Tutorial: DDS first steps with Connector

Javier Povedano, PhD.

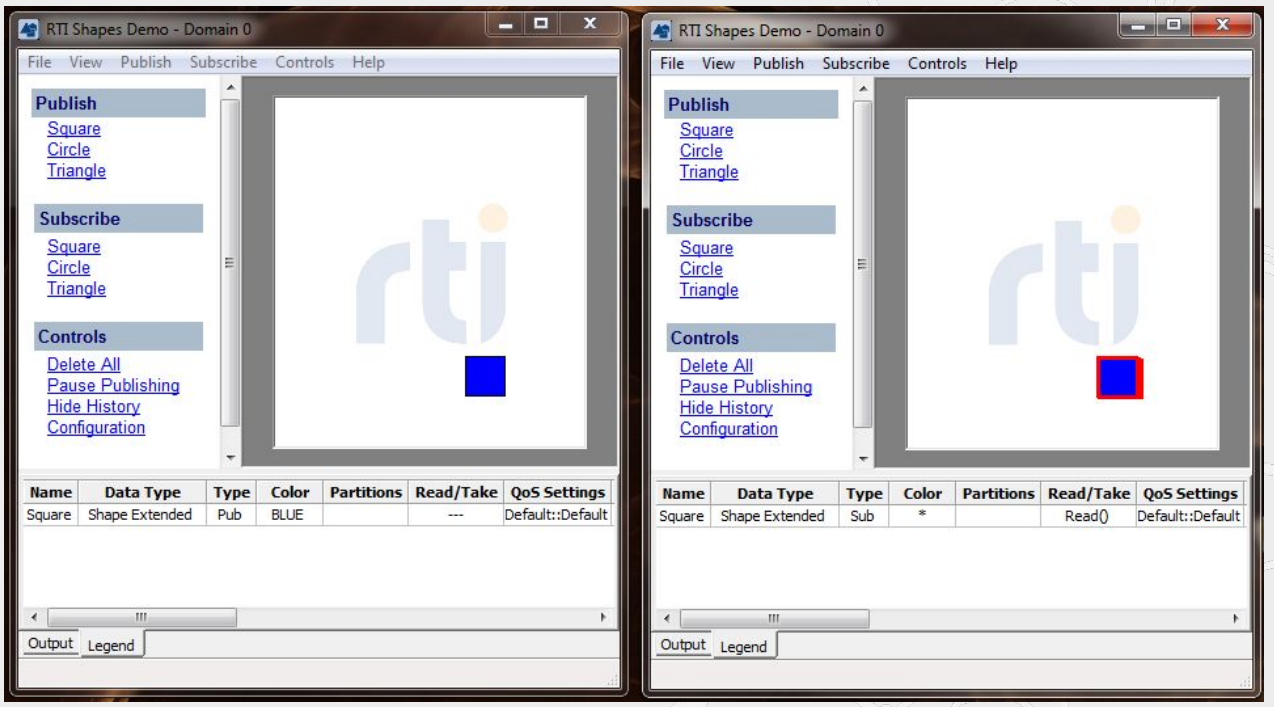
javier@rti.com

Real-Time Innovations Inc.

Agenda

- DDS
- Connector
- Setup your environment
- Hands On!
  - Publishing/Subscribing to Data
  - Data Filtering
  - High-Availability
  - Late joiners
  - Partitions

