Interaction Modelling: Use Cases

Fabrizio Maria Maggi

Institute of Computer Science

(these slides are derived from the book "Object-oriented modeling and design with UML")

Interaction Modelling: INPUT

Interaction Modelling: INPUT

Domain Model

Interaction Modelling: INPUT

Domain Model

Interaction Modelling: INPUT

- To answer this question, the domain model provides classes with attributes and relations among them
- Operations are not specified

Interaction Modelling: Overview

Behaviour

Interactions

Interaction Modelling: Overview

How do objects interact?

Interaction Modelling: Overview

Interaction Modelling: OUTPUT

Operations
carry
come
drink
go
lead
ask
catch
cook
drive
hit
lift
bake
clap
cry
eat
hop
lock
bite
clean
cut
float
juggle
look
bounce
climb
dance
fly
jump
march
brush
close
dig
fold
kick
mix
build
color
draw
follow
knock
mop
call
comb
dream
give
laugh
open

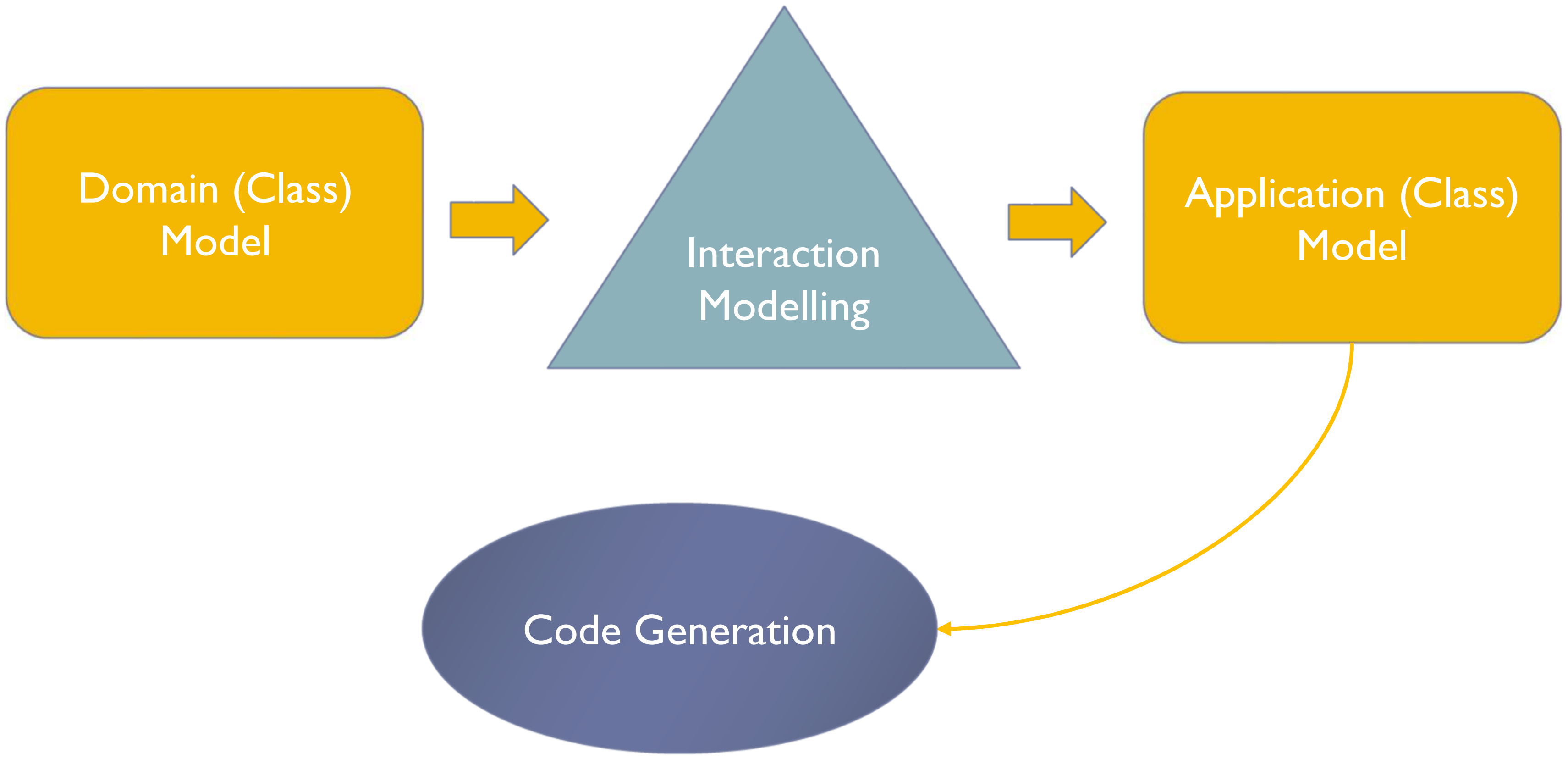
Interaction Modelling: OUTPUT

Application Model

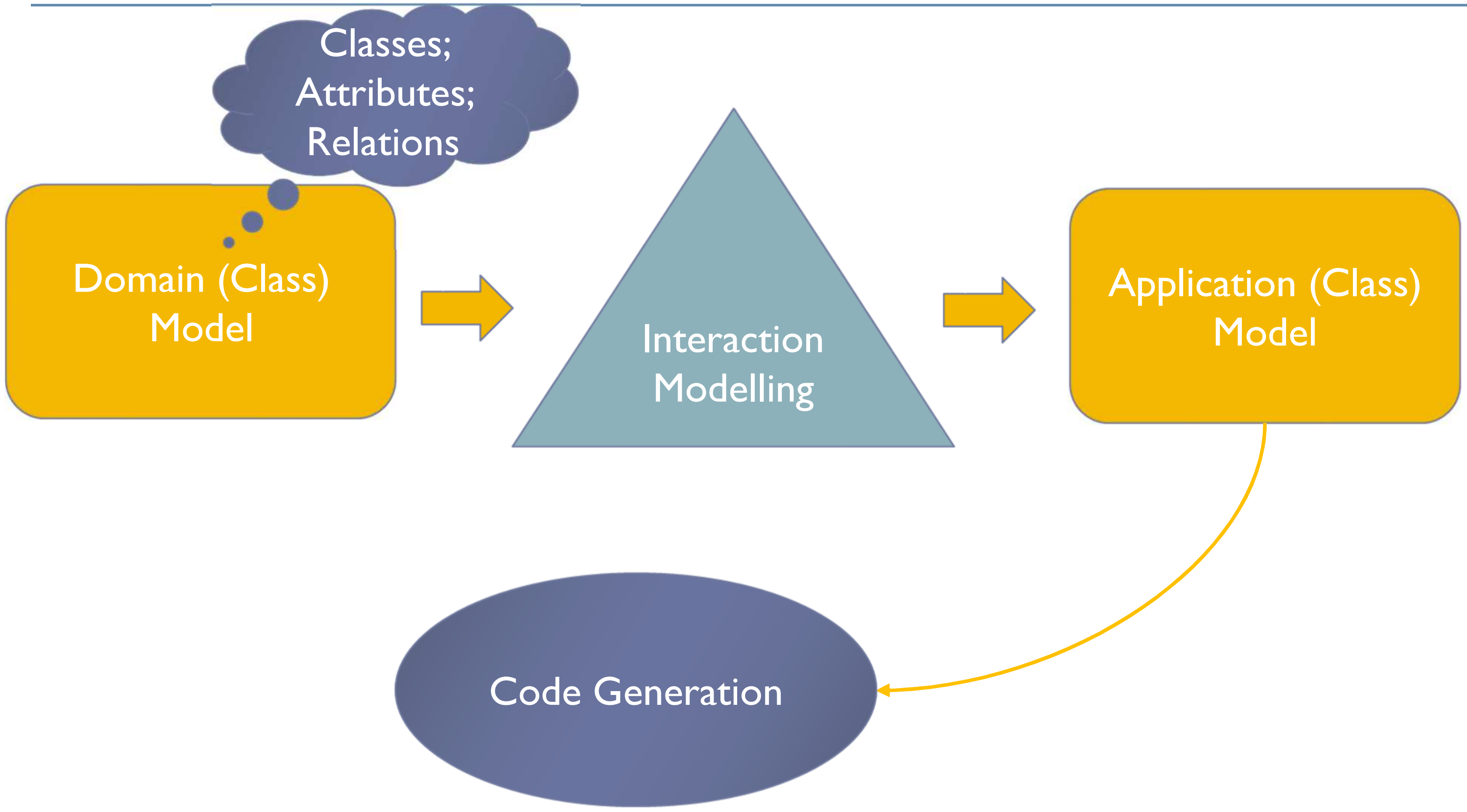
Operations
carry
come
drink
go
lead
ask
catch
cook
drive
hit
lift
bake
clap
cry
eat
hop
lock
bite
clean
cut
float
juggle
look
bounce
climb
dance
fly
jump
march
brush
close
dig
fold
kick
mix
build
color
draw
follow
knock
mop
call
comb
dream
give
laugh
open

