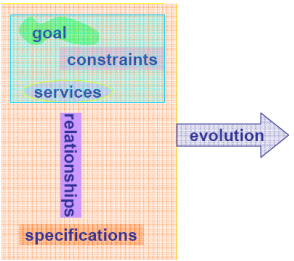
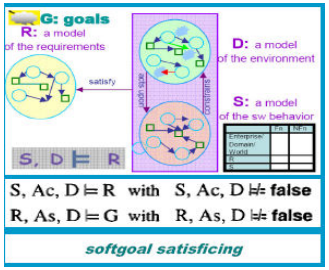
Functional Requirements: *Behavioral* Requirements

- ❑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.

$$M^G, Prog^G \models S^G;\ S^G, D^G \models R^G;\ R^G, D^G \models G;\ (G \models \neg P) \vee (G \models\sim \neg P)$$



Recall:

- Functional Requirements (FRs)
  - Structural Functional Requirements
    - Functional, i.e., Function-oriented
    - Informational. i.e., Information-oriented
  - Behavioral Functional Requirements
- Non-Functional Requirements (NFRs)

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

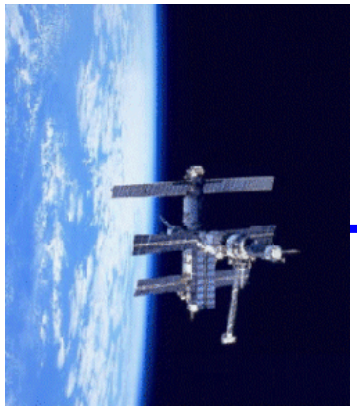
How do we specify what a process does ("process logic")?
How do we represent policies?

- 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."

Domestic	Y	Y	Y	Y	N	N	N	N
>= half full	Y	Y	N	N	Y	Y	N	N
>= $350/seat	Y	N	Y	N	Y	N	Y	N
Serve cocktails	X	X			X	?	?	?
Free cocktails					X			

- 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)



Decision Tables:
Another Ambiguity Test

- 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

originate in critical functions	F	T	F	T	F	T	F	T
Occur during critical sequence	F	F	T	T	F	F	T	T
No fault recovery response	F	F	F	F	T	T	T	T
Report to operator???								

State-Oriented Behavioral Models

- describe a system in terms of states and transitions

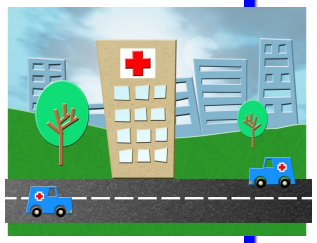
- a state represents the static properties of a system at a particular time;

  states my 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.



www.piperreport.com

- 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

Finite State Machines (FSMs)

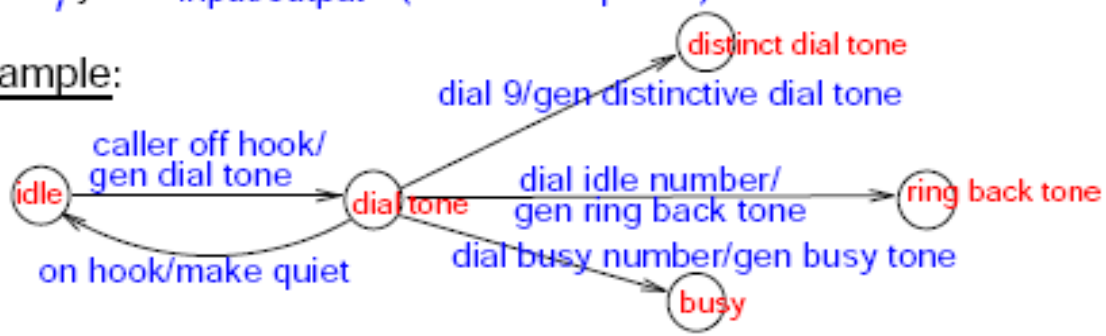
- A finite state machine (FSM) is an abstract machine which can be in exactly one state at any given time;
- While in a certain state Sc, a FSM accepts an input (I) and produces an output (O), while at the same time it changes state to Sn;
- Both output and next state depend only on current state and input:

$$Sn = F(Sc, I)$$
$$O = G(Sc, I)$$

- Two common notations for defining FSMs:
  state transition diagrams (STDs) and state transition tables (STTs)

  STDs:

Example:



. . . requirements should contain *nothing but* information about the environment. [Zave&Jackson, p9]

Finite State Machines (FSMs)

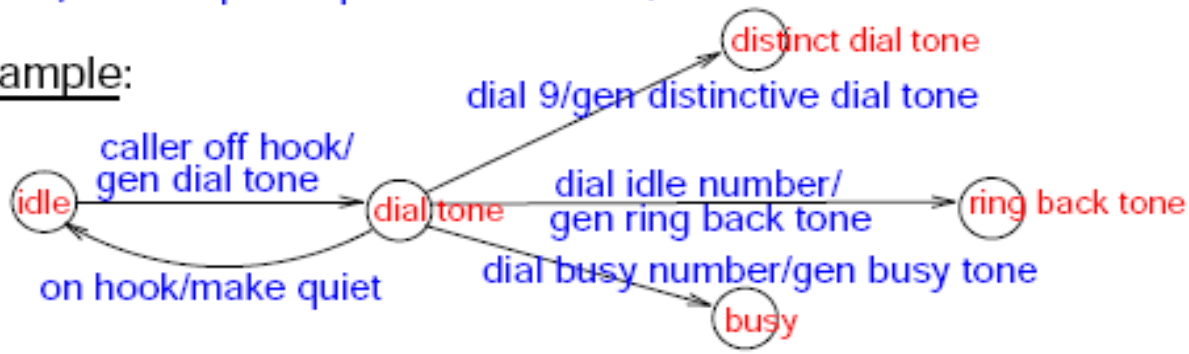
STDs:

state

transition

zxcv/hjkl input/output (stimulus/response)

