Towards a Unified Component & Deployment Model for Distributed Real-Time Systems

Gerardo Pardo-Castellote, Ph.D.  
CTO, Real-Time Innovations  
Co-Chair OMG DDS SIG

Sumant Tambe, Ph.D.  
Research Engineer  
Real-Time Innovations

Johnny Willemsen  
CTO, Remedy IT
Goals

• Review the state of affairs
• Identify basic requirements
• Share the vision
• Take a step towards an OMG “Unified Component Model”
Terminology

• Component model:
  – “Defines a set of standards for component implementation, naming, interoperability, customization, composition, evolution and deployment”
    [ Heineman & Councill ]

• Component:
  – “A software element, conforming to a component model, which can be independently deployed and composed without modification”

• Component-Based System:
  – Set of components + Platform + Binding Mechanism
State of affairs

• Many component frameworks exist
  – OMG has multiple: CCM, LwCCM, RTC, SDR, UML2, SysML, ...
  – Enterprise systems: EJB, SCA, OSGI, COM
  – Industry Specific: AUTOSAR
  – Tool-specific: Simulink, LabView, ControlShell

• However, no clear choice for general distributed real-time systems
  – Existing real-time component frameworks fall short
    • the desirable level of abstraction
    • Insufficient separation of concerns
    • Tools, market adoption, ...
Component model taxonomy

- A comprehensive survey of component models exists

Design Space for Component Models

- Lifecycle management
  - Support in Comp Model for all stages from the component specification to integration into the system
- Construction
  - Mechanisms for building systems: Functional/Interface specification + Connections/Bindings + Communications/Interactions between components
- Non-functional properties
  - Qos, Performance, ...
- Scope: General purpose (e.g. EJB/CORBA) vs. Specialized (e.g. Autosar)
- Platform: Container, Java, CORBA, .NET, Protocol Standards
Component Lifecycle Management

- Modeling
- Implementation
- Packaging
- Deployment
- Execution
Component-based System Construction

- Component Interfaces
  - How to define connection end-points?
- Component bindings
  - Binding between component and platform
  - Binding between components
- Interaction patterns
  - What interaction patterns should the components support?
Design space for Component Interfaces

- **Interfaces**
  - Port-based
    - Data-types and events
  - Operational
    - Like in IDL
- **Level of contract specification**
  - Syntactic
    - Basic type system
    - Like in IDL
  - Functional
    - Ranges
    - preconditions and post-condition
  - Behavioral
    - state machines
- **Evolvability – Can interfaces evolve?**
  - Port-based interfaces easier to evolve
    - DDS XTypes specification supports extensible, mutable types
  - Multiple interfaces? Supported, provided, required, attribute access, etc
  - Interface navigation?
  - Interface discovery? Reflection?
Binding is the process that establishes connections between components (composition, wiring)

- **Compensability of components:**
  - Vertical composition: An assembly is also a component
  - Horizontal composition: An assembly cannot be treated as a component

- **Bindings generalized to support various system compositions**
  - Endogenous – components are connected directly to each other: Interaction logic resides within the component
  - Exogenous – components are connected indirectly via a connector object. Interaction logic resides outside the component
Component Interactions

Style / Communication Patterns

- Request-response
  - Synchronous/Asynchronous
- One-way (Pipes and Filter)
  - Event-triggered?
  - Publish/subscribe?
  - Multicast/broadcast?
- Are components passive or active with their own thread?
  - Any constraints on concurrency?
Scope

• General-purpose (e.g., CCM, EJB, JavaBeans, MS COM)
  – Broader applicability
  – Enforces a certain architectural style

• Specialized (e.g., AUTOSAR, Robocop)
  – Applicable to a particular domain
  – Captures architectural tenants of that specific domain
  – Often designed to guarantee a property of the system
    • Verification & validation
    • Real-time and bounded tardiness
    • Schedulability
## Examples

