F6COM: A Case Study in Extending Container Services through Connectors

Abhishek Dubey, Andy Gokhale, Gabor Karsai, ***William R. Otte***;
Vanderbilt University/ISIS
Johnny Willemsen; Remedy IT
Paul Calabrese, Adam Mitz; OCI

This work was supported by the DARPA System F6 Program under contract NNA11AC08C.

Future Fast, Flexible, Fractionated, Free-Flying Spacecraft (F6)

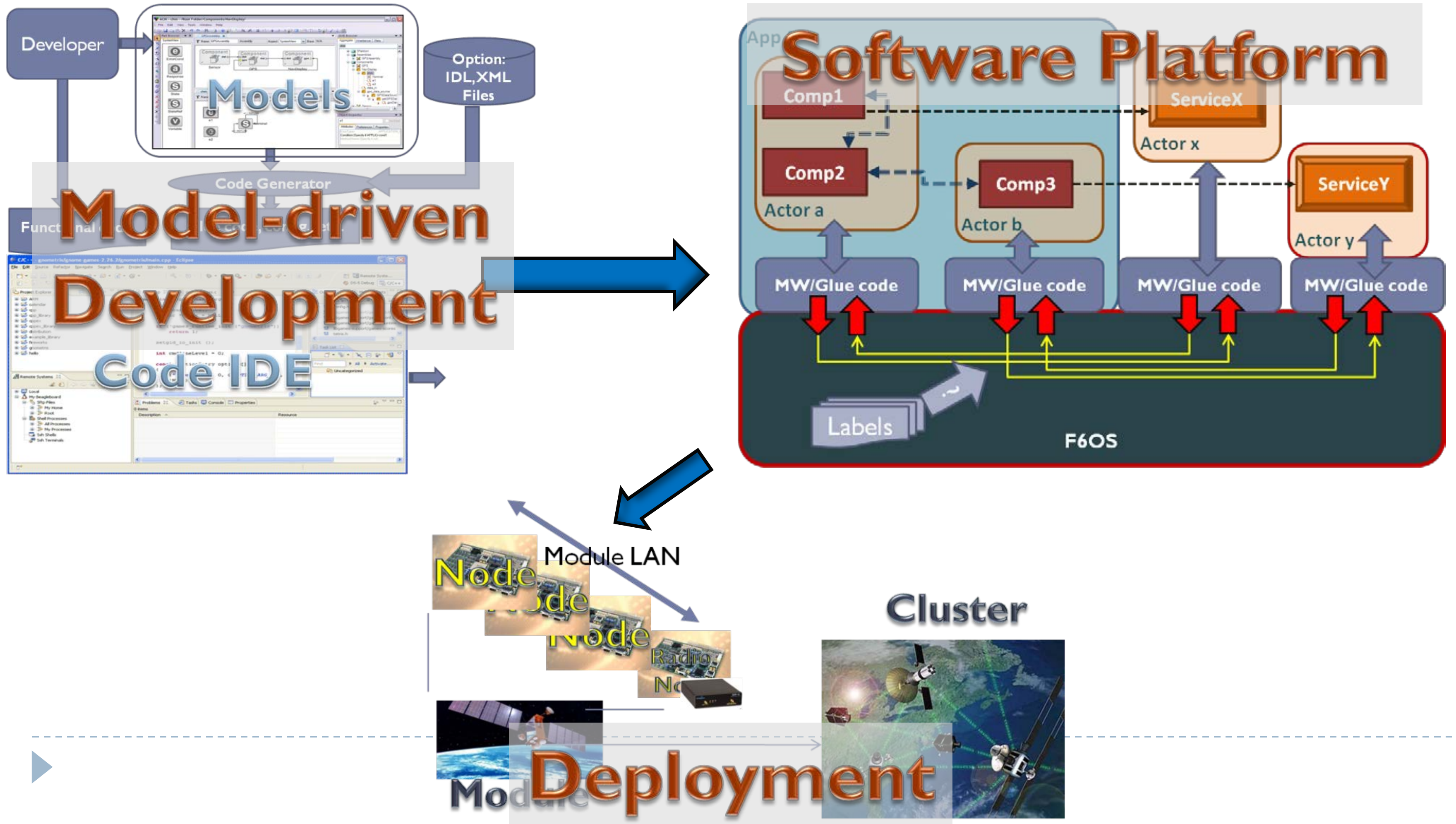
- Objective: Develop and demonstrate a satellite architecture where the functionality of a traditional monolithic spacecraft is replaced by a cluster of wirelessly connected modules
- Advantages:
  - Increased flexibility during design and acquisition
  - Reduced development and launch costs
  - Increased adaptability and survivability of space systems on-orbit
  - Potential to apply economies of scale to satellite design & manufacture
- Key program objective is the promulgation of open interface standards for hardware and software.

