System Sequence Diagrams

Based on Craig Larman, Chapter 10

Dynamic behaviors

- Class diagrams represent ***static*** relationships. Why?
- What about modeling ***dynamic*** behavior?
- **Interaction** diagrams model how groups of object collaborate to perform some behavior
  - Typically captures the behavior of a single use case

Use Case: Order Entry
1) An Order Entry window sends a "prepare" message to an Order
2) The Order sends "prepare" to each Order Line on the Order
3) Each Order Line checks the given Stock Item
4) Remove appropriate quantity of Stock Item from stock
5) Create a deliver item

Alternative: Insufficient Stock
3a) if Stock Item falls below reorder level
then Stock Item requests reorder

Sequence diagrams

- Vertical line is called an object’s **lifeline**
  - Represents an object’s life during interaction
- Object deletion denoted by X, ending a lifeline
  - Horizontal arrow is a message between two objects
- Order of messages sequences top to bottom
- Messages labeled with message name
  - Optionally arguments and control information
- Control information may express conditions:
  - such as [hasStock], or iteration
- Returns (dashed lines) are optional
  - Use them to add clarity

A time for systems analysis

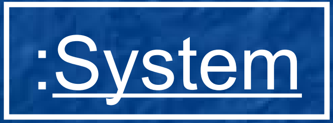
"To every thing there is a season, and a time to every purpose under heaven." Ecclesiastes 3:1

- What UP phase are your projects in?
- What have you determined about your projects?
- Preliminary requirements analysis determines a vision and scope, use cases, risks and estimates
- I.e., is there a project worthy of pursuing?
- Next comes systems analysis:
  - Clarifies input and output System events and which Objects trigger them
  - Generates System Sequence Diagrams from Use Cases
- Why does this activity make sense now?

System Sequence Diagram (SSD)

For a use case scenario, an SSD shows:

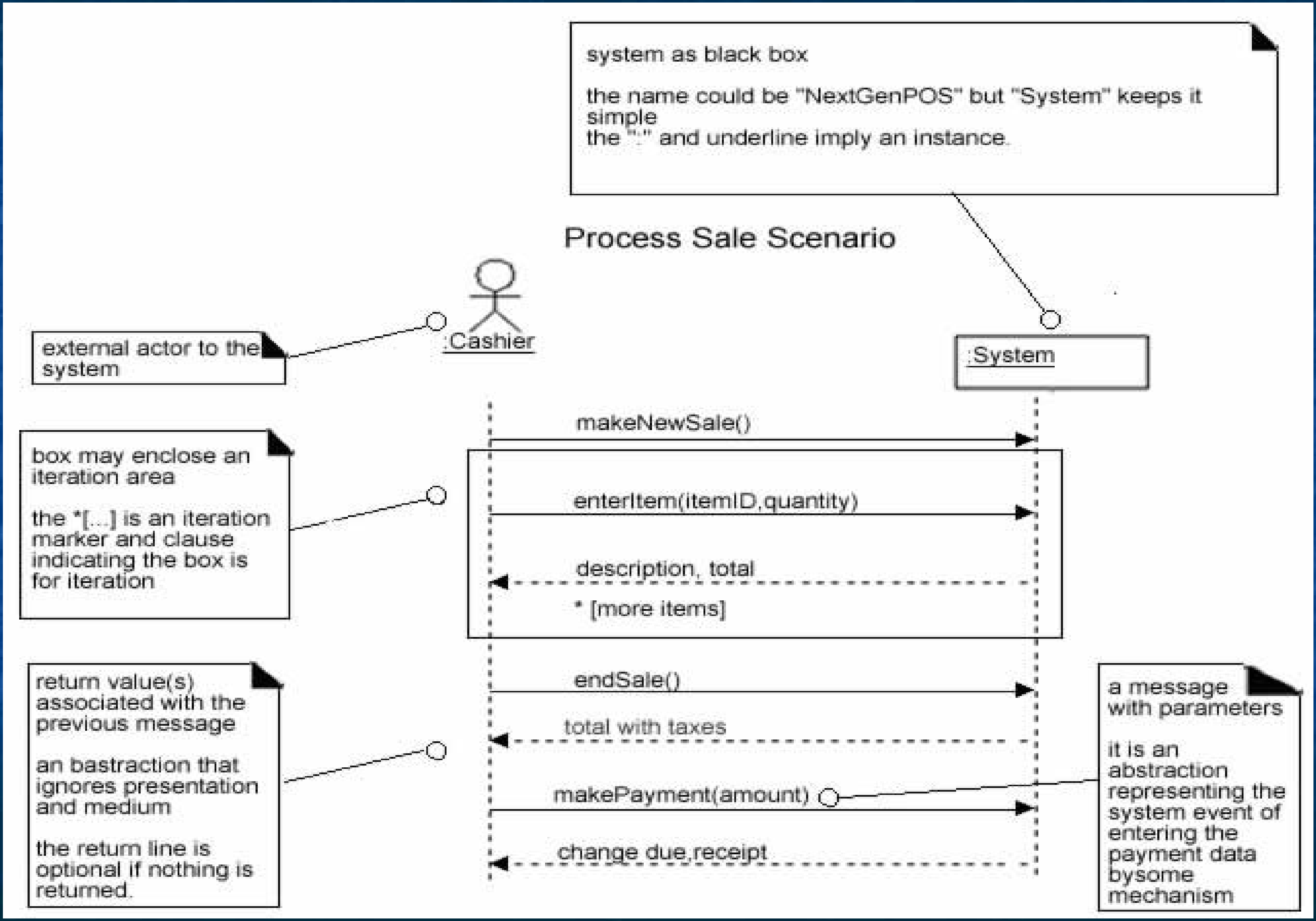
- The System (as a black box)