Interaction Modelling: OUTPUT

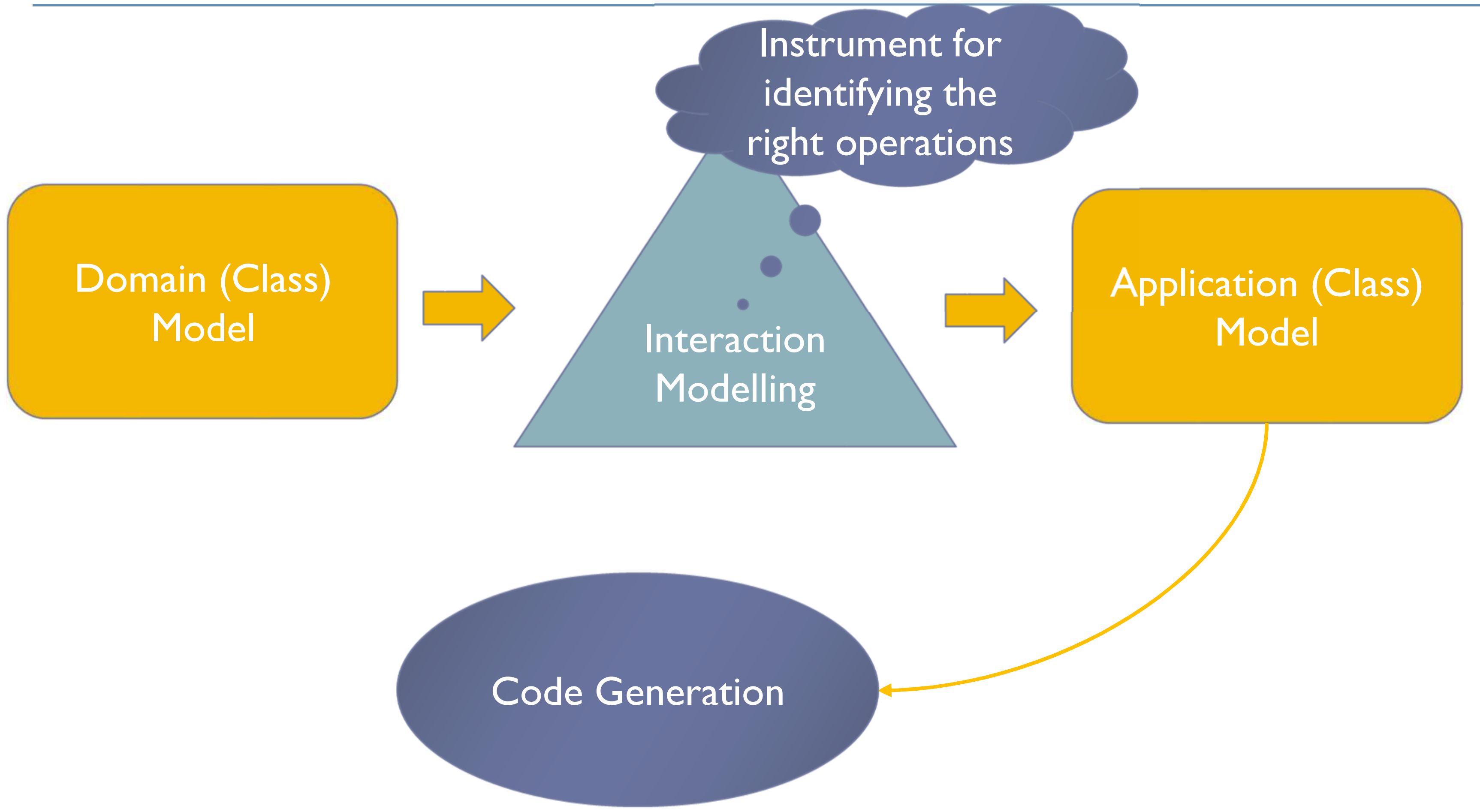
Interaction Modelling: Overview



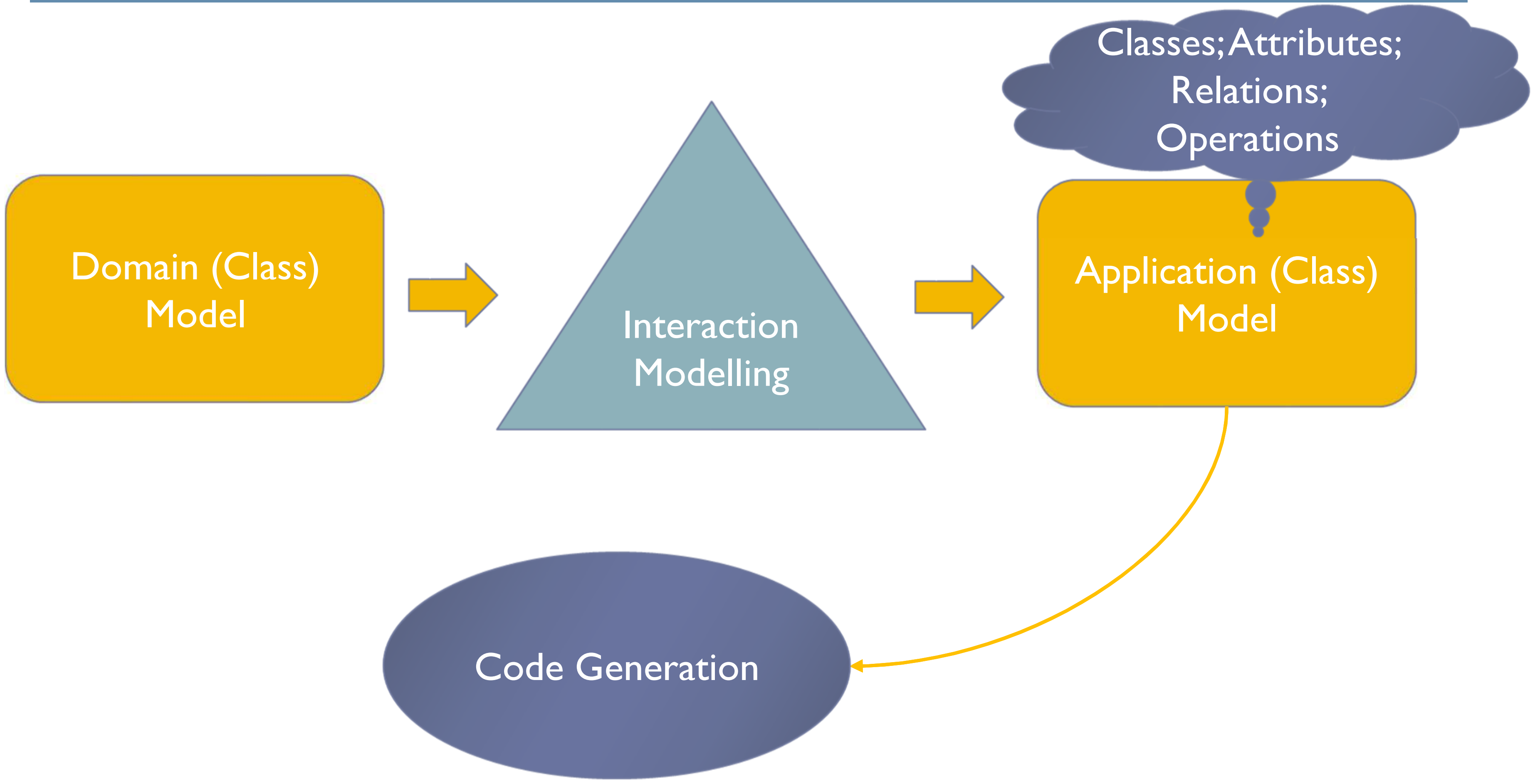
Interaction Modelling: Overview



Interaction Modelling: Overview


Instrument for identifying the right operations
Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation

Interaction Modelling: Overview


Classes; Attributes; Relations; Operations
Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation

Interaction Modelling

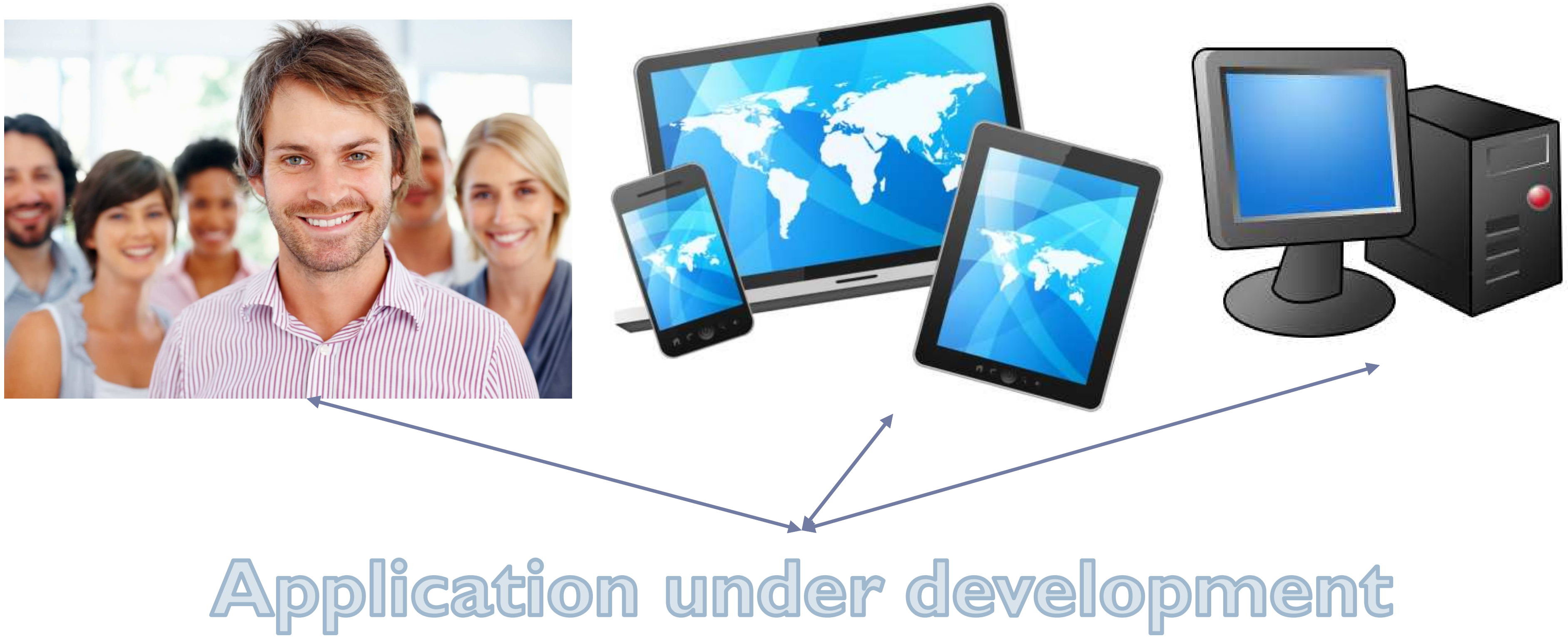
- Interactions can be modeled at different levels of abstraction
  - At a high level use cases describe how a system interacts with outside actors
    - Each use case represents a functionality that a system provides to the user
    - Use cases are helpful for capturing <u>informal requirements</u>
  - Sequence diagrams provide more detail about <u>which operations need to be invoked in a specific scenario</u>

Use Case Models: Actors

- An actor is a direct external user of a system
  - An object or a set of objects that communicates directly with the system but that is not part of the system
  - Examples
    - Customer and Repair Technician are actors of a vending machine
    - Traveler, Agent and Airline are actors of a travel agency system
    - User and Administrator are actors for a computer database system

Use Case Models: Actors

- Actors can be persons, devices and other systems (anything that interacts directly with the system)

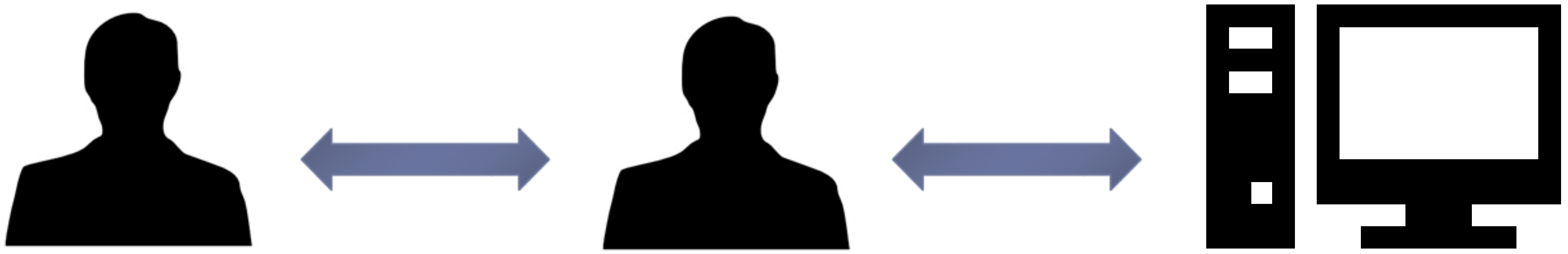


Use Case Models: Actors

- An actor represents a particular facet (i.e., role) of objects in its interaction with a system
- The same actor can represent different objects that interact similarly with a system
  - E.g., many individual persons may use a vending machine but their behavior toward the vending machine can be summarized by the actors Customer and Repair Technician
  - Each actor represents a coherent set of capabilities for its objects

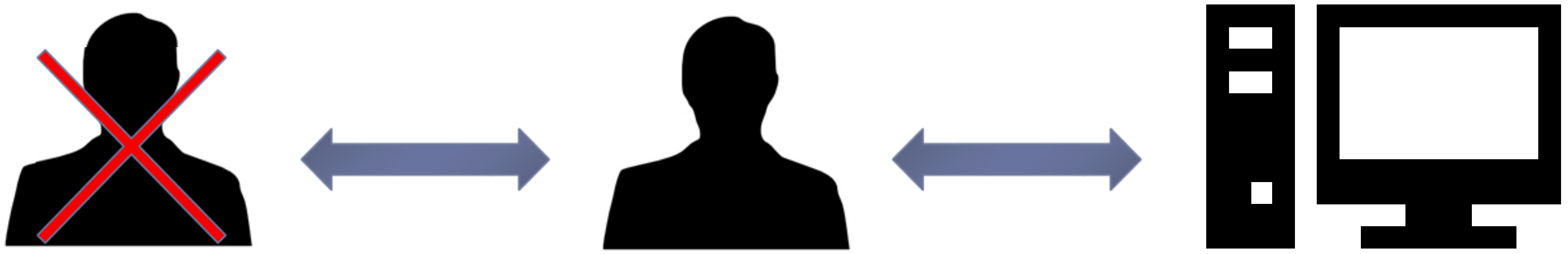
Use Case Models: Actors

- Modelling the actors helps to define a system by identifying the objects within the system and those on its boundary
- An actor is directly connected to the system
  - An indirectly connected object is not an actor and should not be included as part of the system model
    - Example: the Dispatcher of repair technicians from a service bureau is not an actor of a vending machine
      - Model a repair service that includes Dispatchers, Repair Technicians and Vending Machines as actors and use a different model for the vending machine model



