More Design Techniques

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 33

Acknowledgement & Materials Copyright

- I’d like to start by acknowledging Dr. Ken Anderson
- Ken is a Professor and the Chair of the Department of Computer Science
- Ken taught OOAD on several occasions, and has graciously allowed me to use his copyrighted material for this instance of the class
- Although I will modify the materials to update and personalize this class, the original materials this class is based on are all copyrighted © Kenneth M. Anderson; the materials are used with his consent; and this use in no way challenges his copyright

Goals of the Lecture

- Look at using Commonality and Variability Analysis
- Look at an alternative design approach - The Analysis Matrix

More Design Techniques

- You may not always be able to use design patterns to start your designs
  - You can however apply the lessons learned from studying design patterns in ALL of your designs
- For instance, you can apply commonality and variability analysis to identify variation in your problem domains
  - You can then use design pattern principles to encapsulate that variation and protect your software from potential changes
- This example came from the original class textbook, but it’s also available in some detail here:
  - http://www.netobjectives.net/files/pdfs/Introduction_CommonalityVariabilityAnalysis.pdf

Examine the Problem Domain (I)

- One key aspect of design is identifying *what* elements of the **problem domain** *belong* in your **solution domain**
  - You need to identify the right things to model (or track) in your software
  - You want to do this as accurately as possible because the next step is to identify the relationships between those concepts
  - Once you have the relationships defined, changes to the design become more difficult

Examine the Problem Domain (II)

- Once you have concepts and relationships defined, inserting new concepts and relationships is less easy
  - You have to decide where the new concepts “fit” and how they will be integrated into the existing design
  - Existing relationships may change or be removed and new ones will be inserted
- This is why maintenance is so hard, when you are asked to change existing functionality or add new functionality, you must try to understand and deal with the web of concepts and relationships that already exist in the system

Use Commonality and Variability Analysis

- Commonality and Variability Analysis (CVA) attempts to identify the commonalities (generic concepts) and variations (concrete implementations) in a problem domain
  - Such analysis will produce the abstract base classes that can be used to interface with the concrete implementations in a generic way that will enable abstraction, type encapsulation and polymorphism
- We can demonstrate/explain the technique by example by applying it to the previously shown CAD/CAM problem

Applying CVA to CAD/CAM

- The CAD/CAM problem consists of
  - Version 1 and Version 2 of the CAD/CAM System
  - Slots, holes, cutouts, etc.
  - Version 1 Models and Version 2 Models
- These are **concrete variations** of **generic concepts**
  - CAD/CAM system, Feature, Model
- Generically: Commonality C has variations a, b, c

