Use Case Modeling

Lecture 12, v02

John G. Artus

BSEE

MSSE

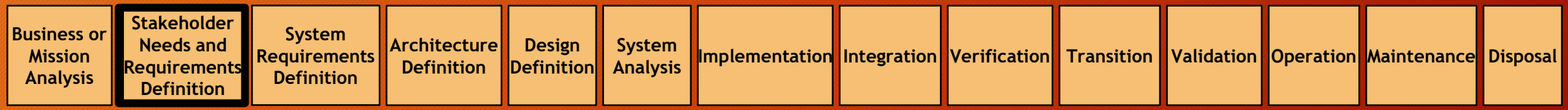
INCOSE ESEP

About This Courseware

- The majority of the material presented in this course is sourced from the textbook “Use Case Modeling” by Kurt Bittner and Ian Spence
- I, John Artus, make no claim of ownership of the material sourced from this textbook
- I, John Artus, am using the material sourced from this textbook, and other indicated sources, as content for this courseware for educational purposes only
- This courseware lecture material has been sourced, interpreted, assembled, formatted, and copyrighted by John G. Artus for use in this educational context
- Anyone may freely access, and reuse this material in an educational context provided the copyright owner, John G. Artus, is recognized as the interpreter, assembler, and formatter of the source material used in the generation of this courseware, and provided that Kurt Bittner and Ian Spence are recognized as authors of the textbook “Use Case Modeling” from which the majority of the content of this courseware has been sourced

What is Use Case Modeling?

- Use case modeling is the development of models of a system, defined in terms of use cases, actors, and the associations between them that describes the requirements of a system having to do with what a user of the system (a kind of stakeholder) wants the system to do for them
- Use case modeling is typically done in the very early stages of a program when establishing what is the problem that stakeholders want solved, specifically with regard to how stakeholders intend on using the system to derive benefits from the system
- Use case modeling is accomplished primarily during the Stakeholder Needs and Requirements stage of the SE life cycle



The 14 INCOSE Systems Engineering Technical Processes

- Use case modeling provides an effective methodology for establishing stakeholder needs regarding use of the system in order to facilitate the development of system requirements in this area
- The focus of information gathering for the purpose of developing system requirements regarding use of the system is on
  - What the user wants the system to do for them and
  - How the user intends on interacting with the system

Goals of Use Case Modeling

The goals of developing and modeling use cases are

- Work with stakeholders to solicit their ideas on what they want the system to do for them (the "whats")
- Ensure that everyone has the same conceptual model (or mental picture) of what the system will do
- Avoid specifying how the system will accomplish the "whats"
  - These are implementation details that, if defined too early, can limit the range of alternative solution concepts, possibility eliminating winning solutions
- Ensure that the use-case models facilitate effective communications with stakeholders
  - The audience for the set of use case models is the entire stakeholder community
- Pursue simplicity in model development
  - Make them as simple and straightforward as possible
- Good enough is as good as it gets
  - There is no such thing as perfection in use-case modeling
  - Do a "good" job at capturing stakeholder system usage needs and move on
- Do a complete job at documenting the use case
  - You will not have a better time to document the use case in a narrative form than the present
  - There is no way to avoid this and doing so will be very painful to recover later

Who does Use Case Modeling?

- This is actually a good question, is subject to modeling tool technology, and is likely to change over time
- Who CURRENTLY performs use case modeling?
  - Since requirements management tools (today, for the most part) DO NOT provide the means for use case modeling
  - And since architecture modeling tools (today, for the most part) DO provide the means for use case modeling (using SysML)
  - It is generally the job of the architect to perform use case modeling and create use case models
- Who SHOULD BE performing use case modeling?
  - Since use case modeling is part of establishing stakeholder needs and stakeholder requirements,
  - This SHOULD BE the job of requirements engineers
- Unfortunately, this landscape is driven for the most part by tool vendors
  - However, the landscape is changing
  - Architecture modeling tools are slowly become System modeling tools
- Having one system modeling tool serve the needs of both requirements engineers and architects will have several benefits
  - Both teams
    - Use the same tool
    - Access the same data
    - Develop a single, unified model
  - Which means that requirements engineers can focus on installing stakeholder/system requirements and use cases into the system model
  - While architect use that same data in the system model as inputs to the architecture development process

Use Case Modeling Terminology

- Use Case
  - Describes a main usage of the System of Interest (SoI) by the user in order for the user to derive some benefit from the SoI
  - These are stories about how the actors use the system to do something they consider important
- Use Case Model
  - A model of a SoI, defined in terms of use cases, actors, and the associations between them
  - Describes the requirements of the SoI having to do with what a user of the SoI (a kind of stakeholder) wants the SoI to do for them
  - A use case model can contain, and is often represented by, a set of use-case diagrams
- Use Case Description
  - A textual description of the properties of a use case
  - Primarily in the form of a sequential flow of events describing the interaction between the user and the SoI
- Actor
  - Defines a role that a user can play while interacting with the SoI
  - More than one actor can participate in a use case
  - An actor can be a person or an external system (not part of the SoI)
  - A single person can fulfill multiple actor roles, but each actor role is distinct in use case modeling
  - One of the actors initiates the use case
- User
  - A person or external system that uses the SoI (an actor)

Actors communicate with the system for many reasons, including:

- To start a use case
  - Use cases are always started by actors
- To ask for some data stored in the system, which the use case then presents to the actor
- To change the data stored in the system by means of a dialog with the system
- To report that something special has happened in the system's surroundings that the system should be aware of

User Role Versus Operator Role

- System of Interest (SOI)
  - The system being designed/developed
- Actor
  - A general term to describe anyone or anything that interacts with the system
  - Can be a person
  - Can be an external system
  - Can be dual role (user & operator)
- User
  - Generally meant to be the primary beneficiary of the primary function of the system
  - Does NOT provide any functionality of the system
  - Example: ATM Customer
- Operator
  - Provides system functionality that is too expensive or impractical to be provided by a system component
  - Example: Aircraft Pilot

Cotton Candy Production System

Which is the …?

- **SOI**
- **Operator**
- **User (Customer)**

Perspective 1

"I want to ***BUY*** Cotton Candy"

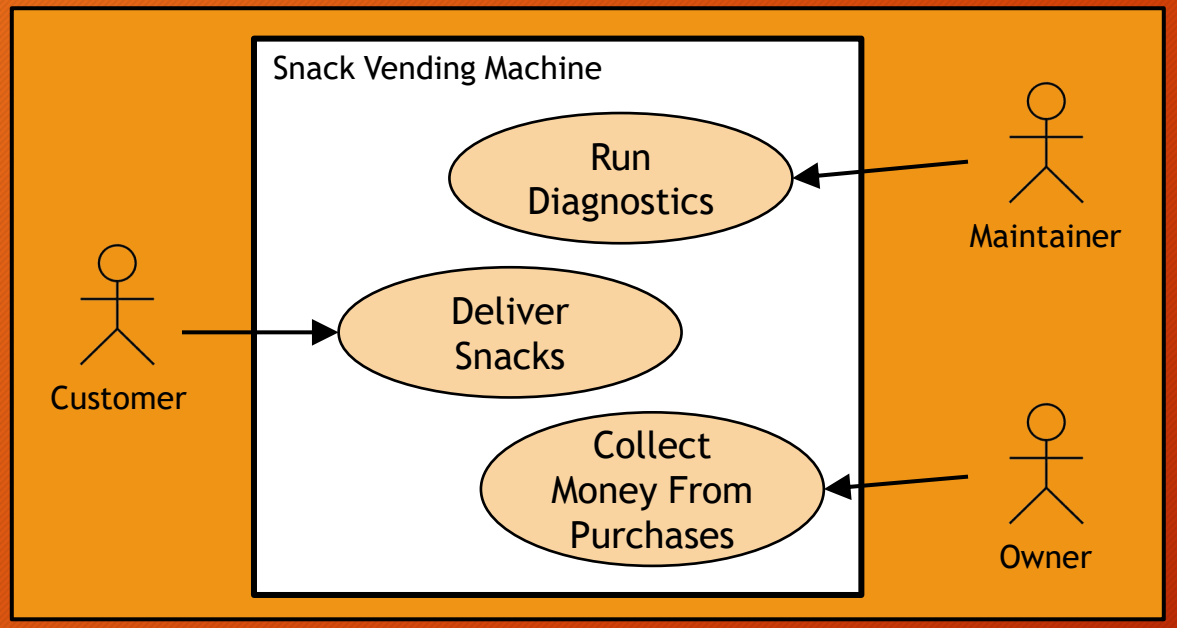
Perspective 2

"I want to ***MAKE*** Cotton Candy"

Image: https://www.agefotostock.com/age/en/Stock-Images/Rights-Managed/G14-1853781

