

# Systems Analysis and Design Behavioral Modeling

---

CIS641

Erik Fredericks // [frederer@gvsu.edu](mailto:frederer@gvsu.edu)

*Adapted from materials provided by Gregory Schymik and the textbook  
(Systems Analysis and Design 5th/6th Ed.)*

# Outline!

More diagrams!

- Behavioral this time!

Analysis!

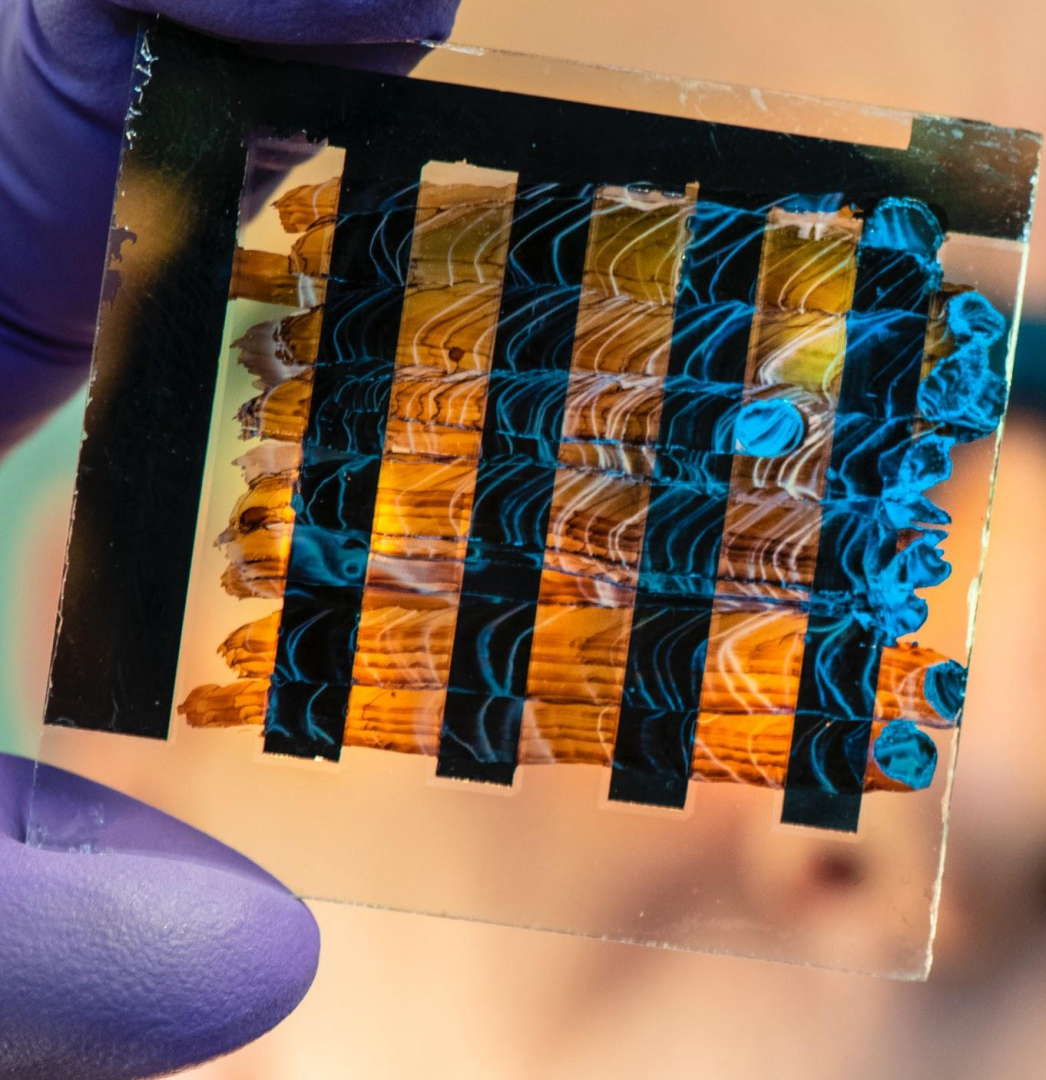
V&V!





TBD





# What does behavioral modeling *do* for us?

We've talked about *functional* and *structural* models thus far...

# Behavioral modeling

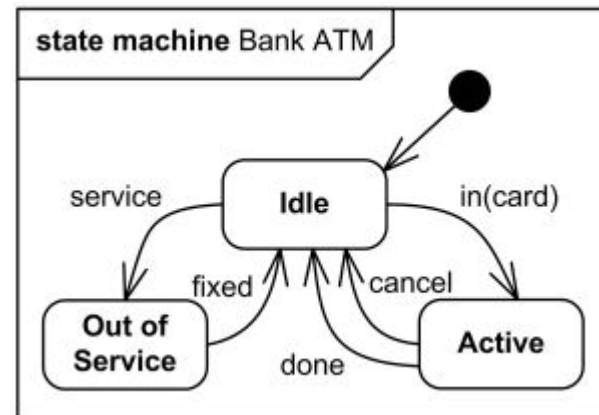
Describes the *internal* behavior of a system

Model types:

- Business process details identified by use cases
  - Interaction diagrams (sequence / communications)
  - Shows **object collaboration** to provide use case functionality
- Changes in data
  - Behavioral state machines

Still not focusing on implementation!

