Model Derivation

From Requirements to Models
What?
How?
Examples?
Lecture Outline

• What?
  – Entities, relationships and behavior
  – xUML model

• How?
  – Linguistic Analysis
  – Object Behavior Analysis
  – Use cases, activity diagrams, etc.

• Examples
  – Simple Microwave Oven
  – Miscellaneous illustrations
Elements of xUML Models

• Systems = Domains and Bridges
• Domains = Class Diagrams
• Bridges = Interactions Among Domains
• Class Diagrams = Classes + Relationships
• Class = Attributes + State Machine + Constraints
• State Machine = Events (Signals) + Actions
What? - Class Diagram

- Identify entities in specification. Entities are classes.
- Identify attributes of entities -> attributes of classes.
- Identify relationships among classes:
  - “has a”
  - “is a”
  - Associative.
- Identify constraints on the possible values of attributes.
What? - Determine Behaviors of Entities

- Construct the lifecycle (state machine) which controls the behavior of each entity.
- Construct the state transition table for each entity.
- Design the action to be executed in each state of the state machine.
What? – System Behavior

• Identify and record the communication among the entities
• Identify a set of use cases which define the environment in which the system will execute
• Define a class or classes which implement these use cases.
How? – Derivation Process

1. Derive Class Diagram
   1. Define classes and select attributes
   2. Define associations/relationships among classes
   3. Construct associative classes
   4. Iterate steps 1, 2 & 3 until consistent

2. Construct State Models
   1. Construct state model for each object with multiple states
   2. For each state model, define actions effecting state changes

3. Iterate until consistent
How? – Class Diagram

Methods for identifying classes, attributes and relationships

• Linguistic analysis of requirements specification - textbook
• Object Behavior Analysis – Rubin and Goldberg paper (related to Use Case analysis)
• Use Case Analysis

Practical Approach - Apply multiple methods and compare resulting class diagram
How? - Class Diagrams via Linguistic Analysis

**Identify the classes.** The classes derive from noun phrases. Go through the requirements statement and pick out all of the important nouns. Categorize these nouns into tangible items or devices, physical classes, roles, interactions, instance specifications, etc.. Decide which ones are significant and which ones are redundant and select classes.

**Identify attributes for each object.** Attributes come from possessive phrases, as descriptions of the nouns. Recall that the attributes define the identity and the state of an object.

**Identify relationships.** Relationships can be derived by looking at the verb phrases of the requirements analysis. Verb phrases include such things as the PC board "is made up of" chips and connectors, etc..

**Specialize and generalize classes** into subtypes and supertypes and create associative, create object definitions for m to n associations or dynamic associations which fall under the heading of events, interactions which have to be remembered.
Classes

• A class is an abstraction from a set of conceptual entities in a domain.
• An object is an instance of a class.
• They all share the same characteristics.
• They all behave according to the same rules and policies.
• These rules and policies may include constraints which supplement the defined behaviors.
Identifying classes

- Tangible Things
- Roles played by people or organizations
- Incidents
- Specifications
- Etc.
Identifying Classes

Classes Are Akin To Pornography Or Beauty

tangible entity or device - airplane
role - professor, student
incident or persistent event - registration of automobile
interaction - contract terms
specification – sets of entities with related characteristics
organization - university
external systems - sensor drives
Identifying Classes

Which Nouns are Classes?

Not all nouns are classes

- Does it have interesting state? Qualified by attributes (adjectives) defining state.
- Does it do anything?
- Does it store data or information accessed by other classes?
Specifications for the Control Software for a One-Button Microwave

The microwave is powered by a powertube with a wattage rating and a part number, it incorporates: an interior light with a specified wattage and part number, a beeper which has a specified part number and a control button. The door of the oven is latched by a friction catch. In the current product the powertube consumes 600 watts and light consumes 40 watts.

This simple oven has a single control button. When the oven door is closed and the user presses the button, the oven will cook (that is, energize the power tube) for 1 minute.

There is a light inside the oven. Any time the oven is cooking, the light must be turned on, so you can peer through the window in the oven's door and see if your food is bubbling. Any time the door is open, the light must be on, so you can see your food or so you have enough light to clean the oven.

When the oven times out (cooks until the desired preset time), it turns off both the power tube and the light. It then emits a warning beep to signal that the food is ready.

The user can stop the cooking by opening the door. Once the door is opened, the timer resets to zero.

