MODEL-BASED SOFTWARE HEALTH MANAGEMENT
NASA Cooperative Agreement NNX08AY49A

Abhishek Dubey
Gabor Karsai
Robert Kereskenyi
Nag Mahadevan
Institute for Software-Integrated Systems, Vanderbilt University
Topics

• Objective of Research
• Hypothesis
• Progress to-date
  – Component-level Health Monitor
  – Modeling language
  – System-level Health Manager
  – Testbed, experiment, demo
• Next steps
Objective of Research

To develop solutions for software health management using model-based software development techniques

Goals
1. Review state-of-the-art in software health management
2. Develop a comprehensive approach to modeling and managing faults in software in the context of the system it is applied
3. Construct (a) a prototype modeling tool, and (b) a prototype multi-layered software health management engine that implements the principles involved
4. Demonstrate the feasibility of the approach through experiments using realistic problems, systems, and scenarios.

Perspective:
Model-based software tools that assist software developers in building complex airborne software that is robust against systemic faults – have positive impact on safety and supportability.
Hypothesis

Software is a complex engineering artifact that can have latent faults, uncaught by testing and verification. Such faults become apparent during operation when unforeseen modes and/or (system) faults appear.

The hypothesis:

A. A Software Health Management system can detect such faults, isolate their source, prognosticate their progression, and take mitigation actions.

B. Detection, isolation, prognostics, and mitigation shall be engineered in a model-based framework.

The work is directly related to the topic “IVHM 2.4 Software Health Management” of the IVHM Technology Plan.
Progress

• Phase II – Summary
  – Task 1: Extended Modeling Language
  – Task 2: System-level Health Manager
  – Task 3: Testbed, experiment, demo
Recap: Component Health Management

• What are the components?
  – Component model

• How do we monitor them?
  – Monitor interfaces

• How do we detect anomalies?
  – Based on specs for anomalous/correct behavior

• How do we mitigate?
  – React to detection events / current state of health manager

• How do we implement all these?
  – Use a modeling language + generate code as needed
A component is a unit (containing potentially many objects). The component is parameterized, has state, it consumes resources, publishes and subscribes to events, provides interfaces and requires interfaces from other components.

Publish/Subscribe: Event-driven, asynchronous communication

Required/Provided: Synchronous communication using call/return semantics. Triggering can be periodic or sporadic.
Recap: Component Interactions

Components can interact via asynchronous/event-triggered and synchronous/call-driven connections.

Example: The *Sampler* component is triggered periodically and it publishes an event upon each activation. The *GPS* component subscribes to this event and is triggered sporadically to obtain GPS data from the receiver, and when ready it publishes its own output event. The *Display* component is triggered sporadically via this event and it uses a required interface to retrieve the position data from the *GPS* component.
Recap: Component Monitoring

**Purpose:** to detect anomalies in the context of components.

Component monitoring must happen on the ‘right’ level of abstraction such that meaningful conditions for the component’s health could be formulated. Low-level, packet-oriented monitoring is ineffective.

Monitor events: trigger on the appearance of the event, access data associated with the event (if any)

Monitor interfaces: trigger on the execution of the method call, before and after execution of the method, access call parameters

Observe state: query values of state variables.

Monitor resource usage: keep track of dynamic and stack memory usage, and keep track of generic resource allocation/de-allocation operations (including timing)

Monitor control flow/triggering: detect invocation and return, keep track of timing (execution time, invocation frequency)

*Goal:* Minimally intrusive monitoring – as little overhead as possible.
Recap: Component Monitoring

- Monitor arriving events
- Monitor incoming calls
- Monitor published events
- Monitor outgoing calls
- Observe state
- Monitor control flow/triggering
- Monitor resource usage
Phase I component monitoring:

- Monitor method execution time
- Monitor pre- and post-condition
- Monitor code execution for exceptions

Conditions are defined over *values* and *rates* of method parameters and component state variables

→ Insufficient to determine whether the component is *invoked correctly* or whether it is *in a correct state*.

Example: Component c supports operations x(), y(), z() and the legal sequences of these operations are expressed using the following regular expression: ( x; y+; z; )*  
→ Pre/post conditions are insufficient
Phase II component monitoring:

- Track component state with an observer automata
- The observer automata can also captures the fault mitigation logic

Example: ‘File’ component

Legal sequences of operations start with an open(), followed by arbitrary number of read() or write() operations, followed by a close().

