

Towards a Real-time CORBA Component Model

Nanbor Wang, Krishnakumar Balasubramanian, Chris Gill

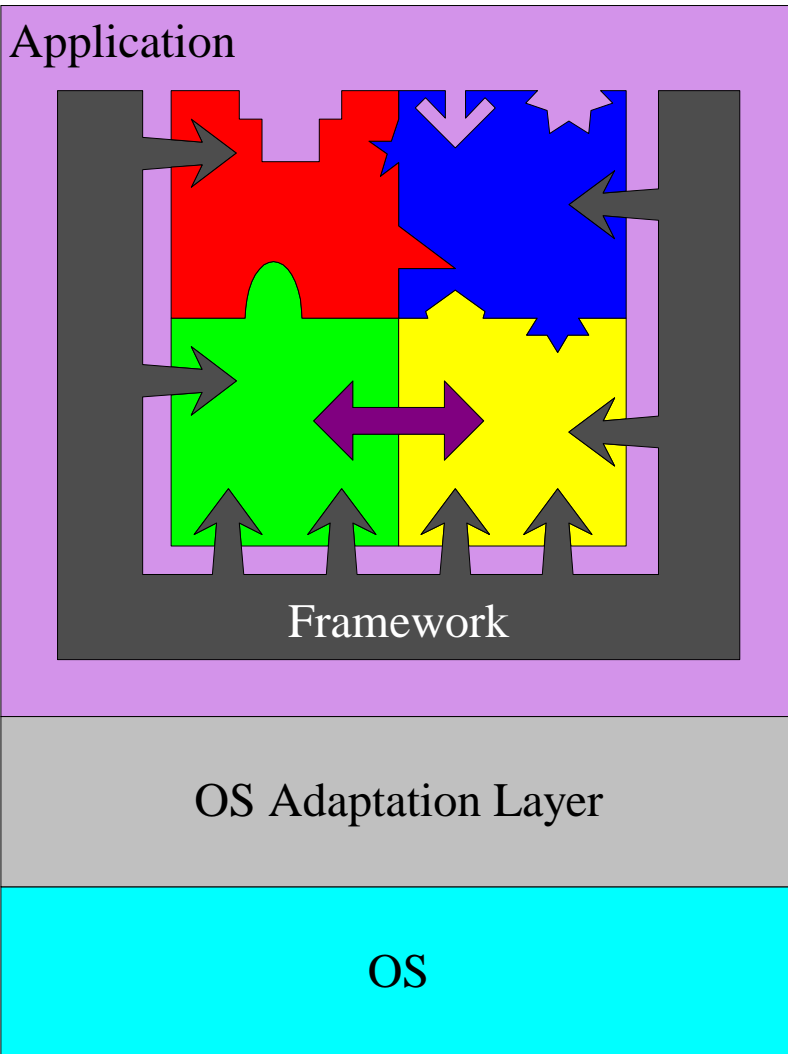
Department of Computer Science
Washington University in St. Louis

