ICT2622 Object-Oriented Analysis Notes Phase Ch 5 – Summary Notes
SATZINGER JACKSON BURD (6TH EDITION)

CHAPTER 5: Extending the Requirements Models

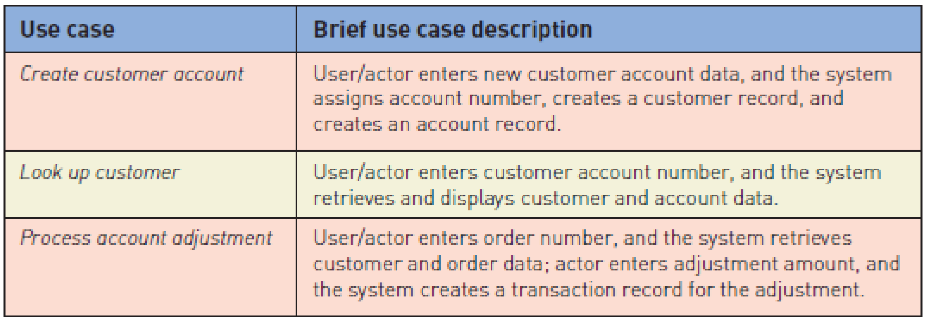
USE CASE DESCRIPTIONS

- A list of use cases and use case diagrams provides an overview of all the use cases for a system.
- Detailed information about each use case is described with a use case description.
- A use case description is a textual model lists and describes the processing details for a use case.
- Implied in all use cases is a person who uses the system - an actor.
- An actor is always outside the automation boundary of the system but may be part of the manual portion of the system.
- By defining actors as those who interact with the system, the exact interactions to which the automated system must respond can be defined. An actor could also be seen as a role. For example, the use case Create Customer Account might involve a customer service rep talking to the customer on the phone.
- To create a comprehensive, robust system that truly meets users' needs, the detailed steps of each use case must be documented.

A **scenario** or **use case instance** is a unique set of internal activities within a use case and represents a unique path through the use case.

Brief Use Case Descriptions

- Brief description can be used for very simple use cases - system to be developed is a small, well-understood application.
- A simple use case would normally have a single scenario and few exception conditions.



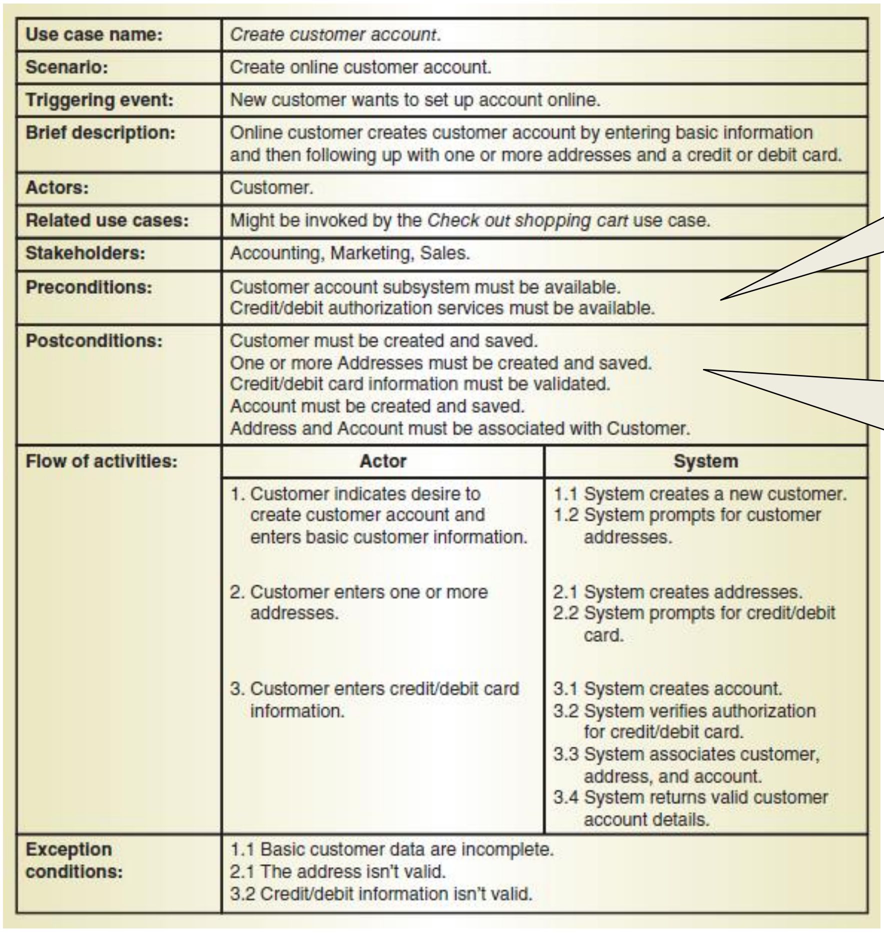
Use case	Brief use case description
Create customer account	User/actor enters new customer account data, and the system assigns account number, creates a customer record, and creates an account record.
Look up customer	User/actor enters customer account number, and the system retrieves and displays customer and account data.
Process account adjustment	User/actor enters order number, and the system retrieves customer and order data; actor enters adjustment amount, and the system creates a transaction record for the adjustment.