Use Case Diagram Versus Use Case Description

- Use cases can be depicted in diagrams, but they are principally described in text
  - Too much focus on the diagrams often leads engineers down the path of use case decomposition
  - Each use case shows up as an ellipse on the diagram
  - The ellipse is just an iconic placeholder for a description of how the system and its actors interact
  - The use-case diagram can be thought of as a visual aid to comprehension but does not tell the whole story
- Use cases are used to build up our understanding of what the system should do
  - They are not used to analyze or break down the requirements into smaller parts; that is the job of system analysis and design performed in the System Architecture Definition process
  - A real usable content of a use case is the narrative text that describes what the system does for a particular actor



Example Use Case Diagram

UC-0015	Register Book Loan	
Dependencies	• OBJ–0001 *To manage book loans* (objective) • OBJ–0005 *To know library users' preferences* (objective) • CRQ–0003 *Maximum number of simultaneous loans* (business rule) • CRQ–0014 *Return date for a loan* (business rule)	
Description	The system shall behave as described in the following use case when *a library user requests a loan of one or more books.*	
Precondition	*The library user has been identified by means of his or her identity card, has picked up the books to loan from the shelves, has not reached the maximum number of simultaneous loans and has no penalty.*	
Ordinary Sequence	Step	Action
	1	Actor *librarian requests the system for starting the book loan registering process.*
	2	The system *requests for the identification of the library user requesting a loan.*
	3	Actor *librarian provides identification data of the library user to the system.*
	4	The system *requests for the identification of the books to be loaned.*
	5	Actor *librarian provides identification data of the books to be loan to the system.*
	6	The system *displays the return date for each of the books to be loan and requests loan confirmation for each of them.*
	7	Actor *library user confirms the librarian which books he or she wants to loan after knowing return dates.*
	8	Actor *librarian re–confirms the book loans confirmed by the library user to the system.*
	9	The system *informs that the book loans have been successfully registered.*
Postcondition	*The library user can take the loaned books away and the system has registered the book loans.*	
Exceptions	Step	Action
	3	*If the library user has already reached the maximum number of simultaneous loans or has a penalty,* the system *informs of the situation,* then this use case *is cancelled.*
Comments	*The maximum number of simultaneous book loans and the loan period depend on the library policy and can change in the future. See business rules CRQ–0003 y CRQ– 0014.*	

Example Use Case Description

https://www.researchgate.net/publication/220536177_Empirical_Evaluation_and_Review_of_a_Metrics-Based_Approach_for_Use_Case_Verification

Example Use Case

- Use cases are expressed from the perspective:
  - "As a user of the SoI, I want to use the system to (insert use case name here)"
- We will use a Snack Vending Machine (electro-mechanical type) as a simple representation of a SoI
  - This is so we can focus on the discussion point without getting lost in the details of a more complicated system

Use Case Name	Description	Valid Use Case?	Why or Why Not
Deposit Coins	The user deposits coin(s) into the coin slot to initiate the interaction	No	This is not the main purpose for using the SoI
Select Snack Item	The user selects the item to be delivered by the SoI	No	This is not the main purpose for using the SoI
Extract Snack From Bin	The user extracts the delivered snack item from the extraction bin	No	This is not the main purpose for using the SoI
Deliver Snacks	The user purchases a snack from the SoI, and wants the SoI to deliver the snack	Yes	This is a main objective of the user Anything else is detail to be placed into the use case description

"As a user of the SoI,
I want to use the system to deliver snacks"

Use cases are drawn graphically as ovals

Making Use of Use Case Models

- As Systems Engineers, we do not create models just for the sake of creating models
  - Use case models are used to generate stakeholder requirements that are then transformed onto system requirements
- The kinds of information gleaned from use case models includes (but is not limited to)
  - Desired “features” to be made into system capabilities
  - Defined sequences of interaction to be made into functional requirements describing observable system behavior
  - Defined system qualities and constraints to be made into non-functional requirements such as performance requirements and quality requirements (“ilities”)
  - Events that are expected to be raised by actors and by the system
  - System responses to actor events
  - The information exchanged between actors and the system
- While use case models are valuable tools for eliciting stakeholder needs and requirements, they cannot and do not capture ALL requirements
  - However, they are capable of capturing the bulk of the behavior-related requirements at the user level
- Use case models are very useful for visualizing how the users (including external systems) and the SoI will interact
  - They can be used in reviews with the customer to ensure a joint understanding of the intended system behavior with users before design activities move forward

The Problem of Naming Use Cases

Why is this a Problem?

- How use cases are named implies the perspective from which the use case is viewed
- The two perspectives are:
  - The perspective of the user taking some action involving the system

From the user's perspective,
the use case here is
"Choose Product to Purchase"

  - The perspective of the system performing some function for the benefit of the user

From the system's perspective,
the use case here is
"Accept Product Selection Choice" from the user

Either perspective is valid

What is important (and can be a problem) is that the architecture team, when developing use cases, be consistent in using the same perspective throughout the entire system design

Me recommendation would be to use the System Perspective, such that use cases are named in a manner in which they can be viewed as "functions that the system needs to perform"

Use Case Descriptions

Use Case Description

- A text-based description of the use case properties and sequential flow of events in the interaction between a user and the SoI while performing a use case
- For every use case, there must be an accompanying use case description
- The basic contents of a use case description are:
  - Use Case Name – A short title for the use case
  - Use Case ID – A tracking number
  - Use Case Brief Description – A paragraph long summary of the use case
  - Primary Actor – The actor principally involved in this use case
  - Secondary Actor(s) – Any other actors involved in this use case
  - Preconditions - A statement of the (optional) condition(s) that must exist in order for the use case to be performed when triggered
  - Trigger – An event that causes the use case to be started (Providing preconditions are met)
  - Basic Flow - Sequential steps involved in the primary actor interaction with the SoI
  - Alternate Flows – Optional flow paths off of the basic flow path that the use case could take, based on events that occur
  - Exception Flows - Optional flow paths off of the basic flow path that the use case could take, based on faults that occur
  - Postconditions - A statement of the (optional) condition(s) that must exist when the use case ends in order for the use case to be considered to have completed successfully
  - Frequency of Use – A specification or estimation as to how often it is expected the use case to be triggered during system use

Use Case Model Development Approach

1. Understand the stakeholder's problem
2. Define the boundary of the system
   a) Anything inside the system boundary is the responsibility of the developer
   b) Anything outside the system boundary is not the responsibility of the developer
   c) Interfaces that cross the system boundary require the developer to participate in the definition of the interface with the actor/system on the other end of the interface
3. Identify all the actors that will have direct contact with (use of) the system
4. Identify what are the principle uses of the system that the actors are interested in (the use cases)
5. For each use case, develop a complete use case description (FIRST)
   a) The use case description will describe the sequential flow of interaction between the actor(s) involved in the use case and the SoI
   b) There can be alternative flows, depending on options the user can select, or based on exceptions to the normal flow
      i. These will be discussed later in this lecture material
   c) The use case description for real systems can be quite long and elaborate – this is the real work of developing a use case model
6. (THEN, and only then) Develop a use case diagram, based on the use case description

Use case	
Use Case Name:	Order Meal
Actors:	Customer
Description:	A customer accesses the Meal Delivery System using his mobile phone, searches for a specific restaurant, selects food items, and places an order for a meal to be delivered to a specified location.
Preconditions:	1. Customer is logged into Meal Delivery System. 2. Customer already found the intended restaurant.
Postconditions:	1. Meal order is stored in the system with a status of "accepted".
Normal Flow:	**1.0 Order a Meal** 1. Customer asks to view the menu. 2. System displays menu of available food items. 3. Customer selects one or more food items from menu. 4. Customer indicates that meal order is complete. 5. System displays ordered menu items, individual prices, and total price and delivery charge. 6. Customer confirms meal order or requests to modify meal order (back to step 3). 7. Customer specifies payment method. 8. System confirms acceptance of the order. 9. System stores order in database and notify the restaurant Cafeteria Staff, sending the food item information.
Alternative Flows:	**1.1 Order multiple meals** (branch after step 4) 1. Customer asks to order another meal. 2. Return to step 2.

http://match.inf.ufsc.br:90/upcase/examples.html

A very simple use case description example

Example Use Case Description

- There is no standard format for use case descriptions
- There are many use case description templates and examples available on the internet

UC-0015	Register Book Loan	
Dependencies	• OBJ–0001 *To manage book loans* (objective) • OBJ–0005 *To know library users' preferences* (objective) • CRQ–0003 *Maximum number of simultaneous loans* (business rule) • CRQ–0014 *Return date for a loan* (business rule)	
Description	The system shall behave as described in the following use case when *a library user requests a loan of one or more books.*	
Precondition	*The library user has been identified by means of his or her identity card, has picked up the books to loan from the shelves, has not reached the maximum number of simultaneous loans and has no penalty.*	
Ordinary Sequence	Step	Action
	1	Actor *librarian requests the system for starting the book loan registering process.*
	2	The system *requests for the identification of the library user requesting a loan.*
	3	Actor *librarian provides identification data of the library user to the system.*
	4	The system *requests for the identification of the books to be loaned.*
	5	Actor *librarian provides identification data of the books to be loan to the system.*
	6	The system *displays the return date for each of the books to be loan and requests loan confirmation for each of them.*
	7	Actor *library user confirms the librarian which books he or she wants to loan after knowing return dates.*
	8	Actor *librarian re–confirms the book loans confirmed by the library user to the system.*
	9	The system *informs that the book loans have been successfully registered.*
Postcondition	*The library user can take the loaned books away and the system has registered the book loans.*	
Exceptions	Step	Action
	3	*If the library user has already reached the maximum number of simultaneous loans or has a penalty*, the system *informs of the situation*, then this use case *is cancelled.*
Comments	*The maximum number of simultaneous book loans and the loan period depend on the library policy and can change in the future. See business rules CRQ–0003 y CRQ– 0014.*	

https://www.researchgate.net/publication/220536177_Empirical_Evaluation_and_Review_of_a_Metrics-Based_Approach_for_Use_Case_Verification

