


**OO Design & Development II:  
State Modeling**  
CS577b  
Fall 2001

Ed Colbert  
USC Center for Software  
Engineering

1 2/14/2002



## Goal of Presentation

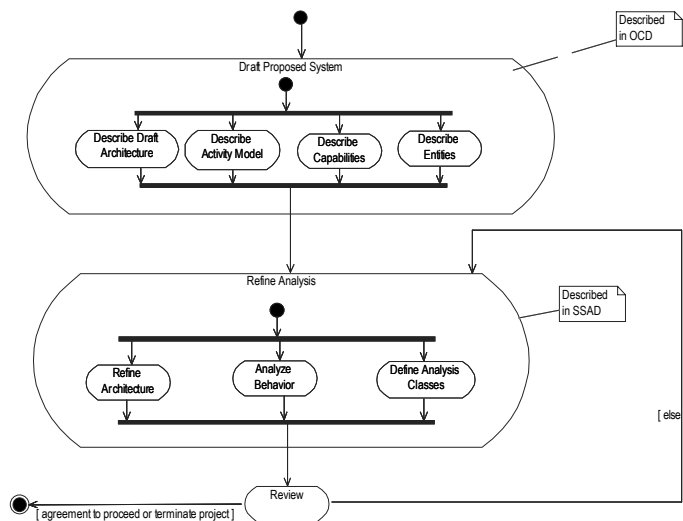
- Understand how to perform Object-Oriented Design & Development
  - Using
    - MBASE
    - Object-oriented techniques
    - RUP
    - Rational Rose
- Understand how to document behavior using Statecharts
- Presentation is part 2 of 4 lectures
  - 1<sup>st</sup> lecture was review of Statechart notation

OO D&D II, CS577b, Spring 2002 2 2/14/2002

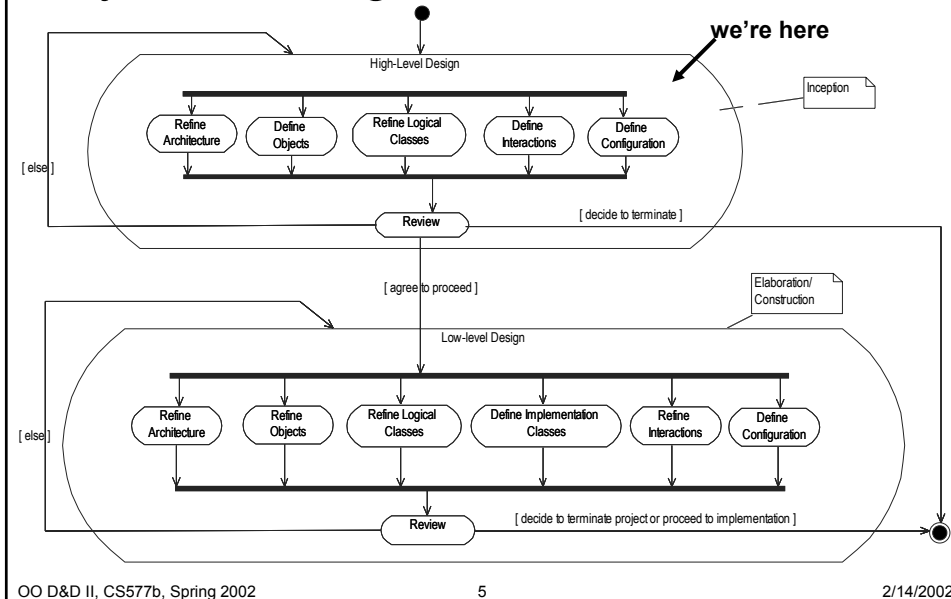
# Outline

- **Review of Process Review**
- **Guidelines for Creating Statecharts**
  - Defining Behavior
  - Exit Criteria
  - Guidelines
- **Statechart Examples**
- **Conclusions**

