

Summary of the UML Diagrams

Originated May 9, 2002; Use case and sequence diagram info added Feb 3, 2005
(from http://www.omg.org/gettingstarted/what_is_uml.htm#12DiagramTypes
and the dictionary <http://softdocwiz.com/UML.htm>)

There are three types of UML diagrams: **behavior**, structural, and *model management* diagrams. Those font styles are used here to indicate when each type is being discussed. Based on UML 1.4.

Behavior Diagrams help to understand system functional requirements, and express that through the system's design.

- **1. Use Case Diagram** - used by many methodologies (not just OO) during requirements gathering
 - Can have a System Use Case Diagram, or more specific Use Case Diagrams
- Interaction diagrams:
 - **2. Sequence Diagram** focuses on the time-ordering of messages between objects.
 - Can have top level System Sequence Diagram, then more specific Sequence Diagrams
 - **3. Collaboration Diagram** has nearly the same information as a Sequence Diagram, but focuses on the structural organization of objects.
- **4. Activity Diagram** shows flows among activities; this is a process flow chart to show how your system functions. It focuses on processes and organizations that perform them.
- **5. Statechart Diagram** shows a sequence of states that an object or an interaction goes through during its lifetime, in response to events, and also the responses that the given object or interaction makes to those events.

Structural Diagrams help organize objects and establish their associations with each other, bridging the gap from design to implementation.

- 6. Class Diagram (static) shows the classes which ever exist, and how they are associated to each other. Looks like an ERD, but isn't.
 - May be broken down into Conceptual, Design, and Application Class Diagrams, using more and more detail
 - Conceptual Class Diagram is also called the Domain Model
- 7. Object Diagram (dynamic) is a Class Diagram at one point in time
- 8. Component Diagram shows how your system is logically composed of software files (from packages), other supporting software, and external interfaces.
- 9. Deployment Diagram shows which components run on which physical processors.

Model Management Diagrams help describe how and where the objects are implemented.

- *10. Package Diagram* shows how classes and other packages are grouped into packages
- *11. Subsystem Diagram* shows how logically related sets of components form subsystems.
- *12. Model Diagram* includes all of the other diagrams to show how the system is structured and functions.

Scope of UML Diagrams

December 2004

The terminology here is based on the draft UML 2.0 specification (e.g. citing *communication diagram* instead of the older term *collaboration diagram*).

- The screen flow diagram isn't part of UML.
- The meaning of shapes (boxes, stick figures, ovals, etc.) and lines given below are just common examples for each diagram; the lists aren't meant to be all-inclusive.

Diagram	Typical Scope	Shapes represent	Lines represent
Activity Diagram	One or a few use cases	Processes Decisions	Process flow
<u>Class Diagram</u>	Entire system	Classes	Associations
Collaboration (Communication) Diagram	One use case	Classes	Message paths
<u>Collaborations</u>	Adds to Class Diagram	Collaborations Classes	Participation in collaboration
<u>Component Diagram</u>	Entire system	Components	Interfaces
<u>Composite Structures</u>	Adds to Class Diagram	Classes	Associations
<u>Deployment Diagram</u>	Entire system	Nodes	Communication paths
Interaction Overview Diagram	One or a few use cases	Processes Decisions Sequence diagrams	Process flows
<u>Object Diagram</u>	Entire system at one moment	Objects	Associations
<i>Package Diagram</i>	Entire system	Packages	Dependencies
Screen Flow Diagram	Entire system	Interface objects	Navigation paths
Sequence Diagram	One use case	Objects Actors	Messages
State Diagram	One or a few use cases; one complex object	States of an object	Transitions
Timing Diagram	One or a few use cases; several simple objects	State timelines	Events which change a state
Use Case Diagram	Entire system	Actors Use cases External systems	Ability to use use cases

Summary of UML Diagram Differences

Interaction Diagrams

Trait	System Sequence Diagram (SSD)	Sequence Diagram (SQD)	Collaboration Diagram (CD)
Scope of Diagram	One use case, possibly with extensions	One use case, possibly with extensions	One use case and its extensions
Messages	Are not numbered	Are not numbered	Are numbered (1, 1.1, ...)
Returns	May be shown, with data described	May be shown, with data described	Are not shown
System appears as	One object	Many objects*	Many objects*
External actors	May be shown	May be shown	May be shown
Primary actor	Is shown	Is shown	Is not shown**
Conditional and/or exclusive messages	May be shown	May be shown	May be shown