Name	Save item for purchase.
ID	UC_001
Description	While browsing items in the eStore, a user finds an item he is not ready to purchase yet, but he wants to save it to a list so that he can later find the item that he was previously interested in.
Actors	eStore customer.
Organizational Benefits	Increase sales by helping the customer remember products he was previously interested in.
Frequency of Use	20% of users save an item to be bought later each time they visit the site. 50% of saved items are purchased within one year of the saved date.
Triggers	The user selects an option to save an item.
Preconditions	User is viewing an item in the catalog.
Postconditions	The item selected to be saved is visible to the user when he views his saved items. The item selected to be saved is reflected as a saved item when the user views his eStore search and browse results.
Main Course	1. System prompts user to confirm saving selected item instead of purchasing it right away. 2. User confirms to save now (see EX1). 3. System determines user is not logged in and redirects user to log on (see AC1). 4. User logs on (see AC2, AC3). 5. System stores the saved item (see EX2). 6. System redirects the user to their saved items list to view the full list.
Alternate Courses	AC1 System determines user is already logged on. 1. Return to Main Course step 5. AC2 User logs off again. 1. Return user to Main Course step 3. AC3 User does not have an account already. 1. User creates an account. 2. System confirms account creation. 3. Return user to Main Course step 4.
Exceptions	EX1 User decides to purchase the item now. 1. See "Purchase item" Use Case. EX2 System fails on saving item to list. 1. System notifies user that an error has occurred. 2. Return user to Main Course step 1.

https://kelvin.ink/2018/10/08/OOAnalysisUML1/

Use Case Description “Flows”

- The sequential steps of an interaction between an actor and the SoI constitute a “flow”
- These flows are described in the use case description text
- Flows are categorized as
  - Basic flow – covers what “normally” happens during the successful implementation of a use case
  - Subflows – a decomposition of the basic flow for the purpose of making it easier to follow the basic flow
    - Subflows are NOT child use cases
    - They are simply a mechanism to break up a long use case basic flow description into more digestible parts for the reader to be able to understand the flow more easily
    - Subflows should be “atomic” – either all or none of the actions described in the subflow are performed
    - Unless absolutely necessary, avoid constructing several levels of sub-subflows
  - Alternative flows – covers behavior that is optional, exceptional, or an alternative to the basic flow
    - The implementation of an alternative flow is the result of some condition or event triggering the alternative flow
    - Under such conditions, the use of alternative flows is preferred to creating subflows because alternative flows represent a defined, meaningful subset of functionality that helps manage scope of the system functionality
    - This way, alternative flows can be easily created, modified, or eliminated without affecting the basic flow or other alternative flows
    - You cannot create, modify, or eliminate subflows without directly affecting the nature of the basic flow

Example Basic Flow

The example uses an extraction of (not all of) a use case titled “Browse Products and Place Orders” for an online shopping app

Basic Flow

1. The use case starts when the actor **Customer** selects to browse the catalog of product offerings.
2. The **System** displays the product offerings highlighting the product categories associated with the Customer’s profile.
3. The **Customer** selects a product to be purchased, entering the number of items required.
4. For each selected item that is in stock, the **System** records the product identifier and the number of items required, reserving them in inventory and adding them to the Customer’s shopping cart.
5. Steps 3 and 4 are repeated until the **Customer** selects to order the products.
6. The **System** prompts the Customer to enter payment instructions.
7. The **Customer** enters the payment instructions.
8. The **System** captures the payment instructions using a secure protocol.

... (remainder of flow not shown for brevity)

There are many different proposed methods for documenting use case flows
This is one method suggested by the book “Use Case Modeling” by K. Bittner and I. Spence

Example Subflow

- This modification to the example includes one subflow – which directs the basic flow to branch off and perform the subflow before returning back to the basic flow
- Subflows can be useful when the same behavior appears more than once in a single use case

Basic Flow

1. The use case starts when the actor **Customer** selects to browse the catalog of product offerings.

2. The **System** displays the product offerings highlighting the product categories associated with the Customer's profile.

3. The **Customer** selects a product to be purchased, entering the number of items required.

4. For each selected item that is in stock, the **System** records the product identifier and the number of items required, reserving them in inventory and adding them to the Customer's shopping cart.

5. Steps 3 and 4 are repeated until the **Customer** selects to order the products.

6. The **System** prompts the Customer to enter payment instructions.

7. The **Customer** enters the payment instructions.

8. The **System** captures the payment instructions using a secure protocol.

9. Perform Subflow **S1: Validate Payment Instructions**. **SUBFLOW**

... (remainder of flow not shown for brevity)

Subflow Validate Payment Instructions

1. The **System** contacts the Paying Institution for validation of Customer payment information.

2. The **System** transmits the Customer payment information to the paying institution.

3. The **Paying Institution** returns a validation code that indicates whether the Customer's payment information is valid or not.

4. The **System** informs the Customer whether the payment information has been validated or not.

... (remainder of flow not shown for brevity)

There are many different proposed methods for documenting use case flows
This is one method suggested by the book "Use Case Modeling" by K. Bittner and I. Spence

It is important to understand that the Basic Flow and the Subflow shown in this example would be part of **the same use case**

Extension Points

- The example uses “extension points” which are simply named places within the flow to indicate where additional behavior can be inserted or attached
- Extension points can be used within the description of alternative flows to indicate where the alternative flow are inserted into the basic flow

Basic Flow

1. The use case starts when the actor **Customer** selects to browse the catalog of product offerings.

{Display Product Catalog} **EXTENSION POINT**

2. The **System** displays the product offerings highlighting the product categories associated with the Customer’s profile.

{Accept Product Selection} **EXTENSION POINT**

3. The **Customer** selects a product to be purchased, entering the number of items required.

4. For each selected item that is in stock, the **System** records the product identifier and the number of items required, reserving them in inventory and adding them to the Customer’s shopping cart.

{Out of Stock Encountered} **EXTENSION POINT**

5. Steps 3 and 4 are repeated until the **Customer** selects to order the products.

{Process the Order} **EXTENSION POINT**

6. The **System** prompts the Customer to enter payment instructions.

7. The **Customer** enters the payment instructions.

8. The **System** captures the payment instructions using a secure protocol.

9. Perform Subflow **S1: Validate Payment Instructions**.

... (remainder of flow not shown for brevity)

Three of these extension points are used simply as topic headings
- **{Display Product Catalog}**
- **{Accept Product Selection}**
- **{Process the Order}**

One of these extension points is used to reflect the state of the use case
- **{Out of Stock Encountered}**

There are many different proposed methods for documenting use case flows
This is one method suggested by the book “Use Case Modeling” by K. Bittner and I. Spence

Handling a Specific Alternative Flow

- It is worth repeating that the reason for breaking out alternative flows specifically is so that we can manage the scope of the system
  - We want to be able to remove subsets of functionality (the alternative flows) without breaking the system and/or failing to provide the expected level of value to the stakeholders
  - If we consider system behavior only at the level of single, monolithic use cases, we will not be able provide use case scope management since use cases will tend to be large and relatively indivisible
  - Alternative flows generally represent optional behavior that is outside of the normally expected system behavior, and therefore can be easily added, modified, or removed without impact to the basic flow

Alternative Flow: A3 Handle Product Out of Stock

At {Out of Stock Encountered} if there are insufficient amounts of the product in the inventory to fulfill the Customer's request, then,

The System informs the Customer that the order cannot be fulfilled.

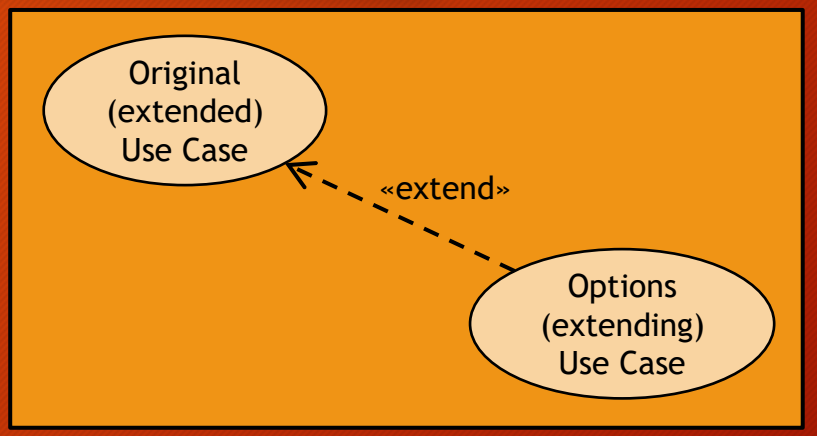
... the flow continues to describe the offering of alternative amounts and products to the Customer ...