# System Analysis Process Overview



# System Design Process Overview



# Life Cycle Objectives (LCO) Guidelines

- **General**
  - Less structured, with information moving around
  - Focus on the strategy or "vision"
    - e.g., for Operational Concept Description & Life Cycle Plan
  - May have some mismatches
    - indicating unresolved issues or items
  - No need for complete forward & backward traceability
  - May still include "possible" or "potential" elements
    - e.g., Entities, Components, ...
  - Some sections could be left as TBD
    - Particularly Construction, Transition, and Support plans
- **System Analysis/Design**
  - Focus on
    - High-level architecture
    - High-risk or complex behaviors

## Life Cycle Architecture (LCA) Guidelines

- General
  - More formal
  - Solid tracing upward & downward
  - No major unresolved issues or items
  - Closure mechanisms identified for any unresolved issues or items
    - e.g., "detailed data entry capabilities will be specified once the Library chooses a Forms Management package on February 15"
  - No TBDs expect possibly within Construction, Transition, & Support plans
  - Basic elements from Life Cycle Plan are indicated within Construction, Transition, & Support plans
  - No "possible" or "potential" elements
    - e.g., Entities, Components, ...
  - No more superfluous, unreferenced items
    - Each element should reference or be referenced by another element
    - Elements not referenced should be eliminated or documented as irrelevant
- System Analysis/Design
  - Stable Architecture
  - Design of objects & classes that implement high-risk or complex behaviors

## Initial Operating Capability (IOC) Guidelines

- General
  - Complete tracings among models & delivered software
    - e.g. comments in code trace to SSAD design elements
  - MBASE models consistent with delivered system
    - e.g. "as built" OCD, SSRD, SSAD, etc. models
    - Not necessarily complete
  - Core system capability requirements have been implemented & tested
  - At least one construction interaction
  - Complete set of CTS plans & reports consistent with development
- System Analysis/Design
  - Stable Architecture
  - For all capabilities in iteration
    - Design of objects & classes that implement capabilities
    - Implementation models sufficient to code

# Outline

- Review of Process Guidelines
- **Guidelines for Creating Statecharts**
  - Defining Behavior
  - Exit Criteria
  - Guidelines
- Statechart Examples
- Conclusions

# Defining Behavior

- Purpose:
  - Describe state logic for classifiers with complex behavior based on state
- Artifacts:
  - Statechart model for classifiers of system, components, or objects
- Note:
  - Concurrent to design of
    - Component Model
    - Class Model
    - Interaction Model

# Outline

- Review of Process Guidelines
- **Guidelines for Creating Statecharts**
  - **Defining Behavior**
  - Exit Criteria
  - Guidelines
- Statechart Examples
- Conclusions

# Outline

- Review of Process Guidelines
- Guidelines for Creating Statecharts
  - Defining Behavior
  - **Exit Criteria**
  - Guidelines
- Statechart Examples
- Conclusions

## Exit Criteria

- UML Statechart Notation
- Set of diagrams showing behavior of either
  - Classifiers with “interesting” behavior
  - All classifiers
- Note:
  - Project must determine whether developing behavior for all objects or only “interesting” objects
  - Problem: often can’t tell if behavior is interesting until develop Statechart
    - Clues
      - Active class
      - Non-datatypes shared by active objects

## Exit Criteria for Behavior of A Class

- Every message to class instance, as shown in Interaction Diagram(s)
  - Operation name & actual parameters must appear as event-signiture of  $\geq 1$  transition (internal or external)

# Exit Criteria for Behavior of A Class

- Every message from class instance, as shown Interaction Diagram(s)
  - If operation isn't functional
    - i.e., operation doesn't return a value
    - Message must appear  $\geq 1$  send-clause(s)
  - If operation is functional
    - Messages name must appear
      - In guard-condition of  $\geq 1$  transition
        - Extension of UML semantics
      - As actual parameter of another message
      - As right-hand expression of assignment in action-expression

