Unit 5

SYSTEM TESTING AND INTERACTION TESTING

Content

- Threads, Basic concepts for requirements specification, Finding threads, Structural strategies and functional strategies for thread testing, SATM test threads, System testing guidelines, ASF (Atomic System Functions) testing example. Context of interaction, A taxonomy of interactions, Interaction, composition, and determinism, Client/Server Testing,

Contents

- Threads.
- Basic concepts for requirements specification.
- Finding threads.

Threads

- A scenario of normal usage
- A system level test case

Threads Possibilities

- Four candidate threads in our SATM system

1. Entry of a digit
2. Entry of a pin
3. A simple transaction
4. An session containing two or more transaction.

ASF: An atomic system function is an action that is observable at the system level in terms of port input and output events.

Basic concepts for requirements specification

- Data: it is described in terms of variable, records, data structure.

1.Account and PIN

2. System developed in terms of CRUD

3.Relationship between data entities

- Action: it have input and output

- Device

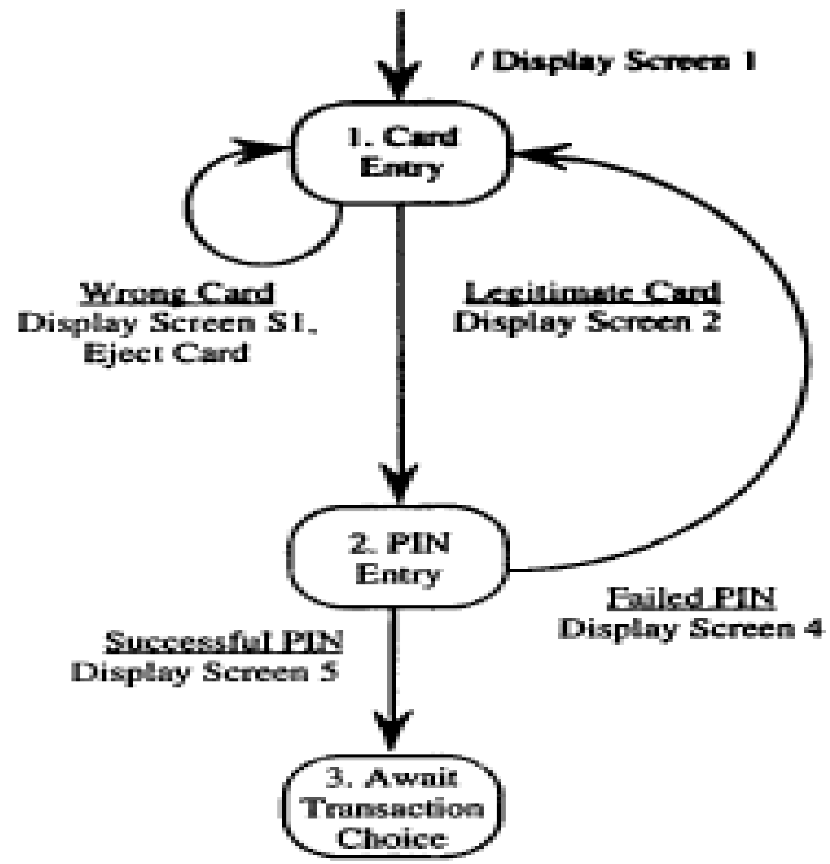
Every system has port devices.

A port is the point at which an IO device is attached to a system.

