Specification of Model Behavior

Lecture Topics

Behavioral Model

State Machines

Relation to Other Behavioral Models

Derivation of State Machines for Single Classes

Derivation of State Machines from System Level

Behavioral Models

1. Software system is collection of interacting entities.
2. Each entity has a behavior by which it interacts with other entities.
3. We have discussed a method of identifying entities and representing them as classes.
4. We must determine and specify the behaviors of each class.
5. Behavior:
    1. What the class does
    2. How it interacts with other classes
6. We must determine and specify the behaviors of the entire system.

Behaviors as State Machines

- We model the behavior of each class as a state machine.
- A state machine is specified for each *active* class. An *active* class has multiple states.
- A separate state machine is created for each instance of a class that is created. That is: the behavior of each instance of a class (object) is controlled by an instance of the class state machine which acts on the local values of attributes of the class instance.
- The behavior of the entire system is determined by the interactions among the state machines of the classes.
- We can derive state machines for single classes or use system level structure to derive state machines. (More later on this.)

State Machines

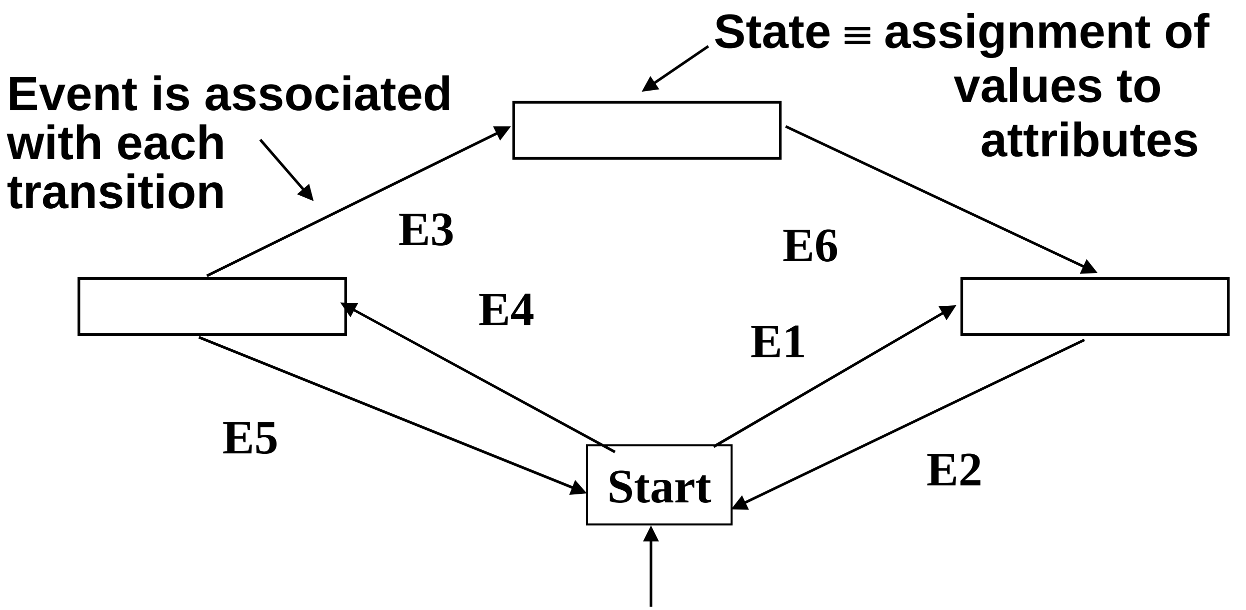
***States*. Each state represents a stage in the life cycle of a typical instance of the class.**

***Events*. Each event represents an incident that causes a a state transition to happen.**

***Transitions*. A transition rule specifies what new state is achieved when an object in a given state receives a particular event.**

***Procedures*. A procedure is an activity or operation that must be accomplished when an object arrives in a state. Each state has its own procedure. Procedures comprise *actions*.**

STATE MODELS



Associated with every state is an action which is executed upon entry to the state and a selection of exits which is executed at the end of the action. The actions generate the events which cause state transitions.

There is a single state model for each class but a separate instance of the state model for each instance of the class. Just like just Java methods are common to all class instances.

Types of State Machines

1. **Moore - A state machine in which the present state depends only on its previous input and previous state, and the present output depends only on the present state. An action (possibly no-op) is associated with a state. The action associated with a state is executed upon entry to the state.**

2. **Mealy - A type of state machine in which the outputs are a function of the inputs and the current state. The actions are associated with transitions. An action is executed when a given transition is taken.**

3. **Statecharts – An integration of Moore and Mealy machines which includes invocation on both entry to a state and exit from a state and a number of further complexities.**

4. **xUML uses a Moore state machine subset of Statecharts**

Execution Models for Programs

1. Control Flow Graph = State Machine (single classes)

   sequential

2. Execution Tree at Procedure level

   sequential

3. Data Flow – Data flow graphs (system level)

   Parallel or interleaved

4. Object-Oriented – Method invocations

   Sequential

4. Interacting State Machines

   sequential, parallel or interleaved

Execution Models for Systems of Interacting State Machines

- Asynchronous Interleaved
- Asynchronous Parallel
- Event queuing – with or without?
- Atomicity of action execution?
- Communication – Entity to entity or broadcast/multicast?