# Exit Criteria for Behavior of A Class (cont.)

- Statechart is valid
  - Correct transition occurs from every state, for every possible requested operation & all guard-conditions
  - Correct actions performed for all sequences of messages from "client" objects
  - Results are what "client" objects expect
    - As shown in Interaction Diagrams
      - No states where object can't transition to another state
        - Except termination state
      - No premature termination
        - Doesn't enter termination state until expected conditions are met

## Exit Criteria for Behavior of A Class (cont.)

- Class Model updated to reflect concurrency semantics for each class
  - Class marked
    - Active
    - Passive
  - Operations of Passive Classes marked as
    - Sequential
    - Guarded
    - Concurrent

## Exit Criteria for Behavior of A Class (cont.)

- Interaction Diagrams updated to concurrency semantics for each class
  - Messages marked as
    - Synchronous
    - Asynchronous
    - Timed
    - Balking

# Outline

- Review of Process Guidelines
- Guidelines for Creating Statecharts
  - Defining Behavior
  - Exit Criteria
  - **Guidelines**
- Statechart Examples
- Conclusions

# Guide – Inputs

- Component &/or Class Model for current build
- Interaction diagrams for current build where class instances appear
- Use-Cases for current build
- Components Behavior for other classes

## Guide – Basic Actions

- Build Statechart for each (“interesting”) class
- Validate Statechart
- Update Class Model
- Update Object Interactions

## Building a Statechart

- Identify Condition–Response Behavior for all instances of classifier being described
- Create top–level behavior diagram
- For each state in diagram
  - Describe work of object when in state
    - Create subdiagrams for sub-behaviors as needed
    - For top-level diagram,
      - Generally only need to describe detailed behavior of “On” or “Alive” state
  - Repeat for all states in sub-behaviors (“subdiagrams”)

# Building a Statechart

## Identify Condition–Response Behavior

- Review
  - Interaction Diagrams
  - Use–cases used to create Interaction Diagrams
  - Source documents used to create Use–Cases
- Identify actions an instance performs when message arrives from client instance
  - Identify how operations are performed with different actual parameters

# Building a Statechart

## Identify Condition–Response Behavior (cont.)

- Identify conditions that cause instance to send messages to “server” objects
  - Instance receives a message, so issues a message
  - Instance reaches some state
  - Instance detects some condition
    - e.g. A value is out of limits

## Building a Statechart

### Identify Condition–Response Behavior (cont.)

- Identify sequences of conditions & actions
- Identify internal behaviors which object performs
  - On “continuous” or “periodic” basis
    - Determine whether these occur only when object is in subset of its possible states
    - e.g. particular “modes”
      - May not be able to determine at this time
  - At same time as other behaviors

## Building a Statechart

### Create top–level behavior diagram (cont.)

- Add state icons to diagram as appropriate
  - Either
    - 2 states, “On” & “Off”
      - Typical if object is persistent
        - i.e., exists beyond execution of encompassing system
    - 1 state, typically named “Exists” or “Alive”
  - Plus a termination state
- Name states

# Building a Statechart

## Create top-level behavior diagram (cont.)

- Add transitions
  - *Default* transition
    - Indicating start state
  - Transitions between states, if applicable
  - *Internal Transitions* for states
    - i.e., for handling events other than those which cause transition to another state state
  - Transition to termination state if appropriate
- For each transition, describe if appropriate
  - Guard-conditions
  - Actions

# Building a Statechart

## For each state, describe work

- Either
  - In label
    - If can totally describe as “Entry”, “Exit”, and internal transitions
      - i.e. Behavior is simple
        - Doesn't involve concurrency
        - Is unconditional, other than entry, exit, and internal transitions
      - Unlikely for the “On” or “Alive” state of most objects
  - In “sub-behavior”
    - Enlarge state to make room for nested diagram, or create subdiagram
      - If creating subdiagram & tool doesn't support UML “hidden decomposition indicator”,
        - Add stereotype <<composite>>

