TIMICO

Data Distribution Service for Real-Time Systems

Christian Fischer Pedersen
Assistant Professor
cfp@eng.au.dk

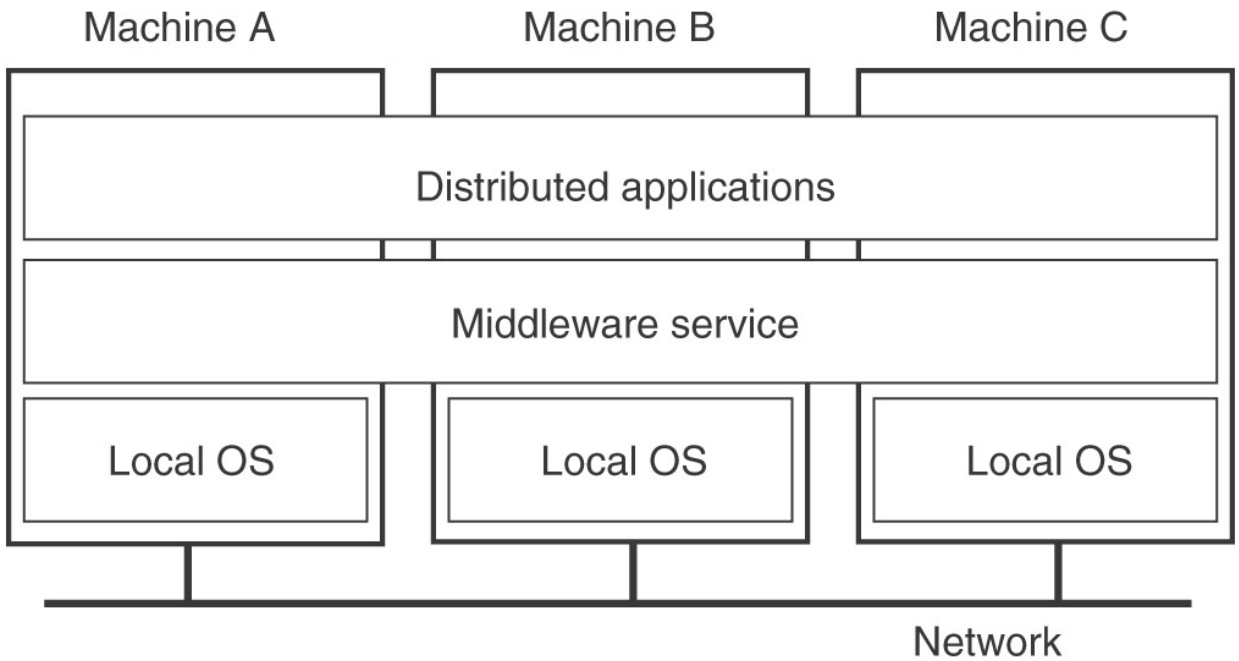
Section of Electrical and Computer Engineering
Department of Engineering
Aarhus University

November 12, 2014

Outline

Outline

Middleware: A graphical perspective



(Tanenbaum, 1995 [4])

Middleware: A loose definition

Definition (Coulouris et al., 2001 [1])

A layer of software residing on every machine, sitting between the underlying operating system and the distributed applications, whose purpose is to mask the heterogeneity of the cooperating platforms and provide a simple, consistent and integrated distributed programming environment.

Middleware homogenizes by abstraction

- Network technologies
- Hardware architectures
- Operating systems
- Programming languages
- Geographical location, concurrency, failure, ...

Middleware is suited for

- Applications in **networked** and **heterogeneous** environments
- Middleware informally called “plumbing”
- Connects parts of a distributed application with “data pipes” and passes data between them, e.g. link DBs and legacy systems at multiple locations

Software development with middleware

- Do (usually) not have to learn a new programming language
- Use a familiar language, e.g. C++, Java, C#
- Middleware systems often provide function libraries
- Some languages comprise middleware components, e.g. Java
- Interface Definition Language (IDL) that "maps" to the language and generates a local proxy

More types of middleware often needed

No middleware tech. supports **all** requirements for **all** systems

- Complex embedded designs often require more than one middleware component to meet requirements

Middleware is simply software like any other

- Introduces, e.g. CPU, RAM, battery, overhead but
- Makes development and integration easier, more efficient and less error-prone
- Do a cost / benefit analysis

Middleware standards or the lack thereof

- Currently, there is **not one single** middleware standards organization that defines and manages standards for embedded systems
- Recommended that **you** keep up to date with standards
- Example: Object Management Group (OMG) standardizes
  - DDS (Data distribution service for real-time systems)
  - CORBA (Common Object Request Broker Architecture)
  - UML (Unified Modeling Language)

