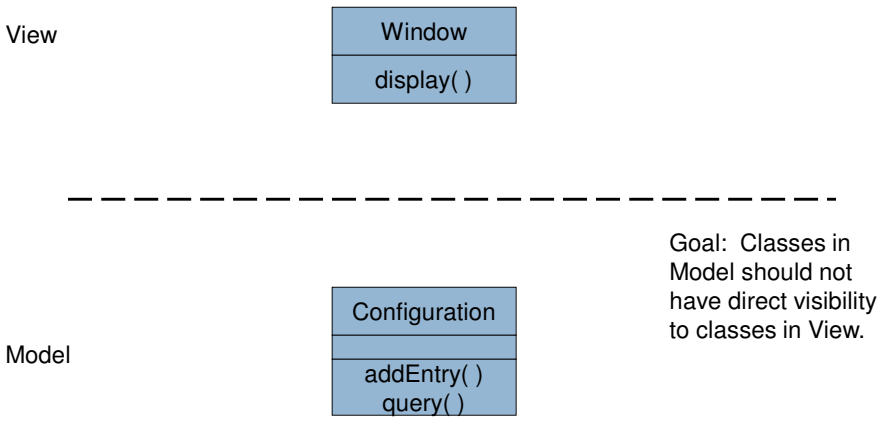
- Context/Problem

  It is desirable to de-couple domain (model) objects from windows (views), to support increased reuse of domain objects, and minimize the impact of changes in the interface upon domain objects.

- Solution

  Define the domain (model) classes so that they do not have direct coupling or visibility to the window (view) classes, and so that application data and functionality is maintained in domain classes, not window classes.

Model-View

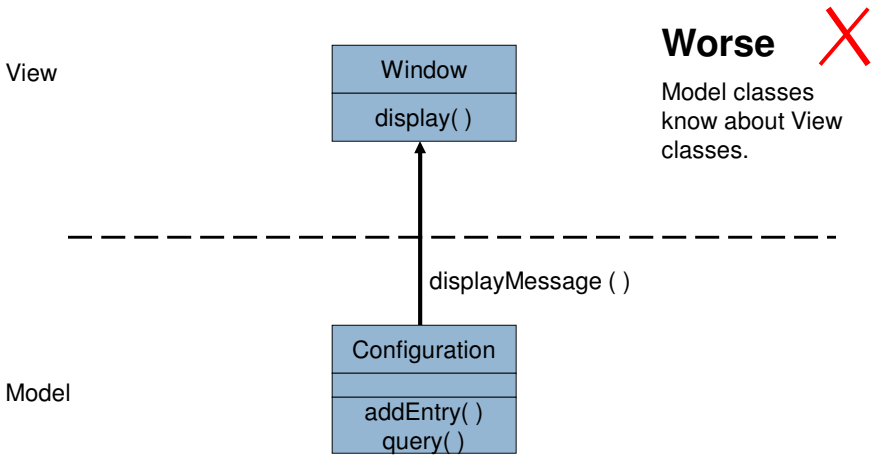


Model-View Separation Motivation

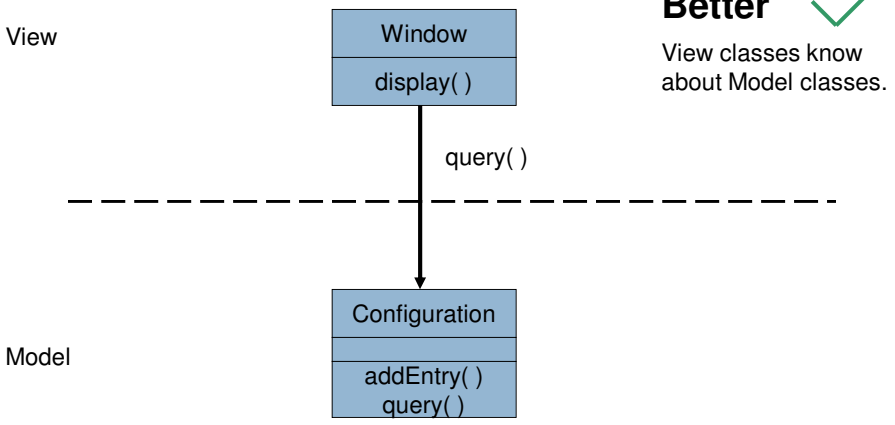
Motivation

- Focus more on the domain processes rather than on computer interfaces.
- Allow separate development of the model and user interface layers.
- Minimize the impact of changes in the interface upon the domain layer.
  - Which is one of the most common changes in software
- Allow new views to be easily connected to an existing domain layer.

Problem



Model-View Separation Pattern



Better

View classes know about Model classes.

