First Deliverable

Worth: 5%
Week 5

- Prepare a 35-40 minute presentation. The presentation should begin by going over the use case diagram, and then consist of stepping through some individual use cases. (Suggestion: start with the use cases that you are the most concerned about because not all use cases will be covered.)

- The TAs and instructors will interrupt with questions.
First Deliverable

Marked out of 10 as follows:

- completeness (/2) – do you have all use cases?
- quality (/4)
- presentation (/2)
- questions (/2) – all group members should be able to answer questions well
Stopwatch So Far

```
but3[!timerRunning] /resetTimer(timer)
but3[timerRunning] /lapTime:=timer; light
but2[!timerRunning] / startTimer( timer ); beep
but2[timerRunning] / stopTimer(timer); beep
```
Today’s Agenda

- UML State Diagrams
  - Concurrency (from last time)
  - State diagram semantics exercise (from last time)
  - Actions vs activities
  - State actions
  - Junctions points
  - Special events:
    - Change events
    - Time events

- System state diagrams

We will continue with the stopwatch/clock example.
Actions vs Activities

Recall: An action is uninterruptible and is considered to be completed instantaneously.

An activity is computation of the system that takes time.

An activity may be associated with a state. (Unlike what we said earlier, the system can now change while sitting in a state.)

States with activities are called activity states.
Activities

Unlike actions, because activities take time, they can be interrupted.

A

- do / computeBill

B

- t1: x

C

- t2:[on]

D

- t3

T1, T2, T3 are used to designate transitions and are not events, conditions, or actions.
Actions vs Activities

$\text{C}$

\[ \text{t2: } x \]

$\text{A}$

\[ \text{do / computeBill} \]

\[ \text{t1} \]

$\text{B}$

$t_1, t_2, t_3$ are used to designate transitions and are not events, conditions, or actions.
Synchrony Hypothesis

- Due to Gerard Berry, creator of the specification notation Esterel.

- The *synchrony hypothesis* is the assumption that the system can respond to an input faster than another input can be provided.

- This is usually a valid assumption during requirements modelling and simplifies state diagrams because actions can be used rather than activities.

- In thinking about whether to make something an action or an activity, consider whether it is *interruptible* by another input, rather than whether it takes time.

  - **Example:** “Creating a user account” can probably be cancelled in the middle, therefore it is a good candidate to make an activity rather than an action.
Actions vs Activities

- In implementation, every task will take time – just because we consider actions to occur instantaneously in a state diagram does not mean that their implementation will be instantaneous.

- Both actions and activities will be refined to actions (i.e., uninterruptible tasks) at the concept-level.

- In RE, the important part is to consider whether the task is interruptable by an input!

- UML 2.0 has dropped actions. This is too bad because the synchrony hypothesis is a very useful, valid, simplifying assumption for requirements engineering. Therefore, we will use actions in our state diagrams.

UML usually only uses state diagrams to describe the behaviour of concepts.
State Actions

As abbreviations, we can list some actions within a basic state:

- Entry actions – actions that occur every time the state is entered by an explicit transition.
- Exit actions – action that occur every time the state is exited by an explicit transition.
- Implicit transitions on events

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry /action</td>
</tr>
<tr>
<td>exit /action</td>
</tr>
<tr>
<td>event [condition] / action</td>
</tr>
</tbody>
</table>
Entry Actions

A \rightarrow \text{ev1} \rightarrow \text{E}

B \rightarrow \text{ev2} \rightarrow \text{E}

\text{entry} / x

\text{ev1} \rightarrow \text{E}

\text{ev2} \rightarrow \text{E}

\text{ev3} \rightarrow \text{E}

\text{ev4} \rightarrow \text{E}

\text{ev5} \rightarrow \text{E}

A \rightarrow \text{ev1} / x \rightarrow \text{E}

B \rightarrow \text{ev2} / x \rightarrow \text{E}

\text{ev3} / x \rightarrow \text{E}

\text{ev4} \rightarrow \text{E}

\text{ev5} \rightarrow \text{E}
Exit Actions

<table>
<thead>
<tr>
<th>A</th>
<th>ev1</th>
<th>B</th>
<th>ev2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exit /x

<table>
<thead>
<tr>
<th>C</th>
<th>ev3</th>
<th>D</th>
<th>ev4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
<th>ev1</th>
<th>B</th>
<th>ev2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exit /x