DDS Shapes Demo

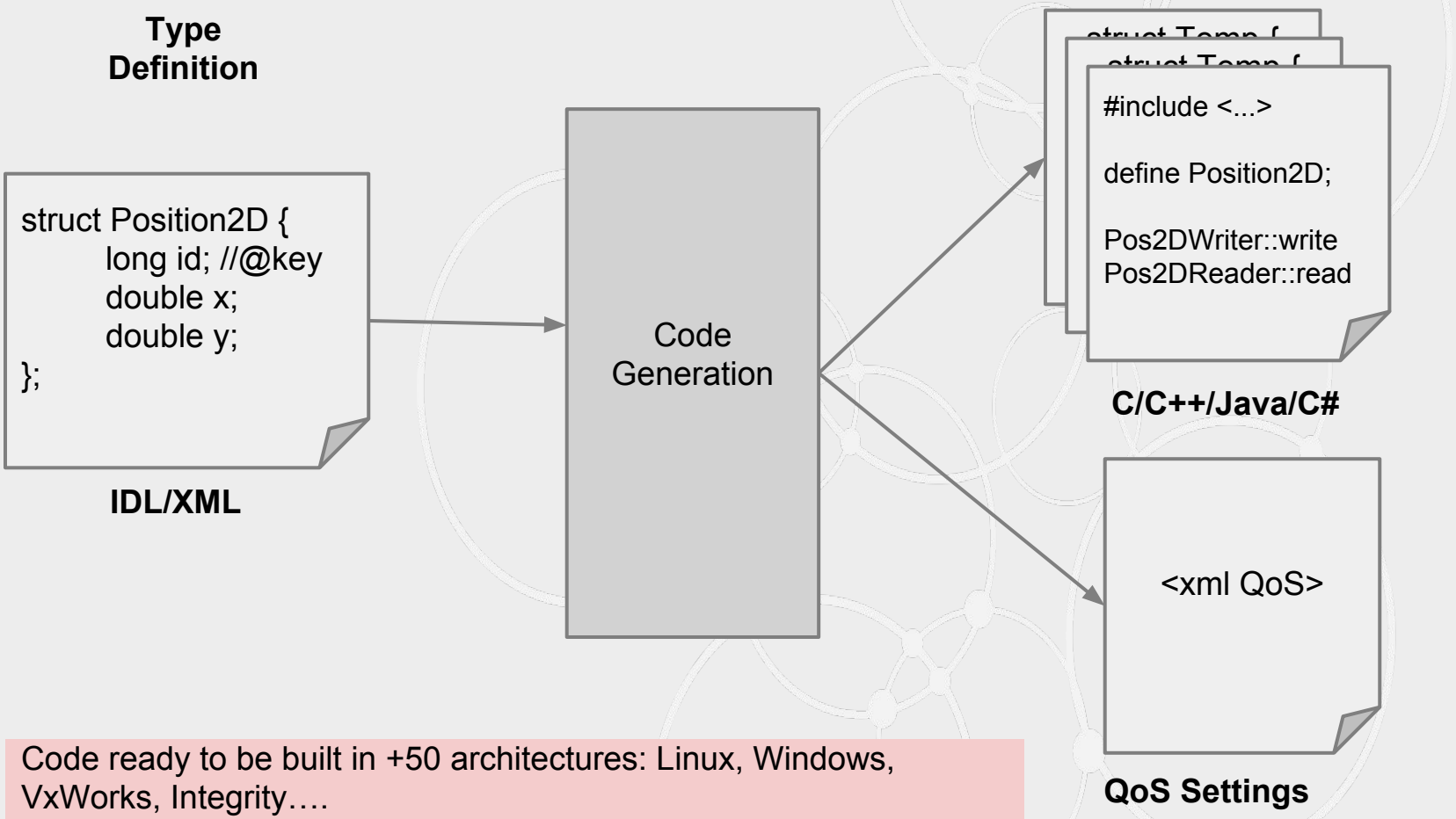


https://www.rti.com/downloads/shapes-demo

What is Connector?