Approach

- UML has three diagrams, use cases, sequence diagrams and collaboration diagrams to represent execution of a classes or interacting classes.
  - Use Cases - http://www.agilemodeling.com/artifacts/sequenceDiagram.htm
  - Sequence Diagrams - http://www.agilemodeling.com/artifacts/sequenceDiagram.htm
  - Collaboration Diagrams - http://www.agilemodeling.com/artifacts/communicationDiagram.htm
- We summarize these into tabular or spreadsheet models.
  - Object-Based Analysis Paper

Microwave Oven Specification

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.

Environment Specification – Use Cases

Door is opened
Door is closed
Button is pushed
Timer completes cooking period.
Use Case: some sequence of the set of externally originated events which starts from a specified state.
Basic Sequence: Initial state DoorClosed(DC)

> **Door is opened;**
> **Door is closed;**
> **button is pushed[i];**
> **Timer completes;**
> **Door is opened.**

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – Cook item in oven for two minutes

Originator	Event	Initial State	Action	New State
User	Open Door	RC	State change$	DO

$ Event to be sent to Light

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – Cook item in oven for two minutes

Originator	Event	Initial State	Action	New State
User	Open Door	RC	State change$	DO
User	Close Door	DO	State change$	RC

$ Event to be sent to Light

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – Cook item in oven for two minutes

Originator	Event	Initial State	Action	New State
User	Open Door	RC	State change	DO
User	Close Door	DO	State change	RC
User	Push Button	RC	Turn on PT,L and T*	C

* Composite action involving sending events to other entities.

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – Cook item in oven for two minutes

Originator	Event	Initial State	Action	New State
User	Open Door	RC	State change	DO
User	Close Door	DO	State change	RC
User	Push Button	RC	Turn on PT,L and T*	C
User	Push Button	C	Add minute to T	CE

* Composite action involving sending events to other entities.

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – Cook item in oven for two minutes

Originator	Event	Initial State	Action	New State
User	Open Door	RC	State change	DO
User	Close Door	DO	State change	RC
User	Push Button	RC	Turn on PT,L and T*	C
User	Push Button	C	Add minute to T	CE
Timer	Timer Interrupt	CE	Off PT,L and T*	CC

* Composite action involving sending events to other entities.

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – Cook item in oven for two minutes but interrupt cooking to stir after one and a half minute.

Originator	Event	Initial State	Action	New State
User	Open Door	RC	State change	DO
User	Close Door	DO	State change	RC
User	Push Button	RC	Turn on PT,L and T*	C
User	Push Button	C	Add minute to T	CE
User	Open Door	CE	Off PT,L and T*	CI

* Composite action involving sending events to other entities.

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – User Error

Originator	Event	Initial State	Action	New State
User	Open Door	RC	State Change	DO
User	Push Button	DO	No Change	DO

How to represent in State Model??

Use Cases – Spreadsheet or Table

Pre-Condition: Door closed, nothing in oven

Script – Expand Event Sequence from Actions

Originator	Event	Receiver	Action	New State
User	Open Door	Oven/RC	State change#	Oven/DO
User	Close Door	Oven/DO	State change$	Oven/RC
User	Push Button	Oven/RC	Turn on PT,L and T*	Oven/C
Oven	On PT	PT/Off	PT state transition	PT/On
Oven	On Light	L/Off	State change	L/On
Oven	On Timer	T/Off	State change	T/On
User	Push Button			