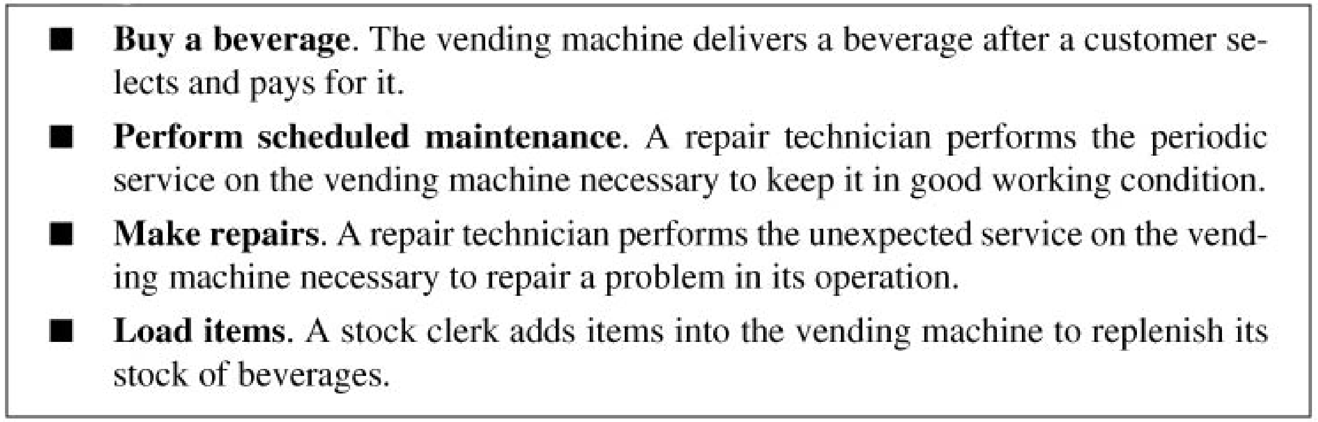
Use Case Models: Actors

- Modelling the actors helps to define a system by identifying the objects within the system and those on its boundary
- An actor is directly connected to the system
  - An indirectly connected object is not an actor and should not be included as part of the system model
    - Example: the Dispatcher of repair technicians from a service bureau is not an actor of a vending machine
      - Model a repair service that includes Dispatchers, Repair Technicians and Vending Machines as actors and use a different model for the vending machine model



Use Case Models: Use Cases

- A use case is a coherent <u>piece of functionality</u> that a system can provide by interacting with actors



- **Buy a beverage**. The vending machine delivers a beverage after a customer selects and pays for it.
- **Perform scheduled maintenance**. A repair technician performs the periodic service on the vending machine necessary to keep it in good working condition.
- **Make repairs**. A repair technician performs the unexpected service on the vending machine necessary to repair a problem in its operation.
- **Load items**. A stock clerk adds items into the vending machine to replenish its stock of beverages.

Figure 7.1 Use case summaries for a vending machine. A use case is a coherent piece of functionality that a system can provide by interacting with actors.

Use Case Models: Use Cases

Operations
carry
come
drink
go
lead
ask
catch
cook
drive
hit
lift
bake
clap
cry
eat
hop
lock
bite
clean
cut
float
juggle
look
bounce
climb
dance
fly
jump
march
brush
close
dig
fold
kick
mix
build
color
draw
follow
knock
mop
call
comb
dream
give
laugh
open

Use Case Models: Use Cases

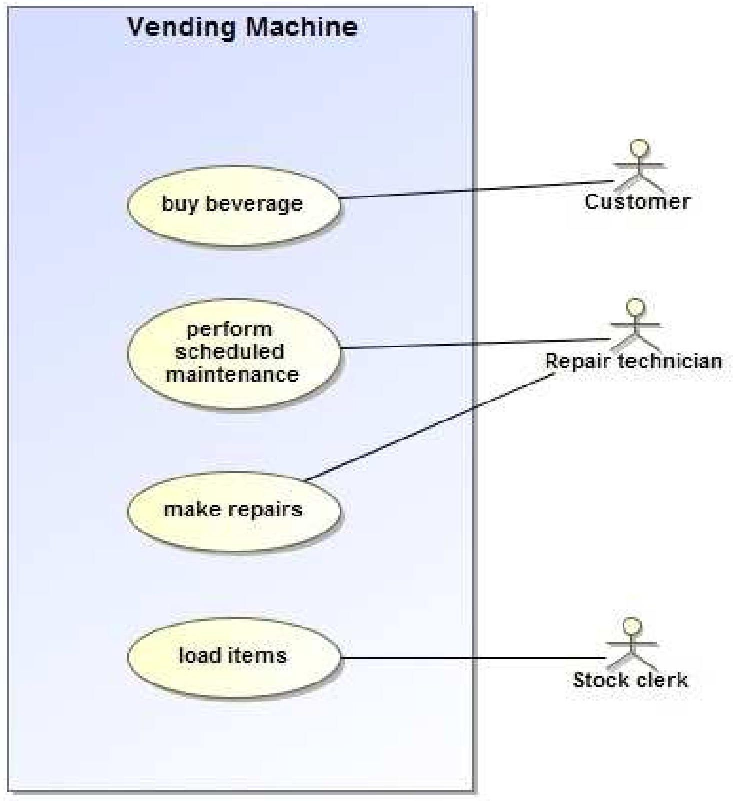
- Each use case involves one or more actors as well as the system itself
  - Examples: the use case "Buy a beverage" involves the Customer; the use case "Perform scheduled maintenance" involves the Repair Technician; in a telephone system the use case "Make a call" involves two actors, a Caller and a Receiver
- An actor is not necessarily a person
  - Example: in an online shop the use case "Checkout" involves the Web Customer and the Credit Payment Service
- A use case partitions the functionality of the system into a mainline behavior sequence, variations on normal behavior, exception conditions, error conditions, cancellations of a request
- Use cases should all be at a comparable level of abstraction
  - Examples: "Make telephone call" and "Record voice mail message" are at a comparable level; "Set external speaker volume to high" is too narrow, "Set speaker volume" or even "Set telephone parameters" would be better