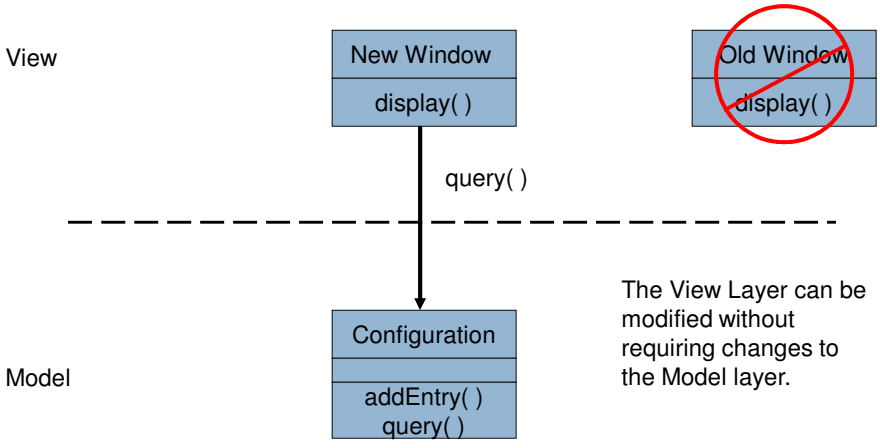
Question

- Why is this better???

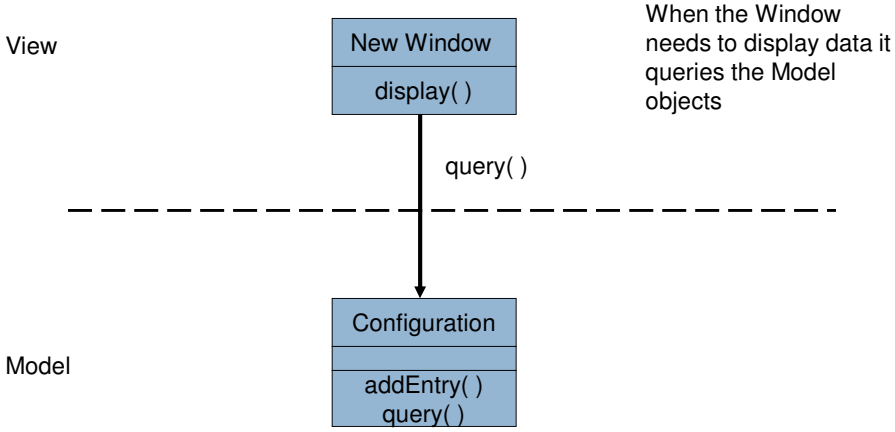
Question

- Why is this better???
  - Views change more often, this minimizes the impact of change.
  - We design for change.

Model-View Separation Pattern



Model-View Separation Pattern



Model-View Separation Pattern

- Problem: Domain objects need to communicate with windows to cause a real-time ongoing display update as the state of information in the domain object changes.
  - Monitoring applications
  - Simulation applications.
- Solution: Indirect Visibility

Model-View Separation Pattern with Indirect Visibility

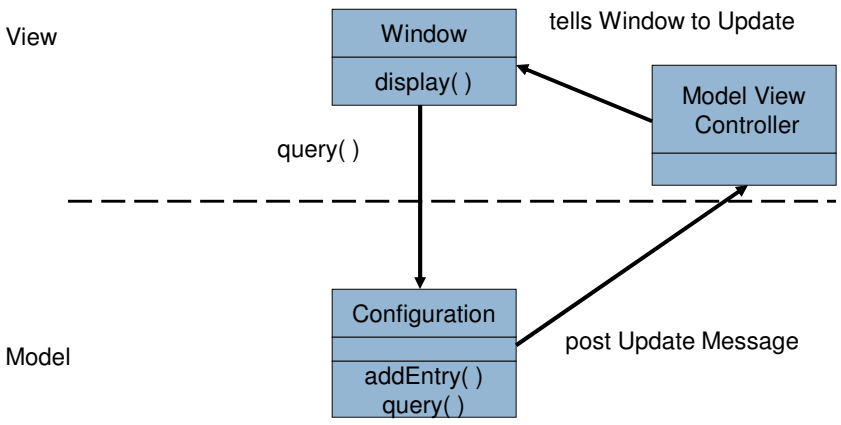
- Named Publish-Subscribe ***Architectural*** Pattern
- Context/Problem:

  A change in state (an event) occurs within a Publisher of the event and other objects are dependant on or interested in this event (Subscribers to the event). However the Publisher should not have direct knowledge of its subscribers.

- Solution:

  Define an event notification system so that the Publisher can indirectly notify Subscribers. Event Manager or Model View Controller (MVC).

Model-View Separation Pattern with Indirect Visibility



MVC

- MVC is an architecture pattern
- An architecture is the higher level design of a system
  - Meaning it's the way a software system is structurally organized, and the communication paths between components is defined
- Other examples include: Layered, Client-Server, Pipe and Filter.