## Building a Statechart

### For each state, describe work

- If multiple behaviors are performed
  - Describe simultaneous behavior
    - See Describing Simultaneous Behaviors below
- Otherwise
  - Describe “sub” state–diagram
    - See Describing A Single Sub-Behavior below
  
- Note: use results of step as guide

## Building a Statechart

### Describing Simultaneous Behaviors

- For each simultaneous behavior discovered
  - Add separator to state
  - Determine which condition & action sequences are handled in this behavior
  - If multiple behaviors are performed simultaneously within this behavior
    - Repeat this step till no additional simultaneous behaviors are discovered
  - Otherwise, describe each behavior

# Building a Statechart

## Describing a Single Sub-Behavior

- Determine which condition/action sequence(s) are defined in this behavior
  - Note: use results of first step as guide
- For each state identified
  - Add & name a state icon to diagram
- If start state not yet identified, do so
  - Add & name a state icon
    - Often a “Wait” state
  - Indicated by drawing *default* transition

# Building a Statechart

## Describing a Single Sub-Behavior (cont.)

- For each state, add transitions to other states as appropriate
  - For each message received in condition/action sequence
    - Ignore message request if it is “queued” until instance is in another state
    - Otherwise, determine state that instance should be in after receiving message
      - (see next slide)

# Building a Statechart

## Describing a Single Sub-Behavior (cont.)

- For each state, add transitions to other states as appropriate (cont.)
  - If *next* state is state which object was in when receiving request,
    - Treat as Internal Transition
  - If *next* state doesn't depend on guard-condition
    - Add & name state, if it's not one of states already discovered
    - Add transition from current state to next state
    - Describe guard-condition for transition
    - Describe actions taken in addition to state change
  - If *next* state depends on a condition
    - Determine next state for each condition that results in a different action or state
    - Add & name state, if not one of states already discovered
    - Add transition from current state to next state
    - Describe event & guard-condition for transition
    - Describe actions taken in addition to state change

# Building a Statechart

## Describing a Single Sub-Behavior (cont.)

- For each message identified as sent by instance for this Condition/Action sequence
  - If condition for issuing message is that this instance received a message
    - Request should be described in action of a transition identified in "For each message received..." step
  - Otherwise
    - If request can only be issued while instance is in specific state
      - Add transition from that state to next state
        - Add & name state, if it's not already discovered
        - Describe event & guard-condition of transition
        - Describe actions taken in addition to state change
      - Verify that request can't be issued from any other known state

# Building a Statechart

## Describing a Single Sub-Behavior (cont.)

- For each message identified as sent by instance for this Condition/Action sequence (cont.)
  - If messages can be sent while instance is in any state (assuming conditions are met)
    - Add transition from each state to an appropriate next state
      - Add & name state, if it's not already discovered
      - Describe event & guard-condition of transition
      - Describe actions taken in addition to state change

# Building a Statechart

## Describing a Single Sub-Behavior (cont.)

- For each message from another simultaneous behavior in same instance that this behavior can respond to:
  - Follow same process as described in “For each message received...” step
- Describe behavior of instance when in each state (as appropriate)

# Validating Statechart

- Simulate execution (manually or automated) of Statechart
  - Verify
    - Correct transition occurs from every state, for every possible requested operation & all conditions
    - Correct actions performed for all sequences of operation requests by “client” objects
    - Results are what “client” objects expect
  - Generate sequence diagrams for class (optional)
    - Focused on class
    - Showing black–box behavior from client object perspective
      - e.g.
        - Op–C can follow Op–B, which can follow Op–A
        - but Op–C following Op–A results in exception

