

The University of Iowa

CS:2820 (22C:22)

Object-Oriented Software Development

Spring 2015

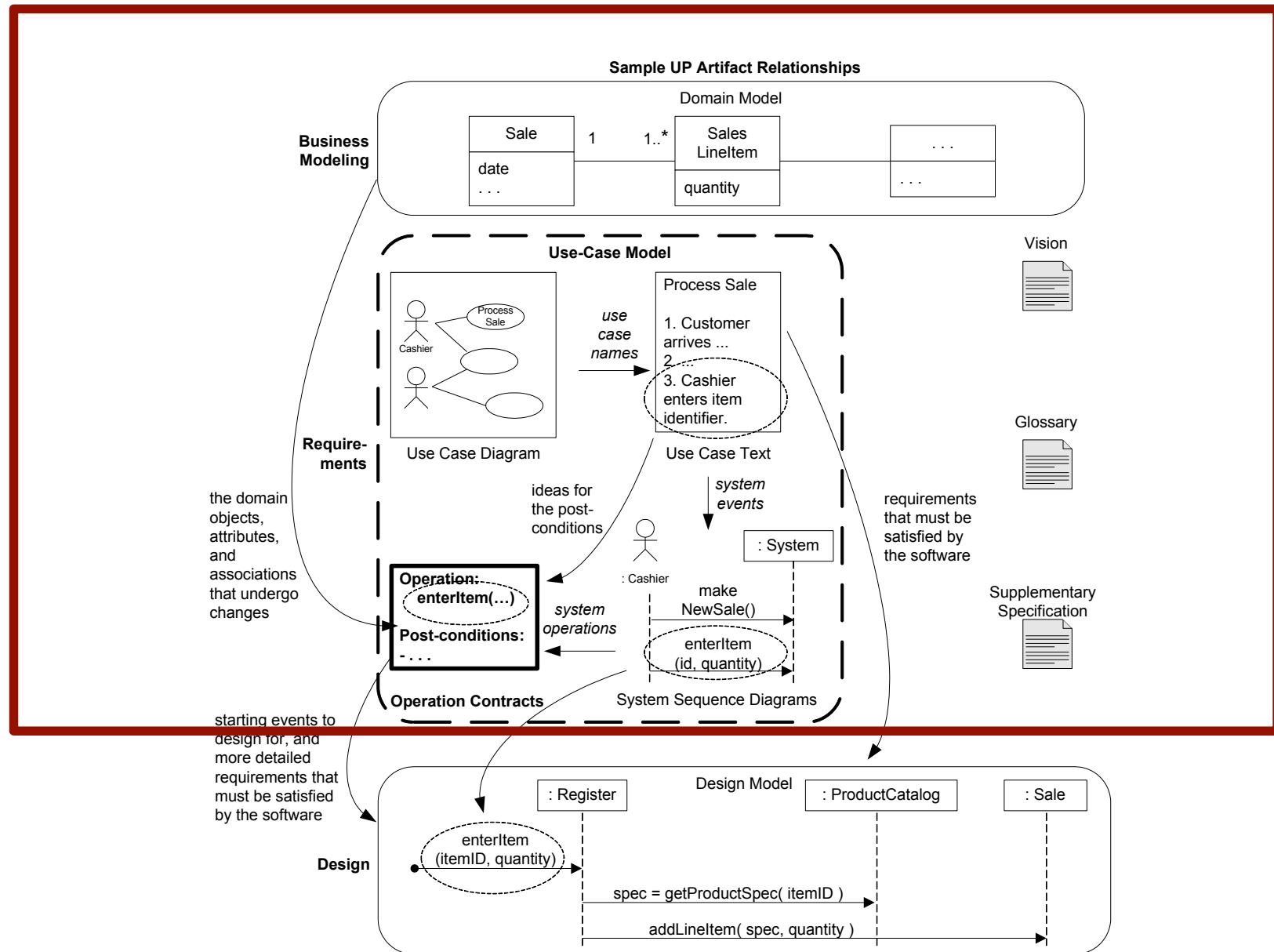
Transition to Design

(Chapters 12, 13, 14)

by

Mauricio Monsalve

Design road



Remember Agile

- Domain Model is covered—done?
- System Model is covered—done?
- Agile principles
 - Models can always be improved
 - Prioritized aspects of system
 - Work is meant to be partial initially!
- Iterate and improve!

