Sequence Diagram Tutorial

From:

UML Distilled, Third Edition, Chapter 4
M. Fowler

Use Cases and Scenarios

- A **use case** is a collection of interactions between external actors and a system
- In UML, a use case is “the specification of a sequence of actions, including variants, that a system (or entity) can perform, interacting with actors of the system.”
- Typically each **use case** includes a primary **scenario** (or main course of events) and zero or more secondary **scenarios** that are alternative courses of events to the primary **scenario**.
- In RUP (Rational Unified Process), user requirements are captured as **use cases** that are refined into **scenarios**.
- **Then**: A **scenario** is one path or flow through a **use case** that describes a sequence of events that occurs during one particular execution of a system.

UML Sequence Diagrams

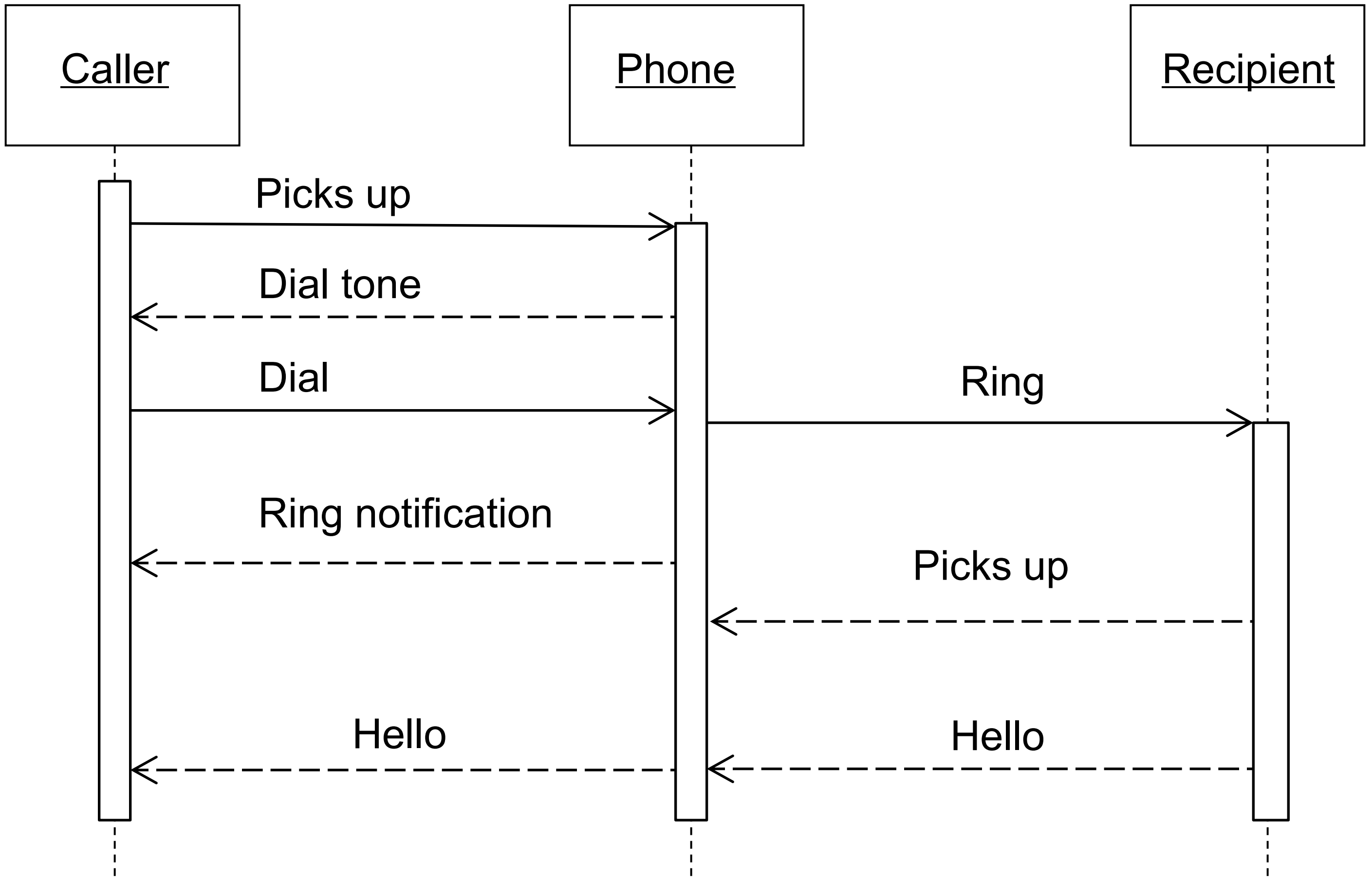
- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Emphasis on time ordering!