* Objects needed for each diagram are based on the conceptual class diagram, with additional interface and control classes defined as needed (thus supporting creation of the design class diagram)

** Unless the tool requires the primary actor to be shown (e.g. Rose)

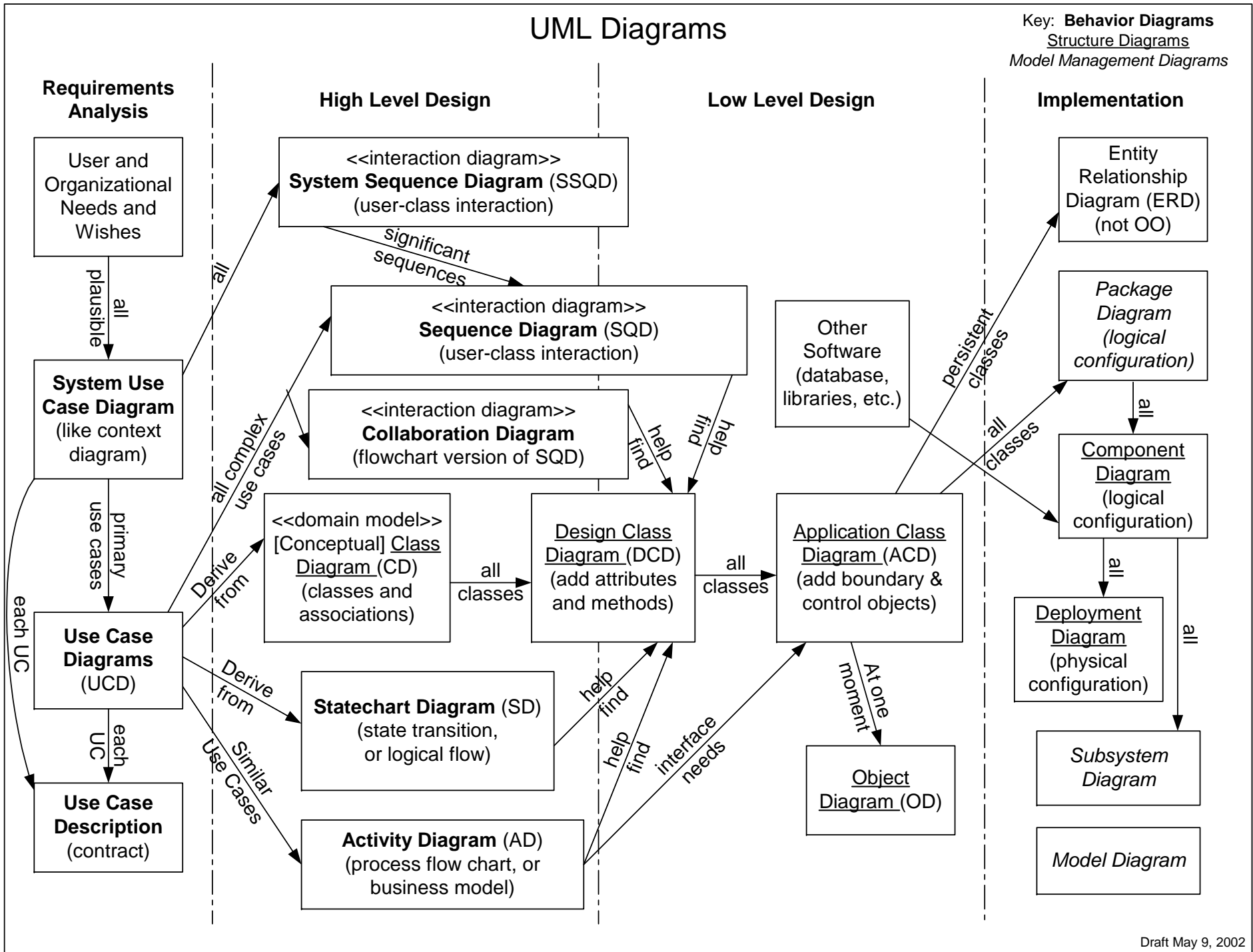
Class Diagrams

Trait	Conceptual	Design	Application
Scope	Whole system or major subsystem	Whole system or major subsystem	Whole system or major subsystem
Class Name	Is shown	Is shown	Is shown
Type of Objects	Conceptual	Software	Software, with boundary and control objects
Attributes	Some shown	All shown, with type and visibility	All shown, with type and visibility
Attribute Types*	Primitive and non-primitive	Primitive and non-primitive	Primitive and non-primitive
Methods	Are not shown	Are shown	Are shown
Associations	Show all possible	Show only needed	Show only needed
Association Directionality	Assumed 2-way	Directional	Directional
Multiplicity	Is shown	Is shown	Is shown
Visibility between classes	Only association	All types	All types
Conditionals	Are not shown	Are not shown	Are not shown
Reference Attributes	Are not shown	Are not shown	Are shown

* Classes which are used only to describe non-primitive data types do not need to be shown with any class diagram; they can be shown separately.

UML Diagrams

Key: **Behavior Diagrams**
Structure Diagrams
 Model Management Diagrams



Use Case Documentation Advice

January 13, 2005 – Refined January 12, 2006

Glenn Booker

The most challenging part of use case documentation is writing the Main Success Scenario (MSS); and for detailed documentation, the Alternate Scenarios (also called Extensions). Here are some thoughts to help improve yours. See examples from [INFO 355](#) Assignments 3 and 4.

General Philosophy & Approach

- The steps in a MSS should include actions by the actors involved, and details of how the system responds to them. There may be times where the actor does several things in a row, or the system does several things in a row, but overall, the MSS should look like a **conversation between the actor and the system**.
 - The MSS and Alternate Scenarios are describing the logic which will guide the detailed design of that portion of the system, so write them as **instructions to an analyst or programmer** who doesn't understand the business process.
 - In writing a MSS, **look for balance in the dialog between actors and the system**. Put yourself in the actor's position, and make sure they stay informed what the system is doing, if a task was completed successfully, etc. Don't leave the actor in the dark!
- A MSS should describe the **most likely or frequent way a process will take place**. The Alternate Scenarios (Extensions) can address more unusual circumstances.
 - For example, if payment can be made with cash, credit card, or money order, then the MSS might handle cash payment, and two Alternate Scenarios should describe how credit card or money order payments are handled.