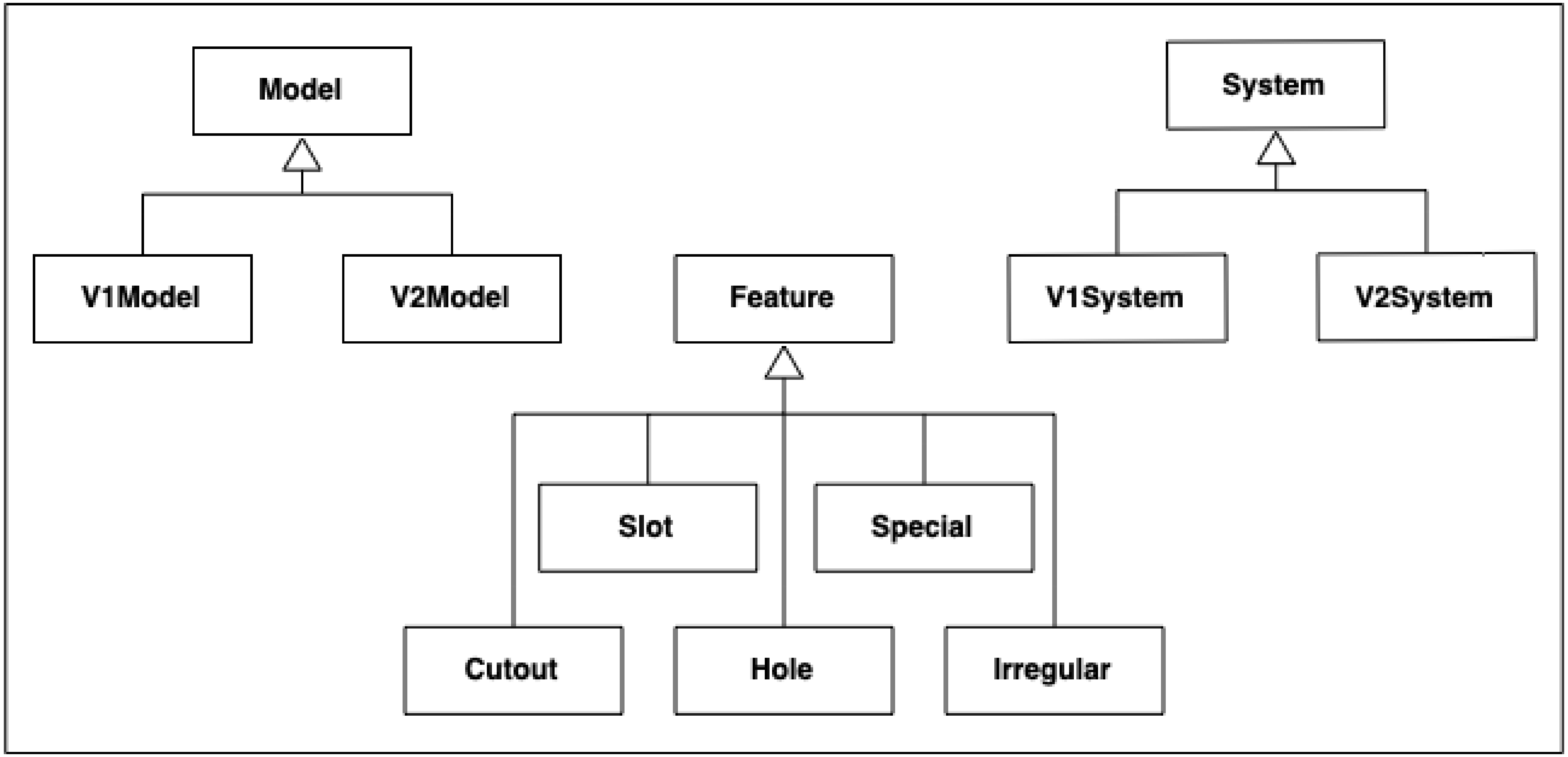
Another technique

- Take any 2 elements of the problem domain
  - And ask
  - Is one of these a variation of the other?
  - Are both of these a variation of something else?
- Iterate until you start to converge on the commonalities
  - Each with their associated variations, which are just concrete elements of the domain

Potential Problem

- Each commonality should be based on one issue
  - Beware collapsing two or more issues into a concept
- For the CAD/CAM domain, you might do something like
  - CAD/CAM Features
    - V1Slot, V2Slot, V1Cutout, V2Cutout
- Here you have collapsed “feature” and “version” into a single concept
- What you need is
  - CAD/CAM System
    - V1 and V2 (versions)
  - Feature
    - Slots, cutouts, holes, etc.
- Now you have one issue per concept
  - and this will lead to more cohesive designs

