Functional Requirements: Behavioral Requirements

- Overview
- Decision-oriented Behavioral Models
- State-oriented Behavioral Models
  - Finite State Machines (Protocol Validation & Verification)
  - Statecharts
  - SCR
  - Petri Nets
- Function-oriented Behavioral Models

Requirements should contain nothing but information about the environment.

\[ M^G, \text{Prog}^G \models S_G; S_G, D^G \models R_G; R_G, D^G \models G; (G \models \neg P) \lor (G \models \neg P) \]
Behavioral Models

- Behavioral models offer facilities for modelling the behavior of a process or activity

- Can be classified into:
  
  - decision-oriented models: view a process as an algorithm for deciding the outcome of an event, given some set of inputs
    
    e.g., decision on a loan application, given info. about the applicant
    
    e.g., decision tables/trees
  
  - state-oriented models: view a process as a collection of "states" and "transitions" among those states
    
    e.g., Finite State Machines (FSMs), StateCharts, Petri Nets
  
  - function-oriented models: define a process in terms of a set of "preconditions" which must be true before the process is initiated, and "postconditions" which must be true after the process is completed.
Decision-Oriented Behavioral Models

How do we specify what a process does ("process logic")?
How do we represent policies?

- Decision tables enumerate parameters (or "conditions")
  and "actions" (or "outcomes") that should take.

"If the plane is more than half full and the flight costs more than $350 per seat
serve free cocktails, unless it is a domestic flight. Charge for cocktails in all
domestic flights where cocktails are served, i.e., those that are more than full."

<table>
<thead>
<tr>
<th>Domestic</th>
<th>Y</th>
<th>Y</th>
<th>Y</th>
<th>Y</th>
<th>N</th>
<th>N</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= half full</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>&gt;= $350/seats</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Serve cocktails</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>✗</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Free cocktails</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>

- Each condition is usually {T, F, don’t-care};
  but can also be multi-valued {never, rarely, sometimes, often, always}
- total # of columns = total # of rules = X (#-input-cond)

Lawrence Chung
### Natural Language

“The system shall report to the operator all faults that originate in critical functions or that occur during execution of a critical sequence and for which there is no fault recovery response.”
(adapted from the specifications for the international space station)

### A decision table

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>originate in critical functions</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Occur during critical sequence</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>No fault recovery response</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Report to operator???</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
State-Oriented Behavioral Models

- describe a system in terms of states and transitions

- a **state** represents the static properties of a system at a particular time; states may have a duration and may have associated invariants.
  
  e.g., patient-waiting (in a doctor’s office) may have as duration the length of the wait and invariant that the person waiting wants to see the doctor.

- a **transition** represents a dynamic change of the system being modelled from one state to another;
  
  transitions may have triggers and associated actions
  
  e.g., the see-doctor transition changes the state of the system from patient-waiting to patient-examined

---

Lawrence Chung
Finite State Machines (FSMs)

% A finite state machine (FSM) is an abstract machine which can be in exactly one state at any given time;

% While in a certain state Sc, a FSM accepts an input (I) and produces an output (O), while at the same time it changes state to Sn;

% Both output and next state depend only on current state and input:

\[ \text{Sn} = F(\text{Sc}, \text{I}) \]
\[ \text{O} = G(\text{Sc}, \text{I}) \]

% Two common notations for defining FSMs:

state transition diagrams (STDs) and state transition tables (STTs)

STDs:

\[ \begin{array}{c}
\text{state} \\
\text{transition} \\
\text{input/output} \ (\text{stimulus/response})
\end{array} \]

Example:

\[ \begin{array}{c}
\text{idle} \quad \text{on hook/ make quiet} \\
\text{dial tone} \quad \text{dial busy number/ gen busy tone} \\
\text{ring back tone} \\
\text{distinct dial tone} \\
\text{dial 9/ gen distinctive dial tone} \\
\text{caller off hook/ gen dial tone} \\
\text{dial idle number/ gen ring back tone}
\end{array} \]

requirements should contain \textit{nothing but} information about the environment. [Zave&Jackson, p9]
Finite State Machines (FSMs)