Example:



STTs:

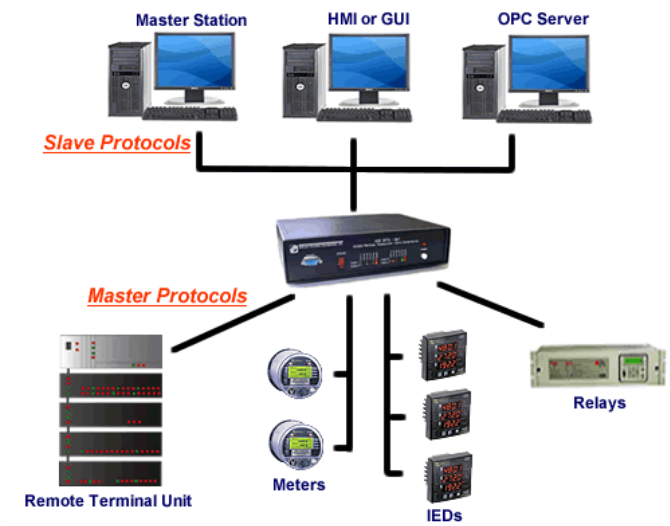
	stimuli		
states			
		next state response	

	caller off hook	on hook	
idle	dial tone gen dial tone		
dial tone		idle make quiet	

Protocol Validation and Verification

Using Finite State Machine

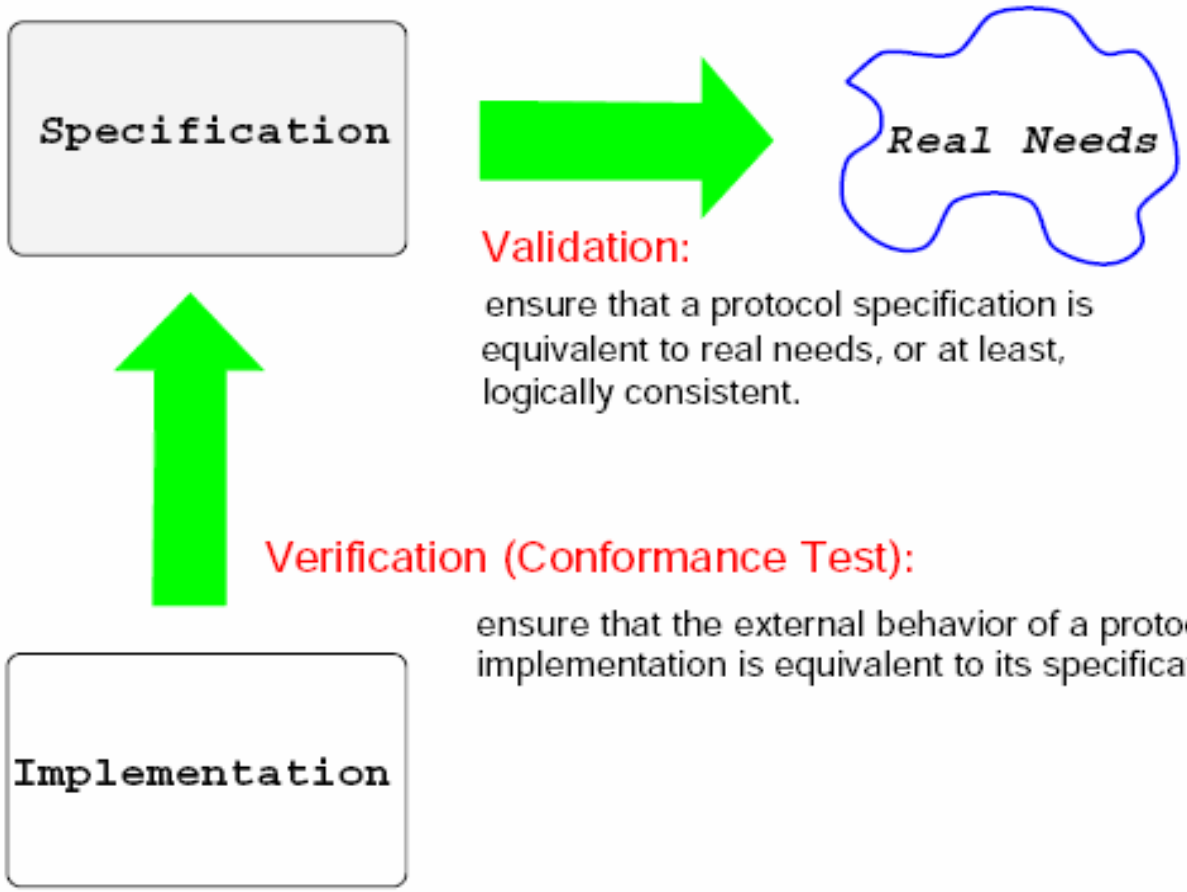
What, Why & How



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

Lawrence Chung

Protocol Validation & Verification: What?