Sequence Diagram Key Parts

- **participant**: object or entity that acts in the diagram
  - diagram starts with an unattached "found message" arrow
- **message**: communication between participant objects
- the **axes** in a sequence diagram:
  - **horizontal**: which object/participant is acting
  - **vertical**: time (down -> forward in time)

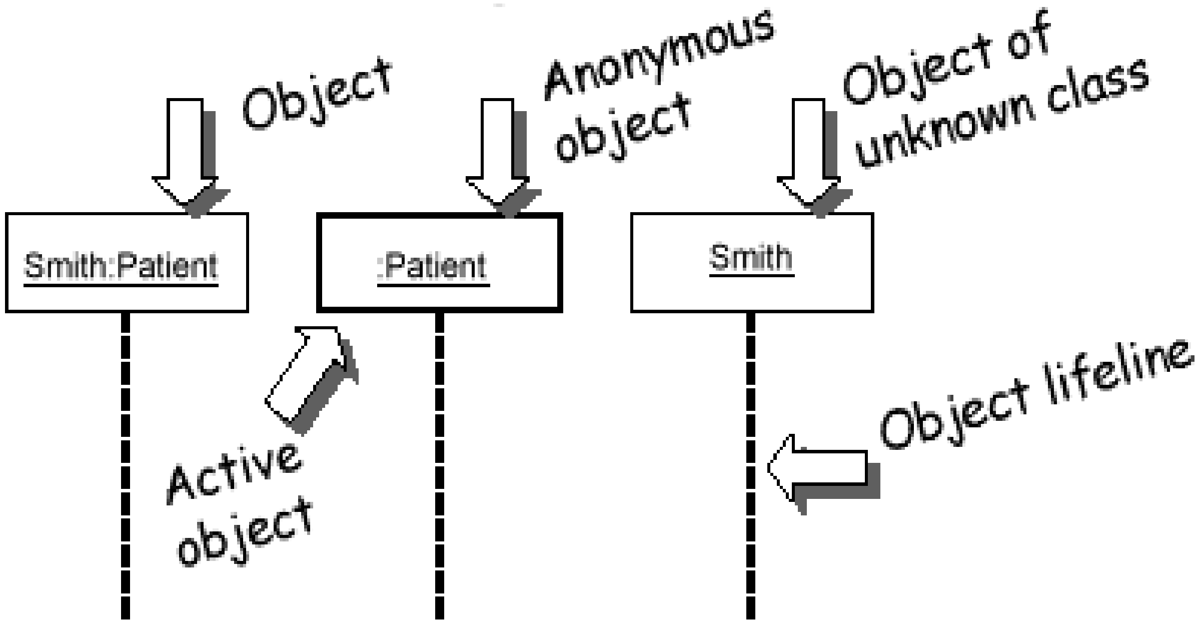
Sequence Diagram (make a phone call)



Representing Objects

Squares with object type, optionally preceded by "*name* :"

- write object's name if it clarifies the diagram
- object's "life line" represented by dashed vert. line