Fully Developed Use Case Descriptions

- **Fully developed use case description** - most formal method for documenting the use case.
- A fully developed use case description increases the probability that the analyst/developers understand the business processes and the ways the system must support them

CHAPTER 5: Extending the Requirements Models

Example: Fully developed use case description for Create customer account (Standard template)



1	**Use case name:**	*Create customer account.*	
2	**Scenario:**	Create online customer account.	
3	**Triggering event:**	New customer wants to set up account online.	
4	**Brief description:**	Online customer creates customer account by entering basic information and then following up with one or more addresses and a credit or debit card.	
5	**Actors:**	Customer.	
6	**Related use cases:**	Might be invoked by the *Check out shopping cart* use case.	
7	**Stakeholders:**	Accounting, Marketing, Sales.	
8	**Preconditions:**	Customer account subsystem must be available. Credit/debit authorization services must be available.	
9	**Postconditions:**	Customer must be created and saved. One or more Addresses must be created and saved. Credit/debit card information must be validated. Account must be created and saved. Address and Account must be associated with Customer.	
10	**Flow of activities:**	**Actor**	**System**
		1. Customer indicates desire to create customer account and enters basic customer information.	1.1 System creates a new customer. 1.2 System prompts for customer addresses.
		2. Customer enters one or more addresses.	2.1 System creates addresses. 2.2 System prompts for credit/debit card.
		3. Customer enters credit/debit card information.	3.1 System creates account. 3.2 System verifies authorization for credit/debit card. 3.3 System associates customer, address, and account. 3.4 System returns valid customer account details.
11	**Exception conditions:**	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.	

