THALES

Component-based Engineering for Embedded Systems

USA – EU workshop

Philippe Kajfasz

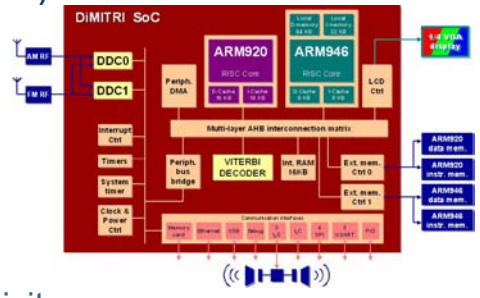
philippe.kajfasz@fr.thalesgroup.com

July 7, 2005

RT/E Systems Characteristics

Software part in RT/E Systems is increasing

- Move from fixed wired hardware ➔ flexible logics (software)
- ... but still heterogeneous HW platforms (GPP, DSP, FPGA)
  - Reconciled approach is needed



Software in RT/E Systems is becoming more complex:

- More functionality
- More variability, versatility
- More integration in large-scale systems:
  - More connectivity
  - More remote manageability
- Move towards dynamic (re)configuration:
  - Self-configuration
  - Self-organisation

RT/E Systems have still to deal with 'real' world and its constraints:

- Time/latency, available resources, (low) power, but also volume, weight, cost,
- Safety, reliability, dependability, QoS,…, security
- Certification, DO178B, DO254, …

How to manage all these trends ?

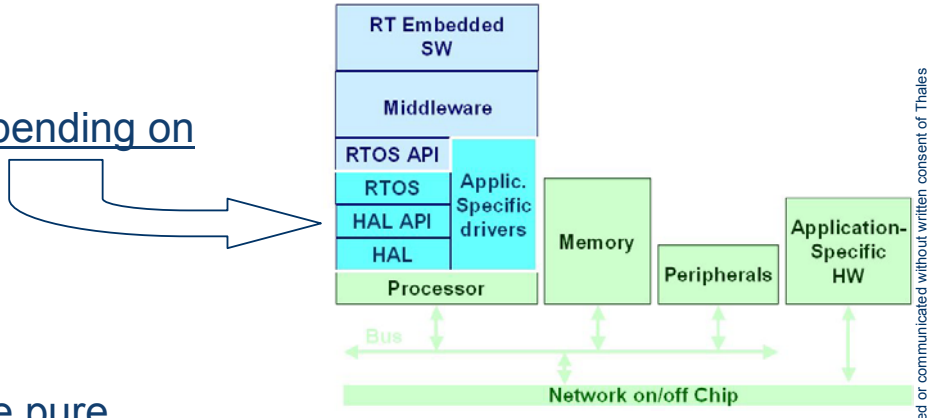
move from performance-centric to complexity-centric…

...w/o loosing the performance and time support!

An abstraction of the HW platform

Hardware Dependent Software

- All the software that is directly depending on the underlying hardware



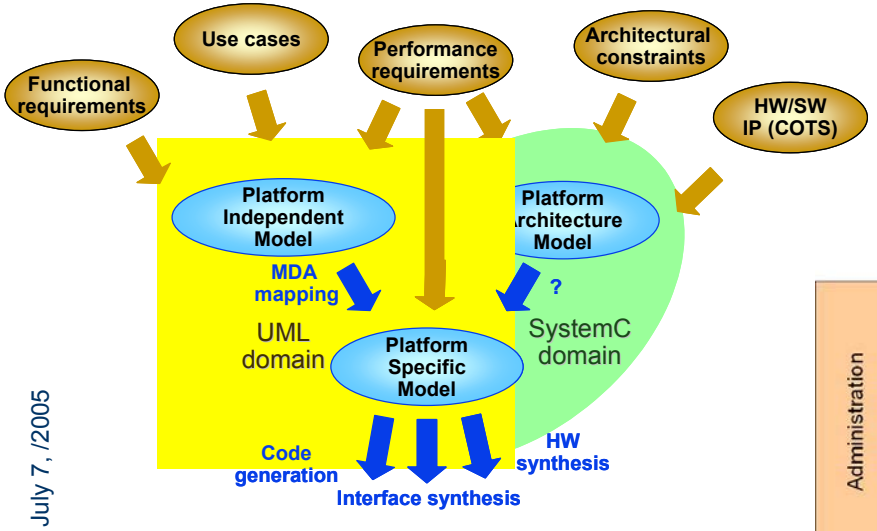
Middleware

- ***Anything*** that stands between the pure application code and the raw (networked) platform
  - Not only the communication support
  - Should be the mediator between the application code and the platform resources and services (HdS)
- What are the main characteristics for a RT/E middleware?
  - Affordable
  - Providing
    - Suitable support for application break-down in manageable (reusable) parts
    - Suitable support for RT/E non-functional properties
    - Separation of concerns
    - Isolation, partitions