Challenges

1. Distributed system with network addressability
   - Everything and anything (modulo security permissions) can be accessed and addressed
2. Highly variable network quality & availability
   - Inter-satellite links are highly unreliable with unpredictable bandwidth; ground links are infrequent and flow
3. Dynamism
   - Dynamically deployed applications, security configurations, and cluster architectures
4. Resource sharing
   - Specific resources can be shared across applications: CPU, communication links, memory, services
5. Fault tolerance
   - Faults in components, services, communication links, computing nodes are detected, isolated, and their effects mitigated
6. Multi-level security
   - The architecture shall address the requirements of MLS

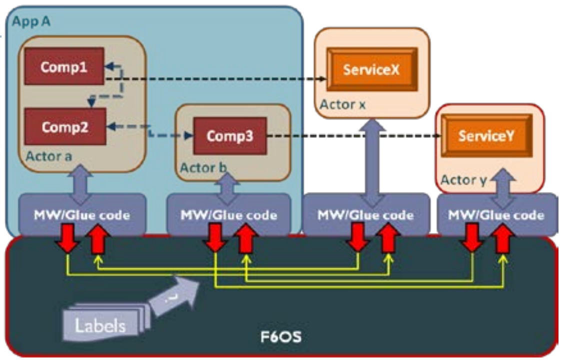
Solution: F6MDA (Model-driven Architecture)

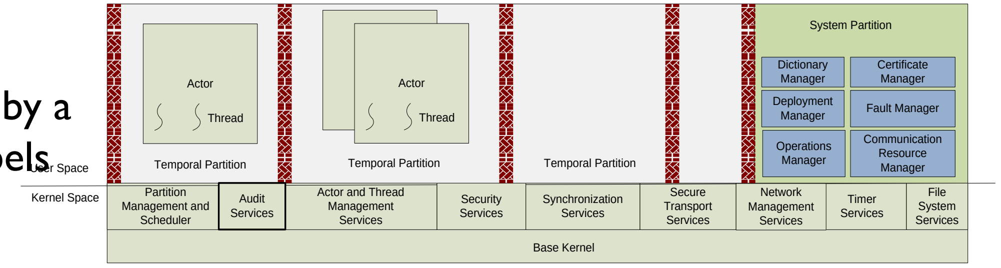
Layered architecture supported by a model-driven development toolchain

