iCCM: A Framework for Servant-based Integration of DDS into the CORBA Component Model
The Data Distribution Services (DDS) is an OMG specification for creating real-time publisher-subscriber applications.

- Provides mechanisms for associating quality-of-service (QoS) properties with key entities involved in a publisher-subscriber application.
- QoS configurations can ensure certain levels of performance guarantees.
Client is responsible for managing many DDS entities
- e.g., Publisher, DataReader, DataWriter, & etc.

No clear separation of concerns
- e.g., deployment vs. configuration vs. runtime

Client must interact with many low-level constructs
Key entities in a CORBA Component Model (CCM) component:

- **Component Servant** – Responsible for hosting a component implementation
- **Context** – Allows a component to send events to other components
- **Component Implementation** – The component implementation is the core “business-logic” of the component
Three approaches for integrating DDS into the CORBA Component Model (CCM):

1. **Direct Integration** – DDS integrated at the component implementation level
2. **Servant-based Integration** – DDS integrated at the servant/context level
3. **Connector/Gateway Integration** – External mediator responsible for transforming “CORBA” event to DDS event

Fig. Points for integrating DDS into a CCM component.
The DDS₄CCM Specification

DDS₄CCM...
- is an OMG specification for integrating DDS into CCM
- uses connector-based approach for integrating DDS into CCM

Design Evaluation
- Uses DDS event types (i.e., structs) instead of CORBA event types
- Hard to swap different component implementations
- Requires modification to IDL₃ grammar (e.g., template modules, port types, etc.)

In the end, DDS₄CCM does not abstract the communication model to work with CCM. Instead, DDS₄CCM modified CCM to work with DDS.
iCCM: Servant-based Integration

Integrated CCM (iCCM)...

- is a framework & architecture for integrating different middleware technologies into CCM
- uses servant-based approach for realizing integration

iCCM Design Goals

- Transparent interchanging of different architectures
  - *e.g.*, OpenSplice, RTI-DDS, OpenDDS, etc.
- Map existing IDL3 constructs to target architecture entities
- Configuration at deployment time
Context has a collection of publisher & emits entities for sending events to other components

The servant has a collection of event consumer entities for receiving events from other components

Need to map the publisher, emits, & event consumer entities to its corresponding entity in the DDS, or visa versa
Mapping DDS onto CCM in iCCM

- **DomainParticipant** – Mapped to a component in CCM
- **DataReader** – Mapped to an event consumer in CCM
- **DataWriter** – Mapped to an emits or publishers port in CCM
- **Publisher** – Configuration entity for grouping emits/publishes ports with same QoS properties
- **Subscriber** – Configuration entity for grouping EventConsumers with same QoS properties
module Components
{
/**
 * Base class for all component event consumer.
 */
interface ::Components::EventConsumerBase
{
    // Method for sending an event.
    void push_event (in EventBase evt) raises (BadEventType);
};
};

- Provided by input port to determine what output ports of a component communicates with it using a CORBA event
- EventConsumer entities are the abstractions that represent the connection
  - e.g., sender component uses connected event consumer to send an event
- Setup by deployment & configuration tools

How can we leverage the EventConsumer when integrating DDS into iCCM?
The DDS EventConsumer

```c++
module DDS {
/**
 * Extension interface for connecting to subscriber using CCM
 * component connections. The event consumer exposes methods that
 * control what topics it listens to.
 */
interface EventConsumer : ::Components::EventConsumerBase {
    void add_topic (in string topic_name);
    void remove_topic (in string topic_name);
};
};
```

- Publishes/emits port connect to an event consumer & determine topic name
  - local – component instance name + port name
  - global – port name
  - custom topic – user-defined name at deployment time
- `add_topic` & `remove_topic` manage the lifecycle of DDS event listeners for the corresponding topic
Component implementation can receive events on same port for different DDS topics.
DDS events are structs in IDL & CCM events are eventtypes (i.e., object-by-value types)

```
// DDS event
struct ShapeType
{
    string color; //@key
    long x;
    long y;
    long shapesize;
};

// CCM event
eventtype ShapeEvent
{
    public string color;
    public long x;
    public long y;
    public long shapesize;
};
```
DDS events are structs in IDL & CCM events are eventtypes (i.e., object-by-value types)

**Solution:** iCCM uses preprocessor statements to map DDS events to CCM events

i.e., #pragma dds2ccm

```c
// DDS event
struct ShapeType
{
    string color; //@key
    long x;
    long y;
    long shapesize;
};

// CCM event
eventtype ShapeEvent
{
    public string color;
    public long x;
    public long y;
    public long shapesize;
};

// DDS to CCM event mapping
#pragma dds2ccm (ShapeType, ShapeEvent)
```
DDS events are structs in IDL & CCM events are eventtypes (i.e., object-by-value types)