Middleware makes decoupling possible

Middleware **may decouple** data producers and consumers in

- Space, time and flow

Space decoupling

- Producer(s) and consumer(s) do not know each other

Time decoupling

- Interaction is asynchronous

Flow decoupling

- Data production and consumption not in main control flow of producer or consumer, i.e. no blocking of flow

Decoupling removes explicit producer-consumer dependencies

- Coordination and synchronization is reduced

Some middleware communication paradigms

- Remote procedure call and derivatives
- Object request brokers
- SQL orientation
- Shared space
- Message passing
- Message queuing
- Publish / Subscribe

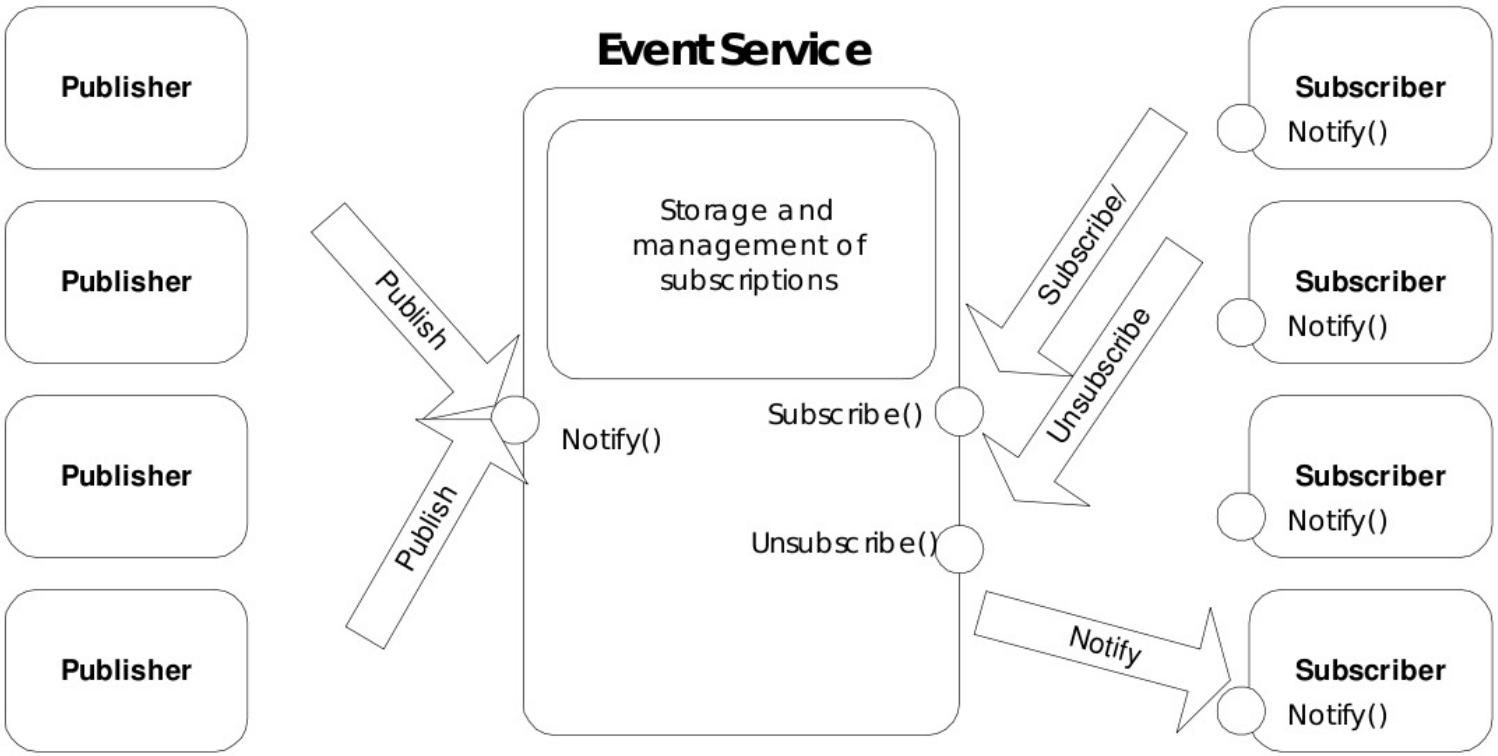
Decoupling interaction paradigms

Abstraction	Space decoupling	Time decoupling	Flow decoupling
Message Passing	No	No	Producer-side
RPC/RMI	No	No	Producer-side
Asynchronous RPC/RMI	No	No	Yes
Future RPC/RMI	No	No	Yes
Notifications (Observer D. Pattern)	No	No	Yes
Tuple Spaces	Yes	Yes	Producer-side
Message Queuing (Pull)	Yes	Yes	Producer-side
Publish/Subscribe	Yes	Yes	Yes