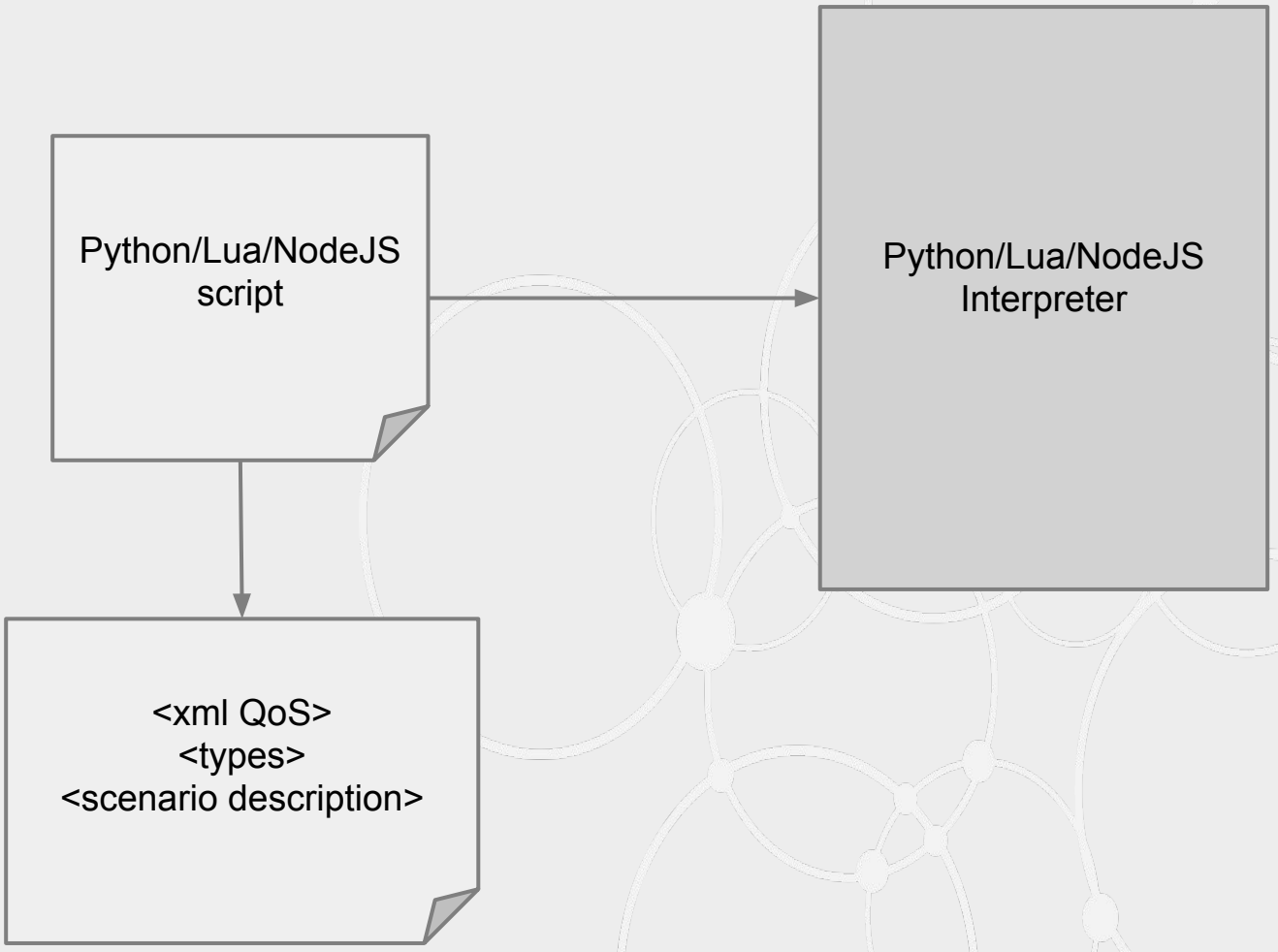
- Simple API for Data-Centric Publish/Subscribe
  - Built on top of DDS
  - Very few methods
  - Experimental
- Scripting language bindings
  - node.js, python, nodered
- Prototyping
  - Entities and QoS configured by XML