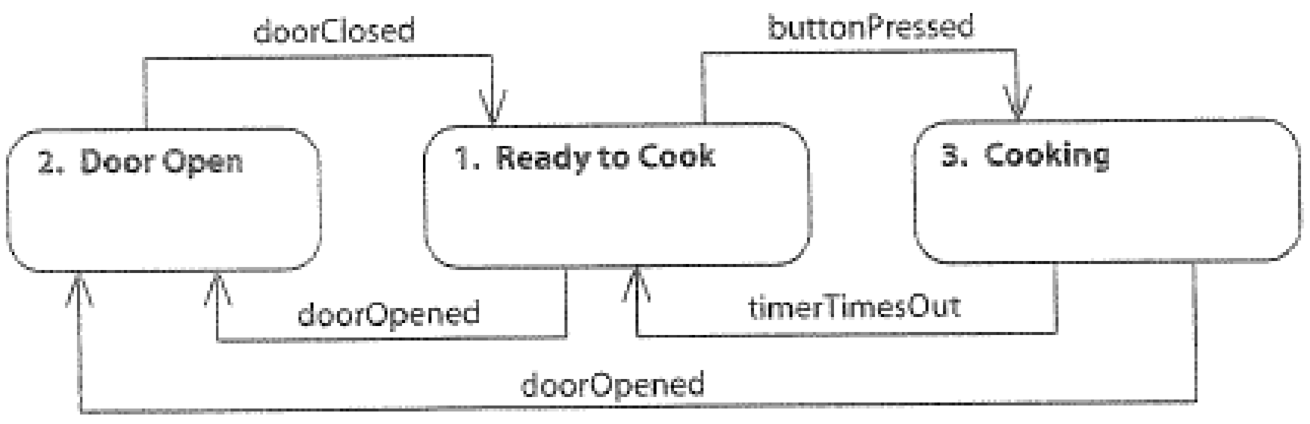
Light should be turned on

$ Light should be turned off

* Composite action involving sending events to other entities.