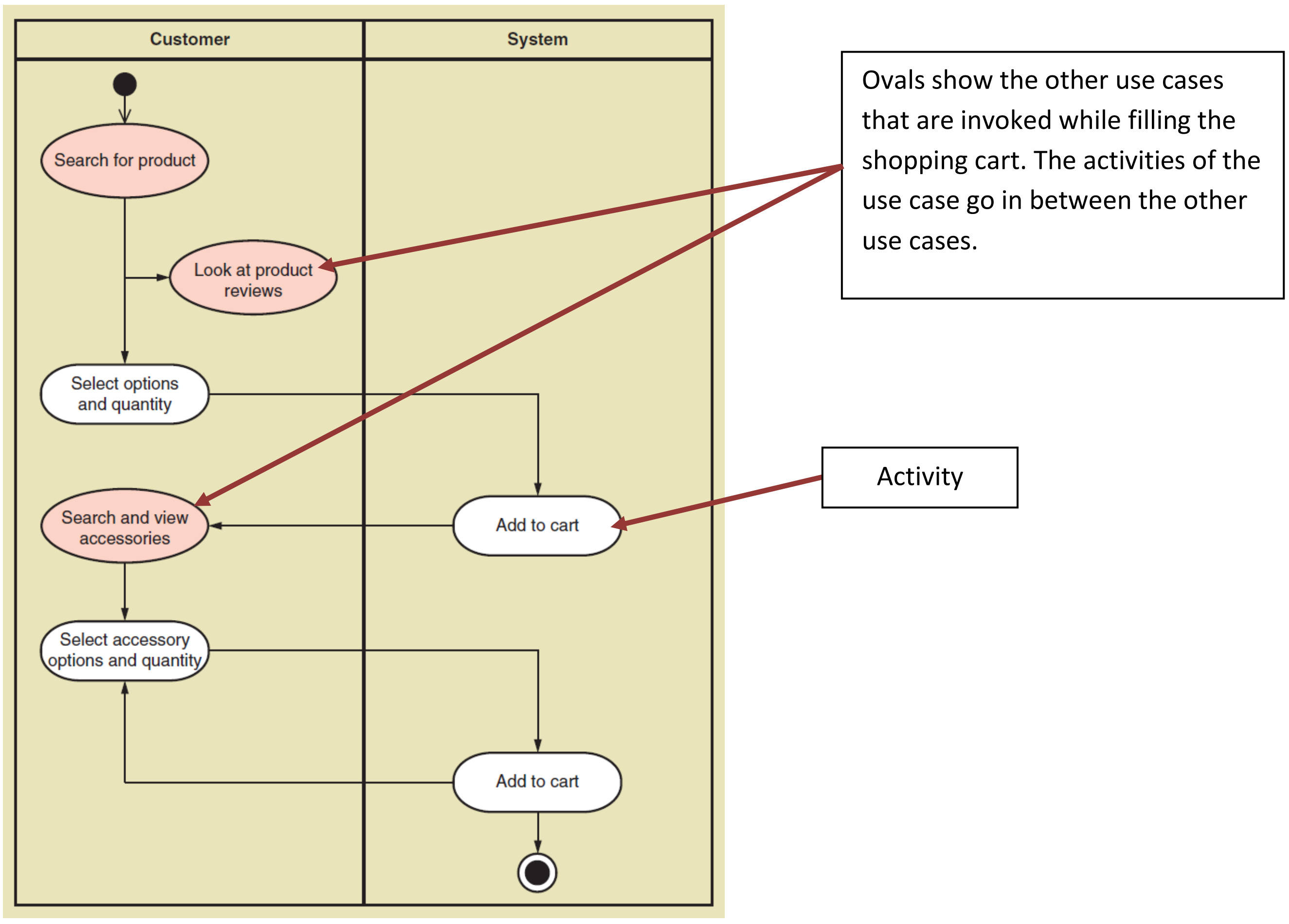
Preconditions = The state of the system must be in in order for the use case to begin, what objects must exist, information required, and condition of the actor prior to beginning the use case

Postconditions = What must be true upon completion of the use case. Indicate what new objects are created or updated by the use case and how objects need to be associated. Are the expected results and indicate which objects are important for design

ACTIVITY DIAGRAMS FOR USE CASES

- Document a use case with an activity diagram
- Activity diagram is an easily understood diagram to document the workflows of the business processes
- Activity diagram is a standard UML diagram, documents flow of activities for each use case
- are helpful when the flow of activities for a use case is complex
-

CHAPTER 5: Extending the Requirements Models


Customer
System
Search for product
Look at product reviews
Select options and quantity
Add to cart
Search and view accessories
Select accessory options and quantity
Add to cart
Ovals show the other use cases that are invoked while filling the shopping cart. The activities of the use case go in between the other use cases.
Activity

Activity diagram for Fill shopping cart showing richer user experience

THE SYSTEM SEQUENCE DIAGRAM—IDENTIFYING INPUTS AND OUTPUTS

- In O-O approach, the flow of information is either to and from actors or back and forth between internal objects.
- System sequence diagram (SSD) a diagram showing the sequence of messages between an external actor and the automated system during a use case or scenario

- The SSD documents the inputs and the outputs and identifies the interaction between actors and the system.
- An SSD is a type of interaction diagram.