The flow of events returns to the basic flow and is resumed from the point at which it was interrupted.

There are many different proposed methods for documenting use case flows
This is one method suggested by the book "Use Case Modeling" by K. Bittner and I. Spence

It is important to understand that the Basic Flow and the Alternative Flow shown in this example would be part of **the same use case**

Transforming an Alternate Flow into an Extended Use Case



- The original intent of use case extension was to specify options that could be added with minimal effect on the existing product
  - So, the «extend» relationship should be thought of as adding behavior contained in the extended use case to an existing use case
  - The extended use case must be able to stand alone (be independent of the base use case)
  - This means that the base use case requires no changes in order to use the extended behavior of the extended use case
- Certain situations can take advantage of extended use cases
  - To describe behavioral features that are optional to the basic behavior of the system
  - To describe complex error- or exception-handling behavior that would otherwise complicate understanding of the basic behavior of the system
  - To describe special handling situations of a particular customer that uses the system in a unique way
  - To control releases of new system behavior that will not be introduced until a later release
- Conceptually, an extending use case works the same way as an alternative flow
  - An extending use case explicitly inserts itself into the flows of the use case it is extending, just like an alternative flow
  - An extending use case knows exactly where in the base use case the behavior is to be inserted
  - For this reason, an extending use case often begins life as an alternative flow
- While it is possible to transform an alternate flow into an extended use case, there are several considerations that the use case developer should contemplate
  - Because alternative flows are actually part of a use case, they can exploit their knowledge of the use case's state, preconditions, and other flow events to end the use case or resume the flow of the use case at the extension points other than the one from which they assumed control
  - As opposed to alternative flows which are aware of the state of the base flow, extending use cases are not aware of the state of the base use case
  - All extending use cases know about is the extension point at which they insert themselves into the flow of events of the use case that they extend

Example of use of Extending Use Cases

Extended Use Case

There are many different proposed methods for documenting use case flows
This is one method suggested by the book "Use Case Modeling" by K. Bittner and I. Spence

Use Case: Process Transactions

Basic Flow

1. The use case starts when the actor **Cashier** initiates transaction processing for a set of unprocessed transactions.

2. The **System** orders all the transactions so that all transactions for a particular account are grouped together, and within this grouping the deposit transactions are processed first to avoid unnecessary overdraft processing.

3. For each account:

    {Determine Customer Account}

    a. The **System** determines the customer account to which the transaction is applied.

    b. For each transaction:

        {Apply Transaction}

        ... (part of this flow not shown for brevity)

        {Record Transaction}

        ... (part of this flow not shown for brevity)

        {Summarize Transaction}

        i. When all transactions for a particular account have been processed, the **System** creates a transaction summary for the account.

4. When all transactions have been processed, the use case ends.

Extending Use Case

Use Case: Notify of Overdraft

Extension

Extends use case **Process Transactions** at **{Summarize Transactions}** if the customer has purchased the overdraft notification service and the set of completed transactions has caused the account to become overdrawn.

Basic Flow

1. The **System** determines the customer's preferred notification mechanism, as recorded in the customer profile.

2. The **System** composes the overdraft notification, providing the transaction information, the date and time the transaction was processed, the account information, the balance prior to the transaction, the balance subsequent to the transaction, and the amount of the overdraft fee, if any.

3. The **System** transmits the overdraft notification message to the customer using the customer's preferred notification mechanism.

4. The use case ends.

It is important to understand that the flow associated with the Extending use case is that of a **separate use case**

Sharing Common Behavior among Multiple Use Cases

- When common behavior is shared among multiple use cases, the common behavior can be culled out to form a separate use case
- The example uses the «include» relationship between two use cases that share common behavior
- It is important to ensure that the common behavior can be culled out completely and that the new included use case is totally independent of the use cases it is included into
- Included use cases are different from Subflows because Subflows are completely "aware" of the state of the base flow, whereas included use cases are clueless as to the state of the use case they are included into

Use Case: Answer Customer Inquiries — Including Use Case

1. The use case starts when the actor **Customer** calls the Customer Service Center Support number.

...

8. The **System** determines whether the Customer's personal information is already on file.

9. If it is, then the **Customer** uses the customer identification number to determine whether there have been any prior calls placed by this Customer; if prior calls have been made, the **System** records references to the prior call information in the Customer Service Request.

10. If it is not, then **«include»** use case **Add Customer Information** so the Customer Service Representative can record information for this customer.

...

Use Case: Order Products — Including Use Case

1. The use case starts when the actor **Sales Representative** selects the menu option to place an order.

2. The **System** asks the Sales Representative to enter the Customer's customer identification number.

3. If the Customer is a new customer, **«include»** use case **Add Customer Information** so the Sales Representative can record information for this Customer.

...

Here, "User" is used to serve as a surrogate for "Customer" and Sales Representative" in the including use cases

Included Use Case

Use Case: Add Customer Information

1. The **System** prompts the **User** to enter the customer information.

2. When adding or modifying the customer information:

   a. When the state or province is entered or changed, the **System** checks to see if the state or province is valid for the country entered.

   b. When the postal code is entered or changed, the **System** checks to see if it is valid for the country and state or province indicated.

3. The use case ends when the additions or changes to the customer information are saved, or the additions or changes are aborted.

There are many different proposed methods for documenting use case flows
This is one method suggested by the book "Use Case Modeling" by K. Bittner and I. Spence

It is important to understand that the flow associated with the Included use case is that of a **separate use case**

Use Case Diagrams

First and Foremost ...

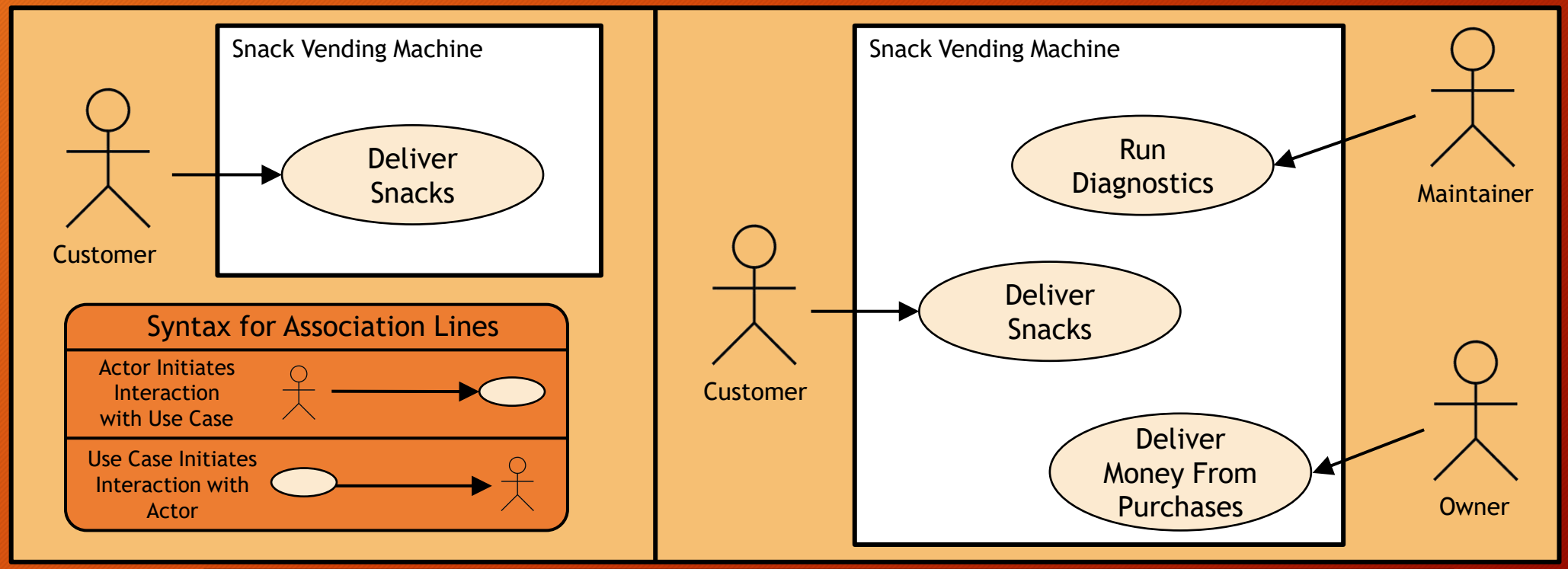
- Use case diagrams are intended to ***SUMMARIZE*** a use case description
  - Such summaries are useful in a review with the customer, for them to understand the overall usage of the system from the user perspective
  - And what the system will be designed to do for the user
  - The use case descriptions should be developed first and should provide the details at reviews, if those details are needed
- A use case diagram alone provides insufficient description of the use case