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



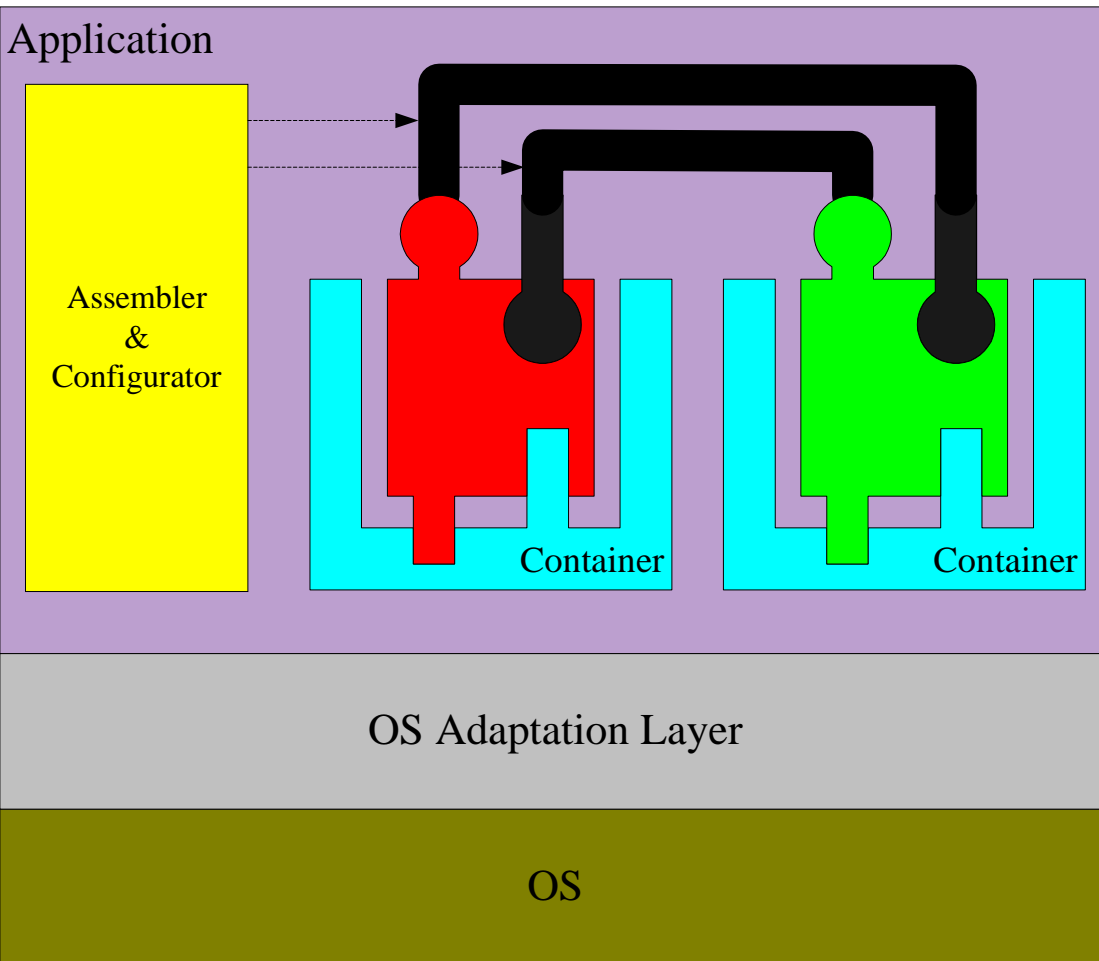
July 18, 2002

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”