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:**

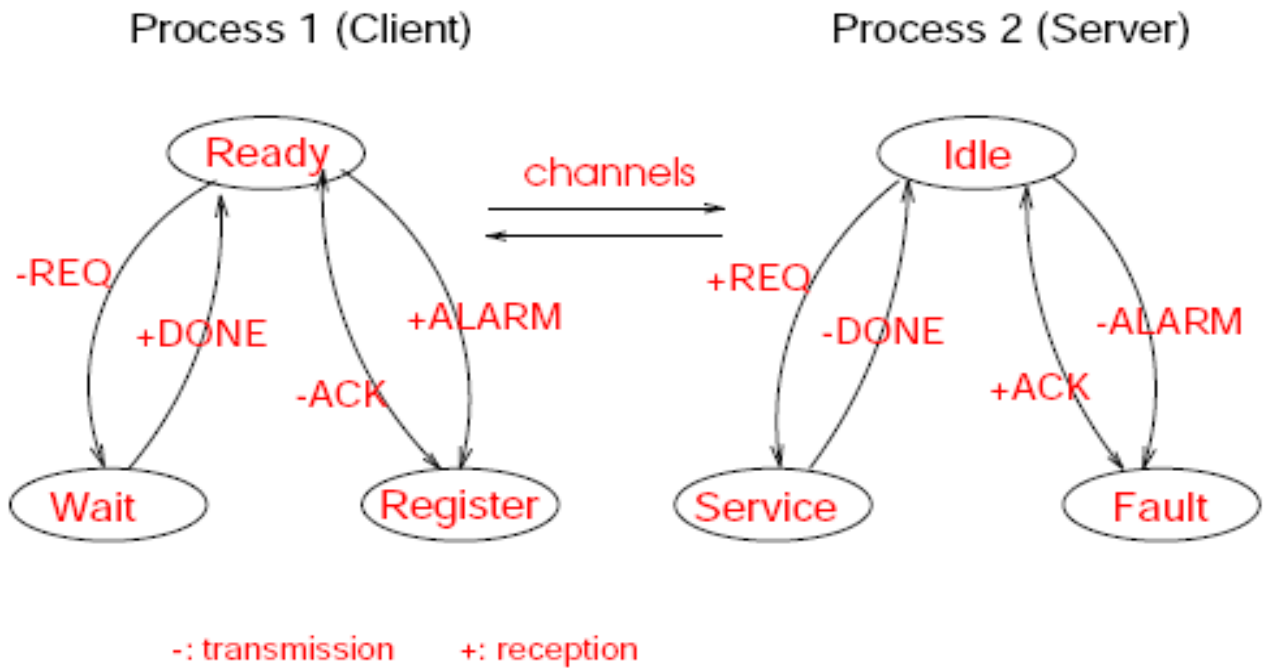
  A channel state in which the number of messages exceeds some predefined bound.

- • •

Protocol Validation & Verification: How?

The Finite State Machine Approach

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

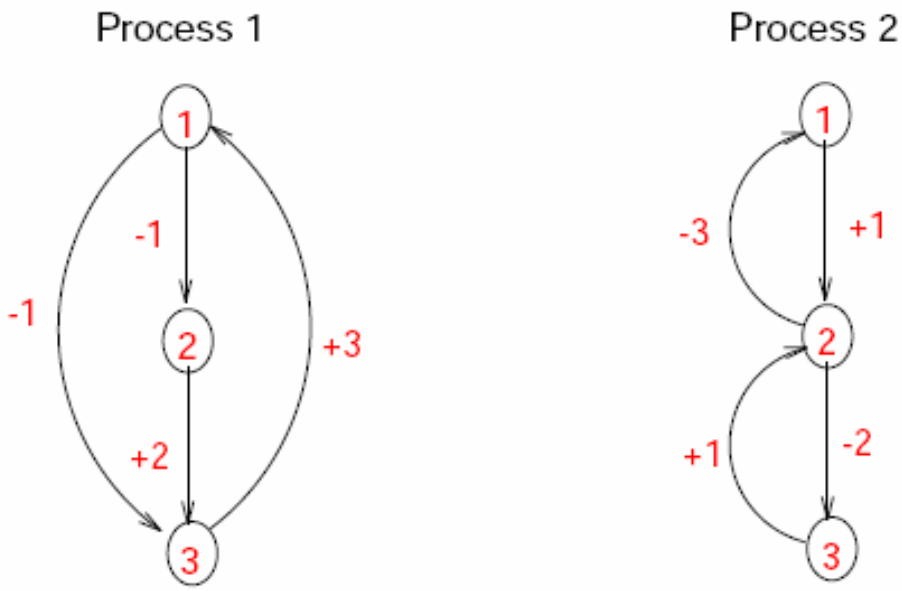


A simple protocol between 2 processes: Client & Server

Error Detection: How?

The Finite State Machine Approach

Example:



deadlock:	1 2 3	1 2 3
non-executable interaction:	1 2 3	1 2 3 +1
unspecified reception:	1 3	1 2 -2

Validation & Verification Techniques:

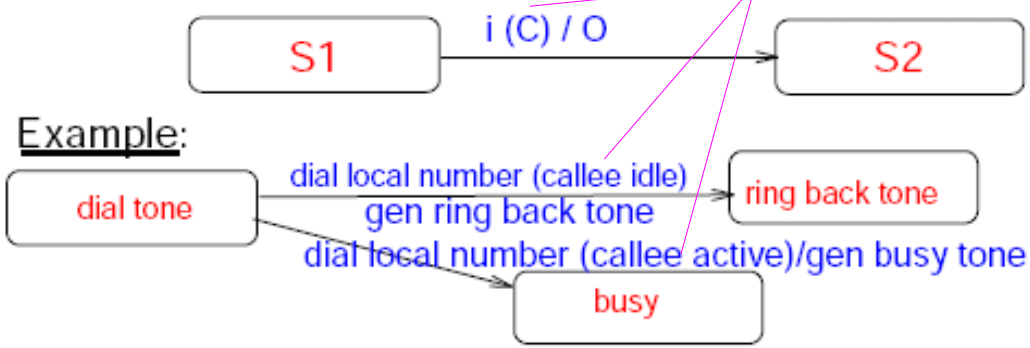
- Reachability Analysis:
  - (Full Search
  - (Controlled Partial Search
  - (Random Search
- Conformance Test:

