Chapter 14

System Testing

System Testing

- “Intuitively clear”
  - customer expectations
  - close to customer acceptance testing
- BUT we need a better basis for really understanding system testing
- Threads—the subject of system testing
- How are they identified?
  - *ad hoc?*
  - from experience?
  - from a possibly incomplete requirements specification?
  - from an executable model? (Model-Based Testing)

Threads...

- (not nice clothes...)
- An execution time concept
- Per the definition, a *thread* can be understood as a sequence of atomic system functions.
- When a system test case executes
  - a thread occurs, and
  - can be observed at the port boundary of the system
- The BIG Question: where do we find (or how do we identify) threads?
- Our approach—Model-Based Testing

Threads—Several Views

- A scenario of normal usage
- A use case
- A stimulus/response pair
- Behavior that results from a sequence of system-level inputs
- An interleaved sequence of port input and output events
- A sequence of transitions in a state machine description of the system
- An interleaved sequence of object messages and method executions
- A sequence of machine instructions
- A sequence of source instructions
- A sequence of MM-paths
- A sequence of atomic system functions (to be defined)

Some Choices—Threads in an ATM System

- Entry of a digit
- Entry of a personal identification number (PIN)
- A simple transaction: ATM Card Entry, PIN Entry, select transaction type (deposit, withdraw), present account details (checking or savings, amount), conduct the operation, and report the results
- An ATM session containing two or more simple transactions

- Each of these can be understood as an interleaved sequence of port level inputs and outputs.

Details of PIN Entry as a Thread

- A screen requesting PIN digits.
- An interleaved sequence of digit keystrokes and screen responses.
- The possibility of cancellation by the customer before the full PIN is entered.
- A system disposition:
  - A customer has three chances to enter the correct PIN.
  - Once a correct PIN has been entered, the user sees a screen requesting the transaction type.
  - After three failed PIN En try attempts, a screen advises the customer that the ATM card will not be returned, and no access to ATM functions is provided.

Definition: *Atomic System Function*

- Definition: An *Atomic System Function (ASF)* is an action that is observable at the system level in terms of port input and output events.
- About ASFs
  - characterized by a sequence of port level inputs and outputs
  - could be just a simple stimulus/response pair (*e.g.* digit entry)
- Sample ASFs in our ATM example
  - card entry
  - PIN entry
  - Transaction selection
  - session termination

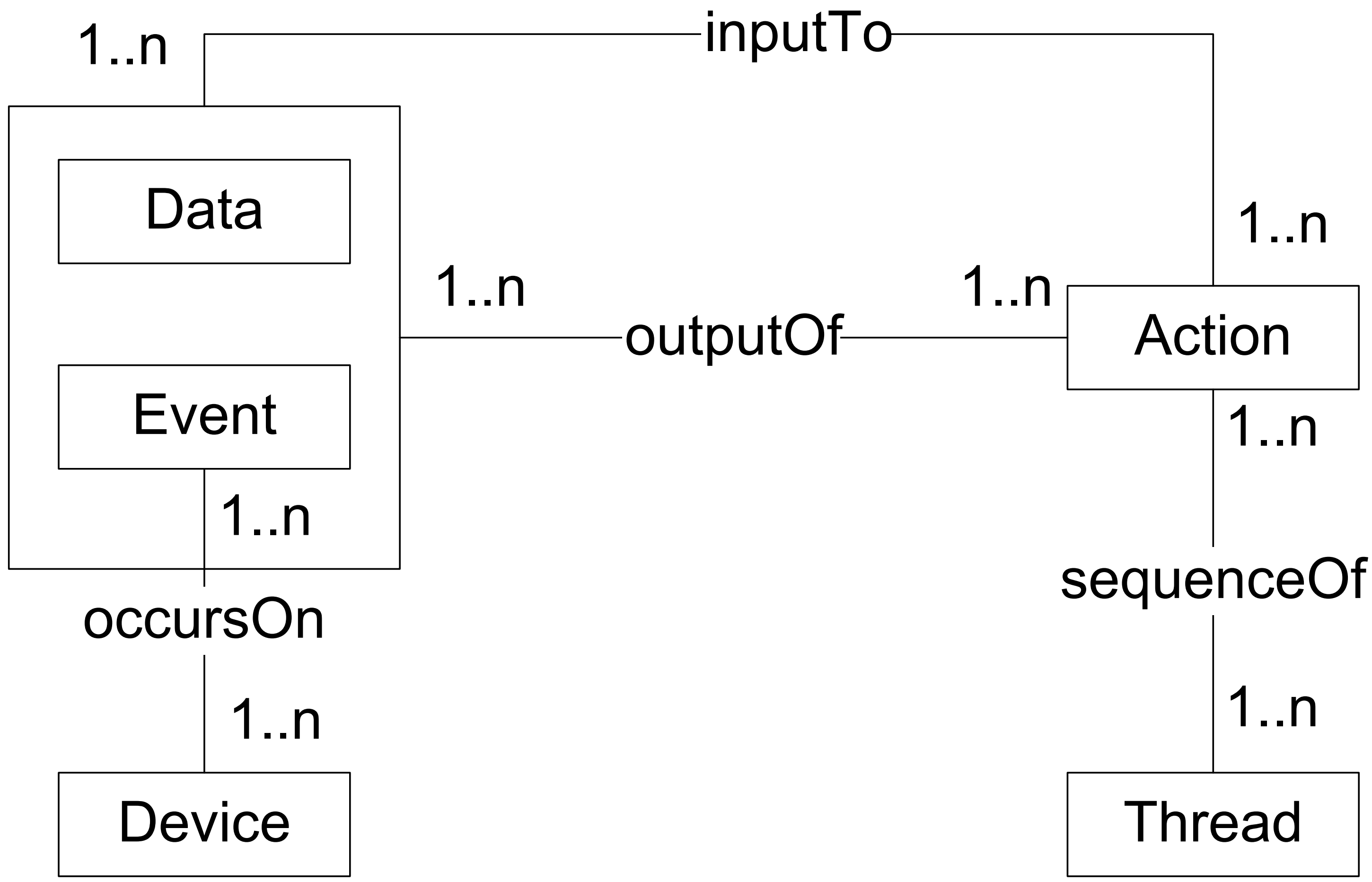
More Definitions…

- Given a system defined in terms of atomic system functions, the *ASF Graph* of the system is the directed graph in which nodes are ASFs and edges represent sequential flow.
- A *source ASF* is an Atomic System Function that appears as a source node in the ASF graph of a system.
- A *sink ASF* is an Atomic System Function that appears as a sink node in the ASF graph.
- A *system thread* is a path from a source ASF to a sink ASF in the ASF graph of a system.