WARNING

- You are likely to encounter many situations were only use case diagrams are provided
- And, where those use case diagrams are very POORLY developed
- This is due to the lack of proper education on the part of the use case modelers who do not understand what is the purpose of developing use cases and who the audience is
  - Many uneducated modelers see use case diagrams as detailed design artifacts and try to design the whole system with them – THIS IS NOT ONLY INCORRECT AS A MODELING PRACTICE, IT IS ALSO AN IMPOSSIBLE TASK
  - Use Case development is an early task that is done to scope out the user interaction with the system and to ensure that you got that part well understood before proceeding with the rest of the system development
  - On-The-Job Suggestion: Discourage, as much as you can, the tendency for engineers/modelers to “design the system” using use cases

Use Case Diagram

- In Systems Engineering, we use the UML/SysML standard Use Case Diagram for representing use cases
  - A basic use case diagram depicts the following
    - System Boundary – a box that indicates the boundary separating the system contents from the external environment
    - Use case(s) – one or more ovals indicating the uses cases involved
    - Actors – user roles or external systems that interact with the SoI
    - Associations – lines between actors and use cases indicating the relationship between the two
  - Beyond these basics there a several additional features of a use case diagram, which will be covered further on in this lecture
- Generally, there could be several actors and several use cases



A use case diagram is created from the user story presented in the use case description

Actor Associations to Use Cases

Actors communicate with the system for many reasons, for example ...

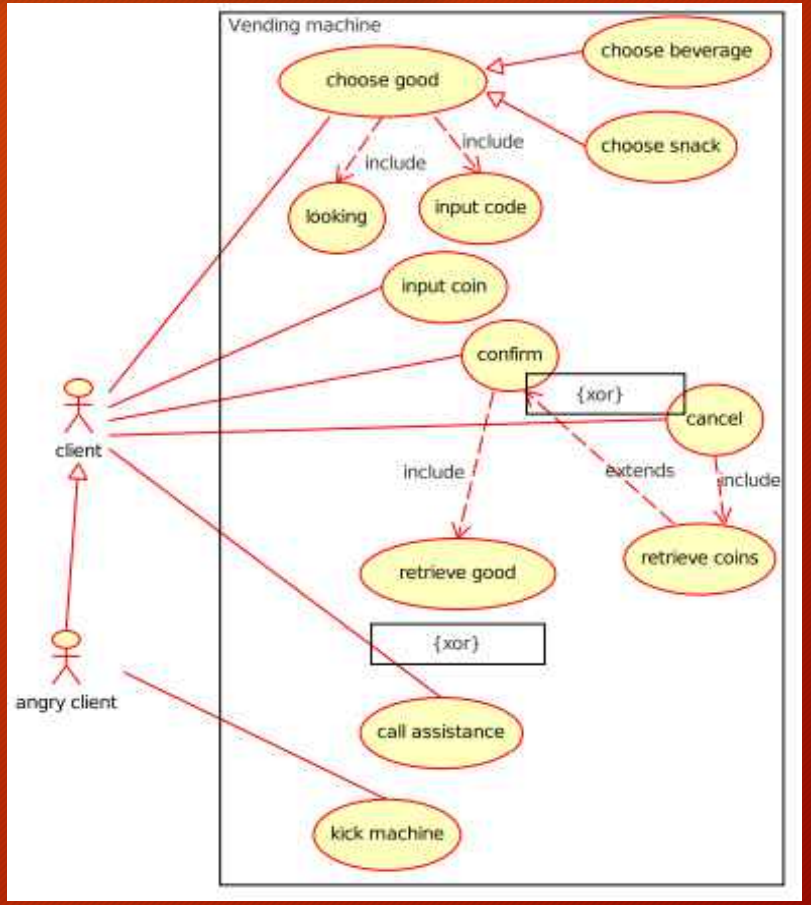
- To start a use case
  - Use cases are always started by actors
- To ask for some data stored in the system, which the use case then presents to the actor
- To change the data stored in the system by means of a dialog with the system
- To report that something special has happened in the system's surroundings that the system should be aware of

When diagraming actor/use case associations, certain rules need to be followed

- A use case has at most one communicate association to a specific actor
- An actor has at most one communicate association to a specific use case
- These rules hold no matter how many interactions there are
- The arrowhead is optional, but where it is used, it indicates which element starts the interaction
  - The initiator is at the tail end of the line

To Decompose or Not To Decompose

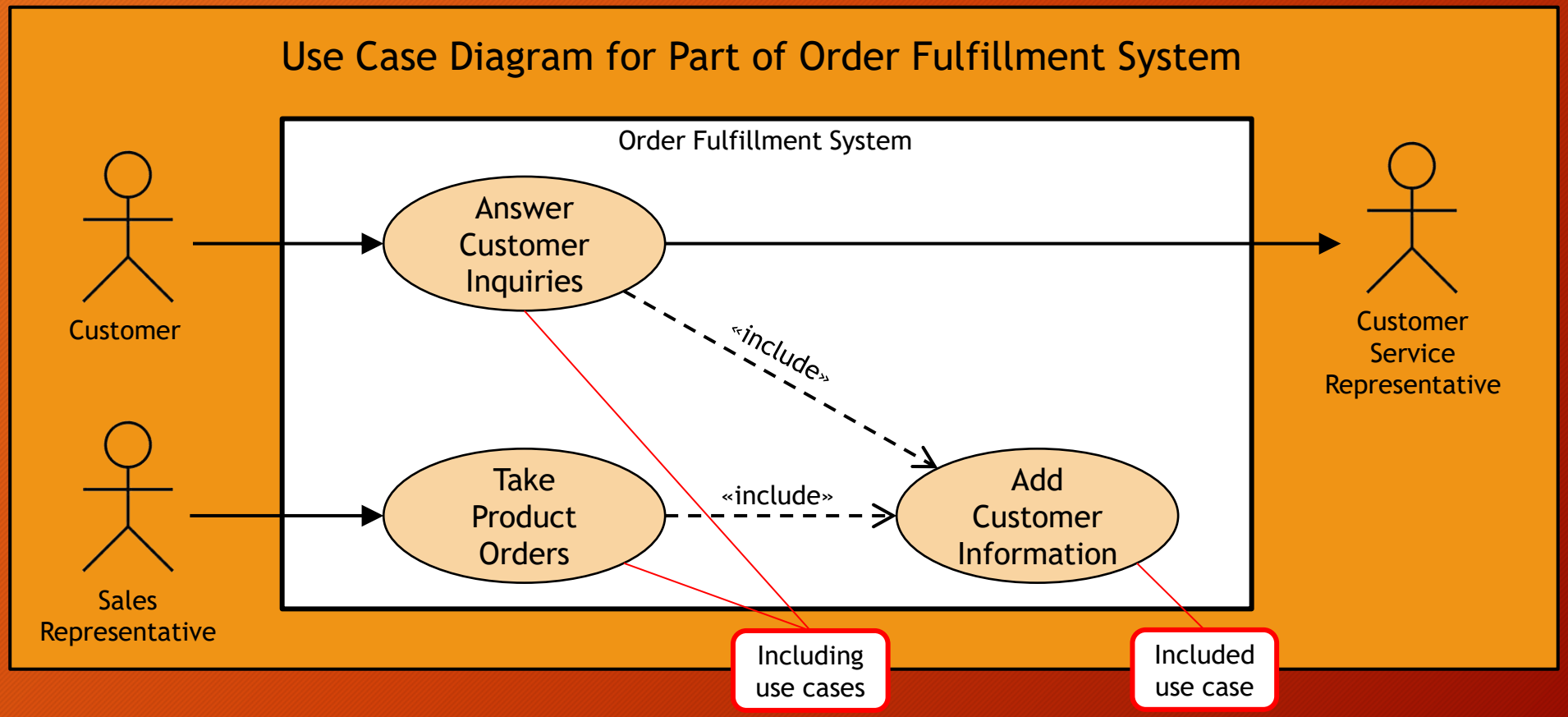
- There exists a debate among engineers whether it is appropriate to “decompose” use cases into lower-level use cases as you would decompose top-level system functions into lower-level sub-functions
- The consensus opinion among professional systems engineers is that it is NOT appropriate to decompose use cases
  - The thinking goes along these lines:
  - Decomposition can continue ad-nauseum
    - This is common and desirable in functional decomposition
    - But use cases ARE NOT the same as system functions
  - Use cases should only be broken down into child use cases as long as
    - The child use case is still within the vision of the user involved in the base use case
    - The child use case can be cleanly carved out of the base use case that,
      - Either the child use case can be a child of multiple base use cases «include»
      - Or the child use case hands an exception in the normal flow of the base use case «extend»
  - Once the user is no longer involved in the interactions at these lower levels, creating lower level child use cases serves no further purpose



Example of unnecessary use case decomposition
It is unlikely that a user will want to use the system simply to “input coin”
This is an example of a use case model that is crossing over into detailed system design work
It defeats the purpose of coordinating with the user over “what the user wants the system to do for them”