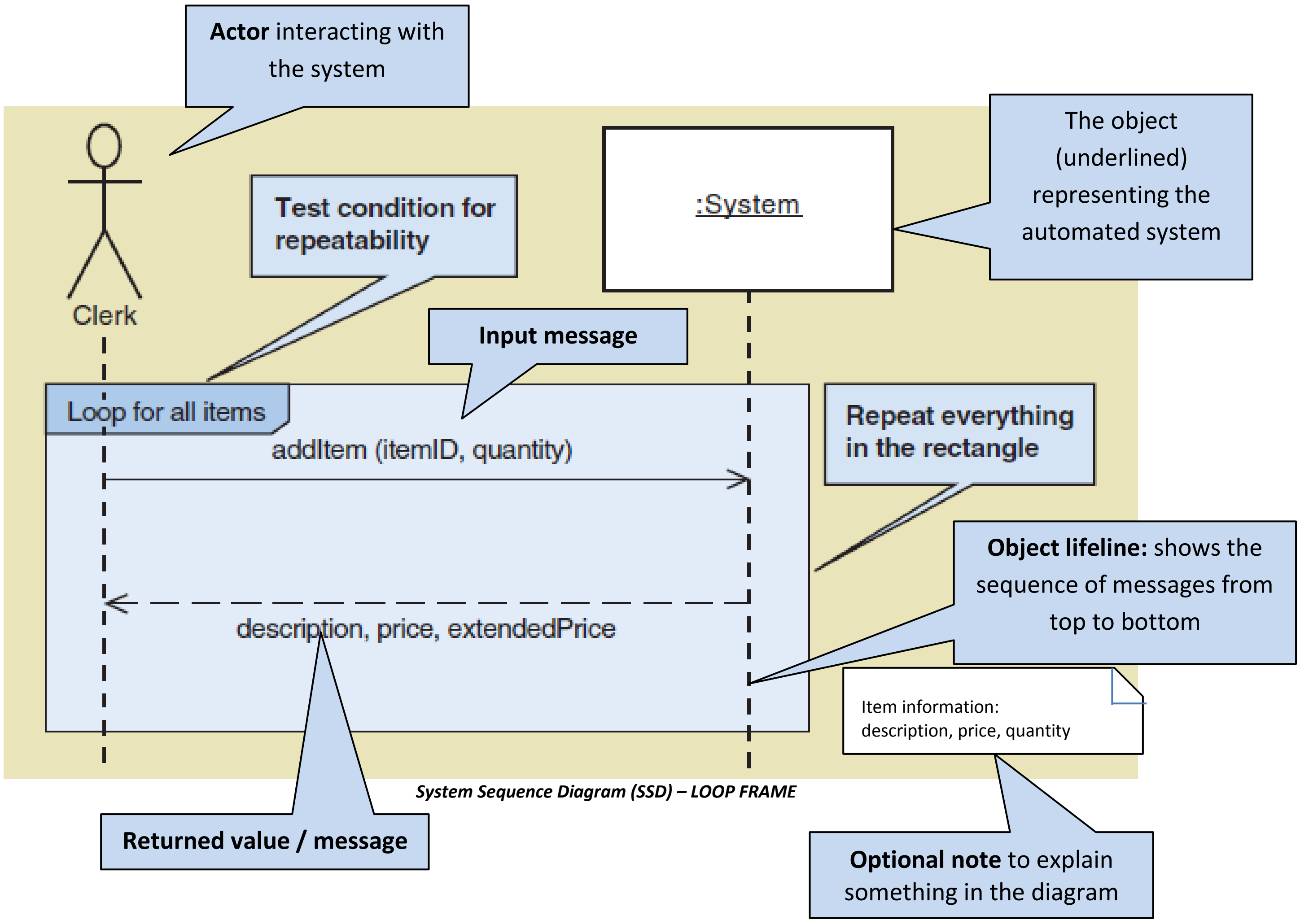
- Interaction diagram - either a communication diagram or a sequence diagram that shows the interactions between objects

SSD Notation

- A generic SSD is shown below.
- As with a use case diagram, the stick figure represents an actor — a person (or role) that interacts with the system.
- In a use case diagram, the actor "uses" the system, but the emphasis in an SSD is on how the actor "interacts" with the system by entering input data and receiving output data.
- The box labelled **:System** is an object that represents the entire automated system.
- **In SSDs and all other interaction diagrams, analysts use object notation** instead of class notation
- In object notation, a box refers to an individual object, not the class of all similar objects.
- The notation is simply a rectangle with the name of the object underlined.
- In an SSD, the only object included is one representing the entire system.
- Underneath the actor and **:System** are vertical dashed lines called **lifelines**.