Creating Use Case Models

- Use case models include
  - Use case diagrams
  - (Textual) use case descriptions
- Useful guidelines: Section 7.1.4 Guidelines for Use Case Models pages 135-136 of the book *"Object-oriented modeling and design with UML"*

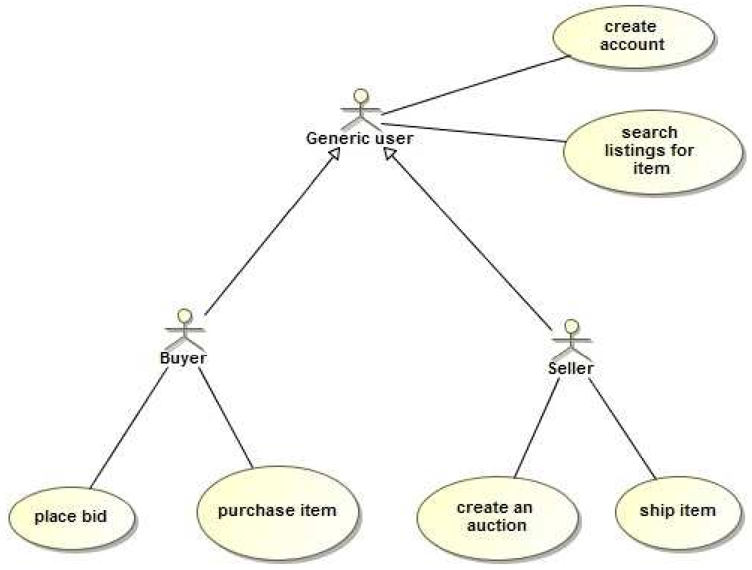
Use Case Diagrams

- UML has a graphical notation for summarizing use cases into use case diagrams
  - A rectangle contains the use cases for a system with the actors listed on the outside
  - The name of the system is written near a side of the rectangle
  - A name within an ellipse denotes a use case
  - A "stick man" icon denotes an actor with the name placed below the icon
  - Solid lines connect use cases to participating actors



Actor Generalization

- The child actor inherits all use case associations from the parent
- Actor generalization should be used if the specific actor has more responsibility than the generalized one (i.e., associated with more use cases)
  - Example: Look at the requirements management use case diagram in the picture and you will see there is duplicate behavior in both the buyer and seller which includes "Create an account" and "Search listings"
  - Rather than having all of this duplication, we will have a more general user that has this behavior and then the actors will "inherit" this behavior from the general user



Use Case Relationships

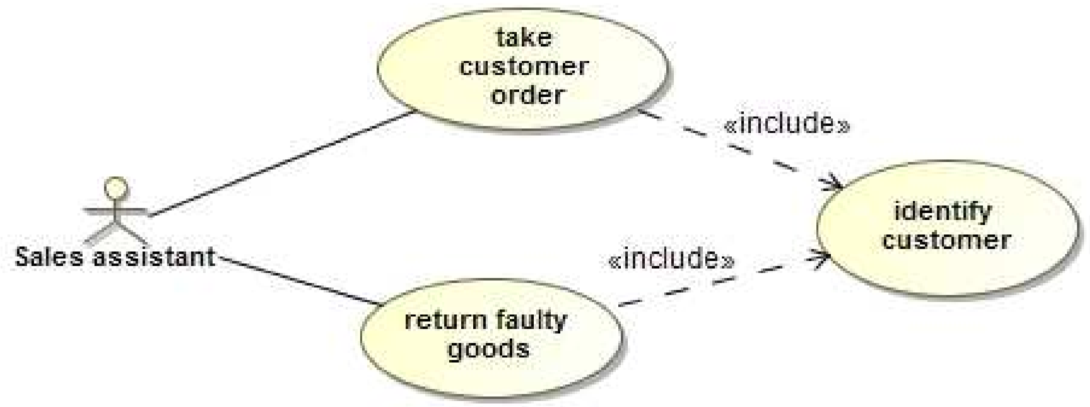
- For large applications complex use cases can be built from smaller pieces
- Linking enables flexibility in requirements specification
  - Isolating functionality
  - Enabling functionality sharing
  - Breaking functionality into manageable chunks
- Three mechanism are used:
  - Include
  - Extend
  - Generalization
- Useful guidelines: Section 8.1.5 Guidelines for Use Case Relationships pages 150-151 of the book *"Object-oriented modeling and design with UML"*

Use Case Relationships: Include

- Include Relationship
  - A use case can make use of other smaller use cases
  - The include relationship incorporates the behaviour of another use case (e.g., subroutines)
- Factoring a use case into pieces is appropriate when the pieces represent significant behaviour units

Use Case Relationships: Include

- The UML notation for an include relationship is a dashed arrow from the source (including) to the target (included) use case. The keyword <<include>> annotates the arrow

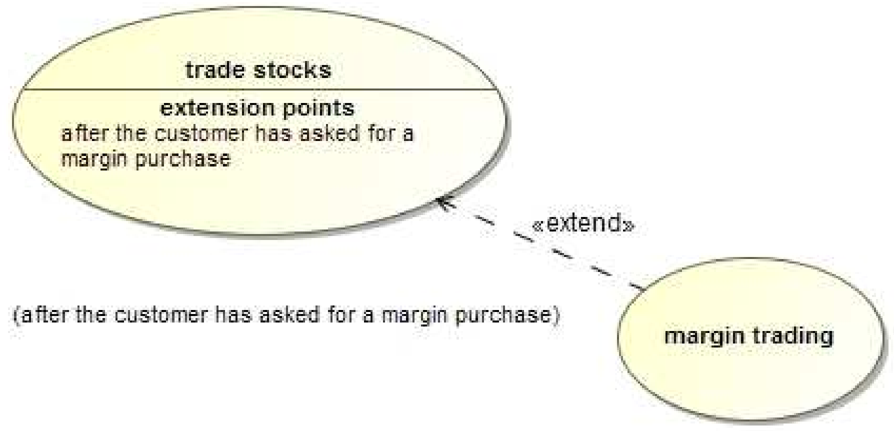


Use Case Relationships: Extend

- Extend Relationship
  - Adds an "extra behaviour" to a base use case
  - Is used in the situation in which some initial capability is defined and later features are added modularly
- Base use case is meaningful on its own, it is independent of the extension. Extension typically defines optional behavior that is not necessarily meaningful by itself