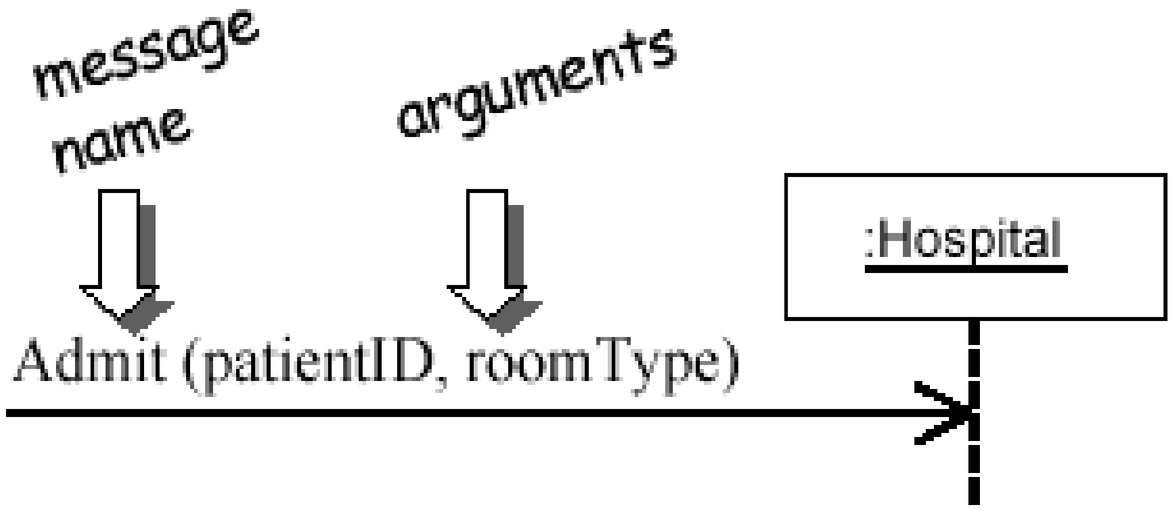


Name syntax: <u><objectname>:<classname></u>

Messages Between Objects

messages (method calls) indicated by arrow to other object

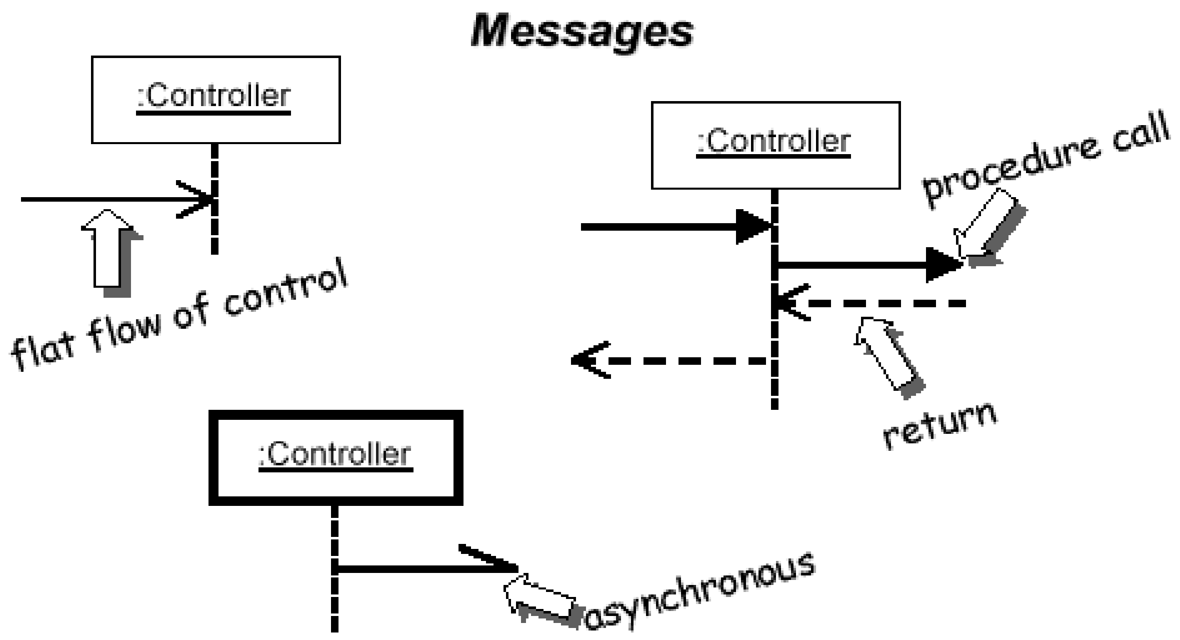
- write message name and arguments above arrow