(Eugster et al., 2003 [2])

Outline

Conceptual Publish/Subscribe overview

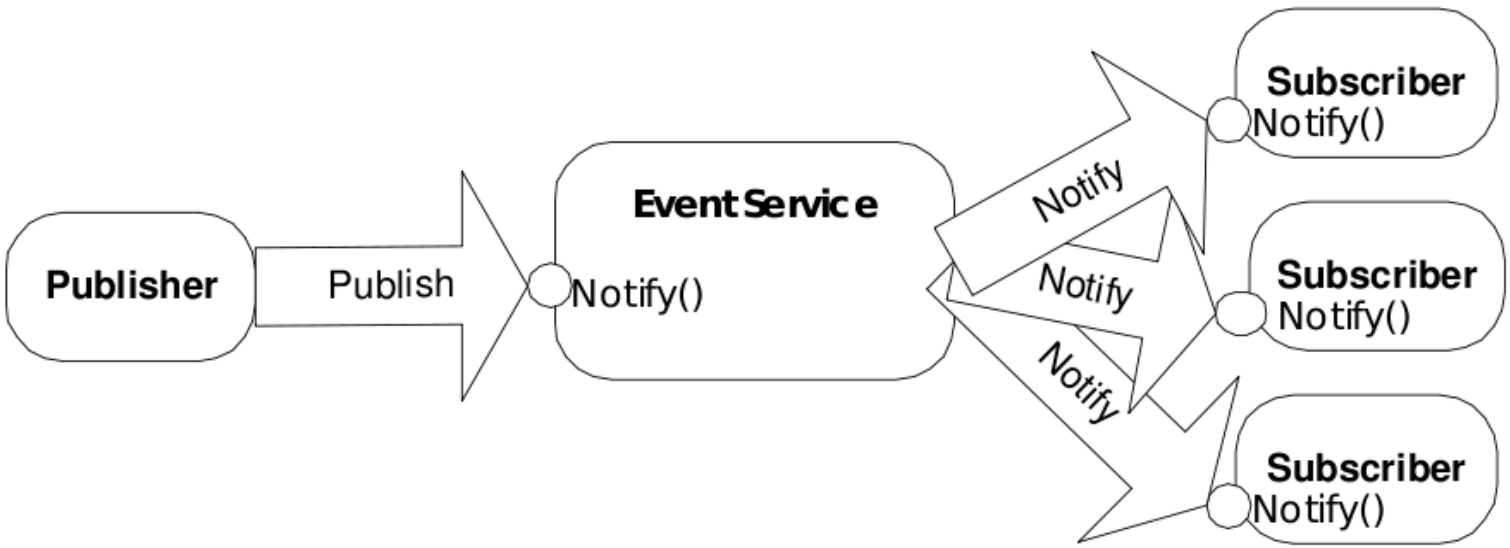


(Eugster et al., 2003 [2])

Decoupling by Publish/Subscribe

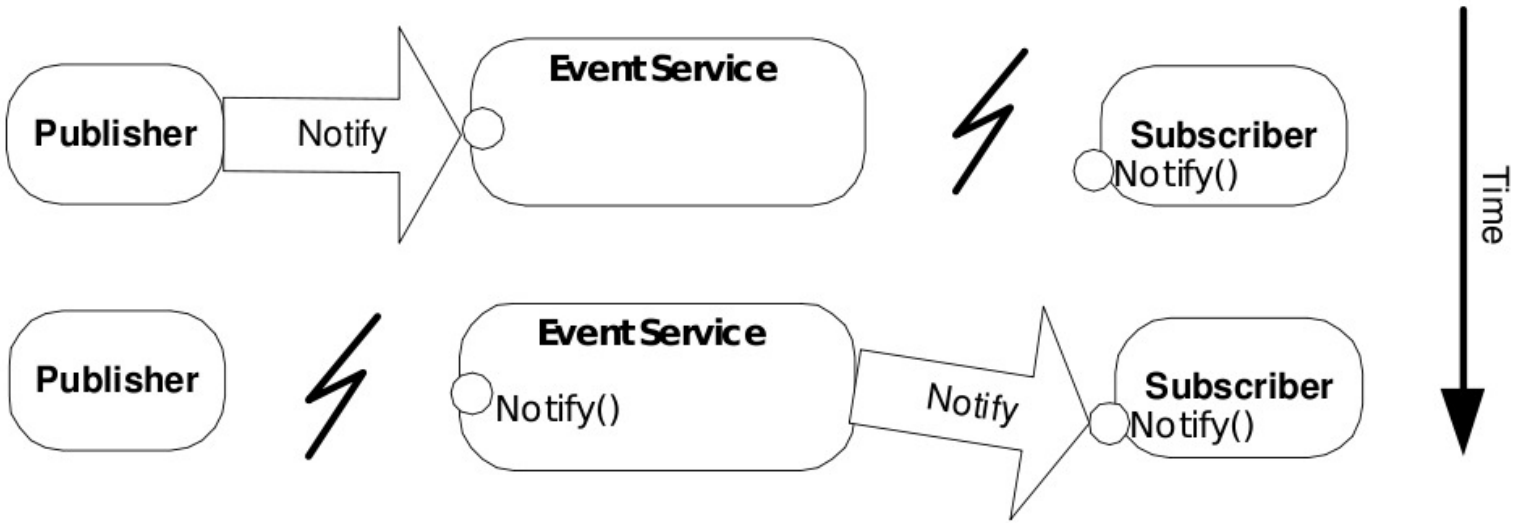
- The event-service provides decoupling
- Space decoupling
  - Publisher(s) and subscriber(s) do not know each other
- Time decoupling
  - Interaction is asynchronous
- Flow decoupling
  - Message production/consumption not in main control flow of Publisher/Subscriber, i.e. no flow blocking occur
- Decoupling removes explicit dependencies between Publisher and Subscriber, i.e. reduces coordination and synchronization

