DOMAIN MODEL VISUALIZING CONCEPTS

Biju R Mohan

Lecture 12

Agenda

- Identify conceptual classes related to the current iteration requirements.
- Create an initial domain model.
- Distinguish between correct and incorrect attributes.
- Add *specification conceptual classes, when appropriate.*
- Compare and contrast conceptual and implementation views.

What is domain model ?

- A domain model illustrates meaningful conceptual classes in a problem domain; it is the most important artifact to create during object-oriented analysis.
- Identifying a rich set of objects or conceptual classes is at the heart of object-oriented analysis, and well worth the effort in terms of payoff during the design and implementation work.
- The identification of conceptual classes is part of an investigation of the problem domain. The UML contains notation in the form of class diagrams to illustrate domain models.

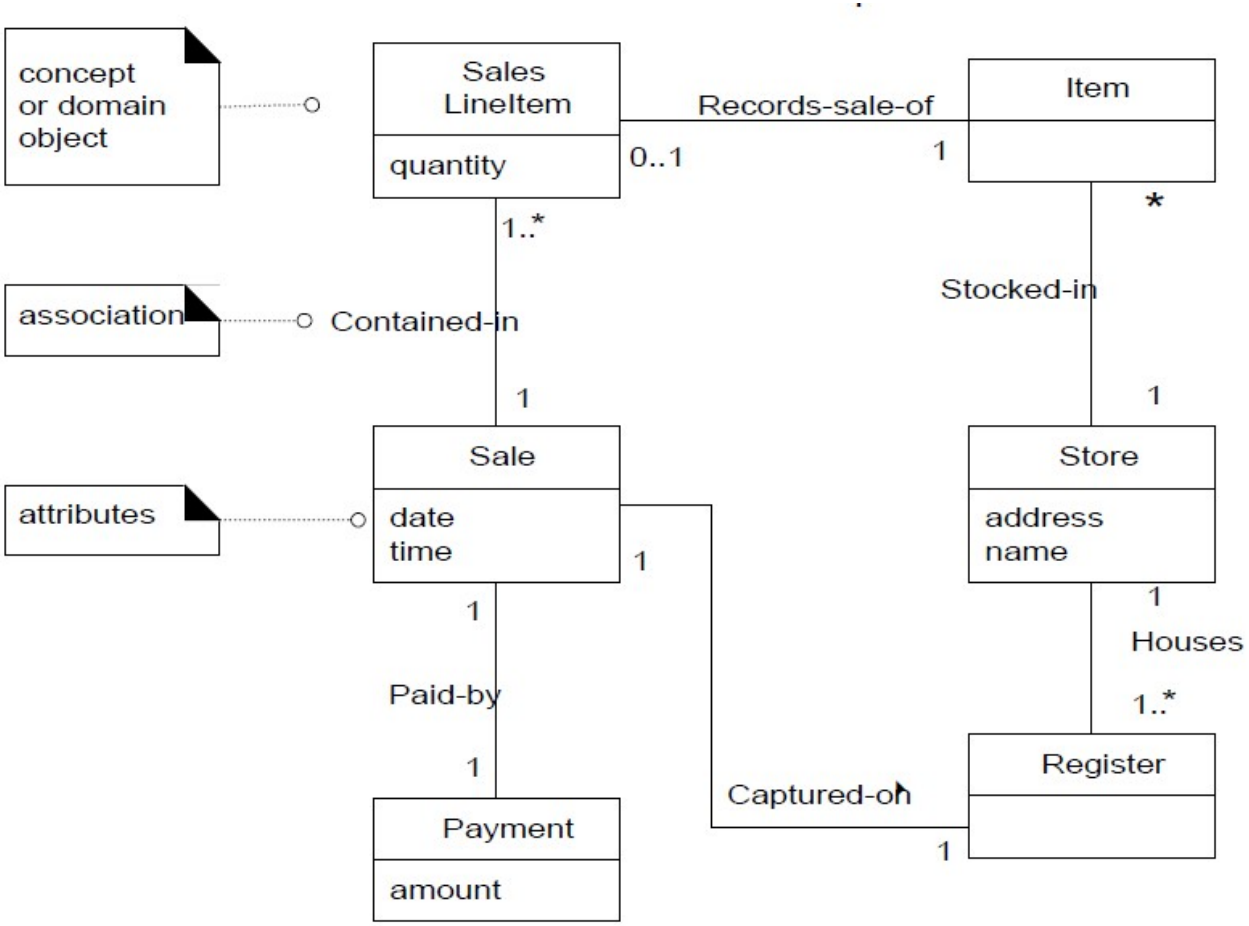
Key Point

- A domain model is a representation of real-world conceptual classes, not of software components. It is *not a set of diagrams describing software classes,* or software objects with responsibilities.

Domain Model

- The UP defines a Domain Model as one of the artifacts that may be created in the Business Modeling discipline.
- Using UML notation, a domain model is illustrated with a set of **class diagrams**
  - domain objects or conceptual classes
  - associations between conceptual classes
  - attributes of conceptual classes