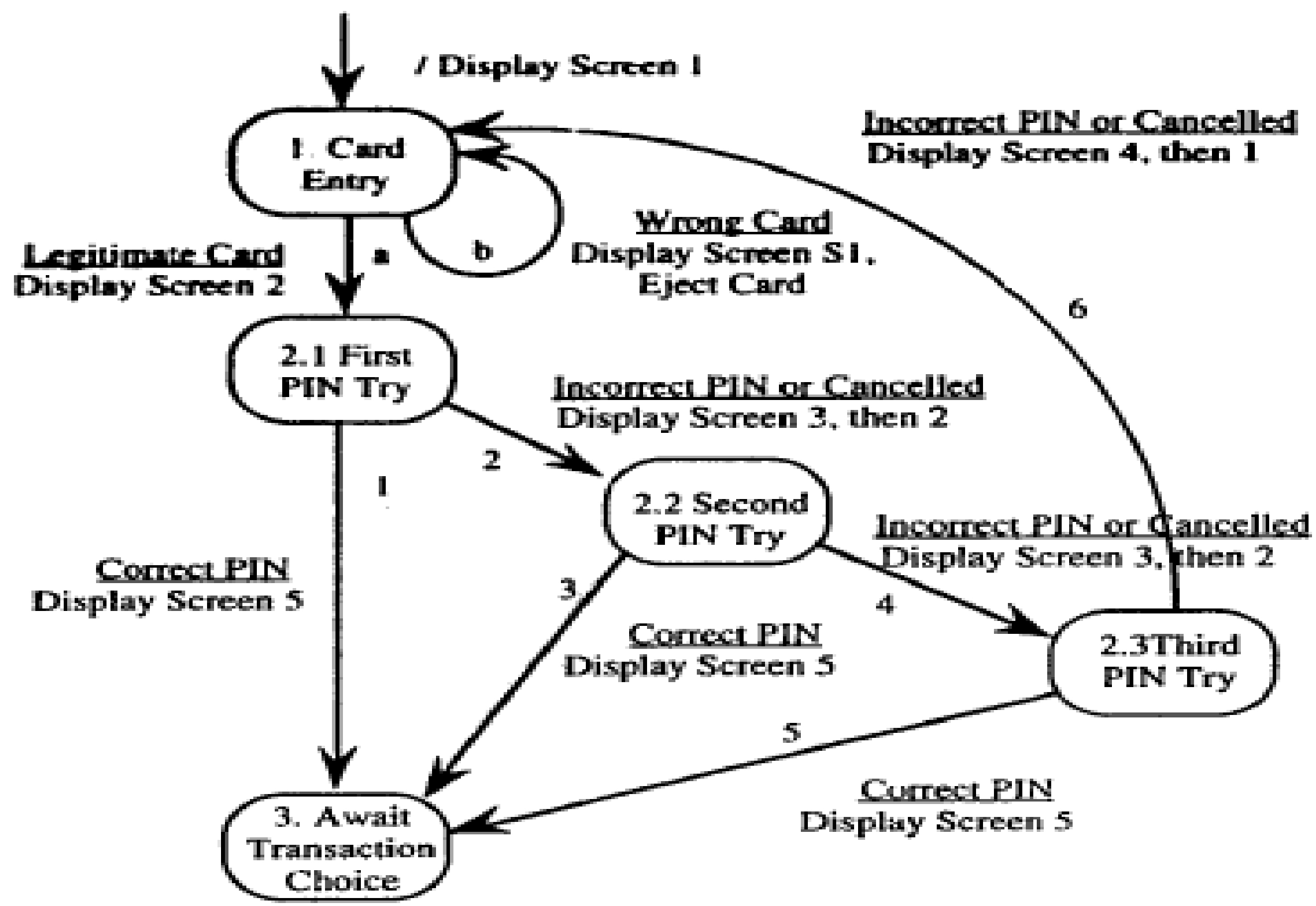
Ex: Display screen, Withdraw doors, card and receipt slot.