Basis Concepts for Requirements Specification

- All of requirements specification models are developed on these basis concepts.
- Data
  - Inputs to actions
  - Outputs of actions
- Events
  - Inputs to actions
  - Outputs of actions
- Actions
- Threads (sequences of actions)
- Devices

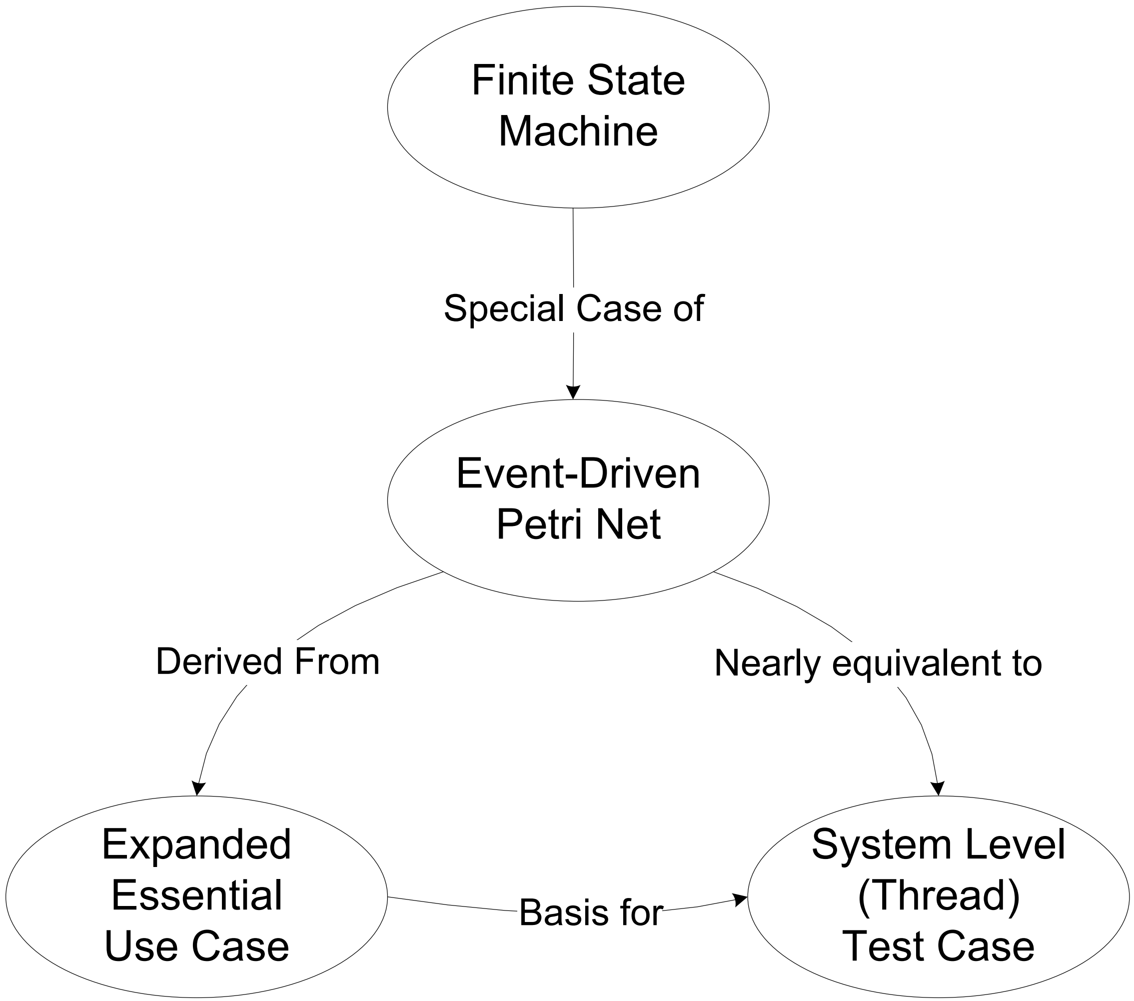
E/R Model of Basis Concepts



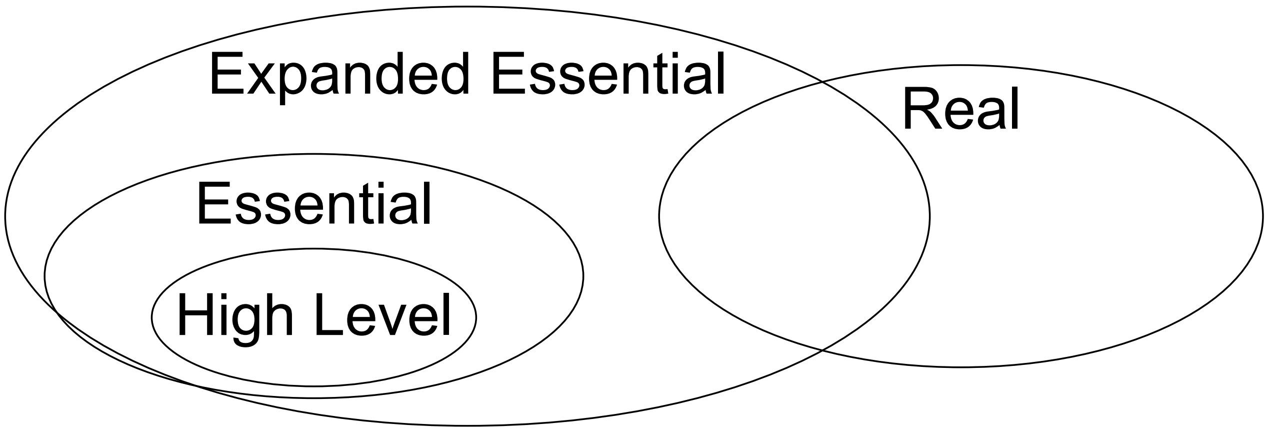
Sources of Threads

- An "Expanded Essential Use Case"
  - (per Craig Larman)
  - pre-conditions
  - interleaved sequence of input and output events
  - post-conditions
- A path in an executable model
  - finite state machine
  - Event-Driven Petri Net
- Continuing example: the Simple ATM System (SATM)

