Your systems. Working as one.

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

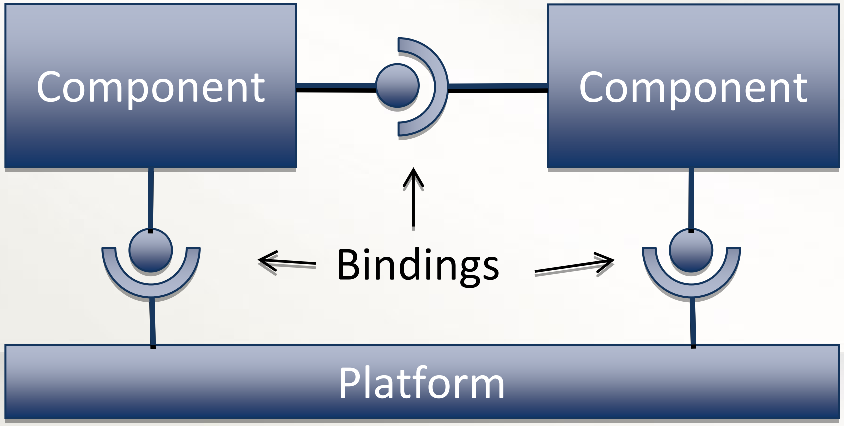
Goals

- 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
  - *A Classification Framework for Software Component Models*, by Ivica Crnkovi´c, S´everine Sentilles, Aneta Vulgarakis, and Michel R. V. Chaudron In IEEE Transactions on Software Engineering, Sept 2011

A Classification Framework for Software Component Models

Ivica Crnković, *Member, IEEE*, Séverine Sentilles, *Student Member, IEEE*, Aneta Vulgarakis, *Student Member, IEEE*, and Michel R. V. Chaudron

Abstract—In the last decade a large number of different software component models have been developed, with different aims and using different principles and technologies. This has resulted in a number of models which have many similarities, but also principal differences, and in many cases unclear concepts. Component-based development has not succeeded in providing standard principles, as has, for example, object-oriented development. In order to increase the understanding of the concepts, and to differentiate component models more easily, this paper identifies, discusses and characterises fundamental principles of component models, and provides a Component Model Classification Framework based on these principles. Further, the paper classifies a large number of component models using this framework.

Index Terms—Software components, software component models, component lifecycle, extra-functional properties, component composition.

1 Introduction

COMPONENT-BASED software engineering (CBSE) is an established area of software engineering. The techniques and technologies that form the basis for CBSE originate from object-oriented design, software architectures and Architecture Definition Languages (ADLs), middleware, and certain older approaches such as structural and modular development. The inspiration for "building systems from components" comes from other engineering disciplines, such as civil or electrical engineering. However, because software is in its nature different from physical products, a direct translation of principles from the classical engineering disciplines into software engineering is not possible. For example, the understanding of the term component has never been a problem in the classical engineering disciplines, since a component can be intuitively understood and this understanding fits well with fundamental theories and technologies. This is not the case with software components. There has been much debate on the notion of software component, as for example in [1].

From the beginning, CBSE struggled with the problem of obtaining a common and sufficiently precise definition of the concept of software component. An early and commonly used definition from Szyperski [2] states that *"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party."*

In spite of its generality, this definition does not capture the entire realm of component-based approaches. The ongoing debates have led to another definition by Heineman and Councill [3]: *"A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard."*

Here the emphasis has been shifted from component to *component model*[1] (a component model determines what is and what is not a component). Heineman and Councill [3] define a component model as follows *"A component model defines a set of standards for component implementation, naming, interoperability, customization, composition, evolution and deployment"*.

This definition points out that a component model covers multiple facets of the development process, dealing with (i) rules for the construction of individual components, and (ii) rules for the assembly of these components into a system.

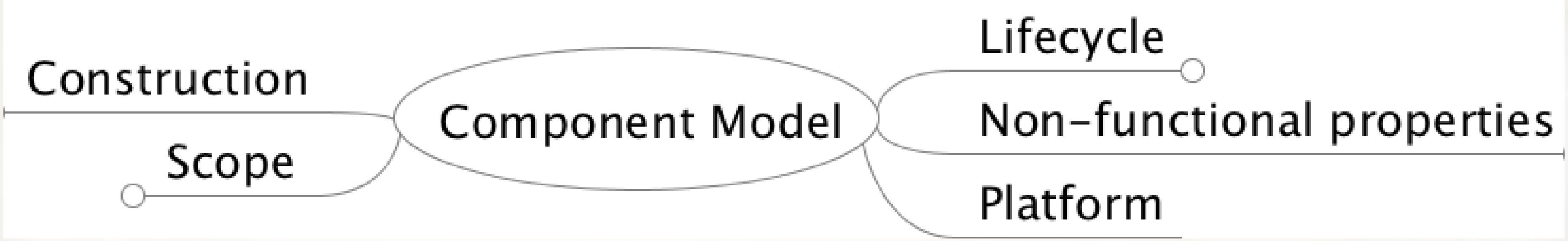
Currently, there exist many component models. Some component models target specific application domains, ranging from embedded systems (automotive software, consumer electronics) to particular business domains (finance, telecommunications, healthcare, transportation). Other component models are based on certain technological platforms (Enterprise Java Beans, DCOM). All component models are based on some, often implicit, assumptions about the architecture of the types of systems they are targeting.

