CPE340

Week 2 Lecture – Use Case Modeling

At the end of the lesson, students should be able to:

1. Write fully developed use case descriptions
2. Develop activity diagrams to model flow of activities
3. Develop system sequence diagrams
4. Use the CRUD technique to validate use cases
5. Explain how use case descriptions and UML diagrams work together to define functional requirements

Overview

The main objective of defining requirements in system development is understanding users' needs, how the business processes are carried out, and how the system will be used to support those business processes.

The models introduced in Chapters 3 and 4 focus on two primary aspects of functional requirements: the use cases and the problem domain classes involved in users' work. User stories are sometimes used in place of use cases with Agile development.

Use cases are identified by using the user goal technique and the Opening case Electronics Unlimited: Integrating the Supply Chain Electronics Unlimited is a warehousing distributor that buys electronic equipment from various suppliers and sells it to retailers throughout the United States and Canada. It has operations and warehouses in Los Angeles, Houston, Baltimore, Atlanta, New York, Denver, and Minneapolis.

Its customers range from large nationwide retailers, such as Target, to medium-sized independent electronics stores. Most large retailers have moved toward integrated supply chains. Information systems used to be focused on processing internal data; however, today, these retail chains want suppliers to become part of a totally integrated supply chain system. In other words, the systems need to communicate between companies to make the supply chain more efficient. To maintain its position as a leading wholesale distributor, Electronics Unlimited has to convert its system to link with its suppliers (the manufacturers of the electronic equipment) and its customers (the retailers). It is developing a completely new system that uses object-oriented techniques to provide these links. Object-oriented techniques facilitate system-to-system interfaces by using predefined components and objects to accelerate the development process.

Fortunately, many of the system development staff members have experience with object-oriented development and are eager to apply the techniques and models to the system development project. William Jones is explaining object-oriented development to the group of systems analysts who are being trained in this approach. "We're developing most of our new systems by using object-oriented principles," he tells them. "The complexity of the new system, along with its interactivity, makes the object-oriented approach a natural way to develop requirements. The object-oriented models track very closely with the new object-oriented programming languages and frameworks."

William is just getting warmed up. “This way of thinking about a system in terms of objects is very interesting,” he adds. “It is also consistent with the object-oriented programming techniques you learned in your programming classes.

User Case Descriptions

A ***use case description*** is a textual model that lists and describes the processing details for a use case.

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.

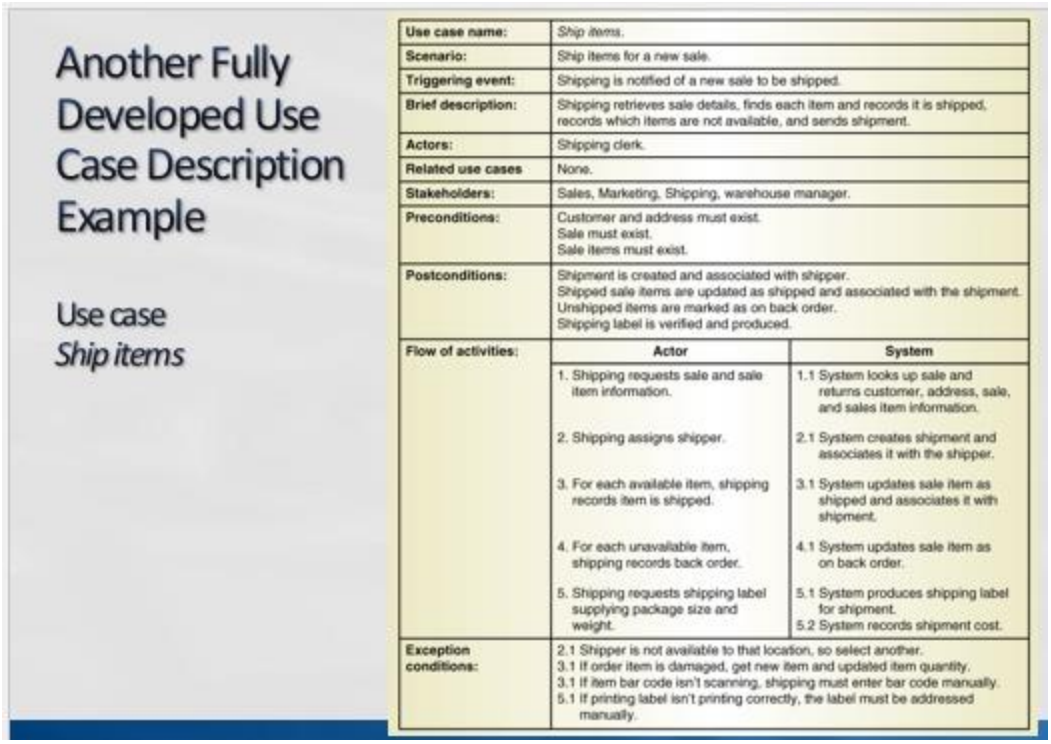
Fig. 1. Use cases and brief use case descriptions