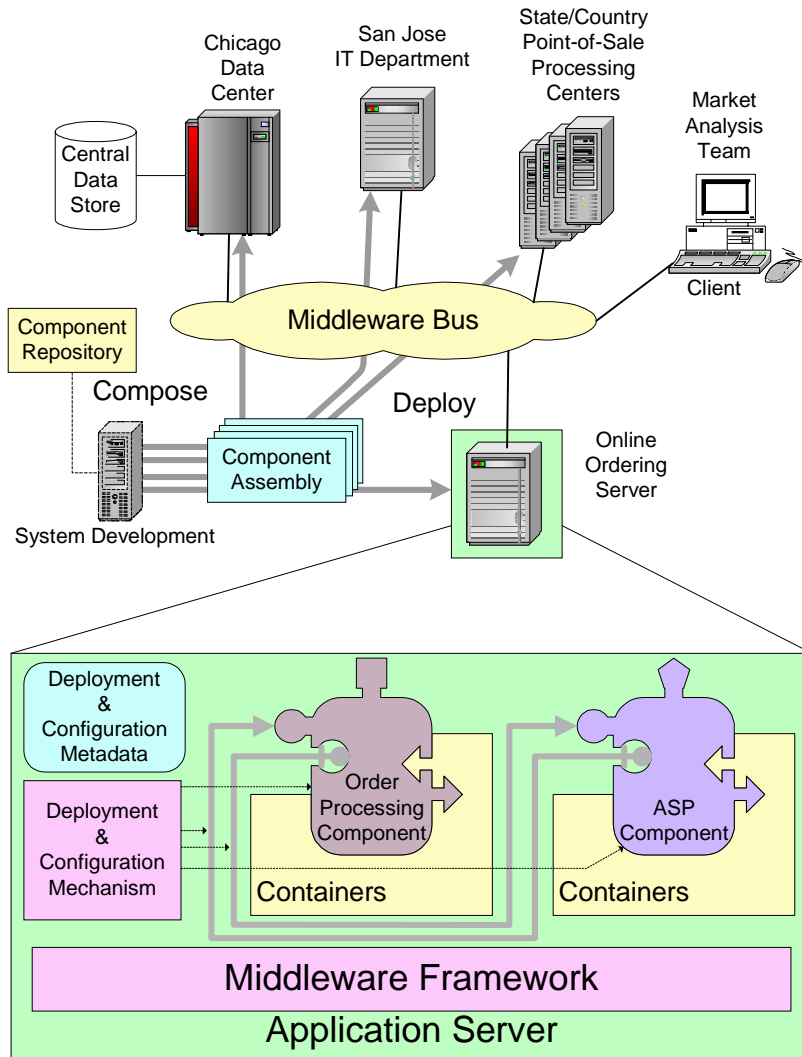
Promising Solution: Component Models



- 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

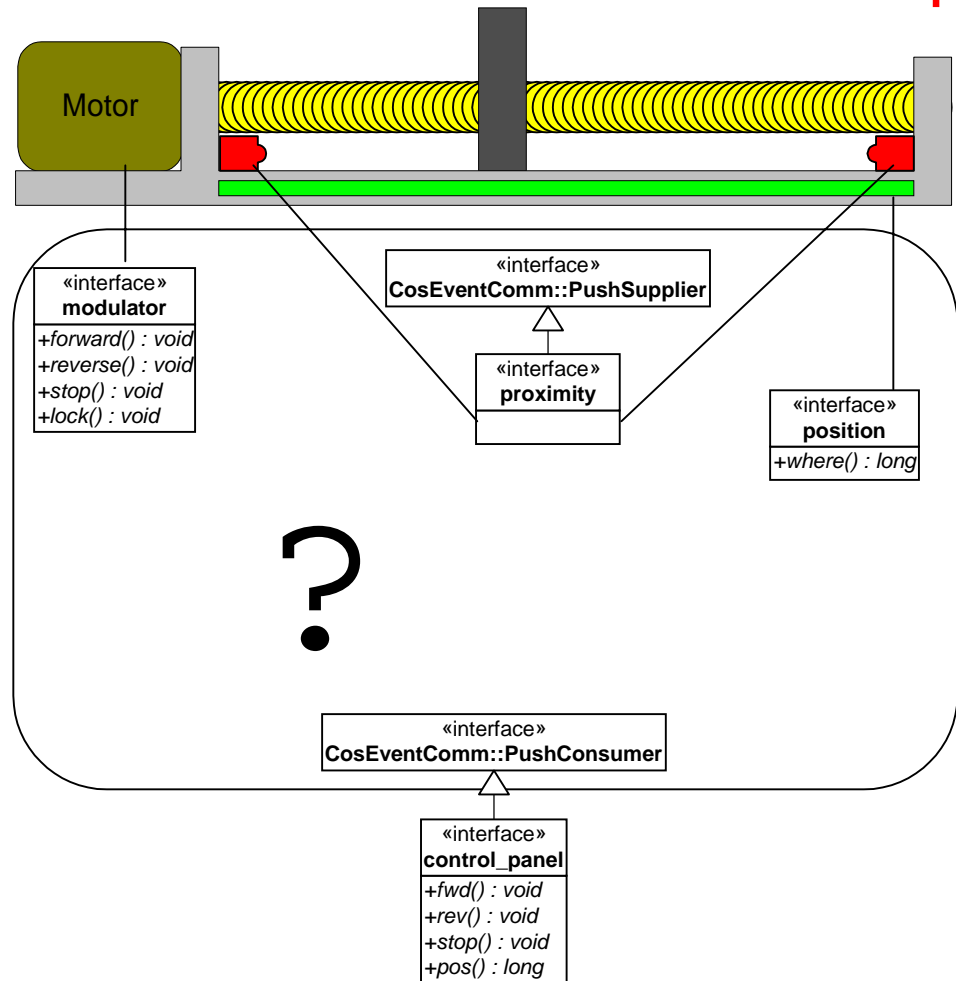
•J2EE (EJB), DCOM & CORBA Component Model

The CORBA Component Model (CCM)