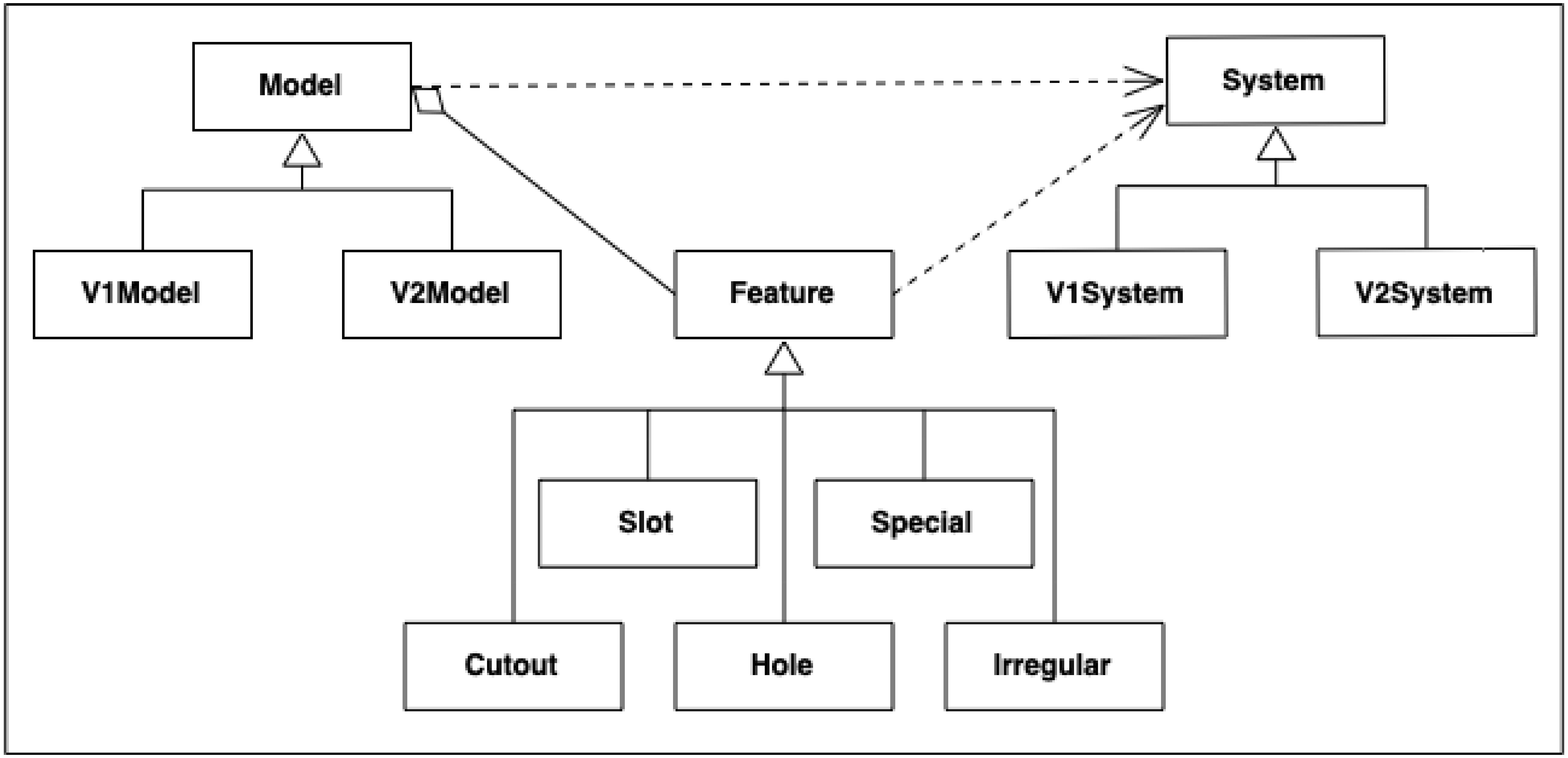
Translating to Classes



Identify Relationships

- If you are confident that you have identified the major concepts of the domain and their variations, you are then ready to identify the relationships that exist between them
    - Models are aggregations of Features
    - Models are generated from a CAD/CAM System
    - Features are generated from a CAD/CAM System
    - We will represent “is generated from” as a uses relationship, since it is conceivable that once these concepts are instantiated, they access the CAD system from time to time