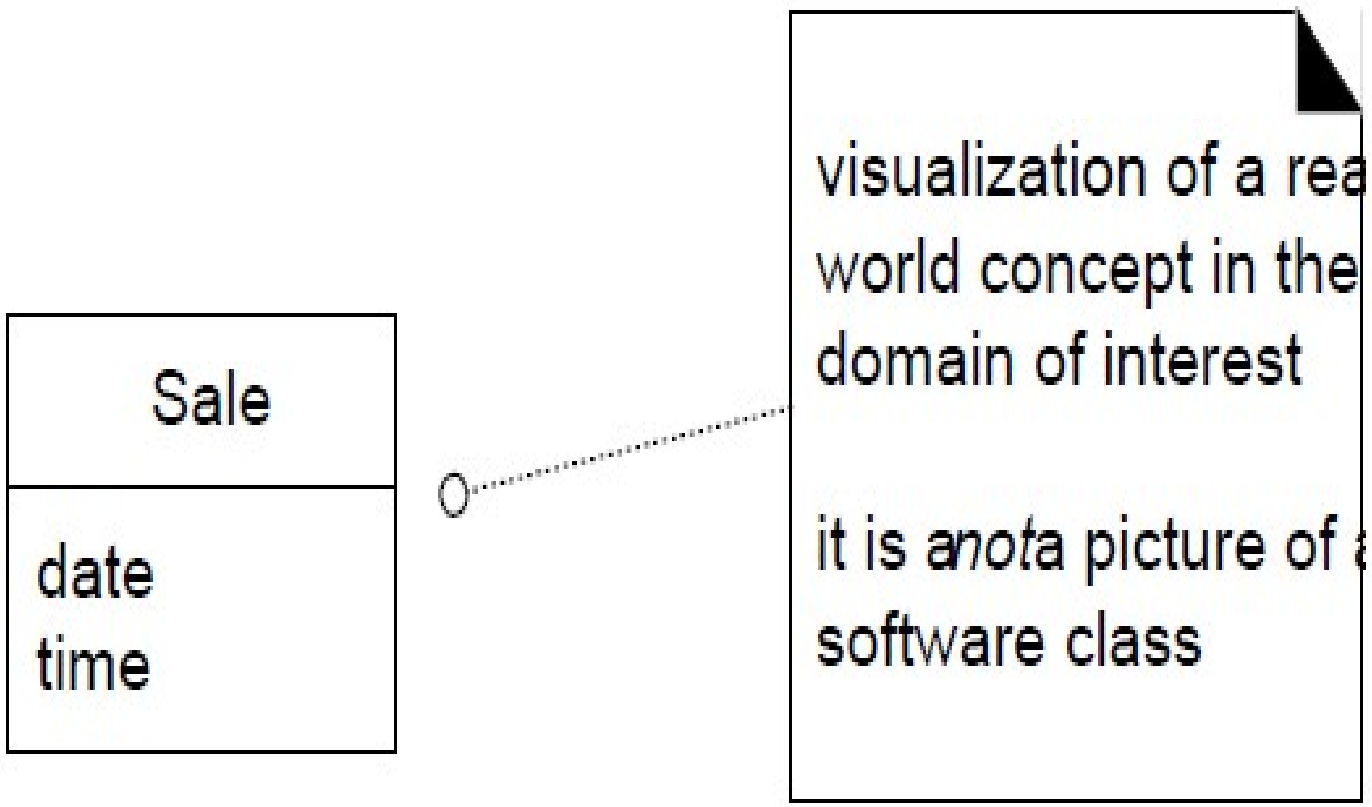
Partial Domain Model of POS problem