The architectural assumptions made by the component models often relate to the way in which system level

- Ivica Crnkovic, Severine Sentilles and Aneta Vulgarakis are with Mälardalen University, School of Innovation, Design and Engineering Box 883, SE-72123 Västerås, Sweden.
E-mail: ivica.crnkovic, severine.sentilles, aneta.vulgarakis@mdh.se.
- Michel R.V. Chaudron is with Universiteit Leiden, Faculty of Science, Leiden Institute of Advanced Computer Science. P.O. Box 9512, 2300 RA Leiden, The Netherlands.
E-mail: chaudron@liacs.nl

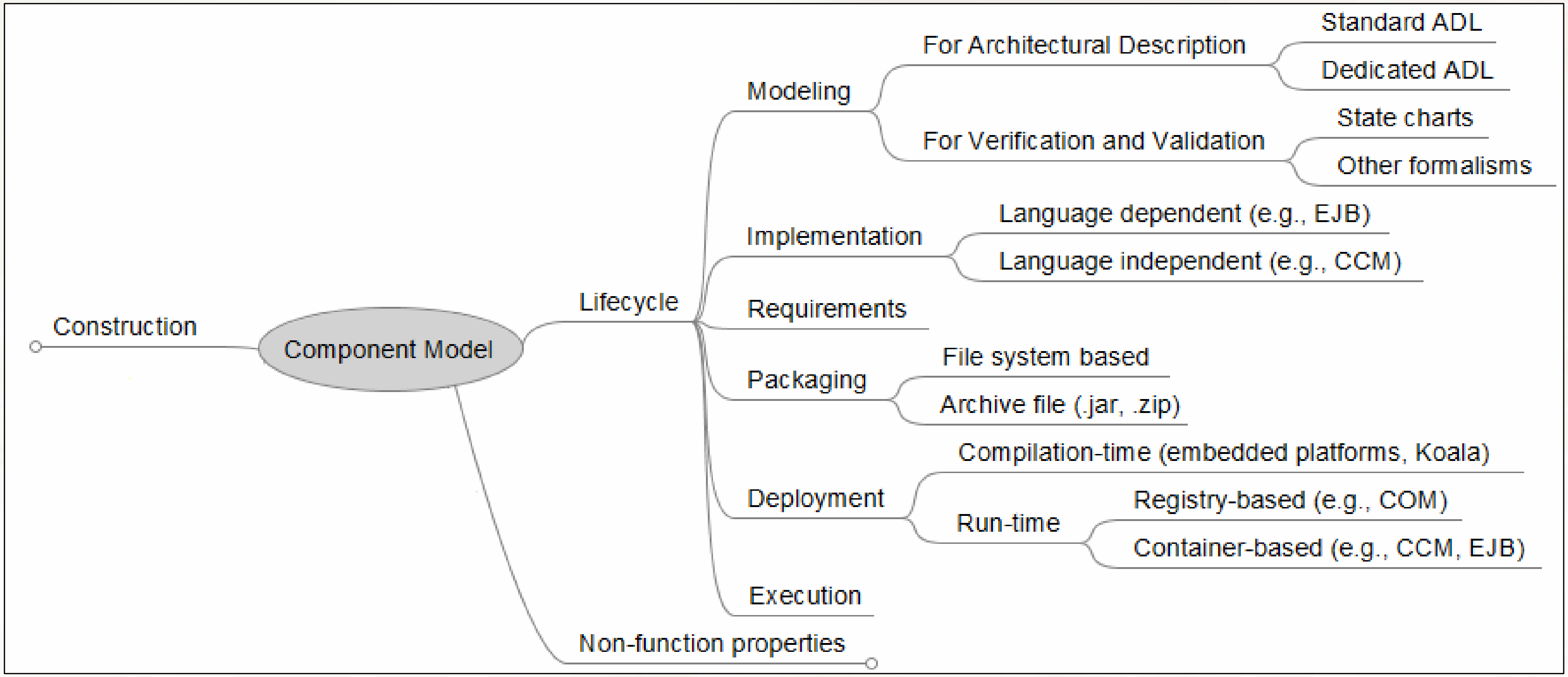
[1] In the remaining of the paper, we abbreviate the term "software component model" to "component model" and "software component" to "component".

