Advanced System Analysis and Design

Understanding Requirements

Introduction to Requirements

- Requirements are system capabilities and conditions to which the system must conform.
- Functional requirements
  - Features and capabilities.
  - Recorded in the Use Case model (see next), and in the systems features list of the Vision artifact.
- Non-functional (or quality requirements)
  - Usability (Help, documentation, …), Reliability (Frequency of failure, recoverability, …), Performance (Response times, availability, …), Supportability (Adaptability, maintainability, …)
  - Recorded in the Use Case model or in the Supplementary Specifications artifact.
- The nature of UP supports changing requirements.

Use-Case Model: Writing Requirements in Context

Use cases and adding value

- Actor: something with behavior, such as a person, computer system, or organization, e.g. a cashier.
- Scenario: specific sequence of actions and interactions between actors and the system under discussion, e.g. the scenario of successfully purchasing items with cash.
- Use case: a collection of related success and failure scenarios that describe actors using a system to support a goal.

Use cases and adding value

Handle returns

Main success scenario: A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item…

Alternate scenarios:

If the credit authorization is reject, inform customer and ask for an alternative payment method.

If item identifier not found in the system, notify the Cashier and suggest manual entry of the identifier code.

…

Use cases and adding value

- A key point is to focus on the question “how can using the system provide observable value to the user, or fulfill their goals?”
- Use cases mainly constitute functional requirements.

Use case types and formats

- Black-box use cases describe system responsibilities, i.e. define what the system must do.
- Uses cases may be written in three formality types
  - Brief: one-paragraph summary, usually of the main success scenario.
  - Casual: Informal paragraph format (e.g. Handle returns)
  - Fully dressed: elaborate. All steps and variations are written in detail.

Fully-dressed example: Process Sale

<u>Use case UC1: Process Sale</u>
<u>Primary Actor</u>: Cashier
<u>Stakeholders and Interests:</u>
- -Cashier: Wants accurate and fast entry, no payment errors, …
- -Salesperson: Wants sales commissions updated.
- …

<u>Preconditions</u>: Cashier is identified and authenticated.
<u>Success Guarantee (Postconditions):</u>
- -Sale is saved. Tax correctly calculated.
- …

<u>Main success scenario (or basic flow):</u> [see next slide]
<u>Extensions (or alternative flows):</u> [see next slide]
<u>Special requirements</u>: Touch screen UI, …
<u>Technology and Data Variations List</u>:
- -Identifier entered by bar code scanner,…

<u>Open issues</u>: What are the tax law variations? …

Fully dressed example: Process Sale (cont.)

<u>Main success scenario (or basic flow):</u>

The Customer arrives at a POS checkout with items to purchase.
The cashier records the identifier for each item. If there is more than one of the same item, the Cashier can enter the quantity as well.
The system determines the item price and adds the item information to the running sales transaction. The description and the price of the current item are presented.
On completion of item entry, the Cashier indicates to the POS system that item entry is complete.
The System calculates and presents the sale total.
The Cashier tells the customer the total.
The Customer gives a cash payment (“cash tendered”) possibly greater than the sale total.

<u>Extensions (or alternative flows):</u>

If invalid identifier entered. Indicate error.
If customer didn’t have enough cash, cancel sales transaction.

Goals and Scope of a Use Case

- At what level and scope should use cases be expressed?
- A: Focus on uses cases at the level of **elementary business process** (EBP).
- EBP: a task performed by one person in one place at one time which adds measurable business value and leaves the data in a consistent state.
  - Approve credit order - OK.
  - Negotiate a supplier contract - not OK.
- It is usually useful to create separate “sub” use cases representing subtasks within a base use case.
  - e.g. Paying by credit