Space decoupling



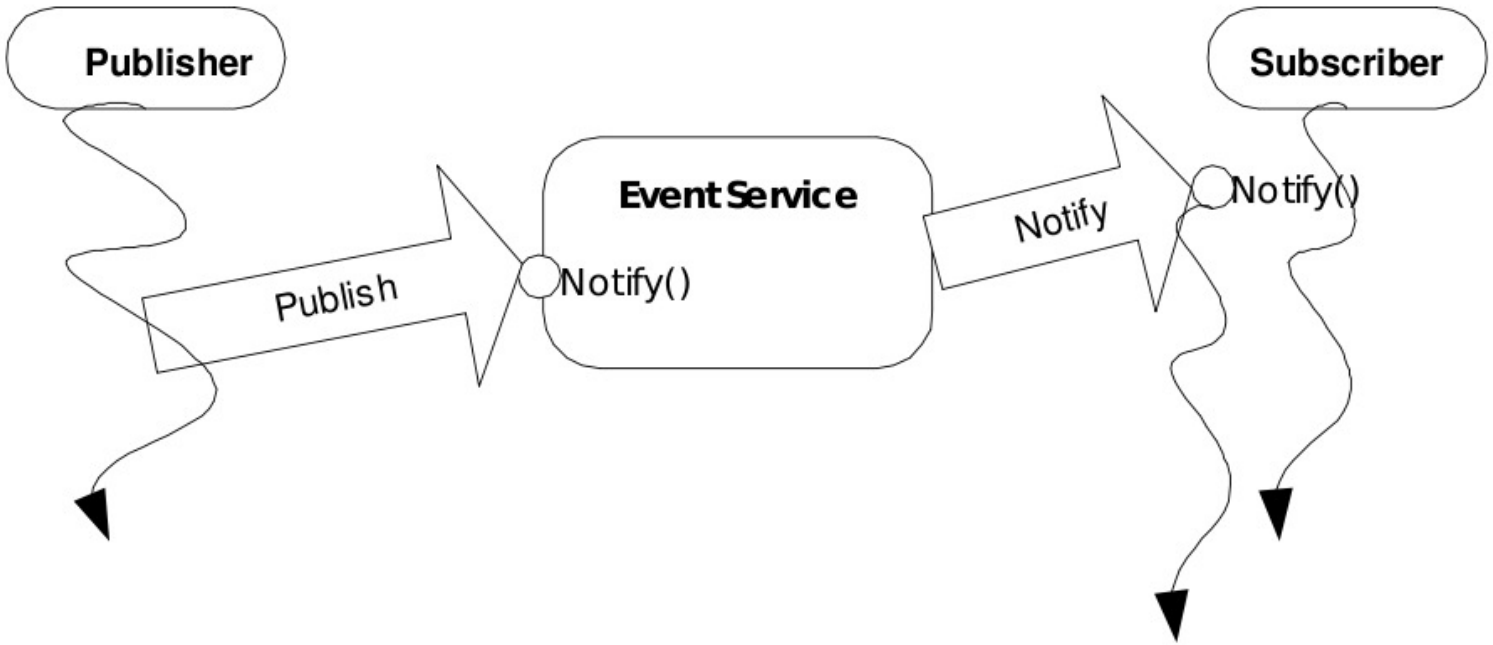
(Eugster et al., 2003 [2])

Time decoupling



(Eugster et al., 2003 [2])