Approach to Diagrams

- **Agile:** diagrams are sketch—accurate diagrams are not as important as advancing
- **Model Driven Engineering:** diagrams must be perfect (or very detailed) to generate code
- **CASE** tools can help MDE and Agile
- Split modeling, model in parallel (try alternatives)
- MDE on portions of diagrams—why not?

Divide and conquer

- Prioritize work (requirements, UCs)
- Model by parts
- Domain Model—use packages
 - Same topic? Same package
 - Related concepts? Same package
 - Relatedness:
 - *Same Use Cases*
 - *Hierarchies*
 - *Associations (coupling)*

Design: Architecture

- A.k.a. **logical design**
- Divide and conquer: **layered architecture**
- Traditional pattern: **MVC**
 - **Model**—underlying *logic* of the system
 - *Domain Model*
 - *Data Model*
 - *Logical Procedures—Algorithms*
 - **View**—user interfaces
 - **Controller**—execution control (flow)
 - *Execution algorithms (start, stop, etc.)*

Design: Architecture

- MVC is convenient for the UP: **Domain Model** corresponds to the Model part
- Domain Layer
- Convenient starting point for Design
- Low representational gap

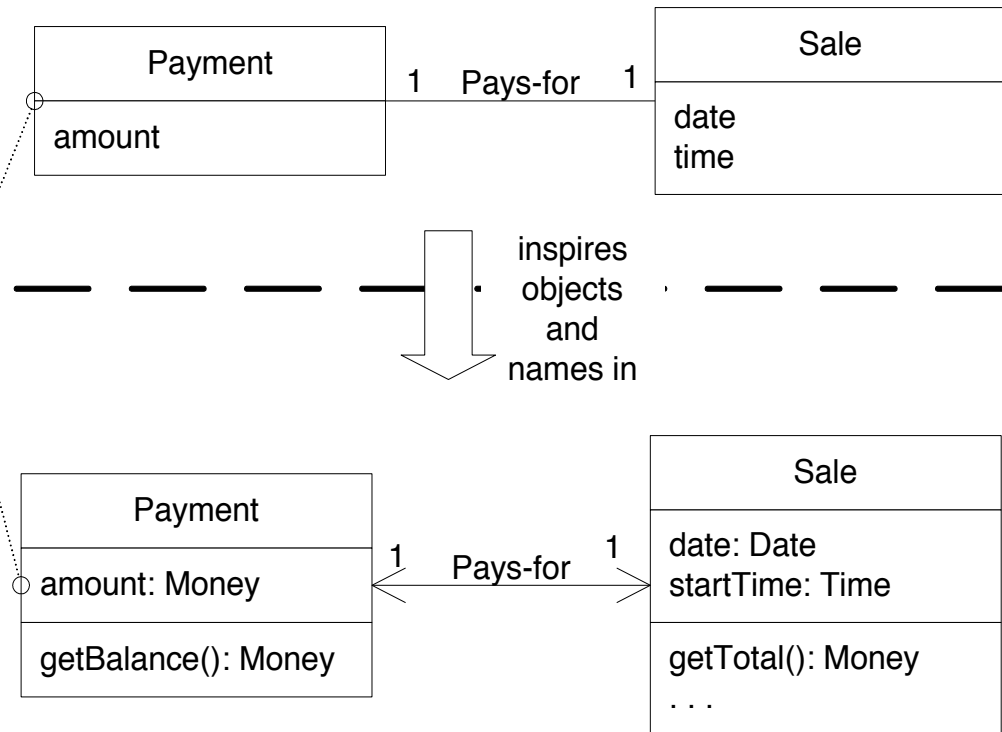
Design: Architecture

A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

UP Domain Model
Stakeholder's view of the noteworthy concepts in the domain.