CHAPTER 5: Extending the Requirements Models

- A lifeline, or object lifeline, is the extension of that object — either actor or object — during the use case.
- The arrows between the lifelines represent the message and input that are sent by the actors.
- Each arrow has an origin and a destination.
- The origin of the message is the actor or object that sends it
- The destination actor or object of a message is indicated by the lifeline that is touched by the arrowhead.
- Purpose of lifelines is to indicate the sequence of the messages sent and received by the actor and object.
- The sequence of messages is read from top to bottom in the diagram.
- A message is labelled to describe its purpose and any input data being sent.
- The message name should follow the verb-noun syntax to make the purpose clear.
- The arrows are used to represent a message and input data


Actor interacting with the system
Clerk
Test condition for repeatability
:System
The object (underlined) representing the automated system
Input message
Loop for all items
addItem (itemID, quantity)
Repeat everything in the rectangle
Object lifeline: shows the sequence of messages from top to bottom
description, price, extendedPrice
Item information: description, price, quantity
Returned value / message
Optional note to explain something in the diagram

System Sequence Diagram (SSD) – LOOP FRAME

- **SSD Loop frames** – The message and its return are located inside a larger rectangle where everything in the frame is repeated

  Notation:
  ***[true/false condition] return-value := message-name (parameter-list)**
  e.g. *address details := enterAddress (address) where *= loop indicator

- The **parameter-list** for each message is more difficult to determine - several iterations may be required before a correct, complete list is obtained. Attributes from the classes should be listed as parameters.

ICT2622 Object-Oriented Analysis Notes Phase Ch 5 – Summary Notes
SATZINGER JACKSON BURD (6TH EDITION)

CHAPTER 5: Extending the Requirements Models

- Sequence diagrams use two additional frames to depict processing logic, as shown in Fig A and Fig B below.
- The **opt frame** is used when a message or a series of messages is **optional** or **based on some true/false condition**.
- The **alt frame** is used with if-then-else logic.

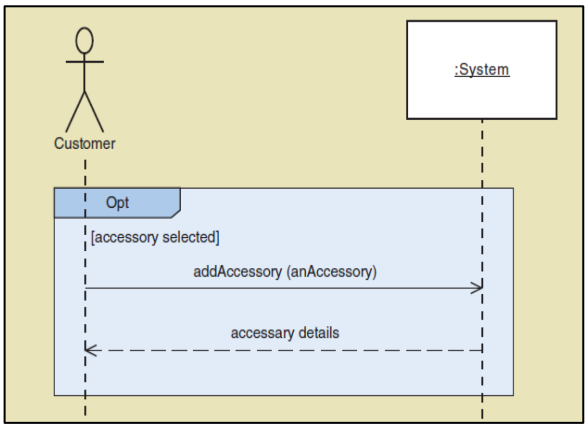


Figure A: Opt Frame (true/false condition)