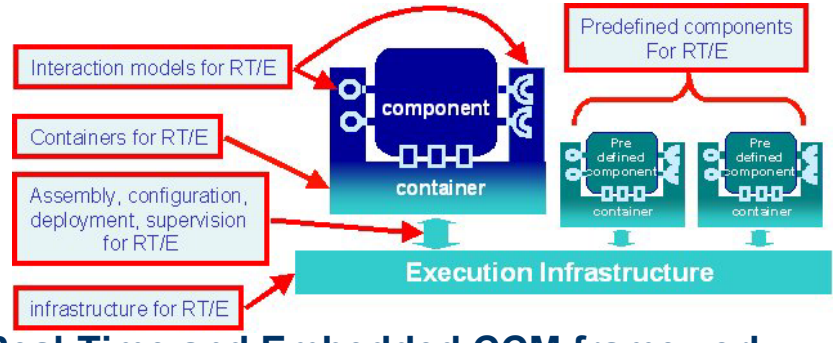
Design Methodology and Tools

MIC approach

- Seamless design flow
- Modeling / Simulation / Code generation
- Strong connection to
  - domain specific environments and tools (UML2, AADL, SystemC, C/C++, EmbeddedC, VHDL/verilog, …)
  - RTOS
  - CCM framework
- Support for legacy code integration
- Support for static & dynamic re-configuration
- Deployment – SCA3.x/CF

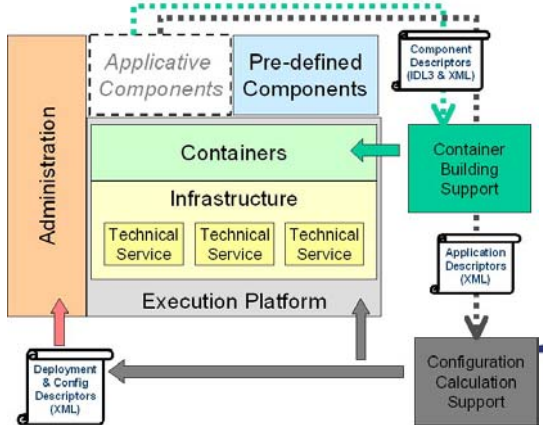


EUREKA/ITEA -Martes project



Real-Time and Embedded CCM framework:

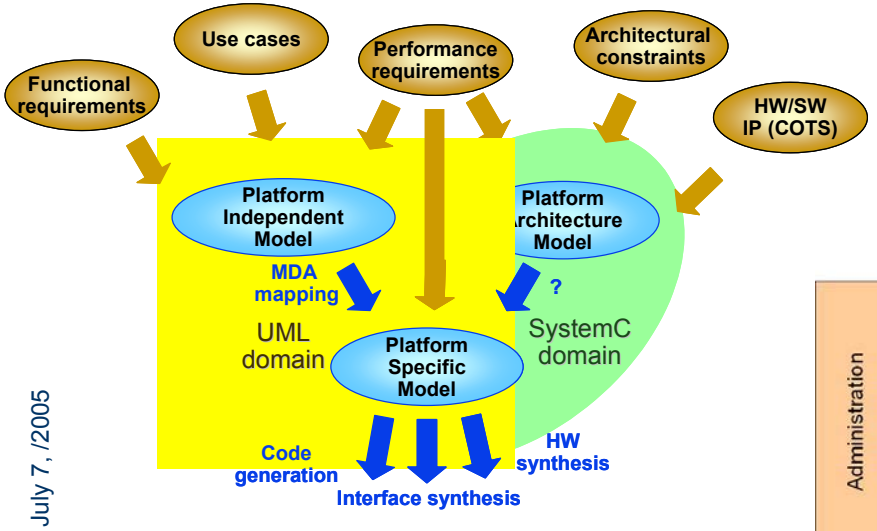
- A Container/Component Model based on OMG Lightweight CCM (CCM ≠ CORBA)
- All CORBA dependencies are managed by the container
  - not visible by the application code
- All interactions are made using native calls
  - only CORBA dependency is with IDL
- Supports reusable components
- Extensions to support partitioned execution platform



