MODEL-BASED SOFTWARE HEALTH MANAGEMENT
NASA Cooperative Agreement NNX08AY49A

Abhishek Dubey

Gabor Karsai

Robert Kereskenyi

Nag Mahadevan

Institute for Software-Integrated Systems, Vanderbilt University
Recap

• Model-based Software Health Management:
  – Model-based approach to constructing Software Health Management solutions

• Year 1 – Technical focus:
  – Component-level Health Management
  – Prototyping and ‘Proof of Concept’ demo
Component Model and Monitoring

- Monitor arriving events
- Monitor incoming calls
- Monitor published events
- Monitor outgoing calls
- Observe state
- Monitor resource usage
- Monitor control flow/triggering
Working hypothesis

The Component Level Health Manager reacts to detected events and takes mitigation actions. It can also report events to higher-level manager/s.

Events: detected by monitoring

Actions:
- Basic mitigation: reset, init, shutdown, destroy
- Intercept related: allow/block call
- Specific mitigation: inject event, call method, deallocate memory, release resource, …
- Event or time-triggered activation

Extension

Consider global system state/mode/objective

Manager’s behavioral model:
- Finite-state machine
- Triggers: monitored events, time
- Actions: mitigation activities

Manager is local to component container (for efficiency) but shall be protected from the faults of functional components.
Notional behavior:
Track component state changes via detected events and progression of time
Take mitigation actions as needed

Design issues:
• Co-location with component (fault containment)
• Local detection may implicate another component
• Mitigation action can include blocking the call
• Mitigation as a complex logic (e.g. try 5 times then switch to a different mode)
## Implementation: Mapping a CM to APEX

### APEX - Abstractions
- Module
- Partition
- Process

### Platform (Linux)
- Host/Processor
- Process
- Thread

### ACM: APEX Component Model

<table>
<thead>
<tr>
<th>Component method</th>
<th>APEX</th>
<th>APEX Concept Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Call-Return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Periodic Target</td>
<td>Co-located</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Non-co-located</td>
<td>N/A</td>
</tr>
<tr>
<td>Sporadic Target</td>
<td>Co-located</td>
<td>Caller method signals callee to release then waits for callee until completion.</td>
</tr>
<tr>
<td></td>
<td>Non-co-located</td>
<td>Caller method sends RMI (via CM) to release callee then waits for RMI to complete.</td>
</tr>
</tbody>
</table>

### Asynchronous Publish-Subscribe

| Periodic Target           | Co-located            | Callee is periodically triggered and polls ‘event buffer’ – validity flag indicates whether data is stale or fresh |
|                           | Non-co-located        | Caller is released when event is available |
| Sporadic Target           | Co-located            | Caller notifies via TCP/IP, callee is released upon receipt |
|                           | Non-co-located        |                                      |

### Invocation

| Synchronous Call-Return   | Perodic Target        | Aperiodic process |
| Asynchronous Publish-Subscribe | Co-located            | Blackboard |
|                           | Non-co-located        | Sampling port, Channel |
| Sporadic Target           | Co-located            | Blackboard, Semaphore, Event |
|                           | Non-co-located        | Queuing port, Semaphore, Event |
### Fault Detection and Mitigation scenarios

#### Partition 1

- **Sensor Component**

#### Partition 2

- **GPS Component**
- **NAVDisplay Component**

<table>
<thead>
<tr>
<th>Fault</th>
<th>Detected at</th>
<th>Fault source</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard deadline violation</td>
<td>GPS Trigger interface</td>
<td>GPS Component</td>
<td>Stop and restart</td>
</tr>
<tr>
<td>Stale data (missing update)</td>
<td>NAVDisplay Subscribe port</td>
<td>GPS Component</td>
<td>Use previous value</td>
</tr>
<tr>
<td>Missing sensor event</td>
<td>GPS Subscribe port</td>
<td>Sensor Component</td>
<td>Use previous value</td>
</tr>
<tr>
<td>Rate of change is too high</td>
<td>NAVDisplay required interface</td>
<td>GPS Component</td>
<td>Use previous value</td>
</tr>
</tbody>
</table>
MAXITERATIONS = 10
CPU=1
HYPERPERIOD = 2
PARTITION_NAME = PART1
PARTITION_NAME = PART2
PART1_EXECUTABLE = ./part1
PART2_EXECUTABLE = ./part2
PART1_SCHEDULE= 0,1
PART2_SCHEDULE=1,1
PART1_SAMPLINGPORT= SENSOR_SAMPLING_PORT
PART2_SAMPLINGPORT= GPS_SAMPLING_PORT

SENSOR_SAMPLING_PORT_MAXMESSAGESIZE= 1024
SENSOR_SAMPLING_PORT_REFRESHPERIOD= 4
SENSOR_SAMPLING_PORT_DIRECTION= SOURCE

GPS_SAMPLING_PORT_MAXMESSAGESIZE= 1024
GPS_SAMPLING_PORT_REFRESHPERIOD= 4
GPS_SAMPLING_PORT_DIRECTION= DESTINATION

CHANNEL_NAME= channel1
channel1_SOURCE = SENSOR_SAMPLING_PORT
channel1_DESTINATION = GPS_SAMPLING_PORT
No Fault Scenario