Classical DDS Workflow

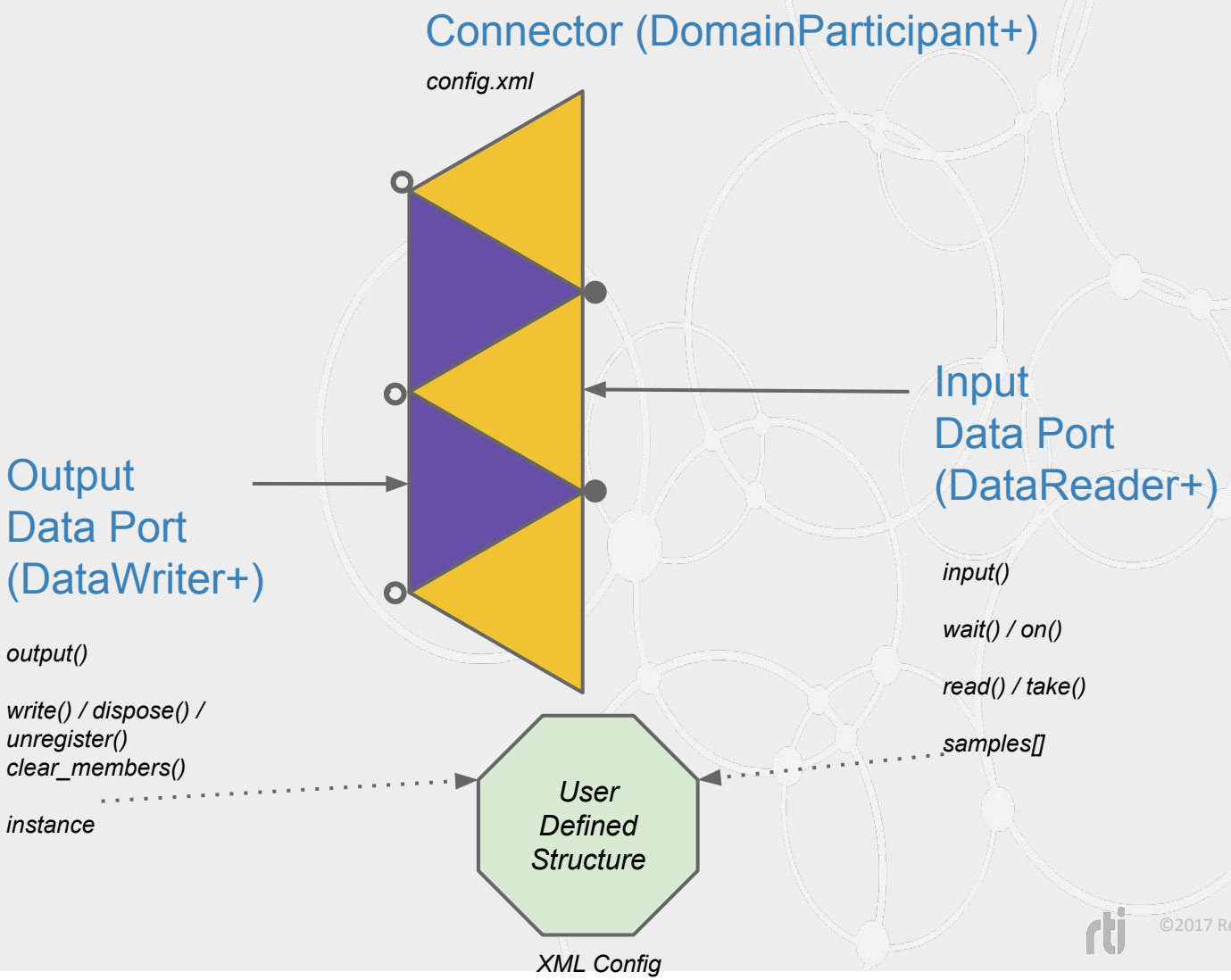


Code ready to be built in +50 architectures: Linux, Windows, VxWorks, Integrity….
This process can be simplified even more: **RTI Prototyper/Connector**

Connector Workflow



DDS Connector Input & Output ports



Anatomy of a Publisher in Connector

```
1.  connector =
    rti.Connector("MyParticipantLibrary::Sensor",'Tutorial.xml')

2.  writer = connector.getOutput("TempPublisher::TempWriter")

3.  writer.instance.setString('id', sensor.id)

4.  writer.write()
```

1. Create a connector
2. Get the DataWriter
3. Set the instance values
4. Write the value

Anatomy of a Subscriber in Connector

```
1.  connector = rti.Connector("MyParticipantLibrary::Sensor", 'Tutorial.xml')
2.  reader = connector.getInput("TempPublisher::TempWriter")
3.  reader.read()
4.  for i in nsamples:
        if reader.infos.isvalid(i)
            sample = reader.samples.getDictionary(i)
```

1. Create a connector
2. Get the datareader
3. Read/Take the value
4. Iterate for all the samples
   a. if is valid
      i. get the sample

Setup

Installing the tutorial

```
> git clone https://github.com/jpovedano/dds-firststeps
> ./setup
```

In each terminal:

```
> . rticonnectenv/bin/activate
```

Let's Rock!