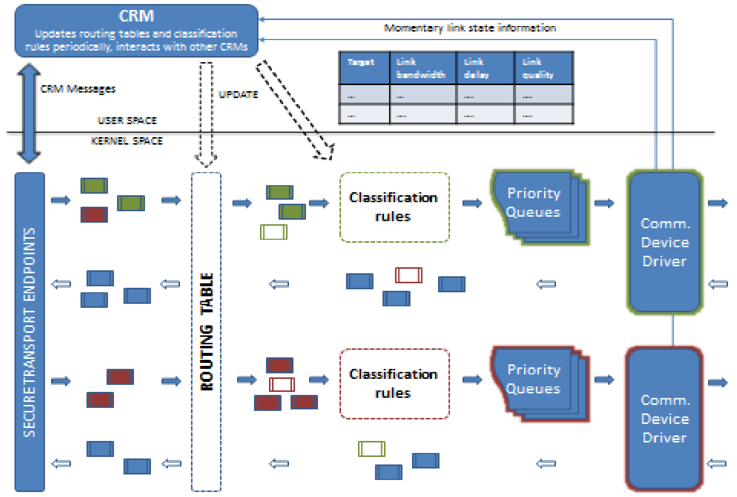


Solution: F6OS

- The 'Operating System' that provides
  - Restricted OS calls for application actors
  - Privileged calls for platform ('service') actors
  - All system calls are time-bounded
- Provides messaging services
  - All component interactions are via messages
  - No other interactions are possible
- All component interactions are facilitated by a 'secure transport' that verifies security labels on messages
- Resource management functions
  - CPU time: temporal partitioning for actors, utilization cap per actor within partition
  - Memory: space partitioning, limit caps
  - Network bandwidth: diffserv, bandwidth budget, differentiated routing
- F6OS is part of the TCB

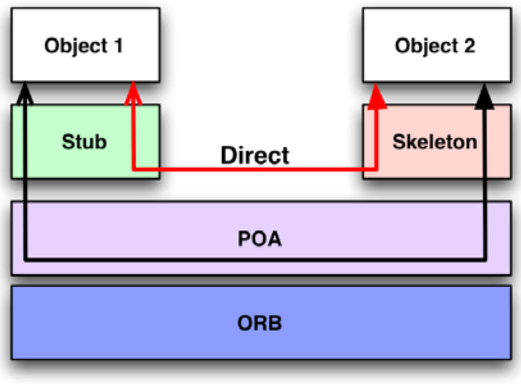


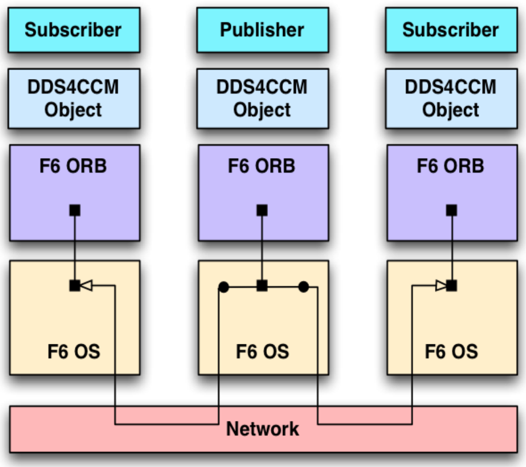




Solution: Middleware

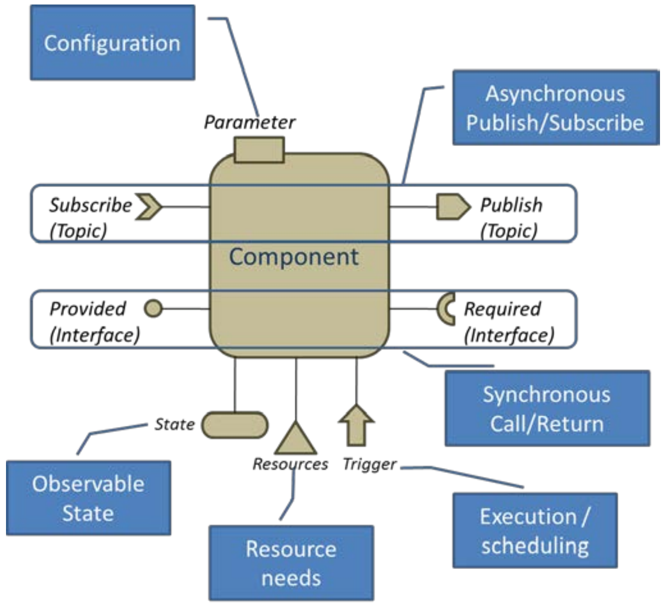
- The 'middleware layer' that provides:
  - Synchronous and asynchronous point-to-point communication with call/response semantics (→ Subset of CORBA RMI)
    - Location transparency
    - Request (de)multiplexing
    - Message (de)marshalling
    - Error handling
    - Support for QoS (client timeouts, reliable one-ways)
  - Anonymous publish/subscribe communications with one/many-to-many data distribution patterns (→ Subset of DDS)
    - Datatype specification
    - Static discovery
    - Selected QoS: reliability, time-based filtering, latency budget, etc.