Part 1 execution

Part 2 execution

Non-Linear Scale
Bad GPS Data

Rate of change of GPS Data is above a threshold

Fault injection

Detection

Mitigation

Non-Linear Scale
GPS Deadline Violation

Fault is injected by changing the execution time of GPS Proc Data In process

Detection

Mitigation

Fault injection

Non-Linear Scale
Missing Sensor Event

Sensor does not publish the output data at the specified rate

Fault injection  Detection  Mitigation

Non-Linear Scale
GPS does not update the timestamp in the event sent to Nav Display

Fault injection
Non-Linear Scale

Detection

Mitigation

Fault injection Detection Mitigation

Non-Linear Scale

Fault injection Detection Mitigation

Non-Linear Scale

Fault injection Detection Mitigation

Non-Linear Scale
Absolute Jitter in the release of Part1
Absolute Jitter in the release of Part2

Drift over time in start of PART2

Drift in micro seconds

Sample. Period=2.0 seconds
Absolute Jitter in the release of Sensor

Drift over time in start of SENSOR_PROC_DATA_OUT

Drift in micro seconds

Sample. Period=4.0 seconds
Absolute Jitter in the release of GPS

Drift over time in start of GPS_PROC_DATA_IN

Drift in micro seconds

Sample. Period=4.0 seconds

-250
-200
-150
-100
-50
0

0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
Lessons learned:
- Developing a Component Framework for APEX is a challenge
  → Hard Real-time Component Framework?
- Significant parts of the CM- and HM-related code can and must be generated from models.
- Component-level HM logic needs to consider non-local effects (problems originating in other components)
Component Monitoring
Specification: CCM/CIDL Style – Events/Interfaces

```c
event Trigger(DateTime when);
component Sampler {
    publishes Trigger out;
}
interface GPSDataSource {
    GPSData read(in int req, out gpsData data);
}
component GPS {
    consumes Trigger in;
    provides GPSDataSource src;
    produces Trigger out;
}
component Display {
    consumes Trigger in;
    requires GPSDataSource rec;
}
assembly Example {
    components Sampler s, GPS gps, Display d;
    connect s.out to gps.in;
    connect gps.out to d.in;
    connect d.rec to gps.src;
}

Monitor publishing events on Sampler’s out port:
intercept s.out(stamp) → call_this_code(stamp);

Monitor arriving events on Display’s in port:
intercept d.in(stamp) → call_this_code(stamp);

Evaluate pre-condition on incoming calls on GPS’s src interface read method:
intercept pre gps.src.read(req, data) → eval_this_expr(req);

Evaluate post-condition on outgoing calls on Display’s rec interface read method
intercept post d.rec(req, data) → eval_this_expr(data);
```
Component Monitoring
Specification: CCM/CIDL Style – State, Resources

event Trigger(DateTime when);
component Sampler {
    publishes Trigger out;
}
interface GPSDataSource {
    GPSData read(in int req, out gpsData data);
    float snRatio;
}
component GPS {
    consumes Trigger in;
    provides GPSDataSource src;
    produces Trigger out;
}
component Display {
    consumes Trigger in;
    requires GPSDataSource rec;
}
assembly Example {
    components Sampler s, GPS gps, Display d;
    connect s.out to gps.in;
    connect gps.out to d.in;
    connect d.rec to gps.src;
}

Monitor the state of GPS’s snRatio: every 5s gps.src.snRatio → call_this_code(value);

Monitor memory allocation in Display: when d allocate(size) → call_this_code(size);

Monitor memory deallocation in GPS: when gps deallocate(block) → call_this_code(block);
Component Monitoring
Expression language

- Expressions are attached to …
  - methods on provided and required interfaces
  - subscribed and published events
  - elapse of time
- Expressions mean …
  - pre-conditions (inputs parameters, state)
  - post-conditions (input and output parameters, state)
  - invariants (state)
- Expressions are constructed from …
  - (current) values of variables
  - rates of variables (change since last evaluation)
  - (limited) history of values of variables
- Expression semantics in …
  - Predicate logic
  - Temporal logic with operators
  - Real-time logic
Next steps

• **Immediate:**
  – **Platform:**
    • Specification language for condition monitoring
    • Extensions to the (CCM) IDL compiler and generator
    • Finalize ARINC emulator
  – **Model-based health manager:**
    • Modeling language/tool for component architecture, monitors, and health manager behavior
    • Generator for component-level health manager
    • Continuous integration, testing, and evaluation

• **Year 2:**
  – Develop modeling approach for cascading faults
  – Design and develop system level health-manager
  – Prototype and evaluate approach