Lecture 12: State Chart Models
State Models

• Components
  – States: a condition of an object
  – Events: a state changing occurrence
  – Transitions: from one state to the next, occur when the associated event occurs
Sample State Diagram

- Breakfast
- InClass
- ClassTime
- ClassOver
- NextClassStartTime
- Dinner
- DinnerFinished
- DinnerTime
- Studying
- Sleeping
- Alarm
Consistent and Complete

- Multiple transitions from a state?
  - Non-overlapping events (consistency)
  - Complete set of events

- Sample state machine
  - What if dinner time occurs at the same time as the next class time?
  - What happens after breakfast if there are no classes?
  - What happens if dinner time event occurs while in class?
Augmented Diagrams

- Constraints/guards needed for better model
  - Condition(s) added to transition(s)
  - If event occurs, transition takes place only if condition is satisfied
- Actions can be added to transitions
  - Performed if transition takes place
- Actions on state entry and state exit also
Student Example 2

Breakfast
  - Alarm
  - Sleeping
  - DinnerFinished

BreakfastOver
  - DinnerTime [NoMoreClasses]
  - ClassTime

Studying
  - ClassOver [<DinnerTime or More Classes]
  - DinnerFinished

Dinner
  - ClassOver [>=DinnerTime & NoMoreClasses]
  - InClass

Copyright W. Howden
State Model Representations

• Graphical: useful for small systems
• Tables
  – State/state or event/state
  – Better for larger models
  – Duplicate events and states are obvious
  – Easier to build up incrementally
• State variable modification equations
# State/State Model Table

<table>
<thead>
<tr>
<th>State ⇒</th>
<th>Sleeping</th>
<th>Dinner</th>
<th>In Class</th>
<th>Breakfast</th>
<th>Studying</th>
</tr>
</thead>
<tbody>
<tr>
<td>State ⇓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sleeping</td>
<td></td>
<td></td>
<td></td>
<td>Alarm</td>
<td></td>
</tr>
<tr>
<td>Dinner</td>
<td>DinnerOver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In Class</td>
<td></td>
<td>ClassOver</td>
<td></td>
<td>ClassOver [&lt;DinnerTime or MoreClasses]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DinnerTime</td>
<td></td>
<td>[&gt;=DinnerTime &amp; NoMoreClasses]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breakfast</td>
<td></td>
<td></td>
<td></td>
<td>Breakfast Over</td>
<td></td>
</tr>
<tr>
<td>Studying</td>
<td></td>
<td>DinnerTime</td>
<td></td>
<td>ClassTime</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[NoMoreClasses]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Event/State Model Table

<table>
<thead>
<tr>
<th>State ⇒ Event ↓</th>
<th>Sleeping</th>
<th>Dinner</th>
<th>In Class</th>
<th>Breakfast</th>
<th>Studying</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm</td>
<td>Breakfast</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BreakfastOver</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Studying</td>
</tr>
<tr>
<td>ClassTime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>InClass</td>
</tr>
<tr>
<td>Dinner Over</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClassOver</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dinner Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[NoMoreClasses] Dinner</td>
</tr>
</tbody>
</table>

- Breakfast Over
- Class Over
- Dinner Time
State Equation Models

• Suppose that $x_1 \ldots x_n$ is a set of state variables. Each combination of values is a possible state.

• State variable update equation
  – shows how a variable is updated for each event

• Model
  – list of update equations
  – state model change
    • when an event occurs go through list in order, executing each equation
    • assume only one event can occur at a time
Variable Value Notation

• State equation variable values
  – $x_i$ is old value of $x_i$
  – $x_i'$ is new value of $x_i$

• Model some “events” using state variables
  – e.g. dinner time occurring event
    • $dt == \text{false}$ and $dt' == \text{true}$
DS State Model Examples

- ext. events: clocktick, breakfastOver, dinnerOver
- variables:
  - cl (clock time), dt (dinner time event) di (dinner state)

Sample equations for some state variables

if (clocktick) then cl' = cl + 1 (mod 1440)
if cl' = 18:00 then dt' = true
...
if (cl’ > 16:30 /*no more classes*/ and dt’)
    {di’ = true}
...
dt = dt’ = false; cl = cl’; di = di’
State Model Applications