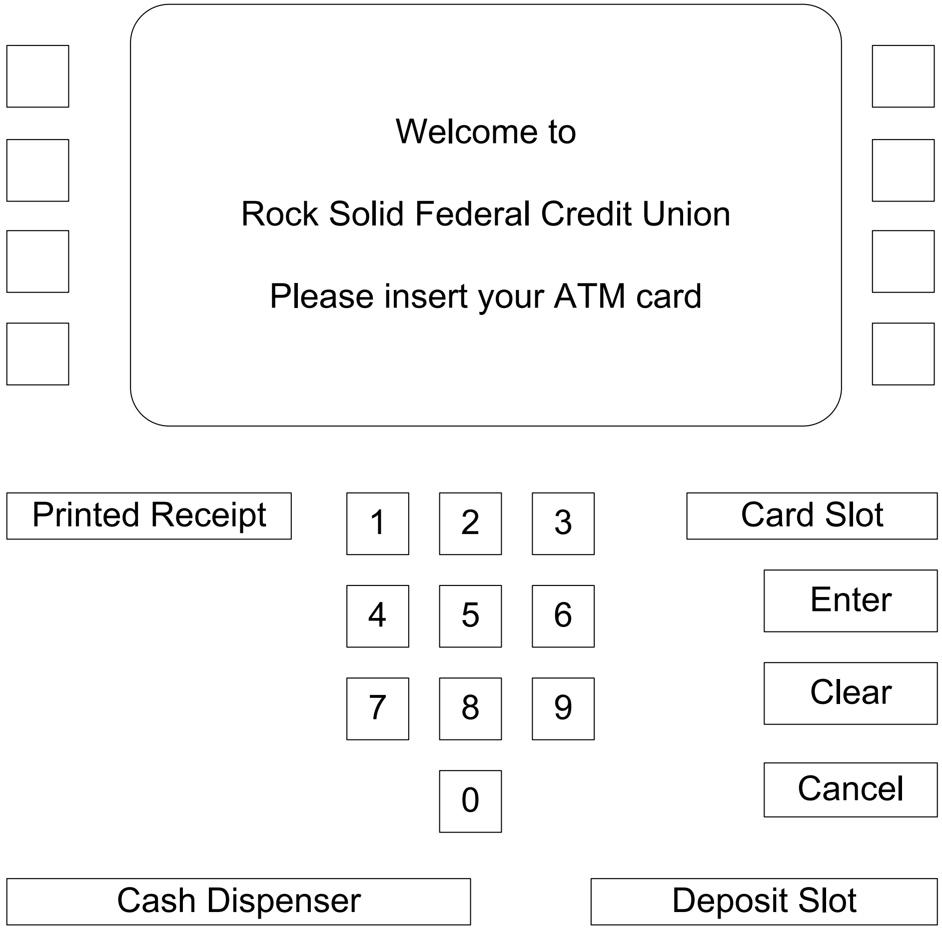
Sources of Threads—Model-Based Testing



Information Content of Larman's Use Cases



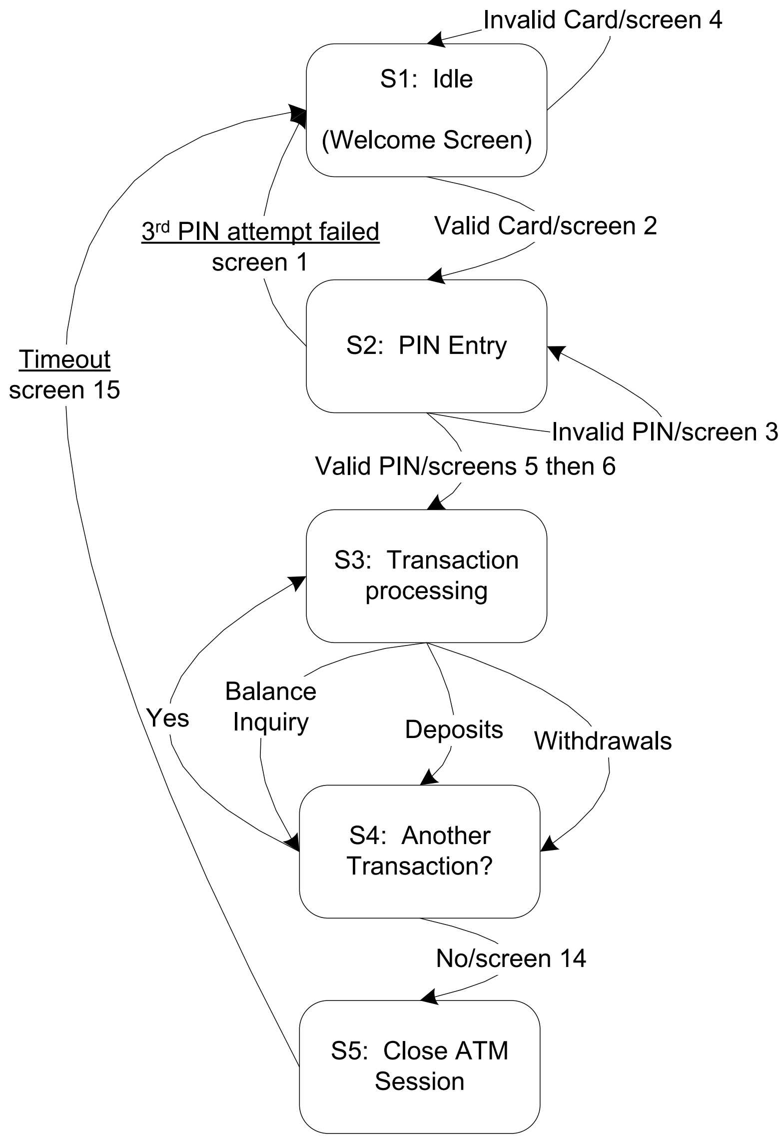
SATM System User Interface



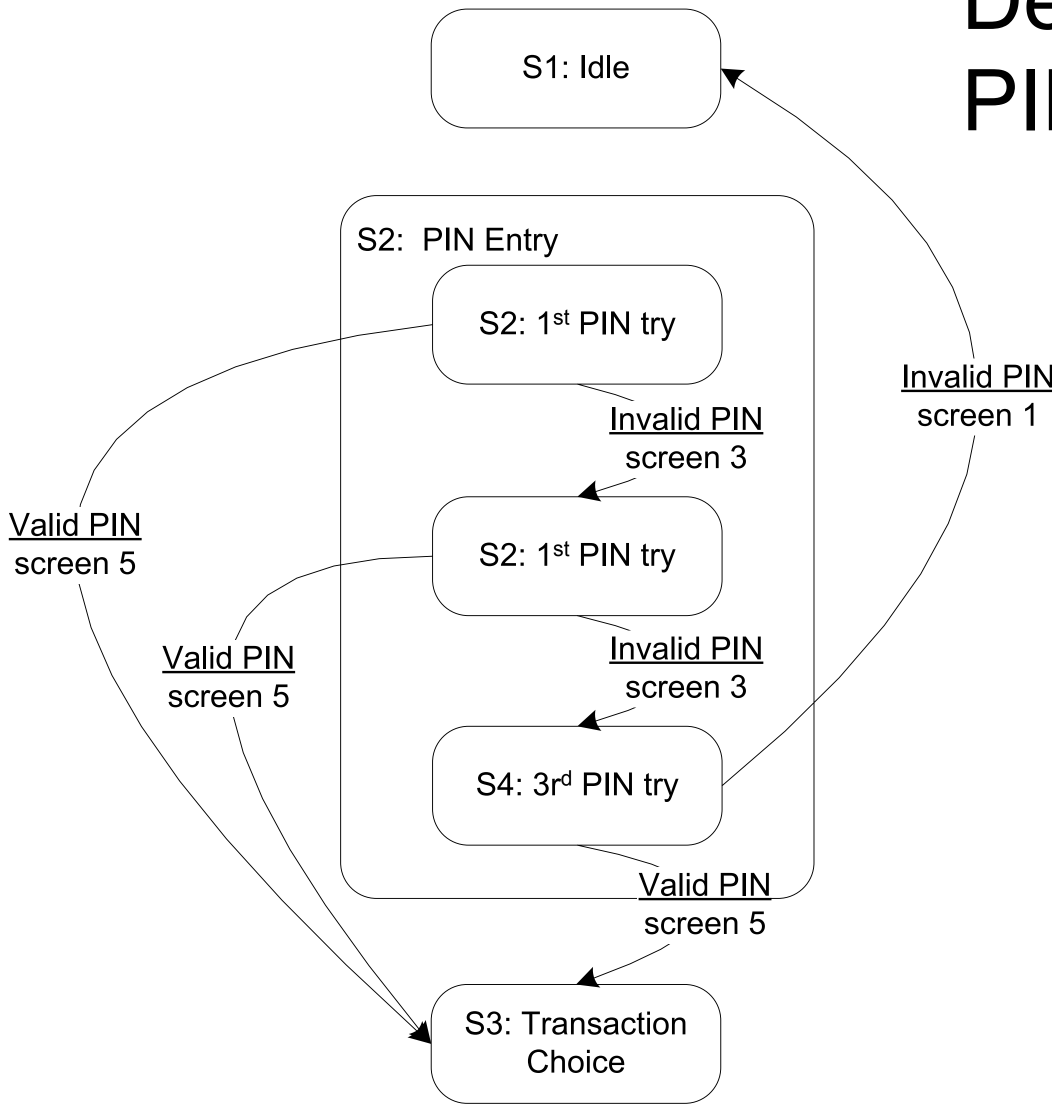
SATM System Screens

Screen 1 Welcome Please insert your ATM card	Screen 2 Please enter your PIN _ _ _ _	Screen 3 Your PIN is incorrect. Please try again.
Screen 4 Invalid ATM card. It will be retained.	Screen 5 Select transaction: balance > deposit > withdrawal >	Screen 6 Balance is $dddd.dd
Screen 7 Enter amount. Withdrawals must be multiples of $10	Screen 8 Insufficient Funds! Please enter a new amount	Screen 9 Machine can only dispense $10 notes
Screen 10 Temporarily unable to process withdrawals. Another transaction?	Screen 11 Your balance is being updated. Please take cash from dispenser.	Screen 12 Temporarily unable to process deposits. Another transaction?
Screen 13 Please insert deposit into deposit slot.	Screen 14 Your new balance is being printed. Another transaction?	Screen 15 Please take your receipt and ATM card. Thank you.