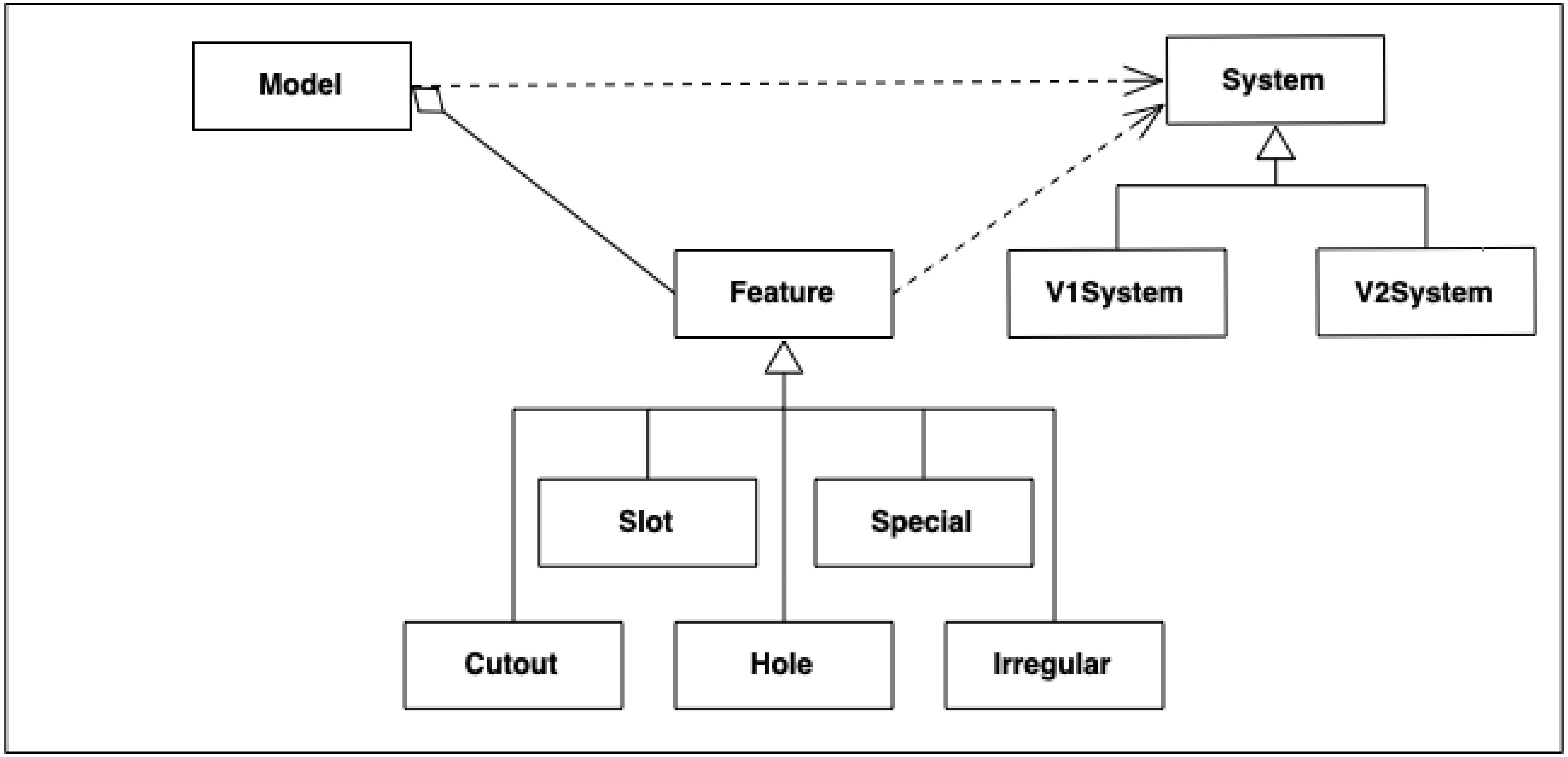
Translating to Class Diagram



Reflect on the Variations

- We have a duplication in the current design
  - V1Model and V2Model
- AND
  - V1System and V2System
- We can remove this duplication by deciding that a model produced by V1 will have features that are different than a model produced by V2 and so we can get by with just the variation in the CAD/CAM system

