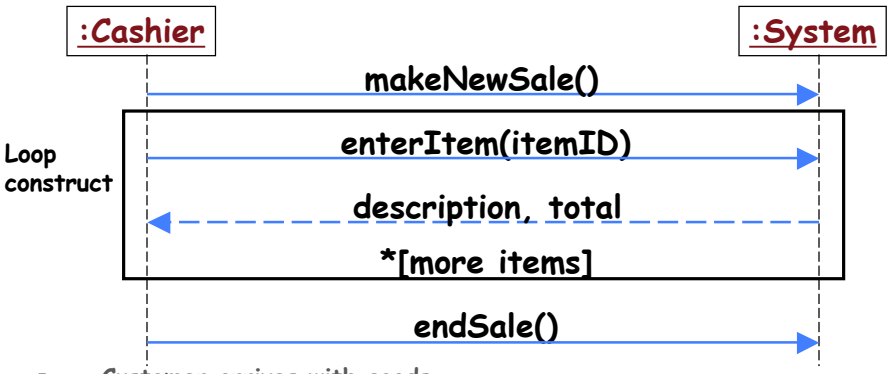
Requirements Analysis - 2

- Focusing on the WHAT not the HOW
- System sequence diagrams and how they relate to use cases

System Sequence Diagram (SSD)

- Describes in more detail a scenario in a use case
  - Created from the text of the use case
- A kind of UML sequence diagram
  - SSD is a simplified version useful for requirements analysis
  - More general version of sequence diagrams: later, when talking about design
- Example: *Process Sale* for POS system

Partial SSD for the Main Scenario



- Customer arrives with goods
- Cashier starts a new sale
- REPEAT UNTIL DONE:
  - Cashier enters item id
  - System records sale line item and presents description and running total

Elements of a SSD

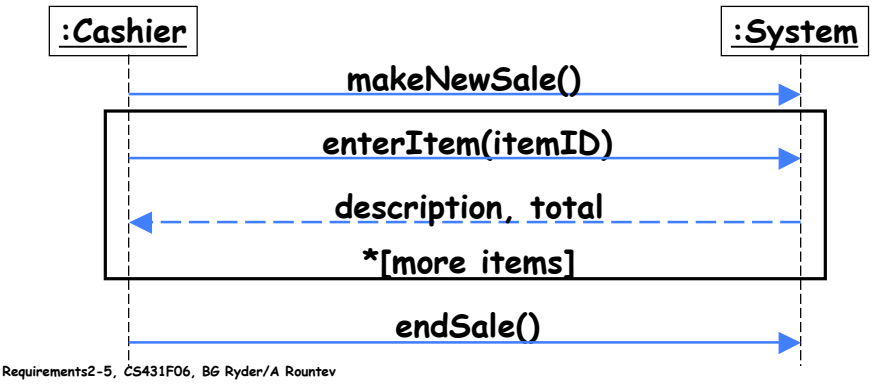
- **Actors** :Cashier

  - **UML notation for an object**
- **System events**
  - **Cashier generates** makeNewSale, enterItem, **and** endSale **system events**
- **(Optional) information from the system back to the actors**
  - **item description, running total, etc.**

SSD for "Process Sale"

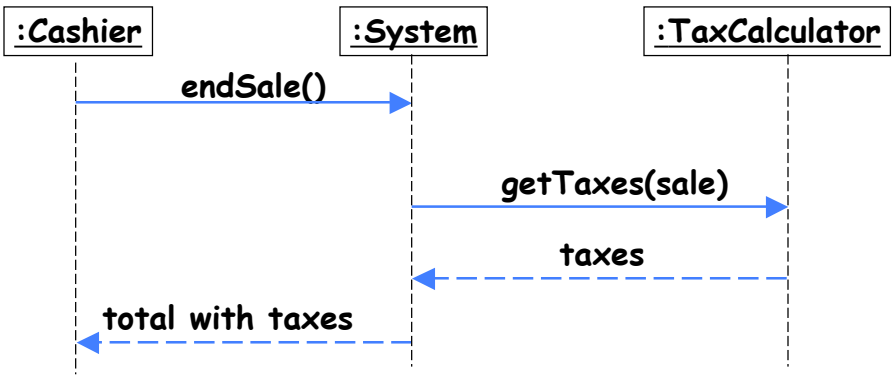
- Cashier starts a new sale
- Cashier enters item id
- System records sale line item and presents description and running total

Repeat 3-4 until Cashier indicates "done"