Design Space for Component Models



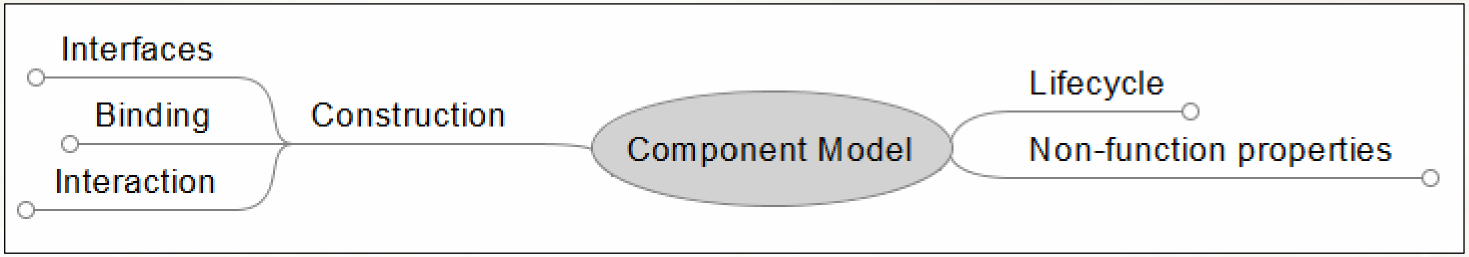
- Lifecycle management
  - Support in Comp Model for all stages from the component specification to integration into the system
- Construction
  - Mechanisms for building systems: Functional/Interface specification + Connections/Bindings + Communications/Interactions between components
- Non-functional 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


Construction
Component Model
Lifecycle
Modeling
For Architectural Description
Standard ADL
Dedicated ADL
For Verification and Validation
State charts
Other formalisms
Implementation
Language dependent (e.g., EJB)
Language independent (e.g., CCM)
Requirements
Packaging
File system based
Archive file (.jar, .zip)
Deployment
Compilation-time (embedded platforms, Koala)
Run-time
Registry-based (e.g., COM)
Container-based (e.g., CCM, EJB)
Execution
Non-function properties

- Modeling
- Implementation
- Packaging
- Deployment
- Execution

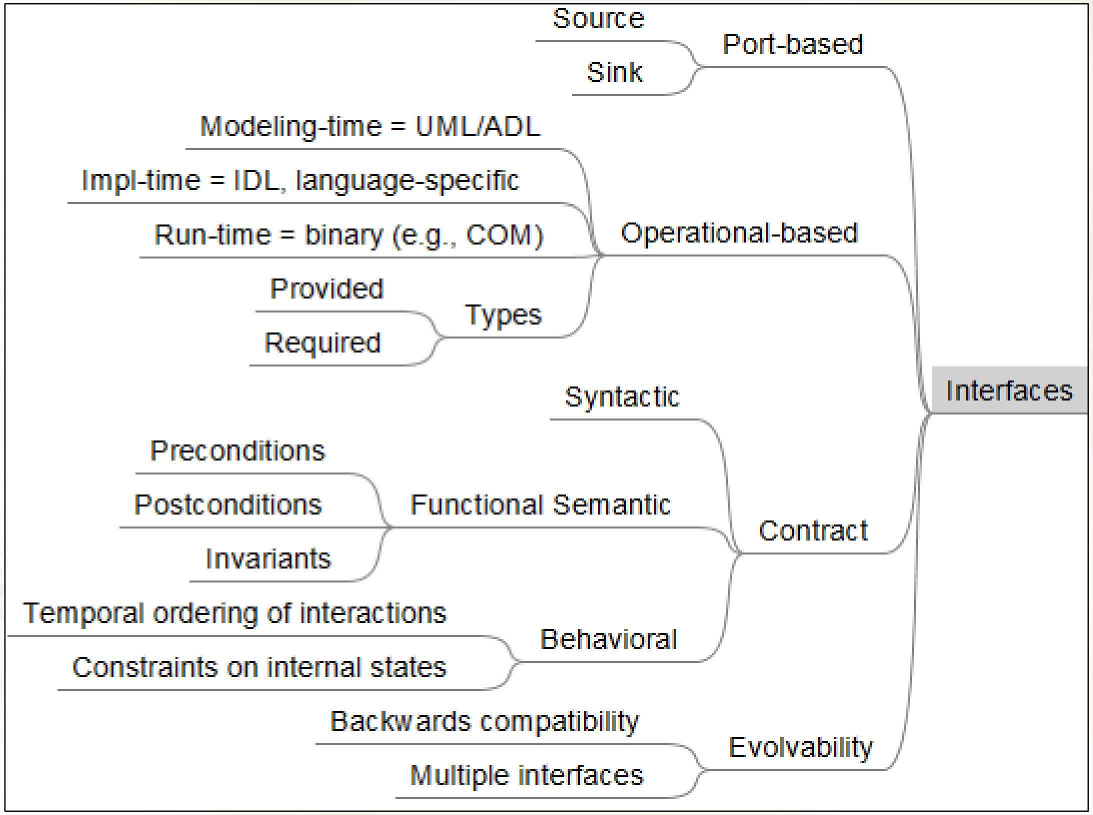
Component-based System Construction



- 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?