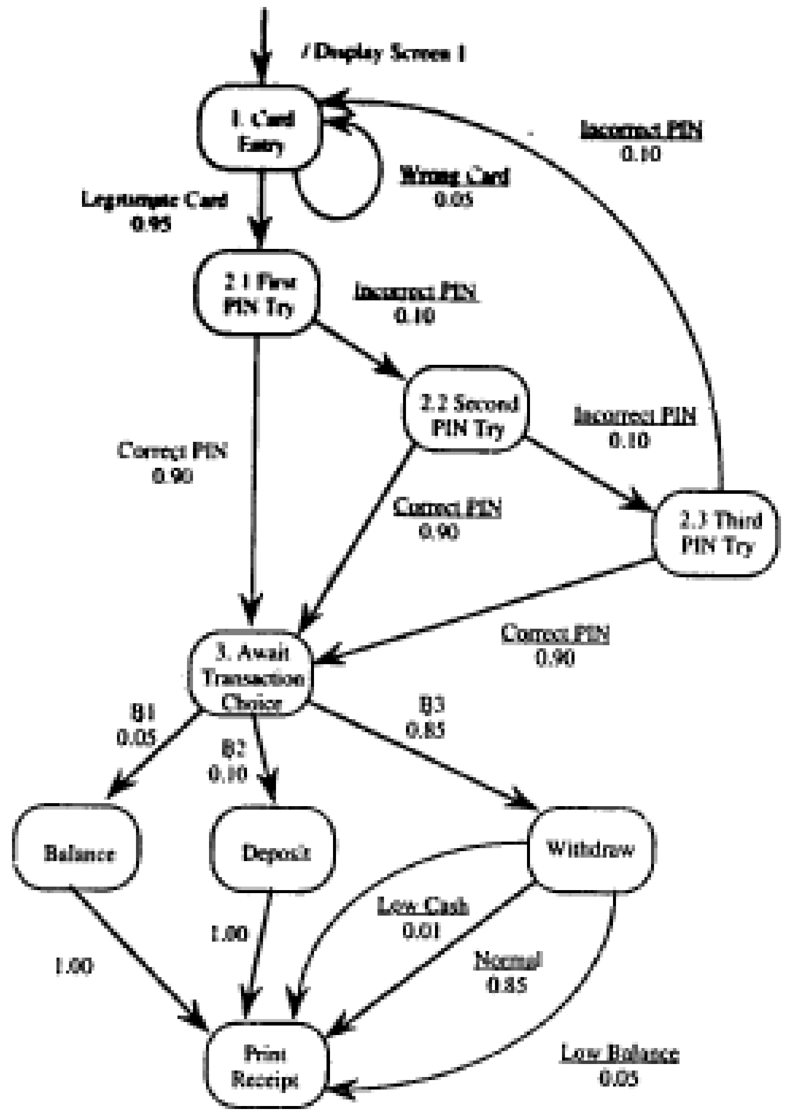
- Event: An event is a system level input that occurs on a port device.

Finding Threads

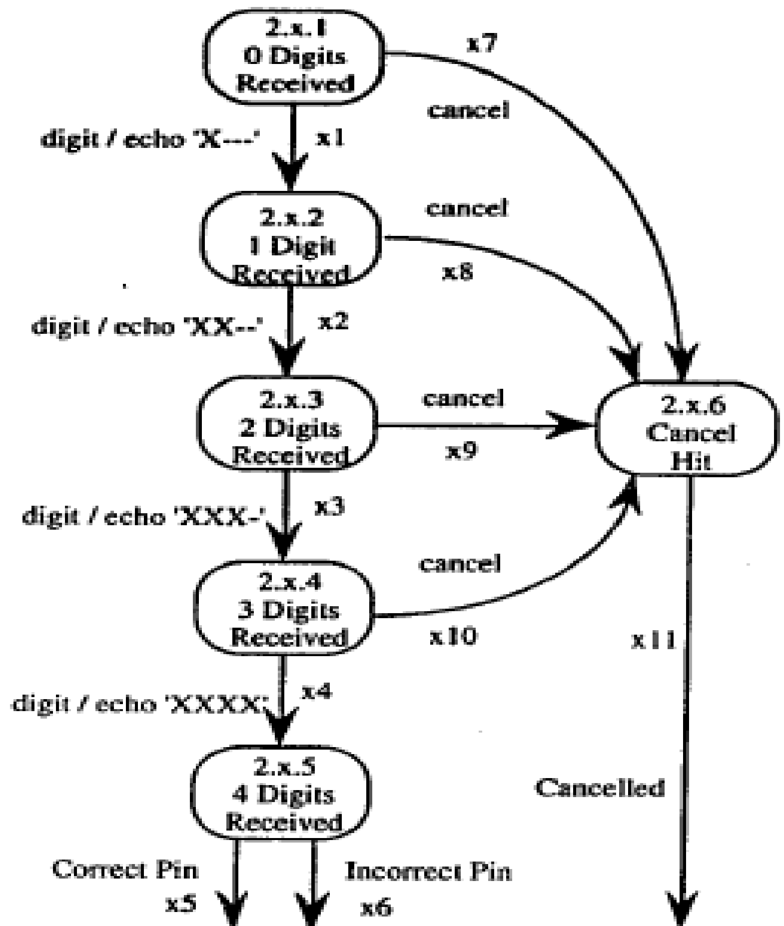


Top-level SATM state machine.