Observer automata:
- States: nominal and mitigation states
- Transitions: triggered by operations, guarded by conditions
- Actions: Health management actions

Anomalies detected:
- Incorrect state of component
  - Cause: component itself
- Incorrect usage
  - Cause: ‘caller’ component
# Health Mitigation Actions

<table>
<thead>
<tr>
<th>HM Action</th>
<th>Semantics</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGNORE</td>
<td>Continue as if nothing has happened</td>
<td>Component</td>
</tr>
<tr>
<td>ABORT</td>
<td>Discontinue current operation, but operation can run again</td>
<td>Component</td>
</tr>
<tr>
<td>USE_PAST</td>
<td>Use most recent data (only for operations that expect fresh data)</td>
<td>Component</td>
</tr>
<tr>
<td>STOP</td>
<td>Discontinue current operation</td>
<td>Component, System</td>
</tr>
<tr>
<td></td>
<td>Aperiodic methods: operation can run again</td>
<td>Component</td>
</tr>
<tr>
<td></td>
<td>Periodic operations: operation must be enabled by a future START HM action</td>
<td>System</td>
</tr>
<tr>
<td>START</td>
<td>Re-enable a STOP-ped periodic operation</td>
<td>Component, System</td>
</tr>
<tr>
<td>RESET</td>
<td>Stop all operations, initialize state of component, clear all queues, start all periodic operations</td>
<td>System</td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>Save component state</td>
<td>System</td>
</tr>
<tr>
<td>RESTORE</td>
<td>Restore component state state to the last saved state</td>
<td>System</td>
</tr>
</tbody>
</table>
Recap: Modeling Language

- **Modeling elements:**
  - Data types: primitive, structs, vectors
  - Interfaces: methods with arguments
  - Components:
    - Publish/Subscribe ports (with data type)
    - Provided/Required interfaces (with i/f type)
    - Health Manager
- **Assemblies**
- **Deployment**
  - Modules, Partitions
  - Component → Partition
Modeling Language - Summary
Modeling Language: Extensions to Component Monitoring

- **Component Health Manager**
  - HM: Local health manager logic
  - HM Publisher: to report component health
  - HM Consumer: to receive system-level health management commands

- **Local health management logic:**
  - Events and actions (responses)
  - States: Observer + Fault Manager

- **Observer:**
  - Tracks component state

- **Fault Manager:**
  - Local mitigation actions

- **State Machine paradigm:**
  - Statechart with timeouts

*The Health Manager: Sophisticated Monitoring and decision making logic*
Recap: System-level Health Management

• Focus issue: Cascading faults
  – Hypothesis: Fault effects cascade via component interactions
  – Anomalies detected on the component level are not isolated → can be caused by other components

• Problem:
  – How to model fault cascades?
  – How to diagnose and isolate fault cascade root causes?
  – How to mitigate fault cascades?
Recap: Modeling Cascading Faults

- The cascades are automatically computed from the component assemblies, if the anomaly types and their interactions are known.
- Sources:
  - Component internal data- and control flows
  - Component Assembly Model
System-level Fault Mitigation

- Model-based system-level mitigation engine
  - Model-based diagnoser is automatically generated
  - Designer specifies fault mitigation strategies using a reactive state machine

Advantages:
- Models are higher-level programs to specify (potentially complex) behavior – more readable and comprehensible
- Models lend themselves to formal analysis – e.g. model checking
System-level Health Management
Functional components

1. Aggregator:
   - Integrates (collates) health information coming from components (typically in one hyperperiod)
2. Diagnoser:
   - Performs fault diagnosis, based on the fault propagation graph model
   - Ranks hypotheses
   - Component that appears in all hypotheses is chosen for mitigation (TBD: OK?)
3. Response Engine:
   - Issues mitigation actions to components based on diagnosis results
   - Based on a state machine model that maps diagnostic results to mitigation actions

Currently these are modeled as components, but they can be generated automatically