 - Always assume the **positive outcome happens in a MSS** – after all, it's a *Success* scenario. Handle negative outcomes in the Extensions or Alternate Scenarios.
 - Dreadful movie analogy: the MSS is the Disney version of the use case, and the Extensions handle the Tim Burton perspective.
- **Follow the outline given in lecture 2 for the scope and structure of 'casual' and 'detailed' use case documentation**. Keep the structure clear for each section. *Do not* use a narrative paragraph to describe a use case (yes, that's a recognized method for very brief use case documentation, but we're not using it in this context).
 - **Don't document trivial use cases**, like system startup, logging into the system, etc. These are simple and common functions, and not worth a formal description.
- Keep focused on **documenting the system's functional requirements**
 - Make sure the **information system plays a significant role** in each use case. Some use cases are important for a person to accomplish as part of their job (they have significant business value), but unless they also involve interaction with the information system, we don't want them documented as use cases.
 - Likewise, only document actors or external systems that **interact directly with your system**. This is a key aspect of modeling anything – only model relevant aspects of the problem.

- Be sure to **number use cases**, both in the use case diagram, and in the corresponding documentation. This makes it easier to find the documentation for a given use case. See discussion of MSS step and Extension numbering later.
 - So if I see use case “12. Generate Monthly Sales Report” in a use case diagram, there should be use case documentation with *exactly that same name and number*.
- As you develop use case documentation, **keep track of what the actor currently sees**. A common continuity error arises from updating a screen that was either previously deleted, or is buried under two other screens that were added more recently.
 - It may help to **label sticky notes with screen names, and walk through a use case**, to make sure you’re creating, updating and/or removing screens consistently. Check both normal processing, and using the extensions, and make sure both work. This is also a good technique for making sure sequence diagram loops work correctly.
- **We are deliberately vague about how the tasks are accomplished** – we just want to capture the spirit of each action, and let the programmers figure out the details later.
 - So we might say the actor is going to ‘Navigate to the New Customer Screen’ – but we do NOT want to specify if they did so by:
 - selecting a dropdown menu option,
 - clicking on a navigation button,
 - typing a text command,
 - using a keyboard shortcut,
 - speaking into a microphone,
 - blinking their left eye twice,
 - or any other specific way of accomplishing that task.
 - Exactly how tasks are performed is a user interface design and implementation detail to be worked out much later.