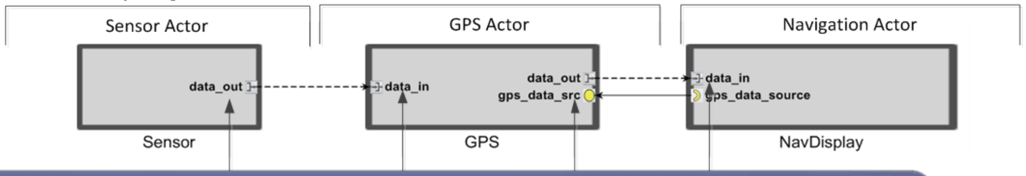




Solution: F6 Component Model (F6COM)

- Applications are constructed from interacting components
- Components interact and exchange data via ports
- Publishers are decoupled from subscribers, publishers can send data at anytime, subscribers are triggered when data is available / can poll data
- Provided and required interfaces are connected and support tightly and loosely coupled, synchronous interactions with call/response semantics
- Component state may be exposed (accessible to the middleware) for fault tolerance
- Components request resources (memory) only from and through the middleware
- Components are triggered (scheduled and released) by the middleware – single-threaded
- Components have >=1 ports
- Applications ➔ Component assemblies





Our Implementation of F6COM has been derived from LwCCM, specifically CIAO, as a test case for UCM concepts.

F6COM Goals (1/2)

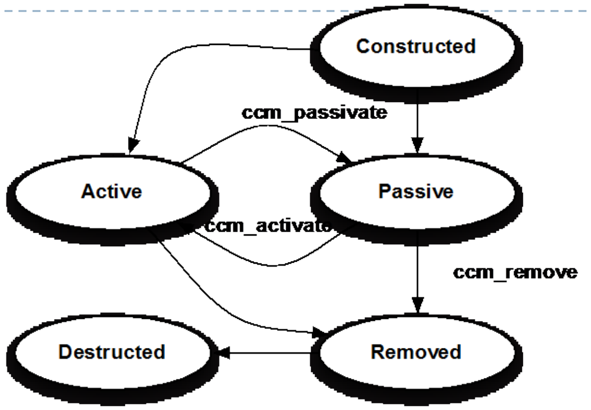
- Support for varied interaction semantics
  - *CCM Deficiency:* Architecture presumes heavy CORBA reliance; standard port types are insufficient to express complex interactions
  - *Solution Approach:* Eliminate events & RMI; leverage the connector concept and extended port types from DDS4CCM specification to implement all distribution middleware interactions
- Support container extensibility
  - *CCM Deficiency*: Core specification suggests at worst a monolithic container, at best provides few standard interfaces through which containers may be extended via QoS4CCM
  - *Solution Approach:* Leverage connectors to implement common container services; providing a well-defined deployment unit

F6COM Goals (2/2)

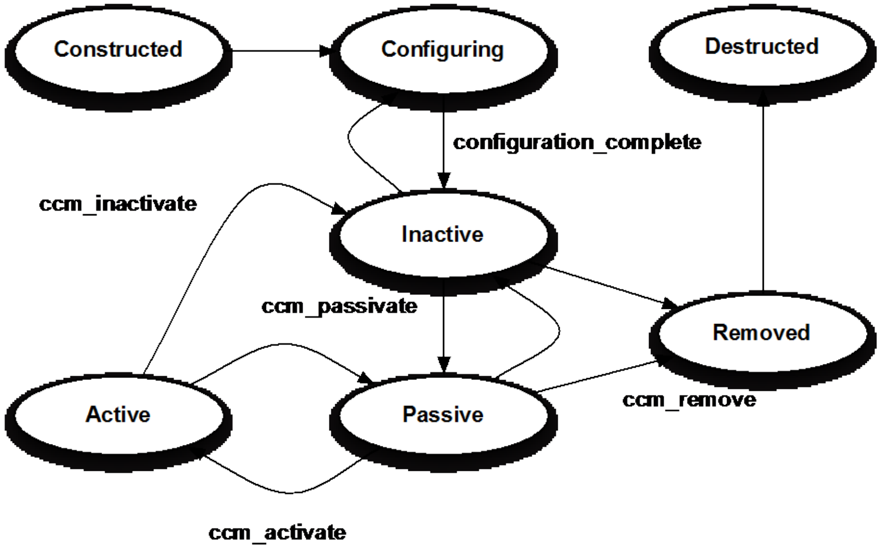
- Minimal runtime overhead
  - *Solution Approach:* Make container extremely lightweight, responsible only for component lifecycle control.
- Expressive Lifecycle Control
  - *CCM Deficiency:* Component lifecycle states are somewhat loosely defined, valid transitions & semantics are unclear
  - *Solution Approach*: Rigidly define valid transitions between lifecycle states and associated semantics.
- Strict control over threading guarantees
  - *CCM Deficiency:* Provides no control or guarantees about component threading; heavily dependent on middleware configuration.
  - *Solution Approach:* Provide mechanisms to control component operation scheduling; guaranteeing only a single operation active at any given time.