Messages, continued

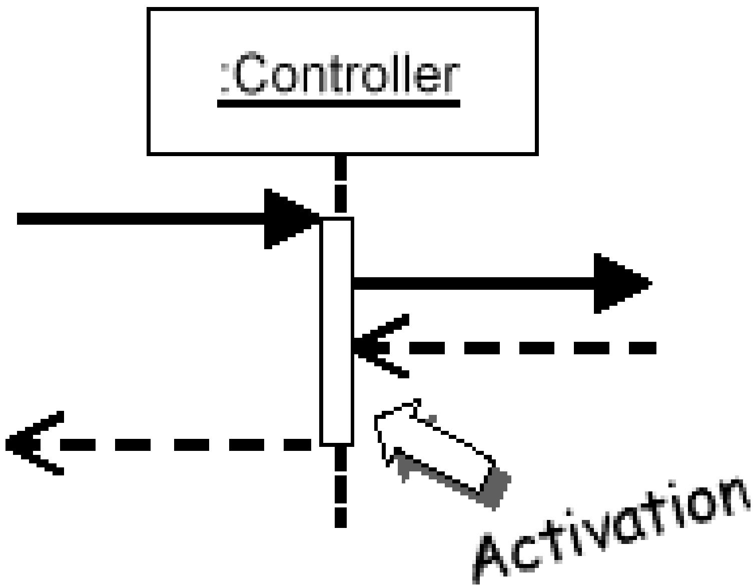
messages (method calls) indicated by arrow to other object

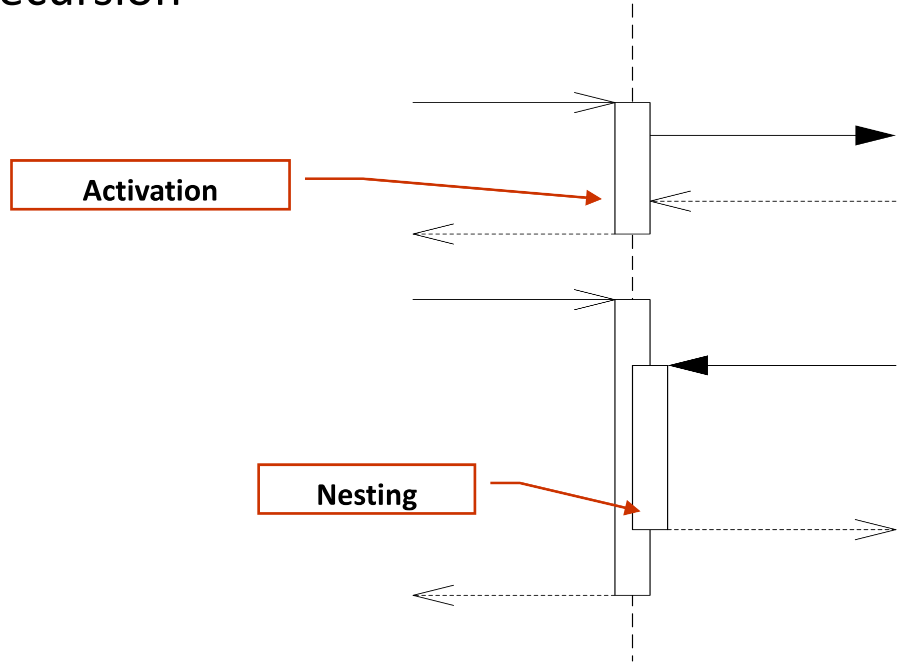
- dashed arrow back indicates return
- different arrowheads for normal / concurrent (asynchronous) calls