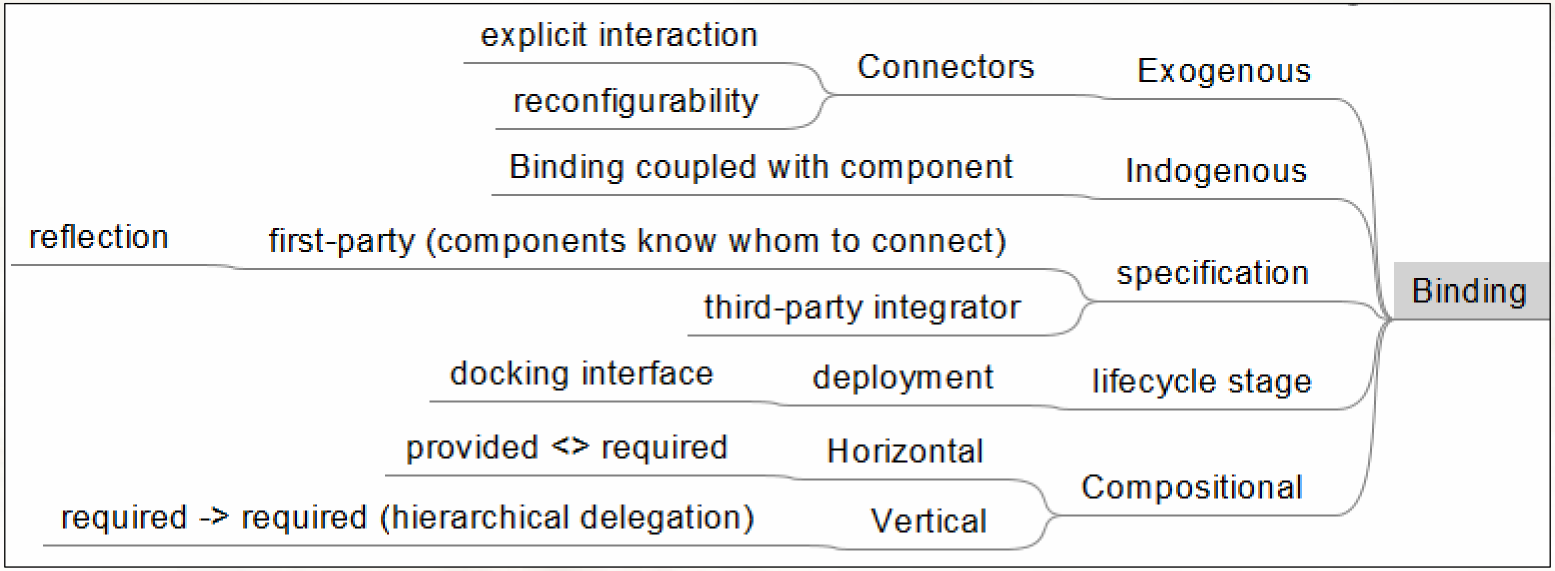
Design space for Component Interfaces

- 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?

Component Bindings



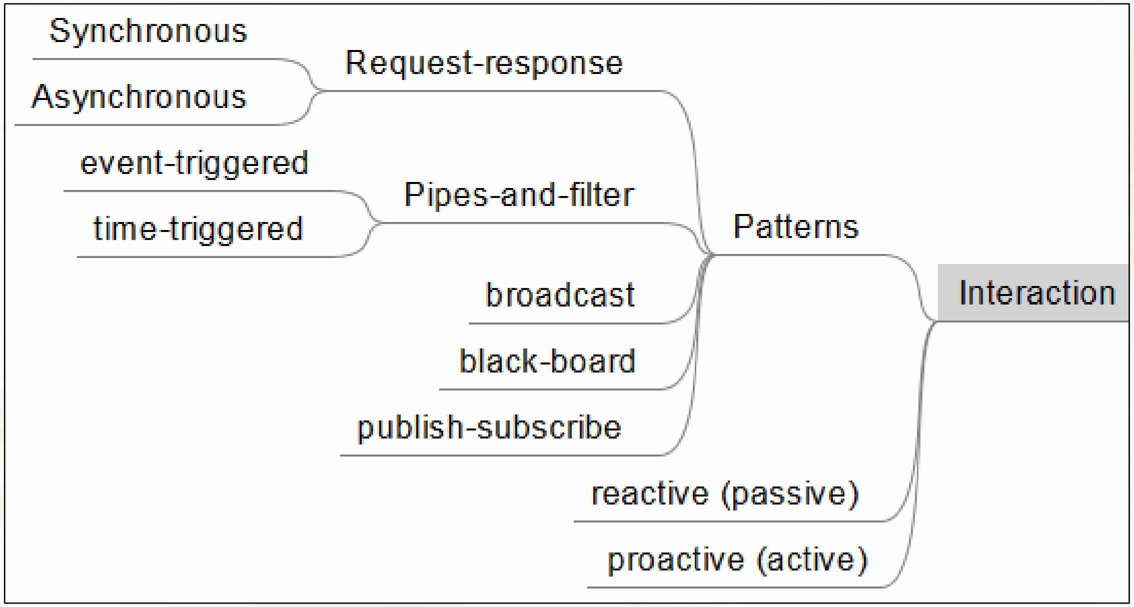
Binding is the process that establishes connections between components (composition, wiring)

- 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

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

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

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

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?

