

# Week 4: OOA

Use Case, Robustness, and  
Sequence Diagrams

# Problem Description

The Assistant Trader (AT) uses an order entity window to enter the data for an order. The system ensures that the trade number entered is unique. Then the system determines what type of investment is going to be traded and brings up the appropriate trade entity window in response. The AT then uses the window to enter the original data for the associated trade. When the AT finishes making entries, the AT indicates that the trade is ready for processing. The system validates certain entries (i.e. it makes sure the settlement date does not precede the trade date) and then processes the trade appropriately.

# Vague Requirements

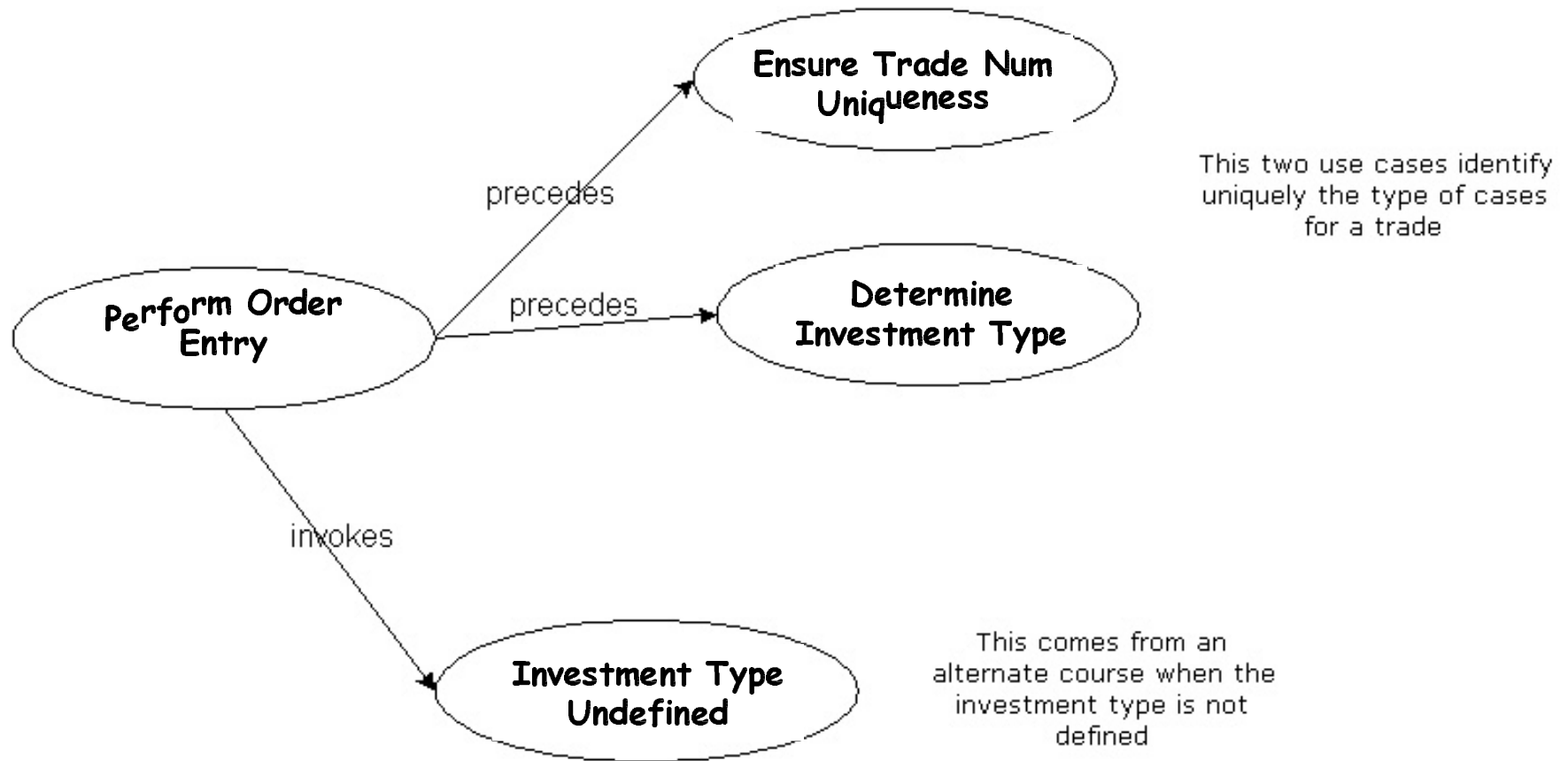
## Questions:

- What does "processing the trade" mean?
  - This possibly leads to a sub use case ...
- What happens if the A.T. does something wrong?
  - The answer is to define basic and alternate courses of action ...

# Use case stereotypes

- We will use two new concepts in place of the known (but rather unclear) **uses** and **extends** stereotypes:
  - **Invokes**: one use case invokes another in the same way a function invokes a peer function
  - **Precedes**: indicates that one use case needs to precede another within a logical sequence
- Word of caution: You may be better off using the standard **uses** and **extends** stereotypes for A1!
- Is it a good idea after all to induce an order on use cases (**precedes** gives us the power to do so) ??

# Example Use Case Diagram



Q: Who is the actor?

# Mistakes to Avoid When Writing Use Cases

- Writing functional requirements instead of usage scenario text.
- Describing attributes or methods rather than usage.
- Writing the use cases too tersely.
- Using a perspective other than the user's.
- Describing only user interactions, and ignoring system responses.
- Omitting text for alternate courses of action.
- Focusing on something other than what's "inside" a use case, like pre-conditions or post-conditions.
- Spending one month to decide whether to use includes or extends.

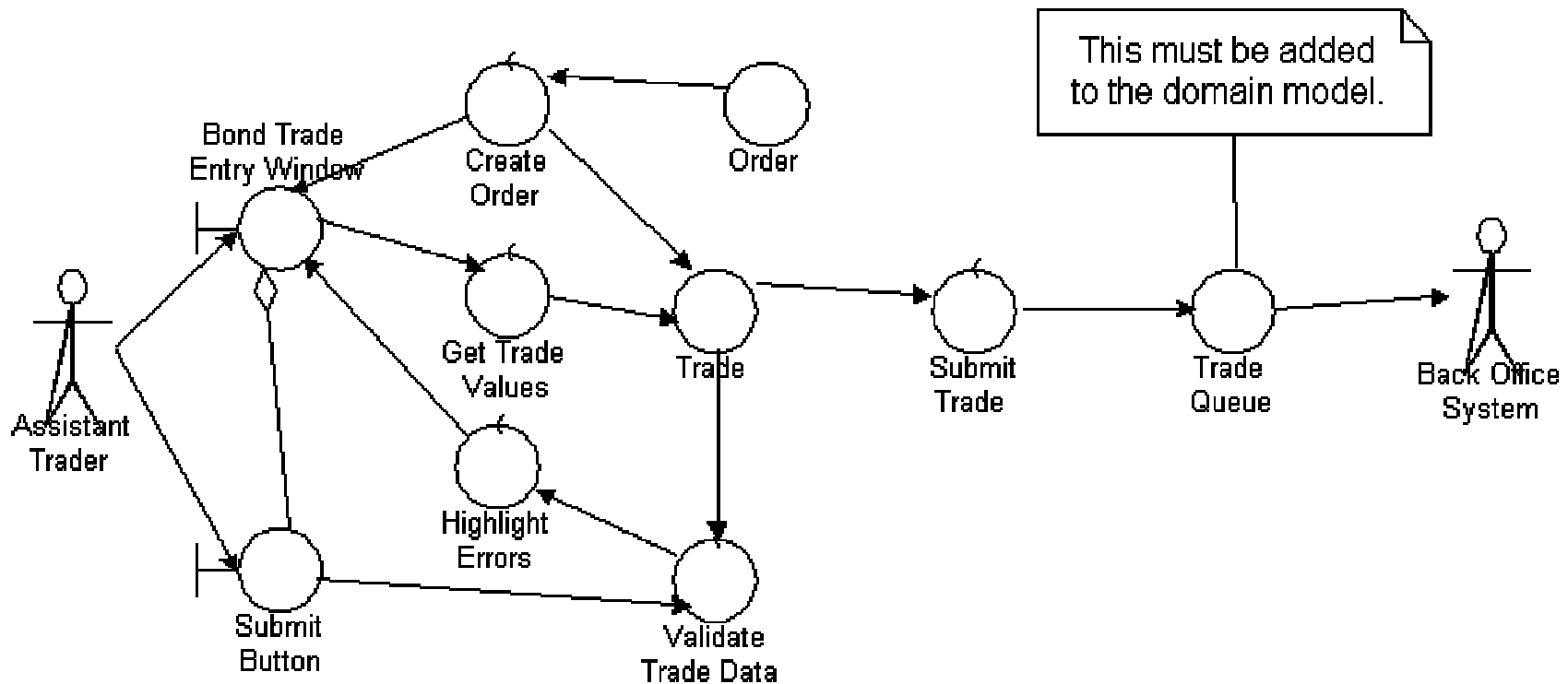
# Robustness Diagrams: Rules of Thumb

- When doing robustness analysis there are two general principles to guide your choices:
  - **Small number (between two and five) of controllers per average use case.** If you have more than 10 controllers per use case you'd better consider splitting it into more use cases.
  - **Arrows should NOT represent software messages.** They simply indicate logical associations.

