System Testing

Chapter 14

Overview

- Common experience
  - **Use functional testing**
  - **Looking for correct behaviour, not looking for faults**
- Intuitively familiar
  - **Too informal**
- Little test time due to delivery deadlines
  - **Too informal**
- Need a good understanding and theory
  - **Use threads**
  - **Atomic system functions**

Possible thread definitions

- Difficult to define
  - A scenario of normal usage
  - A system-level test case
  - A stimulus-response pair
  - Behaviour 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 executions
  - A sequence of
    - Machine instructions
    - MM-paths
    - Program statements
    - Atomic system functions

Thread levels

- Unit level
  - An execution-time path of program text statements / fragments
  - A sequence of DD-paths
  - Tests individual functions
- Integration level
  - An MM-path
  - Tests interactions among units
- System level
  - A sequence of atomic system functions
    - Results in an interleaved sequence of port input and output events
  - Tests interactions among atomic system functions

Basic questions

- What is a thread?
  - **How big is it?**
  - **Where do we find them?**
  - **How do we test them?**

Definition – atomic system function

- Is an action that is observable at the system level in terms of port input and output events
- Separated by points of event quiescence
  - **Analogous to message quiescence at the integration level**
  - **Natural end point**
- Begins at a port input event
- Terminates with a port output event
- At system level no interest in finer resolution
- Seam between integration and system testing
  - **Largest item for integration testing**
  - **Smallest for system testing**