F6COM Lifecycle States

- Strong lifecycle model necessary to support robust fault tolerance strategies & redeployment capabilities
- **Configuring**: Component connections and configuration attributes may be changed
- **Inactive**: Components may not receive or produce operations/events.
- **Passive**: Components may receive events/operations, but may not produce external side effects.
- **Active**: Components may receive events/operations, and may produce outgoing events/operations.
- **Removed**: Components should release resource and prepare for removal.



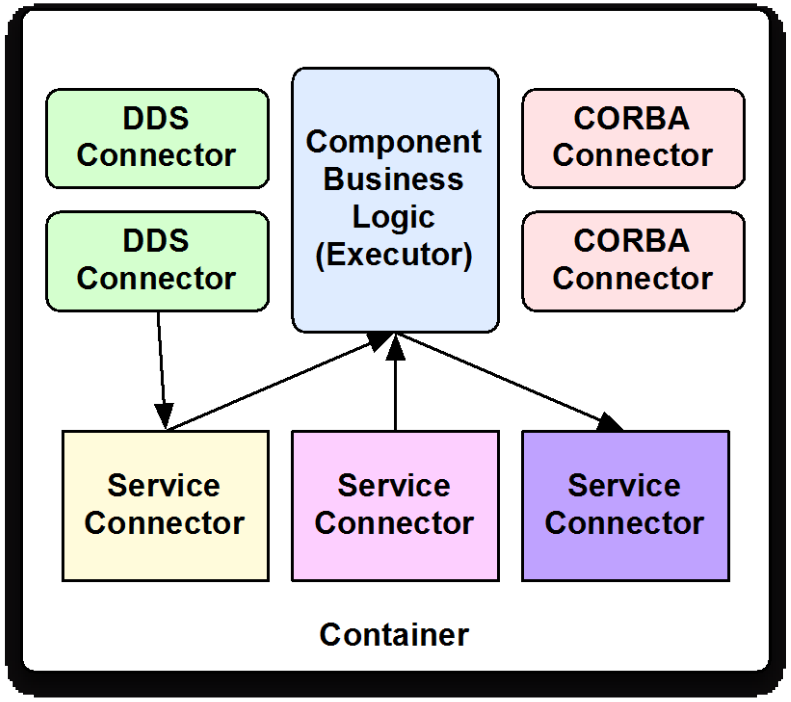
LwCCM Lifecycle



F6COM Lifecycle

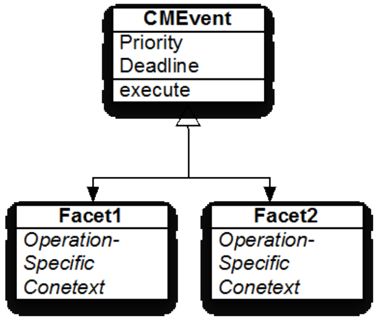
Service connectors

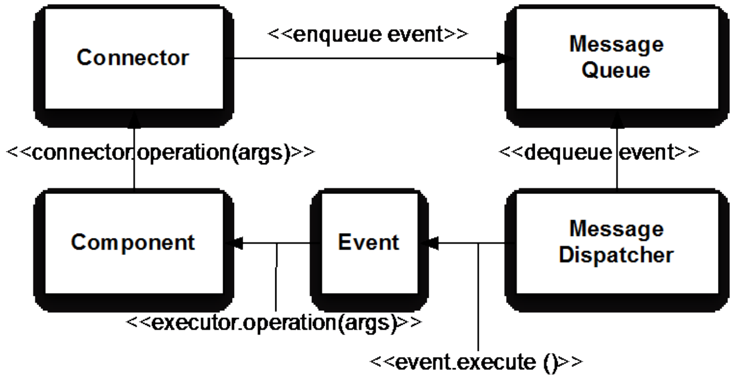
- Per-container singleton connector objects
- Similar to QoS4CCM QoSEnablers
- Provide deployable entity to customize container behavior
- May proactively interact with component/container executors
  - Contrast with QoS4CCM interceptors, which are only activated on incoming/outgoing request