Use Case Relationships: Extend

- The UML notation for an extend relationship is a dashed arrow from the extension to the base use case. The keyword <<extend>> annotates the arrow
  - Use case "trade stocks" is meaningful on its own. It could be optionally extended with "margin trading"
  - **Extension Points**: specify the location at which the behavior of the base use case may be extended. Extension points can have a condition attached. The extension behaviour occurs only if the condition is true when the control reaches the extension point

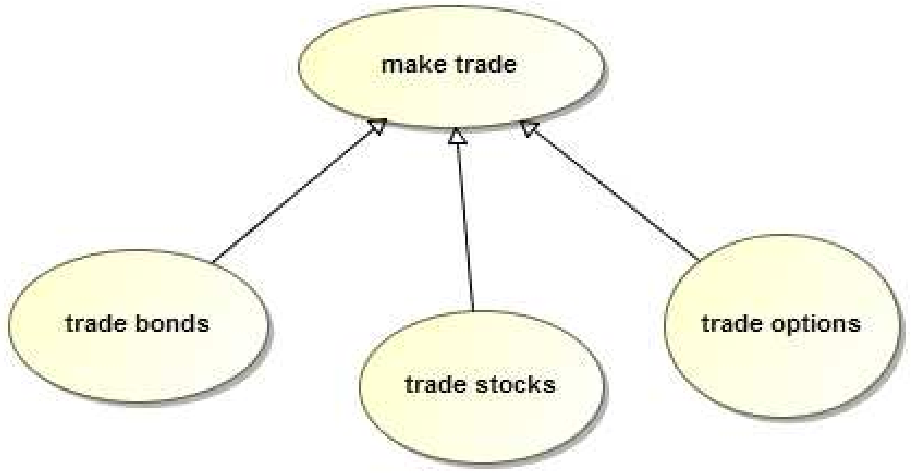

trade stocks
extension points
after the customer has asked for a margin purchase
«extend»
(after the customer has asked for a margin purchase)
margin trading

Use Case Relationships: Generalization

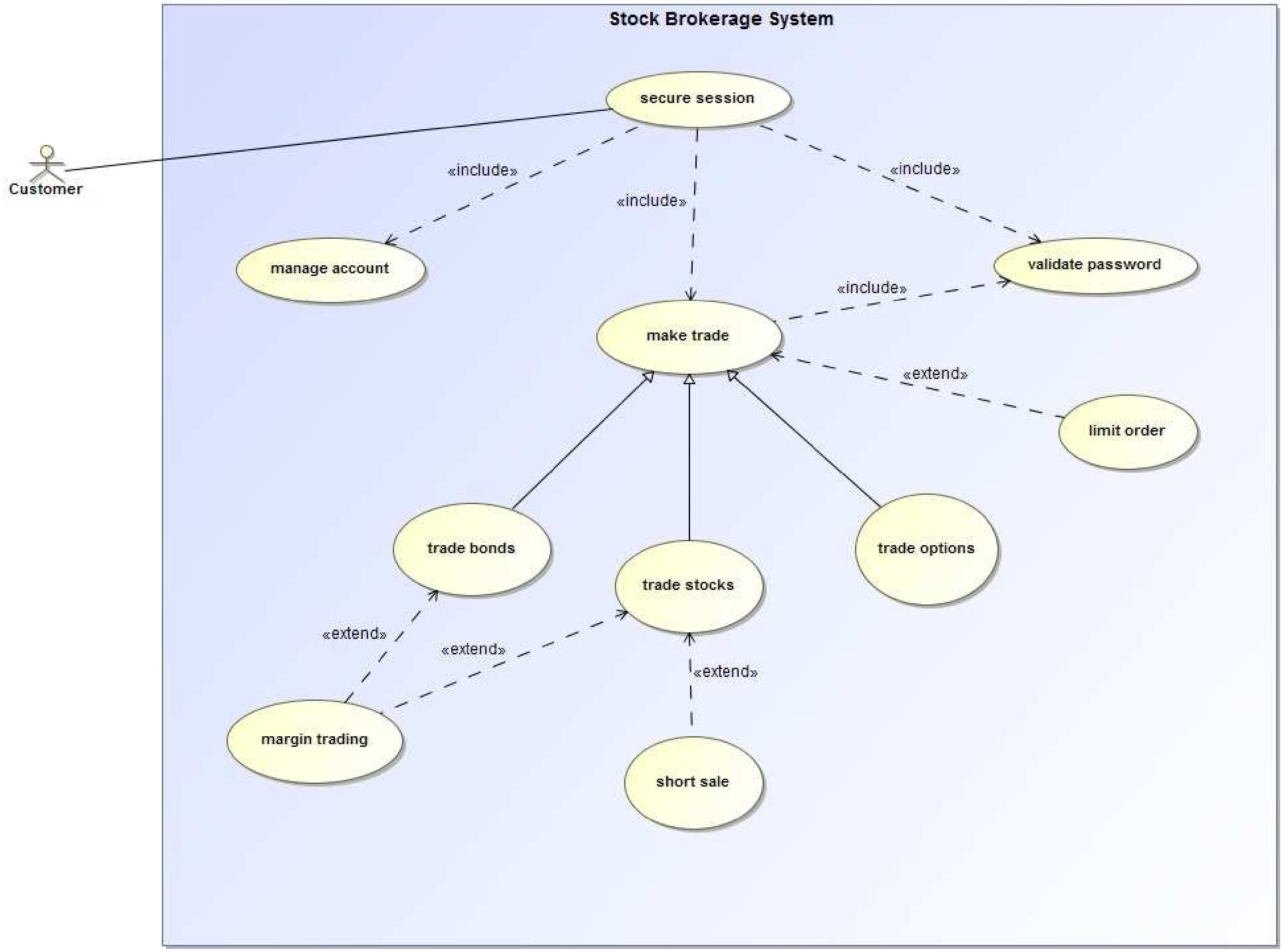
- Generalization Relationship
  - Can show specific variations of a general use case, analogous to generalization among classes
  - A parent use case represents a general behaviour
  - A child use case specializes the parent by inserting additional steps or by refining existing steps
- The child use case inherits the behavior of the parent use case
  - The interaction (described in the textual description)
  - Use case links (associations, include, extend, generalization)
- The child use case can substitute the parent use case
  - Overriding occurs through the textual description