State Machine View of Use Cases

State machine for all useful sequences



External View of the Microwave Oven

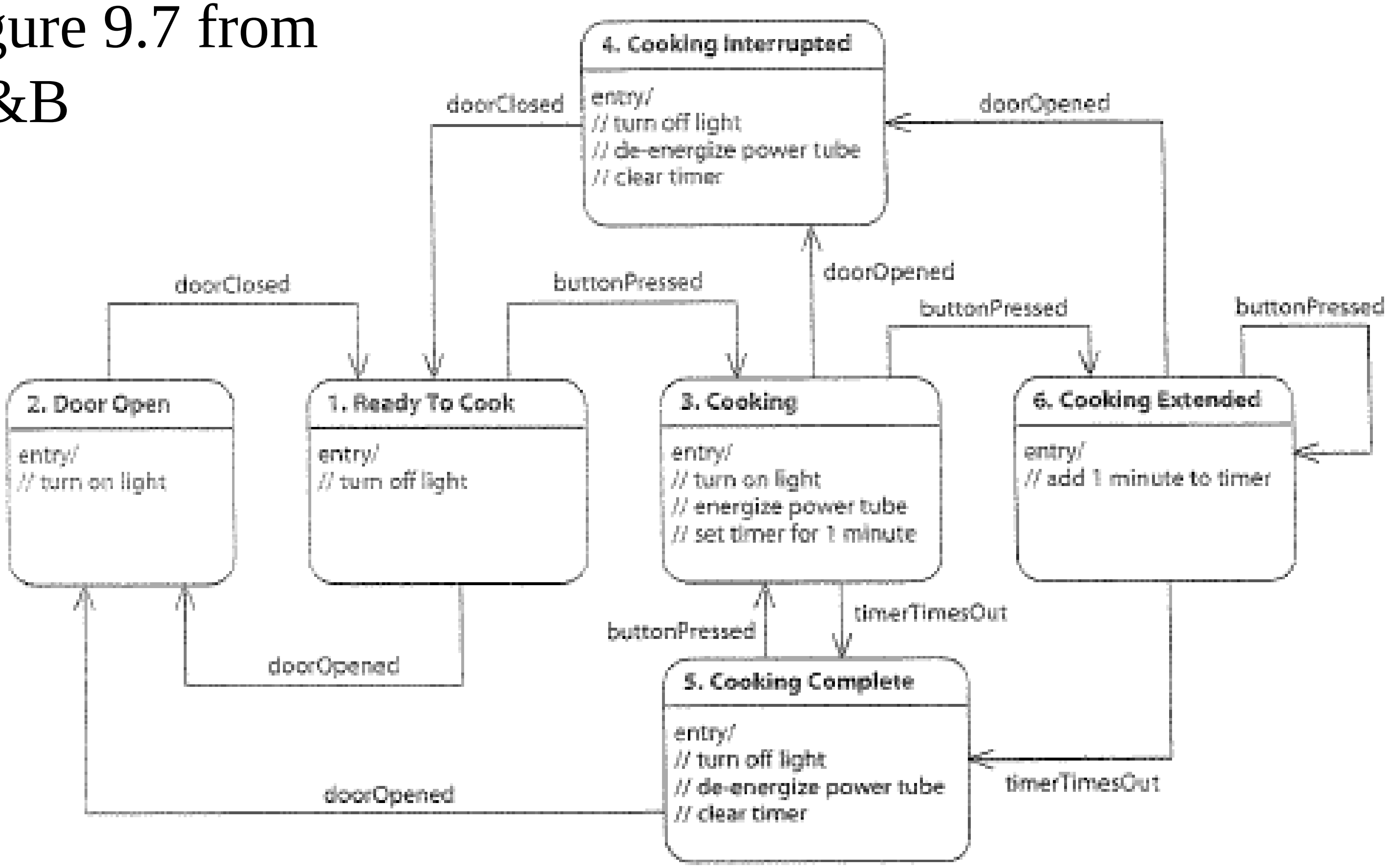
User-Environment with actions

State Machine for Arbitrary Sequence of User Actions


```
microwaveId=userId;
actionsRemaining=5;
Generate U1: act (userId);
```

```
CHOICE=ANY("Decision_Type");
if(CHOICE==1)
    Generate MW1: buttonPressed (microwaveId);
else if(CHOICE==2)
{
    if(Microwave(microwaveId).doorOpen)
        Generate MW3: doorClosed (microwaveId);
    else
        Generate MW2: doorOpened (microwaveId);
}
else
    if(Microwave(microwaveId).tubeOn)
        Generate MW4: timerTimesOut (microwaveId);
actionsRemaining=actionsRemaining-1;
if(actionsRemaining>0)
    Generate U1: act (userId);
```

Figure 9.7 from M&B



Rules For Constructing State Models – Individual Classes

1. Determine the allowed states of the object.
2. Determine the set of allowed transitions.
3. Determine each event which causes a state transition to occur.
4. Define the actions for each state transition.
5. Build state transition graphs.
6. Verify that state transitions are atomic.
7. Verify that actions are context free.
8. Build state transition table.

A Possible Set of Consistency Rules For State Models

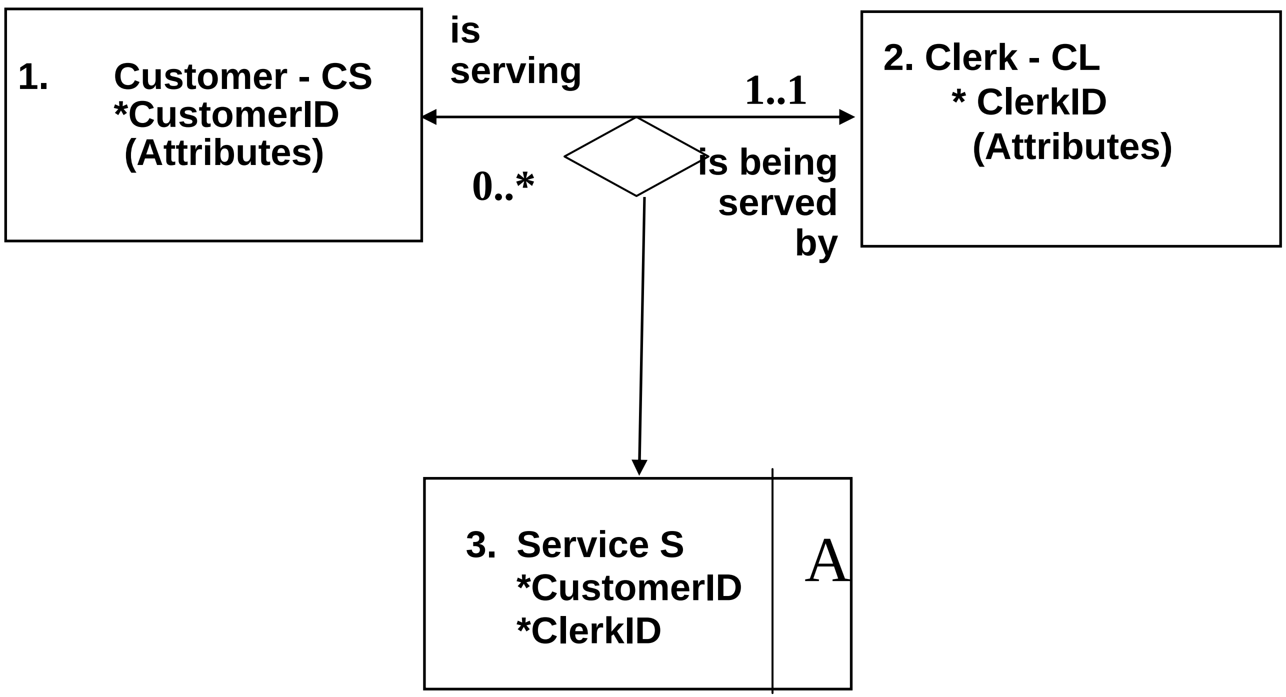
1. A given state machine executes only one action at a time.
2. Multiple state machines can be simultaneously active. (for different objects or different instances of the same object)
3. An action takes time to execute.
4. Actions are atomic.
5. Multiple events can be outstanding for a given instance of a state model. Events are never LOST.
6. Events are consumed by the execution of the receiving action.
7. At the end of the execution of the action associated with the acceptance of an event, the state model is in the new state.
8. Generated events are instantaneously available.
9. A state machine always accepts pending events as quickly as possible.
10. Events from a given source are received in order as generated.
11. Event receipt from multiple sources is non-deterministic.

Competitive Relationships

Consider a service and commission management system for an upscale Women's Ware Store. The customers come into the store and wait for a clerk to show them merchandise. There are typically only a few clerks and at some times of day there are more customers than clerks.