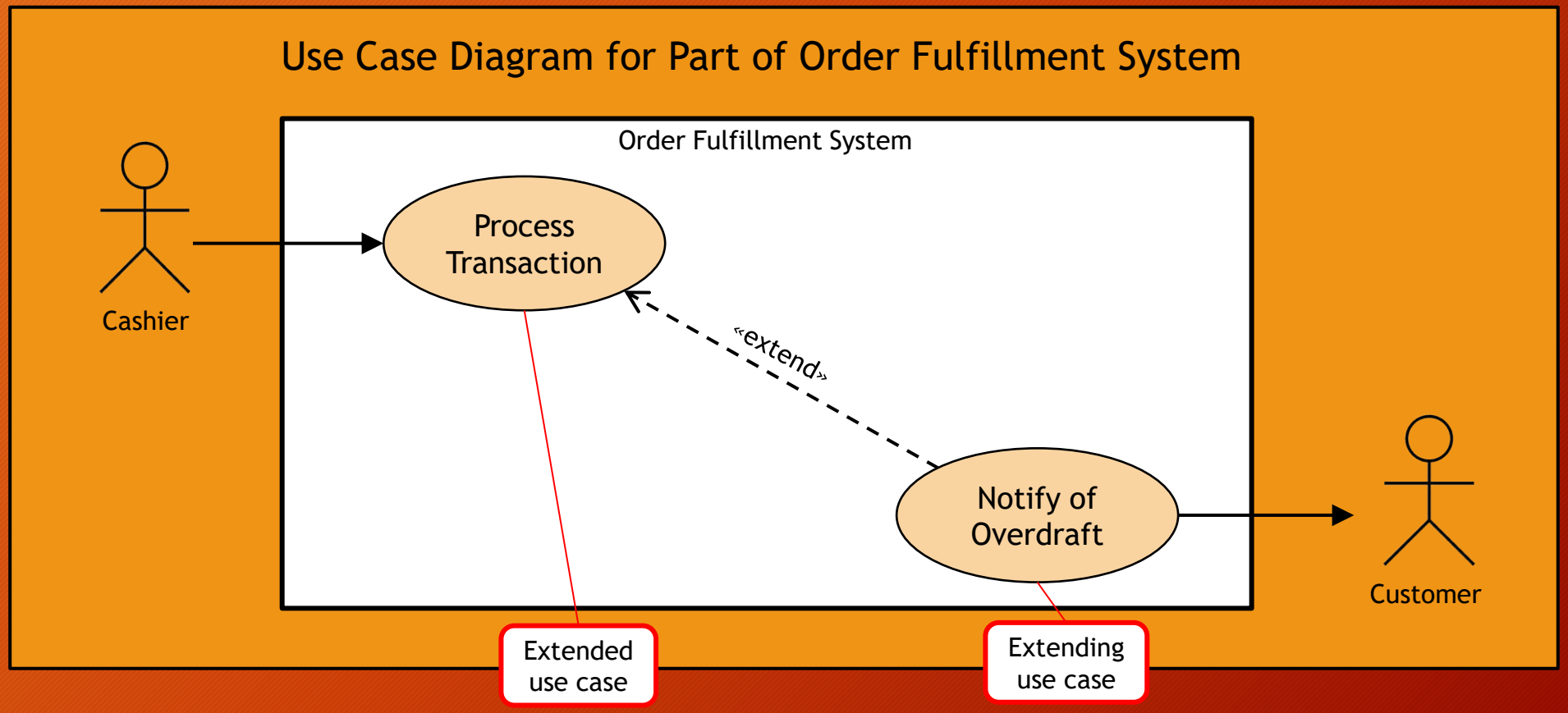
Representing Included Use Cases in Diagrams

- This example use case diagram demonstrates the use of the «include» relationship among use cases
- This example follows the use case description presented in this slide
- This example is not a complete description of an Order Fulfillment System, just the parts of the system addressing the interaction discussed in the associated use case description



Representing Extending Use Cases in Diagrams

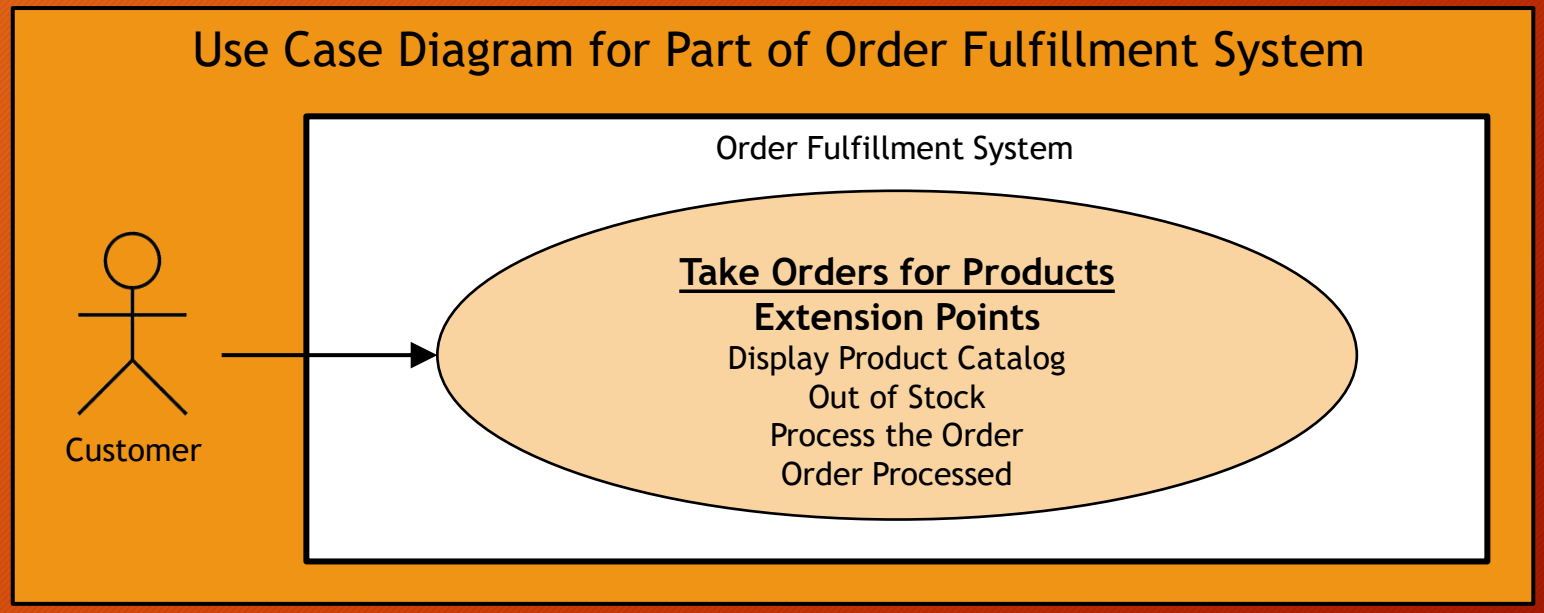
- This example use case diagram demonstrates the use of the «extend» relationship among use cases
- This example follows the use case description presented in this slide
- This example is not a complete description of an Order Fulfillment System, just the parts of the system addressing the interaction discussed in the associated use case description



Extension Points

Extension Points

- Extension points can be illustrated in use case diagrams
  - Not all extension points need be declared in a use case diagram
  - Only those that are considered “public” (accessed by use cases, not only by alternative flows) should be listed
- However, it should be noted that this is simply a method of documenting that these extension points exist
  - They are in no way directly tied to the extension points listed in the use case documentation
  - In fact, like much of SysML, use cases are not executable
  - So, the only purpose for identifying extension points in a use case diagram is simply to inform the reader of the diagram that these extension points exist (somewhere)



Generalization Relationships

Generalization Relationship among Use Cases

- The generalization relationship allows the creation of generalized behavior descriptions that can then be specialized to meet particular needs
  - The need for this kind of relationship arises from the need to describe families of systems in which child use cases inherit behavior from parent use case
  - In a sense, specialization has the mechanics of an include – because the specialized use case reuses the generalized behavior from the generalized use case
  - But it has the semantics of an extend – since it is the specialized use case that provides the additional behavior

- Example: the steps taken to use a bank card for ATM services are very similar to those taken to fuel a vehicle at a gas pump
  - The procedure to use is to establish an abstract use case that captures the common elements of the ATM service and the Fueling service that could apply to either service without specifically mentioning the details of the specific type of service
  - Then create two concrete use cases, one for each specific type of service, in which the particular details of each of the different types of service are specified

Abstract and Concrete Use Cases

Abstract Use Case

Conduct Transaction

Basic Flow

1. The Customer inserts a bank card into the dispenser machine.

2. The System reads the customer account information from the bank card.

3. The System requests the Customer to enter the PIN.

4. The Customer enters the PIN.

5. The System verifies that the PIN entered is correct by comparing it to the PIN that was read from the bank card.

6. The System contacts the Banking System to verify that the customer account information is valid

7. The System asks the Customer for the amount of the transaction.

8. The Customer enters the transaction amount.

9. The System contacts the Banking System to verify that the customer has sufficient funds to cover the transaction.

{Customer Conducts the Transaction}

10. The System records the actual amount of the transaction.

11. The System communicates to the Banking System that the transaction has been completed, indicating the actual amount of the transaction.