Solution: iCCM uses preprocessor statements to map DDS events to CCM events
  i.e., #pragma dds2ccm

iCCM compiler generates stub files that implement CCM events for the DDS event

Stub files are compiled with component skeleton and servant files.
// Upcall dds2ccm event
class ShapeEventUpcall :
  public virtual ShapeEvent,
  public virtual ::CORBA::DefaultValueRefCountBase
{
public:
  ShapeEventUpcall (ShapeEvent & dds_event);
  virtual ~ShapeEventUpcall (void);

  CORBA::Long x (void) { return this->dds_event_.x; }
  void x (CORBA::Long val) { this->dds_event_.x = val; }

  // other methods

private:
  // Reference to read DDS event
  ShapeEvent & dds_event_;
};

- Upcall event, which is an event forwarded from a servant to the implementation, is a lightweight wrapper around an existing DDS event
Upcall event, which is an event forwarded from a servant to the implementation, is a lightweight wrapper around an existing DDS event.
// Downcall dds2ccm event
class ShapeEventDowncall :
    public virtual ShapeEvent,
    public virtual ::CORBA::DefaultValueRefCountBase
{
public:
    ShapeEventDowncall (void);
    virtual ~ShapeEventDowncall (void);

    CORBA::Long x (void) { return this->dds_event_.x; }
    void x (CORBA::Long val) { this->dds_event_.x = val; }

    // other methods
private:
    // Instance of a DDS event
    ShapeEvent dds_event_;  
};

- Downcall event, which is an event forwarded from the implementation to the servant, is a lightweight wrapper around a new DDS event
// Downcall dds2ccm event
class ShapeEventDowncall :
  public virtual ShapeEvent,
  public virtual ::CORBA::DefaultValueRefCountBase {

  public:
    ShapeEventDowncall();
    virtual ~ShapeEventDowncall();

    CORBA::Long x (void) { return this->dds_event_.x; }
    void x (CORBA::Long val) { this->dds_event_.x = val; }

    // other methods

private:
  // Instance of a DDS event
  ShapeEvent dds_event_;
Abstracting Communication Architecture

- Component implementations traditionally control the creation of a CCM event to be published to another component

- Current CCM implementations make it hard to switch between communication architectures
  - e.g., OpenSplice, RTI-DDS, OpenDDS, etc.

```c
void ShapeSender::ccm_activate (void) {
    // create new shape event
    ShapeEvent_var e (new ShapEvent());
    // set event attributes
    // send the event
    ctx->push_ColorChange (e.in ());
}
```
Abstracting Communication Architecture

- Component implementations traditionally control the creation of a CCM event to be published to another component.

- Current CCM implementations make it hard to switch between communication architectures.
  - e.g., OpenSplice, RTI-DDS, OpenDDS

- **Solution:** iCCM context controls event creation for each publishes & emits port.

```cpp
component ShapesComponent {
  publishes ShapeEvent ColorChange;
};

local interface
iCCM_ShapesComponent_Context : CCM_ShapesComponent_Context
{
  ShapeEvent new_ColorChange_event();
};

void ShapeSender::ccm_activate (void)
{
  // create new shape event
  ShapeEvent_var e (ctx->new_ColorChange_event());

  // update event attributes
  ctx->push_ColorChange (e.in());
}
```
Component implementations traditionally control the creation of a CCM event to be published to another component.

Current CCM implementations make it hard to switch between communication architectures e.g., OpenSplice, RTI-DDS, OpenDDS.

Solution: iCCM context controls event creation for each publishes & emits port.

The context, which is auto-generated by the iCCM compiler, controls what concrete event type is created.
Deployment & Configuration Metamodel

- Deployment & configuration (D&C) is managed by DAnCE, which is an implementation of the OMG D&C specification.
- ICCM uses a variation of the DDS QoS Modeling Language (DQML) to model DDS entity QoS properties.
Deployment & Configuration Metamodel

- Deployment & configuration (D&C) is managed by DAnCE, which is an implementation of the OMG D&C specification
- iCCM uses a variation of the DDS QoS Modeling Language (DQML) to model DDS entity QoS properties

Models are transformed into XML configuration files, and attached to its corresponding component implementation as a deployment & configuration property
In the DDS configuration file:

- Elements are used to capture the entity’s standard Qos parameters
- Attributes on the top-level element are for configuration management

### TopicQos

<table>
<thead>
<tr>
<th>Top-level Attribute Name</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the topic</td>
<td>Yes</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### PublisherQos

<table>
<thead>
<tr>
<th>Top-level Attribute Name</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the publisher</td>
<td>Yes</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### SubscriberQos