/ Display Screen 1
1. Card Entry
Incorrect PIN or Cancelled
Display Screen 4, then 1
Wrong Card
Display Screen S1,
Eject Card
b
a
Legitimate Card
Display Screen 2
6
2.1 First PIN Try
Incorrect PIN or Cancelled
Display Screen 3, then 2
2
1
2.2 Second PIN Try
Incorrect PIN or Cancelled
Display Screen 3, then 2
Correct PIN
Display Screen 5
3
4
Correct PIN
Display Screen 5
2.3Third PIN Try
5
Correct PIN
Display Screen 5
3. Await Transaction Choice



Transition probabilities for the SATM system.



PIN Try finite state machine.

Table 14.3 Port Event Sequence for Correct PIN on First Try

Port Input Event	*Port Output Event*
	Screen 2 displayed with '- - - -'
1 pressed	
	Screen 2 displayed with 'X- - -'
2 pressed	
	Screen 2 displayed with 'XX- -'
3 pressed	
	Screen 2 displayed with 'XXX-'
4 pressed	
	Screen 2 displayed with 'XXXX'
(Correct PIN)	Screen 5 displayed

Table 14.4 Port Event Sequence for Correct PIN on Third Try

Port Input Event	*Port Output Event*
	Screen 2 displayed with '- - - -'
1 pressed	
	Screen 2 displayed with 'X- - -'
2 pressed	
	Screen 2 displayed with 'XX- -'
3 pressed	
	Screen 2 displayed with 'XXX-'
5 pressed	
	Screen 2 displayed with 'XXXX'
(Incorrect PIN)	Screen 3 displayed
(Second try)	Screen 2 displayed with '- - - -'
1 pressed	
	Screen 2 displayed with 'X- - -'
2 pressed	
	Screen 2 displayed with 'XX- -'
3 pressed	
	Screen 2 displayed with 'XXX-'
Cancel key pressed	
(End of second try)	Screen 3 displayed
	Screen 2 displayed with '- - - -'
1 pressed	
	Screen 2 displayed with 'X- - -'
2 pressed	
	Screen 2 displayed with 'XX- -'
3 pressed	
	Screen 2 displayed with 'XXX-'
4 pressed	
	Screen 2 displayed with 'XXXX'
(Correct PIN)	Screen 5 displayed

Table 14.7 **Node and Edge Traversal of a Thread**

Port Input Event	*Port Output Event*	*Nodes*	*Edges*
	Screen 2 displayed with '- - - -'	2.1	a
1 pressed		2.1.1	
	Screen 2 displayed with 'X- - -'		x1
2 pressed		2.1.2	
	Screen 2 displayed with 'XX- -'		x2
3 pressed		2.1.3	
	Screen 2 displayed with 'XXX-'		x3
5 pressed		2.1.4	
	Screen 2 displayed with 'XXXX'		x4
(Incorrect PIN)	Screen 3 displayed	2.1.5, 3	x6, 2
(Second try)	Screen 2 displayed with '- - - -'	2.2	
1 pressed		2.2.1	
	Screen 2 displayed with 'X- - -'		x1
2 pressed		2.2.2	
	Screen 2 displayed with 'XX- -'		x2
3 pressed		2.2.3	
	Screen 2 displayed with 'XXX-'		x3
Cancel pressed		2.2.4	x10
(End of second	Screen 3 displayed	2.2.6	x11
try)	Screen 2 displayed with '- - - -'	2.3	4
1 pressed		2.3.1	
	Screen 2 displayed with 'X- - -'		x1
2 pressed		2.3.2	
	Screen 2 displayed with 'XX- -'		x2
3 pressed		2.3.3	
	Screen 2 displayed with 'XXX-'		x3
4 pressed		2.3.4	
	Screen 2 displayed with 'XXXX'		x4
(Correct PIN)	Screen 5 displayed	2.3.5, 3	x5, 5

System Testing

Beyond unit testing

System Testing

- Of the three levels of testing, system level testing is closest to everyday experience
  - We evaluate a product with respect to our expectations
- Concerned with the application’s externals
- We tend to approach system testing from a functional standpoint rather than from a structural one

System Testing

- Functional testing
  - **Objective**: Assess whether the application does what it is supposed to do
  - **Basis**: Behavioral/functional specification
  - **Test case**: A sequence of Atomic System Functions

Atomic System Function

- Atomic System Function (ASF): is an action that is observable at the system level in terms of port input and output events.
  - It begins with a port input event,
  - traverses one or more MM-Paths,
  - and terminates with a port output event.