STDs:
- state
- transition
- input/output (stimulus/response)

Example:
- on hook/make quiet
- caller off hook/gen dial tone
- dial idle number/gen ring back tone
- dial busy number/gen busy tone
- busy
- distinct dial tone
dial 9/gen distinctive dial tone

STTs:

<table>
<thead>
<tr>
<th>stimuli</th>
<th>states</th>
<th>next state response</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>idle</td>
<td>dial tone</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>caller off hook</th>
<th>on hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>dial tone gen dial tone</td>
<td>idle make quiet</td>
</tr>
</tbody>
</table>

AS-IS or TO-BE?
Model or Anti-model?

Lawrence Chung

state invariants?
Object-oriented?
Protocol Validation and Verification

Using Finite State Machine

What, Why & How

Master Station  HMI or GUI  OPC Server
Slave Protocols

Master Protocols

Remote Terminal Unit  Meters  IEDs  Relays

http://www.ase-systems.com/images/sptdiag.gif

Lawrence Chung
Protocol Validation & Verification: What?

**Specification**

Validation:
ensure that a protocol specification is equivalent to real needs, or at least, logically consistent.

**Implementation**

Verification (Conformance Test):
ensure that the external behavior of a protocol implementation is equivalent to its specification

Real Needs
Protocol Validation & Verification: Why?

- Complexity:

  Communication Protocols in distributed systems and open communication networks are highly complex.

  But, an ad hoc approach can make the problem more complex.

- Error-prone:

  Communication Protocols often contain errors.
  Specifications are ambiguous, incomplete, inconsistent, ...

  But, an informal approach has its limitation.

After all, what good is an international standard that is incomplete or even faulty?
General Types of Protocol Errors:

- **Deadlock:**
  A global state reachable from initial global state, in which no transmissions are possible.

- **Non-executable Interaction:**
  A reception/transmission that is specified in the design but never executed.

- **Unspecified Reception:**
  A reception that is executable but not specified in the design.

- **Overflow:**
  A channel state in which the number of messages exceeds some predefined bound.
Protocol Validation & Verification: How?

The Finite State Machine Approach

Example: How to model communication protocols using finite state machine?

Process 1 (Client)          Process 2 (Server)

- Ready
  -REQ
  +DONE
  -ACK

- Wait

- Register
  +ALARM

- Service
  +REQ
  -DONE
  +ACK

- Fault
  -ALARM

A simple protocol between 2 processes: Client & Server

Lawrence Chung
Error Detection: How?

The Finite State Machine Approach

Example:

Process 1

-1

2

+2

3

Process 2

-3

2

+1

-2

3

deadlock: 1 2 3 1 2 3

non-executable interaction: 1 2 3 1 2 3 +1

unspecified reception: 1 3 1 2 -2

Lawrence Chung
Validation & Verification Techniques:

- **Reachability Analysis:**
  - Full Search
  - Controlled Partial Search
  - Random Search

- **Conformance Test:**
  - Test for every state and input that the protocol outputs according to specification.
  - Generate conformance test suite
  - Apply test suite to implementation

Of interest to international standardization bodies
Statecharts

- a generalization of FSM notations, intended to facilitate the definition of large FSMs, defined in terms of states and transitions too.
- however, more than one state may be "on" at any one time (for concurrency); moreover, statecharts consist of several FSMs composed in terms of "AND" or "OR" compositions.
- Conditional transition: transition from state S1 to S2 is to take place if there is input i and condition C is true.