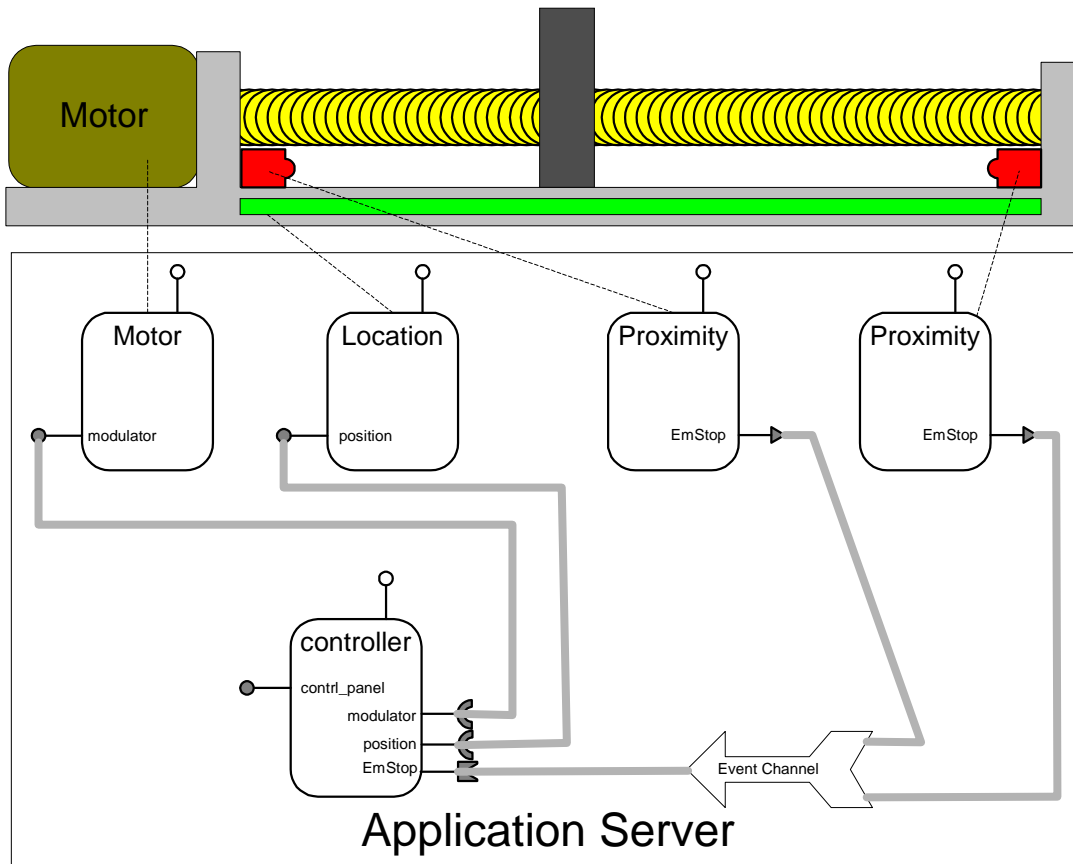
- Extends the CORBA Object Model
- Provides standard run-time environment for components
 - application servers
 - containers
- Uses metadata to describe
 - application server and container configurations
 - component run-time requirements, e.g., transactional, persistence state
 - component configuration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

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

```
<!-- Associate components with impls -->
<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

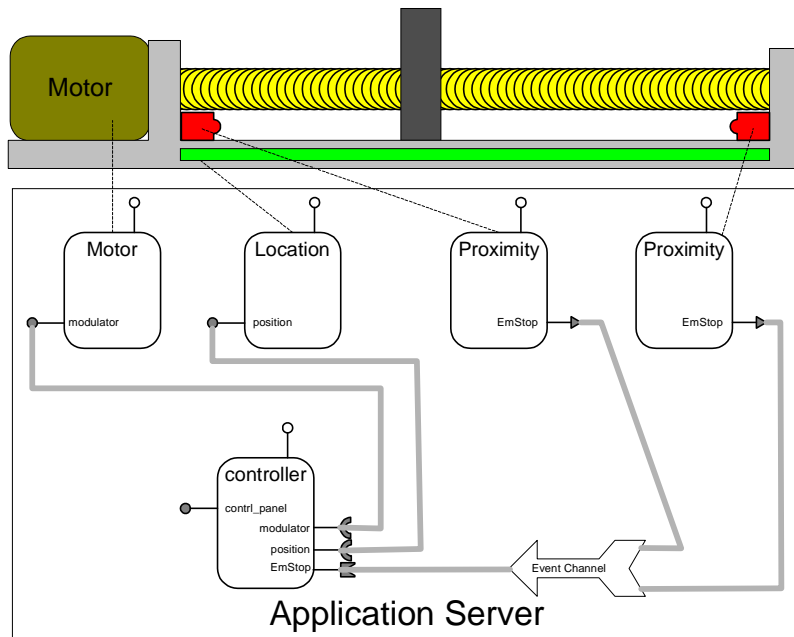
After CCM (cont.) – Component Instantiations

```
<!-- Instantiating component homes/instances -->
<partitioning>
  <processcollocation>
    ...