MSS & Extension Types of Steps

While writing use case documentation, keep in mind that there is a fairly limited range of things that an actor or the system will be doing in a single step of the MSS or an Extension.

Steps within a use case typically include many of the following types of actions:

- Where to begin? Assume that the **primary actor is starting from a generic part of the user interface**. In most cases, you can assume they have logged into the system (if needed) and are starting at the application’s Main Menu or its equivalent.
 - This is a **key assumption about the start of each use case**. The only use cases where this isn’t true are likely to be trivial use cases, for which you don’t write a MSS!
 - The **first step of a MSS must be by an actor**. Something outside the system has to initiate a use case.
 - In the case of timed repeating use cases (e.g. for periodic report generation, system backup, etc.) the **actor could be a Timer**.

- Steps in a MSS should include the **acts of navigating and creating the user interface** (e.g. second part of Assignment 4, steps 2 and 3). This often involves making interface design assumptions, and sometimes data structure assumptions.
 - User interfaces are generically called ‘screens’ or ‘windows’ or ‘reports’, unless you happen to be dealing with a console or text-based application (very rare).
 - It is a significant **decision whether to change the contents of an existing screen, or create a new window**.
 - Often short messages, especially those with positive outcomes, are shown by updating an existing screen. For example, status messages for successful completion of a task might just appear on an existing screen.
 - Longer or more complex output (e.g. search results, or reports), or error messages, are often shown on a new window.
 - A MSS should **state when new screens are created**.
 - For example, if you want to edit a customer’s address, from the Main Menu you might first open a window to Manage Customer Data, then the actor selects an option to Edit Existing Customer. That brings up a screen to search for the desired customer, and the next screen displays their information.
 - You can **distinguish between static and dynamic content** for a screen. Assume that static content (the same information is always presented) can be an assumed part of the screen, so no separate steps are needed to create its contents. Explicit steps should be stated to generate and display dynamic content.
 - Reports are dynamic content, so there are often a set of steps to create the report window, generate the report contents, format the report contents, and display the contents in the window.
 - A MSS should state explicitly when the **contents of an existing screen change** (data is added or updated).
 - A common example might be populating a dropdown list with the results of a search, such as the list of artifacts of a specific type in the second part of Assignment 4, step 6.
- If a screen needs to **display customized data** (such as search results or part of a list of records), then steps need to explicitly describe extracting that data from its source, and displaying it on the screen. This might involve **making some assumptions about how the data is structured**.
 - For example, the first part of Assignment 4 describes extracting a list of possible drivers for a particular shipment, from a list of all drivers. This assumes there is a permanent list of all drivers, and from it we’ll create a temporary list of possible drivers for this particular shipment.
 - The second part of Assignment 4 assumes that a list is created of artifacts to be checked out, and that list is subsequently modified by the security and availability checks.

- Include steps for **background processing** the system must do to accomplish the use case. The preparation of customized data is a common example, but any kind of analysis, computation, validating data, sorting or organizing data, making decisions or comparisons, communicating with external systems, etc. also fit into this category.
 - These steps are typically invisible to the actor unless something went wrong.
- Consider whether multiple selections or **multiple events are allowed** to take place. Be realistic about whether a single event will occur by itself.
 - A grocery checkout system would be easier to write if only one item was purchased at a time, but that's not realistic.
 - Watch for hidden assumptions about one versus many selections.
 - In the first part of Assignment 4, my first thought was that one driver would be needed, but then I realized that some large shipments might require two or more drivers (multiple trucks); or dangerous goods or long trips might require multiple drivers.
 - Multiple events or selections will often result in loops in a sequence diagram.
- Look for **failure conditions**, and address them. Assume for the MSS that success will happen, and handle failure conditions in the Alternate Scenarios.
 - What happens if a search returns nothing?
 - What happens if credit card payment is rejected?
- Consider **unusual conditions** which aren't failures.
 - Will the system behave differently if a search results in exactly one item returned, instead of several to choose from?
 - Are there cases in which sales tax is charged, and others where it isn't?
 - Are there different kinds of processing rules for different kinds of customers, different types of product, different shipping methods, or different destinations?
- Remember for your system to **communicate with the actor**. The only ways it does this is by updating existing user interfaces or creating new user interfaces.
 - If a search fails, how or where will you tell the actor? You might use a popup window to tell them that.
 - Remember to communicate good outcomes too, such as successful completion of a task. Updating the status section of an existing window can handle non-urgent messages.
 - A summary of the transaction can provide closure for the actor.
- Remember the most obvious part of the scenario – the actual manipulation of data.
 - **Remember CRUD** – make sure the system Creates, Read, Updates, and/or Deletes the appropriate data to **fulfill the objective of the use case**.
 - For example, if you are editing customer data, make sure at some point the system will save the modified customer data.