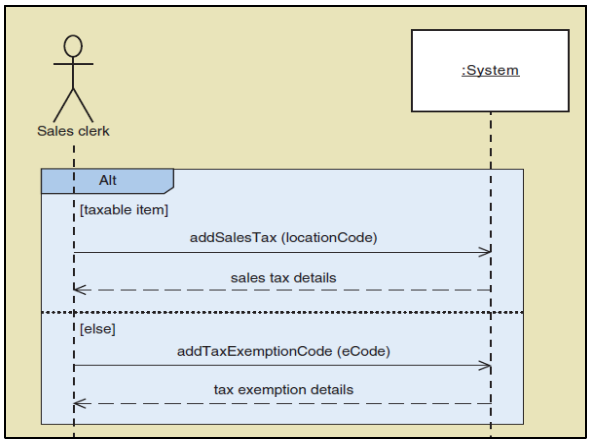
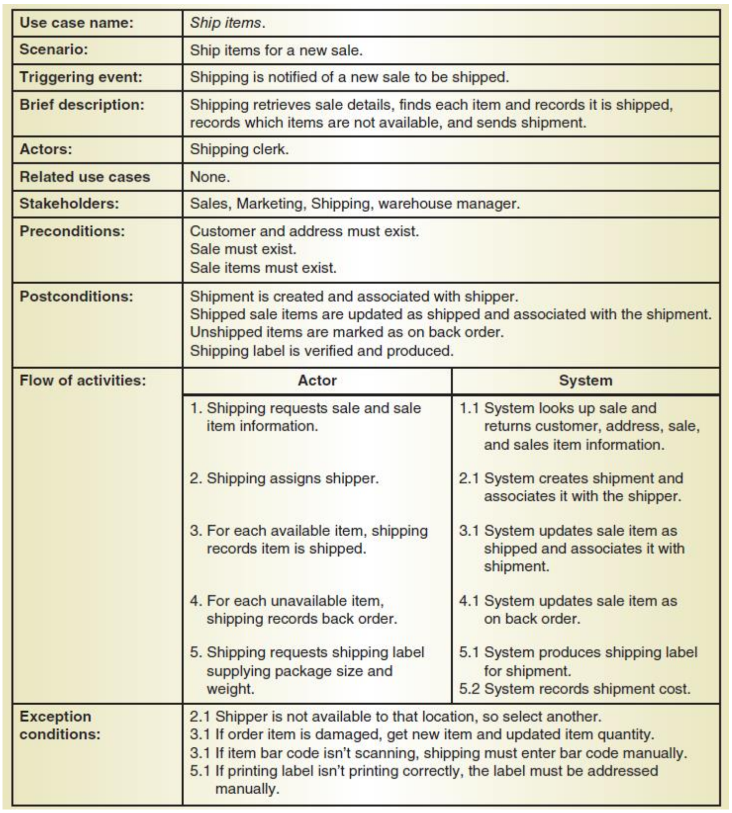


Figure B: Alt frame (if, then, else)

The development of an SSD based on an activity diagram falls into four steps:

1. **Identify the input messages** — identify where there are locations with a workflow arrow crossing the boundary line between the actor and the system. At each location that the workflow crosses the automation boundary, input data is required; therefore, a message is needed.
2. **Describe the message from the external actor to the system** by using the message notation described earlier (names of the messages reflect the services that the actor is requesting of the system e.g. createNewCustomer, createAddress, and enterCreditCard - note verb is in small letters, noun with capital letter)
3. **Identify and add any special conditions on the input messages**, including iteration and true/false conditions
4. **Identify and add the output return messages** - There are two options for showing return information: as a return value on the message itself or as a separate return message with a dashed arrow.