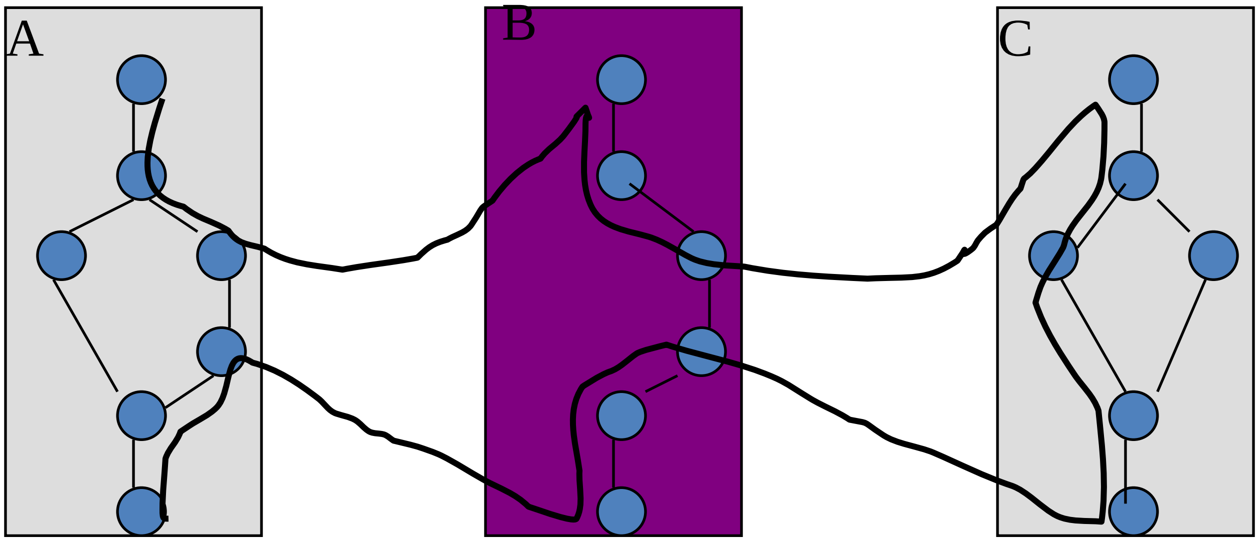
Atomic System Function

- When viewed from the system level, there is no compelling reason to decompose an ASF into lower levels of detail (hence the atomicity)
- For example in an ATM system
  - Digit entry
  - Card entry
  - Cash dispensing
  - PIN entry is probably too big

Atomic System Function

- ASFs are an upper limit for MM-Paths:
  – MM-Paths should not cross ASF boundaries
- ASFs represent the seam between integration and system testing:
  – they are the largest item to be tested during integration testing,
  – and the smallest item for system testing

ASF Example



MM-path: Interleaved sequence of **module exec path** and **messages**
Module exec path: **entry-exit** path in the same module

<u>A</u>tomic <u>S</u>ystem <u>F</u>unction: port input, … {MM-paths}, … port output

Test cases: exercise ASFs

Threads

- We view system testing in terms of threads of system level behavior.
- Many possible views of a thread:
  - a scenario of normal usage
  - a system level test case
  - a stimulus/response pair
  - behavior that results from a sequence of system level inputs
  - an interleaved sequence of port input and output events

Threads

– a sequence of transitions in a state machine description of a system
– an interleaved sequence of object messages and method executions
– a sequence of machine instructions
– a sequence of source instructions
– a sequence of atomic system functions

Thread Levels

- Threads have distinct levels:
  - **Unit** level thread is understood as an execution-time path of instructions or some path on a flow graph
  - **Integration** level thread is a sequence of MM-paths that implement some atomic function. Usually denoted as a sequence of module executions and messages
  - **System** level thread is a sequence of atomic system functions

Thread Levels

- Since ASFs have port events as their inputs and outputs, the sequence of ASFs implies an interleaved sequence of port input/port output events.
- Threads provide a unifying view of the three levels of testing:
  - Unit testing tests individual functions
  - Integration tests examine interaction among units
  - System testing examines interactions among ASFs.

Thread Definitions

- **ASF Graph**: a directed graph in which nodes are ASFs and edges represent sequential flow.
- **Source ASF**: an ASF that appears as a source node in the ASF graph of a system
- **Sink ASF**: an ASF that appears as sink node in the ASF graph.
- **System thread**: a path from a source ASF to a sink ASF in the ASF graph of a system.

