Component based Middleware for real-time embedded systems

Ansgar Radermacher
CEA-List
Agenda

- Introduction – Demands of RT & Embedded Applications
- Towards a common component model for this domain
  - Convergence of models
  - Extensions for Embedded & RealTime applications
  - Complementary activities – validation, model of computation
  - Specific usage: fault tolerance mechanisms

- Model Driven Development
  - From Models to executable middleware code
  - eC3M UML Models (within Papyrus UML)
  - Code generation
Demands of RT & Embedded applications

- Real-time: timing constraints (soft or hard deadlines)
- Resource constraints (small memory, RAM/ROM)
- OS constraints (fixed number of threads, Semaphores, message sizes, ...)

⇒ software tailored to specific target, costly to port and to adapt

- Quote from ITEA-Merced project, same spirit behind IST-Compare

move from performance-centric to complexity-centric... ..without loosing performance and time support!
Component Approach

- CORBA Component model as good base candidate
  - OMG Standard
  - Separation of component & container enabling better portability
  - Lightweight variant exists
- But … only pre-defined container with fixed services
- But … only supports small set of interaction patterns … with specific and fixed implementations
  - Synchronous method calls (via CORBA)
  - Event based communications
  - Streaming (recently added)

⇒ Not enough for embedded systems
Excursion: CORBA Component Model

- Component has provided and required ports (interfaces, events)
- Lightweight CCM = CCM w/o introspection, simple container type (no security/persistence/transaction)
Packaging and Deployment

“Classic” CORBA: No standard means of ...
- Instantiation & Configuration, Deployment on nodes

Deployment & Configuration (OMG standard)
- Components are packaged into a self-descriptive package
- Packages can be assembled
- Assemblies can be deployed

Deployment tools are based on XML descriptors

Deployment Plan “.cdp”
Component Description
Deployment tool
- static
- dynamic
- bootloader file
- Connector fragments
- TFTP (or similar)
- additional compilation step
Towards an enhanced component model

Better container support
- New “simple” container with interception plug-ins
- OMG Standard QoS for CCM

Connector support allows to specify
- Interaction pattern during component development time
- Interaction implementation during deployment time

Basic principles: Connectors are …
- … like components: can be configured, have implementations
- … like connections: ports don’t have fixed interface types, need to be *generated*, are split into connector ends (fragments)
Towards an enhanced component model

- Configuration of services/connectors in *declarative* way
- Advantage: keep platform specific programming (Mutex, Conditions, ...) out of business logic within component

  - eC3M
    - *embedded Component-Container-Connector Middleware* (www.ist-compare.org)

- Ongoing convergence actions among: CCM, Fractal, Autosar, IMA…
  - Ongoing project focus on CCM & Fractal
  - Take Model Driven Development in Account
  - for declarative NFPs http://www.omg-marte.org
Connector/Container Service Examples

**Connector Examples**
- Synchronous Calls via CORBA, OSEK messages
- FIFO – in shared memory or distributed via CORBA
- Events (making pre-defined event ports obsolete)
- Connectors supporting Fault Tolerance, Communication avec FPGA (current projects)

**Container Service Examples**
- Access protection via n-readers, 1 writer
- Logging/Tracing of requests
- Execution time measurement

**Limits:** focus on structural info + declarative NFP
- execution model managed implicitly
How to define port semantics?