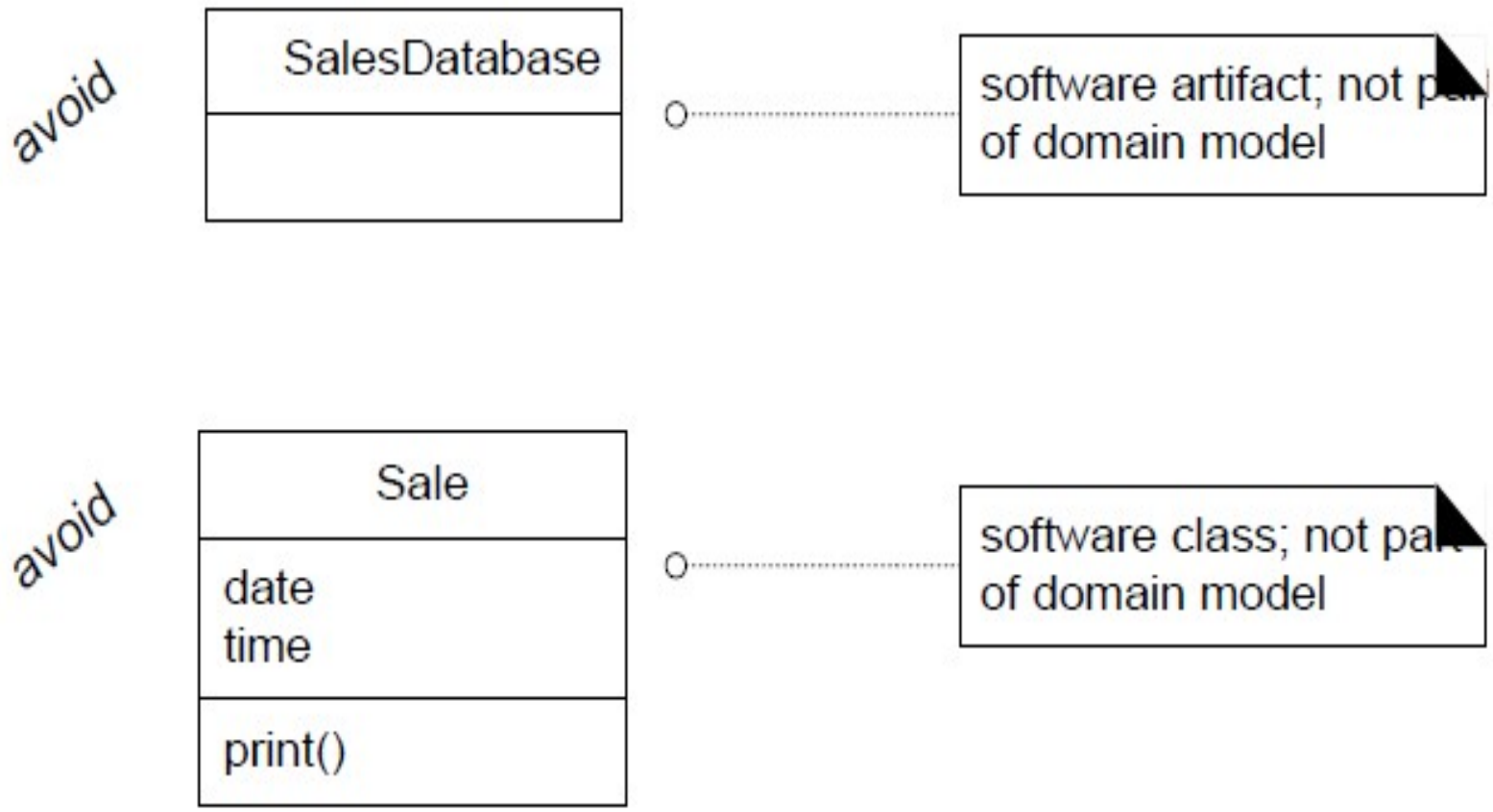
Domain Models Are not Models of Software Components