Example:

in FSM:

- caller off hook/gen dial tone
- dial tone
- dial idle number/gen ring back tone
- dial busy number/gen busy tone
- distinct dial tone
- dial 9/gen distinctive dial tone
- ring back tone
- idle
- on hook/make quiet
- busy

Lawrence Chung
Statecharts

Superstates: aggregate sets of states with common transitions

Example:

- idle
  - caller off hook → caller active
  - caller on hook

- caller active
  - dial local number (callee idle)
  - gen ring back tone
  - ring back tone
  - dial local number (callee active)/gen busy tone

- dial tone
  - busy

Lawrence Chung
Statecharts

OR decomposition: One of the subordinate states is "on"

AND decomposition: All of the subordinate states are "on"

"orthogonal" decomposition: state machines S1 and S2 are independent

Valid configurations?
SCR: Tabular Specifications

Recall: Four Variable Model

### SCR Specifications
\[ \Sigma = (S, S_0, E^m, T) \]

#### Monitored/Controlled Variables

- **TY(Pressure)** = \{TooLow, Permitted, High\}
- **TY(WaterPres)** = \{0; 1; 2; \ldots; 2000\}
- **TY(Overridden)** = \{true, false\}
- **TY(Block)** = \{On; Off\}

### Mode Transition Tables

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TooLow</td>
<td>( \neg T(\text{WaterPres} &gt; \text{Low}) )</td>
<td>Permitted</td>
</tr>
<tr>
<td>Permitted</td>
<td>( T(\text{WaterPres} &gt; \text{Permit}) )</td>
<td>High</td>
</tr>
<tr>
<td>Permitted</td>
<td>( \neg T(\text{WaterPres} &lt; \text{Low}) )</td>
<td>TooLow</td>
</tr>
<tr>
<td>High</td>
<td>( \neg T(\text{WaterPres} &lt; \text{Permit}) )</td>
<td>Permitted</td>
</tr>
</tbody>
</table>

Table 1: Mode Transition Table for Pressure.

### Event Tables

<table>
<thead>
<tr>
<th>Mode Class</th>
<th>Pressure</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>False</td>
<td>( \neg (\text{Pressure}=\text{High}) )</td>
</tr>
<tr>
<td>TooLow, Permitted</td>
<td>( \neg T(\text{Block}=\text{On}) )</td>
<td>( \neg (\text{Pressure}=\text{High}) ) OR ( T(\text{Reset}=\text{On}) )</td>
</tr>
<tr>
<td>Overridden</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 2: Event Table for Overridden.

### Condition Tables

<table>
<thead>
<tr>
<th>Mode Class</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High, Permitted</td>
<td>True</td>
</tr>
<tr>
<td>TooLow</td>
<td>Overridden</td>
</tr>
<tr>
<td>Safety Injection</td>
<td>Off</td>
</tr>
</tbody>
</table>

Table 3: Condition Table for Safety Injection.
SCR Basics

- **Modes and Mode classes**
  - A mode class is a FSM, with states called system modes
  - Transitions in each mode class are triggered by events
  - Complex systems are described using a number of mode classes operating in parallel

- **System States**
  - The system is in exactly one mode from each mode class…
  - Each variable has a unique value

- **Events**
  - An event occurs when any system entity changes value
    - An input event occurs when an input variable changes value
    - Single input assumption - only one input event can occur at once
    - Notation: $@T(c)$ means “c changed from false to true”
  - A conditioned event is an event with a predicate
    - $@T(c) \text{ WHEN } d$ means: “c became true when c was false and d was true”
    - $@T(c) \text{ WHEN } d \stackrel{\text{def}}{=} \neg c \land c' \land d$
    - $@F(c) \text{ WHEN } d$
**SCR: Example - Safety Injection System (SIS)**

**Goal:**
Maintain safety of a nuclear reactor.

**Domain:**
1. Water pressure ranges between 0.0 and 2000.0 psi (pounds/square inch);
2. Low < Permit

**Requirements:**
1. Perform safety injection when water pressure is too low;
2. Operator can either block or reset safety injection.
**SCR: Mode Classes**

- **Mode Class Tables**
  - Define a (disjoint) set of modes (states) that the software can be in.
  - A complex system will have many different modes classes
  - Each mode class has a mode table
  - A mode table defines a partial function from modes and events to modes $M_n = F_m(M_c, E)$