- Finish strongly! Completion of a use case should **address clean-up issues**. Clean-up of a use case might include steps such as:
 - Make sure the actor has been told the use case completed successfully, often accompanied by a summary of what was accomplished.
 - Close any intermediate screens which are still open.
 - Note that closing a screen means the same as destroying that object.
 - Close the screen which was the foundation for conducting the use case (unless it's the Main Menu). So you might close the New Customer Screen, or wherever the actor navigated to in the beginning of the use case.
- Most use cases should **end with the user in the same place they started**; i.e. facing the main menu. *Exception*: depending on the type of use case, and the actor's role in the organization, there can be use cases where the default assumption is that the actor will perform the same function over and over again. In such cases, the use case should end with the actor ready to start that function again.
 - An example might be a use case to process incoming shipments; the person doing such processing might have a large pile of shipments each day to process, and so they would typically start the process, and work through multiple repeats until all the shipments are processed. The use case description and corresponding sequence diagram should reflect how the business process is typically going to be performed. Make sure each time the function is performed has closure – the actor is told whether the task was completed.

MSS and Extension Numbering

- Make sure to **number steps in the MSS**. Extensions should start with the step number from which they depart, and add a letter for each possible variation. If an extension has more than one step, use more numbers for the steps.
 - For example, if step 9 of a MSS has two possible extensions, the first is 9.a, and the second is 9.b. If the latter has three steps, they would be numbered 9.b.1 through 9.b.3.
 - Number extensions as shown in Fowler. If there is an extension to step 9, the extension 9.a describes the condition under which that extension is used. Then the steps of how the extension is processed are numbered 9.a.1, 9.a.2, 9.a.3, etc.
 - The only reason to use extensions with letters b, c, etc. (such as 9.b and 9.c) is if there are multiple extensions that all start from the same step of the MSS.

Summary of Actor and System Steps in a Use Case

Below are typical actor and system actions in a use case. Possible ways of implementing some of the actor actions are given in parentheses, but only to help understand the type of action which could be implied.

Typical ACTOR Steps in a Use Case
Navigate to a screen
Enter data onto a screen (e.g. fill in a text box)
Select items on a screen (highlighting entries on a list, or selecting radio buttons or check boxes)
Confirm or accept a screen (like clicking on a Submit or Next button)
Exit application

Typical SYSTEM Steps in a Use Case
Display a screen, whether a small pop-up window, or a full size screen (this implies a new screen or window is being created – later will see as an interface class)
Conduct some kind of background processing (invisible to the actor) <ul style="list-style-type: none"> • This is the most complex aspect, since it's unique to the needs of each Use Case (UC) • Performing processing unique to this UC could include following any kind of business logic, such as <ul style="list-style-type: none"> ○ Compute something ○ Manipulate or analyze or query data ○ Create a report ○ Choose the right course of action (make decisions) ○ Validate data ○ Compare things • This type of step might involve making assumptions about what kinds of data are kept by the system <ul style="list-style-type: none"> ○ Like the 'list of possible drivers' cited in Assignment 4 ○ Or the 'security database' assumed to exist in Assignment 4
Any of the basic CRUD functions for managing data <ul style="list-style-type: none"> • Create data (save new customer data) • Read data (get the existing value of some data) • Update data (modify or set the value of an existing variable or record) • Delete data (tends to be rare, unless it's temporary data) Even in an object-oriented system, we still need to do these things; now they will be accomplished using methods which belong to classes. Some part of CRUD is often the main purpose of the use case; don't forget it!!
Output to a screen (modify or add contents to a screen which is already displayed) <ul style="list-style-type: none"> • This is the only way the actor knows what's going on, so this type of step should be relatively common
Call on external systems to perform some kind of processing, or provide data <ul style="list-style-type: none"> • Could include using common Internet applications, such as e-mail, FTP, etc.