- A domain model is a visualization of things in the realworld domain of interest, *not of software components such as a Java or C++*
- Therefore, the following elements are not suitable in a domain model:
  - Software artifacts, such as a window or a database, unless the domain being modeled is of software concepts, such as a model of graphical user interfaces.
  - Responsibilities or methods.

Example

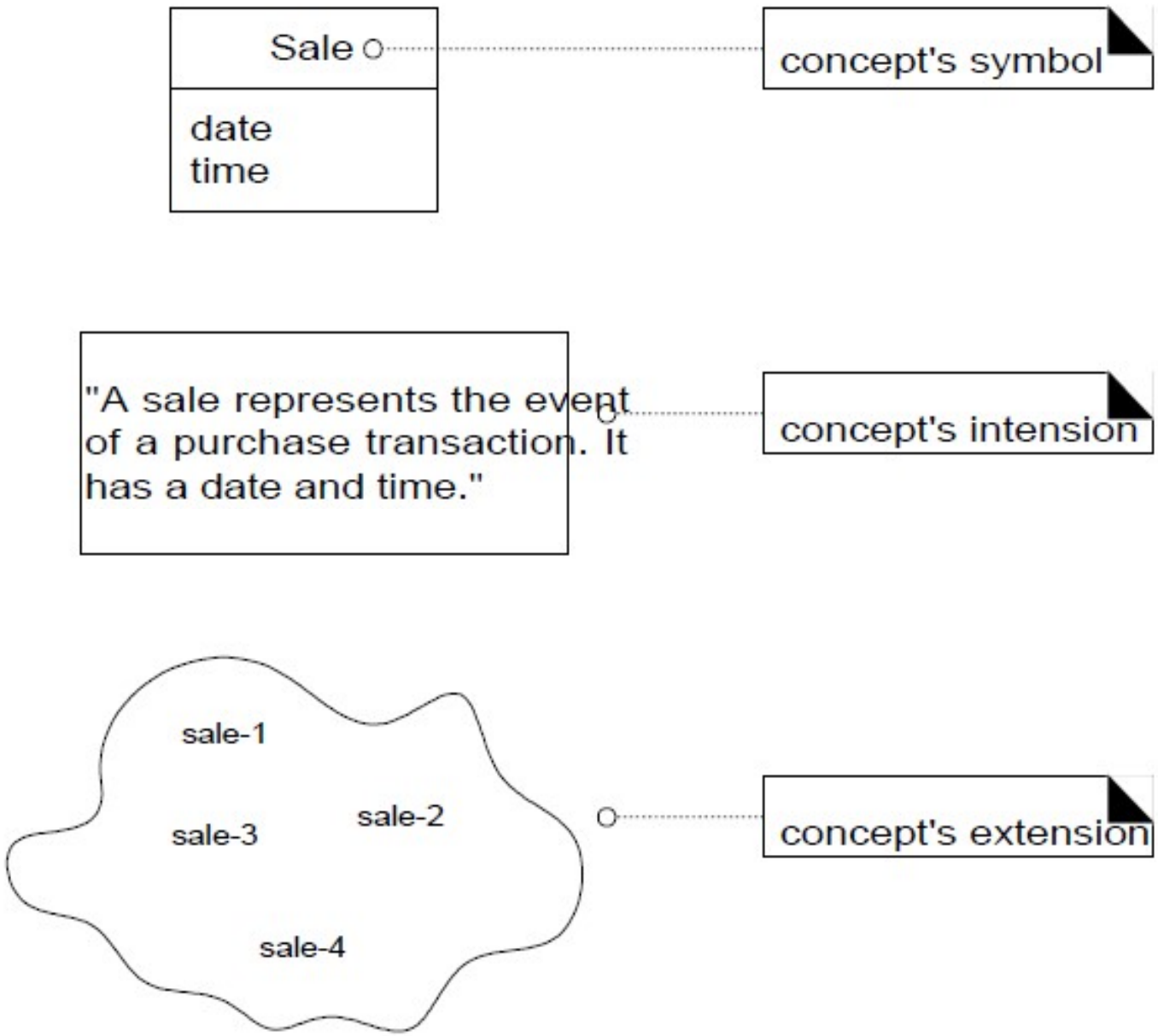

Sale
date
time
visualization of a rea
world concept in the
domain of interest
it is anota picture of
software class

Avoid


avoid
SalesDatabase
software artifact; not p
of domain model
avoid
Sale
date
time
print()
software class; not pa
of domain model

Conceptual Classes

- Informally, a conceptual class is an idea, thing, or object. More formally, a conceptual class may be considered in terms of its symbol, intension, and extension
  - **Symbol—words or images representing a conceptual class.**
  - **Intension—the definition of a conceptual class.**
  - **Extension—the set of examples to which the conceptual class applies**


Sale
date
time
concept's symbol
"A sale represents the event of a purchase transaction. It has a date and time."
concept's intension
sale-1
sale-2
sale-3
sale-4
concept's extension

OOAD Vs Structured Analysis

- A central distinction between object-oriented and structured analysis is: division by conceptual classes (objects) rather than division by functions

Conceptual Class Identification

- Two techniques are presented in the following sections:
- 1. Use a conceptual class category list.
- 2. Identify noun phrases.

Conceptual Class Category	Examples
physical or tangible objects	*Register* *Airplane*
specifications, designs, or descriptions of things places	*ProductSpecification* *FlightDescription* *Store* *Airport*