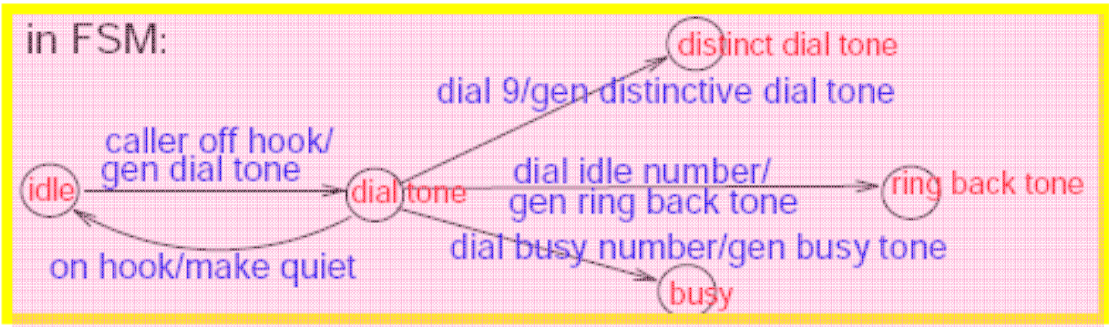
  test for every state and input that the protocol outputs according to specification.
  - (Generate conformance test suite
  - (Apply test suite to implementation

Of interest to international standardization bodies

Statecharts

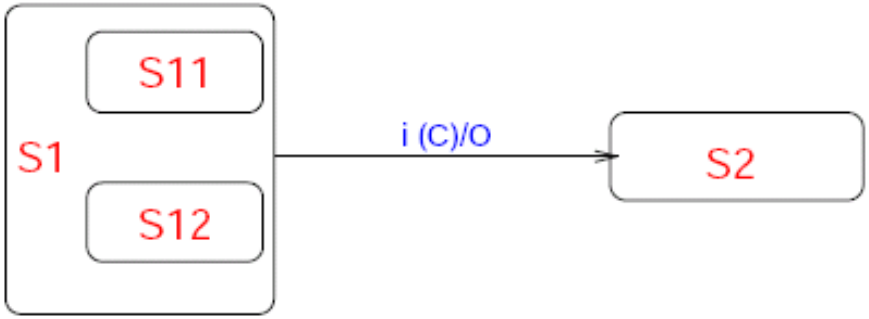
- a generalization of FSM notations, intended to facilitate the definition of large FSMs.
- defined in terms of states and transitions too.
- however, more than one state may be "on" at any one time (for concurrrency)
- moreover, statecharts consist of several FSMs composed in terms of "AND" or "OR"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



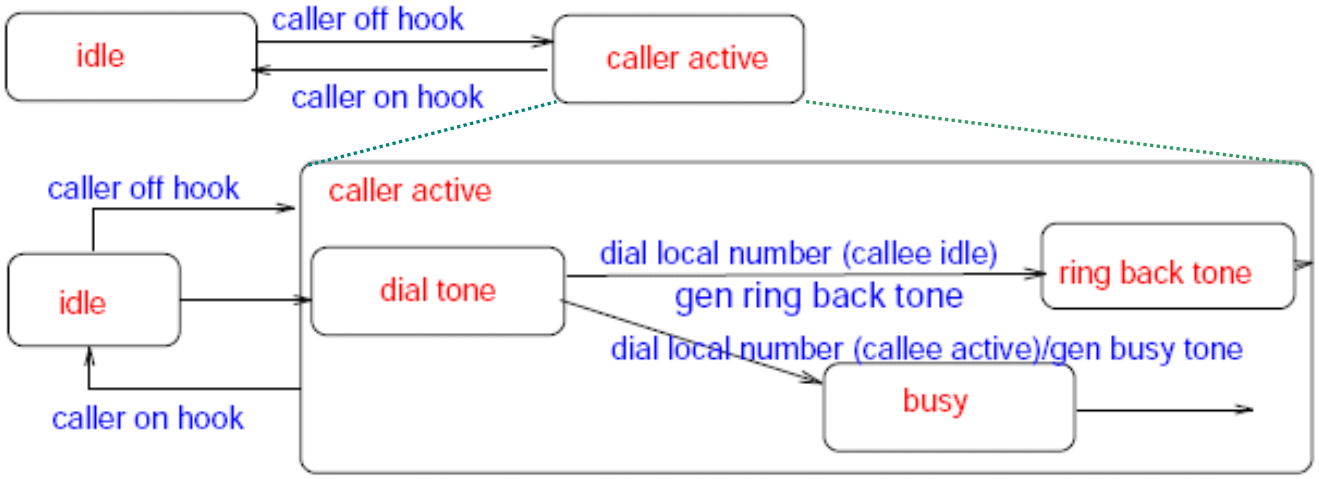


Statecharts

- Superstates: aggregate sets of states with common transitions

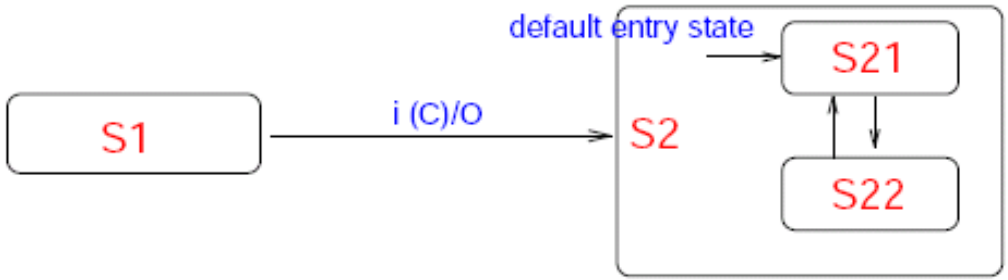


Example:



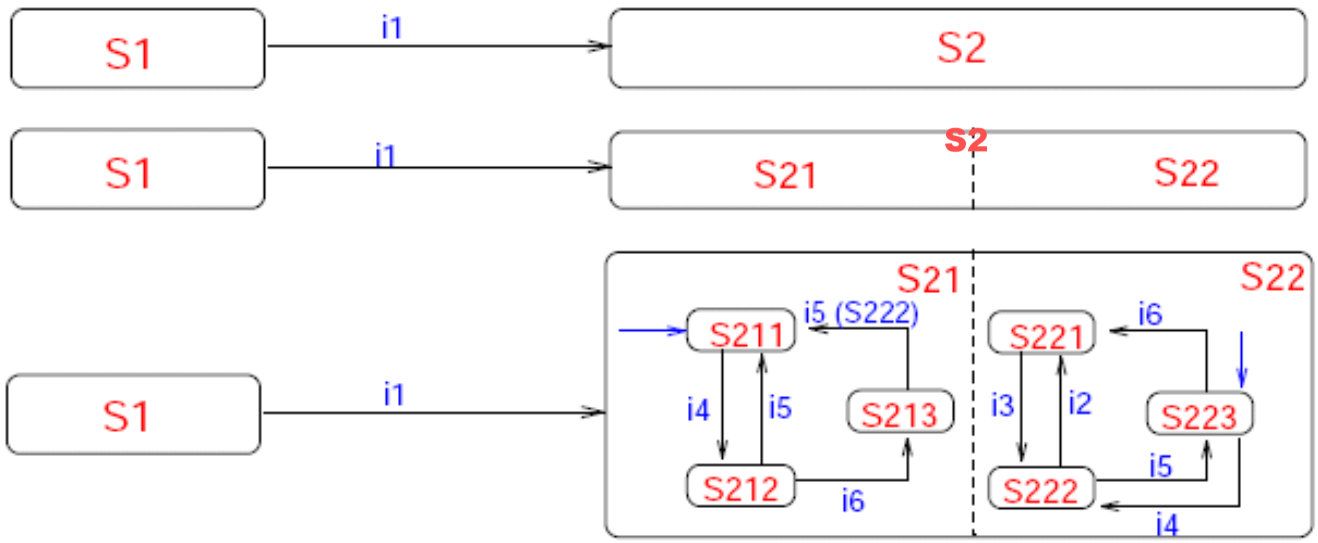
Statecharts

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



Valid configurations?

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

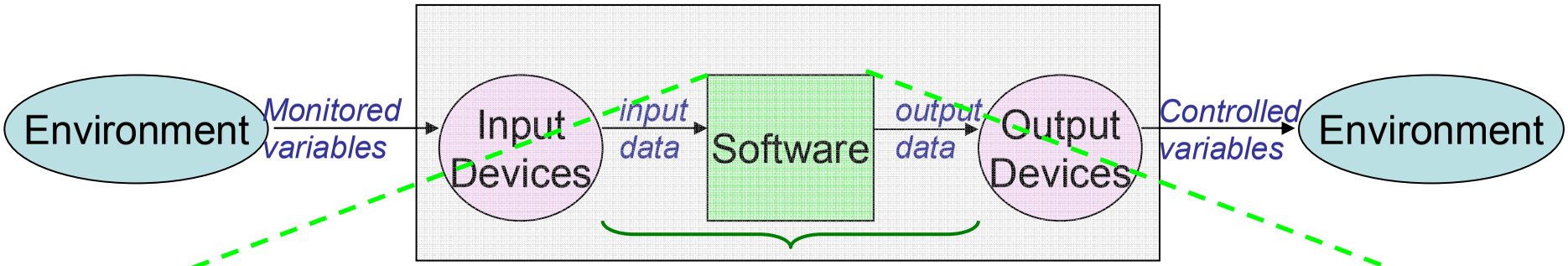


"orthogonal" decomposition: state machines S1 and S2 are independent

SCR: Tabular Specifications

http://chacs.nrl.navy.mil/publications/CHACS/2002/2002heitmeyer-encse.pdf

Recall: Four Variable Model



SCR Specifications

$$\Sigma = (S, S_0, E^m, T)$$

Dictionaries

Monitored/Controlled

Variables

Types

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

Constraints

Mode Transition Tables

Old Mode	Event	New Mode
TooLow	@T(WaterPres $\geq$ Low)	Permitted
Permitted	@T(WaterPres $\geq$ Permit)	High
Permitted	@T(WaterPres $<$ Low)	TooLow
High	@T(WaterPres $<$ Permit)	Permitted

Table 1: Mode Transition Table for Pressure.

Event Tables

Mode Class Pressure	Events	
High	False	@F(Pressure=High)
TooLow, Permitted	@T(Block=On) WHEN Reset=Off	@T(Pressure=High) OR @T(Reset=On)
Overridden	True	False

Table 2: Event Table for Overridden.

Condition Tables

Mode Class Pressure	Conditions	
High, Permitted	True	False
TooLow	Overridden	NOT Overridden
Safety Injection	Off	On

Table 3: Condition Table for Safety Injection.

Assertions, Scenarios, …

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) WHEN d*** means: "c became true when c was false and d was true"

$$@\mathrm{T}(c) \text{ WHEN } d \stackrel{\text{def}}{=} \neg c \wedge c' \wedge d$$

@F(c) WHEN d

SCR: Example - Safety Injection System (SIS)

[adapted from Weimin Ma's drawing]

❑ Goal:

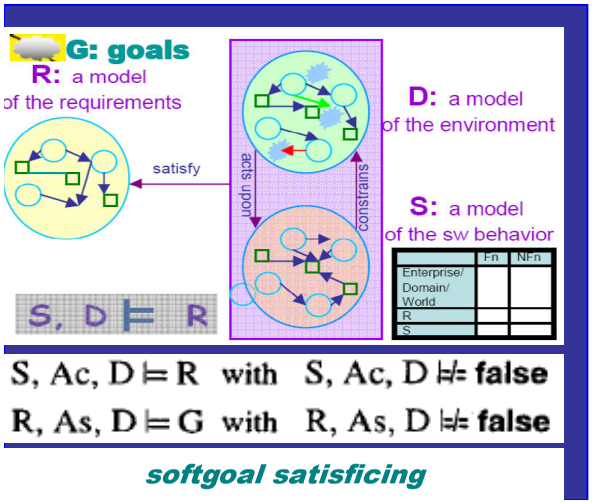
Maintain safety of a nuclear reactor.