- Used to provide 'services' to connectors, components, or both
- Examples:
  - Component Message Queues
  - Timer service
  - Container-level fault management



Component Message Queue (CMQ)/Component Message Dispatching (CMD)

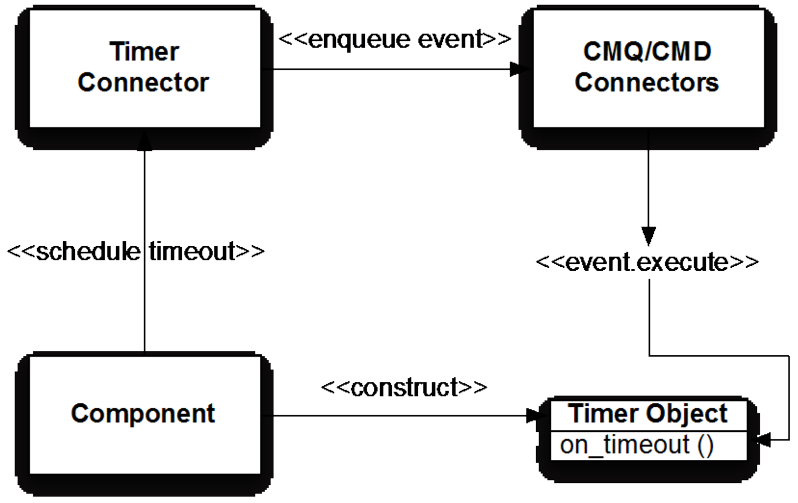
- CMQ and CMD are *Service Connectors*
- Connectors encapsulate incoming operations in operation-specific *Event Objects.*
  - *Event Objects* are operation-specific
  - *Event Objects* have priorities and deadlines
- *Event Objects* are queued in the CMQ.
  - CMQ performs admittance control based on deadline
  - CMQ schedules operations based on priority
- CMD retrieves events and executes them in the context of the component executor
  - CMD has configurable threading policies
  - CMD performs deadline monitoring for executed events
- Allows queuing discipline & threading policy to be varied independently
- Components need not be aware of operation scheduling, or the presence of the CMQ abstraction