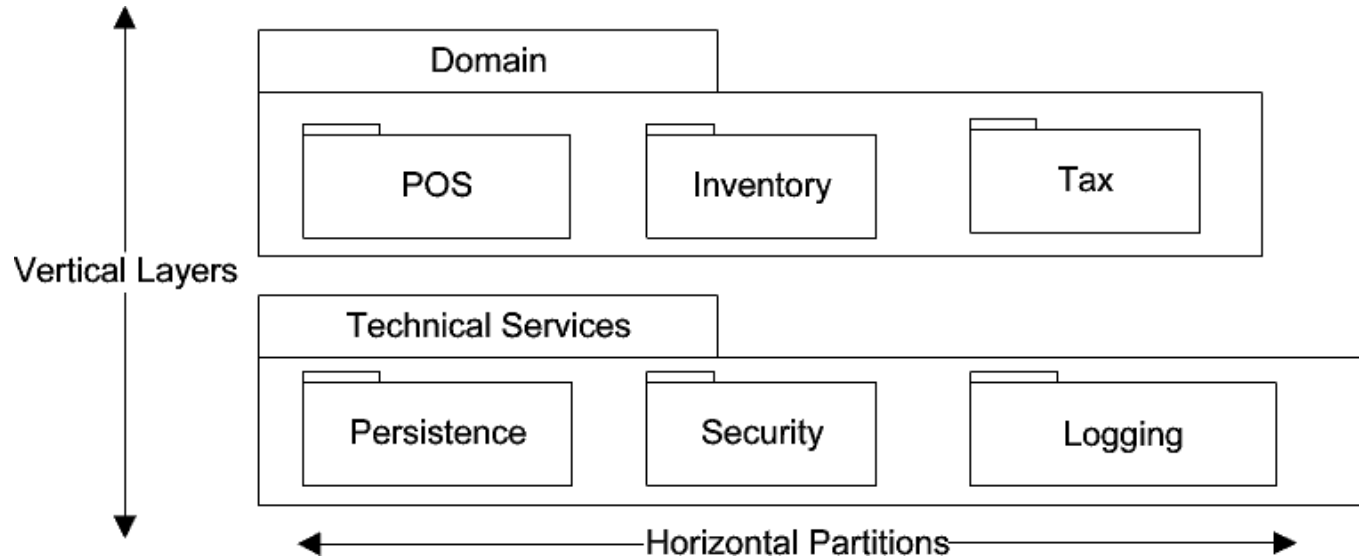


UP Design Model

The object-oriented developer has taken inspiration from the real world domain in creating software classes.