- **Example:**

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TooLow</td>
<td>@T(WaterPres ≥ Low)</td>
<td>Permitted</td>
</tr>
<tr>
<td>Permitted</td>
<td>@T(WaterPres ≥ Permit)</td>
<td>High</td>
</tr>
<tr>
<td>Permitted</td>
<td>@T(WaterPres &lt; Low)</td>
<td>TooLow</td>
</tr>
<tr>
<td>High</td>
<td>@T(WaterPres &lt; Permit)</td>
<td>Permitted</td>
</tr>
</tbody>
</table>

Table 1: Mode Transition Table for Pressure.
SCR: Controlled Variables

Event Tables

- Defines how a controlled variable changes in response to input events
- Defines a partial function from modes and events to variable values
- Example:

\[ V = F_d(M_c, E) \]

Disjointness Property: the pairwise conjunction of conditions (events) in each row of a condition (an event) table must always be false.

Example:

\[
\text{Overridden}' = \begin{cases} 
  \text{true} & \text{if} \ (\text{Pressure} = \text{TooLow} \land \text{Block'} = \text{On} \land \\
  \quad \quad \text{Block} = \text{Off} \land \text{Reset} = \text{Off}) \lor \\
  \quad \quad (\text{Pressure} = \text{Permitted} \land \text{Block'} = \text{On} \land \\
  \quad \quad \text{Block} = \text{Off} \land \text{Reset} = \text{Off}) \lor \\
  \quad \quad (\text{Pressure} = \text{TooLow} \land \text{Reset'} = \text{On} \land \\
  \quad \quad \text{Reset} = \text{Off}) \lor \\
  \quad \quad (\text{Pressure} = \text{Permitted} \land \text{Reset'} = \text{On} \land \\
  \quad \quad \text{Reset} = \text{Off}) \lor \\
  \quad \quad (\text{Pressure'} = \text{High} \land \text{Pressure} \neq \text{High}) \\
  \text{false} & \text{if} \ 	ext{Overridden} \\
 \text{otherwise} & 
\end{cases}
\]

Table 2: Event Table for Overridden.
SCR: Controlled Variables

**Condition Tables**
- Defines the value of a controlled variable under every possible condition
- Defines a total function from modes and conditions to variable values

**Example:**

\[ V = F_c(M_c, \mathcal{C}) \]

**Disjointness Property:** the pairwise conjunction of conditions (events) in each row of a condition (an event) table must always be false.

**Coverage Property:** the disjunction of the conditions in each row of the table must be true.

\[
\text{SafetyInjection} = \begin{cases} 
  \text{Off} & \text{if } \text{Pressure} = \text{High} \lor \text{Pressure} = \text{Permitted} \lor \\
  & (\text{Pressure} = \text{TooLow} \land \text{Overridden} = \text{true}) \\
  \text{On} & \text{if } \text{Pressure} = \text{TooLow} \land \text{Overridden} = \text{false}.
\end{cases}
\]

<table>
<thead>
<tr>
<th>Mode Class Pressure</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High, Permitted</td>
<td>True</td>
</tr>
<tr>
<td>TooLow</td>
<td>Overridden</td>
</tr>
<tr>
<td>Safety Injection</td>
<td>Off</td>
</tr>
</tbody>
</table>

Table 3: Condition Table for Safety Injection.
### SCR: Another Example

#### Mode Class Table

<table>
<thead>
<tr>
<th>Current Mode</th>
<th>Powered on</th>
<th>Too Cold</th>
<th>Temp OK</th>
<th>Too Hot</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>@T</td>
<td>-</td>
<td>t</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>@T</td>
<td>t</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>@T</td>
<td>-</td>
<td>-</td>
<td>t</td>
<td>-</td>
</tr>
<tr>
<td>Inactive</td>
<td>@F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@T</td>
<td>-</td>
<td>-</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Heat</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AC</td>
</tr>
<tr>
<td>Heat</td>
<td>@F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inactive</td>
</tr>
<tr>
<td>AC</td>
<td>@F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inactive</td>
</tr>
</tbody>
</table>