# Initial Use Case Description (Before Robustness Analysis)

- *Basic Course:* The Assistant Trader (AT) uses a bond trade entry window to enter the primary values for the trade. The system validates both general trade values and bond-specific values (for instance, it makes sure the coupon rate is "reasonable") before processing the trade.
- *Alternate Course:* If the validation fails, notify the user, highlight the erroneous values, and get new values from the user.

# Example Robustness Diagram



# Updating the Use Case Desc.

- **Basic Course:** The system creates a new trade with the ticket number attached to the order. It also brings up a Bond Trade Entry window. The AT uses this window to enter the appropriate values for the trade. When the AT chooses to submit the trade, the system validates both general trade values and bond-specific values (for instance, it makes sure the coupon rate is "reasonable") before processing the trade. If the trade passes all validation tests, the system submits it to the Trade Queue, from which the trade is later sent to the Back Office System to be cleared and processed further.
- **Note:** Robustness analysis also affects the domain model

# Major Benefits of Robustness Analysis

- It forces you to write your use cases using a consistent style.
- It provides sanity and completeness checks for the use cases.
- It allows you to apply syntax rules (i.e. "actors talk only to boundary objects") to your use cases.
- Robustness diagrams are quicker to draw and easier to read than sequence diagrams.
- It provides traceability between what the system does (use cases) and how the system works (sequence diagrams).
- It plugs the semantic gap between analysis and design.