# Outline

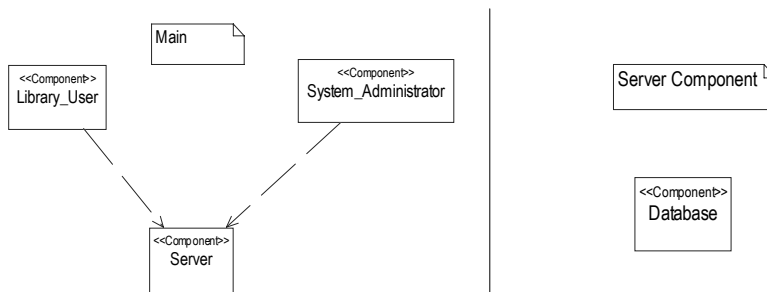
- Review of Process Guidelines
- Guidelines for Creating Statecharts
  - Defining Behavior
  - Exit Criteria
  - Guidelines
- **Statechart Examples**
- Conclusions

# Statechart Examples

- Library\_User Component (Informal)
- User\_Agent (Formal)

# Architecture – LCO

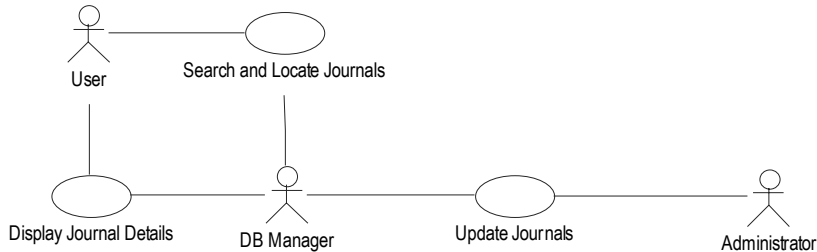
## Component Model For Full-text Title Database System



- Database component from System Block Diagram
- Decision
  - Client-Server Model
- Note: Database is often deferred until later in development
  - Some classes just described as “persistent” early in process

# Use-Case Diagram Example

## 1<sup>st</sup> Build LCO



# LCO Use-Case Description

## Full-text Journal Title Search

<b>Use-Case Name</b>	Full-text Journal Title Search
<b>Abstract</b>	
<b>Purpose</b>	To allow a user to search for journals to which USC Libraries subscribes by keyword
<b>Actors</b>	User, DB Manager
<b>Importance</b>	Primary
<b>Requirements</b>	Full-text Journal Title Search
<b>Risks</b>	
<b>High-Risk?</b>	No
<b>Architecturally Significant?</b>	Yes
<b>Development Status</b>	Draft LCO
<b>Overview</b>	User enters search criteria and system returns lists of journals matching criteria
<b>User Interface</b>	
<b>Pre-conditions</b>	Database has been initialized
<b>Post-conditions</b>	Displayed List of all the complete journal titles containing the user's search criteria
<b>Includes</b>	
<b>Extension Points</b>	

# LCO Use-Case Description

## Full-text Journal Title Search (cont.)

### ■ Typical Course of Action

Seq. #	Actor Actions	System Response
1.	User requests search	
2.		Displays search page
2	User enters search criteria	
3		Queries the database asking for journal titles that match the user's search criteria
4		Displaying the journal list in search result page

# LCO Use-Case Description

## Full-text Journal Title Search (cont.)

### ■ Alternate Course of Action: No results match search criterion

Seq. #	Actor Actions	System Response
4.		Display error page that asks user to search again

### ■ Exceptional Course of Action: None

# LCO Use-Case Description

## Display Journal Details

Unique ID	UC-02
Use-Case Name	Display Journal Details
Abstract	No
Purpose	To allow a user to see detailed information about a journal
Actors	User, DB Manager
Importance	Primary
Requirements	RQ-2
Risks	
High-Risk?	No
Architecturally Significant?	Yes
Development Status	Draft LCO
Overview	System displays full information (title, date of publication, and web address) of a journal that the user selected from list returned by Search for Journal (UC-1)
User Interface	See prototype figure ...
Pre-conditions	Search for Journal (UC-1)
Post-conditions	Details of selected journal displayed
Includes	None
Extension Points	

# LCO Use-Case Description

## Display Journal Details (cont.)

### ■ Alternate Course of Action: Quick Select

Seq. #	Actor Actions	System Response
1.	User "quick" selects journal from displayed list (e.g. double-clicks on name)	
2.		Displays journal details
3	END	