Closing the oven door turns out the light.
### Linguistic Analysis – Microwave Oven

<table>
<thead>
<tr>
<th>Noun or Noun Phrase</th>
<th>Type (Tangible, role, specification, etc)</th>
<th>Is It a Class</th>
<th>Why or Why Not</th>
</tr>
</thead>
<tbody>
<tr>
<td>microwave</td>
<td>tangible</td>
<td>No</td>
<td>Same concept as oven</td>
</tr>
<tr>
<td>powertube</td>
<td>Tangible</td>
<td>Yes</td>
<td>Has attributes, states and actions</td>
</tr>
<tr>
<td>light</td>
<td>Tangible</td>
<td>Yes</td>
<td>Has attributes, states and actions</td>
</tr>
<tr>
<td>wattage rating</td>
<td>Specification</td>
<td>No</td>
<td>Is an attribute</td>
</tr>
<tr>
<td>Part number</td>
<td>Specification</td>
<td>No</td>
<td>Is an attribute</td>
</tr>
<tr>
<td>beeper</td>
<td>Tangible</td>
<td>Yes</td>
<td>Has attributes, states and actions</td>
</tr>
<tr>
<td>Control button</td>
<td>Tangible and Role</td>
<td>yes</td>
<td>Has state and sends signals to other entities.</td>
</tr>
<tr>
<td>door</td>
<td>Tangible</td>
<td>yes</td>
<td>Has state – sends signals to other entities</td>
</tr>
<tr>
<td>Oven</td>
<td>Tangible</td>
<td>Yes</td>
<td>Same as microwave so we choose one of the two.</td>
</tr>
<tr>
<td>Product</td>
<td>Role</td>
<td>No</td>
<td>Same as oven or microwave</td>
</tr>
<tr>
<td>watts</td>
<td>Specification</td>
<td>No</td>
<td>Attribute</td>
</tr>
</tbody>
</table>
Attributes

- Attribute is an abstraction of a single characteristic of the entity abstracted into a class.
- Attributes have types: roles and data types.
- Types of Attributes
  - Descriptive – describes intrinsic characteristics of a thing.
  - Naming – name used to refer to a thing.
  - Referential – provides information about links and composition – arise in “has a” relationships.
  - State or status – names of states of state machine
## Attributes

<table>
<thead>
<tr>
<th>Adjective or Qualifying Phrase</th>
<th>Noun Qualified</th>
<th>Is it an attribute?</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>wattage</td>
<td>powertube</td>
<td>Yes</td>
<td>Property</td>
</tr>
<tr>
<td>wattage</td>
<td>light</td>
<td>Yes</td>
<td>property</td>
</tr>
<tr>
<td>Serial number</td>
<td>oven</td>
<td>yes</td>
<td>property</td>
</tr>
</tbody>
</table>


Entities/Classes with States [attributes]

- Door (open, closed)
- Power tube (on, off) [wattage, etc]
- Light (on, off) [wattage, etc]
- Timer (on and counting down, off and zero)
- Beeper (on, off) – Ignore beeper for now
- Button – Event generator
- Oven Controller - Program
Associations - Relationships Among Classes
Hierarchical Structure among Classes – Generalization and Specialization

Definition: An association is the abstraction of a set of domain relationships that hold systematically between different kinds of conceptual entities, abstracted as classes, in the domain.

Associations have:

Names
Roles or reciprocal functions (verb phrases)

Multiplicities

1…1 – Mandatory, exactly 1 to 1
1…*  - Mandatory, perhaps 1 to n
0…1 – Conditional but not more than 1
0…* - Conditional perhaps more than 1

Associative Classes – Define M to N relationships

(Figures 6.1, 6.2, and 6.3. pp84-85)
Associations Represented by Classes

Definition: An association class is an abstraction, as a class, of an association that may have its own attributes, other associations and behavior.

1. Association classes arise from relationships among classes which have semantic content that cannot be expressed by simple verb phrases and/or multiplicity.
2. Association classes can participate in associations other than those which cause their definition.
3. An instance of an association class exists only if the instances of the classes it couples exists.
4. Association classes very often model contention for resources.
Identification of Relationships

- Relationships can be derived by looking at the verb phrases of the requirements analysis. Verb phrases include such things as Universities have departments, departments have professors, professors advise students.
## Relationships

<table>
<thead>
<tr>
<th>Verb Phrase</th>
<th>Relationship or Action</th>
<th>Classes Related</th>
<th>Type of Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microwave is powered by a powertube</td>
<td>relationship</td>
<td>Oven/powertube</td>
<td>Referential attribute</td>
</tr>
<tr>
<td>It incorporates a ..light ..</td>
<td>relationship</td>
<td>Oven/light</td>
<td>Referential attribute</td>
</tr>
<tr>
<td>It incorporates a … control button</td>
<td>relationship</td>
<td>Oven/control button</td>
<td>Referential attribute</td>
</tr>
<tr>
<td>It incorporates a .. ….beeper …</td>
<td>relationship</td>
<td>Oven/beeper</td>
<td>Referential attribute</td>
</tr>
<tr>
<td>The door of the oven is latched</td>
<td>relationship</td>
<td>Oven/door</td>
<td>Referential attribute</td>
</tr>
</tbody>
</table>
Class Diagram - Cheap Microwave Oven Controller