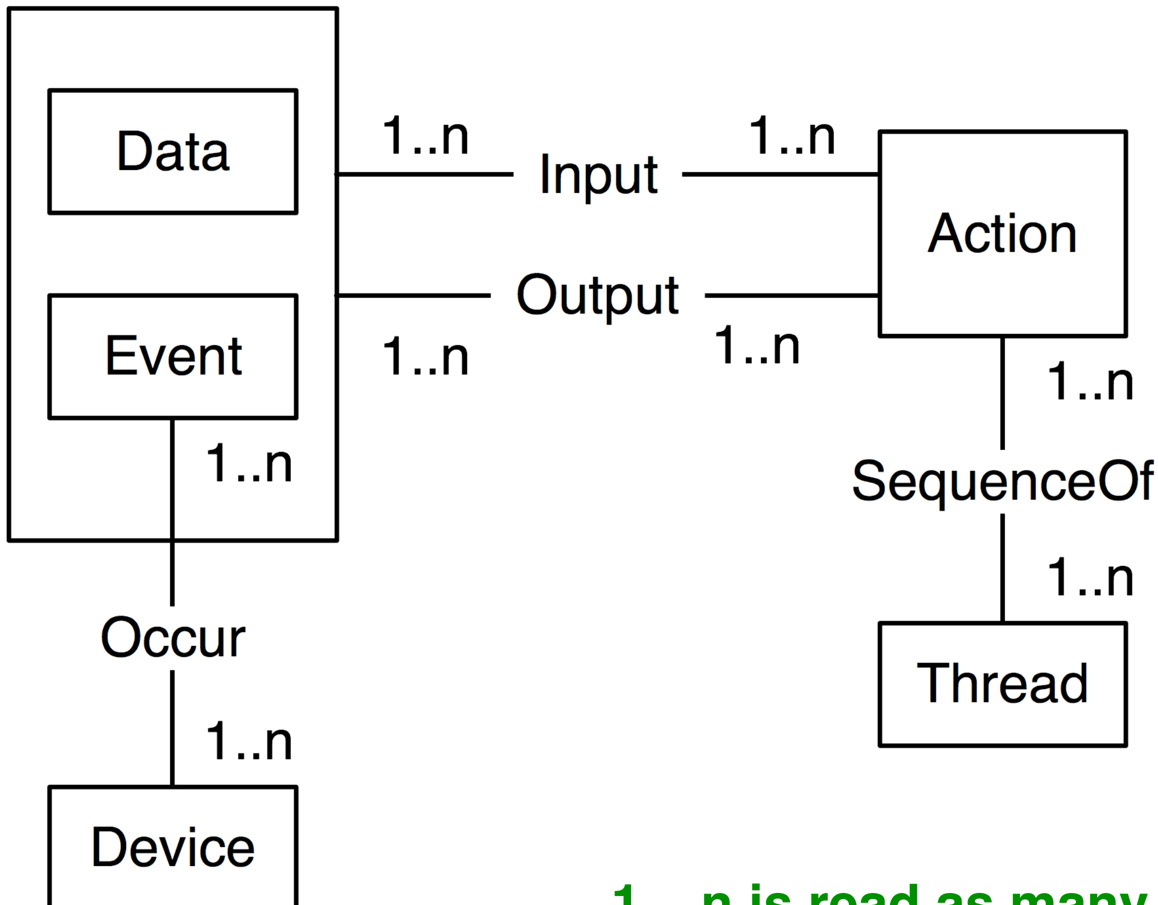
Thread-related definitions

- Atomic system function graph – ASF graph
  - A directed graph where
    - Nodes are ASFs
    - Edges represent sequential flow
- Source / Sink atomic system function
  - A source / sink node in an ASF
- System thread
  - A path from a source ASF to a sink ASF
- Thread graph
  - A directed graph where
  - Nodes are system threads
  - Edges represent sequential execution of threads

Basis for requirements specifications

- All requirements specifications are composed of the following basis set of constructs
  - **Data**
  - **Events**
  - **Threads**
  - **Actions**
  - **Devices**
- All systems can be described in terms of the basis set of constructs

Basis concepts E/R model



1 .. n is read as many

Data

- Focus on information used and created by the system
- Data is described using
  - **Variables, data structures, fields, records, data stores and files**
  - **Entity-relationship models describe highest level**
  - **Regular expressions used at more detailed level**
    - **Jackson diagrams (from Jackson System Development)**
- Data view
  - **Good for transaction view of systems**
  - **Poor for user interface**

Data and thread relationships

- Threads can sometimes be identified from the data model
  - **1-1, N-1, 1-N and N-N relationships have thread implications**
    - **Need additional data to identify which of many entities is being used – e.g. account numbers**
- Read-only data is an indicator of source atomic system functions

Actions

- Action-centered modeling is a common form for requirements specification
- Actions have input and output
  - **Either data or port events**
- Synonyms
  - **Transform, data transform, control transform, process, activity, task, method and service**
- Used in functional testing
- They can be refined (decomposed)
  - **Basis of structural testing**

Devices

- Port input and output handled by devices
- A port is a point at with an I/O device is attached to a system
- Physical actions occur on devices and enter / leave system through ports
  - **Physical to logical translation on input**
  - **Logical to physical translation on output**
- System testing can be moved to the logical level – ports
  - **No need for devices**
- Thinking about ports helps testers define input space and output space for functional testing

Events

- A system-level input / output that occurs on a port device
- Data-like characteristic
  - **Input / output of actions**
  - **Discrete**
- Action-like characteristic
  - **The physical – logical translation done at ports**
- From the tester's viewpoint think of it as a physical event
  - **Logical event is a part of integration testing**

On continuous events

- No such thing
- Events have the following properties
  - **Occur instantaneously – No duration**
    - **A person can start eating and stop eating**
    - **No corresponding event eating**
  - **Take place in the real world, external to the system**
  - **Are atomic, indivisible, no substructure**
  - **Events can be common among entities**
- If you want or need to handle duration, then you need start and end events and time-grain markers to measure the duration
- Events are detected at the system boundary by the arrival of a message