<table>
<thead>
<tr>
<th>CM</th>
<th>Vertical</th>
<th>Exogenous</th>
<th>Implementation</th>
<th>Interaction Style</th>
<th>Interface Type</th>
<th>Interface Language</th>
<th>Standard/Industry Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCM</td>
<td>No</td>
<td>No</td>
<td>Neutral, (C++)</td>
<td>Request-Response Event</td>
<td>Operation Based</td>
<td>OMG IDL</td>
<td>Yes / No</td>
</tr>
<tr>
<td>IwCCM</td>
<td>No</td>
<td>Yes(*)</td>
<td>Neutral (C++)</td>
<td>Extensible: Req-Resp Event, Pub-Sub</td>
<td>Operation Based Port Based</td>
<td>OMG IDL 3.5</td>
<td>Yes / No</td>
</tr>
<tr>
<td>EJB</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>Request-Response</td>
<td>Operation Based</td>
<td>Java + annotation</td>
<td>Yes / No</td>
</tr>
<tr>
<td>OSGI</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>Request-Response, Event</td>
<td>Operation Based</td>
<td>Java</td>
<td>Yes / No</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>Yes, Delegation</td>
<td>Yes, Delegation</td>
<td>C</td>
<td>Request-Response</td>
<td>Operation Based Port Based</td>
<td>C headers</td>
<td>Yes / Yes</td>
</tr>
<tr>
<td>ControlShell</td>
<td>Yes, Delegation</td>
<td>Yes</td>
<td>C++</td>
<td>Pipe &amp; Filter, Event, Request-Reply</td>
<td>Operation Based Port Based</td>
<td>Custom DML</td>
<td>No / No</td>
</tr>
<tr>
<td>Simulink</td>
<td>Yes, Delegation</td>
<td>Yes</td>
<td>C, Matlab</td>
<td>Pipe &amp; Filter</td>
<td>Port based</td>
<td>Graphical</td>
<td>No / No</td>
</tr>
<tr>
<td>LabView</td>
<td>Yes, Delegation</td>
<td>Yes</td>
<td>C, G-Lang.</td>
<td>Pipe &amp; Filter</td>
<td>Port based</td>
<td>Graphical</td>
<td>No / No</td>
</tr>
<tr>
<td>SaveCCM, Rubus</td>
<td>Yes, Delegation</td>
<td>No</td>
<td>C, Java</td>
<td>Pipe &amp; Filter</td>
<td>Port-based</td>
<td>XML-based</td>
<td>No / No</td>
</tr>
</tbody>
</table>
Requirements (my take)

• **Standard** – Obvious reasons
• **Language Neutral**
• **Exogenous** – Separator of connection logic from component
• **Vertical composition** – Assemblies are also components
• **Port-based Interfaces**
• **Richer interaction styles** – Request/Reply, Event, Pub-Sub, Pipe & Filter
• **Not Industry Specific**
## How different models measure against the requirements

<table>
<thead>
<tr>
<th>CM</th>
<th>Vertical</th>
<th>Exogenous</th>
<th>Implementation</th>
<th>Interaction Style</th>
<th>Interface Type</th>
<th>Interface Language</th>
<th>Standard/Industry Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCM</td>
<td>No</td>
<td>No</td>
<td>Neutral, C++</td>
<td>Request-Response Event</td>
<td>Operation Based</td>
<td>OMG IDL</td>
<td>Yes / No</td>
</tr>
<tr>
<td>lwCCM</td>
<td>No</td>
<td>Yes(*)</td>
<td>Neutral (C++)</td>
<td>Extensible: Req-Resp</td>
<td>Operation Based</td>
<td>OMG IDL 3.5</td>
<td>Yes / No</td>
</tr>
<tr>
<td>EJB</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>Request-Response</td>
<td>Operation Based</td>
<td>Java + annotation</td>
<td>Yes / No</td>
</tr>
<tr>
<td>OSGI</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>Request-Response, Event</td>
<td>Operation Based</td>
<td>Java</td>
<td>Yes / No</td>
</tr>
<tr>
<td>AUTOSA</td>
<td>Yes, Delegation</td>
<td>Yes, C</td>
<td>Request-Response</td>
<td>Operation Based</td>
<td>C headers</td>
<td>Yes / Yes</td>
<td></td>
</tr>
<tr>
<td>LabView</td>
<td>Yes, Delegation</td>
<td>Yes, C++</td>
<td>Pipe &amp; Filter, Event, Request-Reply</td>
<td>Operation Based</td>
<td>Port Based</td>
<td>Custom DML</td>
<td>No / No</td>
</tr>
<tr>
<td>Simulink</td>
<td>Yes, Delegation</td>
<td>Yes, C, Matlab</td>
<td>Pipe &amp; Filter</td>
<td>Port Based</td>
<td>Graphical</td>
<td>No / No</td>
<td></td>
</tr>
<tr>
<td>LabView</td>
<td>Yes, Delegation</td>
<td>Yes, C</td>
<td>Pipe &amp; Filter</td>
<td>Port Based</td>
<td>Graphical</td>
<td>No / No</td>
<td></td>
</tr>
<tr>
<td>CCM, Rubus</td>
<td>Yes, Delegation</td>
<td>No, C, Java</td>
<td>Pipe &amp; Filter</td>
<td>Port-based</td>
<td>XML-based</td>
<td>No / No</td>
<td></td>
</tr>
</tbody>
</table>
Current status: CCM4 & lwCCM