The Health Management Approach:
1. Locally detected anomalies are mitigated locally first. – Quick reactive response.
2. Anomalies and local mitigation actions are reported to the system level.
3. Aggregated reports are subjected to diagnosis, potentially followed by a system-level mitigation action.
4. System-level response commands are propagated to components.
## Recap: Implementation: Mapping a CM to APEX

<table>
<thead>
<tr>
<th>ACM: APEX Component Model</th>
<th>APEX</th>
<th>APEX Concept Used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component method</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Periodic</td>
<td>Periodic process</td>
<td>Process start, stop</td>
</tr>
<tr>
<td>Sporadic</td>
<td>Aperiodic process</td>
<td>Semaphores</td>
</tr>
<tr>
<td><strong>Invocation</strong></td>
<td></td>
<td></td>
</tr>
<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>
<tr>
<td>Asynchronous Publish-Subscribe</td>
<td>Periodic Target</td>
<td>Co-located</td>
</tr>
<tr>
<td></td>
<td>Non-co-located</td>
<td>Callee is released when event is available</td>
</tr>
<tr>
<td>Sporadic Target</td>
<td>Co-located</td>
<td>Caller notifies via TCP/IP, callee is released upon receipt</td>
</tr>
<tr>
<td></td>
<td>Non-co-located</td>
<td>Queuing port, Semaphore, Event</td>
</tr>
</tbody>
</table>

### APEX - Abstractions

<table>
<thead>
<tr>
<th>Module</th>
<th>Platform (Linux)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>Host/Processor</td>
</tr>
<tr>
<td>Partition</td>
<td>Process</td>
</tr>
<tr>
<td>Process</td>
<td>Thread</td>
</tr>
</tbody>
</table>
Recap: Implementation

• Platform:
  – ARINC-653 Emulator on Linux
  – MICO (open source CORBA)
  – Module manager, infrastructure

• Code generator
  – Produces ‘glue code’ for the component framework
  – Compiles monitoring expressions
  – Builds code for CHM

Designer supplies functional code
Implementation

• New developments on platform:
  – Extensions to handle communications
    • Module: one processor
    • Network: TCP/UDP based network
    • Communications:
      – Time-triggered messaging (close to TTP/A), no fault tolerance
      – Messages implement synchronous calls and asynchronous publish-subscribe messaging
      – Messages are schedule in-frame (best effort) or at frame-end (timed)

  – Code generators produce implementation code for
    • Observer and health manager state machine
    • System-level aggregator + diagnoser + response engine
Status: System-level HM

- **Modeling**: Extended modeling language defined and modeling tool configured
- **Component-level health manager**: Generator and run-time code for observer automata developed
- **System-level diagnostic model generation**: Prototype generator for producing TFP graph models is functional
- **System-level aggregator, diagnoser, and response engine**: Prototype is functional
- **Integration**: Hand-crafted prototype is functional
  - Fully automated integration - in progress
- **Examples**: 11 Single Processor and 8 Multiprocessor scenarios modeled
  - Boeing Bold Stroke examples from DARPA/MoBIES
DEMO
2005 Malaysian Air Boeing 777 in-flight upset

- Low airspeed advisory.
- Airplane’s autopilot experienced excessive acceleration values.
  - Vertical acceleration decreased to -2.3g within ½ second
  - Lateral acceleration decreased to -1.01g (left) within ½ second
  - Longitudinal acceleration increases to +1.2 g within ½ second
- Autopilot pitched nose-up to 17.6 degree and climbed at a vertical speed of 10,650 fpm.
- Airspeed reduced to 241 knots.
- Stick shaker activated at top of the climb.
- Aircraft descended 4,000 ft.
- Re-engagement of autopilot followed by another climb of 2,000 ft.
- Maximum rate of climb = 4440 fpm.
B777 ADIRU Architecture

- Designed to be serviceable with one fault in each FCA
- Can fly but maintenance required upon landing with two faults in each FCA
- Each ARINC 629 end unit voted on the processor data bit-by-bit.
- Processors monitor the ARINC 629 modules by full data wrap-around
- Processors also monitor the power supplies, any one of which can power the entire unit
- Accelerometer and gyro in skewed redundant configuration
- A S(secondary)AARU also provided inertial data

Based on Air Data Inertial Reference Unit (ADIRU) Architecture (ATSB, 2007, p.5)
Cause of Inflight Upset