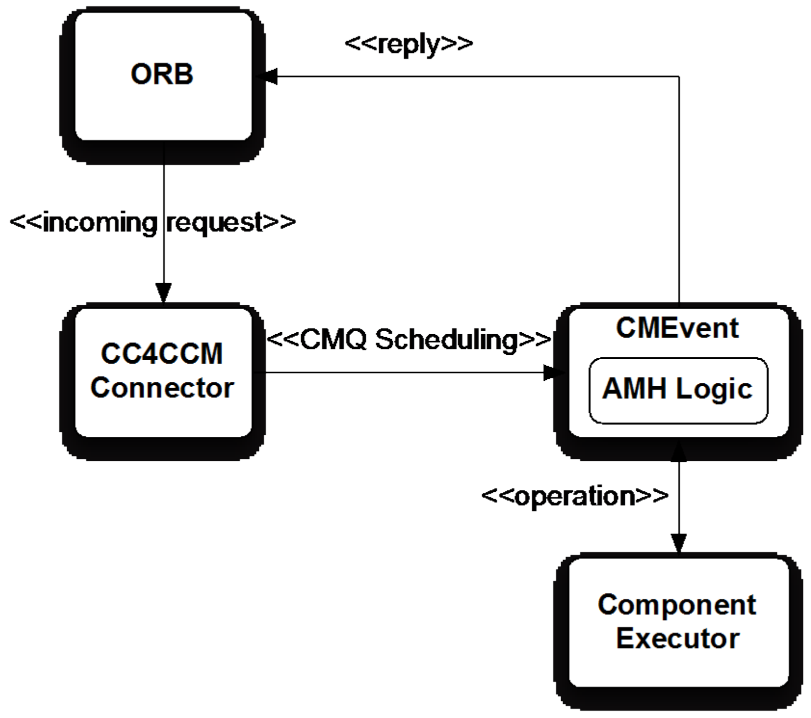
Timer Connector (Timer4CCM)

- A *Service Connector* that allows components to request timed activation
  - *Sporadic Activation*, i.e., one shot timers
  - *Periodic Activation,* i.e., recurring timers
- Component Developers implement callback objects
- Callback objects are scheduled for later activation
- Timer callbacks are scheduled for activation in the CMQ after expiration



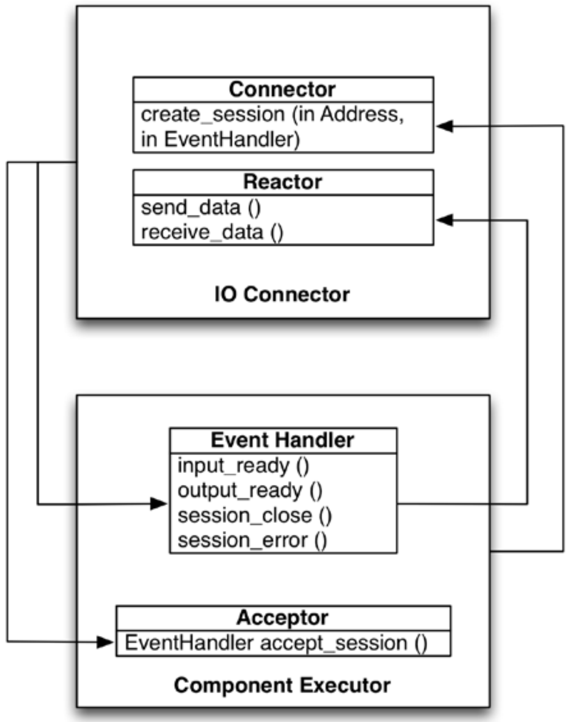
CORBA Connectors (RMI4CCM/AMI4CCM)

- Traditional CCM embeds CORBA deeply in the component container
- New RMI4CCM and standardized AMI4CCM connectors alleviate this
- Existing implementation uses CORBA Asynchronous Method Handling (AMH) to encapsulate requests in *Event Objects*.
- Future implementations could replace CORBA with DDS RMI



IO4CCM: Socket Communication for Components

- Components may require the ability to interact with legacy systems
  - Socket/stream based (TCP)
  - Socket/datagram based (UDP)
  - Device drivers (ioctl)
- Provides a Reactor-based abstraction to component business logic
  - Components need not run threads/implement pernicious low-level interaction code
  - Allows integration with CMQ/CMD infrastructure
- Components accept/establish connections using the Acceptor/Connector ports
- IO Connector notifies component of available data via Event Handler



Lessons Learned

- Connectors provide a powerful mechanism to implement and extend container services
  - Threading/Operation Scheduling
  - Timers
  - (*Future*) Fault mitigation/management
- Connectors are an effective mechanism to integrate many disparate interaction styles and communication middlewares into a single container
  - DDS4CCM
  - CC4CCM/AMI4CCM
  - Socket-style communication for legacy systems
- Deployment & Configuration (D&C) tooling remains tightly coupled with CORBA legacy of CCM

F6COM and F6D&C are derived from CIAO and DAnCE, and will eventually be open source.