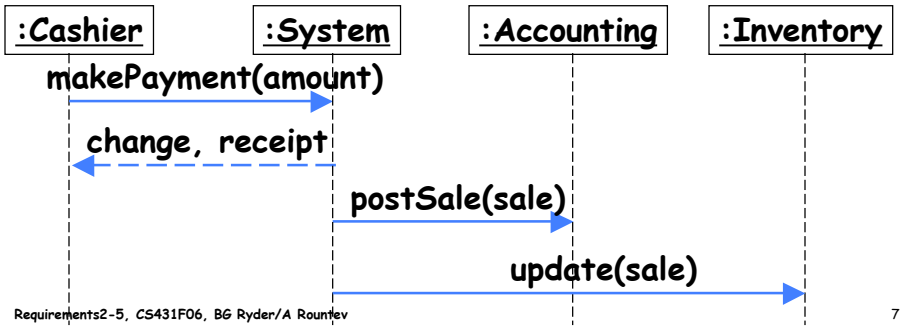
SSD for "Process Sale" (cont)

- System presents total with taxes. To determine taxes, System uses an external Tax Calculator



SSD for "Process Sale" (cont)

- Cashier enters cash amount tendered, and System presents change due
- System presents receipt
- System logs completed sale and sends sale info to the external Accounting system and to the external Inventory system



Abstractions in SSDs

- Events and return values are abstractions
  - Independent of mechanism & representation
- *makePayment(amount)*
  - Shows input info
  - Looks like a method call, but is really an abstraction of an event
- Name: should capture the intent
  - Avoid specifying implementation choices
    - *enterItem(itemID)* is better than *scan(itemID)*

Timeline for SSDs

- SSDs are created during elaboration
  - Clarify the major events that the system should be able to handle
  - Later we design objects to handle these events (object-oriented design)
- SSDs are created for some chosen scenarios from the current iteration
  - Happy path + frequent/complex alternatives

Alternative Scenarios

In the main scenario:

3. Cashier enters item identifier

In the Extensions part of the use case:

3a. Invalid identifier:

1. System signals error and rejects entry

Condition: triggers this alternative

Handing: one or more steps

Alternative Scenarios

In the main scenario:

3. Cashier enters item identifier

In the Extensions part of the use case:

3b. There are multiple items of the same category (e.g., 5 bottles of Coke)

1. Cashier enters item id & quantity

Multiple conditions 3a,3b for one step from the main scenario,with different handlers

Alternative Scenarios

In the main scenario:

6. Cashier asks customer for payment

In the Extensions part of the use case:

6a. Customer doesn't have enough cash

1. Cashier asks for alternative payment method

1a. Customer tells Cashier to cancel sale; Cashier cancels sale on System

An example of a failure scenario

Another Example

In the main scenario:

5. System presents total with taxes. To determine taxes, it uses a TaxCalculator

Alternative: What if some customers are entitled to a discount? – e.g., employees

- Suppose that each such customer has a customer_id that determines discount %
  - Ids and discount % are stored in an external CustomerInfo system

Another Example

In the main scenario:

5. System presents total with taxes …

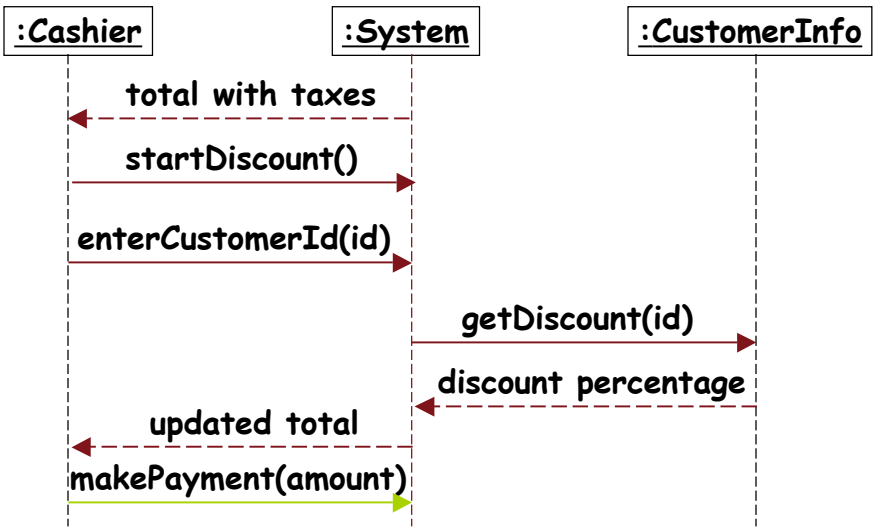
In the Extensions part of the use case:

5a. Customer is eligible for discount

1. Cashier signals discount
2. Cashier enters customer id
3. System presents updated total.

System uses external CustomerInfo system to get discount percentage

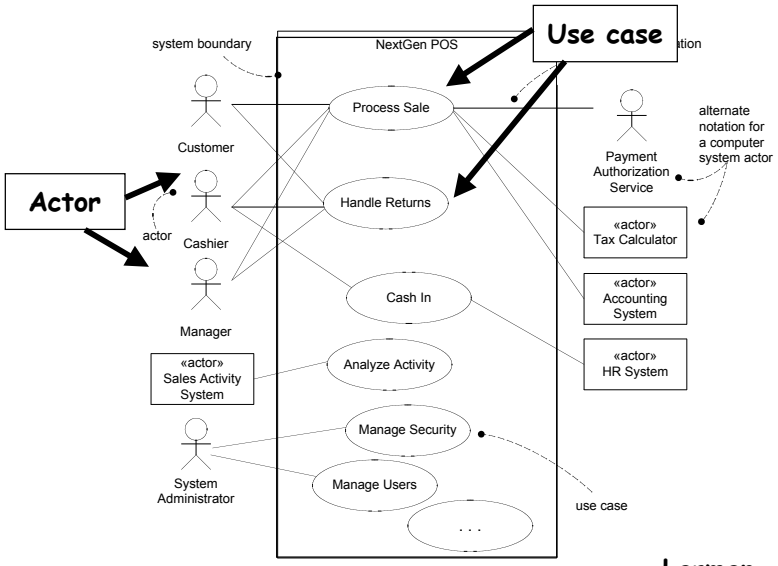
SSD for this scenario



Additional Info in a Use Case

- Non-functional requirements
  - Usability: "The text on the screen should be visible from 5 feet"
  - Performance: "Credit authorization response should be within 30 sec, 90% of the time"
  - Technology: "Identifier entered by laser scanner or by keyboard"
- Eventually gathered in the Supplementary Specification

Use case diagram



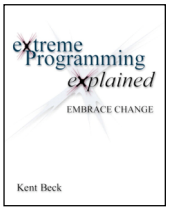
Larman, p 90

UML Use Case Diagram

- UML notation for representing actors, use cases, and their relationships
- The diagram is secondary to the actual use cases: need to focus on text
  - Use-Case Model = text of the use cases
- In this class, the use case diagram is shown only for completeness

Constructing the Use Cases

- Iteratively refined during inception and elaboration
- Communication-intensive process
  - Developers need to talk with domain experts
  - e.g., Extreme Programming requires a user to be co-located full-time with the development team



- Focus on user intentions and goals

Timeline

Artifact	Incep	Elab	Const	Trans
Use-Case Model	X	X		
Supplem. Spec	X	X		
Domain Model		X		
Design Model		X	X	
Implem. Model		X	X	X

Requirements analysis: Use-Case Model + Supplementary Specification

Domain analysis: Domain Model

Design: Design Model

Coding: Implementation Model

Supplementary Specification

- Describes other requirements
  - In addition to the functional requirements described by the use cases
- Functional requirements **common across many use cases**
  - Logging and error handling: e.g. "Log all errors to persistent storage"
  - Security: e.g. "All usage requires user authentication"

Supplementary Specification

- Usability requirements
  - "Avoid colors associated with common forms of color blindness"
  - "The cashier is often looking at the customer, so signals and warnings should be conveyed w/ sound"
- Reliability requirements
  - Recoverability: "If there is a failure to use external services (e.g. accounting system), store the information locally to complete the sale"

Supplementary Specification

- Performance requirements
  - "External payment authorization should be completed within 30 seconds, in 90% of the cases"
- Supportability requirements
  - Adaptability: "At certain points in the scenarios, pluggable rules should be enabled to accommodate different customers"
  - Configurability: "Different customers will have different network configurations"

Supplementary Specification

- Implementation constraints
  - "Management insists on Java for long-term portability, supportability, and ease of development"
- Requirements for purchased components
  - The tax calculator will be purchased from a third party. The system must support pluggable tax calculators for different countries
- And many more: business rules, legal issues, standards, I/O devices, tools, ...

Role of the Supplementary Spec

- Describes issues that are not easily captured by use cases
- Particularly important are system-wide attributes
  - Performance, reliability, testability, etc.
  - Central role during early design and implementation
    - E.g., a system-wide requirement for high fault tolerance has very significant influence on large-scale design decisions

UP Timeline

- Inception: the supplementary specification is only lightly developed
  - Focus on risky system-wide requirements
- Elaboration: continuous refinement
  - In parallel with early design/implementation
  - Feedback for the requirements analysis
  - Stabilized at the end of elaboration

Real Users …

- Real users never know what they want, but they always know when your program doesn't deliver it
- Real users never stop asking for new options
- Real users never know what to do with new options
- Real users never read the documentation