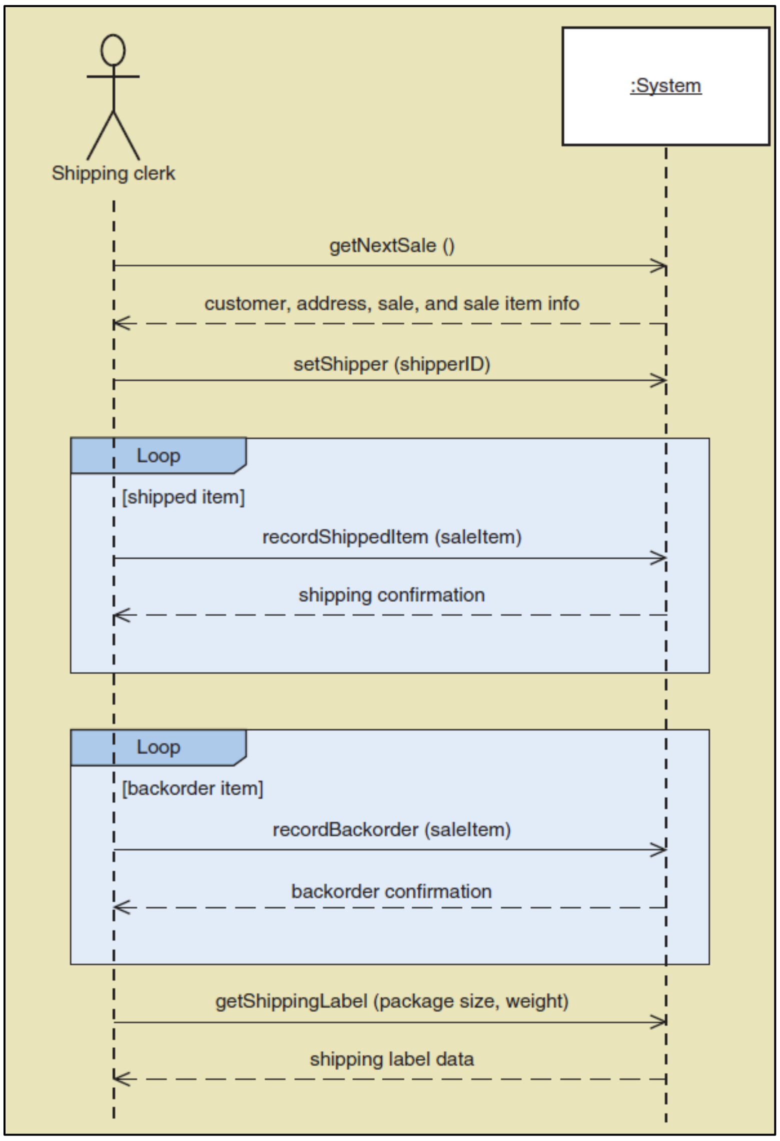
<u>EXAMPLE</u>: Develop an SSD for the *<u>Ship items</u>* use case that is shown as a fully developed use case description:



Use case name:	*Ship items.*	
Scenario:	Ship items for a new sale.	
Triggering event:	Shipping is notified of a new sale to be shipped.	
Brief description:	Shipping retrieves sale details, finds each item and records it is shipped, records which items are not available, and sends shipment.	
Actors:	Shipping clerk.	
Related use cases	None.	
Stakeholders:	Sales, Marketing, Shipping, warehouse manager.	
Preconditions:	Customer and address must exist. Sale must exist. Sale items must exist.	
Postconditions:	Shipment is created and associated with shipper. Shipped sale items are updated as shipped and associated with the shipment. Unshipped items are marked as on back order. Shipping label is verified and produced.	
Flow of activities:	**Actor**	**System**
	1. Shipping requests sale and sale item information.	1.1 System looks up sale and returns customer, address, sale, and sales item information.
	2. Shipping assigns shipper.	2.1 System creates shipment and associates it with the shipper.
	3. For each available item, shipping records item is shipped.	3.1 System updates sale item as shipped and associates it with shipment.
	4. For each unavailable item, shipping records back order.	4.1 System updates sale item as on back order.
	5. Shipping requests shipping label supplying package size and weight.	5.1 System produces shipping label for shipment. 5.2 System records shipment cost.
Exception conditions:	2.1 Shipper is not available to that location, so select another. 3.1 If order item is damaged, get new item and updated item quantity. 3.1 If item bar code isn't scanning, shipping must enter bar code manually. 5.1 If printing label isn't printing correctly, the label must be addressed manually.	

CHAPTER 5: Extending the Requirements Models

Answer: SSD for the Use Case *Ship Items*



Detailed explanation:

- The actor has 5 numbered steps in the flow of activities, therefore there are 5 inputs into the system.
    1. getNextSale - No parameter is needed for getNextSale - the system will return information for the next sale to be shipped. Parameter value is blank i.e. ().
    2. setShipper – selected from a list – use **shipperID**
    3. recordShippedItem – Loops until all items are recorded as shipped
    4. recordBackorder – Loops until all items are unavailable items are recorded as backorders
    5. getShippingLabel – request shipping label supplying parameters package size and weight. The system uses that information, along with the shipper and address, to produce the shipping label and record the cost back to the actor.