• System specification and design
• Class specification
  – shows the states an object can be in and how the transitions occur as the result of events. Events correspond to messages arriving (or values returned from messages sent).
• GUI interface design
Client Server Model

Login

OKButton

UnauthorizedUserMessage

ServerReply [NotAMember] / DisplayErrorMessage

OKButton / Transmit "GetData(name)"

Wait for Reply

ServerReply [IsAMember] / DisplayUserOption

DateOptions

Wait for DB Request

ReceiveRequest / AccessDB

DBReplies / Transmit "ServerReply"

WaitForDBAnswer
Object State Model - Stack

- **Empty**
  - `top() / return "error"`
  - `push(x) / putElementOnStack`
  - `pop() / removeTopElement`

- **Partial**
  - `push(x) / putElementOnStack`
  - `pop() / removeTopElement`
  - `push(x)[numElements < max-1] / putxOnStack`
  - `pop() / removeTopElement`

- **Full**
  - `push(x) / putxOnStack`
  - `push() / return "error"`
  - `top() / returnTopElement`
State Models and GUI Interface Design

• Screens correspond to states
  – e.g. DS: LogOn, MemberOptions, AdminOptions, DatePrefs, DateeData, etc.

• Current state = visible screen/dialog

• Events correspond to user actions
  – e.g. DS: enter button for LogOn dialog
  – non-modeled events: mouse movements, screen gets focus, textdata entry, etc.
State Charts – Additional Notation in DS GUI Example

• Nested Models

• Showing transfer of control from nested substates to common superstate
  – OK Button events in GUI substates to Start/End in higher level state
  – Entry substate showing which substate is entered when control transits to superstate
Additional Notation – Concurrent Sub-models

• Every submodel has its own set of states, modeled by possible values of its state variable vector

• Compound model: states correspond to all possible combinations of states of submodels
Additional Notation –
History Substates

- State A with substates a, b, c, d
- Entry to state A causes entry to designated substate, say b
- Exit from state A on some event
- Re-entry to state A on some event
  - re-enter at initial substate
  - re-enter at previous substate, when exit from A occurred: history states option
State Model Analysis

• Completeness and consistency of set of transition conditions for a state

• Necessary problem dependent model properties
  – e.g. in the DS systems, no matter which state you are in, there is a path of control back to the Start/End state
  - example of “model checking”
State Models and Testing

• Coverage: every transition should be covered by some test

• Context based testing:
  – you are at some state A with an out transition z
  – there is some previous state B with two transitions x and y that lead to A. Each of these is a context
  – test z for both x and y, if possible
Sequence Diagrams vs State Models

• **Sequence diagrams**
  – used to identify input forms for GUI, etc
  – show a particular scenario

• **State Model**
  – merges scenarios into inclusive model
  – gives complete picture of order in which things can occur
State Model Pitfalls

• Too detailed
• Badly drawn/organized
  – need for state model tool
• Not really a state model, but a flow chart
  – states are actions and transitions are conditions on existing values, like a control flow diagram
• Transitions should be made on the occurrence of events
DS Bad State Chart

ChooseAdminOption

ChooseMemberOption

LogOn

[L=AdminLoggedOn] [L=MemberLoggedOn]

[O=SetMemberDataOption] [O=GetADateOption]

EnterData

LookForDate

[R=NoDateFound] [R=DateFound]

DisplaySorry

DisplayDateeData
Comments on Bad State Chart

DS State Chart
- State: presenting user with opportunity to enter data as input
- Event: button pushed indicating data entered.
- Constraint: based on state variables

DS Flow Chart
- Action: procedure carried out in the system on data
- Action terminates: flow leaves action along some transition
- Condition: based on state variables
Terminology

• Modal versus Modeless classes/objects
  
  *Modal*  Object behavior depends on state
  e.g. modal dialog boxes: if include data entry, cannot terminate/exit before input is done

  *Modeless* Behavior (e.g. of method calls) not dependent on state, no need for state model

• External and events and state models

  *External* From outside model or submodel (e.g. events on state model transitions). State charts are *event-driven*. 