MicrowaveOven(O)
* Oven-ID
  . Manufacturer
  . status
  . SerialNumber

USER(U)
* User-ID

BUTTON (B)
* Button-ID
  . B-status
  . Oven ID (R1)

Door(D)
* Door-ID
  . D-status
  . Oven ID (R1)

LIGHT (L)
* Light-ID
  . L-status
  . Oven ID (R4)
  . wattage

TIMER (T)
* Timer ID
  . status
  . time of next event
  . Oven ID (R5)
  . wattage

POWERTUBE (P)
* Tube-ID
  . wattage
  . P-status
  . Oven ID (R3)

BEEPER
* Beeper-ID
  . status
  . Oven ID (R2)

is powered by

Door (D)
* Door-ID
  . D-status
  . Oven ID (R1)

R1

R2

R3

powers

R4

R5
Properties

Specifications for what the system should do and not do.

Safety properties – States which the system should never reach.

Liveness properties – States the system should always reach from specified initial states.
Properties

• Pushing the button while the Door is closed
• turns on the Light and the Tube for one
• minute
• Pushing the button while the Tube is on,
• extends the cooking period by one minute.
• If the Door is open then the Light is on
• If the Door is open then the Tube is off
• If the Light is on then the Door is open or
• the Tube is on
Lifecycles

• Each active object (instance of a class) in the domain has a behavior (lifecycle).
• Lifecycle sequences the actions the object performs.
• Lifecycles in xUML are modeled as state machines.
• UML state charts used to graphically represent state machines.
State Machines

• Lifecycle formally expressed as a state machine.
• State machine is comprised of
  1. States – each stage of the object lifecycle.
  2. Events – trigger.
  3. Transitions – specifies the new state given current state and event received.
  4. Procedures – specifies the actions that an object performs when it arrives in a state. Specified using an action language.
Events - Use Cases – Environment Specification

Door is opened
Door is closed
Button is pushed
Timer completes cooking period.
State Machine View of Use Cases

![State Machine Diagram]

*External View of the Microwave Oven*
User-Environment with actions

1. Initialization

\[
\text{microwaveId} = \text{userId}; \\
\text{actionsRemaining} = 5; \\
\text{Generate } U_1: \text{ act (userId)};
\]

2. Acting

\[
\text{CHOICE} = \text{ANY("Decision_Type");} \\
\text{if (CHOICE} = 1) \{ \\
\text{Generate MW1: buttonPressed (microwaveId);} \\
\text{else if (CHOICE} = 2) \{ \\
\text{if (Microwave(microwaveId)\_doorOpen)} \text{Generate MW3: doorClosed (microwaveId);} \\
\text{else } \text{Generate MW2: doorOpened (microwaveId);} \\
\text{else if (Microwave(microwaveId)\_tubeOn)} \text{Generate MW4: timerTimesOut (microwaveId);} \\
\text{actionsRemaining} = \text{actionsRemaining} - 1; \\
\text{if (actionsRemaining} > 0) \\
\text{Generate } U_1: \text{ act (userId);} \\
\]
Figure 9.7 from M&B
State Transition Table – Figure 9.4 from M&B

<table>
<thead>
<tr>
<th>buttonPressed</th>
<th>doorOpened</th>
<th>doorClosed</th>
<th>timerTimesOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready To Cook</td>
<td>Cooking</td>
<td>Door Open</td>
<td></td>
</tr>
<tr>
<td>Cooking</td>
<td>Cooking</td>
<td></td>
<td>Cooking Complete</td>
</tr>
<tr>
<td></td>
<td>Cooking</td>
<td>Door Open</td>
<td></td>
</tr>
<tr>
<td>Cooking Complete</td>
<td>Door Open</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cooking</td>
<td></td>
<td>Door Closed</td>
<td></td>
</tr>
<tr>
<td>Door Open</td>
<td>Door Closed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

State Transition Table Based on Microwave Oven Statechart Diagram
Figure 9.7 from M&B