Flow decoupling



(Eugster et al., 2003 [2])

Topic based Publish/Subscribe

- Messages are published to topics (keywords/tags)
- Subscribers subscribe to topics and receive all messages published to subscribed topics
- All subscribers to a topic will receive the same messages
- **Publisher is responsible** for defining the topics to which subscribers can subscribe
- Topics usually organized in hierarchies, e.g. subscribe to a topic and all sub-topics

Content/Property based Publish/Subscribe

- Message only delivered to subscriber if attributes or content of message match constraints/filter defined by subscriber
- **Subscriber** categorizes via subscription filter/pattern
- String: Most frequent way to express subscription pattern
  - Syntax, e.g. OMG *Default Filter Constraint Language*
- Template object: Subscribe to messages that match the attributes of subscribers template object. Attributes may be given a *null* wild-card, i.e. match not necessary

Type based Publish/Subscribe

- A scheme that filters events according to their type
- Enables a closer integration of the programming language and the middleware
- Type safety – with compile type checking

Topic/content hybrid

- Publishers post messages to a topic while subscribers register content-based subscriptions to one or more topics

Outline

Data Distribution Service for Real-Time Systems

Data Distribution Service for Real-Time Systems (DDS)

- A **standard** managed by Object Management Group (OMG)
- Specifies a data centric **publish/subscribe** middleware for distributed real-time systems

DDS supports the subscription models

- Topic based
- Content based
- Type based

See the subscription models in the paper

- "The many faces of Publish/Subscribe" (Eugster et al., 2003 [2])

Background of DDS

Two major proprietary DDS implementations have existed for years

- Network Data Delivery Service (NDDS)
  by Real-Time Innovations (RTI)
- Subscription Paradigm for the Logical Interconnection of Concurrent Engines (SPLICE)
  by Thales
  Now OpenSplice DDS by
  Prismtech