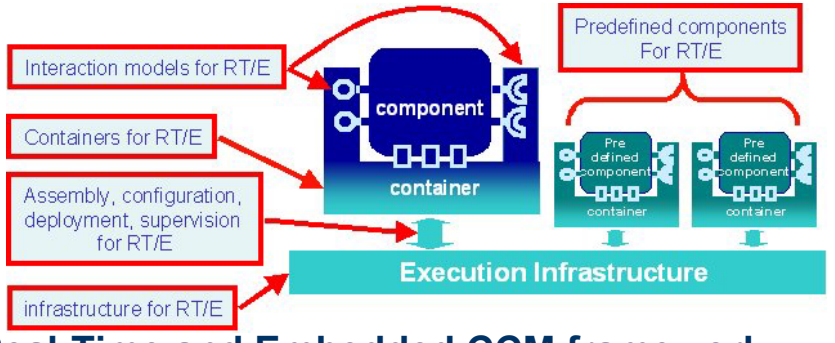
Design Methodology and Tools

MIC approach

- Seamless design flow
- Modeling / Simulation / Code generation
- Strong connection to
  - domain specific environments and tools (UML2, AADL, SystemC, C/C++, EmbeddedC, VHDL/verilog, …)
  - RTOS
  - CCM framework
- Support for legacy code integration
- Support for static & dynamic re-configuration
- Deployment – SCA3.x/CF

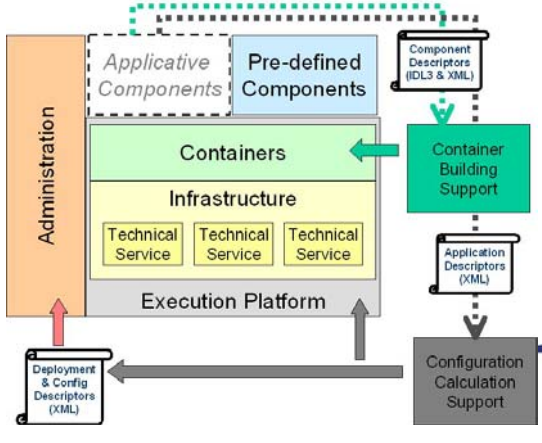


EUREKA/ITEA -Martes project



Real-Time and Embedded CCM framework:

- A Container/Component Model based on OMG Lightweight CCM (CCM ≠ CORBA)
- All CORBA dependencies are managed by the container
  - not visible by the application code
- All interactions are made using native calls
  - only CORBA dependency is with IDL
- Supports reusable components
- Extensions to support partitioned execution platform



Back-up slide

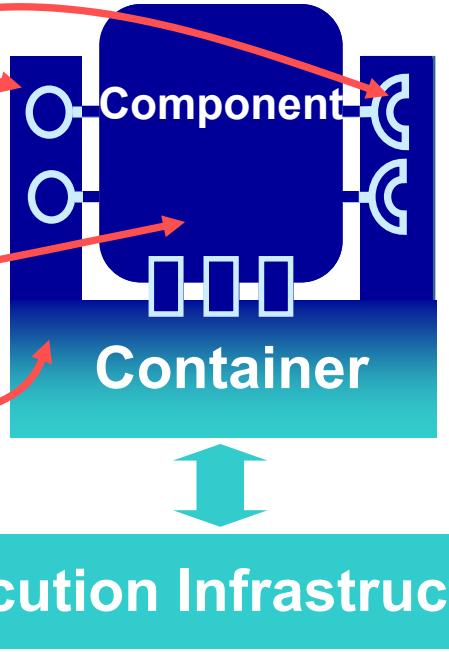
Component/Container Model is a key architectural pattern

Explicit description of:

- provided services
- requested services

Separation of concerns:

- business logic
- 'technical' properties



(containers are provided as part of the infrastructure)

(based on descriptors ➔ move from fully programmatic to declarative)

✓ Easier deployment and reuse, needed for reconfiguration