ICT2622 Object-Oriented Analysis Notes Phase Ch 5 – Summary Notes
SATZINGER JACKSON BURD (6TH EDITION)

CHAPTER 5: Extending the Requirements Models

THE STATE MACHINE DIAGRAM — IDENTIFYING OBJECT BEHAVIOUR

- When defining requirements, analysts need to identify and document which domain objects require status checking and which business rules determine valid status conditions e.g. a business rule is that a customer sale shouldn't be shipped until it has been paid for
- **State** - a condition during an object's life when it satisfies some criterion, performs some action, or waits for an event
- States are described as semi-permanent conditions because external events can interrupt a state and cause the object to transition to a new state.
- A transition is the movement of an object from one state to another state.
- **State machine diagram** - a diagram showing the life of an object in states and transitions
  - Can be developed for any problem domain classes that have complex behaviour or status conditions that need to be tracked
  - Is composed of ovals representing the states of an object and arrows representing the transitions
  - Starting point of a state machine diagram is a black dot, which is called a **pseudostate**.
  - The first shape after the black dot is the first state of the object
  - The arrow leaving the first state (called the **origin**) is called a transition and it links to another state (**destination**) on completion of the event
  - The transition arrow is labelled with a string to describe its components
  - The transition-name is the name of a message event that triggers the transition and causes the object to leave the origin state.
  - The format is very similar to a message in an SSD.

- The transition label consists of the following syntax with three components:

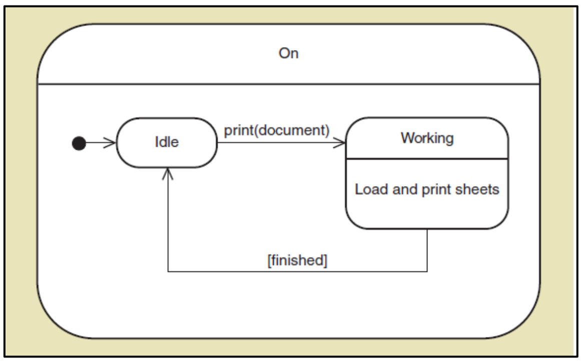
  transition-name (parameters, …) [guard-condition] / action-expression

- The guard-condition is a qualifier or test on the transition, and it is simply a true/false condition that must be satisfied before the transition can fire.
- For a transition to fire, first the trigger must occur and then the guard must evaluate to true.
- System sequence diagrams messages have a similar test, which is called a ***true/false condition***. This true/false condition is a test on the ***<u>sending</u>*** side of the message, and before a message can be sent, the true/false condition must be true
- In contrast, the guard-condition is on the ***<u>receiving side</u>*** of the message. The message may be received, but the transition fires only if the guard-condition is also true.

Composite States and Concurrency

- The condition of being in more than one state at a time is called **concurrency**, or "concurrent states"
- Can be shown using a synchronization bar and concurrent paths, as in an activity diagrams
- A transition could be split with a synchronization bar so one path goes to the **On state** and the other path goes to the **Idle, Printing, and Selfcheck states**.
- A **path** is a sequential set of states and transitions.
- Another way to show concurrent states is to have states nested inside other, higher-level states called **composite states**

SSD for a Printer with concurrency



CHAPTER 5: Extending the Requirements Models

Composite state contains nested states and transition paths

How to develop a state machine diagram

- The primary challenge in building a state machine diagrams:
  1. To identify the right states for the object
  2. To identify and handle composite states with nested threads
- Developing state machine diagrams is an iterative behaviour

Rules for Developing State Machine Diagrams

1. Review the class diagram and select the classes that might require state machine diagrams
   - Include only those classes that have multiple status conditions that are important for the system to track
   - Then begin with the classes that appear to have the simplest state machine diagrams

2. For each selected class in the group, make a list of all the status conditions you can identify - brainstorm session with the whole team
   - The states must reflect the states for the real-world objects that will be represented in software

3. Begin building state machine diagram fragments by identifying the transitions that cause an object to leave the identified state

4. Sequence these state-transition combinations in the correct order — then aggregate these combinations into larger fragments

5. Review the paths and look for independent, concurrent paths

6. Look for additional transitions

7. Expand each transition with the appropriate message event, guardcondition, and action-expression

8. Review and test each state machine diagram