Teamed together in 2004 to create the DDS **standard**

- Approved and now maintained by OMG
- OMG also maintains, e.g. UML and CORBA

Various implementations of the DDS standard

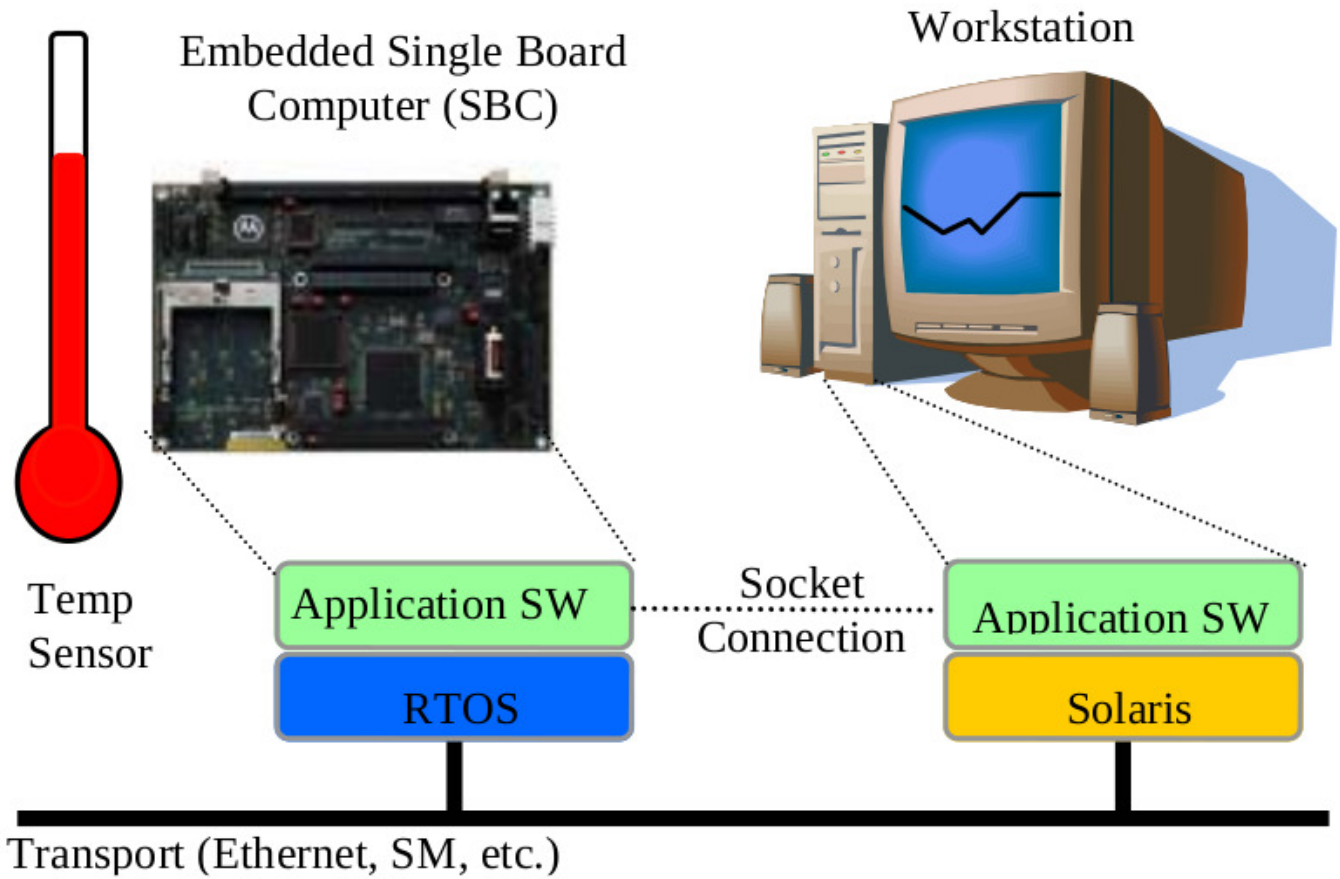
Commercial

- Real-time innovations DDS
- PrismTech OpenSplice DDS (commercial)
- Twin Oaks Computing CoreDX DDS
- MilSOFT DDS
- Gallium InterCOM DDS
- Sparx DDS

Open source

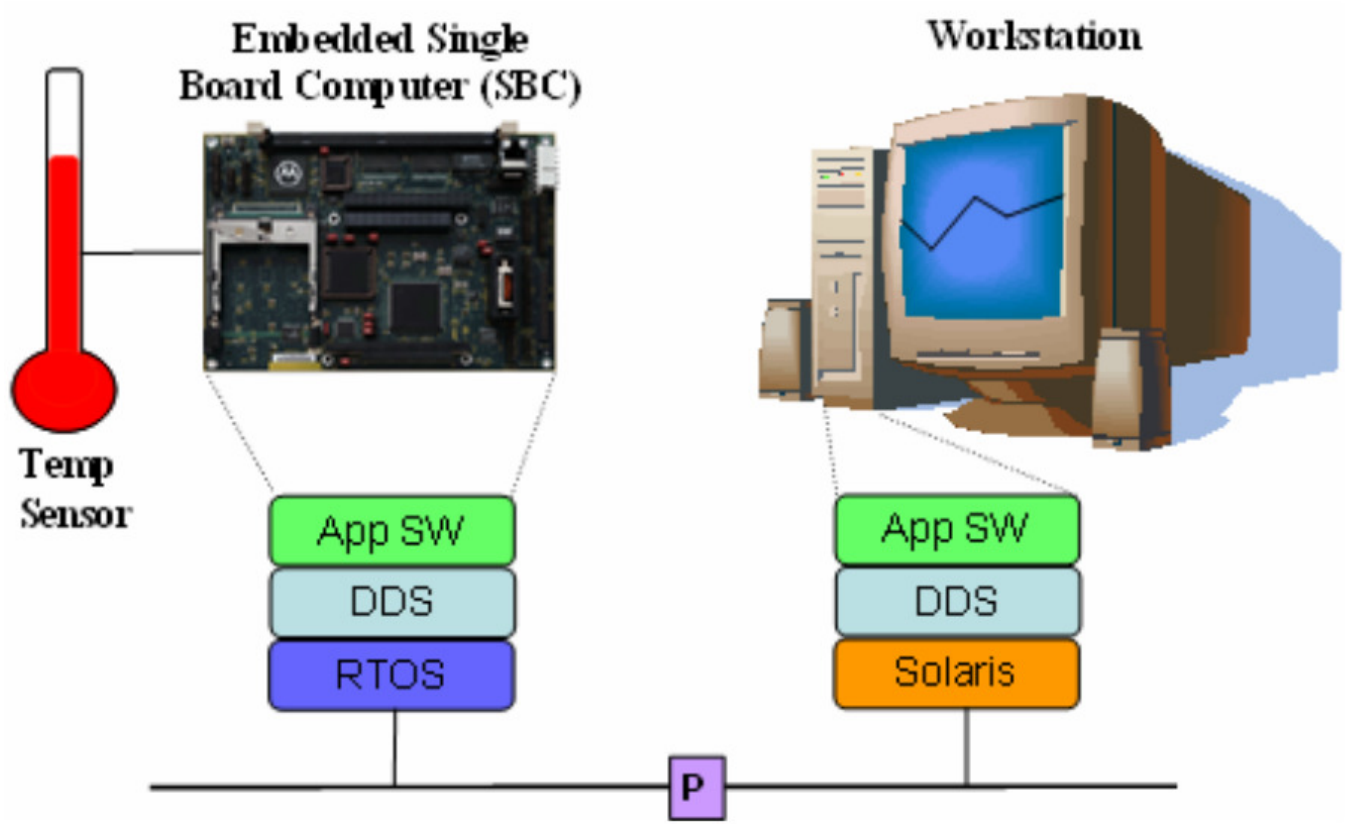
- PrismTech OpenSplice DDS (open version)
- Object Computing Inc. OpenDDS