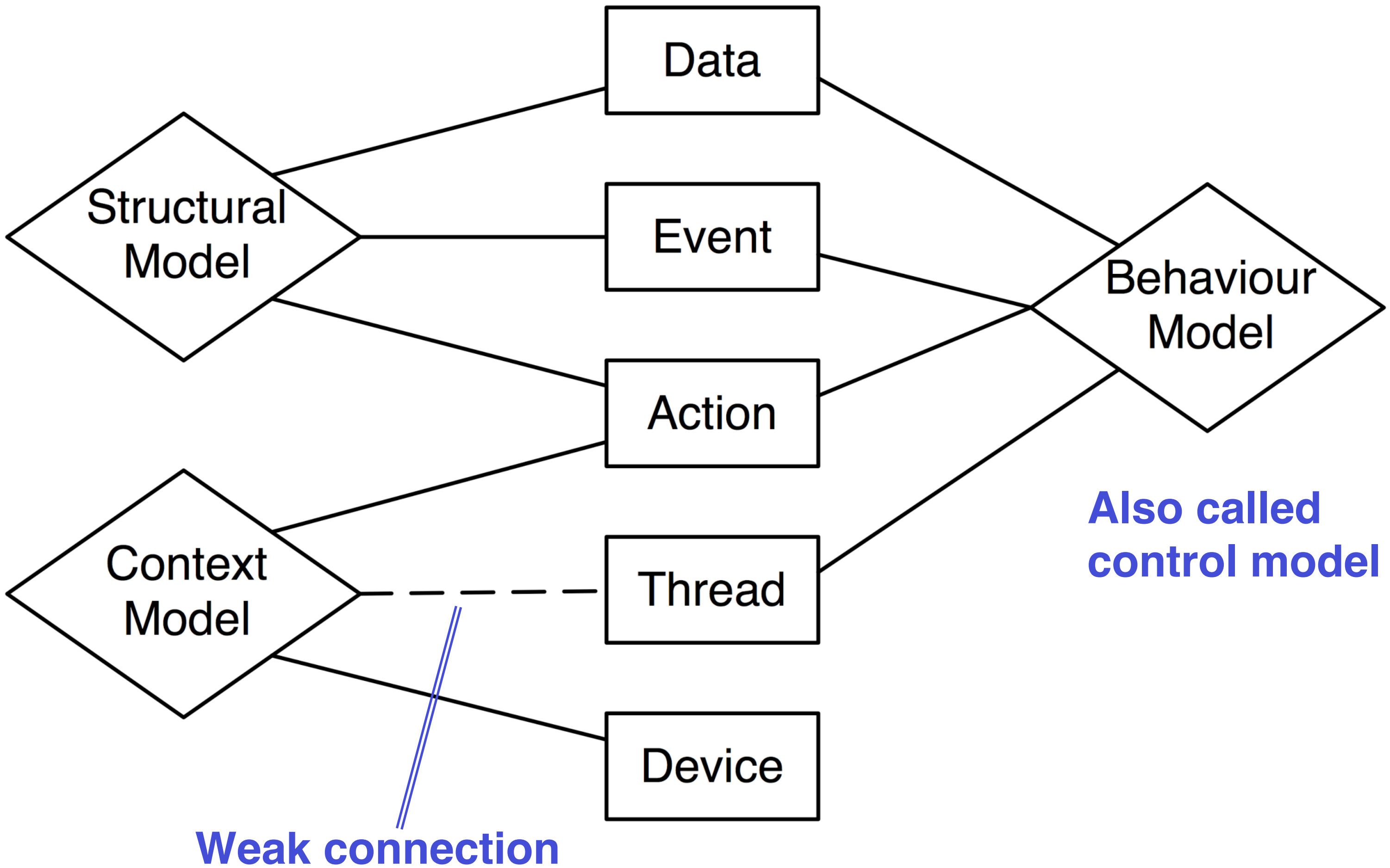
On the temperature event

- Temperature is not an a continuous event
  - **To be continuous a continuous message would have to arrive at the system boundary**
    - **A continuous message is not a meaningful concept**
    - **Messages are discrete**
- In practice, thermometers do not send messages to a system, instead a system reads a thermometer
  - **Reading is at the discretion of the receiver not the sender**
    - **Called a statevector read**
  - **The other option is message sending which is at the option of the sender, receiver can only read after the message is sent**
    - **Called a data read**

Threads

- Almost never occur in requirements specifications
  - **Testers have to search for them in the interactions among data, actions and events**
  - **Can occur in rapid prototyping with a scenario recorder**
- Behaviour models of systems make it easy to find threads
  - **Problem is they are models – not the system**

Modeling with basis concepts



Behaviour model

- Need appropriate model
  - **Not too weak to express important behaviours**
  - **Not too strong to obscure interesting behaviours**
- Decision tables
  - **Computational systems**
- Finite state machines
  - **Menu driven systems**
- Petri nets
  - **Concurrent systems**
  - **Good for analyzing thread interactions**

Finding threads in finite state machines

- Construct a machine such that
  - **Transitions are caused by port input events**
  - **Actions on transitions are port output events**
    - **Definition of the machine may be hierarchical, where lower levels are sub-machines – may be used in multiple contexts**
- Test cases follow a path of transitions
  - **Take note of the port input and output events along the path**
  - **Problem is path explosion**
    - **Have to choose which paths to test**

Structural strategies for thread testing

- Bottom-up
  - **The only one**

Structural coverage metrics

- Use same coverage metrics as for paths in unit testing
  - **Finite state machine is a graph**
- Node coverage is analogous to statement coverage
  - **The bare minimum**
- Edge coverage is the better minimum standard
  - **If transitions are in terms of port events, then edge coverage implies port coverage**

Functional strategies for thread testing

- Event-based
- Port-based
- Data-based

Event-based thread testing

- Five port input thread coverage metrics are useful
  - **PI1: Each port input event occurs**
    - **Inadequate bare minimum**
  - **PI2: Common sequences of port input events occur**
    - **Most common**
    - **Corresponds to intuitive view of testing**
    - **Problem: What is a common / uncommon sequence?**
  - **PI3: Each port input event occurs in every relevant data context**
    - **Physical input where logical meaning is determined by the context in which they occur**
    - **Example is a button that has different actions depending upon where in a sequence of buttons it is pressed**