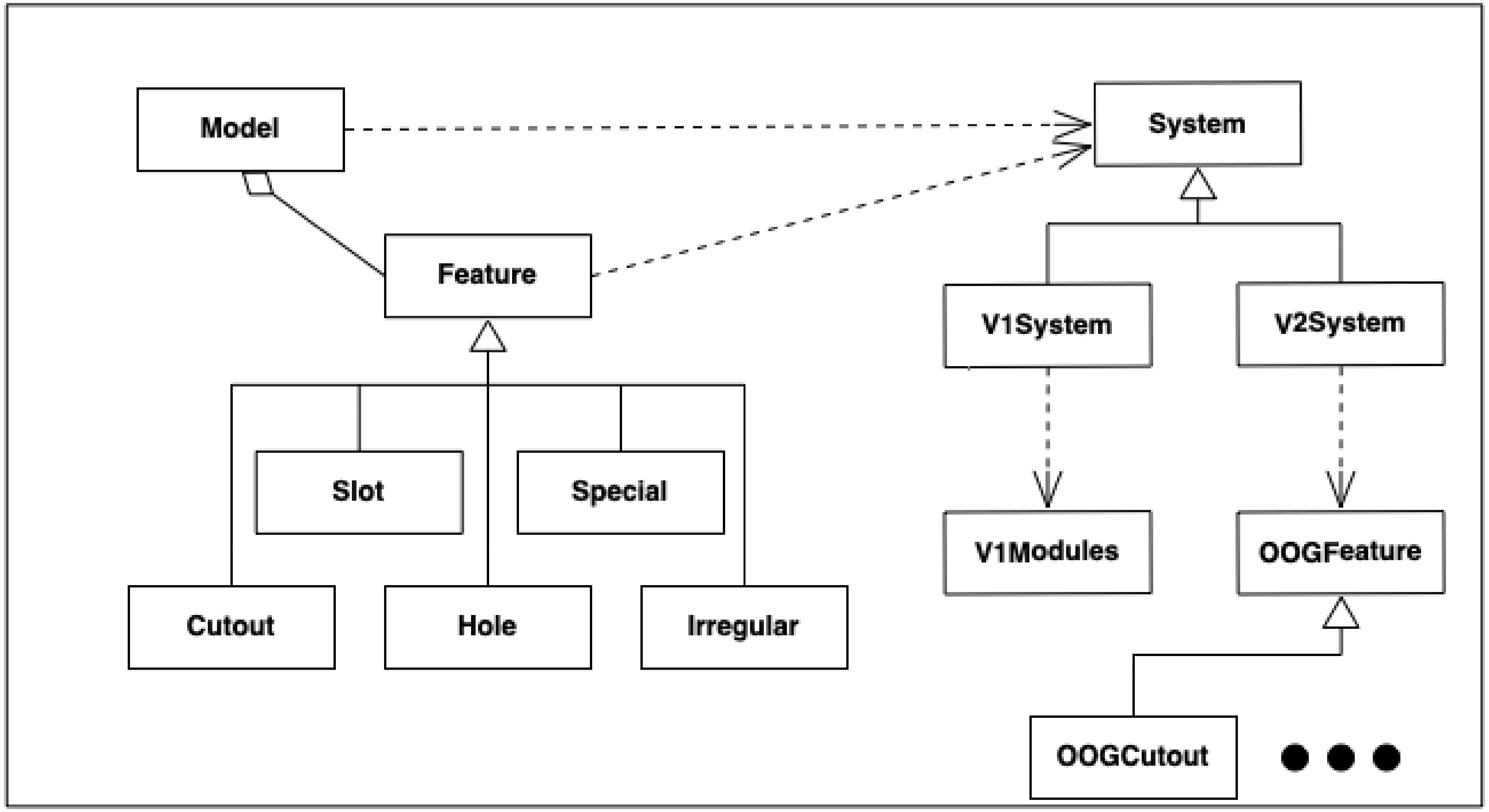
Updated Class Diagram



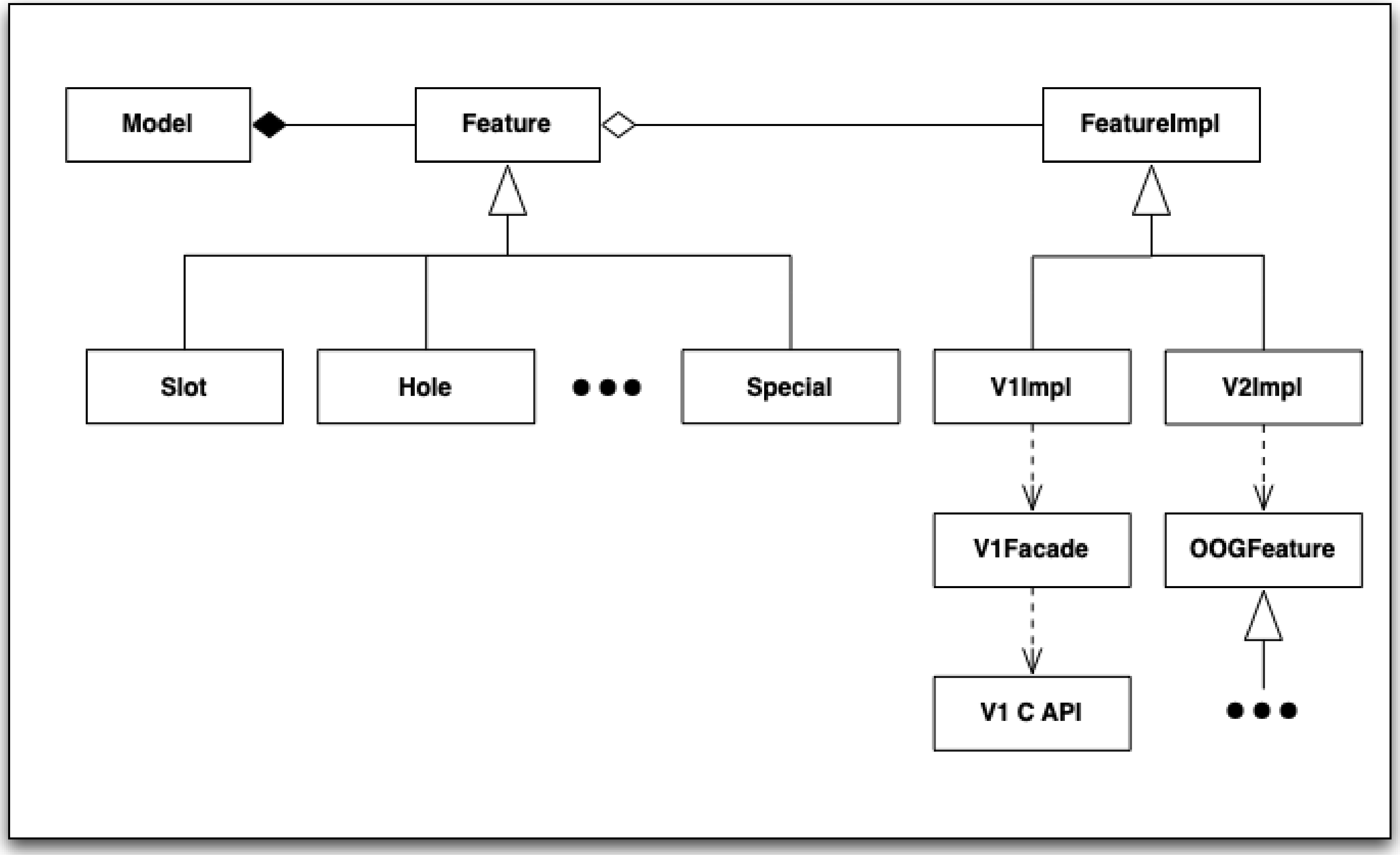
Connect to the Actual Systems

- We must now connect this design to the actual CAD/CAM Systems
  - CAD/CAM Version 1 has the C API
  - CAD/CAM Version 2 has the OO API

