A Real-Time Component Framework: Experience with CCM and ARINC-653

Abhishek Dubey, Gabor Karsai, Robert Kereskenyi, Nagabhusan Mahadevan
ISIS/Vanderbilt University
Outline

- Software components for real-time systems
- ARINC-653 features
- CCM features
- The ARINC Component Model
- Experience
- Next steps
Notional Design Flow for High-Confidence Software Systems

What is a ‘component’?
Hard Real-time Components?

Need:
- A Component Model suitable for hard real-time systems that codifies all component interactions and allows specification of timing requirements

Real-time CORBA?
- QoS and scheduling attributes on CCM

MARTE UML Profile?
- Specifications for timing properties in UML models

AUTOSAR?
- No component execution model
ARINC-653/APEX: Partitioning Kernel API

- Partitions:
  - Spatial and Temporal separation of activities – Fault isolation!
  - Partition memory size and temporal duration are fixed
- Within a partition (shared address space)
  - Multiple processes (static); periodic/aperiodic, with opt deadline
  - Primitives for process interactions: buffers and blackboards, semaphores and events
  - Health monitor (to restart processes)
- Across partitions (isolated address spaces)
  - Fixed allotment of CPU time
  - Message-based interactions via channels connecting sampling and queuing ports
- Multiple processors (‘modules’) – few details
**CORBA Component Model**

- **Components**
  - Generalized ‘objects’ with state
  - Synchronous (call/return) interactions via provided/required interfaces
  - Asynchronous (publish/subscribe) interactions via publish/subscribe interfaces

- **Component homes**
  - Lifecycle and resource management for components
ACM:
The ARINC Component Model

- Provide a CCM-like layer on top of ARINC-653 abstractions
- Notional model:

- Terminology:
  - Synchronous: call/return
  - Asynchronous: publish-return/trigger-process
  - Periodic: time-triggered
  - Aperiodic: event-triggered
ACM:
The ARINC Component Model

- Each ‘input interface’ has its own process
  - Process must obtain read-write/lock on component
- Asynchronous publisher (subscriber) interface:
  - Listener (publisher) process
  - Pushes (receives) one event (a struct), with a validity flag
  - Can be event-triggered or time-triggered (i.e. 4 variations)
- Synchronous provided (required) interface:
  - Handles incoming synchronous RMI call
  - Forwards outgoing synchronous RMI call
- Other interfaces:
  - State: to observe component state variables
  - Resource: to monitor resource usage
  - Trigger: to monitor execution timing
ACM: The ARINC Component Model

Mapping the CCM concepts to APEX in ACM

<table>
<thead>
<tr>
<th>CCM</th>
<th>Target Properties</th>
<th>Features of ACM Implementation</th>
<th>APEX API Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host /Processor</td>
<td>N/A</td>
<td>An Apex module, mapped to a single CPU core.</td>
<td>Module</td>
</tr>
<tr>
<td>ORB Instance</td>
<td>N/A</td>
<td>An Apex partition, mapped to an OS Process.</td>
<td>Partition</td>
</tr>
<tr>
<td>Component Class</td>
<td>N/A</td>
<td>Data structure shared by related ARINC processes.</td>
<td>Semaphores</td>
</tr>
<tr>
<td>Component method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous RMI</td>
<td>Periodic</td>
<td>Periodic process, mapped to an OS Thread</td>
<td>Process start, stop</td>
</tr>
<tr>
<td>Synchronous RMI</td>
<td>Non-colocated</td>
<td>Aperiodic process, mapped to an OS Thread</td>
<td>Aperiodic</td>
</tr>
<tr>
<td>Synchronous RMI</td>
<td></td>
<td>Caller method sends RMI to release callee then waits for completion.</td>
<td>TCP/IP, Event</td>
</tr>
<tr>
<td>Synchronous RMI</td>
<td>Colocated</td>
<td>Caller method signals callee to release then waits for callee until completion.</td>
<td>Semaphore, Event</td>
</tr>
<tr>
<td>Asynchronous Publish-Subscribe</td>
<td>Periodic</td>
<td>Caller is periodically triggered and polls event buffer (Blackboard) - validity flag indicates whether data is stale or fresh</td>
<td>Blackboard</td>
</tr>
<tr>
<td>Asynchronous Publish-Subscribe</td>
<td>Non-colocated</td>
<td>Calllee is periodically triggered and polls “Sampling Port” - validity flag indicates whether data is stale or fresh</td>
<td>Sampling port, Channel</td>
</tr>
<tr>
<td>Asynchronous Publish-Subscribe</td>
<td>Colocated</td>
<td>Calllee is released when event is available</td>
<td>Blackboard, Semaphore, Event</td>
</tr>
<tr>
<td>Asynchronous Publish-Subscribe</td>
<td>Non-colocated</td>
<td>Calllee notifies via TCP/IP, calllee is released upon receipt</td>
<td>Blackboard, Semaphore, Event</td>
</tr>
</tbody>
</table>