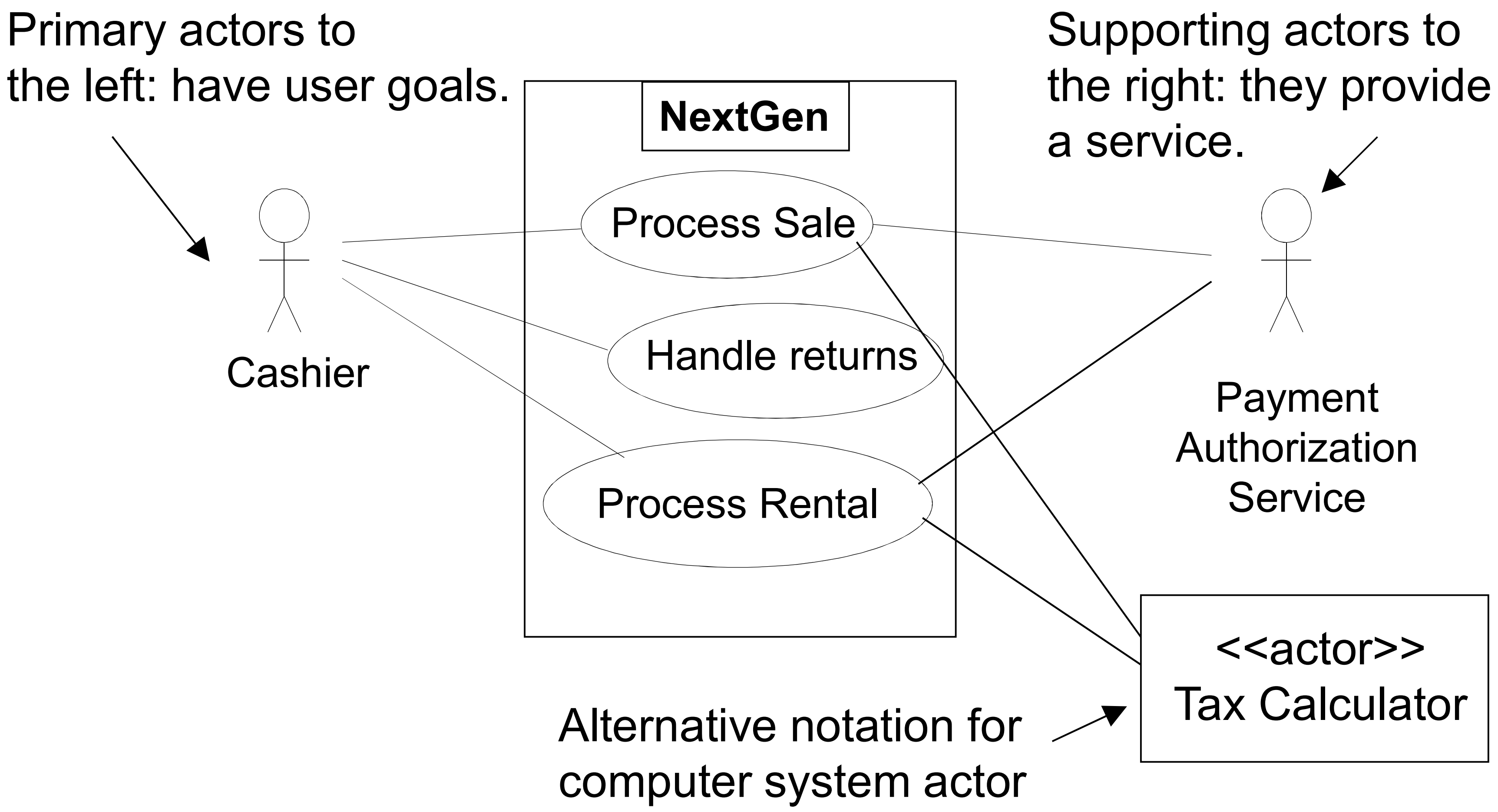
Finding primary actors, goals and use cases

- Choose the system boundary.
- Identify primary actors.
  - Those that have user goals fulfilled through using services of the system
- For each actor identify their user goals.
  - Tabulate findings in the Vision artifact.
- Define use cases that satisfy user goals; name them according to their goal.

Essential vs. Concrete style

- Essential: Focus is on intend.
  - Avoid making UI decisions
- Concrete: UI decisions are embedded in the use case text.
  - e.g. “Admin enters ID and password in the dialog box (see picture X)”
  - Concrete style not suitable during early requirements analysis work.

Use Case Diagrams


Primary actors to the left: have user goals.
NextGen
Process Sale
Handle returns
Process Rental
Cashier
Supporting actors to the right: they provide a service.
Payment Authorization Service
<<actor>> Tax Calculator
Alternative notation for computer system actor

From Inception to Elaboration

Iteration 1 Requirements

- Implement a basic key scenario of the Process Sale use case: entering items and receiving a cash payment.
- No collaboration with external devices (such as tax calculator or product database)
- No complex pricing rules are applied.
- Subsequent iterations will grow on this foundation.