This is an example of a competition for resources. It is representative of a commonly occurring circumstance. The competition for resources requires a special type of state machine which has access to a set of data spanning multiple classes – an assigner state machine.

State Models for Associative Objects



Problems: 1. State models execute for <u>instances</u> of objects

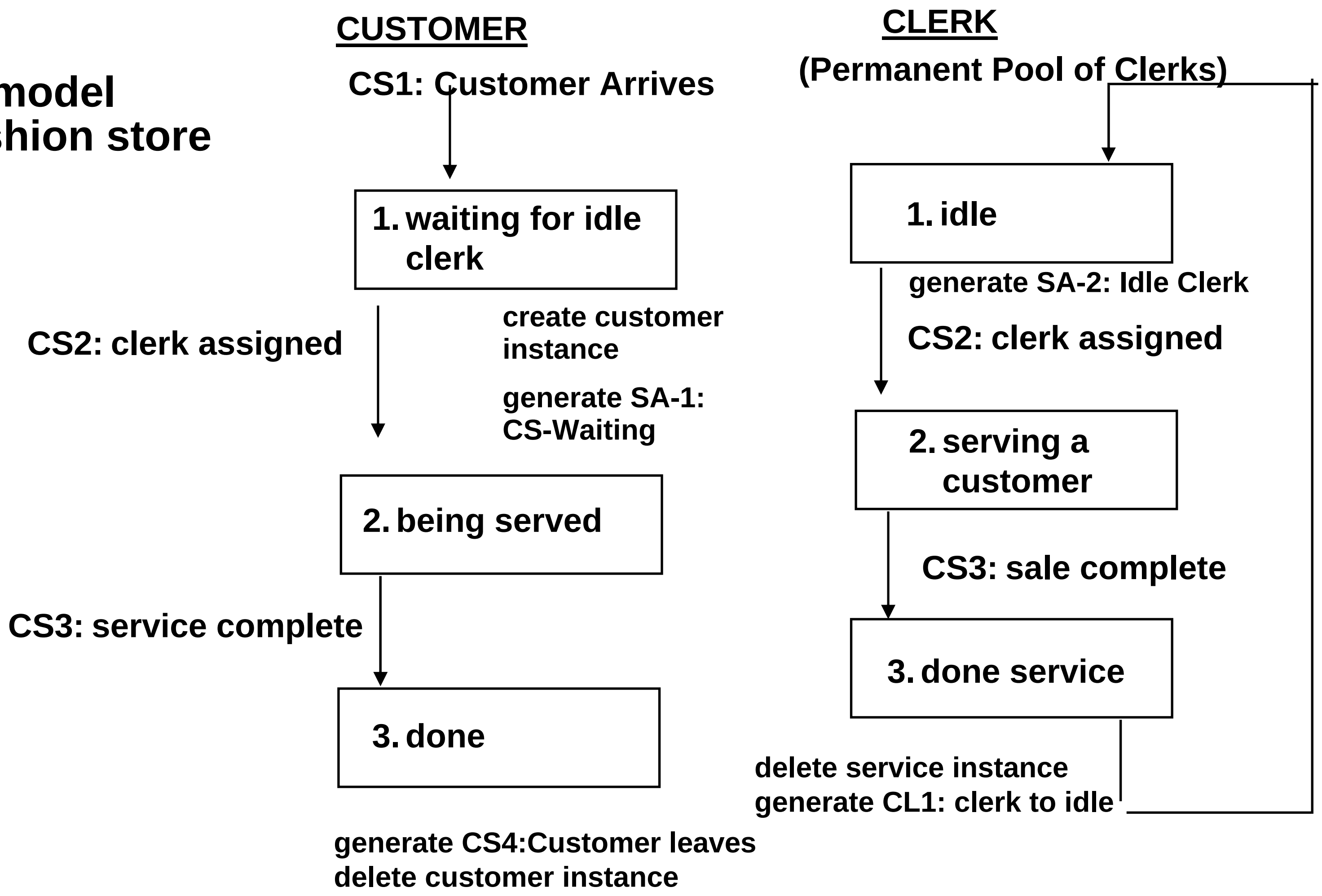
but

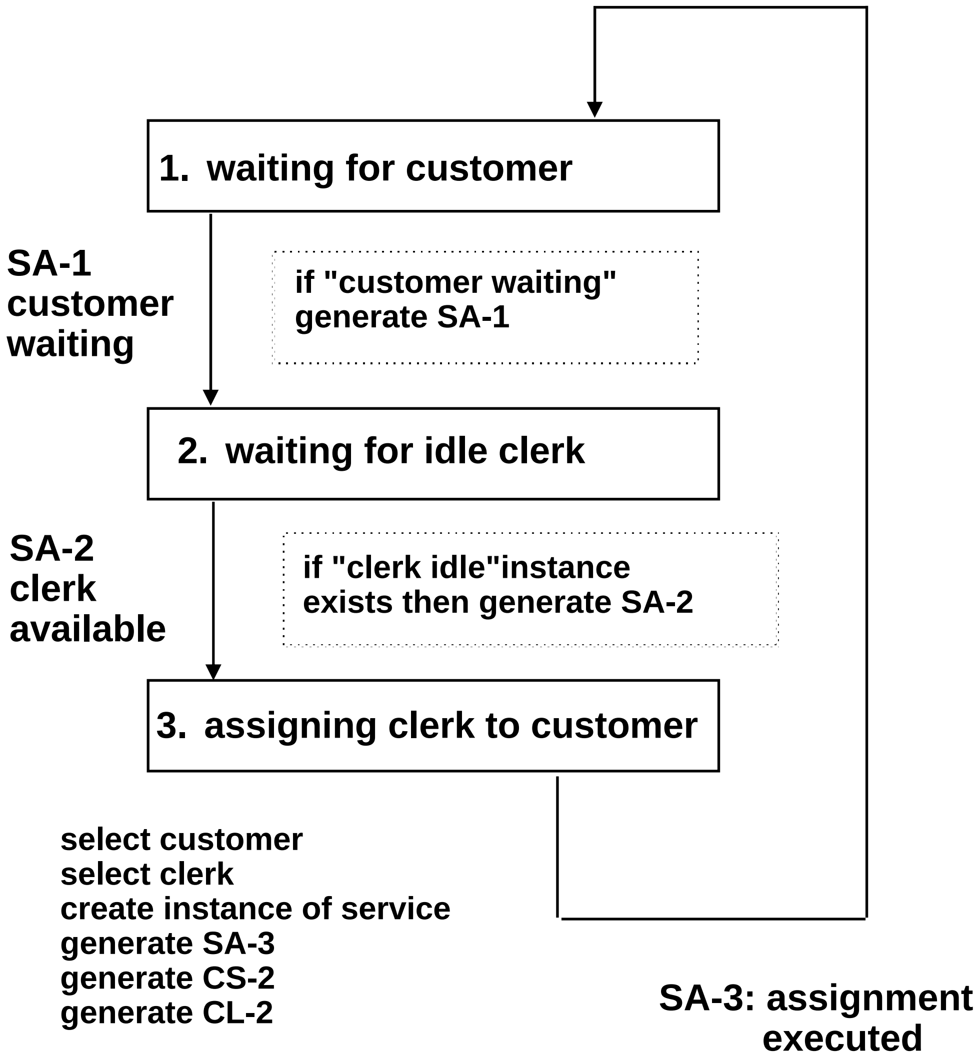
a service object may not be created until both customer and clerk are chosen.

2. Assume there are more clerks than customers. Multiple clerks may descend upon a single customer and a fight may occur.

Solution: Create state machine of special properties to queue and match events as well as manage service object instances – class-based state machine → assigner state machines.