Remedy IT

- 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

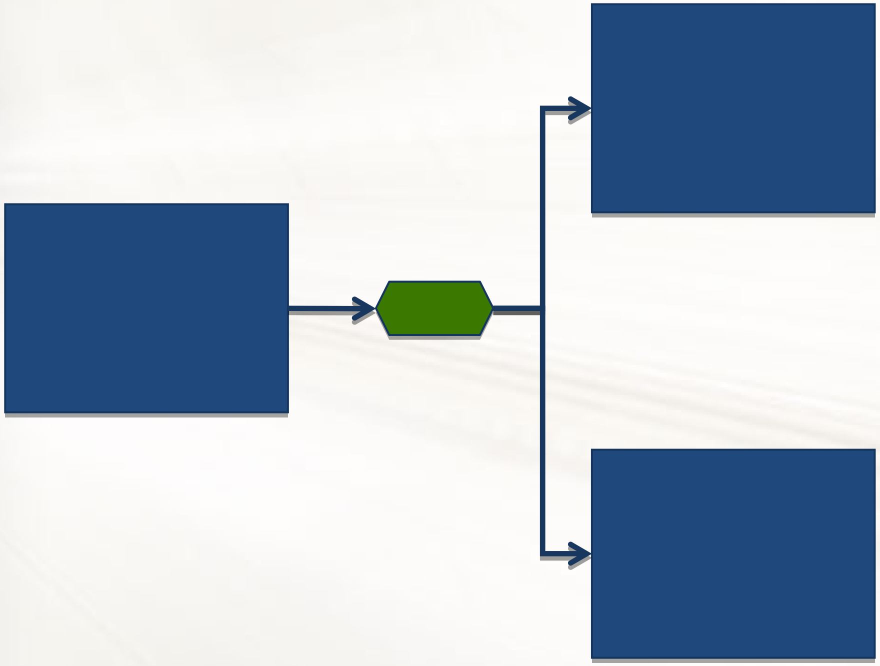
What does lwCCM need to become a UCM?

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

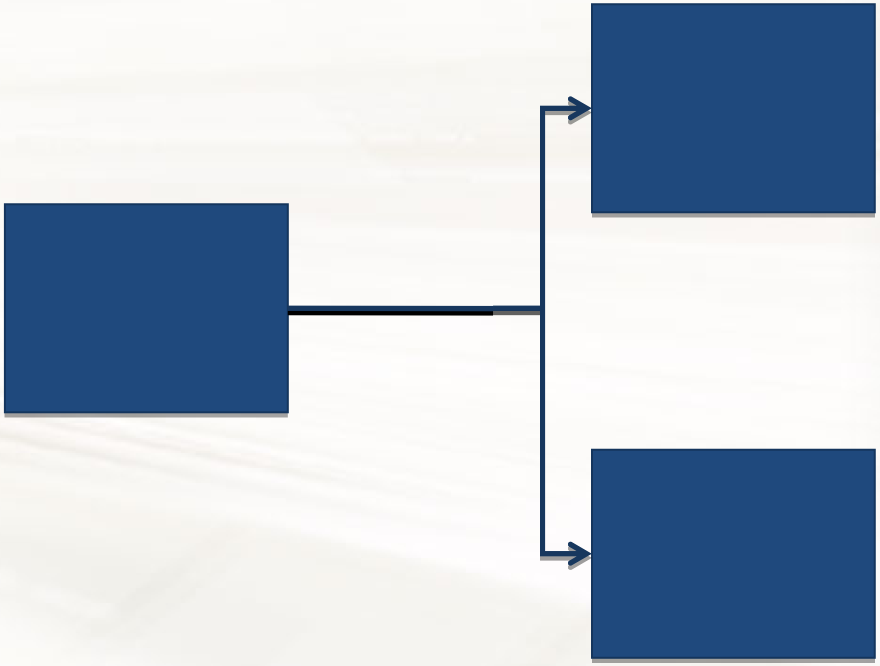
Purely Exogenous

- All component interactions are mediated via connectors
  - Connectors are local interfaces
  - There is no 1st order(*) middleware dependency in the component
    - 2nd 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