Typical MSS sequence

A typical main success scenario involves the following types of activities. This is just an outline, not a literal MSS.

- Actor navigates to the screen where the use case takes place. (This assumes that the use case isn't done from the main menu or opening screen.)
- System displays the desired screen.
 - This could be more than one step if the system has to extract or calculate what data appears on the screen.
- Actor enters data needed for this use case (e.g. new customer information, search criteria, etc.).
- Actor submits the data.
- System validates the data (check for completeness, check for duplicate data, check for logical consistency, etc.).
- If inputs are complex, the system may display a screen to verify the inputs before processing begins. Think of a software install window which tells you what's going to be installed before installation actually begins.
 - If a confirmation screen is used, the actor usually must verify the screen to make it go away.
- System starts processing the use case
 - This may include creating new data records, performing calculations, communicating with external systems, etc. Think of the CRUD functions, and what objects are being created, read (get), updated (modified), or deleted.
- System provides a response to the actor to indicate successful completion of the function.

Typical extensions are needed when

- Input data failed to validate. The actor needs to reenter data until it passes validation or they give up.
- A search yielded no or too few results. This could result in just an error message, or a chance to change the search criteria.
- The use case processing failed for some reason (couldn't save changes to data, couldn't reach external system, etc.).

Handling Conditionals and Failures in a Use Case

In creating a sequence or collaboration diagram, there are three types of conditionals that appear frequently. All are handled somewhat differently for the sequence diagram depending on whether the condition results in one message being processed, or several.

The types of conditionals are:

1. **Mutually exclusive options** – this is when there are two or more possible ways to process a use case, and exactly one of them is chosen each time the use case is performed.
 - Examples include choosing a type of payment (cash, credit card, money order, etc.), choosing Domestic versus International shipping, etc.
2. **Failure conditions** – such as the above extensions; how do you handle it when a normal step isn't successful?
 - Examples include data didn't validate, a search yielded insufficient useful results, couldn't save data, couldn't contact an external system, etc.
3. **Optional processing steps** – this is for handling steps in a MSS which aren't always performed, but don't indicate anything "wrong".
 - Examples could include adding sales tax to a purchase, or gift wrapping a purchase.

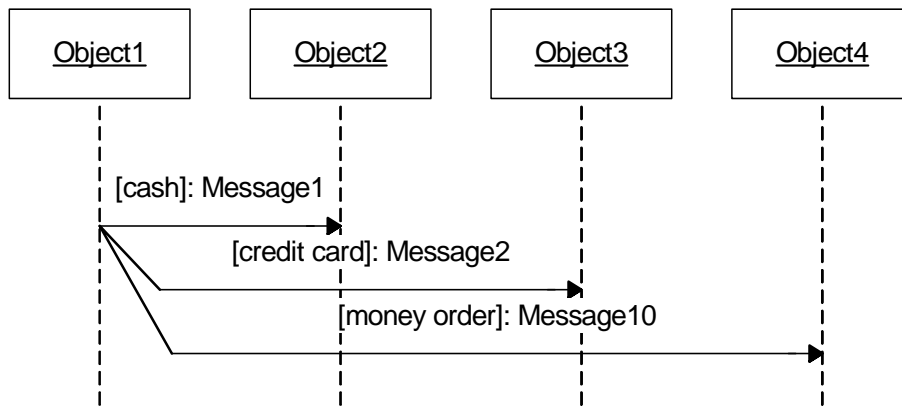
Our tools for showing conditionals include

- The conditional prefix for a single message,
- Using multiple messages from a single point,
- Using a region for 'alternate' processing options (the 'alt' tag),
- Using a region for optional processing (the 'opt' tag) and
- Using loops (the 'loop' tag).