- High-level, dynamic view



# Behavioral models

Set of use cases **supported by collaborating objects**

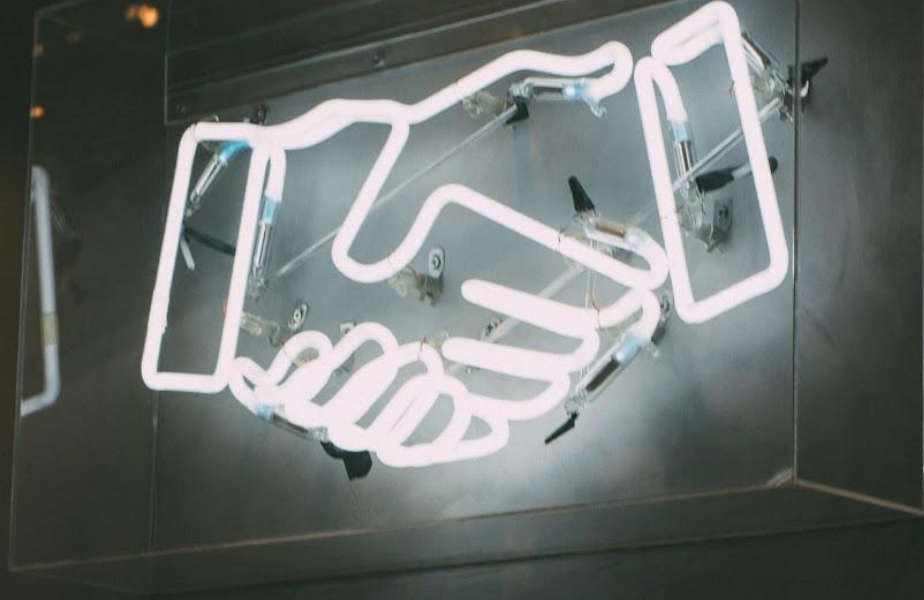
- Organizing/defining software
- Depict interaction/collaboration of business processes

Behavioral model creation is **iterative**

- May impact other models!

Difference to *structural* models?





Keyword of the day: **collaboration**

# Difference to *structural* models?

## **Structural models:**

- Components of system
- Desired structures/operations/etc.
- System framework

## **Behavioral models:**

- Show *dynamic* view of the system
- Flow of data between objects
- Behaviors

# Interaction diagrams

**Objects** → instantiation of a class

- Patient is a class
- Mary Wilson is an instantiation of the patient class (object)

**Attributes** → characteristics of a class

- Patient class: name, address, phone, etc.

**Operations** → the behaviors of a class, or an action that an object can perform

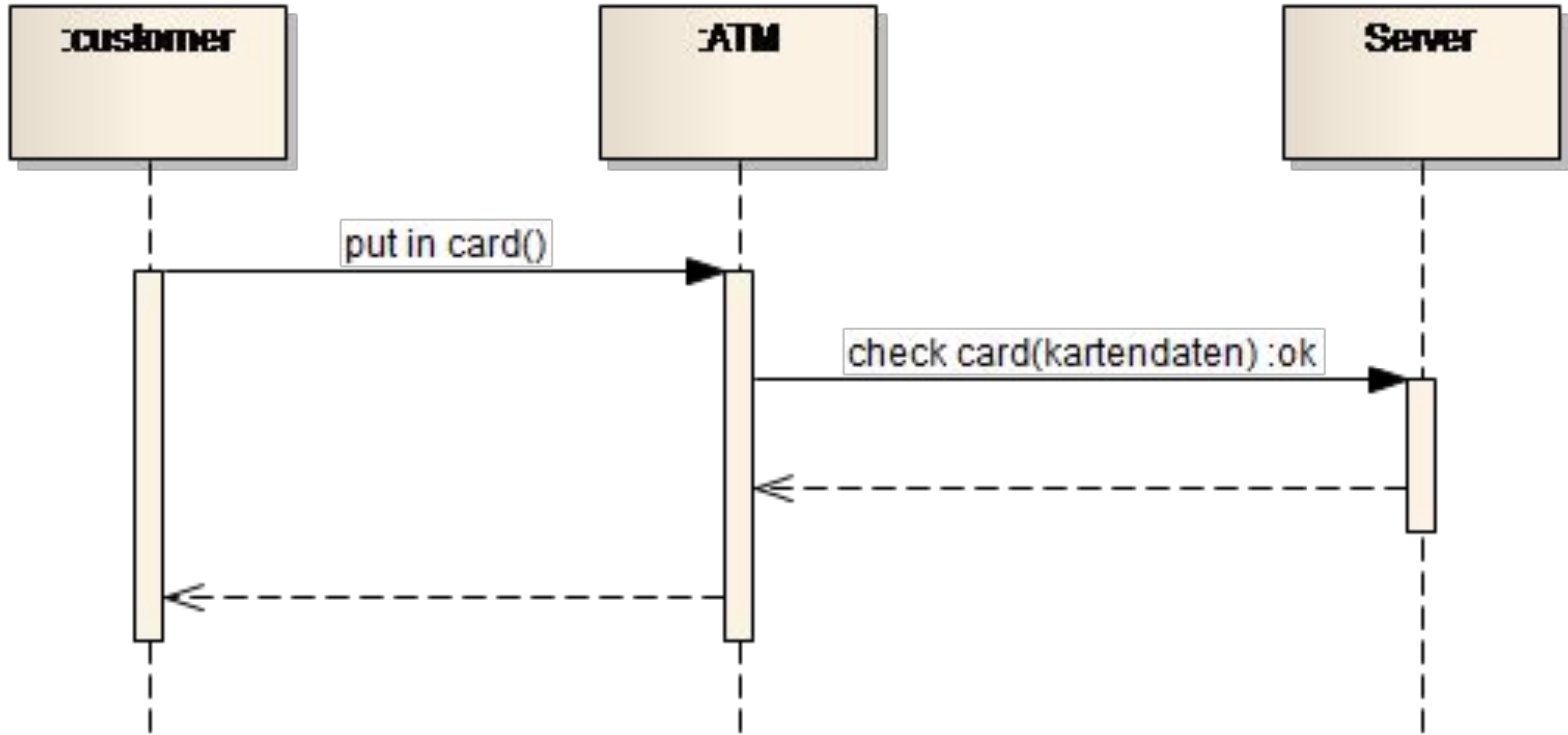
**Messages** → information sent to objects to tell them to execute one of their behaviors

- A function call from one object to another

**Types**

- Sequence Diagrams → emphasize message sequence
- Communication Diagrams → emphasize message flow