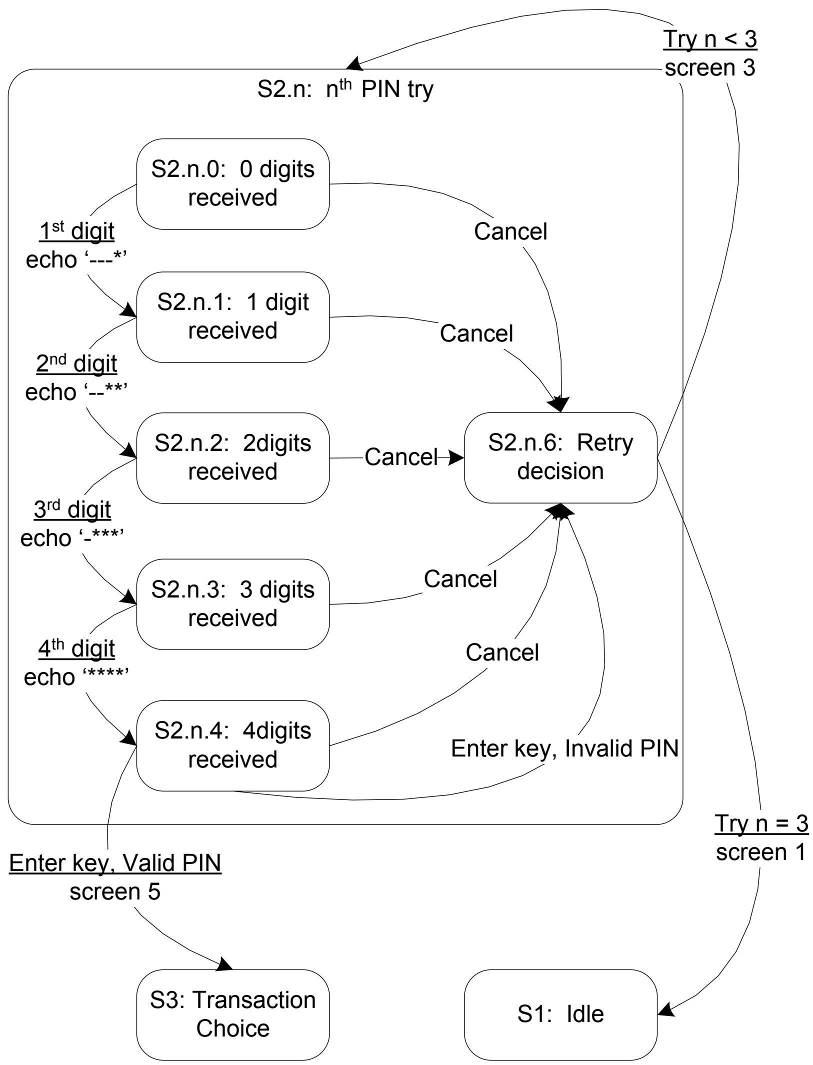
Uppermost Level of the SATM Finite State Machine



Details of SATM PIN Entry State



Details of SATM PIN Try State



Paths in the SATM PIN Try State

- Correct PIN on first try state sequence
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.3, S2.n.4, S3>
- Port Event Sequence
  - 1st digit, echo "- - - *"
  - 2nd digit, echo "- - * *"
  - 3rd digit, echo "- * * *"
  - 4th digit, echo "* * * *"
  - Enter
- Failed PIN on first try state Sequences
  - <S2.n.0, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.3, S2.n.6>
  - <S2.n.0, S2.n.1, S2.n.2, S2.n.3, S2.n.4, S2.n.6>

How Many Paths in the PIN Try State?

- 1st try: 1 correct + 5 failed attempts
- 2nd try: 5 failed 1st attempts * 6 2nd attempts
- 3rd try: 25 failed 1st and 2nd attempts * six 3rd attempts
- Do we really want to test all of these?
- This foreshadows the question of “long” versus “short” use cases.

Port Event Sequence: Correct PIN on 1st Try

Port Input Event	*Port Output Event*
	Screen 2 displayed with '- - - -'
1st digit	
	Screen 2 displayed with '- - - *'
2nd digit	
	Screen 2 displayed with '- - * *'
3rd digit	
	Screen 2 displayed with '- * * *'
4th digit	
	Screen 2 displayed with '* * * *'
(valid PIN)	Screen 5 displayed

Use Case: Correct PIN on 1st Try

Use Case Name	Correct PIN entry on first try
Use Case ID	EEUC-1
Description	A customer enters the PIN number correctly on the first attempt.
Pre-Conditions	1. The expected PIN is known
	2. Screen 2 is displayed
Event Sequence	
Input events	Output events
	1. Screen 2 shows '- - - - '
2. Customer touches 1st digit	3. Screen 2 shows '- - - * '
4. Customer touches 2nd digit	5. Screen 2 shows '- - * * '
6. Customer touches 3rd digit	7. Screen 2 shows '- * * * '
8. Customer touches 4th digit	9. Screen 2 shows '* * * * '
10. Customer touches Enter	11. Screen 5 is displayed
Post conditions	Select Transaction screen is active

Test Case: Correct PIN on 1st Try

Test Case Name	Correct PIN entry on first try
Use Case ID	RealUC-1
Description	A customer enters the PIN number correctly on the first attempt.
Pre-Conditions	1. The expected PIN is ‘2468’
	2. Screen 2 is displayed
Event Sequence	
Input events	Output events
	1. Screen 2 shows '- - - - '
2. Customer touches digit 2	3. Screen 2 shows '- - - * '
4. Customer touches digit 4	5. Screen 2 shows '- - * * '
6. Customer touches digit 6	7. Screen 2 shows '- * * * '
8. Customer touches digit 8	9. Screen 2 shows '* * * * '
10. Customer touches Enter	11. Screen 5 is displayed
Post conditions	Select Transaction screen is active

System Test Case: Correct PIN on 1st Try

Test Case Name, ID	Correct PIN entry on first try, TC-1
Description	A customer enters the PIN number correctly on the first attempt.
Pre-Conditions	1. The expected PIN is '2468'
	2. Screen 2 is displayed
Event Sequence	
Input events	Output events
	1. Screen 2 shows '- - - - '
2. Customer touches digit 2	3. Screen 2 shows '- - - * '
4. Customer touches digit 4	5. Screen 2 shows '- - * * '
6. Customer touches digit 6	7. Screen 2 shows '- * * * '
8. Customer touches digit 8	9. Screen 2 shows '* * * * '
10. Customer touches Enter	11. Screen 5 is displayed
Post conditions	Select Transaction screen is active
Test Result, Run by	Pass, Paul Jorgensen