Updated Class Diagram



Compare with Previous Design



Comparison with Design Pattern Approach

- CVA produced a design that is similar to the design we created using an approach that was driven by design patterns
    - Identify patterns that provide context for other patterns
- The two approaches are synergistic and can be used in tandem
    - Design Patterns establish relationships between entities in the problem domain
    - CVA identifies entities in the problem domain and whether one entity is a variation of another
- The difference is that CVA can be used in all design contexts
    - whereas the design pattern approach requires that you know of design patterns that match the relationships found in the problem domain

Another Technique: Analysis Matrix

- There is another OO design technique known as the Analysis Matrix
- It’s designed to help designers deal with large amounts of variations in a problem domain
- The example we’ll look at is an e-commerce system example in which packages must be shipped to customers in response to orders
  - where different rules become active depending on the countries involved in the order

- For more information, there’s a presentation by the original class textbook author at (starts on pg. 45):
  - https://www.slideshare.net/TechWellPresentations/mi-30537872

Variations

- Real world domains have a lot of variations
- Patients always check in to a hospital before they are admitted
  - UNLESS it is an emergency
  - IN WHICH CASE they go to the emergency room to get stabilized and
  - THEN get admitted to the hospital
  - (IF REQUIRED)

Just like in CVA, Find Concepts

- When dealing with lots of variations, you still ask the question
  - what concept is this “thing” a variation of
- To organize this work, you create a matrix
  - a concept becomes a “row header”
  - a variation is an entry in the matrix
  - related variations go into a column
    - and the column header groups the variations by a particular scenario relevant to the problem domain
- The input to this process are the requirements gathered from the customer
  - For the e-commerce system, we might have reqs like:
    - Calculate shipping based on the country
    - In the US, state and local taxes apply to the orders
    - In Canada, use GST and PST for tax
    - Use U.S. postal rules to verify addresses
    - …

Organize by Matrix

	U.S. Orders	Canadian Orders
Calculate Freight	Use U.S. Postal Rates	Use Canadian Rates
Verify Addresses	Use U.S. Postal Rules	Use Canadian Rules
Calculate Taxes	Use State and Local Tax Rates	Use GST and PST
Money	U.S. Dollars	Canadian Dollars

Organize by Matrix

	U.S. Orders	Canadian Orders
Calculate Freight	CONCRETE IMPLEMENTATIONS OF SHIPPING RATES	
Verify Addresses	CONCRETE IMPLEMENTATIONS OF POSTAL RULES	
Calculate Taxes	CONCRETE IMPLEMENTATIONS OF TAX RULES	
Money	GENERIC CLASS THAT HANDLES CURRENCIES	