# Takeaway: **communication between objects**



# Sequence diagrams

Illustrate the objects that participate in a **single use-case**

A dynamic model

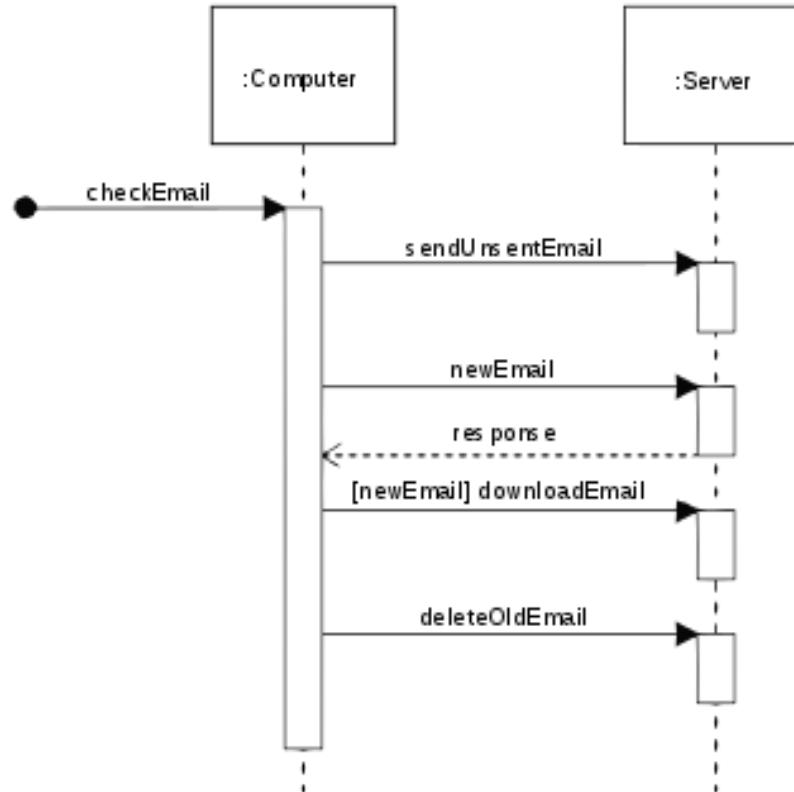
- Shows the sequence of messages that pass between objects
- Aid in understanding real-time specifications and complex use-cases


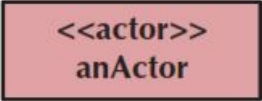
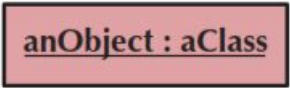

Generic diagram shows all scenarios for a use-case


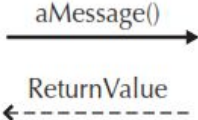
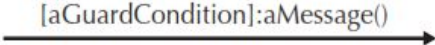

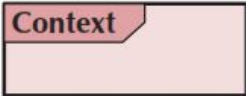
Instance diagrams show a single scenario