Basis Concepts for Requirements Specification

- The objective is to discuss system testing with respect to a basis set of requirements specification constructs
- Every system can be specified in terms of the following requirements specification constructs:
  - Data
  - Actions
  - Ports
  - Events
  - Threads

Data

- For a system that is described in terms of its data,
  - the focus is on the information used/created by the system (described in terms of variables, data structures, fields, records, data stores, and files)
- The data centered view is also starting point for many OO analysis methods.
- Data refers to information that is either initialized, stored, updated or possibly destroyed.

Data

- Data-centric systems are often specified in terms of CRUD actions (Create, Retrieve, Update, Delete)
- Often, threads can be identified directly from the data model
- Also possible to have read-only data (i.e. expected PIN pairs, etc.)
  - this must be part of system initialization process
    - if not, then there must be threads that create the data.
    - Hence read-only data is an indicator of source ASFs.

Actions

- Action-centered modeling is the most common requirements specification form.
  - Actions have inputs and outputs and these can be either data or port events.
  - Actions can also be decomposed in to lower level actions (i.e. typical data flow diagrams).
- The input/output view of actions is the basis of functional testing

Devices

- Every system has ports (and port devices):
  - Sources and destinations of system level inputs and outputs.
- If no physical port devices in system, much of system testing can be accomplished by moving the port boundary inward to the logical instances of port events.

Events

- Events have some characteristics of data and some of actions
- An event is a system level input which occurs at a port.
- Events can be inputs to or outputs of actions:
  - Can be either discrete or continuous
  - Discrete events have a time duration and this can be critical in real-time systems.

Threads

- Threads are the least frequently used of the fundamental constructs.
  - Since threads are tested, it is up to the tester to find them in the interactions of the data, events, and actions.
- Finding Threads
  - A finite state machine model of the system is a good starting point to find threads since the paths are easily converted to threads.

Finding Threads

- Usually, one deals with a hierarchy of state machines i.e. the card entry state of an ATM may be decomposed into lower levels that deal with details like:
  - jammed cards,
  - cards that are upside-down,
  - checking the card against the list of cards for which service is offered, etc.).

Finding Threads

- At this level, states correspond to states of processing, and transitions are caused by logical (rather than port) events.
- Once the details of a macro-state are tested we continue with the next macro-state
- Within the decomposition of the macro state we need to identify the port input and port output events

Finding Threads

- The hierarchy of finite state machines multiplies the number of threads
- Ideal to reach a state machine in which transitions are caused by actual port input events, and the actions on transitions are port output events
  - Ggenerating the test cases for these threads is mechanical
  - Just follow a path of transitions noting the inputs and outputs as they occur along the path

Structural Strategies for Thread Testing

- Generating the threads may be easy, but to decide which one to test is complex
- Encounter the same path explosion problem at system level as at unit level
- Bottom Up Threads
  - When state machines are organized in a hierarchy, it is possible to work bottom up

Structural Strategies for Thread Testing

- As seen in unit testing, structural testing can be misleading
  - The assumption is that path traversal uncovers faults and traversing a variety of paths reduces redundancy
- A more serious flaw with these threads is that it is not really possible to execute them “by themselves” due to the hierarchical state machines.

Coverage Metrics

- Since FSMs are directed graphs, use same test coverage metrics as at the unit level
- The hierarchical relationship indicates that the upper level machine treats the lower level machine as a procedure that is entered and returned from

Coverage Metrics

- Two fundamental choices are node coverage and edge coverage
  - Node coverage is similar to statement coverage at unit level: bare minimum .
  - Edge (state transition) coverage is more acceptable

INTERACTION TESTING

- It is a relationship Interacts With among
- Data
- Events
- Threads
- Actions
- Ports
- The relationship is reflexive It is binary relation between Data & events Data & threads Events & threads

Properties of threads and processors

- Textbook has two meanings for event Causes confusion, ambiguity, wordy explanations Use two words Use event for instant Use state or activity for duration Occurs between two e
- Properties of threads and processors
- Threads have duration
- They are activities
- At one time a processor can execute only one thread Events.
- A processor is in a state of executing a thread Timesharing, multiprocessing interleaves thread execution Processor changes state for each thread Here thread durations overlap in time

- On one processor events can be simultaneous within the minimum resolution of time-grain markers .
- BUT reality (hardware) puts an order on those events – puts them in a sequence.