transactions	*Sale, Payment* *Reservation*
transaction line items	*SalesLineItem*
roles of people	*Cashier* *Pilot*
containers of other things	*Store, Bin* *Airplane*
things in a container	*Item* *Passenger*

Finding Conceptual Classes with Noun Phrase Identification

Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale.**
3. **Cashier** enters **item identifier.**
4. System records **sale line item** and presents **item description, price,** and running **total.** Price calculated from a set of price rules.

Cashier repeats steps 2-3 until indicates done.

5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment.**
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions)** and **Inventory** systems (to update inventory).
9. System presents **receipt.**
10. Customer leaves with receipt and goods (if any).

Candidate Conceptual Classes for the Sales Domain

Register

Item

Store

Sale

Payment

ProductCatalog

ProductSpecification

SalesLineItem

Cashier

Customer

Manager

How to Make a Domain Model

1. List the candidate conceptual classes using the Conceptual Class Category List and noun phrase identification techniques related to the current requirements under consideration.
2. Draw them in a domain model.
3. Add the associations necessary to record relationships for which there is a need to preserve some memory (discussed in a subsequent chapter).
4. Add the attributes necessary to fulfill the information requirements (discussed in a subsequent chapter).

On Naming and Modeling Things

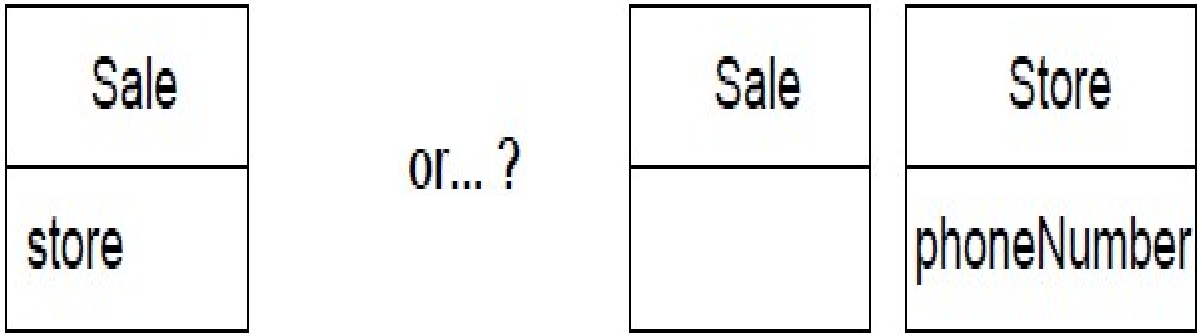
Make a domain model in the spirit of how a cartographer or mapmaker works:

- Use the existing names in the territory.
- Exclude irrelevant features.
- Do not add things that are not there.