Use case name:	*Create customer account.*	
Scenario:	Create online customer account.	
Triggering event:	New customer wants to set up account online.	
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.	
Actors:	Customer.	
Related use cases:	Might be invoked by the *Check out shopping cart* use case.	
Stakeholders:	Accounting, Marketing, Sales.	
Preconditions:	Customer Account subsystem must be available. Credit/debit authorization services must be available.	
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.	
Flow of activities:	**Actor**	**System**
	1. Customer indicates desire to create customer account and enters basic customer information. 2. Customer enters one or more addresses. 3. Customer enters credit/debit card information.	1.1 System creates a new customer. 1.2 System prompts for customer addresses. 2.1 System creates addresses. 2.2 System prompts for credit/debit card. 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.
Exception conditions:	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.	

Fig. 2. Fully developed use case description for **Create customer account**.

Write a fully developed use case description for more complex use cases. Typical use case description templates include:

- Use case name (Verb-noun)
- Scenario (if needed)
    - A use case can have more than one scenario (special case or more specific path)
- Triggering event
    - Based on event decomposition technique
- Brief description
    - Written previously when use case was identified
- Actors
    - One or more users from use case diagrams
- Related use cases (<<includes>>)
    - If one use case invokes or includes another
- Stakeholders
    - Anyone with an interest in the use case
- Preconditions
    - What must be true before the use case begins
- Postconditions
    - What must be true when the use case is completed
    - Use for planning test case expected results
- Flow of activities
    - The activities that go on between actor and the system
- Exception conditions
    - Where and what can go wrong



Another Fully Developed Use Case Description Example

Use case
Ship items

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.

Fig. 3. Fully developed use case description for **Ship Items**

Activity Diagrams for Use Cases

Another way to document a use case is with a UML activity diagram.