<table>
<thead>
<tr>
<th>Top-level Attribute Name</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the publisher</td>
<td>Yes</td>
<td>N/A</td>
</tr>
</tbody>
</table>
In the DDS configuration file:
- Elements are used to capture the entity’s standard QoS parameters
- Attributes on the top-level element are for configuration management

<table>
<thead>
<tr>
<th>Top-level Attribute Name</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Target output port</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Publisher</td>
<td>Name of parent publisher</td>
<td>No</td>
<td>@default</td>
</tr>
<tr>
<td>Topic</td>
<td>Name of topic</td>
<td>No</td>
<td>@default</td>
</tr>
<tr>
<td>IsInstance</td>
<td>Register an event instance</td>
<td>No</td>
<td>True</td>
</tr>
<tr>
<td>Scope</td>
<td>Local, global, or custom</td>
<td>No</td>
<td>global</td>
</tr>
</tbody>
</table>
In the DDS configuration file:
- Elements are used to capture the entity’s standard QoS parameters
- Attributes on the top-level element are for configuration management

**DataReaderQos**

<table>
<thead>
<tr>
<th>Top-level Attribute</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Target input port</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Subscriber</td>
<td>Name of parent subscriber</td>
<td>No</td>
<td>@default</td>
</tr>
<tr>
<td>Topic</td>
<td>Name of topic</td>
<td>No</td>
<td>@default</td>
</tr>
</tbody>
</table>
Deployment & Configuration Example

DDS Domain

DDSParticipantQoS
Deployment & Configuration Example

Domain Participant

Data Reader

Data Writer

Subscriber configuring EventConsumer ports

Subscriber configuring emits/publishes ports

DDSParticipantQoS
Deployment & Configuration Example

Connections validate QoS compatibility

DDSParticipantQoS
Deployment & Configuration Example

Declaration of XML file for configuring DDS entities

DDSParticipantQoS


- iCCM provides a template framework that simplifies integrating DDS (and other architectures in CCM)

Integration point for deployment tools (e.g., DAnCE)
iCCM Template Framework

- iCCM provides a template framework that simplifies integrating DDS (and other architectures in CCM)

```
namespace Servant_T <T> implements default CCM methods for a servant
```

```
Servant

Servant_T <T>

PublisherTable

PublisherTable_T <EVENT>

Publisher

Publisher_T <EVENT>

Consumer

Consumer_T <EVENT>
```

[Diagram showing the relationships between Servant, Servant_T, PublisherTable, PublisherTable_T, Publisher, Publisher_T, Consumer, Consumer_T, and how they implement default CCM methods for a servant.]
iCCM provides a template framework that simplifies integrating DDS (and other architectures in CCM)

- Servant
  - Servant_T <T>
    - PublisherTable
      - PublisherTable_T <EVENT>
    - Publisher
      - Publisher_T <EVENT>
    - Consumer
      - Consumer_T <EVENT>

Common implementation for arch-specific ports
iCCM Template Framework

- iCCM provides a template framework that simplifies integrating DDS (and other architectures in CCM)

![Diagram of iCCM Template Framework]

- Servant
  - Servant_T <T>
    - PublisherTable
      - PublisherTable_T <EVENT>
    - Publisher
      - Publisher_T <EVENT>
    - Consumer
      - Consumer_T <EVENT>

Event type specific implementation
Concluding Remarks

Lessons Learned

- Initial observations show iCCM is a less complex solution compared to DDS4CCM.
- iCCM makes it easier to swap component implementations with different communication architectures.
  - Must use the iCCM variation of the context.

Diagram:
- Component Implementation
  - Use/manage
  - OpenSplice Servant
  - RTI-DDS Servant
  - OpenDDS Servant
  - CoreDX Servant
  - Foo Servant
  - Bar Servant
Lessons Learned

- Initial observations show iCCM is a less complex solution compared to DDS4CCM
- iCCM makes it easier to swap component implementations with different communication architectures
  - Must use the iCCM variation of the context
- Improve event mapping for keys
  - Keys map to private fields
  - configure keys at deployment

```c
// DDS event
struct ShapeType
{
    string color; // @key
    long x;
    long y;
    long shapesize;
};

// CCM event
eventtype ShapeEvent
{
    private string color;
    public long x;
    public long y;
    public long shapesize;
};
```
Concluding Remarks

Future Research Directions

- Integration non-OMG architectures & technologies to improve portability
  - Portico (prototyped)
  - TRON/UPPAAL (prototyped)
  - OpenJAUS
  - SOAP/JSON
- Integration of heterogeneous architectures

iCCM is available in open-source format from the following location:

http://cuts.cs.iupui.edu
Questions