Hands On Exercise

About the examples

- Temperature sensor scenario
  - Sensors (publishers)
    - Publish temperature data every second
  - Console (suscribers)
    - Shows a table with data being published
- Configured by XML
  - Tutorial.xml
    - Define the DDS entities and their QoS settings

```
struct Sensor {
    string id; //@key
    long value;
    long timestamp;
};
```

Anatomy of the XML

1. **QoS Library**
   a. **QoS Settings**
2. Types
   a. Type Definition
3. Domain Library
   a. Domains and Topics
4. Participant Library
   a. DDS Entities hierarchy

```
<!-- Qos Library -->
<qos_library name="QosLibrary">
  <qos_profile name="DefaultProfile" is_default_qos="true">
    <participant_qos>
      <transport_builtin>
        <!-- <mask>UDPV4 | SHMEM</mask>-->
        <mask> SHMEM</mask>
      </transport_builtin>
    </participant_qos>

    <datareader_qos>
   <!-- Modify reader values here -->
    </datareader_qos>

  </qos_profile>
</qos_library>
```

Anatomy of the XML

1. QoS Library
   a. QoS Settings
2. **Types**
   a. **Type Definition**
3. Domain Library
   a. Domains and Topics
4. Participant Library
   a. DDS Entities hierarchy

```
<types>
  <struct name="Sensor" extensibility="extensible">
    <member name="id" stringMaxLength="128" id="0" type="string"
key="true"/>
    <member name="value" id="1" type="long"/>
    <member name="timestamp" id="2" type="long"/>
  </struct>
</types>
```

Anatomy of the XML

1. QoS Library
   a. QoS Settings
2. Types
   a. Type Definition
3. **Domain Library**
   a. **Domains and Topics**
4. Participant Library
   a. DDS Entities hierarchy

```
<!-- Domain Library -->
<domain_library name="MyDomainLibrary">
  <domain name="MyDomain" domain_id="0">
   <register_type name="Sensor" kind="dynamicData"
type_ref="Sensor"/>
   <topic name="Temperature" register_type_ref="Sensor"/>
 </domain>
</domain_library>
```

Anatomy of the XML

1. QoS Library
   a. QoS Settings
2. Types
   a. Type Definition
3. Domain Library
   a. Domains and Topics
4. **Participant Library**
   a. **DDS Entities hierarchy**

```
<!-- Participant library -->
<participant_library name="MyParticipantLibrary">
 <domain_participant name="Console"
domain_ref="MyDomainLibrary::MyDomain">
   <subscriber name="TempSubscriber">
     <data_reader name="TempReader" topic_ref="Temperature"/>
   </subscriber>
  </domain_participant>
  <domain_participant name="Sensor"
domain_ref="MyDomainLibrary::MyDomain">
  <publisher name="TempPublisher">
    <data_writer name="TempWriter" topic_ref="Temperature"/>
    </publisher>
  </domain_participant>
</participant_library>
```

Example 1: Basic pub/sub

- Goal
  - In this example we show how to publish data
  - Learn differences between read/take
  - Learn how to change QoS settings in Connector
- Documentation:
  - QoS Policy Reference Guide (cheat-sheet)

Example 2: Filtering

- Goal
  - Learn how to filter data per subscriber
- Description
  - The console application will only receive the data matching a certain criteria
- Two types of Filtering:
  - Content Based
    - Use a filter with SQL Expression
  - Time Based
    - time_based_filter QoS

Example 3: High availability

- Goal
  - Learn how to add robustness using DDS QoS
- Description
  - Adding a backup sensor that substitute the primary sensor in case of failure
- QoS:
  - Liveliness
  - Ownership
  - Ownership strength

Example 4: Durability

- Goal
  - Learn how to provide recent history to late joiners
- Description
  - A console application will receive the recent history published before it was started
- QoS:
  - Durability QoS

Example 5: Data Isolation/Partition

- Goal
  - Learn how to isolate publishers and subscribers within the same domain
- Description
  - A console application will receive sensor updates only for subscribers in certain zones
- QoS:
  - Partition QoS

Example 6

- Goal
  - create your own distributed application and share it with the community
- Description
  - Let your imagination fly!
- References
  - XML Application creation

Thanks for your attention!