Literature Review Preview
Review of Design Patterns
Discrete Event Models
Major Types of Software System

1. Batch application
2. Interactive (desktop applications)
3. Embedded or real-time application
4. Control systems (robots, manufactory)
5. Distributed data or processes
6. Data centric
7. Knowledge-based
8. Peer-to-peer and client-server
9. Web-based applications
10. New types: mobile applications, game software.

How is UML used to model these systems?


Dr. Xiaolin Hu
Some Problems

1. An elevator control program
2. A traffic light program
3. A Screen Saver Program
4. ......

Develop a state chart diagram for the screen saver program.
Finite State Machine (from NIST)
(http://www.nist.gov/dads/HTML/finiteStateMachine.html)

- **Definition:** A *model of computation* consisting of a set of *states*, a *start state*, an input *alphabet*, and a *transition function* that maps input symbols and current states to a *next state*. Computation begins in the start state with an input string. It changes to new states depending on the transition function. There are many variants, for instance, machines having actions (outputs) associated with transitions (*Mealy machine*) or states (*Moore machine*), multiple start states, transitions conditioned on no input symbol (a null) or more than one transition for a given symbol and state (*nondeterministic finite state machine*), one or more states designated as *accepting states*, etc.

- **Also known as** finite state automaton.
- **Generalization** *model of computation*, *Turing machine*.
- **Specialization** *deterministic finite state machine*, *finite state transducer*, *Markov chain*, *hidden Markov model*.
Formal FSM Model

A set of primary inputs: \( X \)

A set of primary outputs: \( Y \)

A set of states: \( S \)

A state transition function: \( \delta: X \times S \rightarrow S \)

An output function:

\( \lambda: X \times S \rightarrow Y \) for Mealy machine

\( \lambda: S \rightarrow Y \) for Moore machine
A Real World Example

You are in the kitchen (state) and you felt a sudden urge to drink water. You then decide to walk over to the cupboard and grab a glass. This is an event (walk to cupboard) and you are now in a new state with new options available to you. You then open the cupboard (action) and grab a glass (action). With the glass, you walk over to the sink (event) and turn on the tap (action). Water poured into the glass (event) and you then drink it (action).

How to differentiate action from event?
A possible Finite State Machine implementation

Input events may be generated internally and externally.

Inputs trigger rules

State Transition

State

Outputs generated by the current state

Dr. Xiaolin Hu
Example: a lamp

Push switch

On

Push switch

Off
A Moore Machine is a type of finite state machine where the outputs are generated as products of the states. In the below example the states define what to do; such as apply power to the light globe.
Mealy Machine

A Mealy Machine, unlike a Moore Machine is a type of finite state machine where the outputs are generated as products of the transition between states. In below example the light is affected by the process of changing states.

Dr. Xiaolin Hu
An Example
Source: http://ai-depot.com/FiniteStateMachines/

Consider the game Quake

- A rocket in Quake is a projectile fired from the Rocket Launcher weapon which may be possessed and operated by a human player.
Let’s take a look at a slightly more advanced FSM from Quake. A Shambler is a big bad monster entity from in Quake. Its mission in life is to kill the player, once it is aware of the player.

https://www.youtube.com/watch?v=busfAqVwyv4
A robot example

<table>
<thead>
<tr>
<th>Event</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>turnOn</td>
<td>Activated</td>
</tr>
<tr>
<td>turnOff</td>
<td>Deactivated (Idle)</td>
</tr>
<tr>
<td>stop</td>
<td>Stopped</td>
</tr>
<tr>
<td>walk</td>
<td>Walking</td>
</tr>
<tr>
<td>run</td>
<td>Running</td>
</tr>
<tr>
<td>raiseLeftArm</td>
<td>LeftArmRaised</td>
</tr>
<tr>
<td>lowerLeftArm</td>
<td>LeftArmLowered</td>
</tr>
<tr>
<td>lowerLeftArm</td>
<td>LeftArmLowered</td>
</tr>
<tr>
<td>raiseRightArm</td>
<td>RightArmRaised</td>
</tr>
<tr>
<td>lowerRightArm</td>
<td>RightArmLowered</td>
</tr>
<tr>
<td>turnHead</td>
<td>HeadTurned(direction)</td>
</tr>
<tr>
<td>speak</td>
<td>Talking(text)</td>
</tr>
</tbody>
</table>

Table: 1.1. Small list of known events with Bender

Source: http://www.generation5.org/content/2003/FSM_Tutorial.asp
Two problems exist in this FSM model.