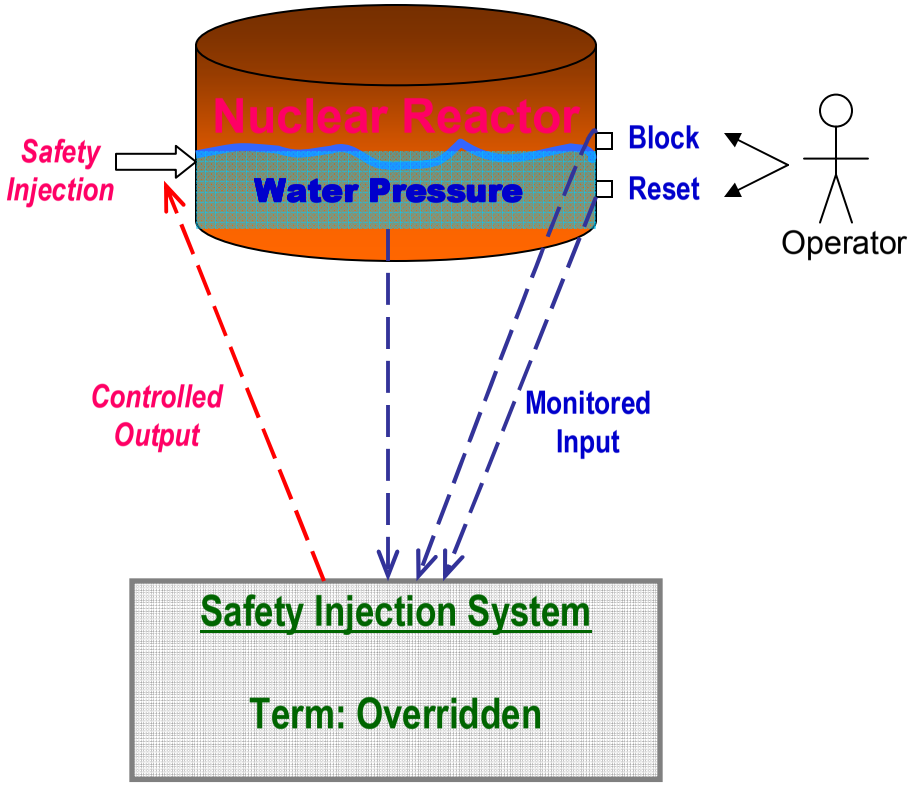
❑ Domain:

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:**

Old Mode	Event	New Mode
TooLow	@T(WaterPres $\geq$ Low)	Permitted
Permitted	@T(WaterPres $\geq$ Permit)	High
Permitted	@T(WaterPres $<$ Low)	TooLow
High	@T(WaterPres $<$ Permit)	Permitted

Table 1: Mode Transition Table for Pressure.

SCR: Controlled Variables

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

❑ 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:

$V = F_e(M_c, E)$

could be null

$$\texttt{Overridden}' = \begin{cases} \textit{true} & \text{if} \quad (\texttt{Pressure} = \texttt{TooLow} \wedge \texttt{Block}' = \texttt{On} \wedge \texttt{Block} = \texttt{Off} \wedge \texttt{Reset} = \texttt{Off}) \vee (\texttt{Pressure} = \texttt{Permitted} \wedge \texttt{Block}' = \texttt{On} \wedge \texttt{Block} = \texttt{Off} \wedge \texttt{Reset} = \texttt{Off}) \\ \textit{false} & \text{if} \quad (\texttt{Pressure} = \texttt{TooLow} \wedge \texttt{Reset}' = \texttt{On} \wedge \texttt{Reset} = \texttt{Off}) \vee (\texttt{Pressure} = \texttt{Permitted} \wedge \texttt{Reset}' = \texttt{On} \wedge \texttt{Reset} = \texttt{Off}) \vee (\texttt{Pressure}' = \texttt{High} \wedge \texttt{Pressure} \neq \texttt{High}) \\ \texttt{Overridden} & \text{otherwise} \end{cases}$$

no event can cause Overridden to become true.

Mode Class Pressure	Events	
High	False	@F(Pressure=High)
TooLow, Permitted	@T(Block=On) WHEN Reset=Off	@T(Pressure=High) OR @T(Reset=On)
Overridden	True	False

Table 2: Event Table for Overridden.

SCR: Controlled Variables

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

❑ 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:

$V = F_c(M_c, C)$

could be null

$$\texttt{SafetyInjection} = \begin{cases} \texttt{Off} & \text{if} \quad \texttt{Pressure} = \texttt{High} \vee \texttt{Pressure} = \texttt{Permitted} \vee \\ & \qquad (\texttt{Pressure} = \texttt{TooLow} \wedge \texttt{Overridden} = \mathit{true}) \\ \texttt{On} & \text{if} \quad \texttt{Pressure} = \texttt{TooLow} \wedge \texttt{Overridden} = \mathit{false}. \end{cases}$$

Mode Class Pressure	Conditions	
High, Permitted	True	False
TooLow	Overridden	NOT Overridden
Safety Injection	Off	On

Table 3: Condition Table for Safety Injection.