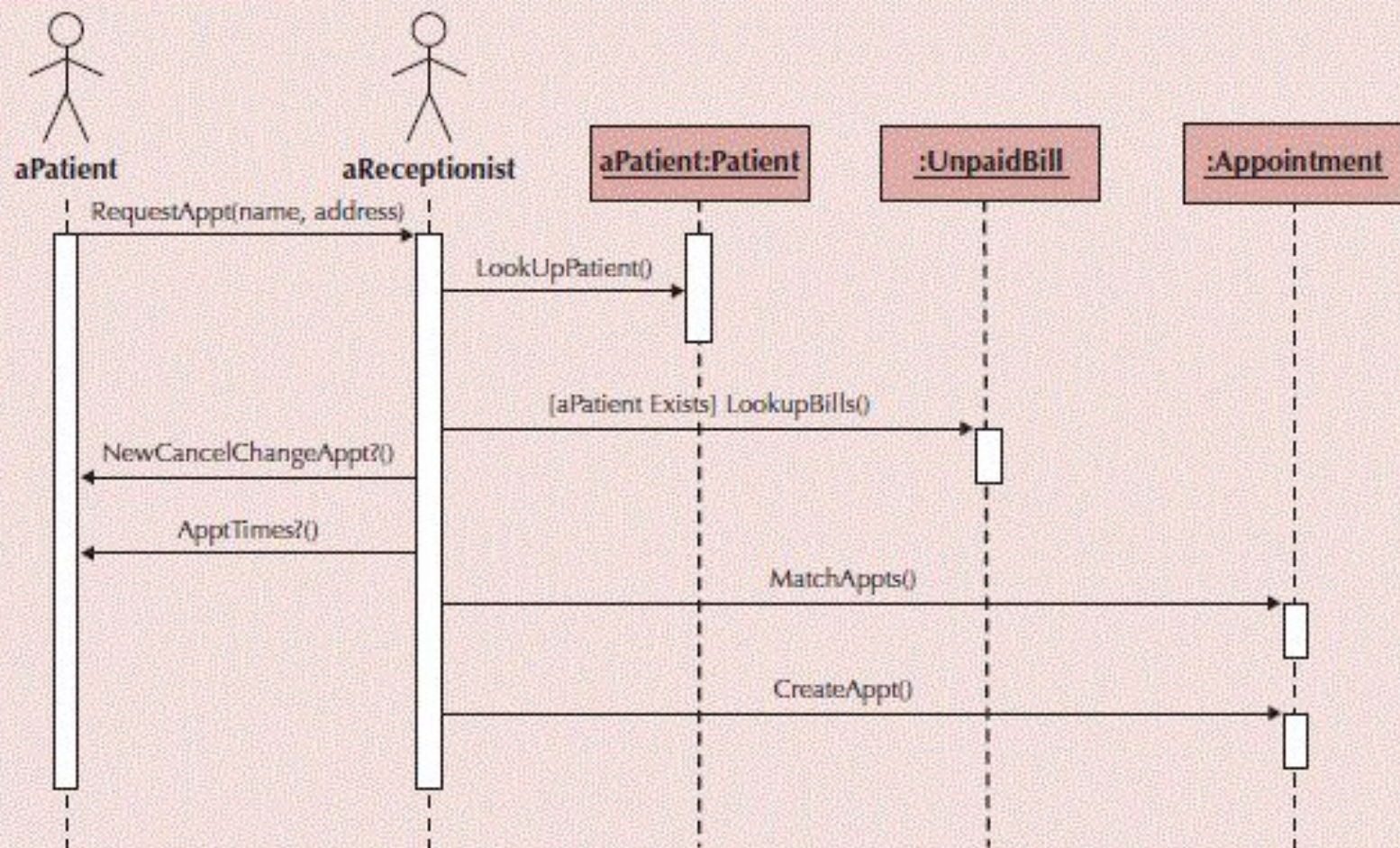
# Takeaway: **sequential interaction**



Term and Definition	Symbol
<p><b>An actor:</b></p> <ul style="list-style-type: none"> <li>■ Is a person or system that derives benefit from and is external to the system.</li> <li>■ Participates in a sequence by sending and/or receiving messages.</li> <li>■ Is placed across the top of the diagram.</li> <li>■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with &lt;&lt;actor&gt;&gt; in it (alternative).</li> </ul>	 <p><b>anActor</b></p> 
<p><b>An object:</b></p> <ul style="list-style-type: none"> <li>■ Participates in a sequence by sending and/or receiving messages.</li> <li>■ Is placed across the top of the diagram.</li> </ul>	
<p><b>A lifeline:</b></p> <ul style="list-style-type: none"> <li>■ Denotes the life of an object during a sequence.</li> <li>■ Contains an X at the point at which the class no longer interacts.</li> </ul>	

<p><b>An execution occurrence:</b></p> <ul style="list-style-type: none"> <li>■ Is a long narrow rectangle placed atop a lifeline.</li> <li>■ Denotes when an object is sending or receiving messages.</li> </ul>	
<p><b>A message:</b></p> <ul style="list-style-type: none"> <li>■ Conveys information from one object to another one.</li> <li>■ A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.</li> </ul>	
<p><b>A guard condition:</b></p> <ul style="list-style-type: none"> <li>■ Represents a test that must be met for the message to be sent.</li> </ul>	
<p><b>For object destruction:</b></p> <ul style="list-style-type: none"> <li>■ An X is placed at the end of an object's lifeline to show that it is going out of existence.</li> </ul>	
<p><b>A frame:</b></p> <ul style="list-style-type: none"> <li>■ Indicates the context of the sequence diagram.</li> </ul>	

# sd Make Appt Use Case



# A PROCESS (for building sequence diagrams)

Set the context

Identify actors and objects that interact in the use-case scenario

Set the lifeline for each object

Add messages by drawing arrows

- Shows how they are passed from one object to another
- Include any parameters in parentheses
- Obvious return values are excluded



# A PROCESS (for building sequence diagrams)

Add execution occurrence to each object's lifeline

Validate the sequence diagram

- Ensures that it depicts all of the steps in the process



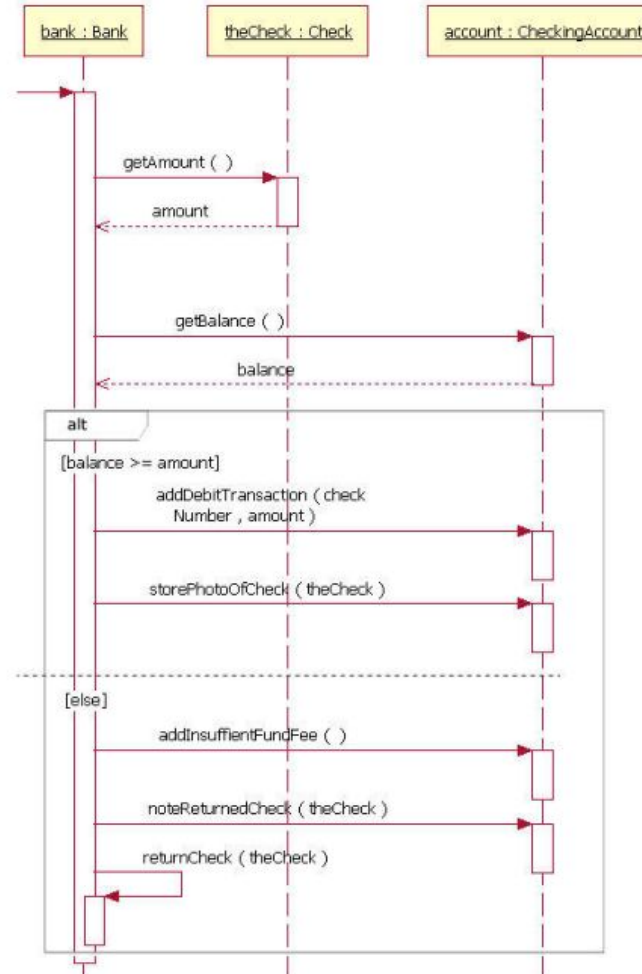
# Check processing

As an example to show how an alternative combination fragment is read, Figure 8 shows the sequence starting at the top, with the bank object getting the check's amount and the account's balance. At this point in the sequence the alternative combination fragment takes over. Because of the guard `[balance >= amount]`, if the account's balance is greater than or equal to the amount, then the sequence continues with the bank object sending the `addDebitTransaction` and `storePhotoOfCheck` messages to the account object. However, if the balance is not greater than or equal to the amount, then the sequence proceeds with the bank object sending the `addInsufficientFundFee` and `noteReturnedCheck` message to the account object and the `returnCheck` message to itself. The second sequence is called when the balance is not greater than or equal to the amount because of the `[else]` guard. In alternative combination fragments, the `[else]` guard is not required; and if an operand does not have an explicit guard on it, then the `[else]` guard is to be assumed.

# Check processing

Alternative combination fragments are not limited to simple “if then else” tests. There can be as many alternative paths as are needed. If more alternatives are needed, all you must do is add an operand to the rectangle with that sequence’s guard and messages.