Observe:

- All component interactions are realized via the framework
- Process (method) execution time has deadline, which is monitored
ACM: A Prototype Implementation

Background: Project on Model-based Software Health Management

- How to build ‘software health management functions’ into systems that monitor, diagnose, and mitigate software defects at run-time?

Concept

- Use model-based fault diagnostics techniques for monitoring and diagnosis
- Use model-based software development techniques to design, analyze, and generate the code for the software health management function
ACM:
A Prototype Implementation

- ARINC-653 Emulator
  - Emulates APEX services using Linux API-s
  - Partition → Process, Process → Thread
  - Module manager: schedules partition set
  - Partition level scheduler: schedules threads within partition

- CORBA foundation
  - MICO CCM ORB
  - No modifications
  - CLHM: Component-level Health Manager
ACM: Model-based Development

- Graphical models represent:
  - Data types, interfaces
  - Components (with interfaces and state variables)
  - Component assemblies
  - Component to partition allocation
ACM: Model-based Development

- Graphical models are used to generate ‘infrastructure’ code.
ACM: Example

- Simple, 3 component system

<table>
<thead>
<tr>
<th>Part -tion</th>
<th>Process Name</th>
<th>Period</th>
<th>Dead -line</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td>Part1 ORB Process</td>
<td>Aperiodic</td>
<td>Infinite</td>
<td>SOFT</td>
</tr>
<tr>
<td>Part 1</td>
<td>Sensor.DataOut</td>
<td>4sec</td>
<td>4sec</td>
<td>HARD</td>
</tr>
<tr>
<td>Part 2</td>
<td>Part2 ORB Process</td>
<td>Aperiodic</td>
<td>Infinite</td>
<td>SOFT</td>
</tr>
<tr>
<td>Part 2</td>
<td>GPS.DataIn</td>
<td>4sec</td>
<td>4sec</td>
<td>HARD</td>
</tr>
<tr>
<td>Part 2</td>
<td>GPS.GetGPSData</td>
<td>Aperiodic</td>
<td>4sec</td>
<td>HARD</td>
</tr>
<tr>
<td>Part 2</td>
<td>GPS HM</td>
<td>Aperiodic</td>
<td>Infinite</td>
<td>SOFT</td>
</tr>
<tr>
<td>Part 2</td>
<td>Navigation HM</td>
<td>Aperiodic</td>
<td>Infinite</td>
<td>SOFT</td>
</tr>
</tbody>
</table>

ARINC Processes created by the framework for the GPS example.

<table>
<thead>
<tr>
<th>Process</th>
<th>Std (µs)</th>
<th>Mean (µs)</th>
<th>Max (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td>2.26</td>
<td>68.29</td>
<td>72.47</td>
</tr>
<tr>
<td>Part 2</td>
<td>2.69</td>
<td>3.49</td>
<td>7.39</td>
</tr>
<tr>
<td>GPS.DataIn</td>
<td>0.96</td>
<td>228.16</td>
<td>229.12</td>
</tr>
<tr>
<td>Sensor.DataOut</td>
<td>0.74</td>
<td>153</td>
<td>153.77</td>
</tr>
</tbody>
</table>
ACM: Lessons Learned

- Two worlds: The highly dynamic CCM and the strictly static ARINC do not mesh well
- Allocating a thread to every method is probably a waste of resources
- For analyzability a deeper modeling of component behavior is desired
Summary

- ACM: Steps towards a hard real-time component model
  - CCM: provides the essential component abstraction
  - ARINC: provides the API / platform

- Model-based configuration and code generation helps

- ACM is an experiment – work in progress

- Further research is needed to address the lessons learned