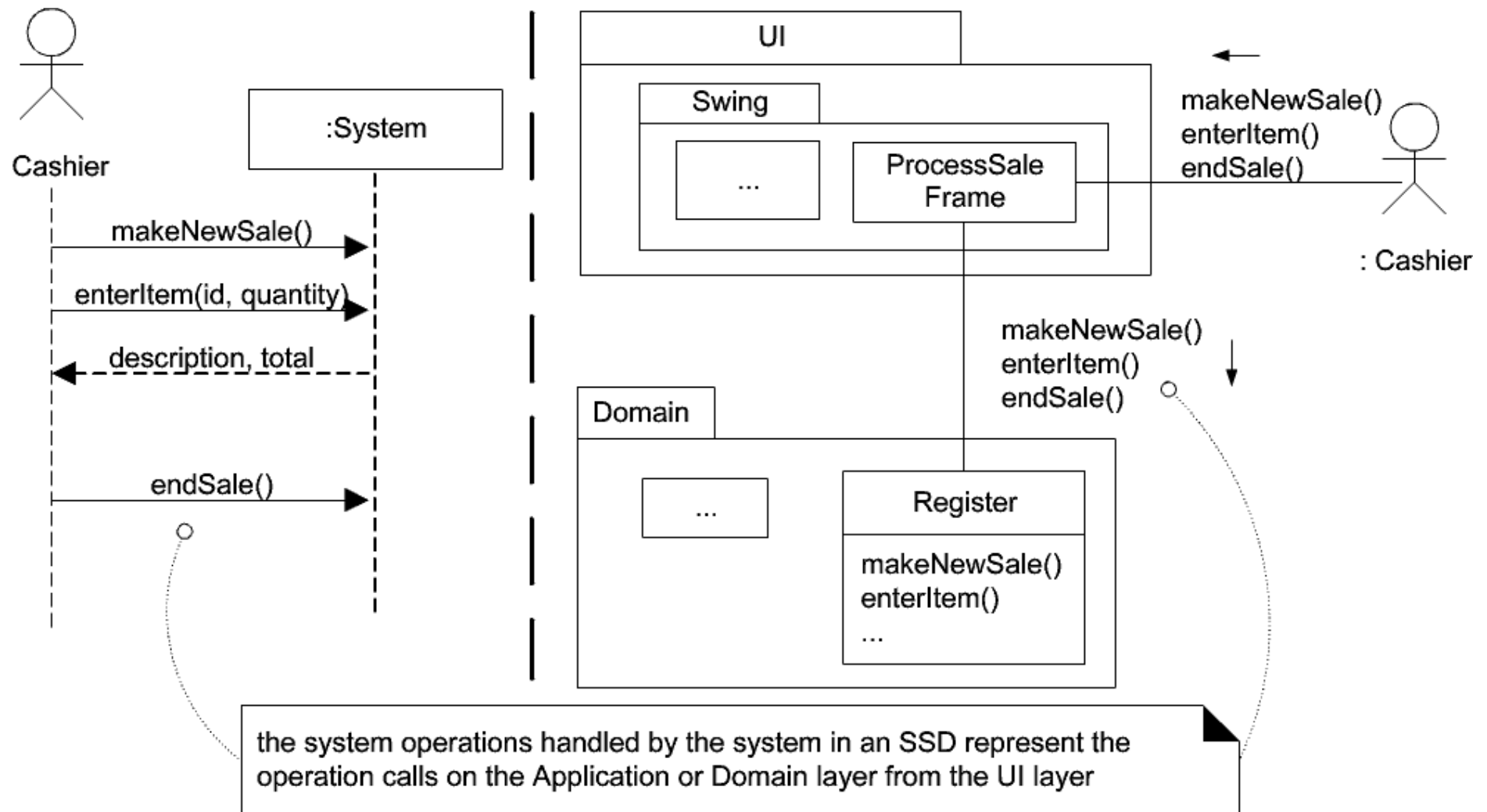
Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

Design: Architecture



- The architecture can be split into more layers, or each layer can be split into **partitions** (like the packages in the DM)