Figure 8. A sequence diagram fragment that contains an alternative combination fragment



# NOW GO FORTH AND BUILD A SEQUENCE DIAGRAM

- 1) Pick a **single** use case from the last in-class work we had
- 2) Build me a sequence diagram!
- 3) (15 minutes then we'll *chat*)

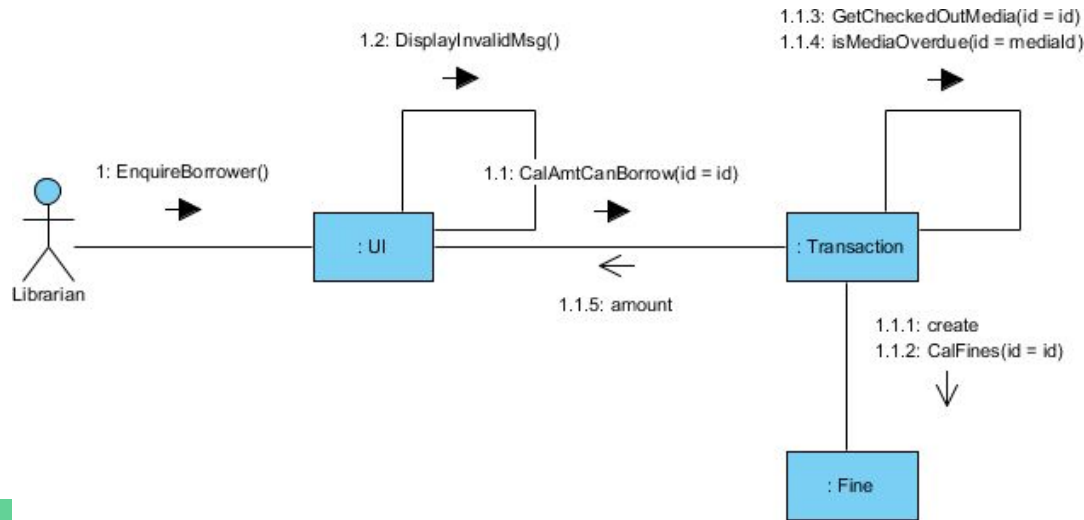



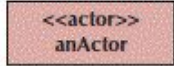


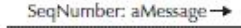

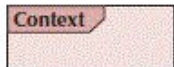
# Communication diagrams

Depict the dependencies among the objects

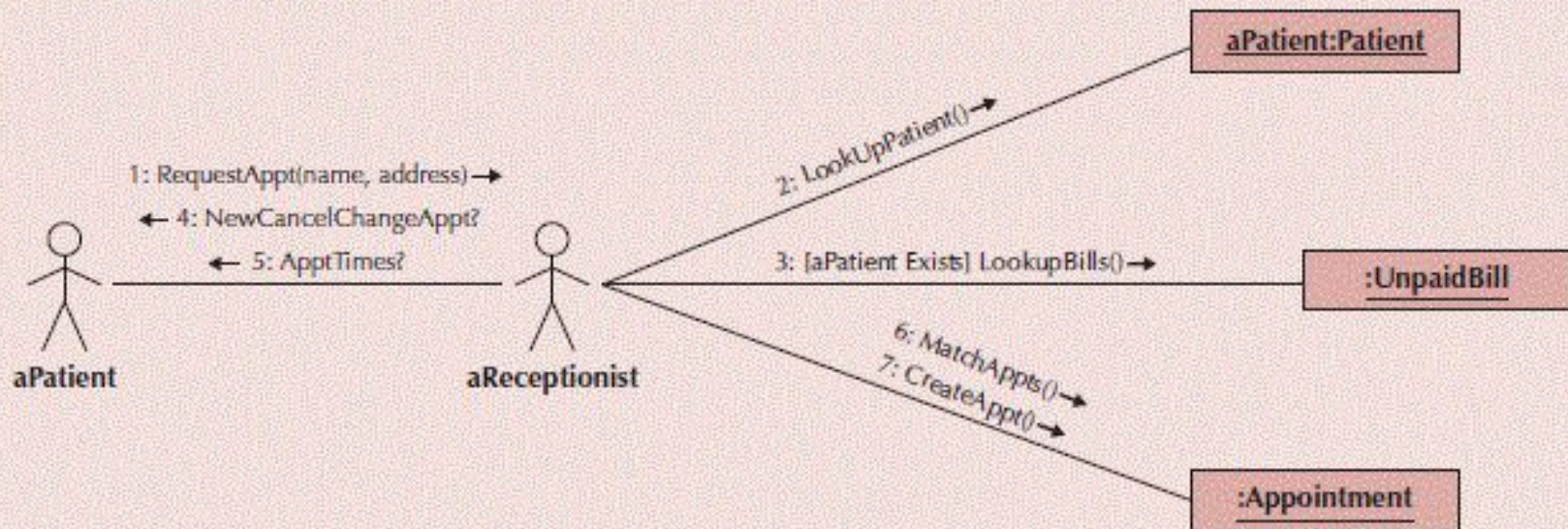
An object diagram that shows message passing relationships