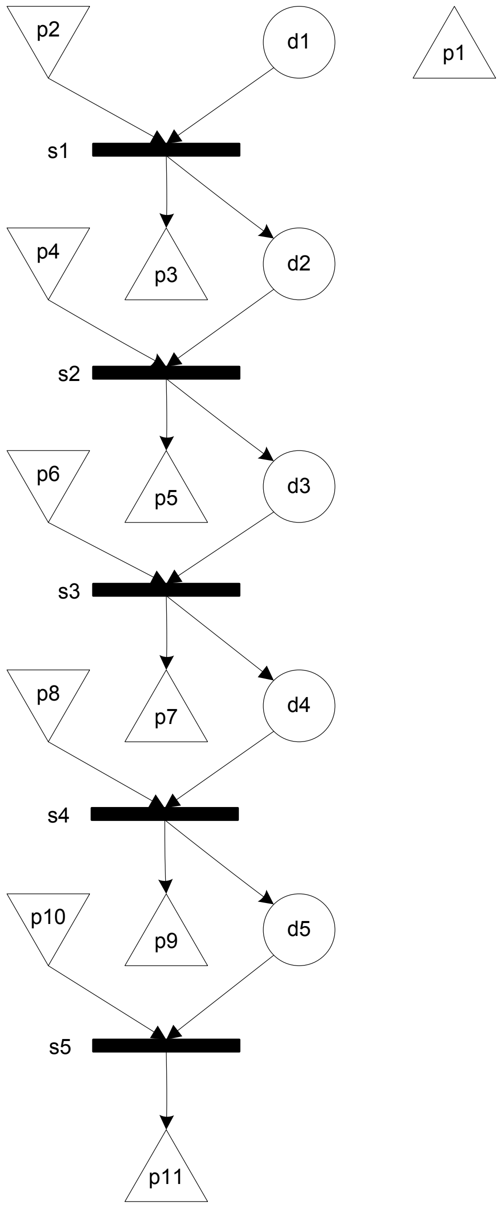
Event-Driven Petri Net of Correct PIN on First Try

Port Input events
p2: 1st digit
p4: 2nd digit
p6: 3rd digit
p8: 4th digit
p10: Enter

Port Output Events
p1: screen 2 '- - - -'
p3: screen 2 '- - - *'
p5: screen 2 '- - * *'
p7: screen 2 '- * * *'
p9: screen 2 '* * * *'
p11: screen 5

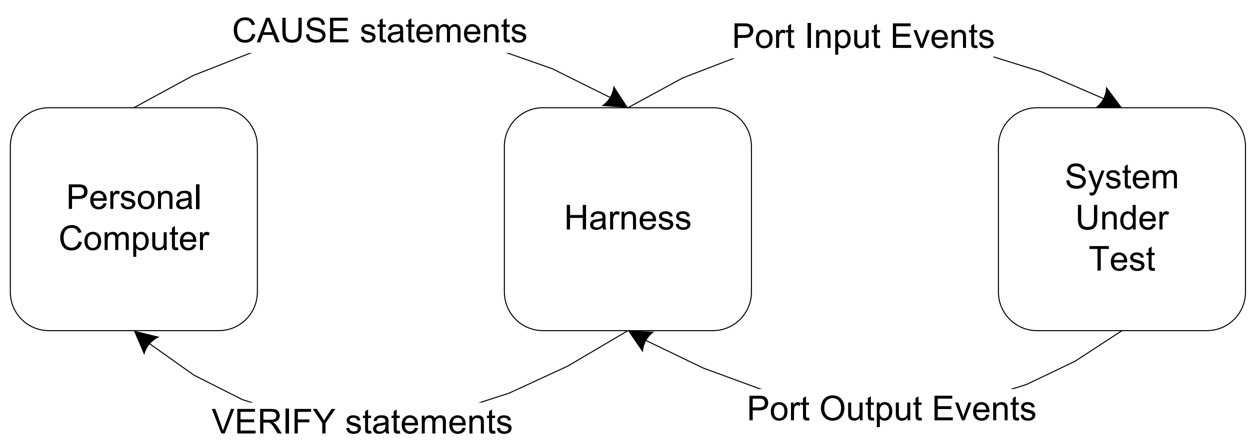
Data Places
d1: expecting digit 1
d2: expecting digit 2
d3: expecting digit 3
d4: expecting digit 4
d5: entered PIN

Transitions
s1: (not named)
s2: (not named)
s3: (not named)
s4: (not named)
s5: (not named)



Automated Test Case Execution

- Basic architecture Use cases interpretively executed
  - CAUSE inputs
  - VERIFY outputs
  - Check observed versus expected outputs
- via a harness
- connected to the System Under Test



Long versus Short Use Cases

- A “Long” use case is typically an end-to-end transaction.
- SATM example: A full traversal of the high level finite state machine, from the Welcome screen to the End Session screen: <s1, s2, s3, s4, s5>
- A “Short” use case is at the level on an atomic system function.
- Examples
  - PIN Entry
  - Transaction selection
  - Session closing

Short Use Cases

- A "Short" use case is at the level on an atomic system function (ASF).
- In the directed graph of ASFs,
  - nodes are ASFs
  - edges signify possible sequential execution of ASFs
- Consider an ASF as a "Short" use case, with
  - pre-conditions
  - post-conditions
- Short use case (ASF) B can follow short use case (ASF) A if the pre-conditions of B are consistent with the post-conditions of A, that is...
- Short use cases "connect" at their pre- and post condition boundaries.

Short Use Cases for the SATM System

Short Use Case	Description
SUC1	Valid ATM card swipe
SUC2	Invalid ATM card swipe
SUC3	Correct PIN attempt
SUC4	Failed PIN attempt
SUC5	Choose Balance
SUC6	Choose Deposit
SUC7	Choose Withdrawal: valid withdrawal amount
SUC8	Choose Withdrawal: amount not a multiple of $20
SUC9	Choose Withdrawal: amount greater than account balance
SUC10	Choose Withdrawal: amount greater than daily limit
SUC11	Chose no other transaction
SUC12	Chose another transaction