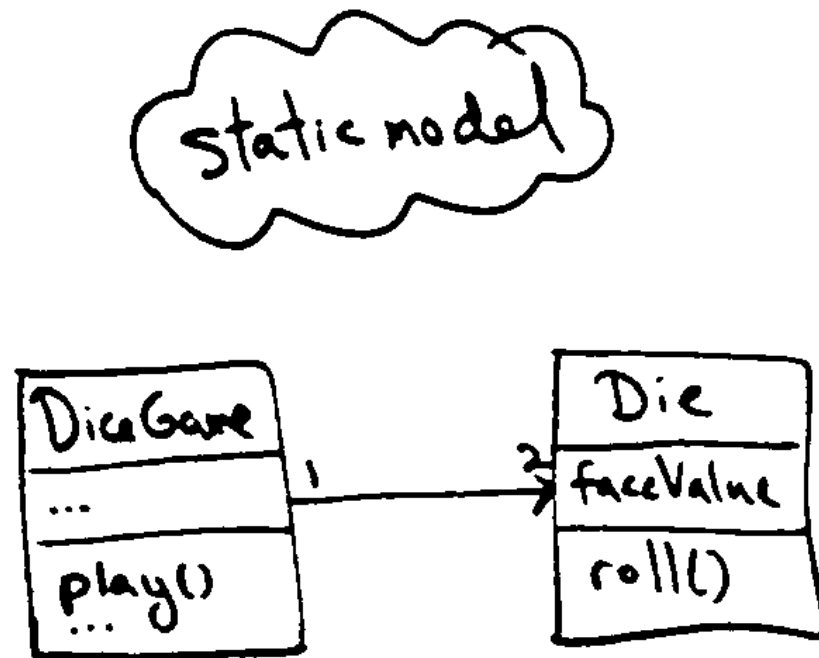
Design: Architecture



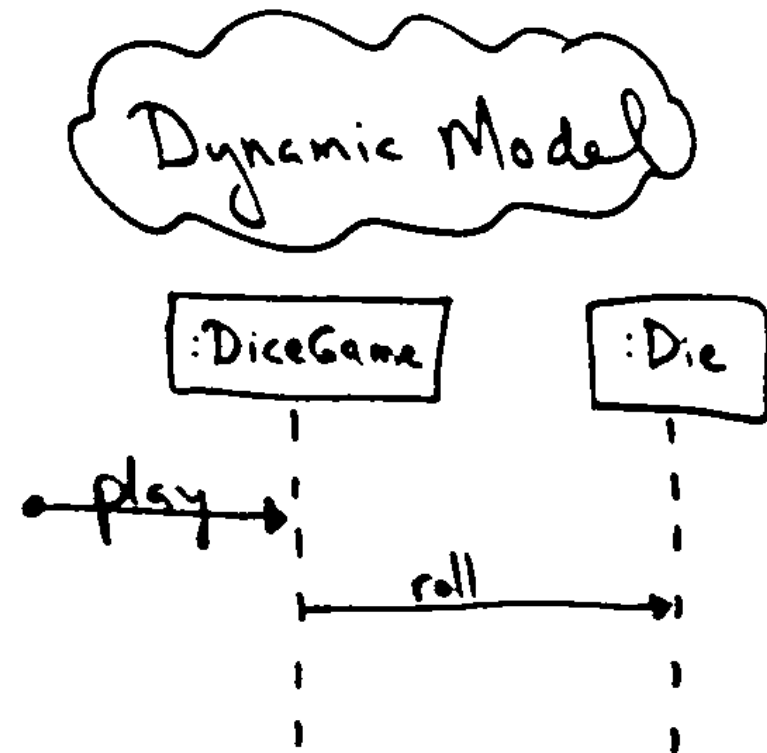
Getting the objects

- Domain Layer—we can *reuse* the concepts from the Domain Model
- Must add **operations**
- Operations can be identified through **interaction diagrams**
 - Sequence diagrams
 - Communication diagrams

Getting the objects



UML Class Diagram



UML Sequence Diagram

Getting the objects

- **CRC card approach**—brainstorming approach
- In papers (cards) put the CRCs:
 - **Class** name
 - **Responsibilities**—what the class does (e.g., operations) in plain language
 - **Collaborations**—other classes that are associated/interact with this one
- Agile strategy, often seen in XP

Getting the objects

Class Name

- Responsibility-1

- Responsibility-2

- Responsibility-3

Collaborator-1

Collaborator-2

Getting the objects

Group Figure

Holds more Figures.
(not in Drawing)

Forwards transformations

Cache image, void
on update of member.

Figures

Drawing

Holds Figures.

Accumulates updates,
refreshes on demand.

Figure
Drawing View
Drawing Controller

Selection tool

Selects Figures (adds
Handles to Drawing View)

Invokes Handles

Drawing Cache
Drawing View
Figures
Handles

Scroll tool

Adjusts the View's
Window

Drawing View

Credits

Notes and figures adapted from

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development by C. Larman. 3rd edition. Prentice Hall/Pearson, 2005.