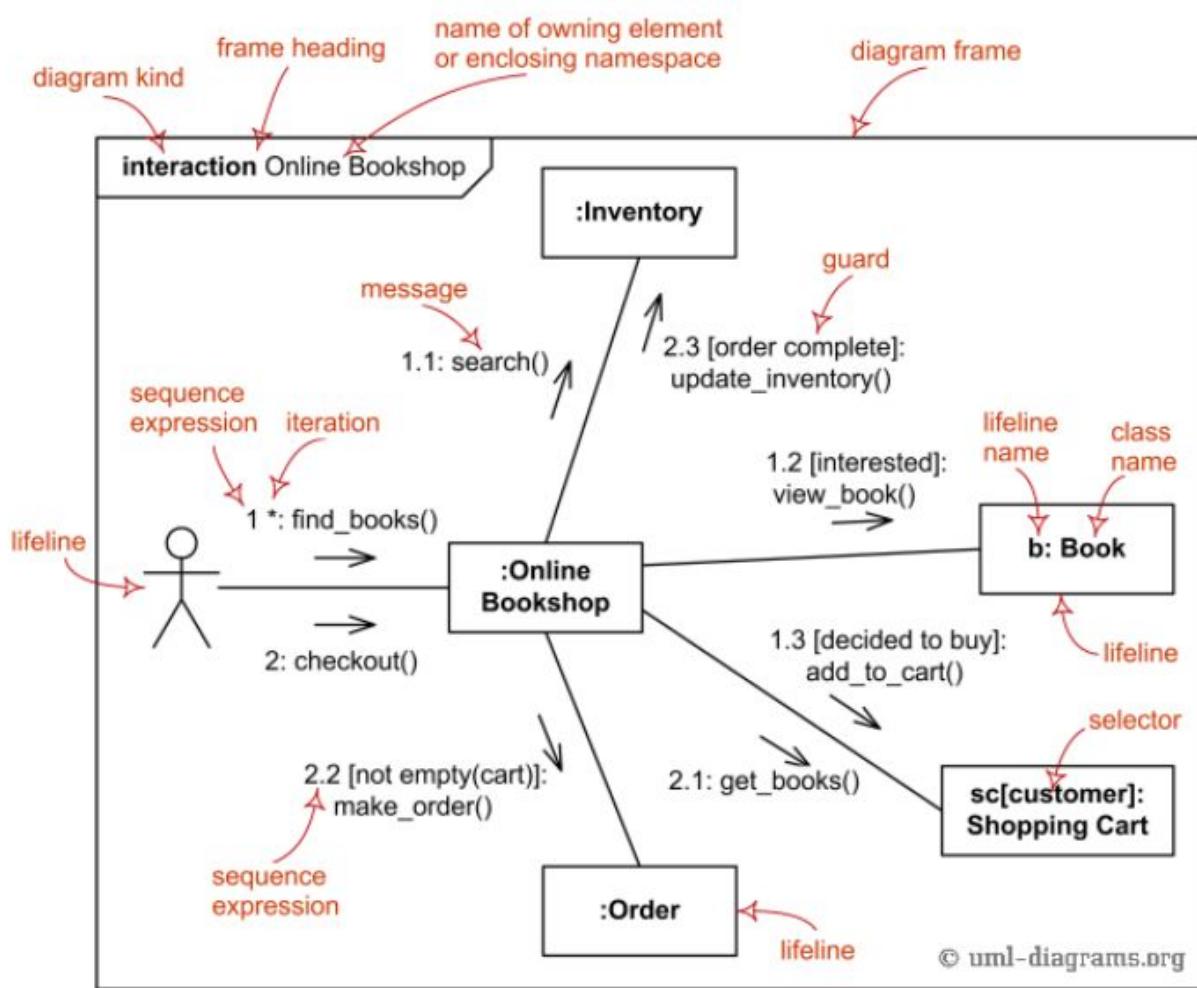
Emphasize the flow through a set of objects



Term and Definition	Symbol
<p><b>An actor:</b></p> <ul style="list-style-type: none"> <li>■ Is a person or system that derives benefit from and is external to the system.</li> <li>■ Participates in a collaboration by sending and/or receiving messages.</li> <li>■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with &lt;&lt;actor&gt;&gt; in it (alternative).</li> </ul>	 
<p><b>An object:</b></p> <ul style="list-style-type: none"> <li>■ Participates in a collaboration by sending and/or receiving messages.</li> </ul>	
<p><b>An association:</b></p> <ul style="list-style-type: none"> <li>■ Shows an association between actors and/or objects.</li> <li>■ Is used to send messages.</li> </ul>	
<p><b>A message:</b></p> <ul style="list-style-type: none"> <li>■ Conveys information from one object to another one.</li> <li>■ Has direction shown using an arrowhead.</li> <li>■ Has sequence shown by a sequence number.</li> </ul>	
<p><b>A guard condition:</b></p> <ul style="list-style-type: none"> <li>■ Represents a test that must be met for the message to be sent.</li> </ul>	
<p><b>A frame:</b></p> <ul style="list-style-type: none"> <li>■ Indicates the context of the communication diagram.</li> </ul>	

# sd Make Appt Use Case





The major elements of UML communication diagram.

# Building communications diagrams

Set the context

Identify objects, actors and associations between them

Lay out the diagram

Add the messages

Validate the model



# Behavioral state machines

Objects may change state in response to an event

Different states are captured in this model

- Shows the different states through which a single object passes during its life
- May include the object's responses and actions

Example: patient states

- New patient → has not yet been seen
- Current patient → is now receiving treatment
- Former patient → no longer being seen or treated

Typically used only for complex objects

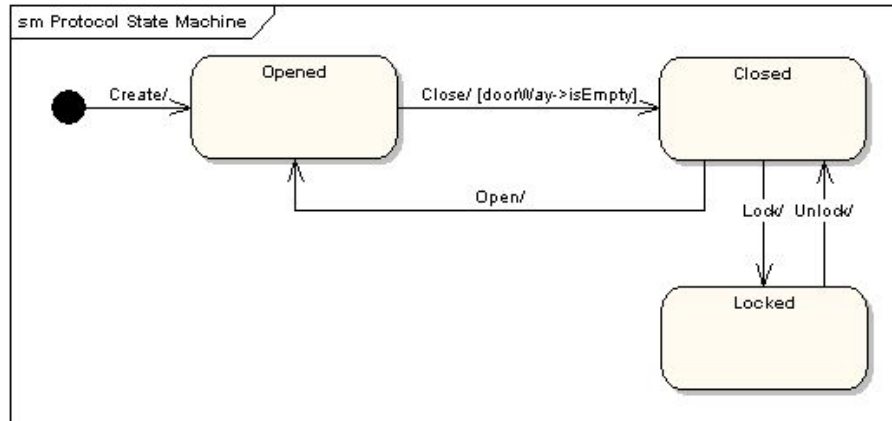
# State machine components






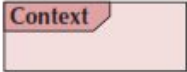
**States** → values of an object's attributes at a point in time