Incremental Development for the Same Use Case Across Iterations

- Not all requirements in the Process Sale use case are being handled in iteration 1.
- It is common to work on varying scenarios or features of the same use case over several scenarios and gradually extend the system to ultimately handle all the functionality required.
- On the other hand, short, simple use cases may be completed within one iteration.

Use-Case Model: Drawing System Sequence Diagrams

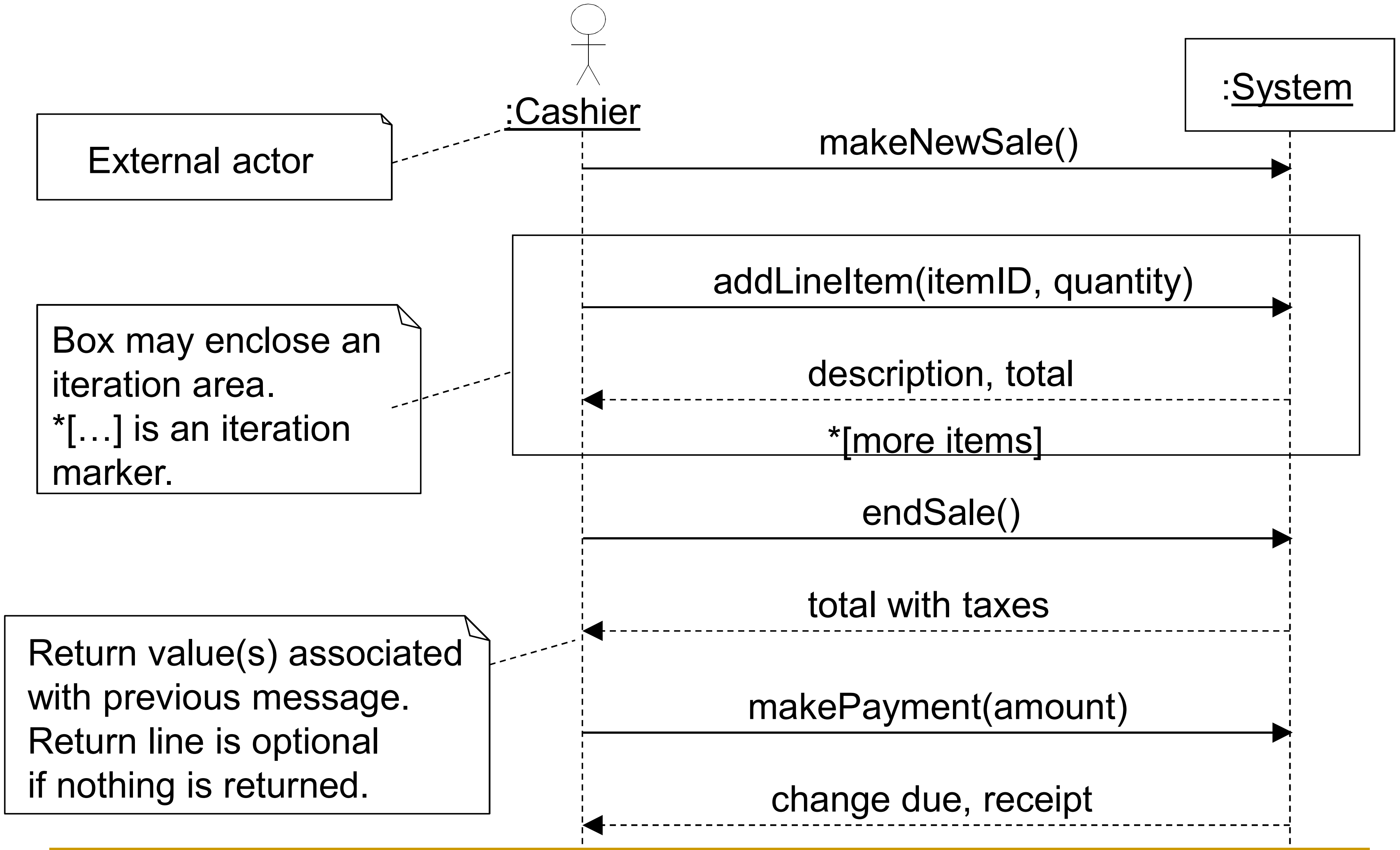
System Behavior and UML Sequence Diagrams

- It is useful to investigate and define the behavior of the software as a “black box”.
- System behavior is a description of *what the system does* (without an explanation of how it does it).
- Use cases describe how external actors interact with the software system. During this interaction, an actor generates events.
- A request event initiates an operation upon the system.