12. The System logs the transaction, capturing the date and time of the transaction, the amount of the transaction, and the account from which the funds were withdrawn.

13. The use case ends.

Concrete Use Case

Withdraw Cash

Basic Flow

At {**Customer Conducts the Transaction**}

1. The **System** checks to see if it has sufficient funds on hand to dispense the requested amount.

2. The **System** dispenses the requested amount of cash.

3. The **System** asks the Customer to take the cash.

4. The **Customer** takes the cash.

5. The behavior described in use case **Conduct Transaction** resumes.

Concrete Use Case

Fuel Vehicle (concrete use case)

There are many different proposed methods for documenting use case flows This is one method suggested by the book "Use Case Modeling" by K. Bittner and I. Spence

Basic Flow

At {**Customer Conducts the Transaction**}

1. The **System** asks the Customer to lift the pump handle and begin dispensing fuel.

2. The Customer dispenses fuel up to the value entered, or until the tank is full.

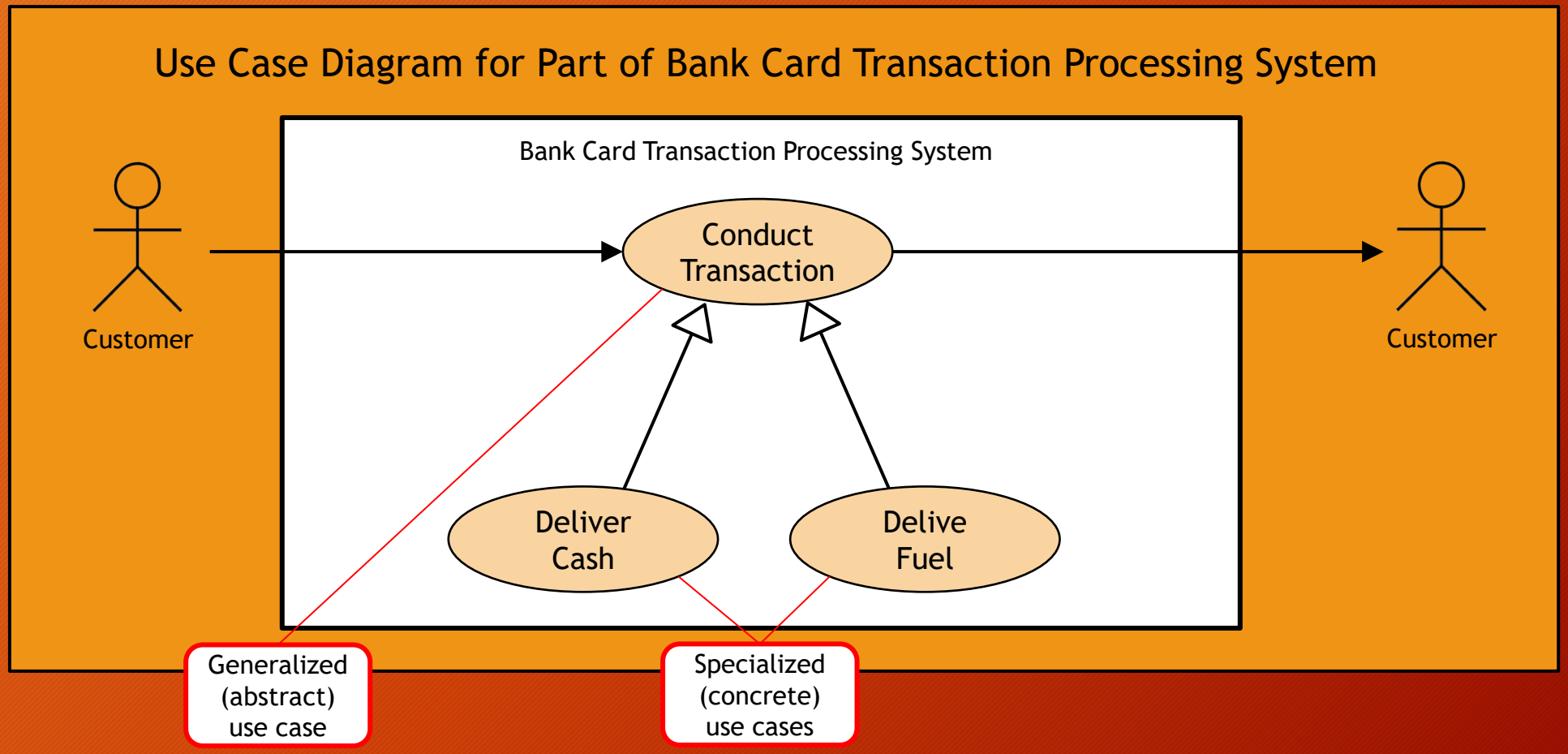
3. The Customer replaces the pump handle.

5. The behavior described in use case **Conduct Transaction** resumes.

It is important to understand that the flow associated with the concrete use cases is that of **separate use cases**

Use Case Generalization on Use Case Diagrams

- It is the behavior of the specialized use cases (concrete use cases) that is performed, depending on the type of engagement being experienced
  - The specialized use cases are actually reusing parts of the generalized use case when their behavior is invoked



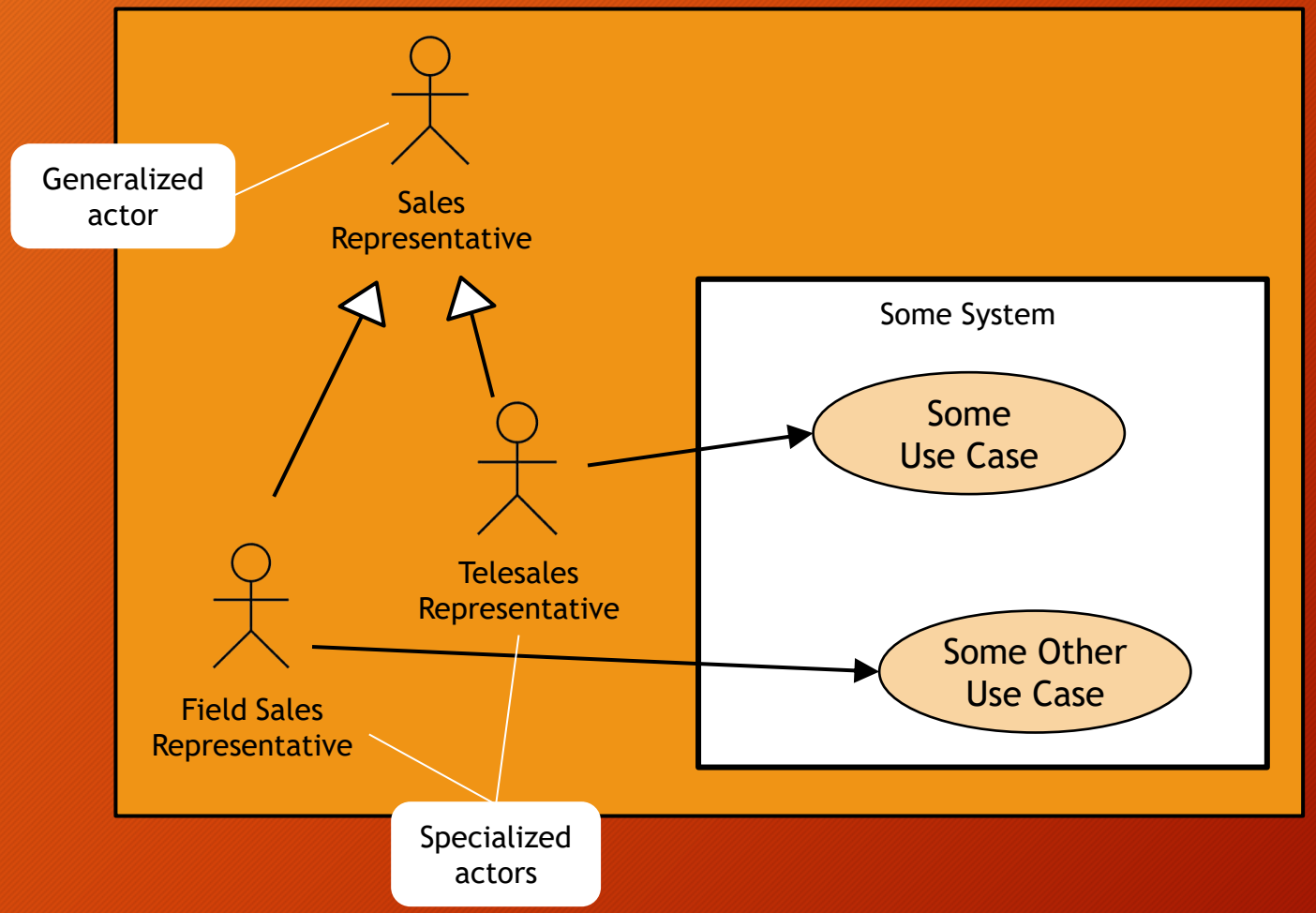
Generalization Relationship among Actors