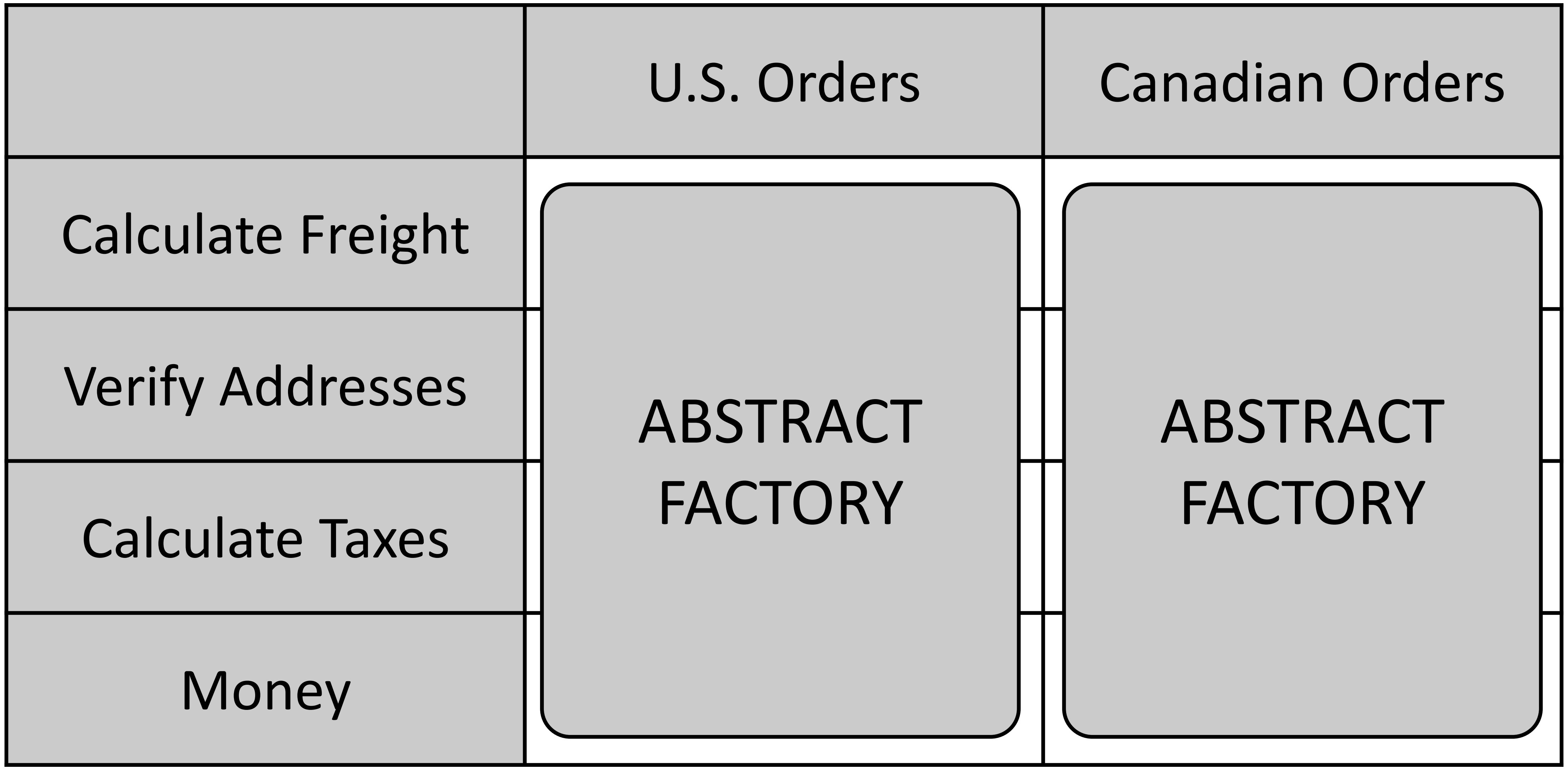
Organize by Matrix

	U.S. Orders	Canadian Orders
Calculate Freight	STRATEGY PATTERN	
Verify Addresses	STRATEGY PATTERN	
Calculate Taxes	STRATEGY PATTERN	
Money	GENERIC CLASS THAT HANDLES CURRENCIES	

Organize by Matrix

	U.S. Orders	Canadian Orders
Calculate Freight	THESE IMPLEMENTATIONS ARE USED WHEN WE HAVE A U.S. CUSTOMER	THESE IMPLEMENTATIONS ARE USED WHEN WE HAVE A CANADIAN CUSTOMER
Verify Addresses		
Calculate Taxes		
Money		

Organize by Matrix



Discussion

- This technique gets more useful as the matrix gets bigger
  - If you have requirements for a new scenario that adds an additional row (concept) that you have not previously considered
    - this indicates that your previous scenarios were incomplete
    - you can now go back and fill in the missing pieces
- Sometimes your special cases will have special cases
  - In the U.S. different shippers may have different requirements and different fees
  - You can capture this information in another analysis matrix that shares some of the columns and rows of the original but which add additional concepts just for those special situations

Summary

- We’ve now seen several OO design techniques
  - The OO A&D/UML Iteration Approach
    - Requirements to use cases to UML diagrams in iterative cycles
  - Design Pattern-Driven Design
    - aka Thinking in Patterns
  - Commonality and Variability Analysis
  - Analysis Matrix
  - CRC Cards
- Which one should we use?
  - Whatever helps you make progress! You might use more than one and switch between them until they converge