Use Case Relationships: Generalization

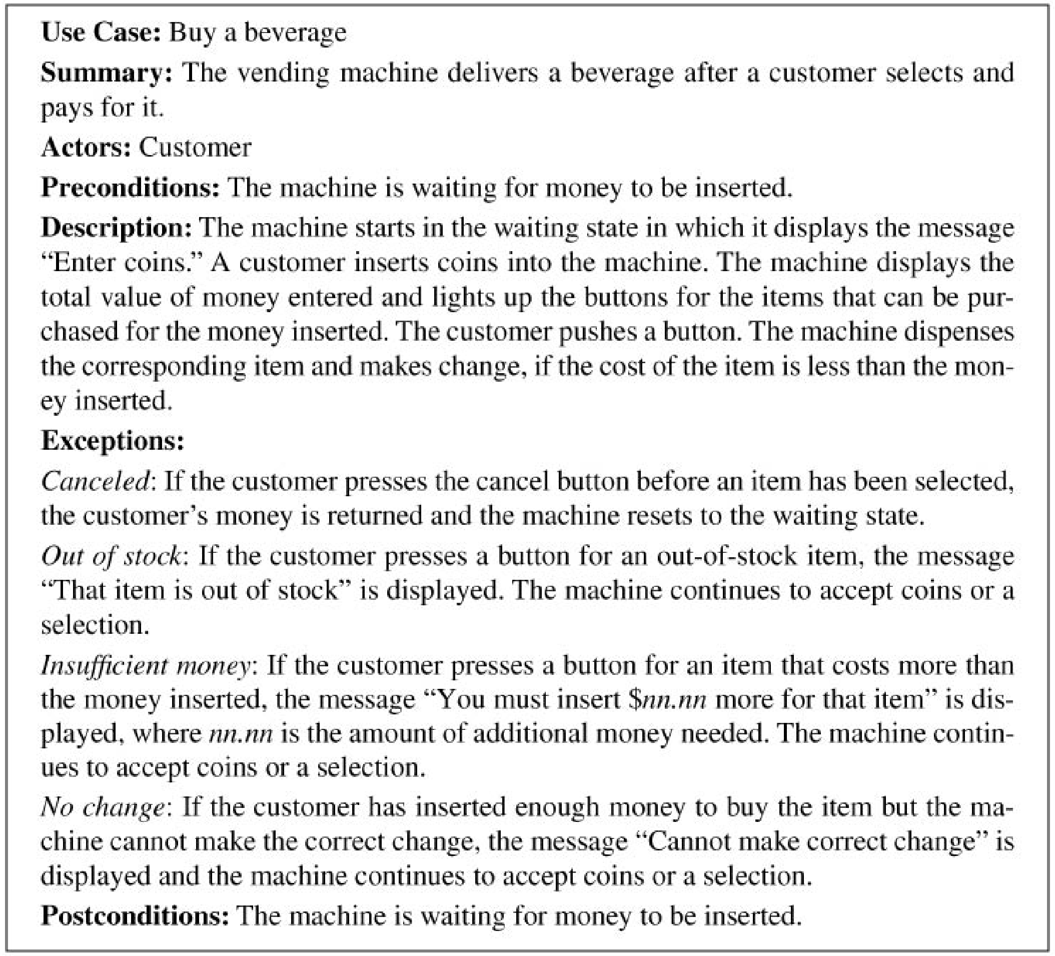
- The UML notation for an generalization relationship is an arrow with its tail on the child use case and a triangular arrowhead on the parent use case (the same notation that is used for classes)
  - Use case "make trade" can be specialized into child use cases "trade bonds", "trade stocks" and "trade options" (based on different types of financial items)
  - The parent use case contains steps that are performed for any kind of trade
  - Each child contains additional steps that are particular to a specific kind of trade



Combinations of Use Case Relationships



Use Case Descriptions



Use Case: Buy a beverage

Summary: The vending machine delivers a beverage after a customer selects and pays for it.

Actors: Customer

Preconditions: The machine is waiting for money to be inserted.

Description: The machine starts in the waiting state in which it displays the message "Enter coins." A customer inserts coins into the machine. The machine displays the total value of money entered and lights up the buttons for the items that can be purchased for the money inserted. The customer pushes a button. The machine dispenses the corresponding item and makes change, if the cost of the item is less than the money inserted.

Exceptions:

Canceled: If the customer presses the cancel button before an item has been selected, the customer's money is returned and the machine resets to the waiting state.

Out of stock: If the customer presses a button for an out-of-stock item, the message "That item is out of stock" is displayed. The machine continues to accept coins or a selection.

Insufficient money: If the customer presses a button for an item that costs more than the money inserted, the message "You must insert $*nn.nn* more for that item" is displayed, where *nn.nn* is the amount of additional money needed. The machine continues to accept coins or a selection.

No change: If the customer has inserted enough money to buy the item but the machine cannot make the correct change, the message "Cannot make correct change" is displayed and the machine continues to accept coins or a selection.

Postconditions: The machine is waiting for money to be inserted.

Figure 7.2 Use case description. A use case brings together all of the behavior relevant to a slice of system functionality.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.