Testing MDA Platform Independent Models

Greg Eakman
Pathfinder Solutions
grege@pathfindermda.com
Model Driven Architecture

- Model components independent of implementation technologies
- Transformations map model to implementation
- Leverage models into integration testing
- Test Independently
  - Models
  - Transformation rules
Embedded MDA

- Executable models
- Large number of embedded systems platforms
- Non-functional requirements
- Models tested on one platform may not work on another
What Is a Platform?

• Provides the support for model execution:
  – Architectural topology (processors, processes, threads)
  – Communication technologies
  – Supporting components
  – Operating system
  – Implementation language
  – Design and implementation patterns
• Initial State + Inputs $\Rightarrow$ Final State + Outputs
Testability

- **Testability** - relative effort to verify a system is correct
- **Observability** - the ability to detect errors in the output or the internal state
- **Controllability** - the ability to force system through a particular path
- **Repeatability** - the ability to reproduce test results
MDA Components: Domains

- Logical Separation
- Well defined boundaries
- Modeled domains consist of:
  - API
  - Classes
  - Statecharts
  - Action Language
Design for Test

- Set of rules and methods applied during design and translation to make high quality testing possible
- Facilitate the execution of tests
- Simplify the process of finding defects
- Design for Test does not add any additional functionality to the application.
Instrumentation

- Test hooks
- Source code debugger symbol table
- Test and debug at model level
- Need visibility into internal state
  - Testability
  - Controllability
  - Observability
- Probe effect
- Emulation
• **Event Loop**
  - Central point of control for sequencing events
  - Thread of control event dequeue algorithm:
    • while events in queue:
      - get next event from queue
      - dispatch to destination class instance
      - delete event
Event Queue Instrumentation

- Access to event queue allows
  - Reordering of events
  - Control over sequence of execution
  - Add an event to simulate external input
  - Simulate time by firing a timeout event
• Monitoring and Browsing of instance data
• Event breakpoint code fragment

```c
static List regBreaks;
void checkBreakpoint(Event e) {
    for (Breakpt b=regBreaks.top();
        b!=NULL;b=regBreaks.next()){
        if (b.matches(e)){
            Agent::notifyBreakpoint(e);
            Agent::waitForGo();
        }
    }
}
```
void Agent::waitForGo() {
    while (msg = Agent::waitForMessage() != 0) {
        if (msg.isGo()) { return; }
        else { msg.process(); }
    }
}
Test Instrumentation Interface

- Standardize test driver-application interface
  - Model-level Testing and Debugging
- Facilitate testing of models across platforms
- Package test cases with PIMs
• Communication channel: TCP/IP, RS232, ...
• Messages are commands and supplemental data or responses

<table>
<thead>
<tr>
<th>Type</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_INSTANCE</td>
<td>&quot;Aircraft&quot;</td>
<td>&quot;Altitude=13000&quot;</td>
</tr>
</tbody>
</table>
Test Execution - Initialization

• Determine the initial state of the system
• Does not need to be the startup state
• Determine instance population and values
• Error conditions
• **Stimulus is application dependent**
  – API calls
  – Message buffers
• **Timing and sequencing of calls**
• **Emulate target environment**
Test Execution - Data Collection

- Collect outputs
  - Outgoing messages
  - Calls to other components
  - Trace information
- Capture/playback
- Collect relevant aspects of internal state
- Verify results against expected data
Emulation

- The replacement of part of the system with a simulated version for the purposes of testing
- Greater controllability of software under test
- Understanding of interfaces and emulation boundary provided by models
- Tests conditions that would otherwise be difficult to recreate
Component Boundaries

- Test individual components
- API
- Understand boundaries and dependencies
• Emulation approaches:
  – Simple stubs
  – Test driver to apply input stimulus
  – Controllable scripted or interactive support for flexible emulation

• Scenarios to help determine required drivers and emulation stubs
Modeling the Test

Radar
- direction: Degrees = 90
- maxSideAngle: Degrees = 60
- range: Meters = 500000
- trackUpdatePeriod: Seconds = 1

Track
- range: Meters = 50000
- angle: Degrees
- elevation: Meters = 5000
- velocity: MetersPerSecond = 500
- vector: Degrees
- initialDelay: Seconds = 0
- type: String = "Plane"
- id: Integer = 0
- isActive: Boolean = FALSE

Statechart to emulate track through radar coverage
Built-in Self Test

• Check the state of the system
• Tagged as integration support
• Usually not in final product
• Types of checking
  – Assertions
  – Preconditions
  – Postconditions
  – Invariant
Invariants

• An expression that should hold true at all times during execution
  – Checked periodically during execution
  – Used to automatically detect faulty internal states during execution

• Express safety properties as invariant
• The scalability of a system can be verified through stress testing targeting:
  – Number of instances
  – Number of events
  – Application-specific throughput
  – Degraded and error conditions
Performance

• Throughput
• Response time
• Real-time constraints
• Memory usage
• Tailor instrumentation
  – Collect required metrics
  – Minimize impact on execution
Coverage

- Use models to help measure test coverage
- Use trace logs to determine coverage
- Non-executed states may be unreachable or point to holes in the test suite
- Operation coverage
- Path Coverage
• Critical for embedded systems of any complexity
• Repeatability
• Test harness
• Allow for interactive debug when problems are found
• Validation of test results
  – Visual inspection
  – Result prediction
Conclusion

• Dynamic testing of models is required
• Test \{ Model + Transformation + Platform \}
• Instrumentation increases controllability, observability, and testability
• Used during integration testing
  – In development environment
  – On target platforms
• Translation generic instrumentation test harness applicable to any application
Thank You!