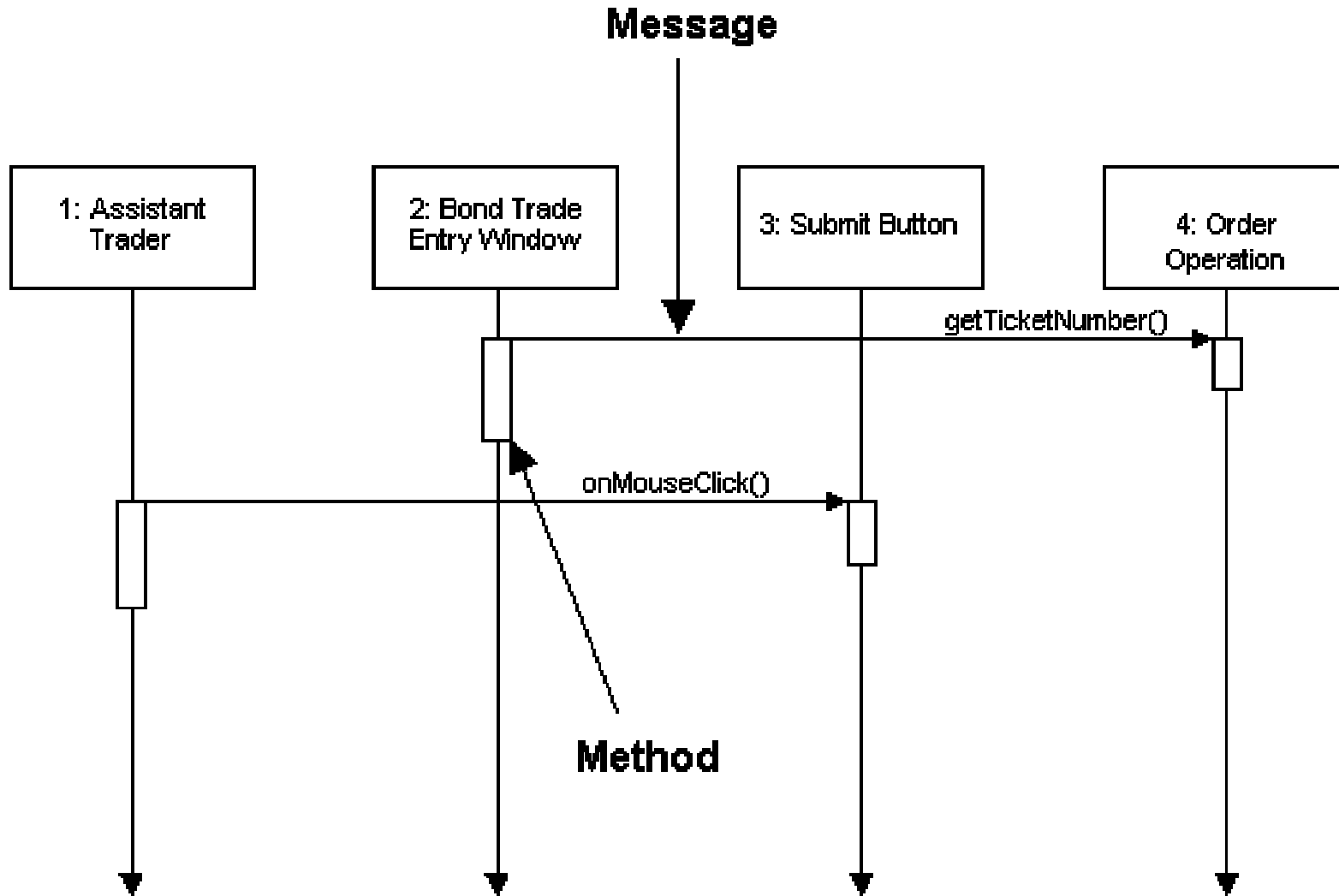
# Sequence Diagrams

- **Goals of interaction modeling:**
  - Allocate behavior among boundary, entity and control objects
  - Show the detailed interactions (over time) among the objects associated with each of your use cases
  - Finalize the distribution of operations among classes

# Starting a Sequence Diagram

- **Copy the textual description of the use case** from the specification and paste it into the margin of the page.
- **Add the entity objects** from the robustness diagrams.
- **Add the boundary objects** from the robustness diagrams. Note that boundary objects are part of the *solution space*, while classes in the domain model are in the *problem space*.

# Example Sequence Diagram (not complete)



# Important Things to Remember About Sequence Diagrams

- The diagram should match the narrative flow of the associated use case.
- Whenever possible, do a single sequence diagram for the entire use case, including all alternative courses of action.
- By exploring the dynamic behavior of the system you identify attributes and methods of your class diagrams.
- The sequence diagram is the primary means for making behavior allocation decisions.
- You are adding solution space objects to the problem domain objects as you explore system usage at a detailed level.
- Design patterns are often helpful here.