State model for fashion store





Class State Model for Associative Class Service – Assigner State Model

What is knowledge assumed to be available to the assigner model in this instance?

Parking Garage Arm Motor Controller

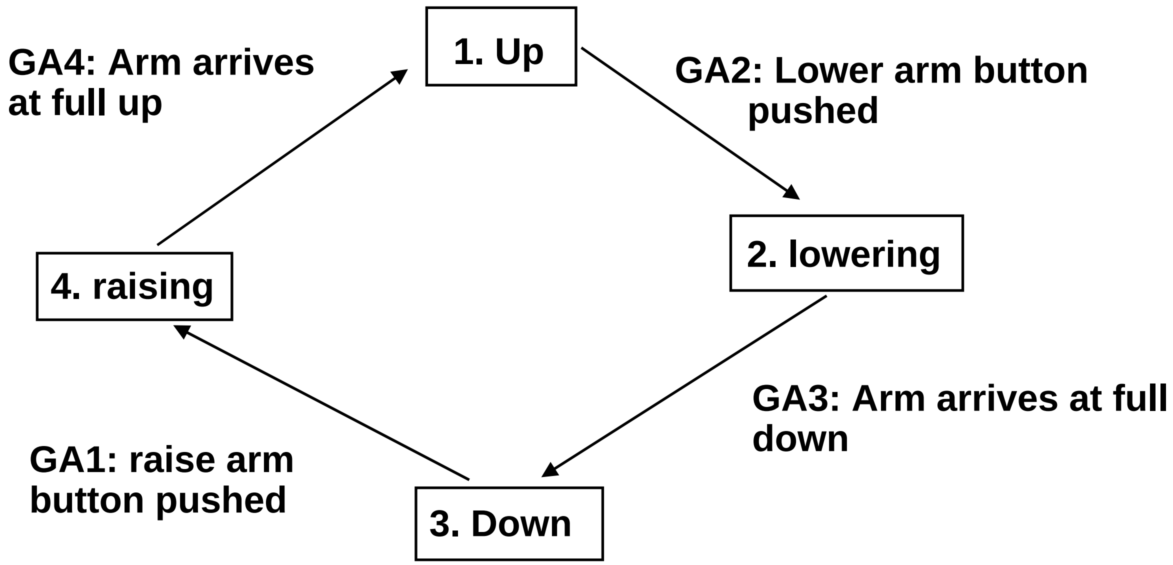
There is a sensor controlled parking garage arm which controls admission to a parking garage. Its rest position is down. When a car arrives and the ticket button is pressed the arm rises to allow the car into the garage. When the car passes through the gate the driver must push a button on the garage side of the gate to make the arm descend. The motions of the arm take a finite duration to execute.