Simple distributed application without DDS



(Pardo-Castellote et al., 2005 [3])

Simple distributed application with DDS



(Pardo-Castellote et al., 2005 [3])

Data-centricity via data oriented QoS parameters

Provides the ability to specify various **QoS** parameters

- Rate of publications
- Rate of subscriptions
- Persistence, i.e. how long the data is valid
- Memory utilization
- Etc...

QoS enables

- Tailoring the communication mechanism to the application

Scalable via discovery protocol

- To hundreds or thousands of publishers and subscribers

DDS application domains

DDS may be applied in a broad range of applications

- Industrial automation
- Distributed control and simulation
- Telecom equipment control
- Sensor networks
- Network management systems
- Etc...

A DDS application: Bluefin Robotic's UUV

Sensors and actuators

- Radar
- Sonar
- Hydrophone arrays
- Acoustic doppler velocity
- Acoustic doppler current
- Thermometer
- GPS

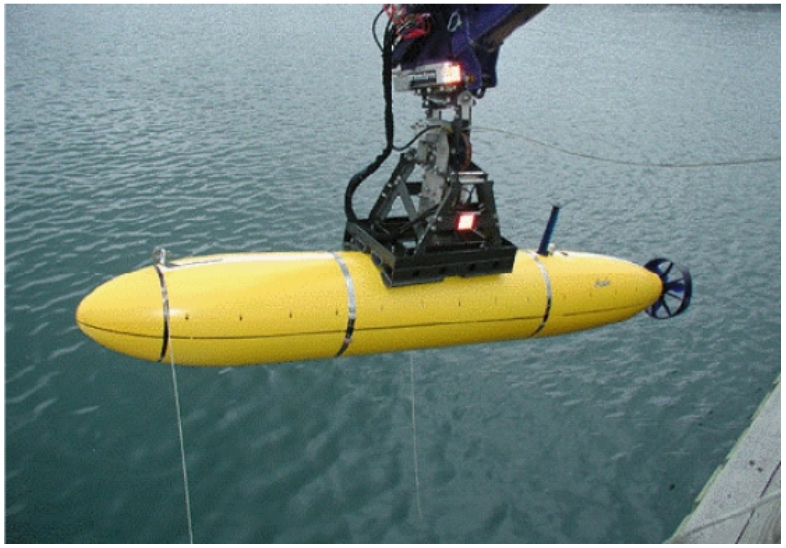
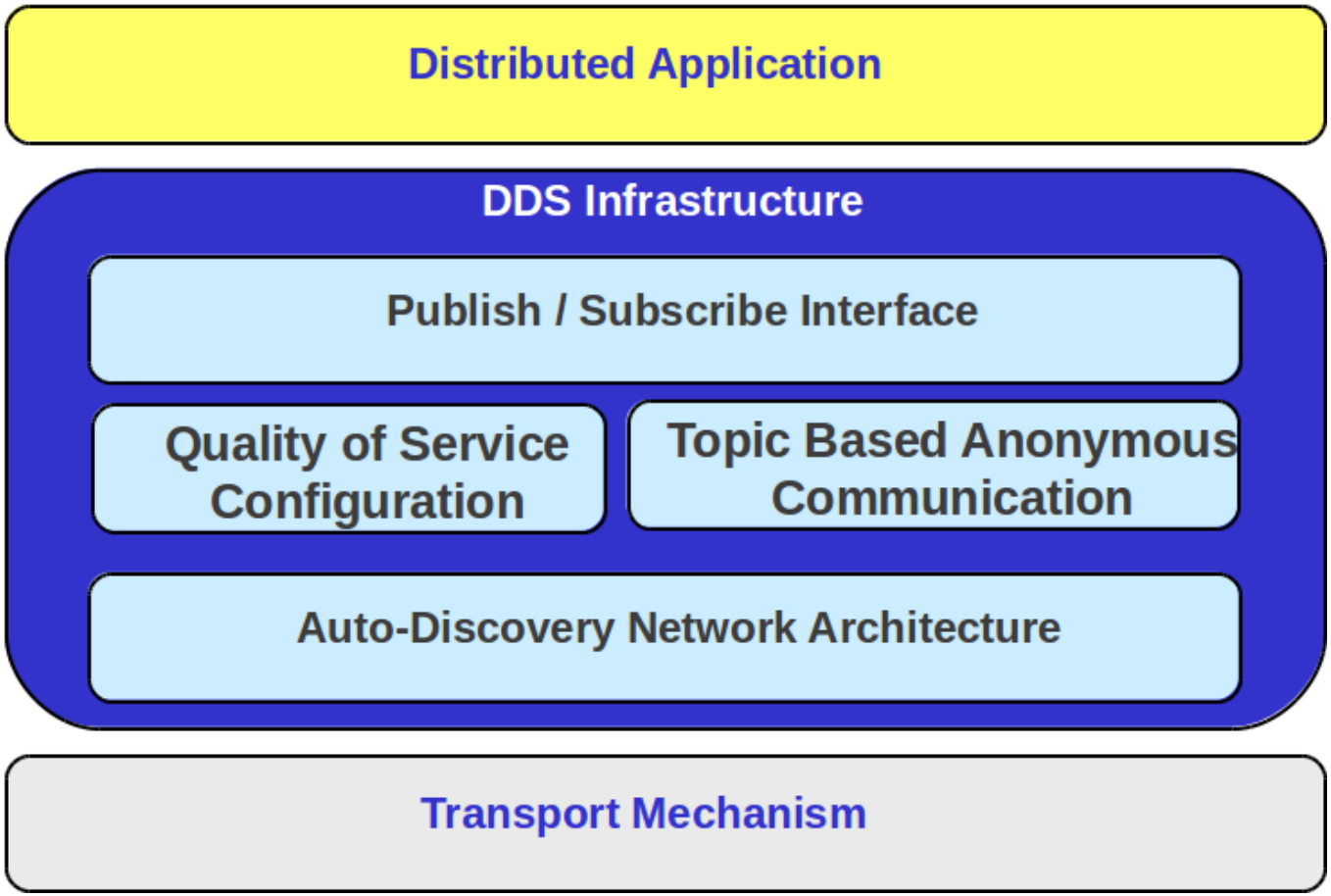