- The external actors that interact with System
- The System events that the actors generate
- SSD shows operations of the System in response to events, in temporal order
- Develop SSDs for the main success scenario of a selected use case, then frequent and salient alternative scenarios

SSD for Process Sale scenario

(Larman, page 175)


system as black box

the name could be "NextGenPOS" but "System" keeps it simple
the ":" and underline imply an instance.
Process Sale Scenario
external actor to the system
:Cashier
:System
makeNewSale()
box may enclose an iteration area

the *[...] is an iteration marker and clause indicating the box is for iteration
enterItem(itemID,quantity)
description, total
* [more items]
endSale()
return value(s) associated with the previous message

an bastraction that ignores presentation and medium

the return line is optional if nothing is returned.
total with taxes
makePayment(amount)
a message with parameters

it is an abstraction representing the system event of entering the payment data bysome mechanism
change due,receipt

From Use Case to Sequence System Diagram

How to construct an SSD from a use case:

1. Draw System as black box on right side
2. For each actor that directly operates on the System, draw a stick figure and a lifeline.
3. For each System events that each actor generates in use case, draw a message.
4. Optionally, include use case text to left of diagram.

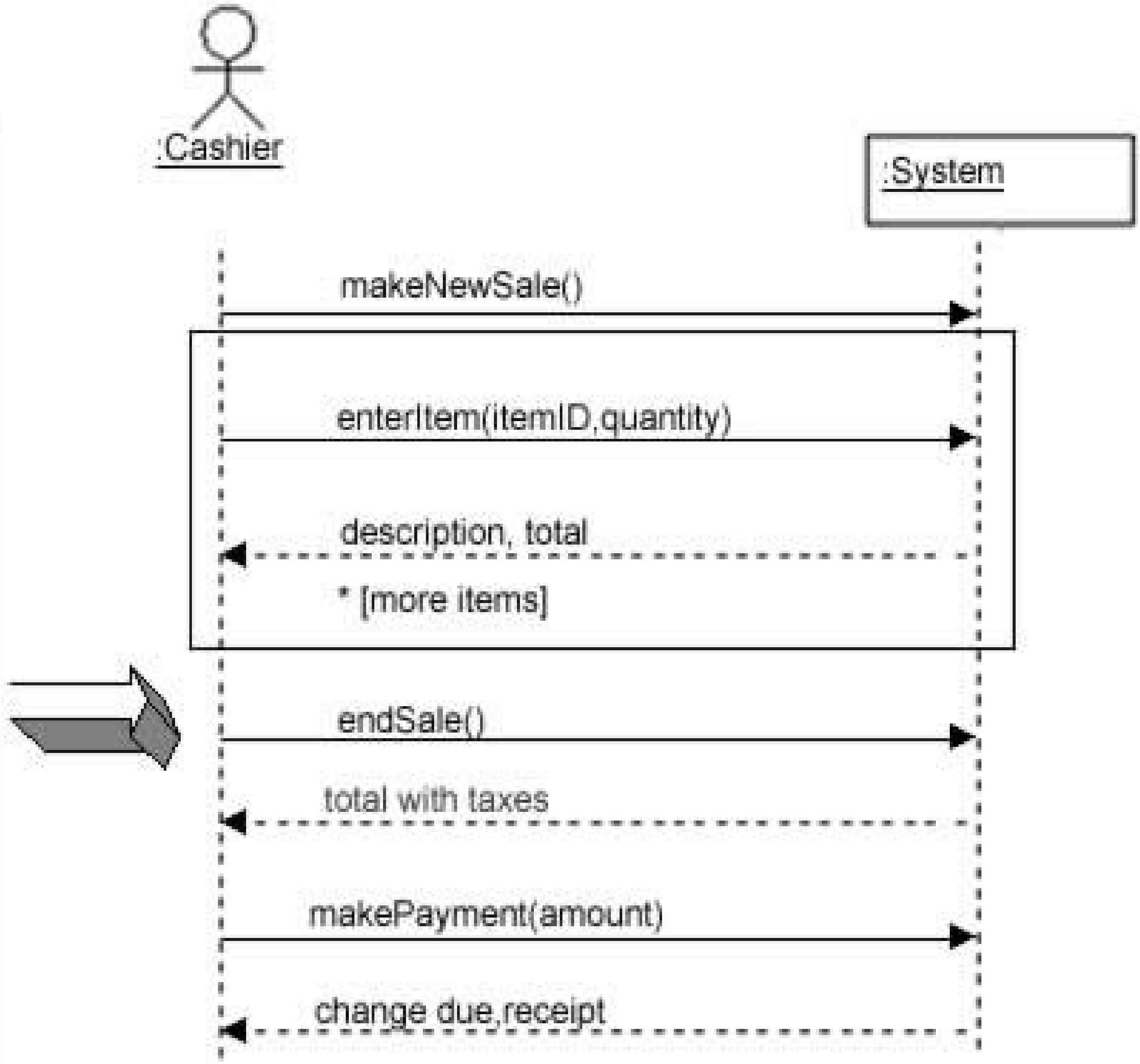
Example: use cases to SSD

Simple Cash-only Process Sale scenario:

1.customer arrives at aPOS check out with goods and/or services to purchase.
2.Cashier starts a new sale.
3.Cashier enters a new item identifier.
4.System records new sale line item and presents item description,price and running total.

Cashier repeats steps steps 3-4 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.
....



Identifying the right Actor

- In the process Sale example, does the customer interact directly with the POS system?
- Who does?
- Cashier interacts with the system directly