Mutually exclusive options

For One Message Per Option

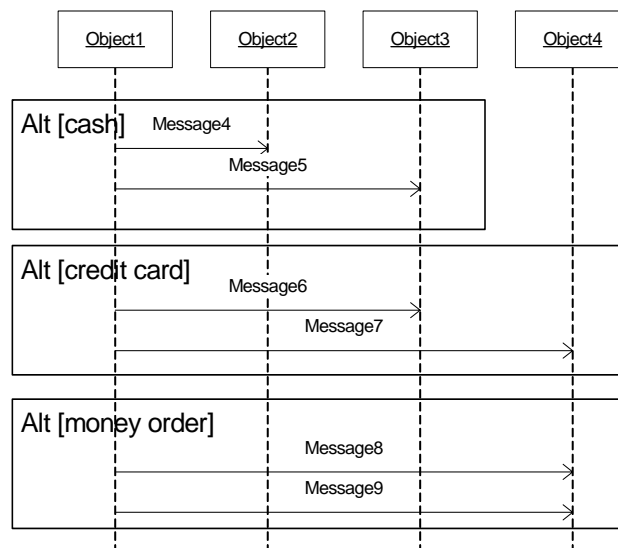
If each possible option has only one message that needs to be performed, then the split message can be used.



Notice that this is the *ONLY* situation where multiple messages start on the same point.

For Multiple Messages Per Option

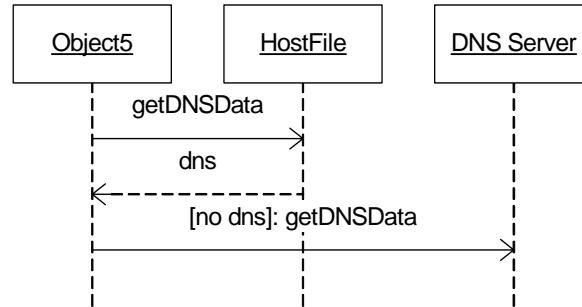
If each possible option requires more than one message to be performed, then the 'alt' box can be used to show how each type of processing takes place. (See page 59 of UML Distilled for a full list of special types of boxes.) The 'alt' condition is kind of like a Case statement – it's assumed the conditions are mutually exclusive, but one set of alt conditions will be performed each time.



Failure conditions

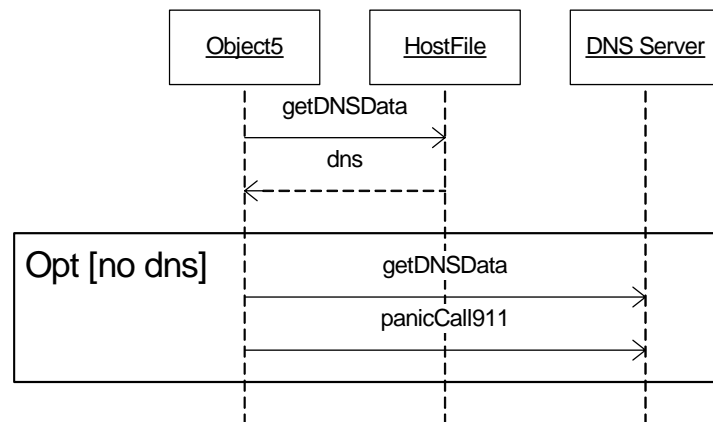
One Message After A Failed Condition

If one message is needed to handle a failure condition, then the condition on that message can be the failure condition. For this example, if DNS information wasn't received from the Host file, then it is obtained from a DNS Server (external system).