    <homeplacement id="ProximityHome">
      <componentfileref idref="Proximity"/>
      <componentinstantiation id="left">
        <componentproperties>
          <fileinarchive name="left-switch.cpf"/>
        </componentproperties>
      </componentinstantiation>
      <componentinstantiation id="right">
        <componentproperties>
          <fileinarchive name="right-switch.cpf"/>
        </componentproperties>
      </componentinstantiation>
    </homeplacement>
    ...
  </processcollocation>
</partitioning>
```

- 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

After CCM (cont.) – Connecting Components



```

<connections>
  ...
  <connectinterface>
    <usesport>
      <usesidentifier>modulator</usesidentifier>
      <componentinstantiationref idref="Motor"/>
    </usesport>
    <providesport>

<providesidentifier>modulator</providesidentifier>
  <componentinstantiationref idref="Controller"/>
</providesport>
</connectinterface>
  <connectevent>
    <consumesport>
      <consumesidentifier>EmStop</consumesidentifier>
      <componentinstantiationref idref="Controller"/>
    </consumesport>
    <publishesport>

<publishesidentifier>EmStop</publishesidentifier>
  <componentinstantiationref idref="left"/>
  </publishesport>
</connectevent>
  ...
</connections>

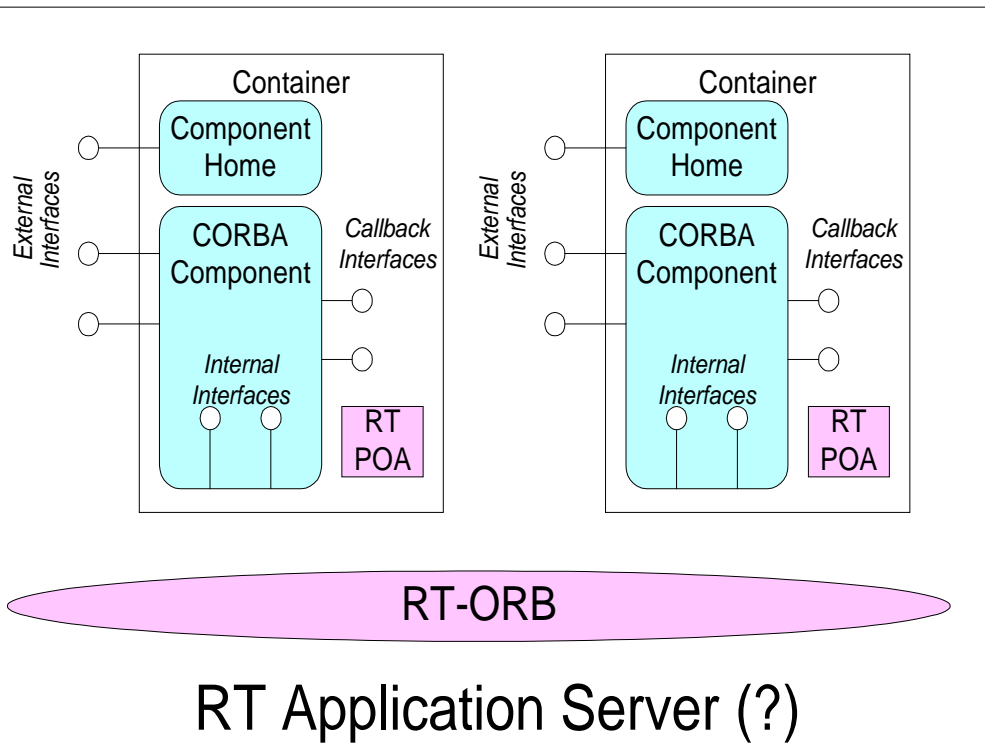
```

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

RTCCM \neq CCM + RTCORBA

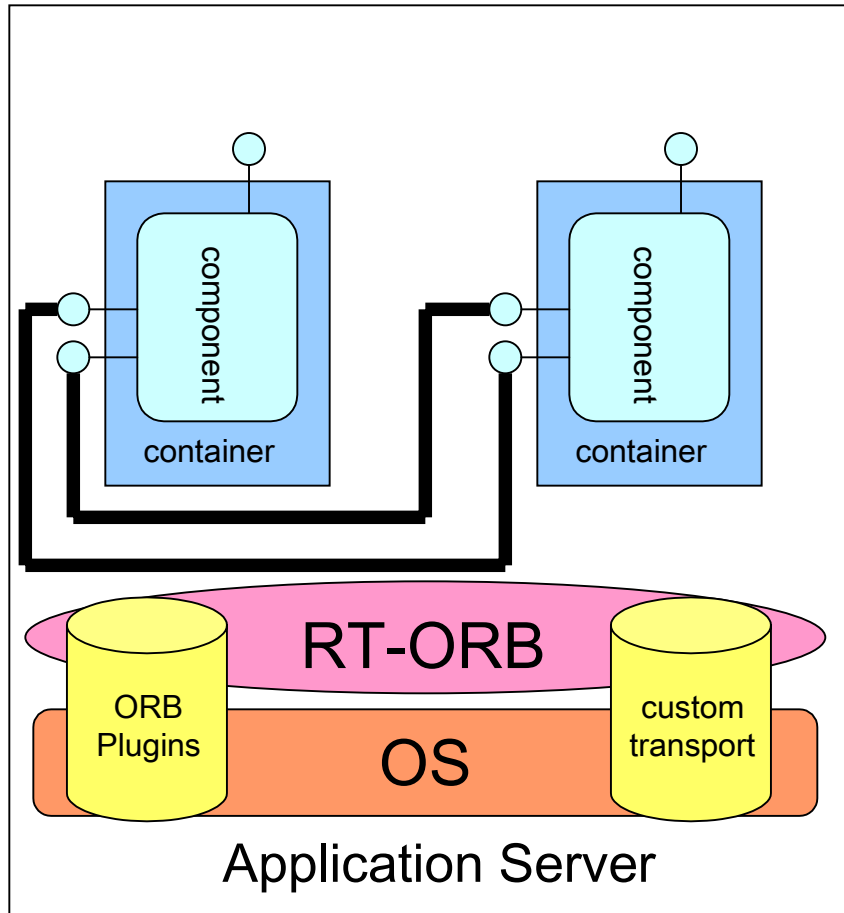
Why doesn't running a RTORB
beneath CCM make it an
RTCCM implementation?

- 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)

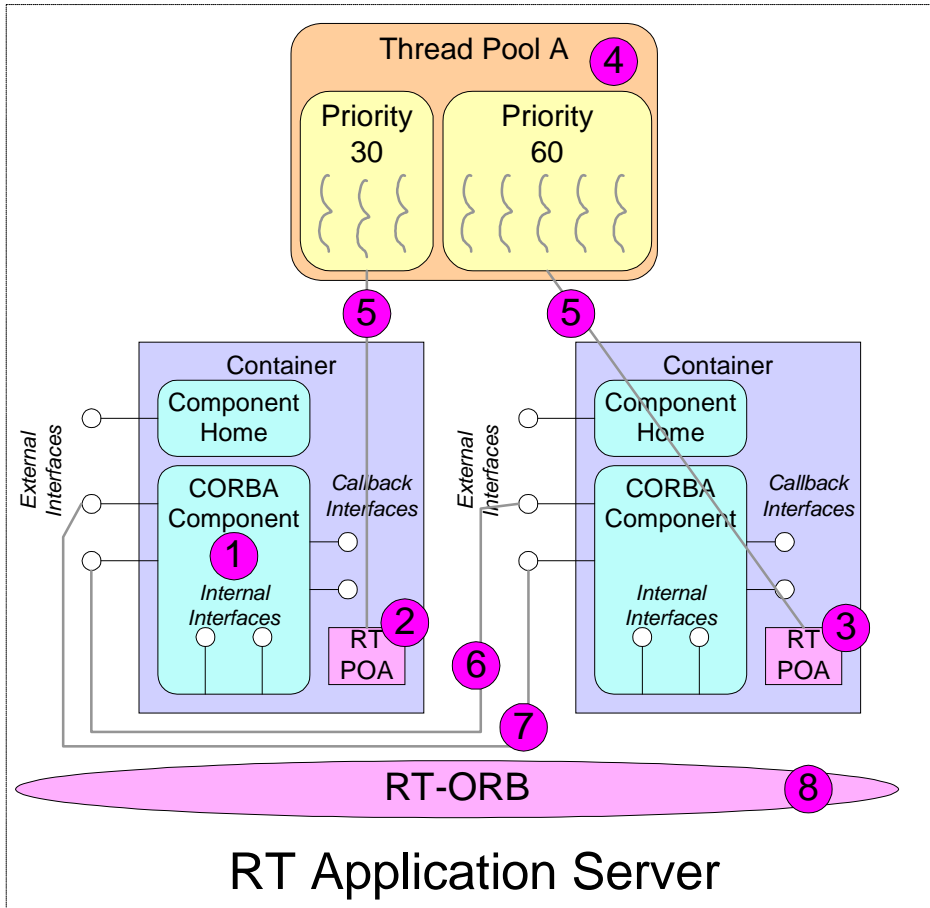
- Solution: Configure RT-policies/mechanisms using CCM's metadata



- 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

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
 - Custom protocols
 - Priority mappings

RTCCM Descriptor Examples