# LCO Use-Case Description

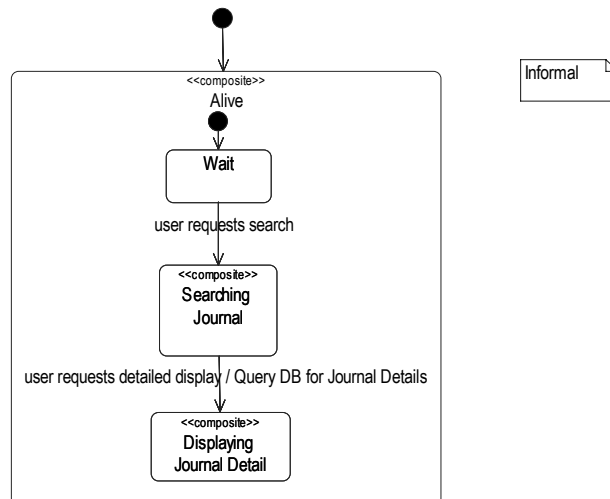
## Display Journal Details (cont.)

### ■ Typical Course of Action

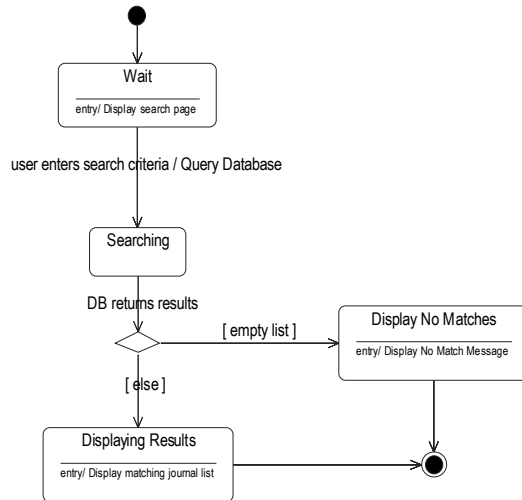
Seq. #	Actor Actions	System Response
1.	User selects journal from displayed list	
2.		Highlights selection
3.	User selects display	
4.		Displays journal details

# Library\_User Component

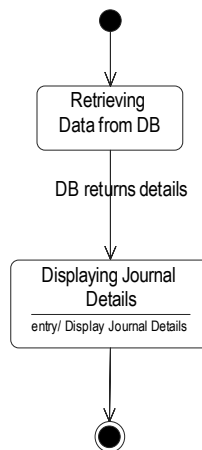
## Statechart – Top Level



# Library\_User Component Statechart – Searching Journals State



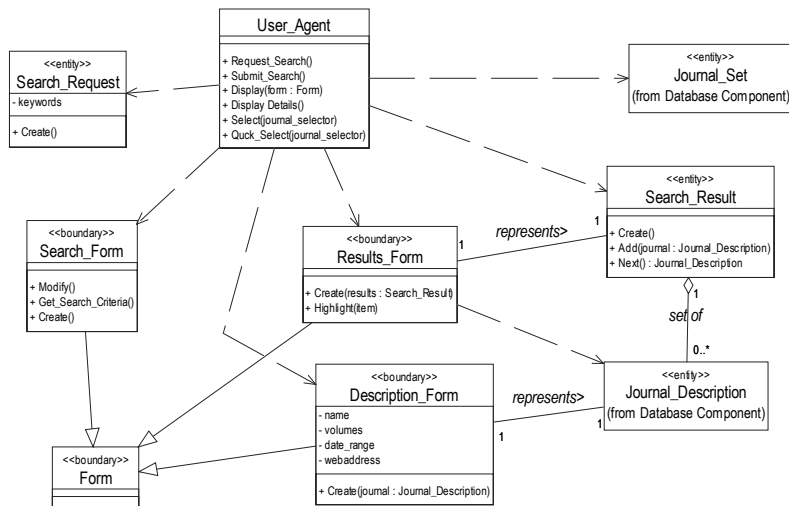
# Library\_User Component Statechart – Displaying Journal Details State