System Behavior and System Sequence Diagrams (SSDs)

- A sequence diagram is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events.
- All systems are treated as a black box; the diagram places emphasis on events that cross the system boundary from actors to systems.

SSDs for Process Sale Scenario

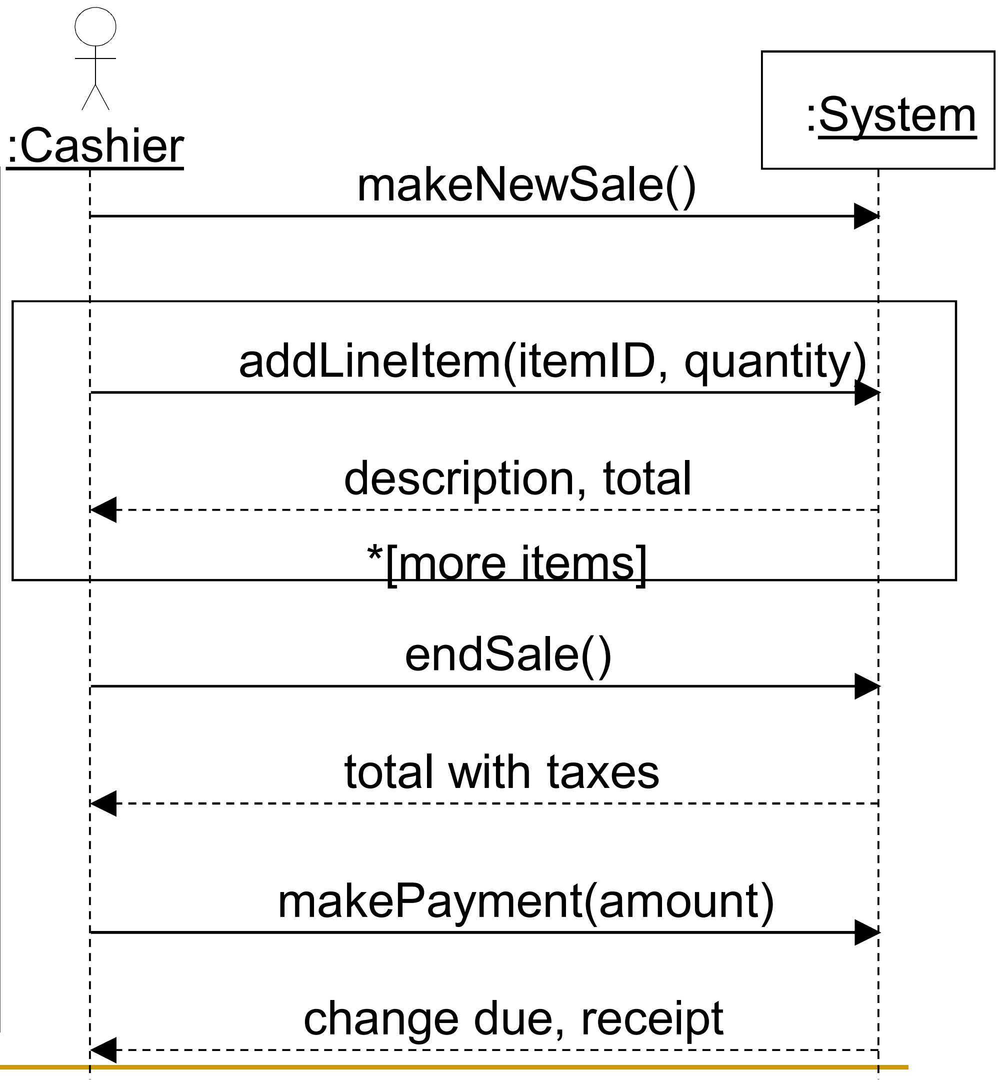

:Cashier
:System
External actor
makeNewSale()
addLineItem(itemID, quantity)
description, total
*[more items]
Box may enclose an iteration area. *[…] is an iteration marker.
endSale()
total with taxes
Return value(s) associated with previous message. Return line is optional if nothing is returned.
makePayment(amount)
change due, receipt

SSD and Use Cases

Simple cash-only Process Sale Scenario

1. Customer arrives at a POS checkout with goods 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.
   cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.

…



Naming System Events and Operations

- The set of all required system operations is determined by identifying the system events.
  - makeNewSale()
  - addLineItem(itemID, quantity)
  - endSale()
  - makePayment(amount)