  - As far as we can tell it is a random choice
  - At another occurrence the events may be ordered in a different sequence
  - That is an essential difficulty of interaction testing

- On different processors, events can occur simultaneously
- Common events by definition must occur at the same time
- Consider a two people colliding – the collision is a common event to the two people (processors)
- Synchronous communication for processors start and end with common events

- For a single processor
- Input and output events occur during thread execution
- From the perspective of a thread they cannot occur simultaneously, because they occur at instructions and instructions are executed sequentially
- From the perspective of devices port events can be simultaneous
- For each port events occur in time sequence

- Threads occur only within one processor
- Do not cross processor boundaries
- Have trans-processor quiescence when threads reach processor boundaries
- Analogous to crossing unit boundaries in integration testing

- What we want is sane behaviour
- This results from considering events to be in a linear sequence
- For example synchronous communications takes into account message transmission time Break the communication into events such as Sender starts sending
- Receiver starts receiving
- Sender ends sending

- Receiver ends receiving

Taxonomy of interactions

- Static interactions in a single processor system Static interactions in multiprocessor system Dynamic interactions in a single processor system
- Dynamic interactions in multiprocessor system

- Given two propositions P and Q
- They are contraries if both cannot be true
- Sub-contraries if both cannot be false
- Contradictories if exactly one is true
- R is a subaltern of P if the truth of P guarantees the truth of R – i.e. $P \rightarrow R$
- Rules in a decision table, if correct, are contradictories

Static interactions in a single processor

- Analogous to combinatorial circuits
- Model with decision tables and unmarked event-driven Petri nets
- Telephone system example
- Call display and unlisted numbers are contraries
- Both cannot be satisfied
- Both could be waived

Data-data connectedness – Logical relationships

- 0-connected
- Logically independent
- 2-connected
- Sub-alternation
- 3-connected – bidirectional
- Contraries
- Contradictories
- Sub-contraries

EXAMPLES

- 3-connected data-data
- When data are deeply related, as in repetition and semaphores
- 1-connected data-event
- Context-sensitive port input events

Dynamic, single processor interactions

- Six potential interaction pairs
- Combination pairs of
- Data
- Events Threads
- Each interaction can exhibit 4 different graph connectedness attributes
- Result is 24 sub-categories for these interactions IAT–31

Thread –thread interaction

- Each thread can be represented by an EDPN
- The symbolic names of the places and transitions correspond to those in the EDPN for the system
- Synonyms in thread nets need to be resolved when they interact

Dynamic Multiprocessor Interactions

- Problem here is threads and events occur in parallel
- We have concurrent behaviour with a collection of communicating sequential processors (CSP)
- Have non-deterministic behaviour
- To fully understand need to learn the mathematics of CSP
- Without that can only work through an example

Determinism

- A system is deterministic if, given its inputs, we can always predict its outputs
- A system is deterministic if it always produces the same outputs for a given set of inputs
- (For a non-deterministic system it may be difficult to demonstrate different output
- Process P chooses non-deterministically at every step whether to engage in event

- a or b Process Q chooses non-deterministically once whether to engage only with event a or only with event b
- P = (a → P) (b → P) Q = (a → Qa) (b → Qb) Qa = (a → Qa) Qb = (b → Qb)
- P is deterministic ↔ ∀s : traces (P) • X ∈ refusals (P / s) ↔ X ∩ (P / s)1 = {} P1 = { e * 〈 e 〉 ∈ traces (P) } ? A system is deterministic if at every step the system never refuses to engage in any external event appropriate at that step

- P is deterministic ↔ ∀s : traces (P) • X ∈ refusals (P / s) ↔ X ∩ (P / s)1 = {} P1 = { e * 〈 e 〉 ∈ traces (P) }
- P1 definition is the set of events in which P may engage on the first step
- P / s is the process after P has engaged in all of the events in the trace s
- A trace is a record of the external events in which a process has engaged
- A refusal is a set of events in which a process refuses to engage

Client Server Complexities

- Base system has program components Database, application, presentation (logical output) Have a centralized, fat server and distinction
- Entire system includes above items plus Network
- GUI May have homogeneous or heterogeneous processors

Client Server Testing

- Extend notion of threads beyond an EDPN
- CS transaction
- A sequence of threads across EDPN boundaries
- Client processor --> network --> application -- >DBMS back again
- Much of the system is stable – e.g. DBMS, existing application Should testing be needed Use functional testing – no source text