- Cashier is the generator of the system events
- Why is this an important observation?

Naming System events & operations

- System events and associated system operations should be expressed at the level of intent
- Rather than physical input medium or UI widget
- Start operation names with verb (from use case)
- Which is better, scanBarCode or enterItem?

SSDs and the Glossary in parallel

- Why is updating the glossary important when developing the SSD?
- New terms used in SSDs may need explanation, especially if they are not derived from use cases
- A glossary is less formal, easier to maintain and more intuitive to discuss with external parties such as customers

SSDs within the Unified Process

Create System Sequence Diagrams during Elaboration in order to:

- Identify System events and major operations
- Write System operation contracts (Contracts describe detailed system behavior)
- Support better estimates
- Remember, there is a season for everything:
  it is not necessary to create SSDs for all scenarios of all use cases, at least not at the same time

Concurrency in Sequence Diagrams

- Concurrent processes:
  - UML 1: **asynchronous** messages as horizontal lines with **half** arrow heads
  - UML 2 makes this distinction by not filling an arrowhead
  - Fowler prefers older notation.
  - *Why? Which do you prefer?*
- After setting up Transaction Coordinator, invoke concurrent Transaction Checkers
  - If a check fails, kill all Transaction Checker processes
- Note use of comments in margin
  - *When is this a good idea?*

Collaboration diagrams

- Objects are rectangular icons
  - e.g., Order Entry Window, Order, etc.
- Messages are arrows between icons
  - e.g., prepare()
- Numbers on messages indicate sequence
  - Also spatial layout helps show flow
- *Which do you prefer: sequence or collaboration diagrams?*
- Fowler now admits he doesn't use collaboration diagrams
  - Interaction diagrams show flow clearly,
    but are awkward when modeling alternatives
- UML notation for control logic has changed in UML 2
  but Fowler isn't impressed