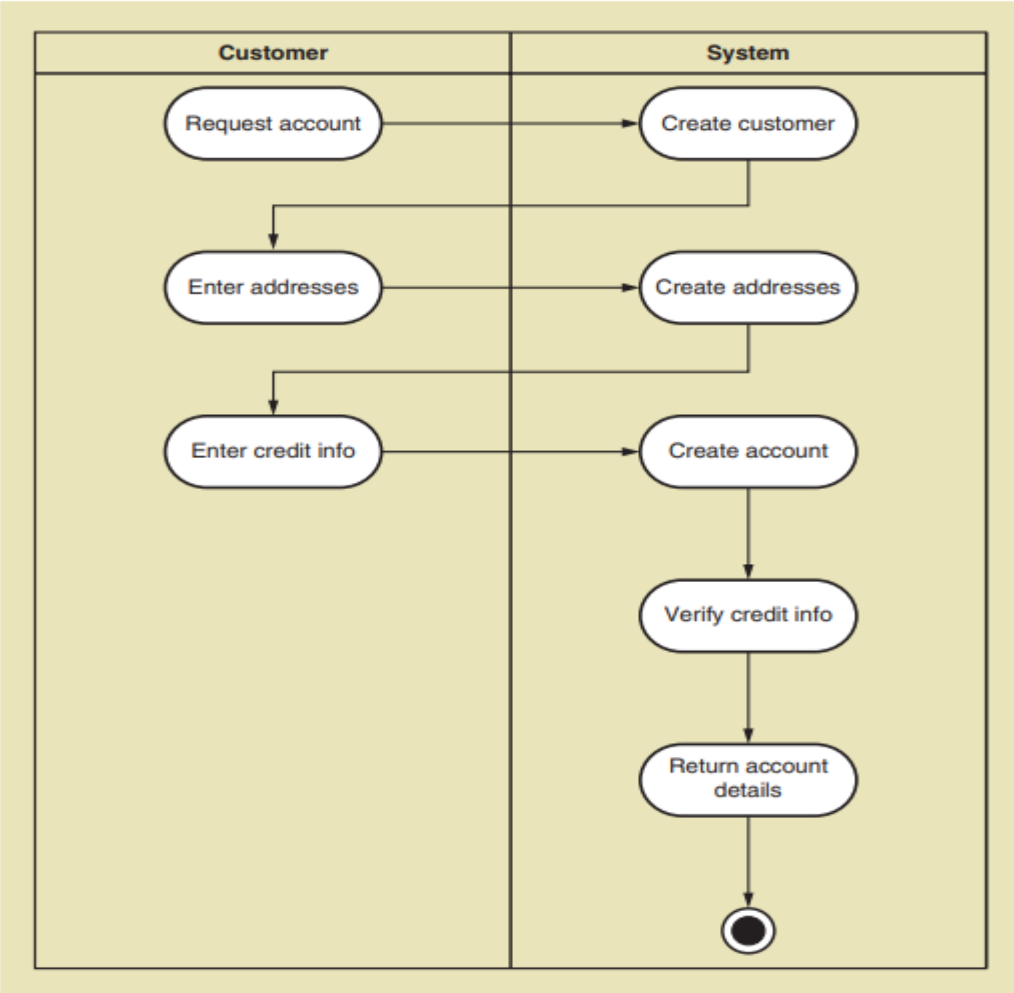


Fig. 4. Activity diagram for **Create customer account** use case

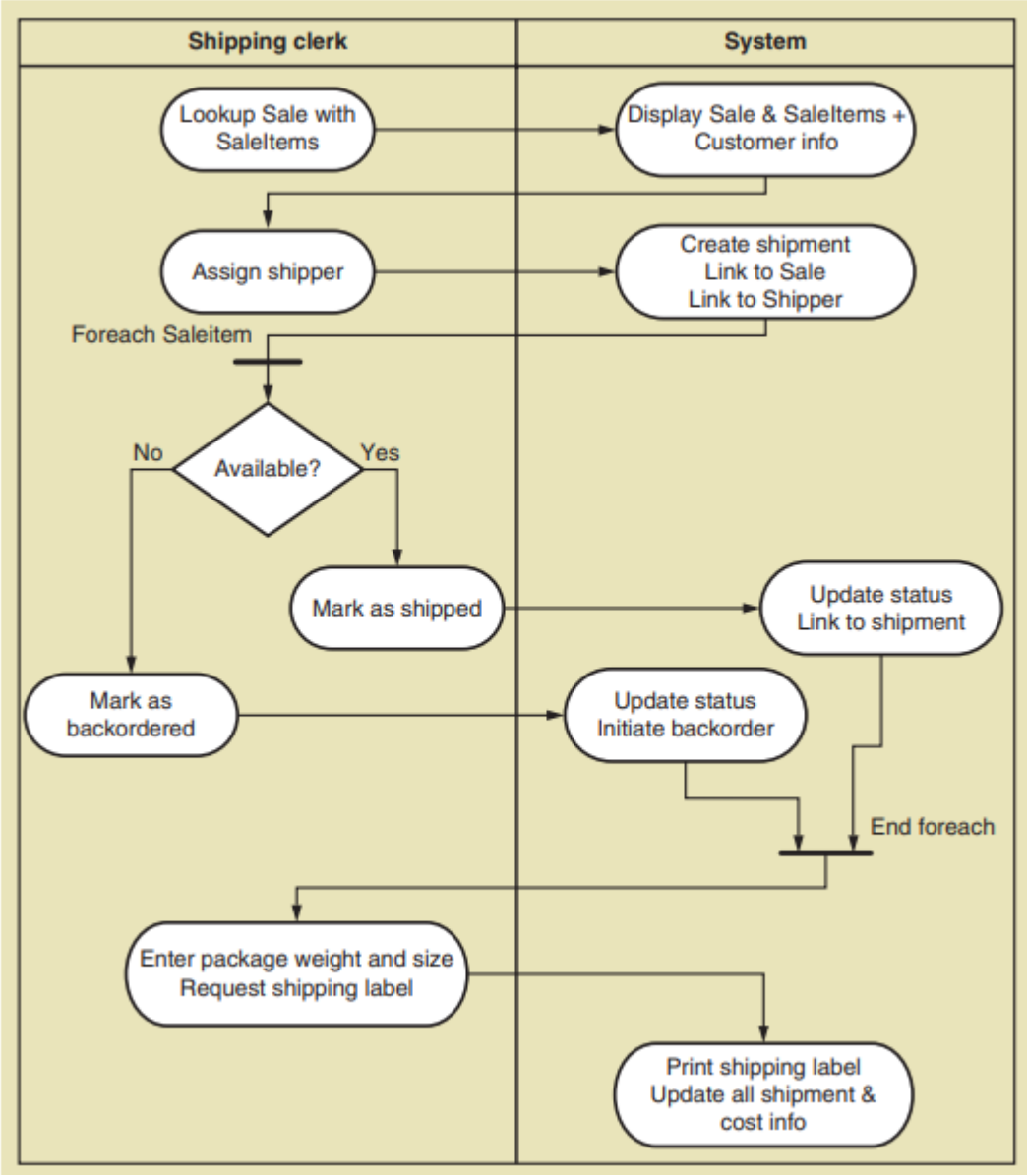


Fig. 5. Activity diagram for **Ship Items** use case

System Sequence Diagram

In the object-oriented approach, the flow of information is achieved through sending messages either to and from actors or back and forth between internal objects. A system sequence diagram (SSD) is used to describe this flow of information into and out of the automated portion of the system. Thus, an SSD documents the inputs and the outputs and identifies the interaction between actors and the system. It is an effective tool to help in the initial design of the user interface by identifying the specific information that flows from the user into the system and the information that flows out of the system back to the user. An SSD is a special type of UML sequence diagram.

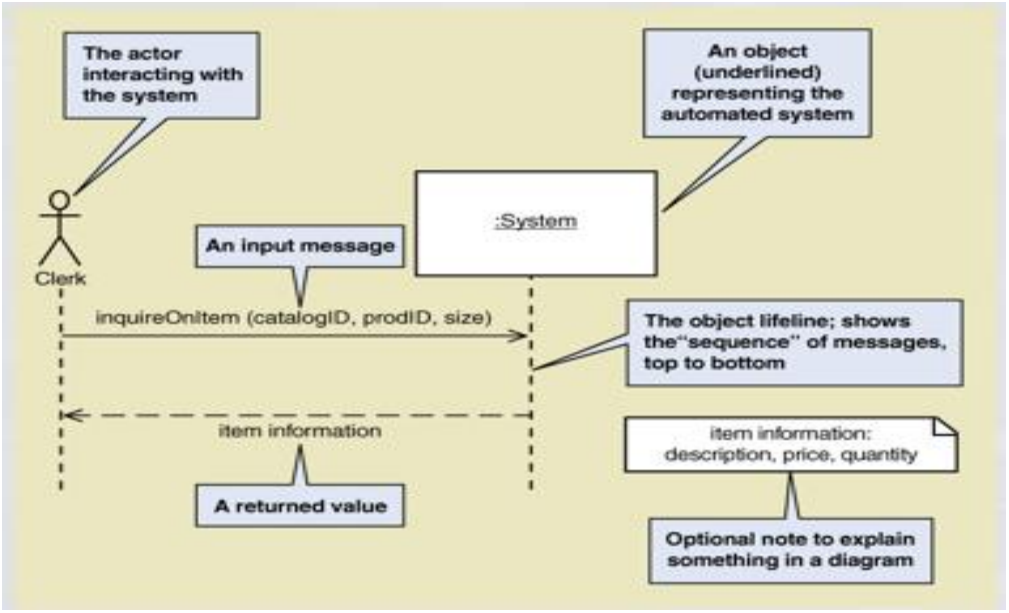


Fig. 6. Sample system sequence diagram

The clerk is sending a request (a message) to the system to find an item. The input data that is sent with the message is contained within the parentheses, and in this case, it is data to identify the item. The syntax is simply the name of the message followed by the input parameters in parentheses. This form of syntax is attached to a solid line with arrow.

The returned value has a slightly different format and meaning. Notice that the line with arrow is dashed. A dashed arrow indicates a response or an answer (in programming, a return), and as shown in the figure, it immediately follows the initiating message. The format of the label is also different. Because it is a response, only the data that are sent on the response are noted. There is no message requesting a service—only the data being returned. In this case, a valid response might be a list of all the information returned—for example, the description, price, and quantity of an item. However, an abbreviated version is also satisfactory. In this case, the information returned is named item information. Additional documentation is required to show the details.