This model representation is wrong. In this model, we can only have one state at a time with no transition between them. It is possible for Bender to both walk and talk, or first walk then run, so there should be a link between those two states.
Another State Model

Diagram:

- **Idle**
- **TurnedOn**
- **Activity**

Transitions:
- turnOn
- turnOff
- requestTurnOff
- walk
- talk(text)
- run
Implementation of FSM

Implement the logic for the following FSM.

- screen_On
- screen_Off

If 5 minutes expire

keyboard or mouse

Dr. Xiaolin Hu
Implementation of FSM

- Switch statement
- State design pattern

```java
switch(stateIndex) {
    case k:
        switch(event){
            case i:
                action(event);
                stateIndex = m;
                break;
            case j:
                ......
        }
    break;
    case m:
    ...
}
```

Design the “state design pattern”. Consider the previous example.
Write the pseudo code for the Screen Saver program using the state design pattern.
Consider a subway turnstile. It can be locked, or it can be unlocked. When the turnstile is locked, a person can drop a coin into its slot. This will unlock the turnstile to allow the person to pass. After the person passes, the turnstile locks again.

- What event will cause the turnstile to lock again? Timer, or sensor detector?
- Abnormal logic
- A better way to handle alarm
- Diagnosis mode
- How to implement the FSM
Advantages of FSM

• Predictability (in deterministic FSM), given a set of inputs and a known current state, the state transition can be predicted, allowing for easy testing
• Due to their simplicity, FSMs are quick to design, quick to implement and quick in execution
• FSMs are relatively flexible. There are a number of ways to implement a FSM based system in terms of topology, and it is easy to incorporate many other techniques
• Easy to transfer from a meaningful abstract representation to a coded implementation
• Easy determination of reachability of a state, when represented in an abstract form, it is immediately obvious whether a state is achievable from another state, and what is required to achieve the state
Disadvantages of FSM

• The predictable nature of deterministic FSMs can be unwanted in some domains such as computer games (solution may be non-deterministic FSM).

• Larger systems implemented using a FSM can be difficult to manage and maintain without a well thought out design.

• Not suited to all problem domains, should only be used when a systems behavior can be decomposed into separate states with well defined conditions for state transitions. This means that all states, transitions and conditions need to be known up front and be well defined. For example: consider the design of an autonomous robot to be operated in an unknown environment.

• The conditions for state transitions are ridged, meaning they are fixed (this can be overcome by using a Fuzzy State Machine (FuSM))
Applying the idea of FSM to GUI Development
– The DEVS-FIRE software example
A Petri net is a graphical and mathematical modeling tool. It consists of **places**, **transitions**, and **arcs** that connect them. **Input arcs** connect places with transitions, while **output arcs** start at a transition and end at a place.

- Places can contain **tokens**; the current state of the modeled system (the *marking*) is given by the number of tokens in each place.
- Transitions model activities which can occur (the transition *fires*), thus changing the state of the system (the marking of the Petri net).
- Transitions are only allowed to fire if they are *enabled*, which means that all the preconditions for the activity must be fulfilled (there are enough tokens available in the input places).
Figure 1. Petri net elements and firing sequence (a to d).
Petri Net (cont.)

• When the transition fires, it removes tokens from its input places and adds some at all of its output places. The number of tokens removed / added depends on the cardinality of each arc.

• Petri nets are a promising tool for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic.

• As a graphical tool, Petri nets can be used as a visual-communication aid. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems.