# Statechart Examples

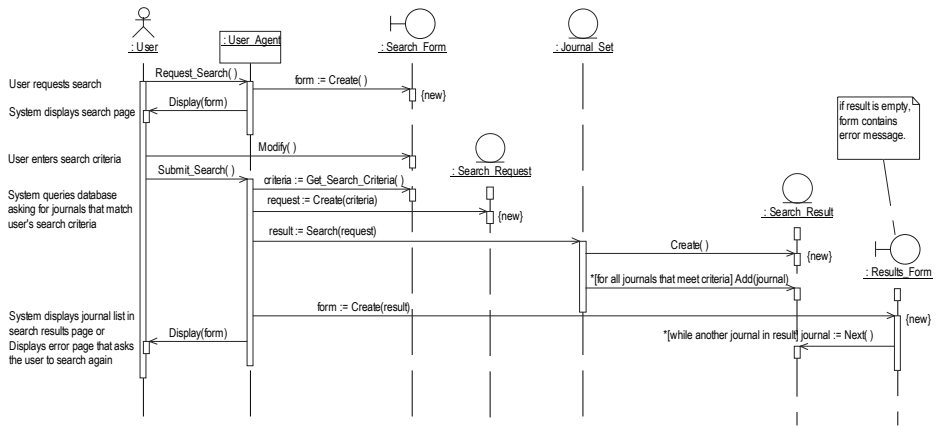
- Library User Component (Informal)
- User\_Agent (Formal)