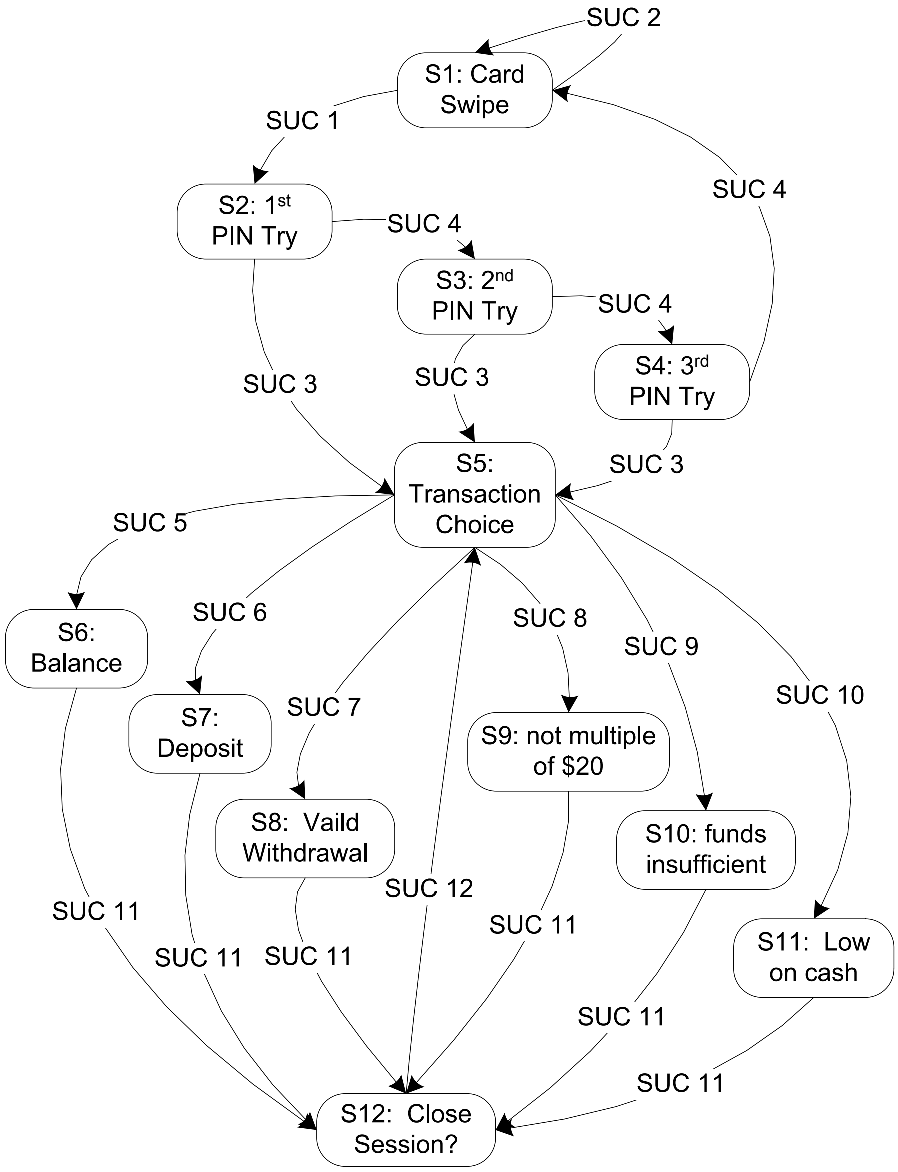
Short Use Cases for Failed PIN Attempts

Short Use Case	Description
SUC13	Digit 1 entered
SUC14	Digit 2 entered
SUC15	Digit 3 entered
SUC16	Digit 4 entered
SUC17	Enter with valid PIN
SUC18	Cancel before digit 1
SUC19	Cancel after digit 1
SUC20	Cancel after digit 2
SUC21	Cancel after digit 3
SUC22	Cancel after digit 4
SUC23	Enter with invalid PIN
SUC24	Next PIN try
SUC25	Last PIN try

How Many Use Cases?

- 1909 “long” use cases
- 25 “short” use cases
- Ways to determine “how many?
  - Incidence with input events (cover every input event)
  - Incidence with output events (cover every output event)
  - Incidence with classes (need a use case/class incidence matrix)
- These lead directly to system testing coverage metrics.

Short Use Cases for the SATM System



System Testing with Short Use Cases

- Basic idea: a short use case is an atomic system function (ASF)
- ASFs ...
  - begin with a port input event
  - end is one of possibly several port output events
- ASFs can be identified
  - in source code
  - in executable models
  - from short use cases
- Example: the integration version of NextDate
  - (see text) pseudo-code on next 7 slides

Integration Version: NextDate Pseudo-Code

```
1 Main integrationNextDate    'start program event occurs here
  Type  Date
     Month As Integer
     Day As Integer
     Year As Integer
  EndType
  Dim today As Date
  Dim tomorrow As Date
2   Output("Welcome to NextDate!")
3   GetDate(today)
4   PrintDate(today)
5   tomorrow = IncrementDate(today)
6   PrintDate(tomorrow)
7 End Main
```

NextDate Pseudo-Code (continued)

```
Function isLeap(year)  Boolean
  If (year divisible by 4)
     Then
       If (year is NOT divisible by 100)
         Then isLeap = True
         Else
           If (year is divisible by 400)
             Then isLeap = True
             Else isLeap = False
           EndIf
       EndIf
  Else isLeap = False
  EndIf
  End (Function isLeap)
```

NextDate Pseudo-Code (continued)

```
Function lastDayOfMonth(month, year)  Integer
  Case month Of
    Case 1: 1, 3, 5, 7, 8, 10, 12
      lastDayOfMonth = 31
    Case 2: 4, 6, 9, 11
      lastDayOfMonth = 30
    Case 3: 2
      If (isLeap(year))
        Then lastDayOfMonth = 29
        Else lastDayOfMonth = 28
      EndIf
  EndCase
End (Function lastDayOfMonth)
```

NextDate Pseudo-Code (continued)

```
35  Function GetDate(aDate)  Date
      dim aDate As Date
36    Function ValidDate(aDate)  Boolean  'within scope of GetDate
64    End (Function ValidDate)
    ' GetDate body begins here
65    Do
66      Output("enter a month")
67      Input(aDate.Month)
68      Output("enter a day")
69      Input(aDate.Day)
70      Output("enter a year")
71      Input(aDate.Year)
72      GetDate.Month = aDate.Month
73      GetDate.Day = aDate.Day
74      GetDate.Year = aDate.Year
75    Until (ValidDate(aDate))
76    End (Function GetDate)
```

ValidDate Pseudo-Code

```
Function ValidDate(aDate)  Boolean  'within scope of GetDate
dim aDate As Date
dim dayOK, monthOK, yearOK As Boolean
  If ((aDate.Month > 0) AND (aDate.Month <=12)
    Then   monthOK = True
        Output("Month OK")
    Else   monthOK = False
        Output("Month out of range")
  EndIf
  If (monthOK)
    Then
    If ((aDate.Day > 0) AND (aDate.Day <=
  lastDayOfMonth(aDate.Month, aDate.Year))
        Then   dayOK = True
            Output("Day OK")
        Else   dayOK = False
            Output("Day out of range")
      EndIf
    EndIf
```

ValidDate Pseudo-Code (continued)

```
  If ((aDate.Year > 1811) AND (aDate.Year <= 2012)
    Then   yearOK = True
        Output("Year OK")
    Else   yearOK = False
        Output("Year out of range")
  EndIf
  If (monthOK AND dayOK AND yearOK)
    Then ValidDate = True
        Output("Date OK")
    Else ValidDate = False
      Output("Please enter a valid date")
  EndIf
End (Function ValidDate)
```

NextDate Pseudo-Code (continued)

```
Function IncrementDate(aDate)  Date
  If (aDate.Day < lastDayOfMonth(aDate.Month))
    Then aDate.Day = aDate.Day + 1
    Else aDate.Day = 1
      If (aDate.Month = 12)
        Then  aDate.Month = 1
            aDate.Year = aDate.Year + 1
        Else  aDate.Month = aDate.Month + 1
      EndIf
  EndIf
End (IncrementDate)
Procedure PrintDate(aDate)
  Output( "Day is ", aDate.Month, "/", aDate.Day, "/", aDate.Year)
End (PrintDate)
```