Event-based thread testing – 2

- **PI4: For a given context, all inappropriate input events occur**
  - **Start with a context and try different events**
  - **Often used on an informal basis to try to break the system**
  - **Partially a specification problem**
    - Difference between prescribed and proscribed behaviour
    - Proscribed behaviour is difficult to enumerate

- **PI5: For a given context, all possible input events occur**
  - **Start with a context and try all different events**

Event-based thread testing – 3

- PI4 & PI5 are effective
  - **How does one know what the expected output is?**
  - **Good feedback for requirements specification**
  - **Good for rapid prototyping**

Event-based thread testing – 4

- Two output port coverage metrics
  - **PO1: Each port output event occurs**
    - **An acceptable minimum**
    - **Effective when there are many error conditions with different messages**
  - **PO2: Each port output event occurs for each cause**
    - **Most difficult faults are those where an output occurs for an unsuspected cause**
    - **Example: Message that daily withdrawal limit reached when cash in ATM is low**

Port-based thread testing

- For each port
  - **Try threads that exercise ports with respect to the events in which they can engage**
  - **Useful when port devices come from outside suppliers**
  - **The many-to-many relationship between ports and events should be exercised in each direction**
    - **See E/R diagram**
- Complements event-based testing

Event driven systems

- Event and port based testing is good for event driven systems
- Reactive systems – react to input events, often with output events
  - **Are long running**
  - **Maintain a relationship with the environment**
  - **E/R model is simple and not particularly useful**

Note: payroll example when properly designed is a long running program. It is a sequence of payroll runs, where each run is in the context of previous runs.

Data-based thread testing

- Good for systems where data is of primary importance
  - **Static**
  - **Transformational**
    - **Support transactions on a database**
  - **E/R model is dominant**

Data-based thread testing – 2

- Data-based coverage metrics – based on E/R model
  - **DM1: Exercise the cardinality of every relationship**
    - **1-1, 1-N, N-1, N-N**
  - **DM2: Exercise the participation of every relationship**
    - **Does every specified entity participate**
    - **Can have numerical limits**
  - **DM3: Exercise the functional dependencies among relationships**
    - **Functional dependencies are explicit logical connections**
      - Cannot repair a machine that one does not have

Thread explosion – Pseudo-structural testing

- Use the graph-based metrics as a cross-check on the functional coverage metrics
  - **Analogous to using DD-paths to identify gaps and redundancies of functional testing at the unit level**
- Pseudo occurs because graph is on the control model, which is not the system itself
- Weak method if model is poor
  - **used the incorrect model for type of system; transformational, interactive, concurrent**
  - **Did not design a good model**

Thread explosion – Pseudo-structural testing – 2

- Decision tables and finite state machines good for atomic system function testing
- Thread-based testing is best done with Petri nets
  - **Devise tests to cover every place, every transition, every sequence of transitions**

Thread explosion – Operational profiles

- Make use of Zipf's law
  - **80% of activities occur in 20% of the activity space**
- Make use of the idea
  - **Testing is to find cases that when a failure occurs the presence of a fault is revealed**
- Make use of the fact
  - **Distribution of faults is indirectly related to the reliability of a system**
- Make use of the definitions
  - **System reliability is the probability that no failure occurs within a given time-period**
  - **Faults are on low use threads – the system is reliable**
  - **Faults are on high use threads – system is unreliable**

Thread explosion – Operational profiles – 2

- When test time is limited maximize probability of finding faults by finding failures in the most frequently used threads
- Use a decision tree
  - **Works well with hierarchy of finite state machines**
  - **Estimate the probability of each outgoing transition (sum to 1)**
    - **Can get statistics from customer monitoring / feedback**
  - **Probabilities in sub-states split the probability of the parent state**
  - **The probability of a thread is the product of the transitions comprising the thread**
  - **Test from high to low probability**

Thread explosion – Progressive & regressive testing

- Use of builds makes a need for regression testing
  - **20% of changes to a system create new faults**
  - **Regression testing takes a significant amount of time**
  - **Reduce by looking at difference between progression and regression testing**
- Most common regression testing is to run all the tests
- Progressive testing needs to be diagnostic to isolate faults more easily
  - **Use short threads**
- Regressive testing not as concerned with fault isolation
  - **Use long threads**

Thread explosion – Progressive & regressive – 2

- Together have good coverage
  - **State & transition coverage sparse for progressive tests, dense for regressive tests**
- Different from operational profiles
  - **Good regressive tests have low operational probability**
  - **Good progressive tests have high operational probability**