<table>
<thead>
<tr>
<th>C</th>
<th>ev3</th>
<th>D</th>
<th>ev4</th>
<th>ev5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ev3 / x
Implicit Transitions

Note: The above equivalence is valid if there are not entry or exit actions. Entry and exit actions are NOT performed on implicit transitions.
State Actions and Activities

States can be annotated with entry or exit actions, implicit transitions (also called internal actions), and activities:

- do/activity
- entry/action
- exit/action
- event/action

If a transition with no event label exits a state, it can occur once any activity associated with the state is complete.
State Actions and Activities

In an explicit transition (including self-looping transitions), the order of effects:

1. exit actions of source state,
2. transition actions (in order),
3. entry actions of destination state, and
4. state activities

If you want a self-looping transition that does not activate exit and entry events, you can use an internal transition.
When are the following equivalent?

1. A ev/alert B
2. A ev B entry/alert B
3. A ev B do/alert B
4. A ev C do/alert B
Junction Points

![Diagram of Junction Points]

- A
  - ev1
  - ev2
  - C

- B
  - ev2
  - C

- A
  - ev1
  - C

- B
  - ev2
  - C

- [cond]
  - C

- [cond]
  - D
Change Events

A change event is the event of a condition becoming true. The event occurs when the condition goes from false to true because the values of some variables used in the condition change their values.

Examples:

- when(temperature > 100 degrees)
- when(on)

The event does not reoccur unless the condition turns to false and then returns to true.
Time Event

An time event is the occurrence of a specific date/time or the passage of time.

Examples:

- **Absolute time:**
  - when(date=25/12/2005)

- **Relative time:**
  - after(10 seconds since exit from state A)
  - after(10 seconds since x)
  - after (20 minutes)
  - Assumed to be since entered the state that is the source of the transition
Event Summary

An event is instantaneous.

Kinds of events:

- (external) a change in the environment (e.g., “off-hook”)
- (external/internal) change events (occurrence of a condition becoming true)
- time events (occurrence of relative or absolute passage of time)
- (internal) a message from another object (operation call)
Each event, action, and activity may have parameters.
Creating a System State Diagrams

Inputs to this process:

- Use cases
- System sequence diagrams
- Input events
- Output events

Output:

- System state diagram (possibly with concurrent components)

This can be done for major subsystems rather than the whole system.
Creating a System State Diagram

Every scenario of the use cases (and system sequence diagrams) must be possible behaviours of the system state diagram.

Process:

1. Think of a natural partitioning into states
2. Consider the behaviour of the system for each input at each state.
3. Iterate, introducing new states

Alternatively: build a state diagram for each use case and merge these state diagrams.
Data Abstraction

- What data does the system manipulate?
- Create an abstract set of operations on this data to represent steps found in use cases.
- Pass data as parameters to operations, e.g., createUser(name,address)
- You can assume an implicit store. (This data will become objects!)
- Are these operations interruptible? If so, use activities, otherwise use actions.
- Make sure state diagram captures preconditions of operations (e.g., Should an account exist before a phone call can be made? Can multiple accounts be created at the same time?)
Issues

- Choose a natural set of basic states
  - Think of the modes of the system; i.e., when the system waits for input from the environment (or the passage of time) before it can change

- Use concurrency to clarify the specification
  - Recognize what aspects of the system are orthogonal

- Use hierarchy to capture common behaviour and clarify the specification.

- Avoid flower diagrams!
State Diagrams: Validation

- **Avoid inconsistency**: don’t have multiple transitions leave the same state under the same event.

- **Ensure completeness**: a reaction is specified for every possible input at a state.

  This is an issue particularly when transitions are conditional. If there are transitions triggered by an event conditioned on some guard, what happens if the guard is false?

- **Walkthrough**: compare the behaviour of your state diagrams with the sequence diagrams. All paths through the sequence diagrams should be paths in the collection of state machines.
Summary

- UML State Diagrams
  - Concurrency
  - State diagram semantics exercise
  - Actions vs activities
  - State actions
  - Junction points
  - Special events:
    - Change events
    - Time events

- System state diagrams

Next Lecture: Reference Model for RE

Reading: Jackson and Zave (course pack)