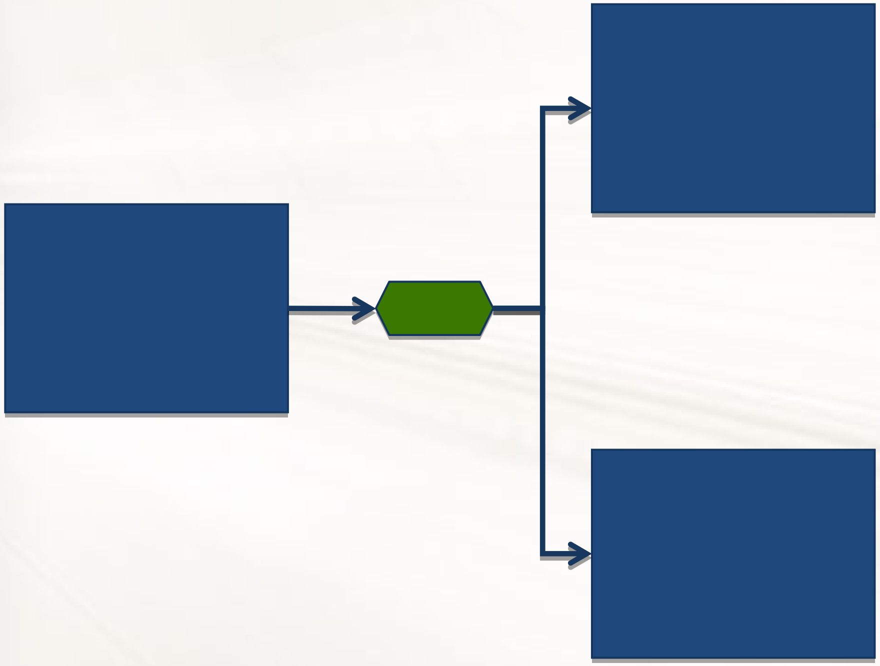


This

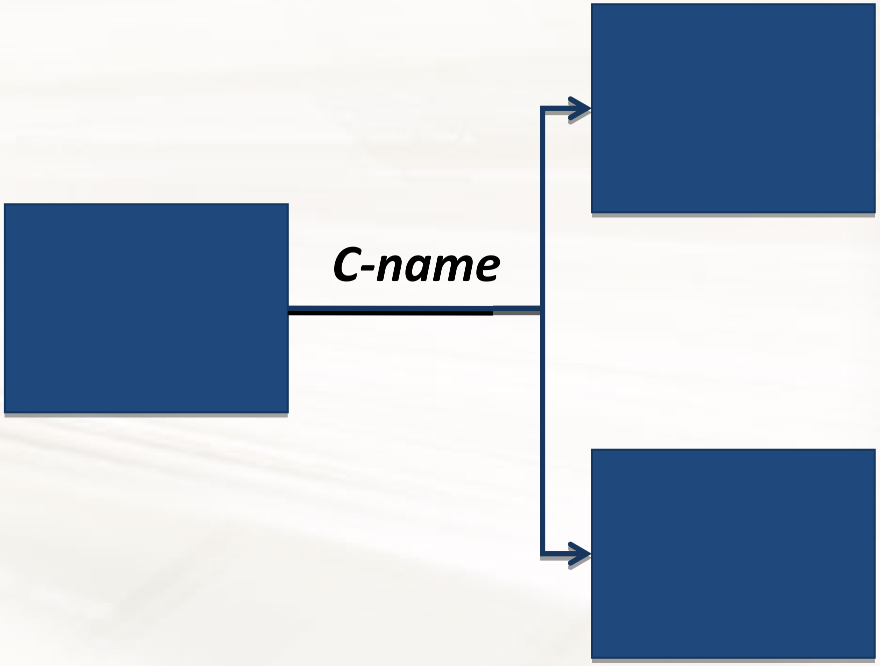


Not this

Explicit connectors



OK
Connector directly
visible



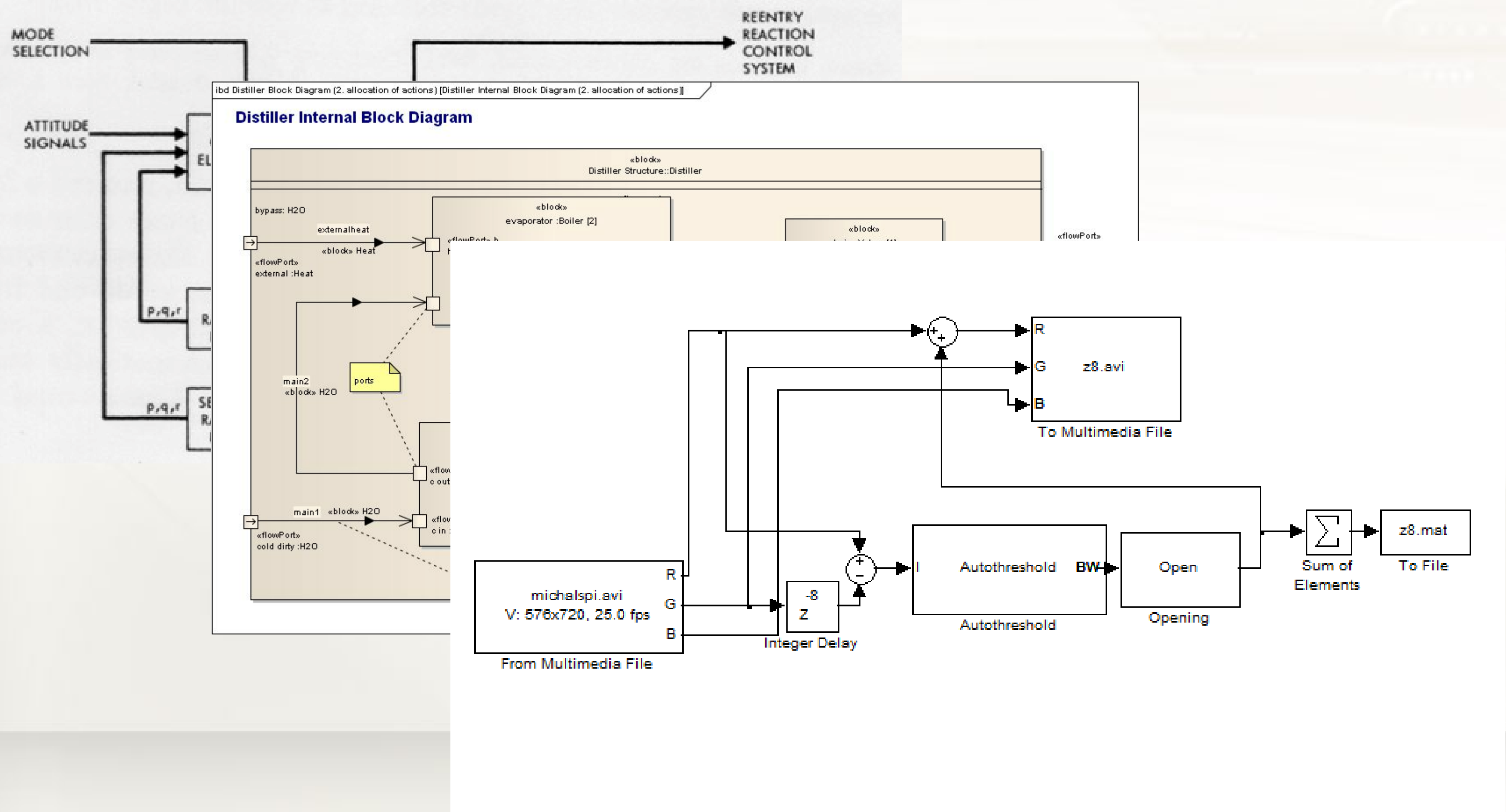
OK
Uses name to represent
Connector

Include pipes & filters as interaction style

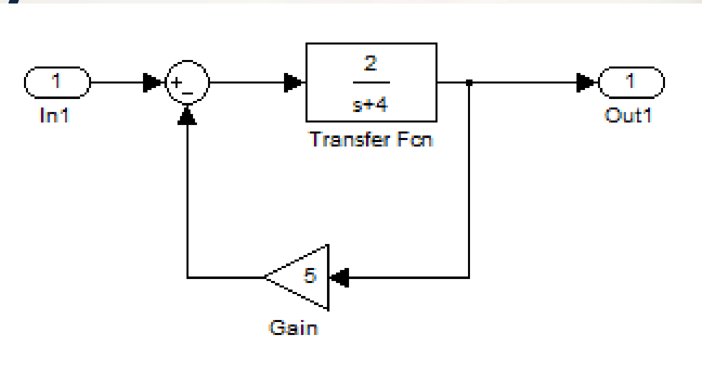
- Unidirectional communication
- Components are like filters that process the data, and the bindings are the pipes that transfer the data to the next filters.
- Allows separate control of the data-flow and control- flow between components.