A Common Mistake in Identifying Conceptual Classes

- If we do not think of some conceptual class X as a number or text in the real world, X is probably a conceptual class, not an attribute.

Example

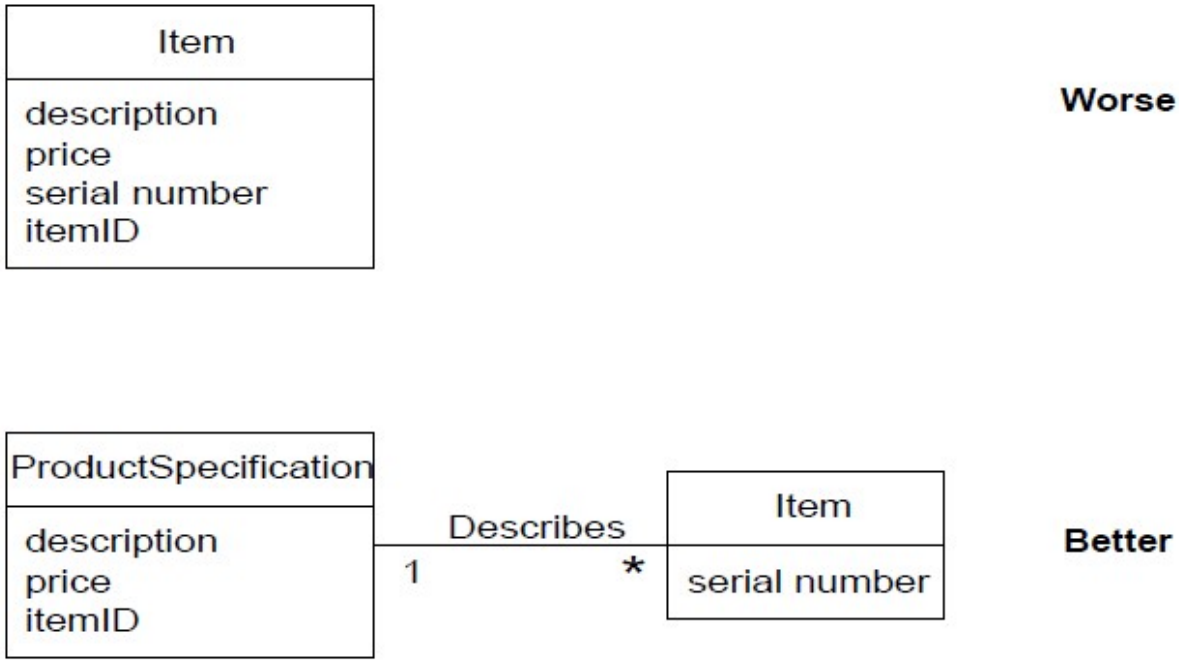


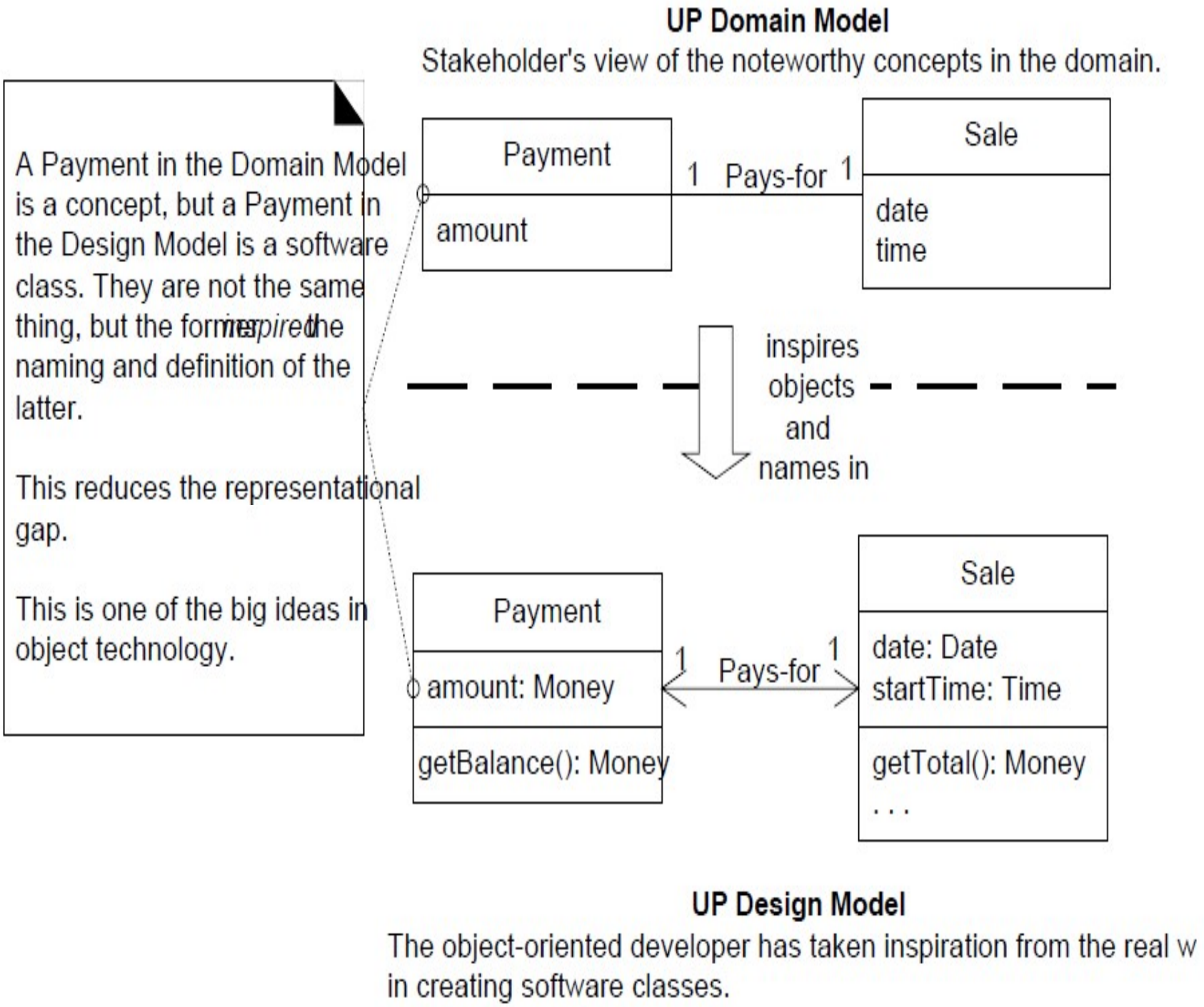
Modeling the *Unreal World*

- Some software systems are for domains that find very little analogy in natural or business domains; software for telecommunications is an example.

- For example, here are some candidate conceptual classes related to a telecommunication
- switch: *Message, Connection, Port, Dialog, Route, Protocol.*

The Need for Specification or Description Conceptual Classes




UP Domain Model
Stakeholder's view of the noteworthy concepts in the domain.
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.
Payment
amount
1
Pays-for
1
Sale
date
time
inspires objects and names in
Payment
amount: Money
getBalance(): Money
1
Pays-for
1
Sale
date: Date
startTime: Time
getTotal(): Money
. . .
UP Design Model
The object-oriented developer has taken inspiration from the real w
in creating software classes.

References

- Chapter 10 **Applying UML Patterns (Applying UML Patterns: An Introduction To Object-Oriented Analysis And Design) Craig L**