

Process and Data Modeling

Lecture 4, Fall 2005
UML – behavior diagrams

Agenda

- Use cases
- Activity Diagrams
- Sequence Diagrams
- State Machine Diagrams

Use Cases

- Purpose: writing down behavioral requirements
- Users interaction with the system to accomplish some goal
- A prose essay – or narrative...
- Textual, not graphical!
- What does a *good* use case look like?

Use cases can be used to:

- Describe a business' work process
- Focus discussions about requirements
- Discover requirements
- Describe functional requirements
- Document the design of a system
- Use cases can have different writing styles, depending on their purpose
 - Casual or full template
 - Business or system focus
 - Levels (summary, user goal, subfunction)
 - White box or black box
- Don't get caught up with precision and rigor when it is not needed!

Different levels

- Kite level: Summary goals
 - Ex. advertise, order, invoice
- Sea level: User goals
 - Ex. set up promotion, place order, create invoice
- Fish level: Subfunctions
 - Ex. identify promotion, register user, identify customer

Example use case (A. Cockburn) "Get paid for car accident" (1)

Primary actor: The claimant

Scope: The insurance company

Level: Summary

Stakeholders and Interests:

The claimant – to get paid the most possible

The insurance company – to pay the smallest appropriate amount

Insurance department – to see that all guidelines are followed

Precondition: none

Minimal guarantees: Insurance company logs the claim

Success guaranties: claimant and insurance company agree on amount to get paid, claimant gets paid that

Trigger: claimant submits a claim

Example use case (A. Cockburn) "Get paid for car accident" (2)

Main success scenario

1. Claimant submits claim with substantiating data
2. Insurance company verifies claimant owns a valid policy
3. Insurance company assigns agent to examine case
4. Insurance company verifies all details are withing policy guidelines
5. Insurance company pays claimant and closes file

Extensions

- 1a. Submitted data is incomplete
 - 1a1. Insurance company requests missing information
 - 1a2. Claimant supplies missing information
- 2a. Claimant does not own a policy
 - 2a. Insurance company declines claim, notifies claimant, terminates proceedings

Writing use cases

1. Name the system scope and boundaries
2. Brainstorm and list the primary actors
3. Brainstorm and exhaustively list user goals for the system
4. Capture the outermost summary use cases
5. Revise summary use cases
6. Select one use case to expand
7. Capture stakeholders, preconditions and guarantees
8. Write main succes scenario (MSS)
9. Brainstorm and list extension conditions
10. Write extension-handling steps
11. Extract complex flows to sub use cases
12. Readjust: add, subtract, merge as needed

Activity Diagrams

- Describe procedural logic, business process, workflow
- Similar to flowcharts, but support parallel behavior
- Can they be understood by end users?
- Might play a role in describing the behavioral aspects of UML as a programming language

Activity Diagrams

- Nodes are called actions
- An activity is a sequence of actions
- The lines connecting the nodes are called flows or edges
 - Arrows
 - Connectors
- For showing parameters
 - Object boxes
 - Pins

Activity Diagrams Conditional behavior

- Decision
 - Also known as a *branch*
 - One incoming flow
 - Several guarded outbound flows
 - A guard is a Boolean expression in square brackets
 - Guards are mutually exclusive (only one path can be chosen)
 - [else] means that the flow should be used when all other flows evaluate to false
- Merge
 - Multiple input flows
 - One outbound flow
 - Flow continues from a merge if *any* input flow triggers
- Decisions and merges are depicted with a diamond shape

Activity Diagrams Parallel execution

- Fork
 - One incoming flow
 - Multiple outgoing flows
 - No sequence indicated
- Join
 - Synchronization of flows
 - Multiple incoming flows, one outgoing
 - Flow continues from a join when all incoming flows have reached the join
 - There can be join specifications attached to a join – Boolean expression that should evaluate to true before the flow can continue
- Forks and joins are depicted with a horizontal bar

Activity Diagrams Decomposing an action

- Actions can be decomposed into subactivities
- A rake symbol indicates a subactivity diagram
- If parameters are passed to the subactivity, they can be indicated by object boxes

Activity Diagrams

- Partitions give an overview of who is performing the activity
 - They are also known as *swimlanes*
- Initial node and activity final node
- Expansion regions
 - Where actions occur once for each item in a collection
 - Can be either concurrent or iterative, indicated by a keyword
 - A flow final is when one token in the expansion is destroyed and its flow stops
 - The expansion region then has a filtering function

Activity Diagrams Signals

- When an action listens for events from an outside process
- Time signals occur when time passes – they are depicted with a hour-glass like symbol
- There can be a time-out on time signals
- An activity can send signals and wait for a reply before it continues

State Machine Diagrams

- Describing the life time behavior of a single object
- States and transitions
- A transition
 - Trigger-signature [guard] /activity
 - Event [Boolean condition]/ behavior that is executed
 - All parts are optional
 - With multiple transitions from the same event, the guards must be mutually exclusive
 - If an event occurs and no transition is valid, it is ignored

State Machine Diagrams

- Different states imply a different way of reacting to events
- Not good at showing collaborations between objects
- A transition shows a movement from one state to another

State Machine Diagrams

- Internal activities and self-transition
 - The activity does something when an event happens, and go back into the same state
 - There is not transition
 - Entry and exit activities are only executed when you enter or leave a state
- Activity states
 - The object has an on-going activity
 - A do-activity
 - It can be interrupted
- Super states
 - When shared behavior is moved into a common super state

State Machine Diagrams

- How to implement that an objects behavior is depending on its state?
- Example: The State Pattern

Sequence Diagrams

- Purpose: to show collaborations of objects through the sending of messages
- Have similarities with stack frames
- Each participant has a lifeline
- When a participant is active, the lifeline becomes an activation bar
- Showing return calls is optional, but can sometimes be useful
- The first message sent is called a found message – the source of it is undetermined

Sequence diagrams

- Not good at showing loops and conditional behavior
- Pseudocode or other textual representations are better at showing this
- Give a clear picture of the interaction
- Ex. whether the design style emphasizes centralized or distributed control
- A way to sanity check design criteria
- Sequence diagrams are very useful for understanding and communicating about the system design

Sequence Diagrams

- There is notations for creating and deleting participants
- Creation is marked with a 'new' on the message arrow
- Explicit deletion is not necessary in a garbage collected environment – however, it might be useful to show on the diagram when the object is ready to be collected

Sequence Diagrams

- There is a notation to show conditionals and loops called *interaction frames*
- They divide the diagram into fragments, that have an operator (ex. loop, alt) and a guard
- Pseudomessages can be used instead of frames
- Synchronous and asynchronous calls are depicted by different styles of arrowheads