Towards a Real-time CORBA Component Model

Nanbor Wang, Krishnakumar Balasubramanian, Chris Gill

Department of Computer Science
Washington University in St. Louis

{nanbor,kitty,cdgill}@cs.wustl.edu

July 18, 2002
Limitations of Current Approaches

- Non-functional aspects are controlled using interfaces at the same level as functional objects
  - ORB configuration
  - POA policies
  - Objects/services configuration and composition
- Lack of “reuse boundary”
  - hard to specify object dependencies explicitly
  - can only be enforced with “good programming practice”
Towards a Real-time CORBA Component Model

Promising Solution: Component Models

- Forces behind component models: Separation of concerns, e.g.:
  - Run-time environment configuration
  - Connections between objects & run-time
  - Composition of objects
- Supporting run-time object composition
  - **Component**: a reusable physical entity
  - **Container**: a standardized environment for a component to interact with run-time & vice versa
  - **Application Server**: a generic server process
  - A deployment mechanism to compose components

- J2EE (EJB), DCOM & CORBA Component Model
Towards a Real-time CORBA Component Model

The CORBA Component Model (CCM)

- Extends the CORBA Object Model
- Provides standard run-time environment for components
  - application servers
  - containers
- Uses metadata to describe
  - application server and container configurations
  - component run-time requirements, e.g., transactional, persistence state
  - component configuration
  - component dependencies
  - component connections
Before CCM: Development via Engineering

- Invoke ORB_init()
- Initialize RootPOA
- Initialize motor “modulator” servant, register with POA, acquire its object reference
- Same for “actuator position sensor”
- Same for left and right limit switches
- Instantiate the control_panel servant using the previously acquired object references
- Register the servant with POA and acquire the object reference
- Initialize an EventChannel
- Connect two proximity objects as event suppliers
- Connect the control_panel object as event consumer
- Activate POA(s), now we are ready.
Towards a Real-time CORBA Component Model

After CCM: Development via Composition

- Application server
  - run-time management
  - service initialization
- CCM Assembly Archive
  - Assembly descriptor
    - Install components
      - Component implementations
      - Component descriptors
      - Configuration property files
    - Establish connections
After CCM (cont.) – Component Implementations

```xml
<componentfiles>
  <componentfile id="Motor">
    <fileinarchive name="AB-motor.csd"/>
  </componentfile>

  <componentfile id="Location">
    <fileinarchive name="linear-encoder.csd"/>
  </componentfile>

  <componentfile id="Proximity">
    <fileinarchive name="p-switch.csd"/>
  </componentfile>

  <componentfile id="controller">
    <fileinarchive name="AB-panel-if.csd"/>
  </componentfile>
</componentfiles>
```

- An assembly descriptor specifies what component implementations are needed by referring to their component descriptors
- A Component descriptor (.csd) records component features and dependencies to other software modules
Towards a Real-time CORBA Component Model

After CCM (cont.) – Component Instantiations

- An assembly descriptor specifies how homes and components should be instantiated
- A component property file (.cpf) can be associated with a home or a component instantiation to override default component properties
Towards a Real-time CORBA Component Model

After CCM (cont.) – Connecting Components

- An assembly descriptor also specifies how components instances are connected together

```xml
<connections>
  ...
  <connectinterface>
    <usesport>
      <usesidentifier>modulator</usesidentifier>
      <componentinstantiationref idref="Motor"/>
    </usesport>
    <providesport>
      <providesidentifier>modulator</providesidentifier>
      <componentinstantiationref idref="Controller"/>
    </providesport>
    <providesidentifier>EmStop</providesidentifier>
    <componentinstantiationref idref="Controller"/>
  </connectinterface>
  <connectevent>
    <consumesport>
      <consumesidentifier>EmStop</consumesidentifier>
      <componentinstantiationref idref="Controller"/>
    </consumesport>
    <publishesport>
      <publishesidentifier>EmStop</publishesidentifier>
      <componentinstantiationref idref="left"/>
    </publishesport>
  </connectevent>
  ...
</connections>
```
Why doesn't running a RTORB beneath CCM make it an RTCCM implementation?

RTCCM ≠ CCM + RTCORBA

• Plain CCM has no mechanisms to specify and enforce RT policies
• RT policies need to be assured end-to-end for components & connections
• Trying to ensure RT policies are met in components leads to:
  – Tight couplings among component implementations
  – Difficulty in reusing existing components (without RT knowledge)
  – Failure to utilize many RT mechanisms that go beyond component implementations
    • Component connections
      – private connections
      – pre-connections
    • Component collaborations
      – Thread pools
      – Thread borrowing
Overview of Real-time CCM (RTCCM)

• Solution: Configure RT-policies/mechanisms using CCM’s metadata

- Abstracting RT CORBA related systemic (QoS) aspects
  - Specify RT policies of a component instance
  - Specify RT policies of a connection between components
  - Allocating & computation and communication resources for components
  - Specify sharing & collaboration of resources among components
  - Configuring ORB with custom communication mechanisms and options
Towards a Real-time CORBA Component Model

Component-Integrated ACE ORB (CIAO)

- We are extending CIAO’s meta-model to make RT policies an integral part of CCM

1. Component default priority model
2. Override component priority model
3. Priority level of a component instance
4. Defining thread pools
5. Associate thread pools with components
6. Specify queuing policies
7. Specify pre-connections
8. Specify private connections
9. Configure ORB components
   - Custom protocols
   - Priority mappings
RTCCM Descriptor Examples

- Component default priority model
- Override component priority model
- Specify the priority level of a component instance
- Associate component instances with customized protocol
RTCCM Descriptor Examples (cont.)

- Define thread pools
- Define QoS aggregates
- Associate thread pools with components
- Associate QoS aggregates with connections
Towards a Real-time CORBA Component Model

Current Status of CIAO

- **CIAO**: A CCM implementation based on *the ACE ORB* (TAO)
- Extending (component and assembly) descriptors for configuring RT policies
- Applying reflective middleware techniques to support other non-functional aspects with CCM metadata
  - Bandwidth reservation
  - Memory management
  - Transport selection

---

Washington University, St. Louis
Future Work

- RT-CCM can only enforce RT policies and provide the supporting mechanisms
- Correct combinations of these policies are beyond the scope of RT-CCM
- Support other QoS assurance mechanisms
- Integration with Vanderbilt University’s Model-Integrating Computing Tools
  1. Configuring and deploying an application services end-to-end
  2. Composing components into application server components
  3. Configuring application component containers
  4. Synthesizing application component implementations
  5. Synthesizing middleware-specific configurations
  6. Synthesizing middleware implementations
Concluding Remarks

• Component Model promotes reuse by separating non-functional concerns
• RTCCM ≠ CCM + RTCORBA
• Components descriptors and assembly descriptors can be used to specify RT policies & mechanisms
• CCM can be extended to support other non-functional properties, such as QoS properties
• CCM only enforces the specified policies, it does not ensure they are correctly composed
• Integrating with MIC tools to ensure correct deployment of non-functional policies