• The CCM4 specification is heavy and has CORBA as mandatory middleware
• With DDS4CCM support for DDS as middleware was added
• AMI4CCM introduced the concept of asynchronous invocations
Current status: D&C

- The current D&C specification is limited
- Needs to be extended with several features
  - Plugin support
  - Reconfiguration/redeployment
  - More flexibility
- Our vision is a RFP that asks for a revised specification
What has to happen to our idea?

- Simplify CCM4 to just be LwCCM
- Make CORBA optional instead of mandatory. This can be done through the connector concept
- Move also the event support to a connector
- But keep IDL as the way to define the components but only use local interfaces
What does lwCCM need to become a UCM?

1. Make bindings purely Exogenous
2. Include pipes & filters as interaction style
3. Add vertical composability
4. Abstract and enrich container interaction
5. Enhance container support for passive components
6. Simplify deployment/assembly model
Purely Exogenous

• All component interactions are mediated via connectors
  – Connectors are local interfaces
  – There is no 1\textsuperscript{st} order(*), middleware dependency in the component
    • 2\textsuperscript{nd} order via introspection/narrow can be permitted
• Connectors should be explicit, not just a way to bind components
• lwCCM Extended ports provide a mechanism for this.
  – They must become the *only* ports for the component
Explicit connectors

This

Not this
Explicit connectors

OK
Connector directly visible

OK
Uses name to represent Connector
Include pipes & filters as interaction style

- Unidirectional communication
- Components are like filters that process the data, and the bindings are the pipes that transfer the data to the next filters.
- Allows separate control of the data-flow and control-flow between components.
  - The control flow is activated by a triggering interaction model, which enables the activation of a particular component in response to a particular signal such as an event, a clock tick, or a stimulus from another component
- This interaction model includes event-triggering, or event-driven, and time-triggering.
- Uses:
  - Maps well to control widely used in this domain
  - Streaming of data
Pipes and filters characteristics

- Needed to support Function Block Diagrams
- SysML Internal Block Diagrams
- Control/Dataflow diagrams (e.g. LabView / simulink-like)
Add vertical composability

- Components can be recursively composed of components
- Makes assemblies first-class citizens in component model
- Needed for reuse and packaging
- Part of SysML
- Common in industrially-successful models like Simulink and LabView
Abstract and enrich container interaction

• lwCCM Container interaction depends on CORBA... Should also be local interfaces...
• What additional services (beyond activation) should the container provide?
  — Mode switching?
Enhance container support for passive components

• Should container should also provide periodic execution services for passive components?
• Should container provide ordered execution per dataflow relationships?
• Should container provide threading/mutual exclusion guarantees?
Simplify deployment/assembly model

• Should deployment of components across containers be a separate spec/compliance point
  – Assemblies are components so can deployment on single container be simplified to that of a single component?
  – Can OSGI or similar packaging standards be leveraged to simplify spec?
Conclusion

• There is growing interest on lwCCM
• But current spec is limited and clunky due to history
• Main users and vendors are motivated to significantly simplify & enhance the spec
• It has the potential to offer a UCM. The only one that will be:
  – Standards based
  – Language Independent
  – Not industry specific
  – Sufficiently reach for real-time systems
Thank You
Non-functional properties

• Very large design space

- Independent deployment
  - Dynamic reconfigurability
    - Specification
      - Various semi-formal mech.
      - Annotations
      - Exogenous managed externally
      - Endogenous managed by component
    - Management
      - Interaction-determined
      - Platform-determined
      - Hybrid and QoS dependent
        - directly composable (e.g., static memory)
        - architecture-related (e.g., parallelism)
        - emergent (response-time = exec-time and resource util)
        - usage-dependent (reliability)
        - environment-dependent (safety, security)
        - non-composable (portability, deadlock freedom)
Non-functional properties (2/2)

• Who manages QoS?
  – Component = Endogenous (threading?)
  – Container = Exogenous (security, replication?)
  – Component interactions determine non-functional properties (reliability)
  – Hybrid models

• Specifying QoS
  – Policies
  – Annotations
  – Run-time Negotiable?

• Reconfigurable QoS