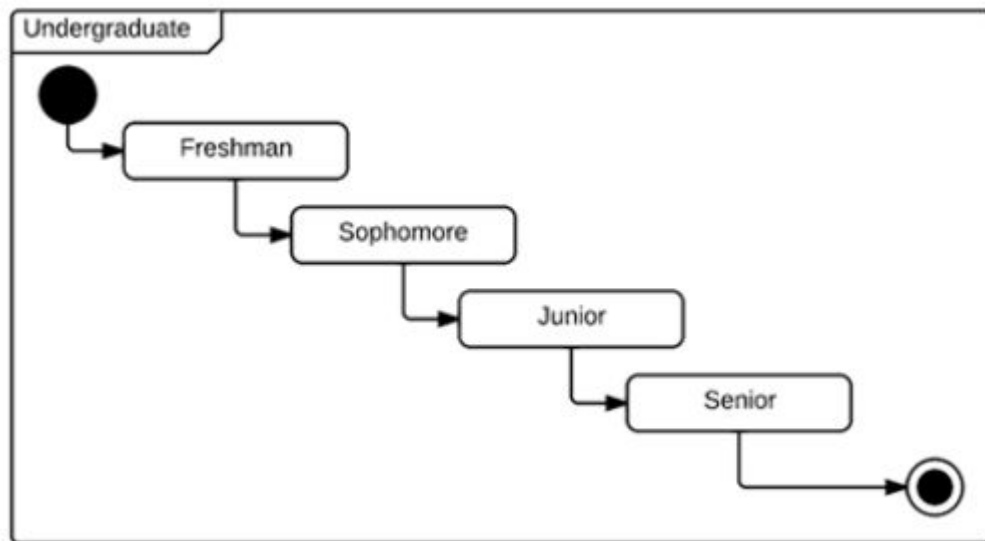
**Events** → the cause of the change in values of the object's attributes

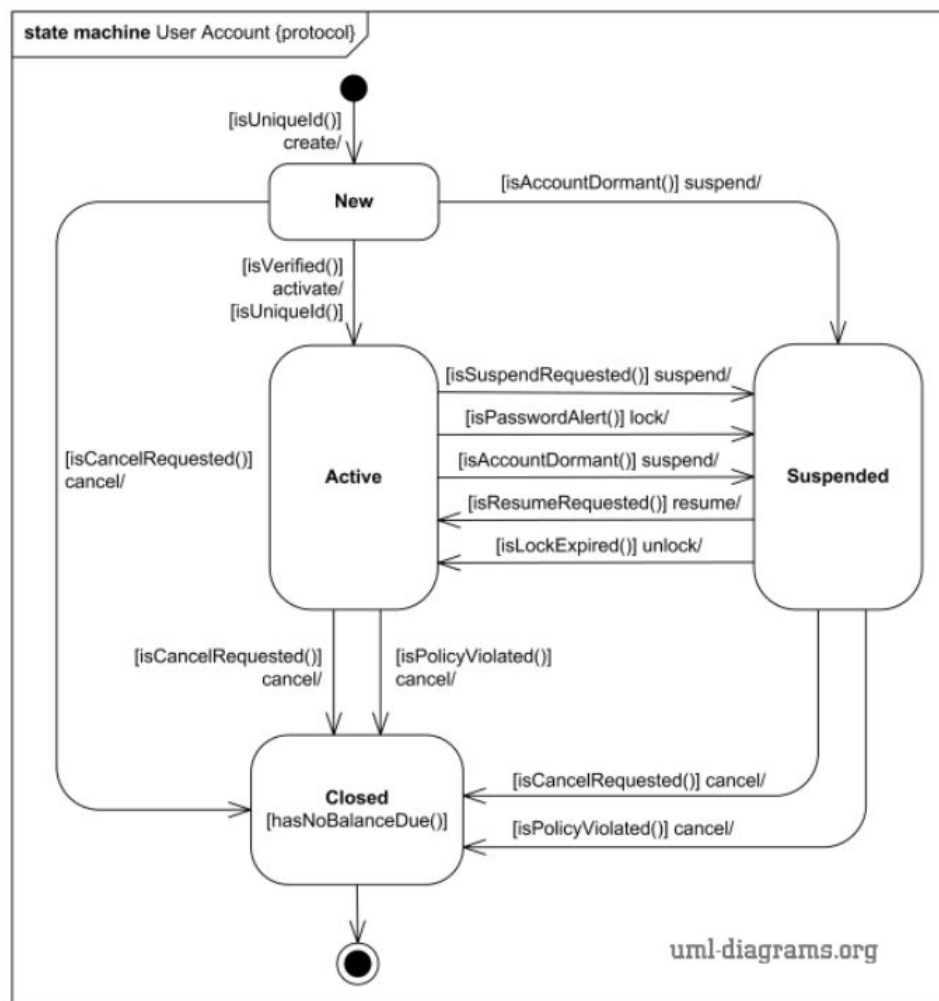
**Transitions** → movement of an object from one state to another

- May include a guard condition to flag that a condition is true and allow the transition