A ***loop frame*** is a notation on a sequence diagram showing repeating messages.

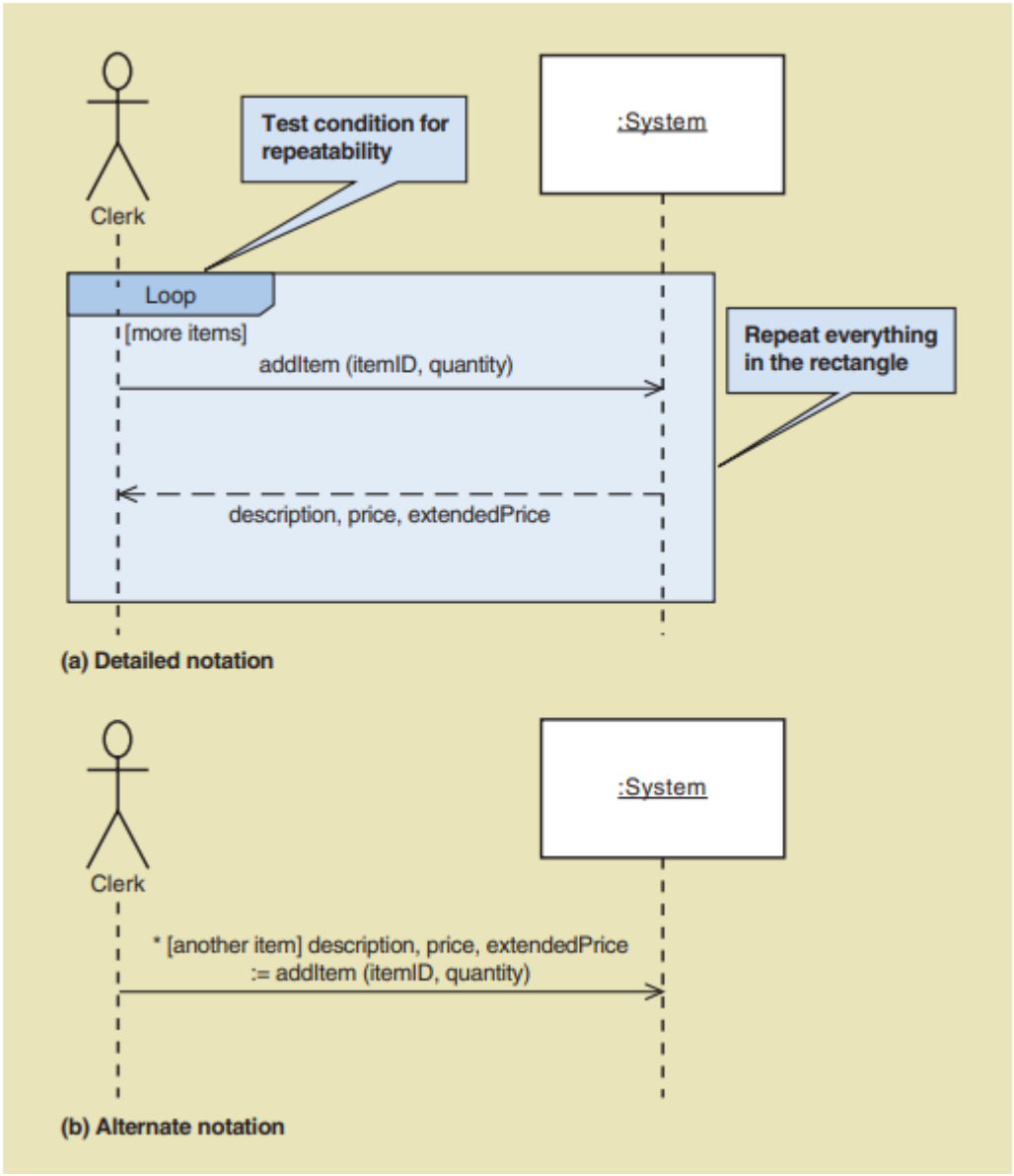


Fig. 7. Repeating message in a) detailed loop frame notation and b) alternate notation

Here is the complete notation for a message:

[true/false condition] return-value := message-name (parameter-list)

Any part of the message can be omitted. In brief, the notation components do the following:

■ An asterisk (*) indicates repeating or looping of the message.

■ Brackets [ ] indicate a true/false condition. This is a test for that message only. If it evaluates to true, the message is sent. If it evaluates to false, the message isn't sent.