• June 2001: accelerometer 5 fails with high output value, ADIRU disregards it.
• A power cycle on ADIRU occurs. A latent software bug disregards the faulty status of accelerometer 5.
  – Status of failed unit was recorded on-board maintenance memory, but that memory was not checked by the software.
• An inflight fault was recorded in accelerometer 6 and it was disregarded.
• FDI software allowed use of accelerometer 5.
• High acceleration value was passed to all computers.
• Due to common-mode nature of fault, voters allowed high accelerometer data to go on all channels.
• This high value was used by primary flight computer.
• Mid value select function used by the flight computer lessened the effect of pitch motion.

Problem: System relied on redundancy to mask a fault. But due to latent software bug and common-mode fault, the effect cascaded into the system failure.

Reading Material: The dangers of failure masking in fault-tolerant software: aspects of a recent in-flight upset event
• Modeled the architecture as a software component assembly
• Created the fault scenario
• Only modeled part of the system to illustrate the point of SHM
Component: Accelerometer
ADIRU Computer Observer

- Only shown for 1 accelerometer. But there 5 parallel machines
Component: Display (Mimics PFC)
Deployment Model
Next steps

• System-level Health Manager
  – Finalize SHM toolsuite
  – Integrate and test

• Health management for systemic faults
  – Develop modeling approach for system goals (functions)
  – Develop extended version for SHM
  – Experiment/demonstration
Publications

During the past 6 months:


Software:

• ACM Tool suite (ARINC Component Model tools including: modeling tool, generators, run-time platform – ARINC-653 emulator, documentation, experimental sample, etc.)

[Link](https://wiki.isis.vanderbilt.edu/mbshm/index.php/ACMTOOLSUITE)
DEMONSTRATION - SNAPSHOT SPORTS
Component Assembly
Accelerometer Machine – durip02

ADIRU Computers Machine – durip03

SHM Machine – durip09

Voter + Display Computer Machine – durip06
**Accelerometer Module**

```
1 //////////////////////////////////////////////////////////////////////////////////////////
2 SYNCHRONIZED_EXECUTION=TRUE
3 CONFIGURATION_PORT=4211
4 BROADCAST_ADDRESS=192.168.1.255
5 MODULE_NAME=AccelerometerModule
6 PARENT_MODULE_NAME=SHMModule
7 SHMModule_HOSTNAME=durip09
8 //////////////////////////////////////////////////////////////////////////////////////////
9 DISCOVERY_PROTOCOL = IOR
10 MAXITERTATIONS = 10
11 CPU=1
12 NUMBER_OF_INIT_STAGES = 2
13 HYPERPERIOD = 1.0
14 PARTITION_INIT_TIMEOUT = 60
15 PARTITION_NAME = Partition1
16 PARTITION_NAME = Partition2
17 PARTITION_NAME = Partition3
18 PARTITION_NAME = Partition4
19 PARTITION_NAME = Partition6
20 PARTITION_NAME = Partition5
21 Partition1_EXECUTABLE = ./Partition1/Partition1
22 Partition2_EXECUTABLE = ./Partition2/Partition2
23 Partition3_EXECUTABLE = ./Partition3/Partition3
24 Partition4_EXECUTABLE = ./Partition4/Partition4
25 Partition5_EXECUTABLE = ./Partition5/Partition5
26 Partition6_EXECUTABLE = ./Partition6/Partition6
27 //////////////////////////////////////////////////////////////////////////////////////////
28 PARTITION1_SCHEDULE = 0, 0.16
29 Part12_SCHEDULE = 0.16, 0.16
30 Part13_SCHEDULE = 0.32, 0.16
31 Part14_SCHEDULE = 0.49, 0.16
32 Part15_SCHEDULE = 0.66, 0.16
33 Part16_SCHEDULE = 0.83, 0.16
34 //////////////////////////////////////////////////////////////////////////////////////////
```