  - The control flow is activated by a triggering interaction 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

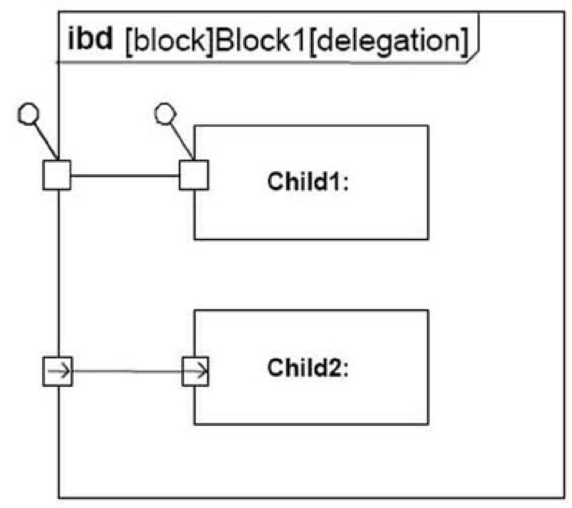
- Needed to support Function Block Diagrams
- SysML Internal Block Diagrams
- Control/Dataflow diagrams (e.g. LabView / simulink-like)



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

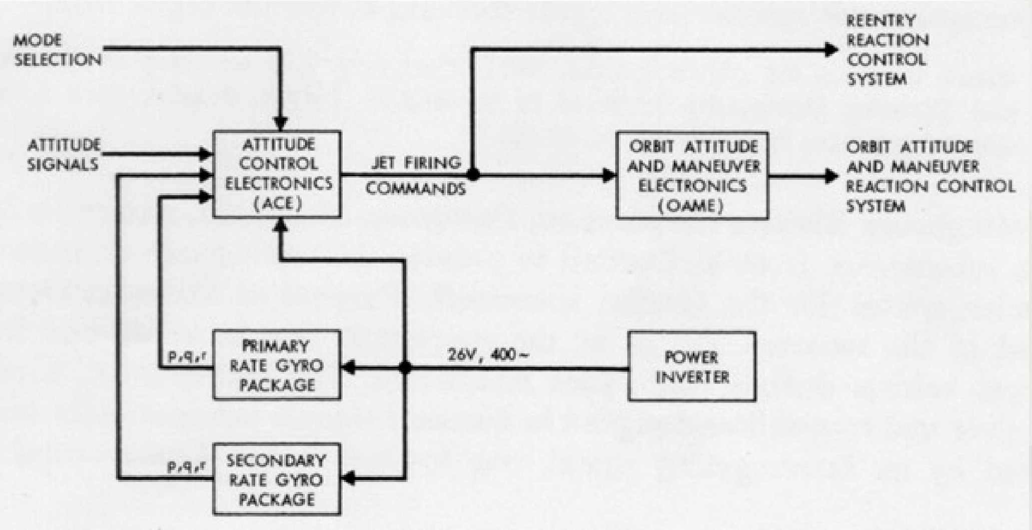


Abstract and enrich container interaction

- 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?