■ Message-name is the description of the requested service written as a verb-noun.

■ Parameter-list (with parentheses on initiating messages and without parentheses on return messages) shows the data that are passed with the message.

■ Return-value on the same line as the message (requires :=) is used to describe data being returned from the destination object to the source object in response to the message.

Steps for Developing SSD

1. Identify input message
    - See use case flow of activities or activity diagram
2. Describe the message from the external actor to the system using the message notation
    - Name it verb-noun: what the system is asked to do
    - Consider parameters the system will need
3. Identify any special conditions on input messages
    - Iteration/loop frame
    - Opt or Alt frame
4. Identify and add output return values
    - On message itself: aValue:= getValue(valueID)
    - As explicit return on separate dashed line

Use Cases and CRUD

CRUD technique is an acronym for Create, Read/Report, Update, and Delete, a technique to validate or refine use cases. The CRUD technique is most useful when used as a cross-check along with the user goal technique. Users will focus on their primary goals and use cases that update, or archive data will often be overlooked. The CRUD technique makes sure all possibilities are identified. Sometimes, domain classes are shared by a set of integrated applications.

Data entity/domain class	CRUD	Verified use case
Customer	Create	Create customer account
	Read/report	Look up customer Produce customer usage report
	Update	Process account adjustment Update customer account
	Delete	Update customer account [to archive]

Fig. 8. Verifying use cases with the CRUD technique.

CRUD Analysis Steps:

1. Identify all domain classes
2. For each class verify that use cases exist to
    - Create a new instance
    - Update existing instances
    - Reads or reports on information in the class
    - Deletes or archives inactive instances
3. Add new use cases as required. Identify responsible stakeholders
4. Identify which application has responsibility for each action: which to create, which to update, which to use

Use case vs. entity/domain class	Customer	Account	Sale	Adjustment
Create customer account	C	C		
Look up customer	R	R		
Produce customer usage report	R	R	R	
Process account adjustment	R	U	R	C
Update customer account	UD (archive)	UD (archive)		

Fig. 9. Sample CRUD Matrix

Integrating Requirements Models

- Use cases
    - Use case diagram
        - Use case description
        - Activity diagram
        - System sequence diagram (SSD)
- Domain Classes
    - Domain model class diagram
        - State machine diagram

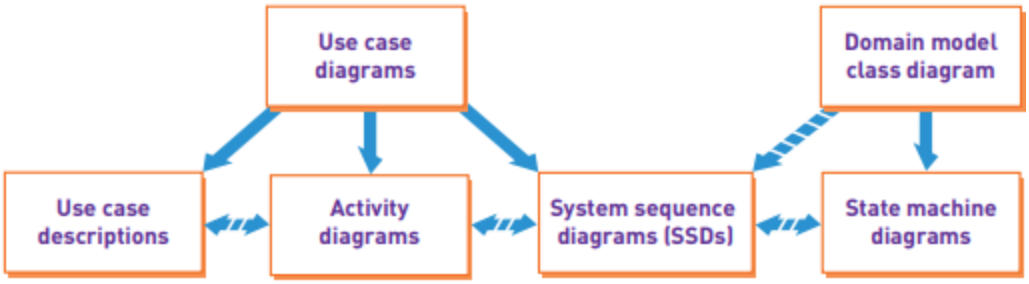


Fig. 10. Relationships among object-oriented requirements models

Fig. 10 illustrates the primary relationships among the requirements models for object-oriented development. The use case diagram and other diagrams on the left are used to capture the processes of the new system. The class diagram and its dependent diagrams capture information about the classes for the new system. The solid arrows represent major dependencies, and the dashed arrows show minor dependencies. The dependencies generally flow from top to bottom, but some arrows have two heads to illustrate that influence goes in both directions. Note that the use case diagram and the domain model class diagram are the primary models from which others draw information. You should develop those two diagrams as completely as possible.

The detailed descriptions—either in narrative format or in activity diagrams—provide important internal documentation of the use cases and must completely support the use case diagram. Such internal descriptions as preconditions and postconditions use information from the domain model class diagram. These detailed descriptions are also important for development of system sequence diagrams.

Thus, the detailed descriptions, activity diagrams, and system sequence diagrams must all be consistent about the steps of a particular use case. As you progress in developing the system and especially as you begin doing detailed systems design, you will find that understanding the relationships among these models is an important element in the quality of your models