**ADIRU Module**

```
1 //////////////////////////////////////////////////////////////////////////////////////////
2 SYNCHRONIZED_EXECUTION=TRUE
3 CONFIGURATION_PORT=4211
4 BROADCAST_ADDRESS=192.168.1.255
5 MODULE_NAME=ADIRUComputer
6 PARENT_MODULE_NAME=SHMModule
7 SHMModule_HOSTNAME=durip09
8 //////////////////////////////////////////////////////////////////////////////////////////
9 DISCOVERY_PROTOCOL = IOR
10 MAXITERTATIONS = 10
11 CPU=1
12 NUMBER_OF_INIT_STAGES = 2
13 HYPERPERIOD = 1.0
14 PARTITION_INIT_TIMEOUT = 60
15 PARTITION_NAME = Part1
16 PARTITION_NAME = Part2
17 PARTITION_NAME = Part3
18 PARTITION_NAME = Part4
19 Part1 EXECUTABLE = ./Part1/Part1
20 Part2 EXECUTABLE = ./Part2/Part2
21 Part3 EXECUTABLE = ./Part3/Part3
22 Part4 EXECUTABLE = ./Part4/Part4
23 //////////////////////////////////////////////////////////////////////////////////////////
24 Part1_SCHEDULE = 0, 0.24
25 Part2_SCHEDULE = 0.25, 0.24
26 Part3_SCHEDULE = 0.50, 0.24
27 Part4_SCHEDULE = 0.75, 0.24
28 //////////////////////////////////////////////////////////////////////////////////////////
```
Configuration Files

- **SHM Module**

```plaintext
SYNCHRONIZED_EXECUTION=TRUE
CONFIGURATION_PORT=4211
BROADCAST_ADDRESS=192.168.1.255
MODULE_NAME=SHMModule
CHILD_MODULE_NAME=AccelerometerModule
CHILD_MODULE_NAME=ADIRUComputer
CHILD_MODULE_NAME=VoterModule
AccelerometerModule_HOSTNAME=durip02
ADIRUComputer_HOSTNAME=durip03
VoterModule_HOSTNAME=durip06

DISCOVERY_PROTOCOL = IOR
MAXITERATIONS = 20
CPU=1
NUMBER_OF_INIT_STAGES = 2
HYPERPERIOD = 1.0
PARTITION_INIT_TIMEOUT = 60
PARTITION_NAME = SHMPartition
SHMPartition_EXECUTABLE = ./SHMPartition/SHMPartition

SCHEDULING INFORMATION BEGINS
SHMPartition_SCHEDULE = 0, 0.1
SCHEDULING INFORMATION ENDS
```

- **Voter + Display Module**

```plaintext
SYNCHRONIZED_EXECUTION=TRUE
CONFIGURATION_PORT=4211
BROADCAST_ADDRESS=192.168.1.255
MODULE_NAME=VoterModule
PARENT_MODULE_NAME=SHMModule
SHMModule_HOSTNAME=durip09

DISCOVERY_PROTOCOL = IOR
MAXITERATIONS = 10
CPU=1
NUMBER_OF_INIT_STAGES = 2
HYPERPERIOD = 1.0
PARTITION_INIT_TIMEOUT = 60
PARTITION_NAME = Left
PARTITION_NAME = Center
PARTITION_NAME = Right
Left_EXECUTABLE = ./Left/Left
Center_EXECUTABLE = ./Center/Center
Right_EXECUTABLE = ./Right/Right

SCHEDULING INFORMATION BEGINS
Left_SCHEDULE = 0, 0.33
Center_SCHEDULE = 0.33, 0.33
Right_SCHEDULE = 0.66, 0.33
SCHEDULING INFORMATION ENDS
```
Multi-Module – ADIRU

Fault Detected in Accelerometer &
Diagnosis in SHM

Partition4|HME|POSTCHECK - cond1 - IN Accelerometer4::Acceleration
FAILED :
Partition4||HME|RAISE ERROR: (CHM sends Ignore)
Partition4||APP|Accelerometer4_impl::APEX_MANAGE_HEALTH Error Received

AM_Accelerometer4_Acceleration_POSTCONDITION_FAILURE

Results at time = 10.3426
Hypothesis 0 Failure Modes = [ FM_Accelerometer4_Acceleration_POSTCONDITION ]
Hypothesis 1 Failure Modes = [ FM_Accelerometer4_Acceleration_USER_CODE ]
Multi-Module – ADIRU
Fault Detected in Display Computers & Diagnosis in SHM