MODE SELECTION
ATTITUDE SIGNALS
ATTITUDE CONTROL ELECTRONICS (ACE)
JET FIRING COMMANDS
REENTRY REACTION CONTROL SYSTEM
ORBIT ATTITUDE AND MANEUVER ELECTRONICS (OAME)
ORBIT ATTITUDE AND MANEUVER REACTION CONTROL SYSTEM
p,q,r
PRIMARY RATE GYRO PACKAGE
26V, 400 ~
POWER INVERTER
p,q,r
SECONDARY RATE GYRO PACKAGE

Simplify deployment/assembly model

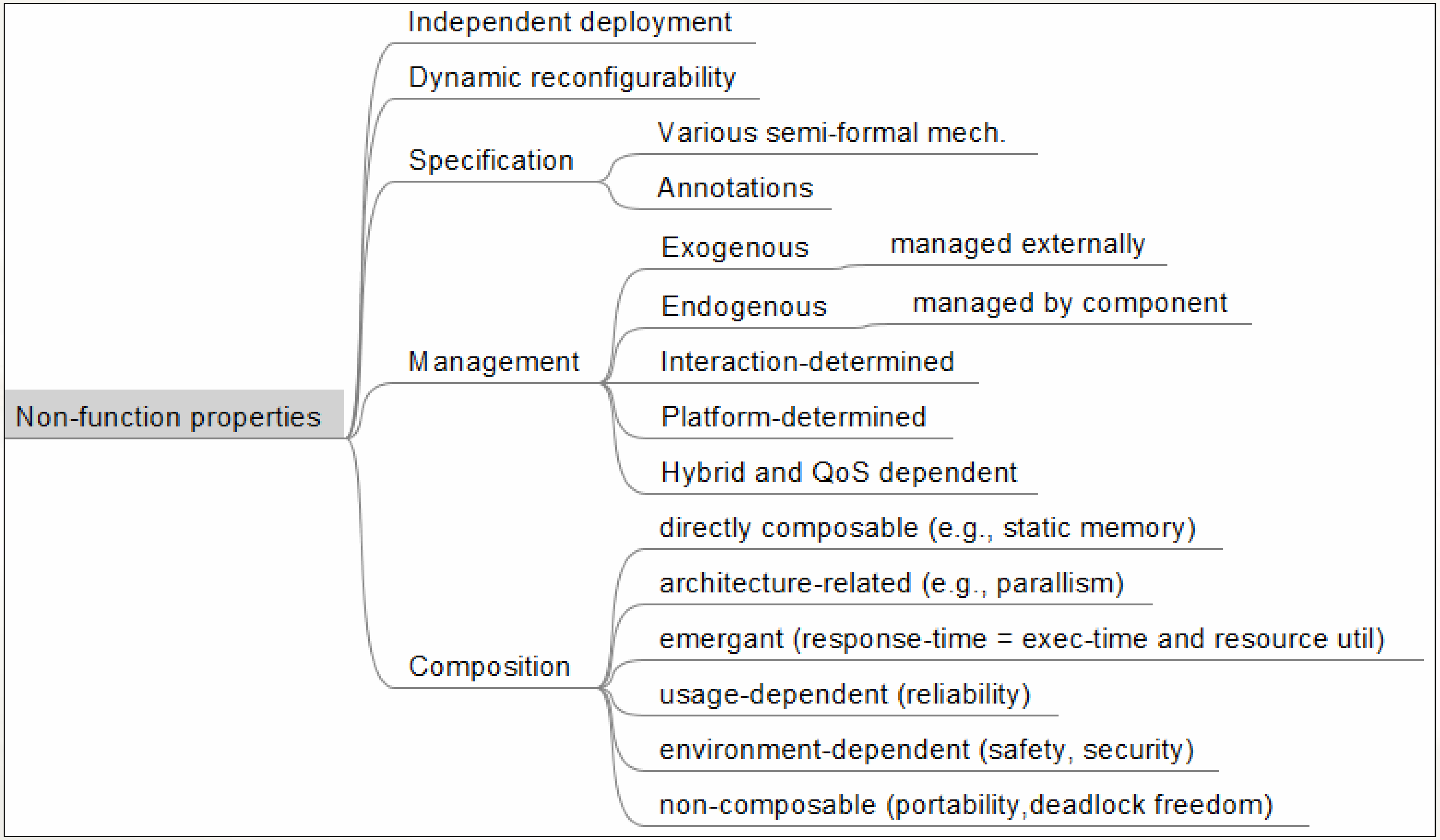
- Should deployment of components across containers be a separate spec/compliance point
  - Assemblies are components so can deployment on single container be simplified to that of a single component?
  - Can OSGI or similar packaging standards be leveraged to simplify spec?

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



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

- Very large design space

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