```
<corbacomponent>
...
<ciao:prioritymodelpolicy name="priority_model_policy"
value="SERVER_DECLARED"
priority="20"/>
...
</corbacomponent>
```

```
<partitioning>
  <processcollocation>
    ...
    <homeplacement id="ProximityHome">
      <componentfileref idref="Proximity"/>
      <componentinstantiation id="left">
        <componentproperties>
          <fileinarchive name="left-switch.cpf"/>
          <ciao:rtprioritylevel value="60"/>
          <ciao:rtprioritymodel value="CLIENT_PROPAGATED"/>
        </componentproperties>
      </componentinstantiation>
      <componentinstantiation id="right">
        <componentproperties>
          <fileinarchive name="right-switch.cpf"/>
          <ciao:rtprioritylevel value="60"/>
          <ciao:rtprioritymodel value="CLIENT_PROPAGATED"/>
        </componentproperties>
      </componentinstantiation>
    </homeplacement>
    ...
  </processcollocation>
</partitioning>
```

- 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.)

```

<componentassembly>
  ...
  <partitioning>
    <ciao:createthreadpool id="tp_a" priority="30"
                          number="10"/>
    <ciao:createthreadpoolwithlane id="tp_l">
      <ciao:prioritylevel lane="30" number="3"/>
      <ciao:prioritylevel lane="60" number="3"/>
      <ciao:prioritylevel lane="80" number="10"/>
    </ciao:createthreadpool>
  </partitioning>
</componentassembly>

```

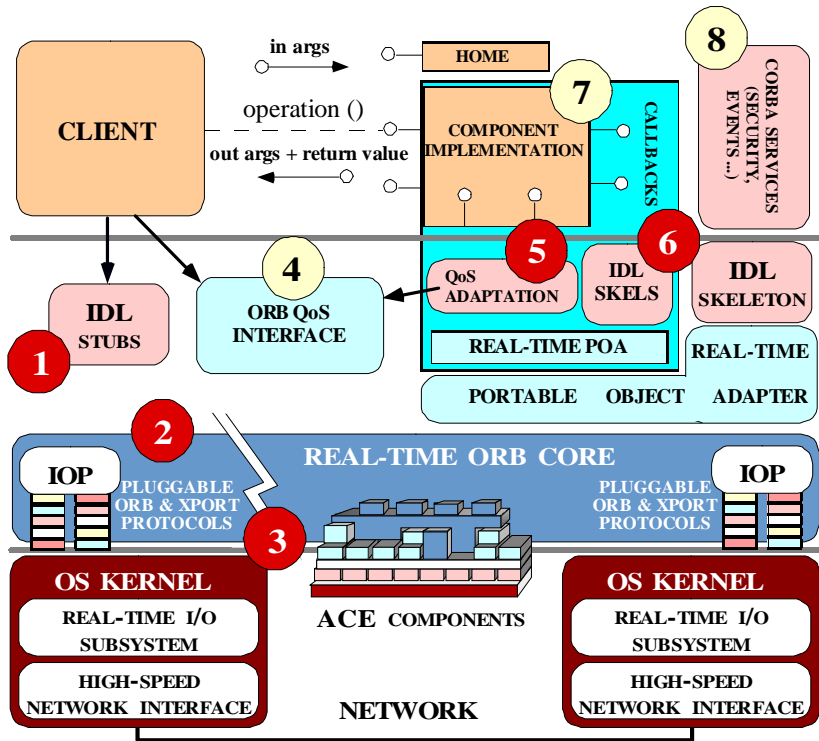
```