Voter + Display Computer

Center|HME|PRECHECK-cond1-IN DisplayComponentC::Cc FAILED

Center|HME|RAISE ERROR: DisplayComponentC_Cc+Abort from chn

SHMPartition|DiagnosisEngine setting alarm AM_DisplayComponentR_Cc_PRECONDITION_FAILURE

Hypothesis 0 Failure Modes = [ FM_Accelerometer4_Acceleration_POSTCONDITION ]

Hypothesis 1 Failure Modes = [ FM_Accelerometer4_Acceleration_USER_CODE ]

SHMPartition|DiagnosisEngine : Faulty Component - ID 8 Name Accelerometer4
SHM Action based on Diagnosis Results

Multi-Module – ADIRU

SHMPartition | CompBehavior – Action – RESETALL Followed by the STOPALL (ID=2)
Component – ACCELEROMETER4 (ID=8)

Component – ACCELEROMETER4

ID=8
Multi-Module – ADIRU

SHM command in Accelerometer

26901:Partition4|1289927899.846563938|APP|Accelerometer4 :: consuming HMConsumer

26901:Partition4|1289927899.846569912|HME|Received STOPALL.

Setting StopSystem=true

26901:Partition4|1289927900.845982663|APP|PUBLISH: Accelerometer4_Acceleration

26901:Partition4|1289927900.846000895|APP|Accelerometer4 :: publishing Acceleration

Multi-Module – ADIRU
After Reconfig

Observer Detects that Accelerometer 4 is not publishing and stops using it.
Multi-Module – ADIRU
After Reconfig

Acceleration Estimates from Voter return to acceptable values
Voter before Reconfig

```
11300:Center|1289945338.391624797|TRC|START_PROCESS: COMP_VoterCenter_VotingTask
11300:Center|1289945338.391637292|DEB|successcount is 1
11300:Center|1289945338.391640918|DEB|successcount is 2
11300:Center|1289945338.391645538|INF|ID of PROCESS NAME COMP_VoterCenter_VotingTask is 32
11300:Center|1289945338.391649509|APP|METHOD : COMP_VoterCenter_VotingTask
11300:Center|1289945338.391674442|ID of PROCESS NAME COMP_VoterCenter_VotingTask is 32
11300:Center|1289945338.391682876|HME|HME Not available for process id 32
11300:Center|1289945338.391693248|APP|ComputationTask: BodyAccelerationValue a1: ax=21.8397
81, ay=-206.113922, az=-172.504044
11300:Center|1289945338.391703433|APP|ComputationTask: BodyAccelerationValue a2: ax=21.8397
81, ay=-206.113922, az=-172.504044
11300:Center|1289945338.391717360|APP|ComputationTask: BodyAccelerationValue a3: ax=3.29472
0, ay=3.857370, az=0.177870
11300:Center|1289945338.391732247|APP|ComputationTask: BodyAccelerationValue a4: ax=21.8397
81, ay=-206.113922, az=-172.504044
11300:Center|1289945338.391744247|APP|Voted value: x21.839781, y:-206.113922, z:-172.504044
11300:Center|1289945338.391757503|INF|size of new length 36
11300:Center|1289945338.391762640|INF|*****Calling timed send on Center_QP_4*******
11300:Center|1289945338.391771516|INF|*****Result of timed send on Center_QP_4 0 Success***
******
11300:Center|1289945338.391775937|INF|size of new length 36
11300:Center|1289945338.391780178|INF|*****Calling timed send on Center_BUF_0*******
11300:Center|1289945338.391789594|INF|*****Result of timed send on Center_BUF_0 0 Success**
******
11300:Center|1289945338.391798411|INF|ID of PROCESS NAME COMP_VoterCenter_VotingTask is 32
11300:Center|1289945338.391802580|INF|HME Not available for process id 32
11300:Center|1289945338.391807227|DEB|Setting Event RESPONSE_32
11300:Center|1289945338.391810437|TRC|END_PROCESS: COMP_VoterCenter_VotingTask
11300:Center|1289945338.391821041|TRC|STARTED PARSE_AND_VALIDATE_DATA
11300:Center|1289945338.391825117|TRC|COPIED MESSAGE
```