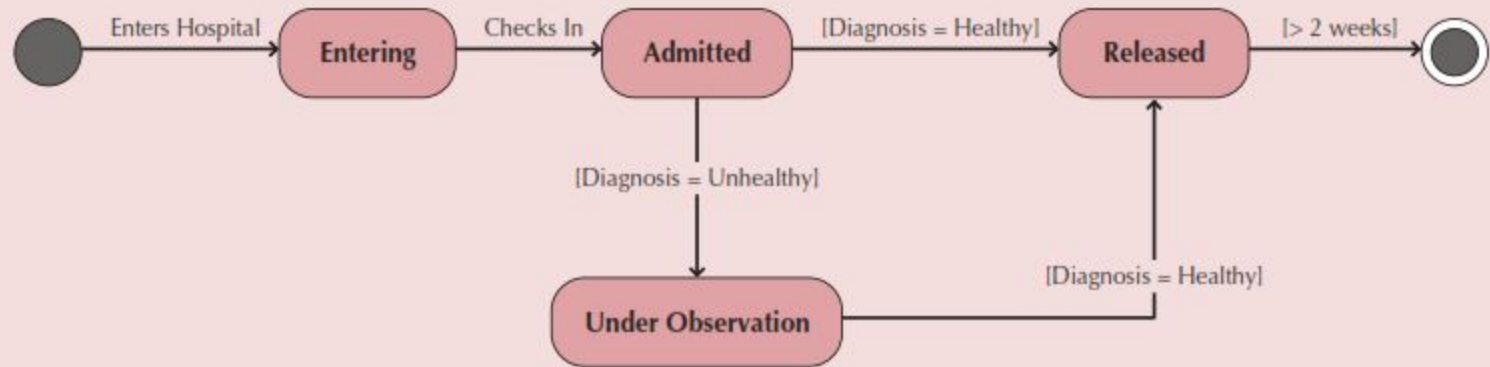


Term and Definition	Symbol
<b>A state:</b> <ul style="list-style-type: none"> <li>■ Is shown as a rectangle with rounded corners.</li> <li>■ Has a name that represents the state of an object.</li> </ul>	
<b>An initial state:</b> <ul style="list-style-type: none"> <li>■ Is shown as a small, filled-in circle.</li> <li>■ Represents the point at which an object begins to exist.</li> </ul>	
<b>A final state:</b> <ul style="list-style-type: none"> <li>■ Is shown as a circle surrounding a small, filled-in circle (bull's-eye).</li> <li>■ Represents the completion of activity.</li> </ul>	
<b>An event:</b> <ul style="list-style-type: none"> <li>■ Is a noteworthy occurrence that triggers a change in state.</li> <li>■ Can be a designated condition becoming true, the receipt of an explicit signal from one object to another, or the passage of a designated period of time.</li> <li>■ Is used to label a transition.</li> </ul>	
<b>A transition:</b> <ul style="list-style-type: none"> <li>■ Indicates that an object in the first state will enter the second state.</li> <li>■ Is triggered by the occurrence of the event labeling the transition.</li> <li>■ Is shown as a solid arrow from one state to another, labeled by the event name.</li> </ul>	
<b>A frame:</b> <ul style="list-style-type: none"> <li>■ Indicates the context of the behavioral state machine.</li> </ul>	





## Patient



# State machine creation (guidelines)

Use only for complex objects

Draw the initial state in the upper left corner

Draw the final state in the bottom right corner

Use simple, but descriptive names for states

Look out for “black holes” and “miracles”

Ensure guard conditions are mutually exclusive

Ensure transitions are associated with messages and operations

# State machine creation (more of)

Set the context

Identify the states of the object

- Initial
- Final
- Stable states during its lifetime

Lay out the diagram—use a left to right sequence

Add the transitions

- Identify the triggers (events that cause the transition)
- Identify the actions which execute
- Identify the guard conditions

Validate the model—ensure all states are reachable



# Let's build a state machine for a phone app

(You're probably sick of seeing my little project demo at this point)

## What is this app?

- Game changing, dollar generating, etc.
- **Resource for managing IoT devices in a home**
  - Bluetooth/WiFi devices strewn throughout the home
- What states do we have?
- What transitions will happen?

# Just ... just go build a model

Generate a state machine for your term projects

Should have at least **4 states**

- Consider what causes each transition to occur

# CRUDE analysis

CRUD + E! (Create/Read/Update/Delete/Execute)

Helps to identify object collaborations

Labels object interaction in 5 possible ways:

- **Create** → can one object create another?
- **Read** → can one object read the attributes of another?
- **Update** → can one object change values in another?
- **Delete** → can one object delete another object?
- **Execute** → can one object execute the operations of another?

Utilizes a matrix to represent objects and their interactions

Most useful as a system-wide representation



[illegible]

CRUD MATRIX							
Calling Item	Item Type	COUNTER	CUSTOMER	EMPLOYEE	PRODUCT	SALES_ORDER_ITEMS	SALES_ORDER
ALL_SALES_ORDER_ITEMS_DETAILS	View			R	R	R	R
ALL_PRODUCTS	View				R		
ALL_ORDERS_BY_EMPLOYEE	View						R
EMPLOYEE_COUNT	Trigger			R			
TRG_ORDER	Trigger				R		
CUSTOMERS.InsertCustomer	Procedure		C				
CUSTOMERS.UpdateCustomerName	Procedure		U				
CUSTOMERS.DeleteCustomerById	Procedure		D				
EMPLOYEES.OrdersByEmployee	Procedure		R	R		R	R
PRODUCTS.ProductsByCustomer	Procedure		R		R	R	R
PRODUCTSBYCUSTOMER	Procedure		R		R	R	R
pfc_updateprep	Event	RU					
itemchanged	Event				R		
dempfc.pbl.d_tab_customer	Datawindow Object		RU				
dempfc.pbl.d_tab_sales_order	Datawindow Object		R	R	R	R	R
dempfc.pbl.d_ff_sales_order	Datawindow Object		R	R			RU
dempfc.pbl.d_tab_employee	Datawindow Object			RU			

Figure 1 <http://www.visual-expert.com/EN/visual-expert-blog/posts-2017/blog-post-crud-matrix.html>

# V&V behavioral models

1. Actors must be consistent between models
2. Messages on sequence diagrams must match associations on communication diagrams
3. Every message on a sequence diagram must appear on an association in a communication diagram

# V&V behavioral models

4. Guard conditions on a sequence diagram must appear on a communication diagram
5. Sequence of messages must correspond to the top down ordering of messages being sent
6. State transitions must be associated with a message on a sequence diagram
7. Entries in a CRUDE matrix imply messages being sent