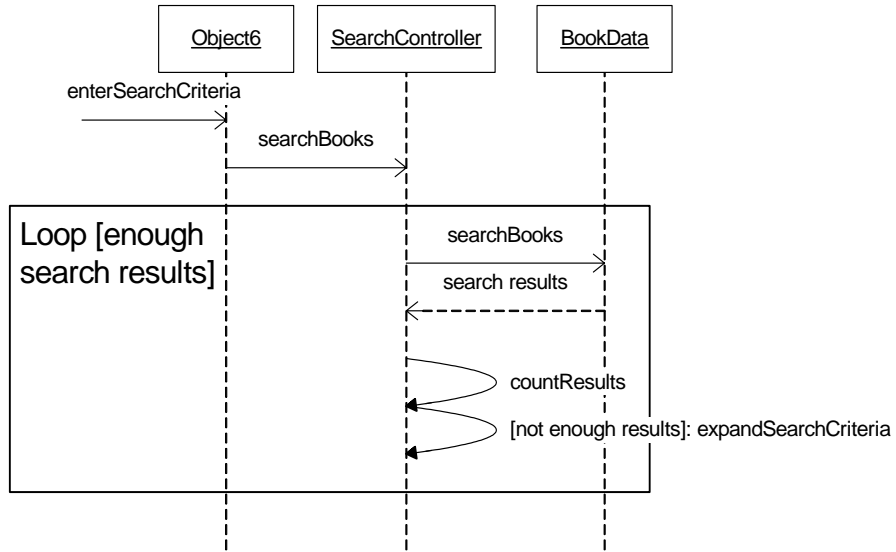


Multiple Messages After A Failed Condition

If you need to have two or more steps after a failure condition is reached, use an 'opt' box. The contents of an opt box will only be executed if the condition is met; otherwise the box will be skipped over.



Another, somewhat sneakier, approach is to use a loop condition until the failure condition is NOT met, then proceed. For example, if a search needs to be repeated until enough results have been obtained, then it could look like this:

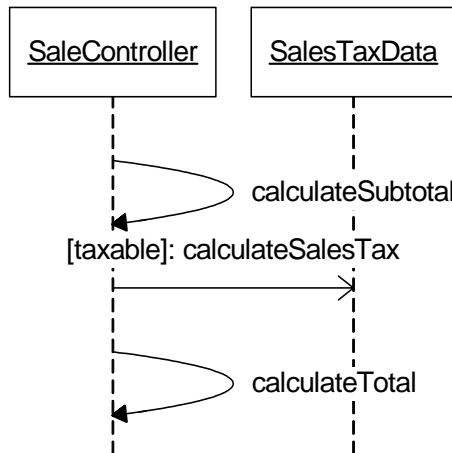


This assumes that the SearchController object has logic to be able to expand the search criteria; hence you don't have to ask the actor for new criteria.

Optional processing steps

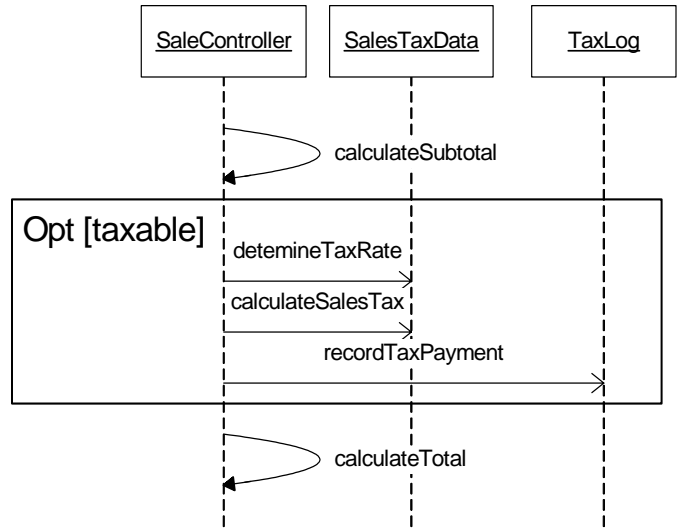
Single Optional Processing Step

If one step needs to be performed for optional processing, a single message with that condition can be used, much like a single failure condition.



Multiple Optional Processing Steps

For more than one optional processing step, use an ‘opt’ box. Again, this is like processing for failure conditions.



These tools can be nested – for example, a loop can contain a series of alt tags. Below is a summary of when these tools are used

Tool	Number of choices	Number of times the tool is evaluated
Conditional prefix on one message	1 – Either execute the message, or not.	Once
Multiple messages from a single point	2 or more – one of the messages will be executed, the rest ignored	Once
Alt tag	2 or more – each possible choice gets an Alt tag labeled with its condition	Once
Loop tag	1 – there is one set of messages that is executed some number of times	Many – loop until 1) a condition is true, or 2) a fixed number of times
Opt tag	1 – the contents of the Opt tag are either executed once, or not at all	Once