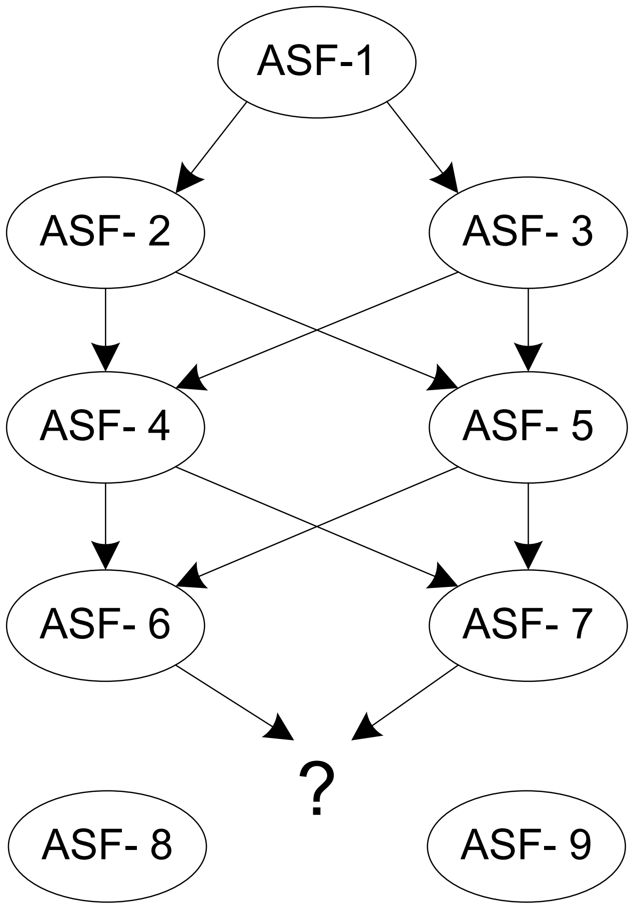
NextDate Input and Output Events

Input Events	Node	Output Event description	Node
e0: start program event	1	e7: Welcome message	2
e1: center a valid month	67	e8: print today's date	4
e2: enter an invalid month	67	e9: print tomorrow's date	6
e3: enter a valid day	69	e10: "month OK"	39
e4: enter an invalid day	69	e11: "month out of range"	41
e5: enter a valid year	71	e12: "day OK"	47
e6: enter an invalid year	71	e13: "day out of range"	49
		e14: "year OK"	54
		e15: "year out of range"	56
		e16: "Date OK"	60
		e17: "please enter a valid date"	62
		e18: "enter a month"	66
		e19: "enter a day"	68
		e20: "enter a year"	70
		e21: "Day is month, day, year"	89

NextDate ASFs (first attempt)

Atomic System Function	Inputs	Outputs
ASF-1 start program	e0	e7
ASF-2 enter a valid month	e1	e10
ASF-3 enter an invalid month	e2	e11
ASF-4 enter a valid day	e3	e12
ASF-5 enter an invalid day	e4	e13
ASF-6 enter a valid year	e5	e14
ASF-7 enter an invalid year	e6	e15
ASF-8 print for valid input		
ASF-9 print for invalid input		

NextDate ASF Graph
(transitions to ASF-8 and ASF-10 require memory)

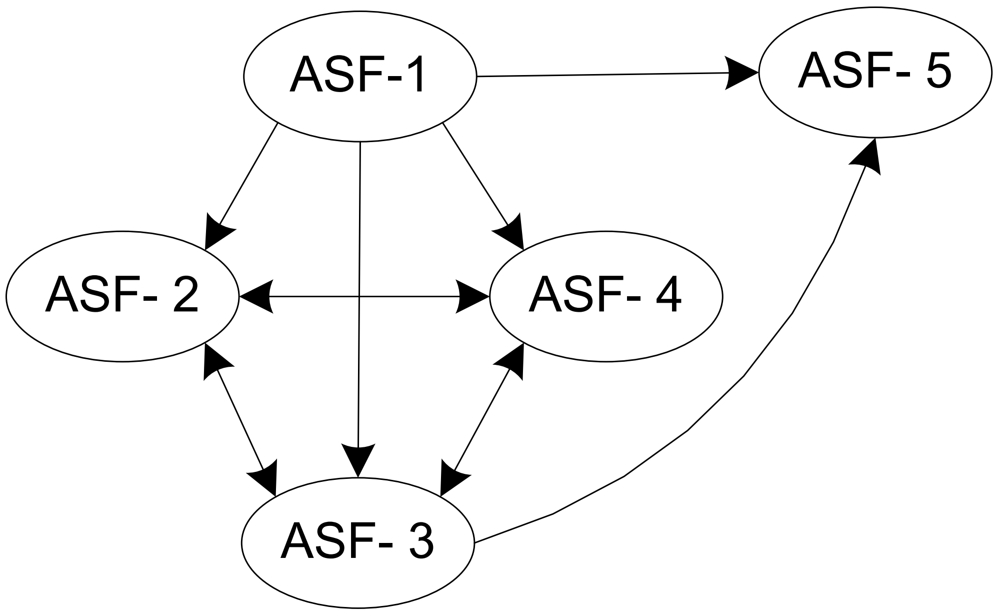


NextDate ASFs (second attempt)

Atomic System Function	Inputs	Outputs
ASF-1 start program	e0	e7
ASF-2 enter a date with an invalid month, valid day and valid year	e2, e3, e5	e11, e12, e14, e17
ASF-3 enter a date with an invalid day, valid month and valid year	e1, e4, e5	e10, e13, e14, e17
ASF-4 enter a date with an invalid year, valid day and valid month	e1, e3, e6	e10, e12, e15, e17
ASF-5 enter a date with valid month, day, and year	e1, e3, e5	e10, e12, e14, e16, e21
ASF-6 enter a date with valid month, day and year invalid	e1, e4, e6	e10, e13, e15, e17
ASF-7 enter a date with valid day, month and year invalid	e2, e3, e6	e11, e12, e15, e17
ASF-8 enter a date with valid year, day and month invalid	e5, e4, 46	e14, e13, e15, e17
ASF-9 enter a date with invalid month, day, year	e2, e4, e6	e11, e13, e15, e17

Improved NextDate ASF Graph

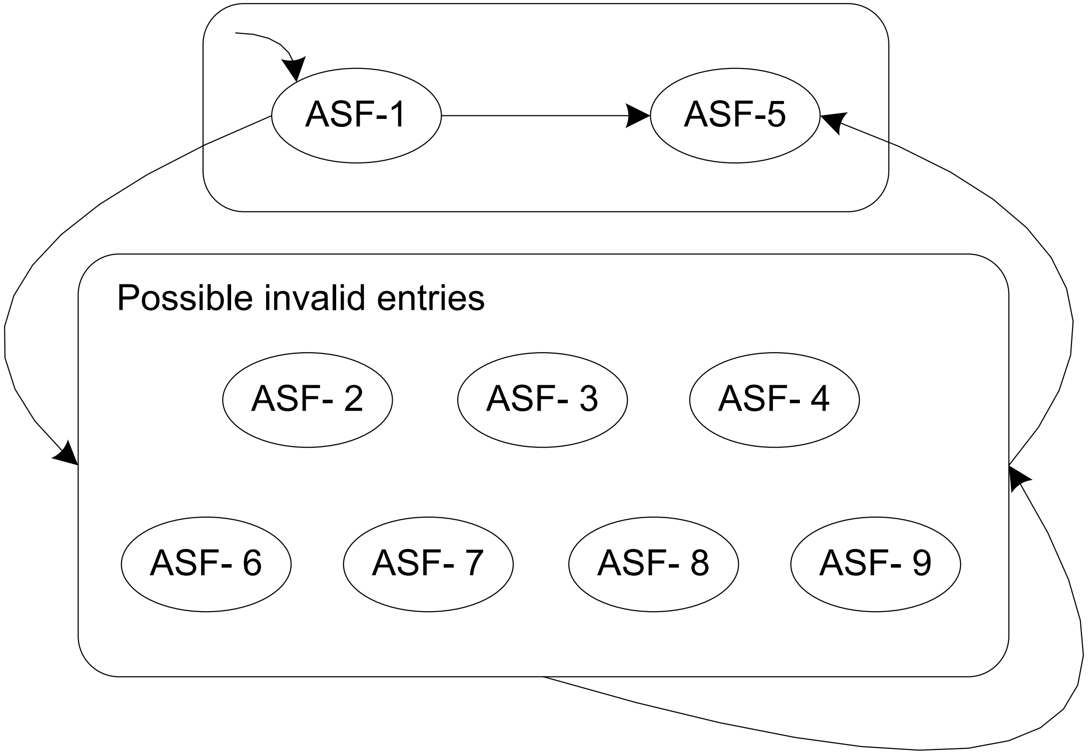
(Very incomplete: can transit from most ASFs to most other ASFs.)



Statechart for NextDate ASFs

Note careful use of StateChart transitions:

Transitions to or from the possible invalid entries blob refer to any ASF in the blob.