• As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems.
A Petri net is a quadruple $(P, T, F, W)$

- Finite set of places $P$ (denoted by circles)
- Finite set of transition $T$ (denoted by bars)
- Finite set of arrows $F$ connecting either places to transitions or transitions to places. Arrows are described by the flow relation
  \[ F \subseteq (P \times T) \cup (T \times P) \]

- A weight function $W: F \rightarrow N - \{0\}$ which associates a non-zero natural number to each element of $F$. If no weight is associated with an arrow, default weight of 1 is assumed.
Firing of transitions

- A transition is **enabled** if each of its input places contains a number of tokens that is greater than or equal to the weight of the arrow connecting the input place to the transition.
- An enabled transition may **fire**.
- Firing of a transition $T$ removes from each input place $P_i$ the number of token equal to the weight of arrow from $P_i$ to $T$ and then inserts into each output place $P_j$ the number of tokens equal to the weight of arrow from $T$ to $P_j$. 
An Example

![Diagram of an example with states and transitions before and after firing]

Dr. Xiaolin Hu
Graphic Representation

A switch to turn on/off light
In modeling a system using a Petri net, usually,

- A *transition* represents an *event or an activity*.
- *Firing* of a transition represents *occurrence* of the event or *execution* of the activity.
- Presence of a token in a place represents *existence* of a condition.

For example, a place may model a resource and existence of tokens in the place indicate the availability of the resource.

**Problem**: Two threads share a common memory. At any time, only one thread is allowed to access the memory. Develop a petri net model for it.
Features

• Petri nets are characterized by their ability to handle operation sequence, concurrency, conflict and mutual exclusion in systems.

• These features make them a promising tool for describing and analyzing concurrent and real-time systems.
Figure 3. Using Petri nets to model some of the real-time system features.
Warehouse Automation Example

Specifies requirements of a system
• a truck arrives at the loading dock
• paperwork is processed and inventory checked
• conveyor belts move from the loading dock to the warehouse (transporting people, robots, forklifts, etc)
• conveyor belt move back from warehouse to loading dock with merchandise.
• goods loaded into the truck
• P1 => a truck arrives at the loading dock
• P2 => paperwork is processed and inventory checked
• P3 => conveyor belts move from the loading dock to the warehouse (transporting people, robots, forklifts, etc)
• P4 => conveyor belt move back from warehouse to loading dock with merchandise.
• P5 => goods loaded into the truck
• How to handle multiple trucks?
• Concurrent handling of multiple trucks can be modeled by multiple tokens.
• **Problem**: the conveyor belts cannot move in two opposite directions at the same time.

• A token is present in P3, then another token cannot be present in P4 (the conveyor belts cannot move in two opposite directions at the same time).
• Therefore, the above model can be modified as follows to specify the above requirements of the system.

• Now a token can enter P3 only if there is no token in P4
Similarly, to specify the requirements that only paperwork of one truck can be processed at a time and merchandise of only one truck can be loading into the truck at a time, the above model can be modified as follows.
A Flexible Manufactory System (FMS) Example

Consider the operations of a manufacturing cell comprising of one machine and one robot.
A Flexible Manufactory System (FMS) Example

Model the sequence of Machine
Model the sequence of Robot
Question: how about there are two robots? two machines?

Loading part includes “picking up parts” and “dropping parts”, can a robot picks up a part before machine becomes idle?

Question: how about there are two robots? two machines?
Timed Petri Nets

- Time is introduced in Petri nets to model the interaction among several activities considering their starting and completion times.
- Time may be associated with Places (TPPN)
  - Tokens generated in an output place become available to fire a transition only after a delay has elapsed.
  - The delay is an attribute of the place.
- Time may be associated with Tokens
  - Tokens carry a time stamp that indicates when they are available to fire a transition.
- Time may be associated with Transitions (TTPN)
  - Transitions represent activities.
  - Activity Start corresponds to Transition Enabling.
  - Activity End corresponds to Transition Firing.
Formal treatment and properties of Petri nets

- **Reachability**  this determines whether a system can reach a specific state or exhibit a particular functional behavior. The reachability set can be denoted by $R(M_0)$, where $M_0$ is the initial marking.

- **Liveness**  this detects whether deadlock situation will be occurred in the system or not.

- **Boundedness and Safeness**  a Petri net is said to be bounded and safe if no overflow condition is detected.

- **Conservativeness**  a Petri net is described as conservative if the number of tokens in the model remains constant irrespective of the markings it takes on.
A reference

Introduction to Discrete Event Systems
Christos G. Cassandras, and Stéphane Lafortune