#### Event Table

<table>
<thead>
<tr>
<th>Modes</th>
<th>@C(target)</th>
<th>never</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heat, AC</td>
<td>@C(target)</td>
<td>never</td>
</tr>
<tr>
<td>Inactive, Off</td>
<td>never</td>
<td>@C(target)</td>
</tr>
<tr>
<td>Ack_tone =</td>
<td>Beep</td>
<td>Clang</td>
</tr>
</tbody>
</table>

#### Condition Table

<table>
<thead>
<tr>
<th>Modes</th>
<th>target - temp (\leq 5)</th>
<th>target - temp (&gt; 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inactive, Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warning light =</td>
<td>Off</td>
<td>On</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EFSMs, Statecharts & SCR

In dialtone: @T(offhook) WHEN callee_offhook -> Ringing
In ringtone: @F(offhook) -> Idle
EFSMs, Statecharts & SCR

All 3 models are approximately equivalent

☐ State machine models
  □ Emphasis is on states & transitions
  □ No systematic treatment of events
  □ Different event semantics can be applied
  □ Graphical notation enhances understandability
  □ Composition achieved through Statechart nesting
  □ Hard to represent complex conditions on transitions
  □ Hard to represent real-time constraints (e.g. elapsed time)

☐ SCR models
  □ Emphasis is on events
  □ Clear event semantics based on changes to environmental variables
  □ Single input assumption simplifies modeling
  □ Tabular notation easy to understand, modulo complex relationships (?)
  □ Composition achieved through parallel mode classes
  □ Hard to represent real-time constraints (e.g. elapsed time)
Petri Nets

- A form of FSMs, first proposed by C. A. Petri in 1962
- A notation for defining abstract concurrent processes
- Primitives
  - place
  - transition
  - token

When all input places of a transition are enabled (i.e., with a token) and the external stimulus associated with the transition occurs, the tokens move from the input places to output places.

Example:

**BEFORE**

![Before Diagram]

**AFTER**

![After Diagram]

The particular Petri Net notation here makes a single token assumption and avoids no inhibitor arcs [LK].

Lawrence Chung
Petri Nets

Fundamental notions:
- Sequencing
- Concurrency
- Synchronization
- Decision making
Example

*Stimuli occur in the following order: T1, T3, T5, T4, T6*
Augmented Petri Net

- **Activation conditions and actions**
  - e.g., for a patient waiting to see a doctor, an activation condition may be "doctor ready" and an associated action may be "patient move to doctor’s office"

- **Invariants on states:**
  - e.g., "patient not feeling well"

- **Communication primitives:**
  - Actions may involve communication with other Petri nets; e.g., "finding out test results" by communicating with the "labTests" Petri net

---

Lawrence Chung
Expressive Power Revisited:

"Can we precisely specify behavior using OO-methodology?"
   [BOF session, by Booch & Rumbaugh, OOPSLA, 94]

"Q: ... I don’t see any methods providing something for precise
    specification of behavior."
   [... Booch turns the question around ...]

"Q: ... Well, again, I’m not asking to precisely specify everything.
    I’m asking for the method to give me some way to precisely
    specify something when I need to precisely specify.
    Otherwise the programmer will not know what I’m saying."
   [Rumbaugh: I am not a great believer in formal languages ...]

"Q: ... I’m asking for some way to precisely specify behavior."

Finally, Grady Booch answered:

"I understand. Short answer is we’re getting defensive here,
   Hmm... Short answer is look at our pre- and post- conditions."

Lawrence Chung
Function-Oriented Behavioral Models

- describe what the process does, often in terms of pre/postconditions
  e.g., room-booking activity:

  BookMeetingRoom (room, time, who)
  pre: not (exists w) Booked (room, time, w)
  post: Booked’ (room, time, w)

  "Precondition" provides a predicate that must be true in the state prior to the initiation of the process

  "Postcondition" describes what will be true after the process execution

  Predicates and functions are unprimed/primed depending on whether they refer to the state before/after the activity respectively.

  Specialization of activities:
  e.g., To book a talk, we also need to book a projector

  BookTalk (room, time, who) IS-A BookMeetingRoom
  pre: not (exists r) AVRequested (r, time, overhead)
  post: AVRequested’ (room, time, overhead)