Metrics for System Testing

- In PIN Entry, for a given PIN, there are 156 distinct paths from the First PIN Try state to the two successor states.
- Of these, 31 correspond to eventually correct PIN entries.
  - 1 on the first try
  - 5 on the second try
  - 25 on the third try
- The other 125 paths correspond to failed PIN attempts/
- Model-based coverage metrics can control this.

Model-Based Coverage Metrics

- Decision table metrics
  - every condition
  - every action
  - every rule
- Finite state machine metrics
  - every state
  - every transition
  - every path (cycles need to be addressed as in code coverage metrics)
- Petri net metrics
  - every place
  - every port event
  - every transition
  - every marking

Event-Based System Test Coverage Metrics

- Port Input 1: each port input event occurs
- Port Input 2: common sequences of port input events occur
- Port Input 3: each port input event occurs in every "relevant" data context
- Port Input 4: for a given context, all "inappropriate" input events occur
- Port Input 5: for a given context, all possible input events occur
- Port Output 1: each port output event occurs
- Port Output 2: each port output event occurs for each cause

Coverage Metrics for the SATM System

- 19 port input events
- 15 port output events (15 screens)
- 12 “short” use cases

Input Event Incidence Matrix

SUC	Port Input Events																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	x																		
2		x																	
3			x	x															
4				x	x	x													
5							x	x	x										
6							x	x		x	x								
7							x	x				x	x	x					
8							x	x				x	x		x				
9							x	x				x	x			x			
10							x	x				x	x				x		
11																			x
12																		x	

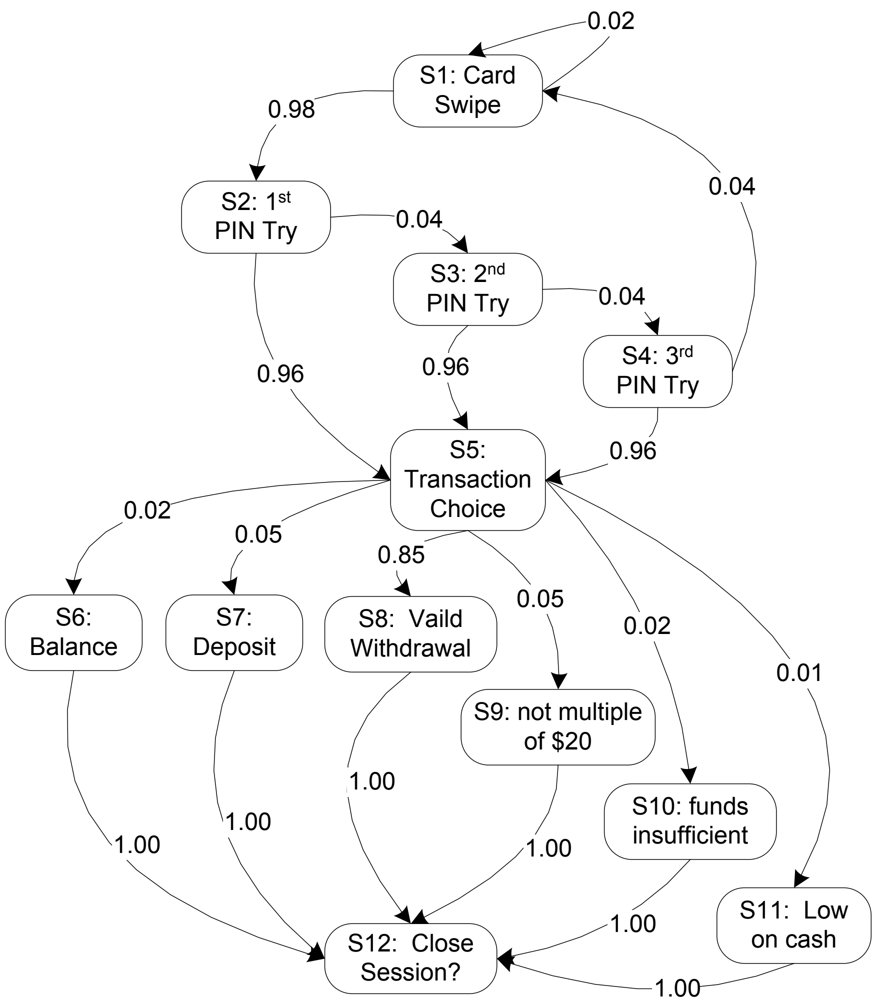
Operational Profiles

- Zipf's Law: 80% of the activities occur in 20% of the space
  - productions of a language syntax
  - natural language vocabulary
  - menu options of a commercial software package
  - area of an office desktop
  - floating point divide on the Pentium chip
- Rationale for operational profile testing
  - a small fraction of all possible threads represents the majority of system execution time
  - find the occurrence probabilities of threads and use these to order thread testing.

Operational Profiles

- A shortcut when test time is limited
- Estimate transition probabilities in a high level finite state machine
- Product of probabilities on a path is the probability of that path executing
- Method
  - Estimate probabilities
  - Compute path probabilities
  - Sort paths most to least likely
  - Test most likely paths until time expires

SATM System Operational Profiles



Selected Path Probabilities

	path	Transition Probabilities					Path Probability
1st	S1, S2, S5, S8, S12	0.999	0.96	0.85	1	1	81.5184%
1st	S1, S2, S5, S7, S12	0.999	0.96	0.05	1	1	4.7952%
1st	S1, S2, S5, S9, S12	0.999	0.96	0.05	1	1	4.7952%
2nd	S1, S2, S3, S5, S8, S12	0.999	0.04	0.96	0.85	1	3.2607%
1st	S1, S2, S5, S6, S12	0.999	0.96	0.02	1	1	1.9181%
1st	S1, S2, S5, S10, S12	0.999	0.96	0.02	1	1	1.9181%
1st	S1, S2, S5, S11, S12	0.999	0.96	0.01	1	1	0.9590%
2nd	S1, S2, S3, S5, S7, S12	0.999	0.04	0.96	0.05	1	0.1918%
2nd	S1, S2, S3, S5, S9, S12	0.999	0.04	0.96	0.05	1	0.1918%
3rd try	S1, S2, S3, S4, S5, S8, S12	0.999	0.04	0.04	0.96	0.85	0.1304%
Bad card	S1, S1	0.001	1	1	1	1	0.1000%

Risk-Based Testing

- Risk = Cost * (Probability of occurrence)
- (An extension of/to operational profiles)
- Hans Schaefer's risk categories
  - Catastrophic: deposits, invalid withdrawals
  - Damaging: normal withdrawals
  - Hindering: invalid ATM card, PIN entry failure
  - Annoying: balance inquiries
- Logarithmic weighting (low = 1, medium = 3, high = 10)

Selected Path Risks

Use Case Description	Use Case Probability	Cost of Failure	Risk
1st try, normal withdrawal	81.5184%	3	2.4456
1st try, deposit	4.7952%	10	0.4795
1st try, withdrawal but insufficient funds	1.9181%	10	0.1918
1st try, withdrawal not multiple of $20	4.7952%	3	0.1439
2nd try, normal withdrawal	3.2607%	3	0.0978
1st try, withdrawal, ATM low on cash	0.9590%	10	0.0959
1st try, balance inquiry	1.9181%	1	0.0192
2nd try, deposit	0.1918%	10	0.0192
Insertion of invalid ATM card	0.1000%	10	0.0100
2nd try, withdrawal insufficient funds	0.0767%	10	0.0077
2nd try, withdrawal not multiple of $20	0.1918%	3	0.0058

Conclusions and Observations

- System testing is based on threads
  - thread identification is the hard part
  - automated thread execution is a good idea
- Model-Based system testing works well
- Helpful to have system level coverage metrics