<homeplacement id="ProximityHome">
  <componentfileref idref="Proximity"/>
  <componentinstantiation id="left">
    <componentproperties>
      <fileinarchive name="left-switch.cpf"/>
      <ciao:usethreadpool idref="tp_a"/>
    </componentproperties>
  </componentinstantiation>
  <componentinstantiation id="right">
    <componentproperties>
      <fileinarchive name="right-switch.cpf"/>
      <ciao:usethreadpoolwithlane idref="tp_l"
                                lane="60"/>
    </componentproperties>
  </componentinstantiation>
</homeplacement>

```

- Define thread pools
- Define QoS aggregates
- Associate thread pools with components
- Associate QoS aggregates with connections

Current Status of CIAO



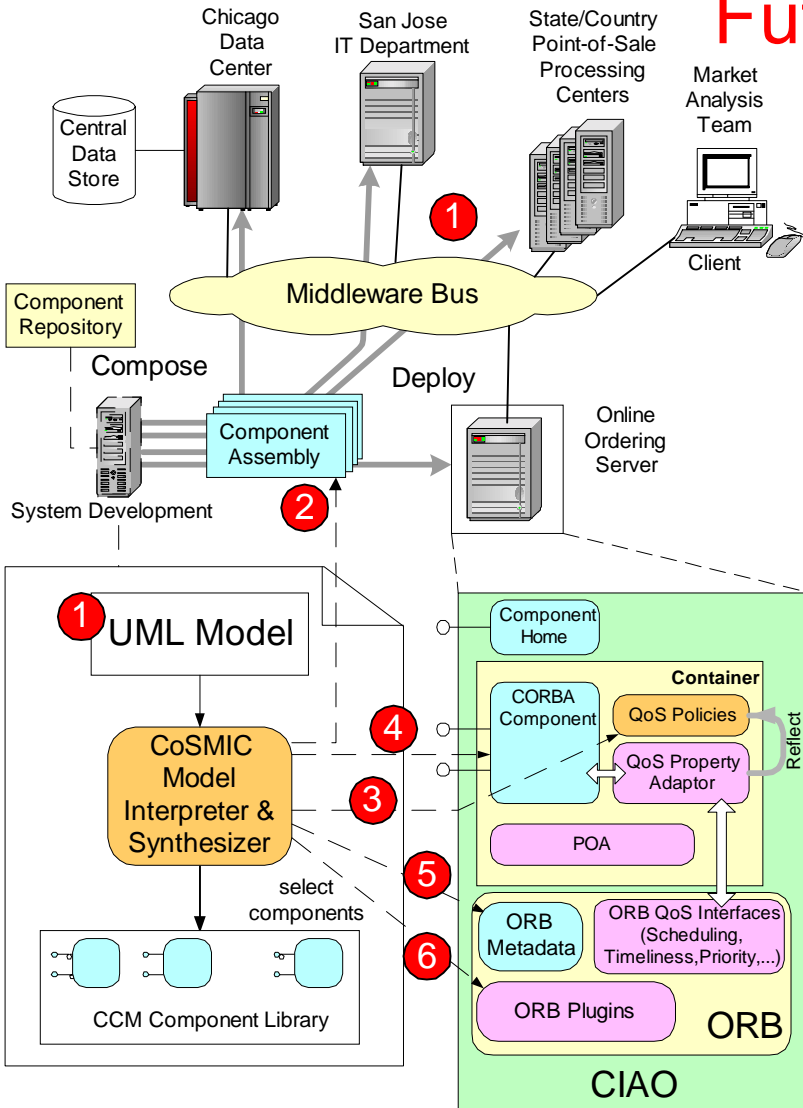
- 1) COLLOCATION STUBS
- 2) COLLOCATION XPORT SELECTION
- 3) SHARED MEMORY XPORT
- 4) ORB LEVEL QoS INTERFACE
- 5) COMPONENT QoS ADAPTATION
- 6) DYNAMIC LINKING OF COMPONENT SERVANTS
- 7) DYNAMIC CONFIGURATION OF COMPONENTS
- 8) INTEGRATION OF SERVICES

- *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

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
- $RTCCM \neq CCM + RTCORBA$
- 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