Garage Arm *** Arm ID** **. status**

States and Events for Garage Arm Motor

States - down,raising,up, lowering

Events - turn on motor to raise arm (GA1)
turn off motor after raising arm (GA4)
turn on motor to lower arm (GA2)
turn off motor after lowering arm (GA3)



Railroad Crossing Access Arm Example

State Transition Table

event / action	GA1	GA2	GA3	GA4	
1. **up**		lowering			turn motor off
2. lowering			down		turn motor on
3. down	raising			**up**	in down mode
4. raising					**turn motor on in up mode**

State Transition Table

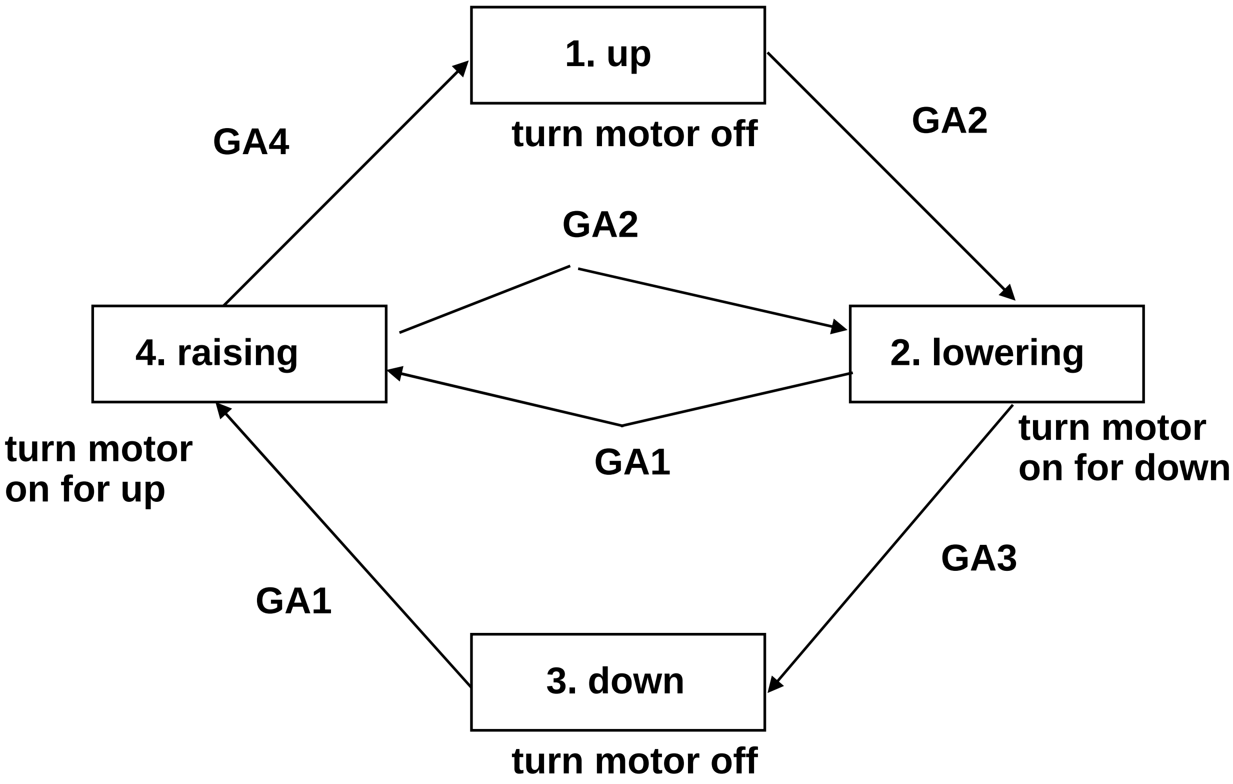
event	GA1	GA2	GA3	GA4	action
1. up		lowering			turn motor off
2. lowering			down		turn motor on
3. down	raising				turn motor off
4. raising				up	turn motor on

event	GA1 up button pushed	GA2 down button pushed		GA4 arm arrives at full up	action
1. up	event ignored	lowering	can't happen	can't happen	turn motor off
2. lowering	?	event ignored	down	can't happen	turn motor on for up
3. down	raising	event ignored	can't happen	can't happen	turn motor off
4. raising	event ignored	?	can't happen	up	turn motor on for up

GA1 up button pushed

GA2 down button pushed

event	GA1	GA2	GA3	GA4	action
1. up					
2. lowering	raising?				
3. down					
4. raising		**lowering?**			


1. up
turn motor off
GA2
GA4
GA2
4. raising
2. lowering
turn motor
on for up
GA1
turn motor
on for down
GA3
GA1
3. down
turn motor off

FAULT TOLERANCE

Up to 80% of system code may be error analysis and recovery

OOA provides a means for integration of error analysis and fault tolerance directly in system design.