Indicating method calls

- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest activations to indicate recursion

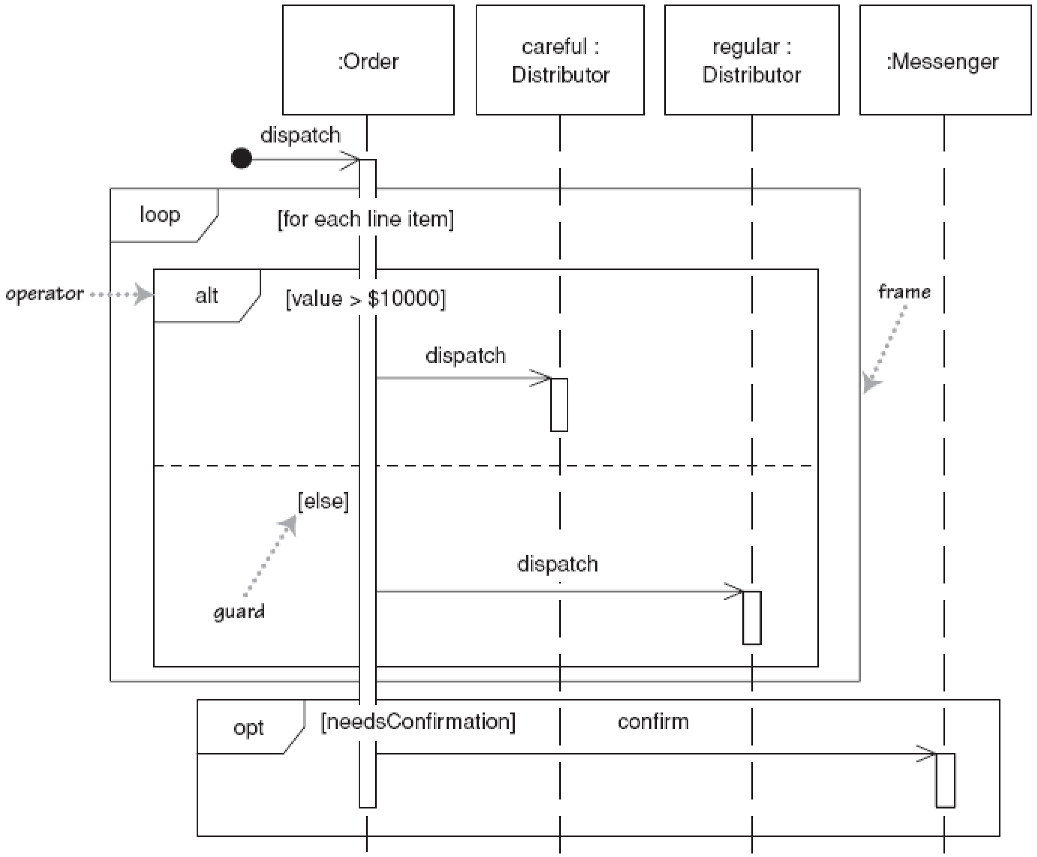




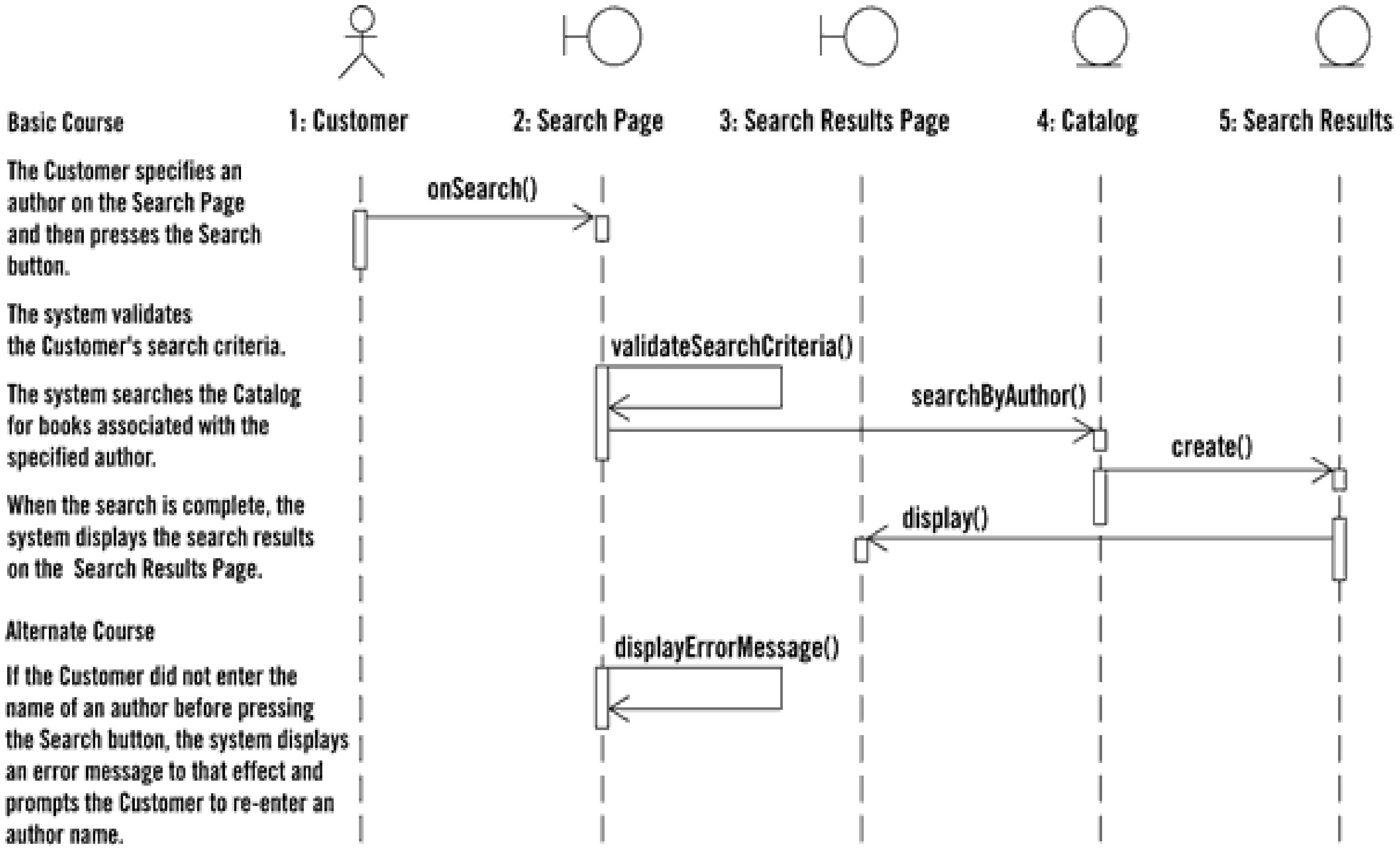
Selection and loops

frame: box around part of diagram to indicate `if` or loop

- `if` -> (opt) [condition]
- `if/else` -> (alt) [condition], separated by horizontal dashed line
- loop -> (loop) [condition or items to loop over]



Sequence diagram from use case scenario



Basic Course

The Customer specifies an author on the Search Page and then presses the Search button.

The system validates the Customer's search criteria.

The system searches the Catalog for books associated with the specified author.

When the search is complete, the system displays the search results on the Search Results Page.

Alternate Course

If the Customer did not enter the name of an author before pressing the Search button, the system displays an error message to that effect and prompts the Customer to re-enter an author name.

Why not just code it?

- Sequence diagrams can be somewhat close to the code level.
- So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)