# LCO Logical Class Model For Library\_User Component



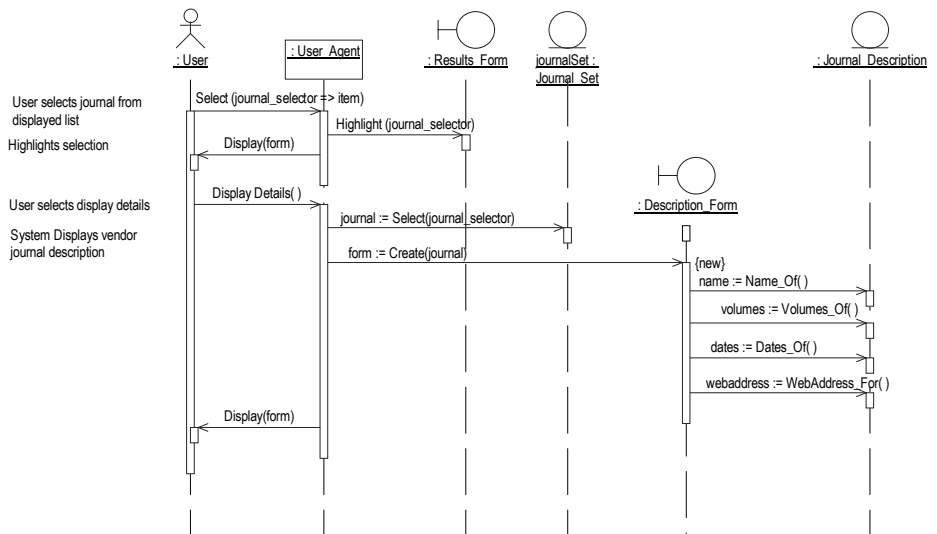
# LCO Sequence Diagram

## Search for Journal Use-Case (All Courses of Action)



# LCO Sequence Diagram

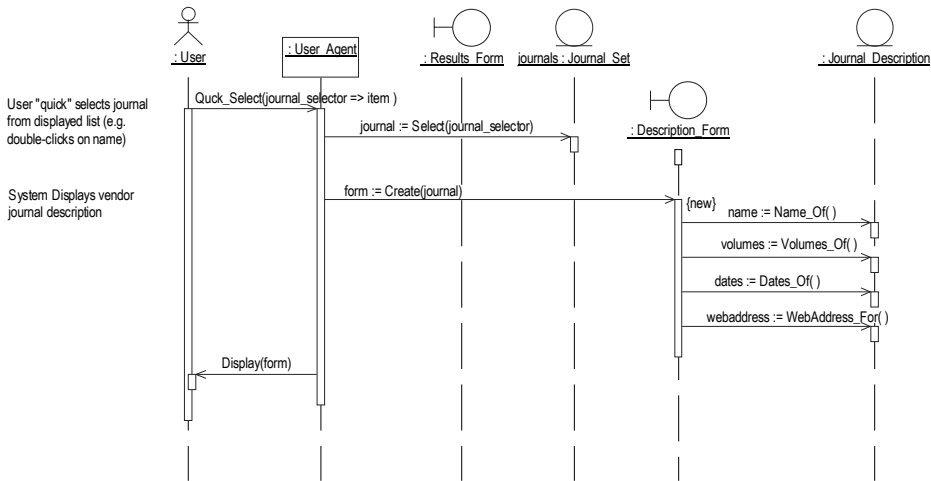
## Display Journal Details Use-Case (Typical COA)



# LCO Sequence Diagram

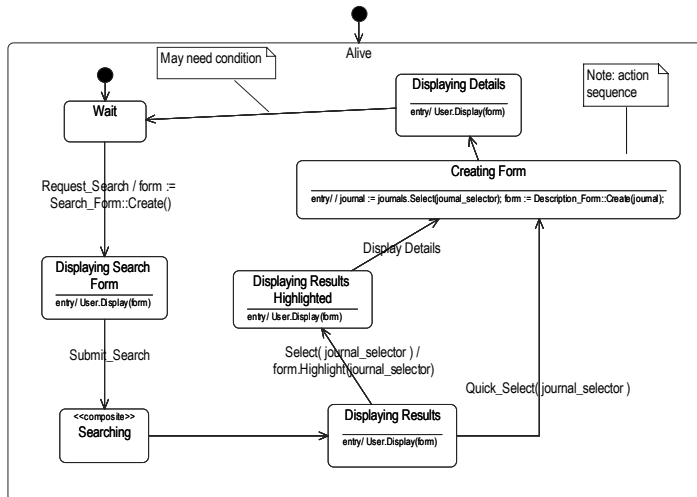
## Display Journal Details Use-Case

### ("Quick Select" Alt COA)

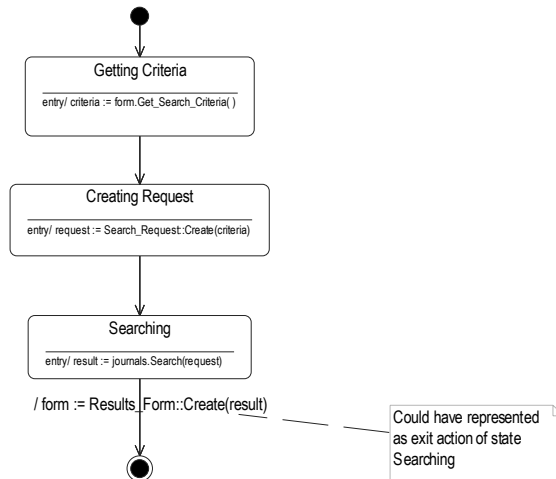


# Library\_User::User\_Agent

## Statechart – Top Level



# Library\_User::User\_Agent Statechart – Alive::Searching State



## Outline

- Review of Process Guidelines
- Guidelines for Creating Statecharts
  - Defining Behavior
  - Exit Criteria
  - Guidelines
- Statechart Examples
- **Conclusions**

# Conclusions

- Valuable technique for validating
  - Model of systems & its parts
    - Right operations?
    - Right attributes?
    - Right communication sequences?
  - Behavior of system & its parts
    - Statecharts are executable
- Valuable for generating code
- Expensive
  - Takes time to produce
  - Takes time to validate
    - Tools that automate execution help