- The generalization relationship among actors is used to show similarity between actors
  - It shows that some group of actors share common responsibilities or common characteristics
  - Useful characteristics to attach to actors include:
    - Their expertise
    - Things they need from the SoI
    - Usability and response requirements the actors impose on the system
  - Sometimes, actor generalization can be used to reduce the number of internal system communication relationships
  - But generally, actor generalization is of little value to use case modeling

- Misuse of actor generalization include:
  - Confusing system actors with user organizational roles and job titles
  - This is in appropriate because actors only define roles with respect to the SoI, nothing more

Actor Generalization on Use Case Diagrams

- There is no need for any other type of relationship between actors since they do not communicate with one another in use case analysis
  - Any such communications do not involve the system, and therefore do not belong on the use case diagram



Scenarios and Sequence Diagrams

Scenarios

- Scenarios (within the context of use cases) are instances, or specific occurrences, of use cases
- Scenarios are useful in that they force the use case designer to think within a certain context of use of the system
  - They help the use case designer walk through everything that will happen during the user interchange with the system
  - They can be useful later on in defining the test cases that will be used to verify system behavior and performance
- A single scenario will walk through one particular path of the use case from beginning to end
  - That single scenario will explore a particular way that the use case can be performed
  - Other scenarios can investigate other paths taken within the use case based on alternative flows, etc.
- It is especially important to consider boundary conditions – the points at which a small change in the value of some variable causes some very different behavior in the system as a whole
  - Consider things such as “what if ...?” situations
  - These may expose errors or gaps in the use case flow
- It is important to be realistic about how many scenarios can be covered on a limited budget
  - Stick with the basic flow, and alternative flows
  - Do not consider every single possible event that can occur in the system – there will be too many to consider
  - If alternative flows are truly independent, then they won’t affect each other and each flow can be an individual scenario
  - Further, truly independent alternative flows can often be combined into a single scenario, thus reducing effort

Sequence Diagrams

- Sequence diagrams are a great mechanism for documenting scenarios
- Sequence diagrams are one of the three SysML diagrams that document detailed system behavior
  - The other two are Activity diagram and State Machine diagram
  - Sequence diagrams are excellent at documenting detailed system behavior, but they also have a role in documenting use case scenarios
- Sequence diagrams are well suited for documenting scenarios as the primary objective of a sequence diagram is
  - To document the sequential nature of actor-to-actor (to-system) interactions
- The sequence diagram offers a great amount of facility for defining very detailed and elaborate descriptions of system behavior
  - Usually, for describing use case scenarios, only the most basic features of sequence diagramming capability are required
  - It is recommended that a use case designer considering the use of sequence diagrams for documenting scenarios refer to other resources for additional information on the full set of capabilities of sequence diagrams
  - Only the most basic features of sequence diagrams are covered here

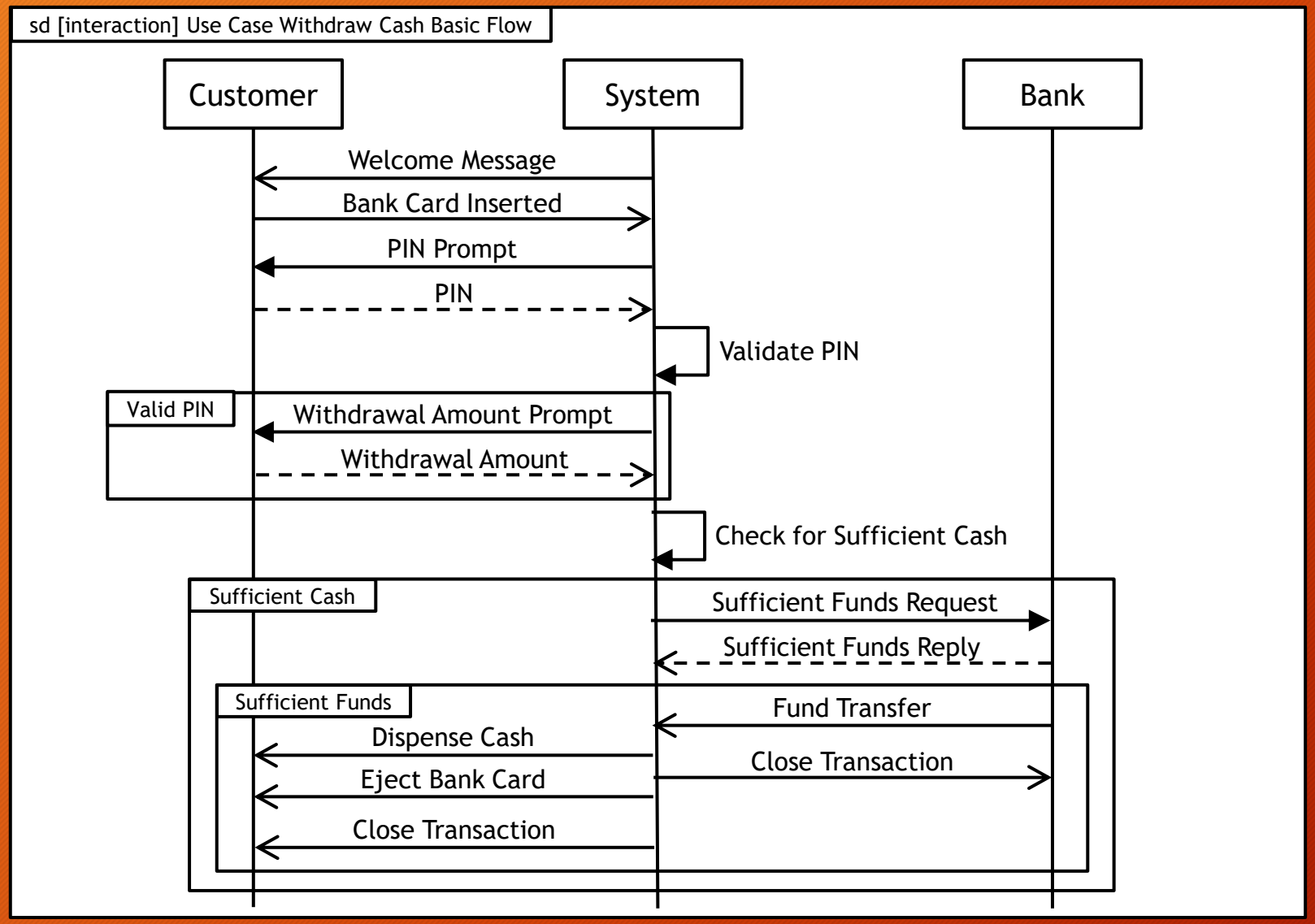
It should be noted that the Sequence Diagram is only one of three diagrams that can be used to detail system behavior
(The Activity Diagram and the State Machine Diagram are the other two)
In fact, any of these three behavior diagrams can be used to detail the use case, depending on the nature of the interactions being described by the use case
We are emphasizing the sequence diagram here only because most interactions are “message-based” that the sequence diagram is particularly good at describing

Main Sequence Diagram Features

- The main features of a sequence diagram include
  - Diagram frame – representing the use case being examined
  - Lifeline – the relative lifetime of an actor
  - Occurrence of an event – for simplicity, these are the interactions between the actor(s) and the system, and are represented as exchanged messages
  - Execution - optional indication on a lifeline of the receiver of a message indicating execution of some behavior for an amount of time on the lifeline
  - Combined fragments – simply put: fragments of interaction controlled by guard statements that determine whether the conditions exist for the fragment of interaction to execute
    - Opt – the guard statement is a single condition statement
    - Alt/Else – evaluation of the guard statement resolves to one of several options
    - Par – provides two or more parallel fragments that can execute simultaneously
    - Loop – the guard statement provides the condition for exiting a processing loop
- Message types
  - Synchronous – the sender of the message waits for a rely from the receiver before proceeding with further interaction
  - Reply – the receiver of a synchronous message sends a reply that is specifically associated with the received message
  - Asynchronous – the sender of the message does not wait for a rely and continues normally with further interaction
  - Message-to-Self – Indicates that the actor (or system) on the lifeline is executing some kind of behavior

Sequence Diagram Syntax

- This is a simple example of a sequence diagram for a use case scenario indicating the main features



Not all details of the scenario are represented in this simple model

LEGEND

- Asynchronous Message
- Synchronous Message
- Reply Message
- Message-to-Self

References

- Bittner, K., Spence, I. (2003), *Use Case Modeling*, Addison-Wesley Publishers
- Liberti, L. (2008), *Software Modelling and Architecture: Exercises*, LIX, The Computer Science Laboratory of Ecole Polytechnique,
  - https://www.researchgate.net/publication/251224162_Software_Modelling_and_Architecture_Exercises