<table>
<thead>
<tr>
<th>buttonPressed</th>
<th>doorOpened</th>
<th>doorClosed</th>
<th>timerTimesOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready To Cook</td>
<td>Cooking</td>
<td>Door Open</td>
<td></td>
</tr>
<tr>
<td>Cooking</td>
<td>Cooking</td>
<td>Cooking</td>
<td>Cooking</td>
</tr>
<tr>
<td></td>
<td>Cooking Extended</td>
<td>Interrupted</td>
<td>Complete</td>
</tr>
<tr>
<td>Cooking Complete</td>
<td>Cooking</td>
<td>Door Open</td>
<td></td>
</tr>
<tr>
<td>Cooking Interrupted</td>
<td></td>
<td></td>
<td>Ready To Cook</td>
</tr>
<tr>
<td>Door Open</td>
<td></td>
<td></td>
<td>Ready To Cook</td>
</tr>
<tr>
<td>Cooking Extended</td>
<td>Cooking</td>
<td>Cooking</td>
<td>Cooking</td>
</tr>
<tr>
<td></td>
<td>Extended</td>
<td>Interrupted</td>
<td>Complete</td>
</tr>
</tbody>
</table>
**Event Ignored cell entry**

When a certain event can happen but causes no effect, the STT cell is filled in as "Event Ignored." In the case of the oven, pressing the button when the door is open has no effect. Figure 9.8 shows the state transition table with Event Ignored entries added.

When an event is ignored, the object stays in the same state it is in and *does not* re-execute the procedure. In this way, an Event Ignored is different from a transition to the same state.

Although the event is ignored in the sense of not causing a transition, it has been detected and consumed by the state machine. The event is not held until the object arrives in a state where the event can cause a transition.
Figure 9.8 from M&B

<table>
<thead>
<tr>
<th>State</th>
<th>buttonPressed</th>
<th>doorOpened</th>
<th>doorClosed</th>
<th>timerTimesOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready To Cook</td>
<td>Cooking</td>
<td>Door Open</td>
<td></td>
<td>Event Ignored</td>
</tr>
<tr>
<td>Cooking</td>
<td>Cooking Extended</td>
<td>Cooking Interrupted</td>
<td></td>
<td>Cooking Complete</td>
</tr>
<tr>
<td>Cooking Complete</td>
<td>Cooking</td>
<td>Door Open</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cooking Interrupted</td>
<td>Event Ignored</td>
<td></td>
<td>Ready To Cook</td>
<td>Event Ignored</td>
</tr>
<tr>
<td>Door Open</td>
<td>Event Ignored</td>
<td></td>
<td>Ready To Cook</td>
<td></td>
</tr>
<tr>
<td>Cooking Extended</td>
<td>Cooking Extended</td>
<td>Cooking Interrupted</td>
<td></td>
<td>Cooking Complete</td>
</tr>
</tbody>
</table>

*State Transition Table with “Event Ignored” Entries*
**Can't Happen cell entry**

If the event cannot happen when the object is in a particular state, record the fact by entering "Can't Happen" in the cell.

The *Can't Happen* entry is reserved for occasions when the event simply cannot occur according to the rules and policies of the domain. For example, the oven cannot receive a door Closed event when the oven is cooking. The door is already closed in this state; detecting this event in this state makes no sense.
Figure 9.9 from M&B

<table>
<thead>
<tr>
<th>State</th>
<th>buttonPressed</th>
<th>doorOpened</th>
<th>doorClosed</th>
<th>timerTimesOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready To Cook</td>
<td>Cooking</td>
<td>Door Open</td>
<td>Can't Happen</td>
<td>Event Ignored</td>
</tr>
<tr>
<td>Cooking</td>
<td>Cooking Extended</td>
<td>Cooking</td>
<td>Can't Happen</td>
<td>Cooking Complete</td>
</tr>
<tr>
<td>Cooking Complete</td>
<td>Cooking</td>
<td>Door Open</td>
<td>Can't Happened</td>
<td>Can't Happen</td>
</tr>
<tr>
<td>Cooking Interrupted</td>
<td>Event Ignored</td>
<td>Can't Happen</td>
<td>Ready To Cook</td>
<td>Event Ignored</td>
</tr>
<tr>
<td>Door Open</td>
<td>Event Ignored</td>
<td>Can't Happen</td>
<td>Ready To Cook</td>
<td>Can't Happen</td>
</tr>
<tr>
<td>Cooking Extended</td>
<td>Cooking Extended</td>
<td>Cooking</td>
<td>Can't Happen</td>
<td>Cooking Complete</td>
</tr>
</tbody>
</table>

State Transition Table with "Can't Happen" Entries
Actions

- Turn on the light
- Turn off the light
- Energize the power tube
- De-energize the power tube
- Set the timer for 1 minute
- Add 1 minute to the timer
- Clear the timer
Figure 9.7 from M&B