Component implementation assumes a certain semantics associated with call on port (and being called on a port)

- Connector maybe incompatible with a component, even if port interfaces are matching

Options

- Specify required connector type at component design time
  - A bit restrictive with respect to component reuse

- Via *port type* definitions
  - Port type captures certain semantics
  - Better component reuse of component implementations
  - Support complex-ports (single port can provide & require)
More insight on connectors for CCM (runtime)

- connectors realizations are fragmented
  - fragments are co-localized with components
    - Implement interface between component and connector
  - communication between fragments is connector specific
Complementary activities: computation models

- **Formal technique:** compute execution paths from finite state machine
  - Rules to trigger transitions, consume events, chain actions
    - are parameters
  - Symbolic execution engine
  - Applied to test generation, property verification...
  - Reuse free tools for formal techniques
    - reduce symbolic expressions, optimize trees, solve constraints...
    - Integrated to MDE technologies (Eclipse, EMF, ATL, Papyrus)

- Manage heterogeneity by dynamically changing execution rules
  - Global view of system architecture: components (state machines)

✓ **Limits:** rules are defined case by case
Heterogeneity of computation and communication models

**THeSys (Tackling Heterogeneous Systems)**

Mathematical foundation (denotational):
LEM (Language for expressing Execution Models)
- integrate hierarchy, component, interconnection and (unrestricted) time
- values and times of exchanged data are synchronised

→ formal description of Model of Computation and Communication (MoCC)
Related work (THeSys)

- Ptolemy II: not formal, heterogeneous (hierarchical), no language for expressing models of computation

- CommUnity (Univ. Leicester, Lisbon): formal, no time but causality (pre/post).

- BIP: formal, powerful interaction language, more limited to interactions

- Time treated externally, not meta model oriented

- Kermeta: not formal, meta programming language
Model Driven Development

- Meta-Model of component approach eC3M $\Rightarrow$ UML Profile
- UML + MARTE Profile (for NFPs) + eC3M Profile
- Use of Papyrus (www.papyrus-uml.org) UML modeler
  - Strong Profile Support
  - “Invented here”
- Build model, apply stereotypes/values
- Generate eC3M descriptor files from model
  - IDL, Component Descriptor (CCD), Deployment plan (CDP)
- Generate container, bootloader (glue) via MicroCCM (joint implementation from Thales and CEA-List, hopefully released soon as OpenSource)
Refinement steps

Model Level

1. Redundance parameters
   - Style
   - Number of replicas
   - ...

Allocation constraints

Calc (cosine)

Display

2

Calc

Display

Calc

Display

2

FIFO

Allocation (automatic)

FT parameter domaine

1. FaultDetector-DeployPolicy (Infrastructure)

«sysComponent» FaultDetector

Node A

Calc₂

FD-A

Calc₁

FD-B

Node B

Display

FD-C

Node C
Deployment of a component (replica) on multiple nodes

Model Level

Node A
Calc₂
FD-A

Node B
Calc₁
FD-B

Node C
Display
FD-C

Packaging Tool

XML D&C

<!-- Instance of Calc -->
<instance id="instance1">
  <name>Calc_inst</name>
  <node>nodeA, nodeB</node>
  <implementation ref="implCalc"/>
  <configProperty>…
</instance>

<connection id="connexion3">
  <name>Calc_Display</name>
  <implementation ref="implFT-FIFO"/>
  …
</connection>
void FTFIFO_IDisplay_f_conn::display (CORBA::Float value) {
    if (m_voter != NULL) {
        // calculate hash of request (used to simplify comparisons).
        Hash hash;
        hash.add (m_voter->getRequestNr ());
        hash.add (value);
        m_voter->acknowledgeRequest (hash.get ());
    }
    for (int i = 0; i<MAX_NR_OF_NODES; i++) {
        if (m_set.isOnNode (i)) m_set.getObj (i)->display (value);
    }
    Communication between replicas
    Multiplier: Communication with other instance (maybe replicated)
Conclusions

Specific support for embedded requirements through
- Container services
- Connectors (configurable)
  \(\Rightarrow\) Towards adaptive executions & communication platforms
  (heterogeneity at several levels)
  \(\Rightarrow\) Declarative specification of non-functional properties

Complete MDD tool chain
- Model, Analysis (not shown Thesis H. Espinoza), deployment, execution & operation

Future work
- Convergence of component models based on standards
- Overcome “two worlds” separation:
  \(\Rightarrow\) integrate formal approaches
Questions?