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

SCR: Another Example

- **Mode Class Table**

Current Mode	Powered on	Too Cold	Temp OK	Too Hot	New Mode
Off	@T	-	t	-	**Inactive**
	@T	t	-	-	**Heat**
	@T	-	-	t	**AC**
Inactive	@F	-	-	-	**Off**
	-	@T	-	-	**Heat**
	-	-	-	@T	**AC**
Heat	@F	-	-	-	**Off**
	-	-	@T	-	**Inactive**
AC	@F	-	-	-	**Off**
	-	-	@T	-	**Inactive**

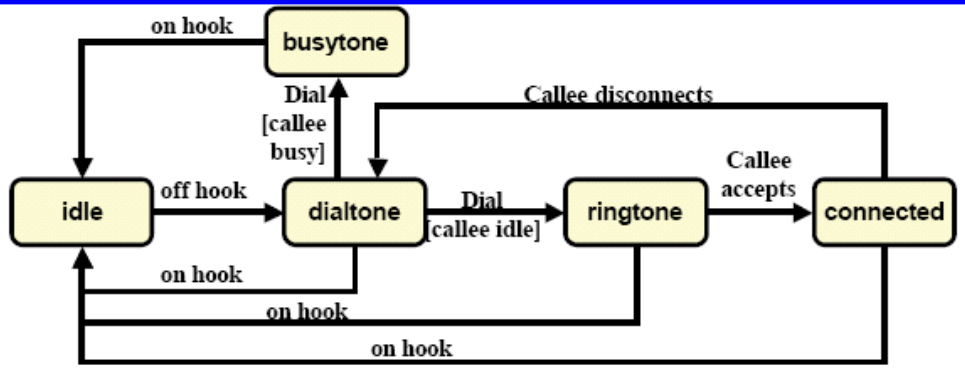
- **Event Table**

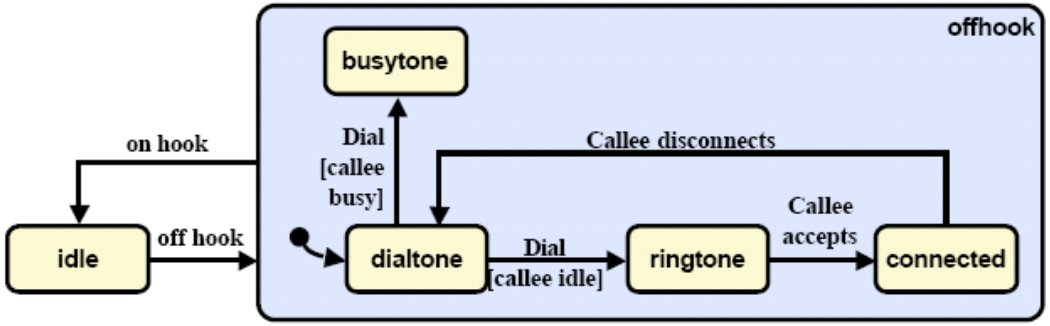
Modes		
Heat, AC	@C(target)	never
Inactive, Off	never	@C(target)
Ack_tone =	**Beep**	**Clang**

- **Condition Table**

Modes		
Heat	target - temp ≤ 5	target - temp >5
AC	temp - target ≤ 5	temp - target >5
Inactive, Off	true	never
Warning light =	**Off**	**On**

EFSMs, Statecharts & SCR





Current Mode	offhook	dial	callee offhook	New mode
idle	@T	-	-	dialtone
dialtone	-	@T	F	ringtone
	-	@T	T	busytone
	@F	-	-	idle
busytone	@F	-	-	idle
ringtone	-	-	@T	connected
	@F	-	-	idle
connected	-	-	@F	dialtone
	@F	-	-	idle

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

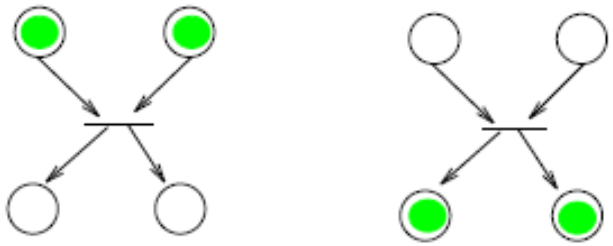
transition

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 **AFTER**

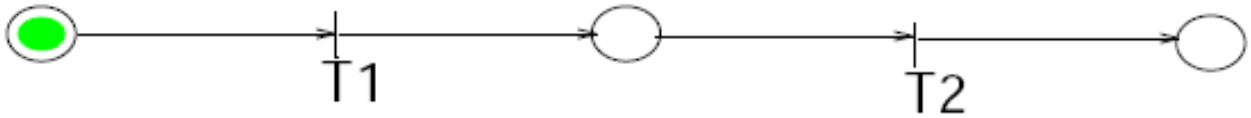


The particular Petri Net notation here makes a single token assumption and avoids no inhibitor arcs [LK]

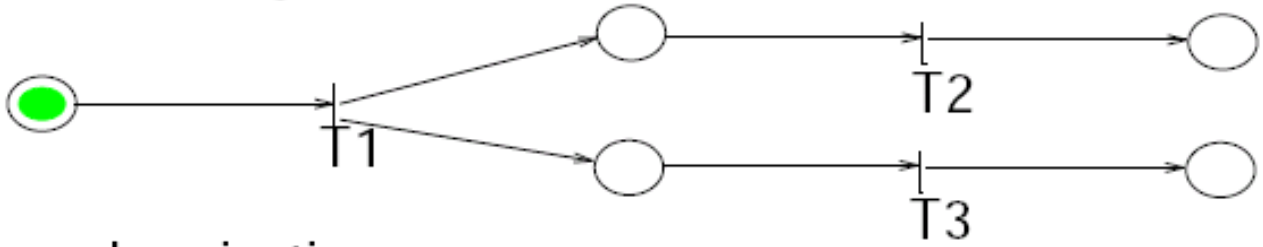
Petri Nets

Fundamental notions:

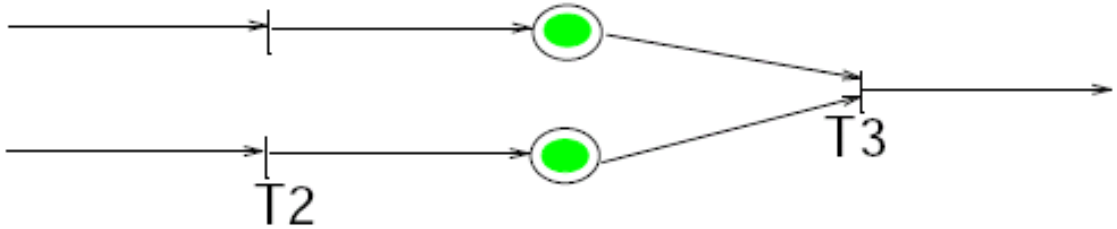
- sequencing



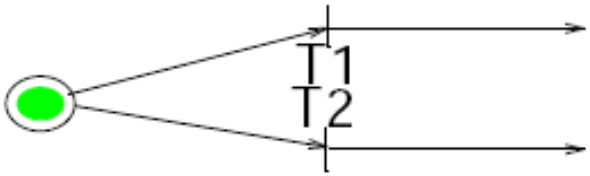
- concurrency



- synchronization



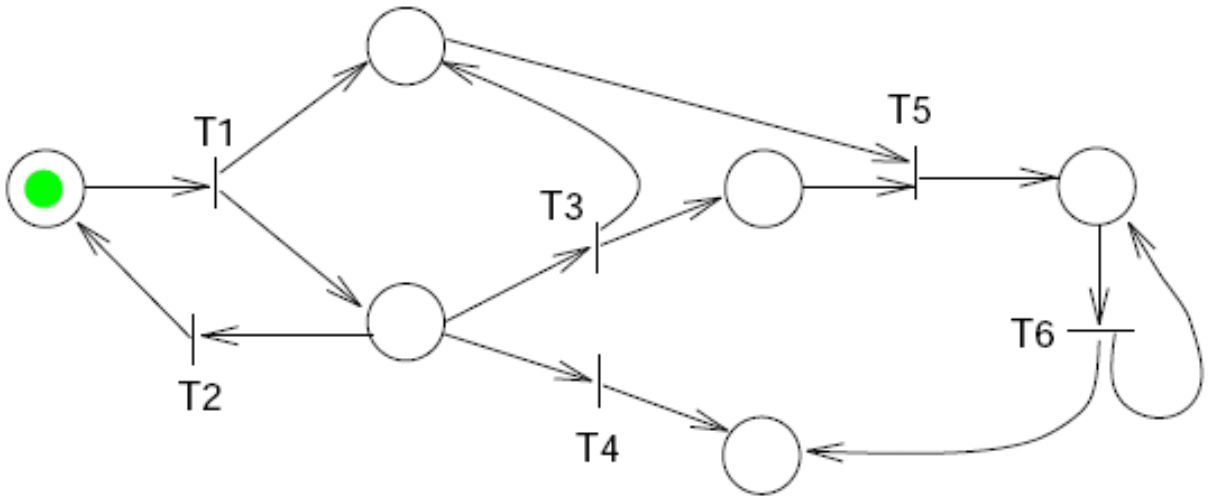
- decision making



Petri Nets

Example

Stimuli occure in the following order: T1, T3, T5, T4, T6



Augmented Petri Net

- activation conditions and actions

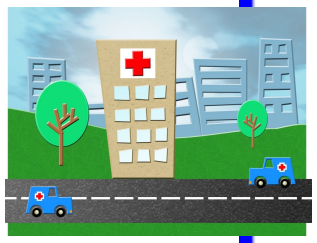
  e.g., for a patient waiting to see adoctor, an activation condition may be
  "doctor ready" and
  an associated action may be "patient move to doctor's office"

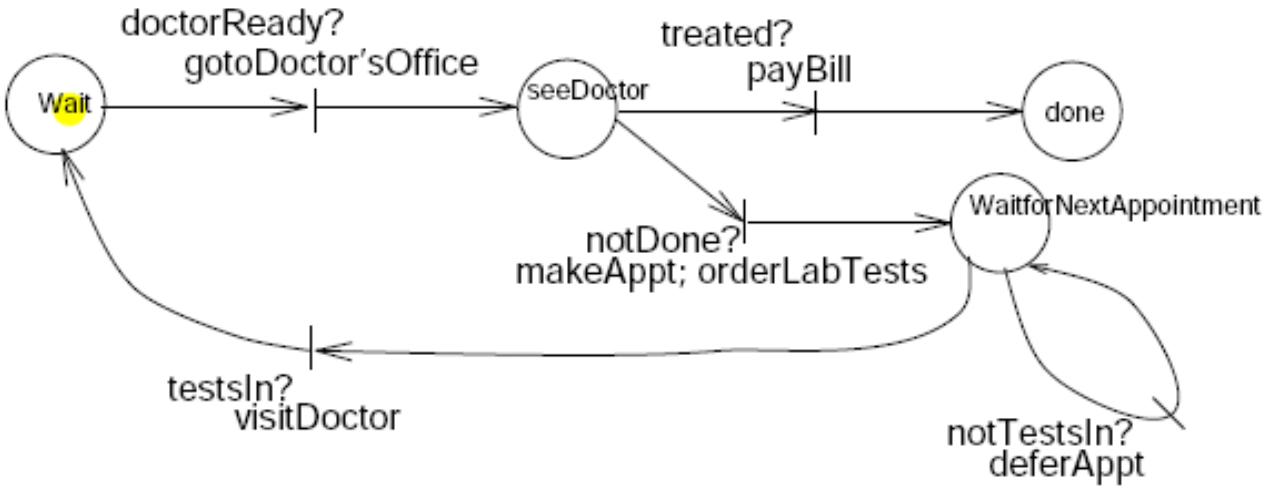
- invariants on states:

  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





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."

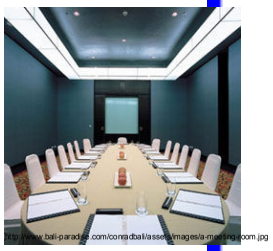
Finally, Grady Booch answered:

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

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)
```