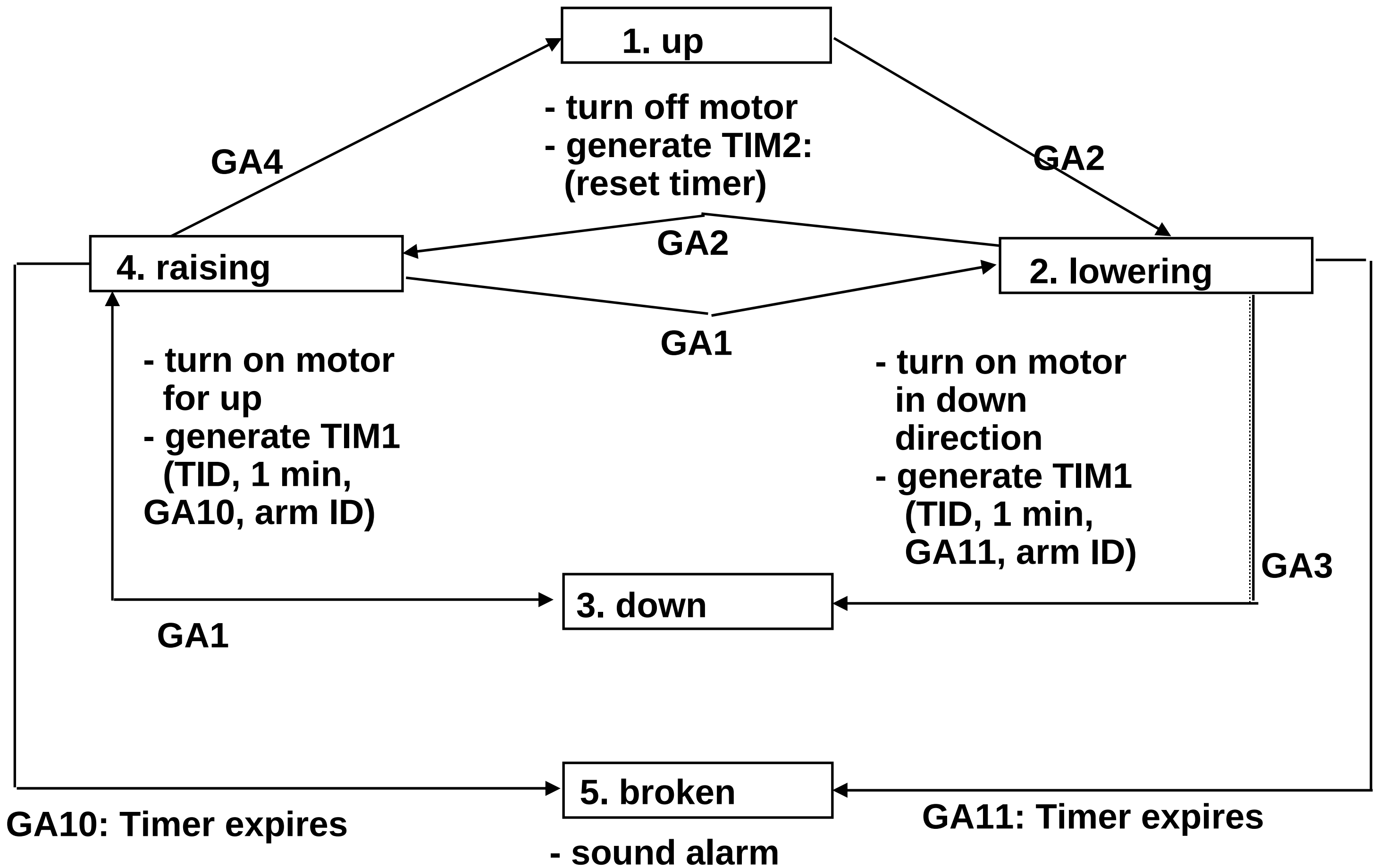
Information Model

- **External attribute domains to include fault conditions and recognizable errors**
- **Define consistency relations among attribute values**

State Models

- **Define events resulting from faults**
- **Define state models to support diagnosis**
- **Extend state models to include fault states**
- **Design actions to diagnose for errors on initiation of action**
- **Integrate time-out behavior into state model**

Garage Arm Example



Note: Extension of attribute domain requires redo of all parts of OOA