Figure : Unmanned underwater vehicle (UUV)

(Real-Time Innovations)

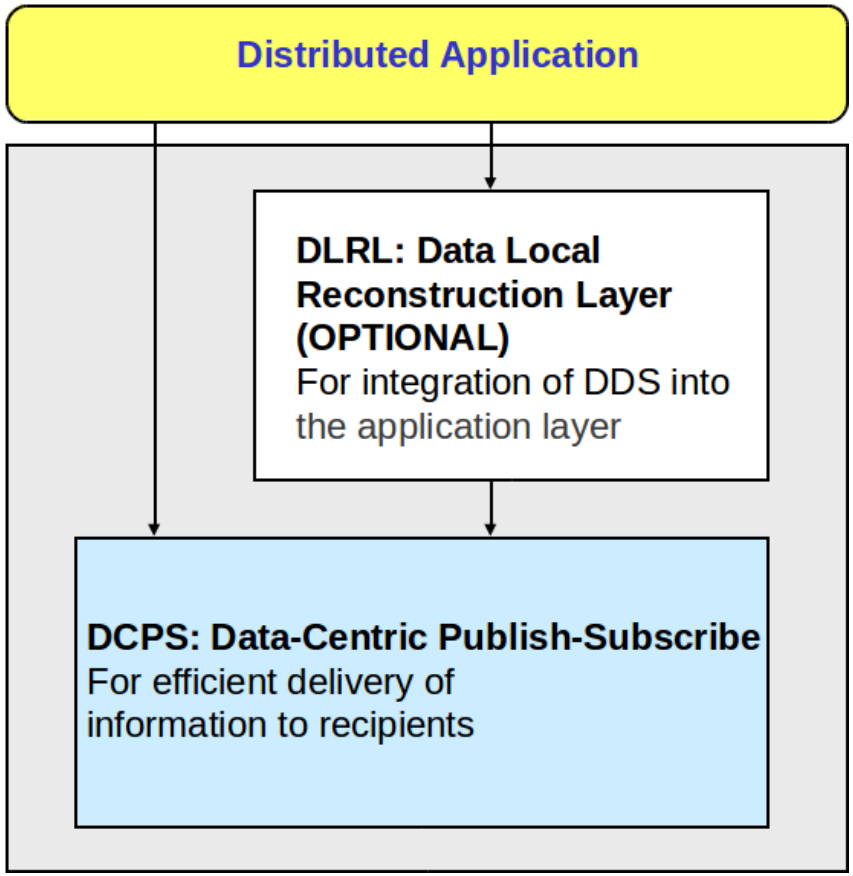
Outline

The DDS infrastructure: Overview



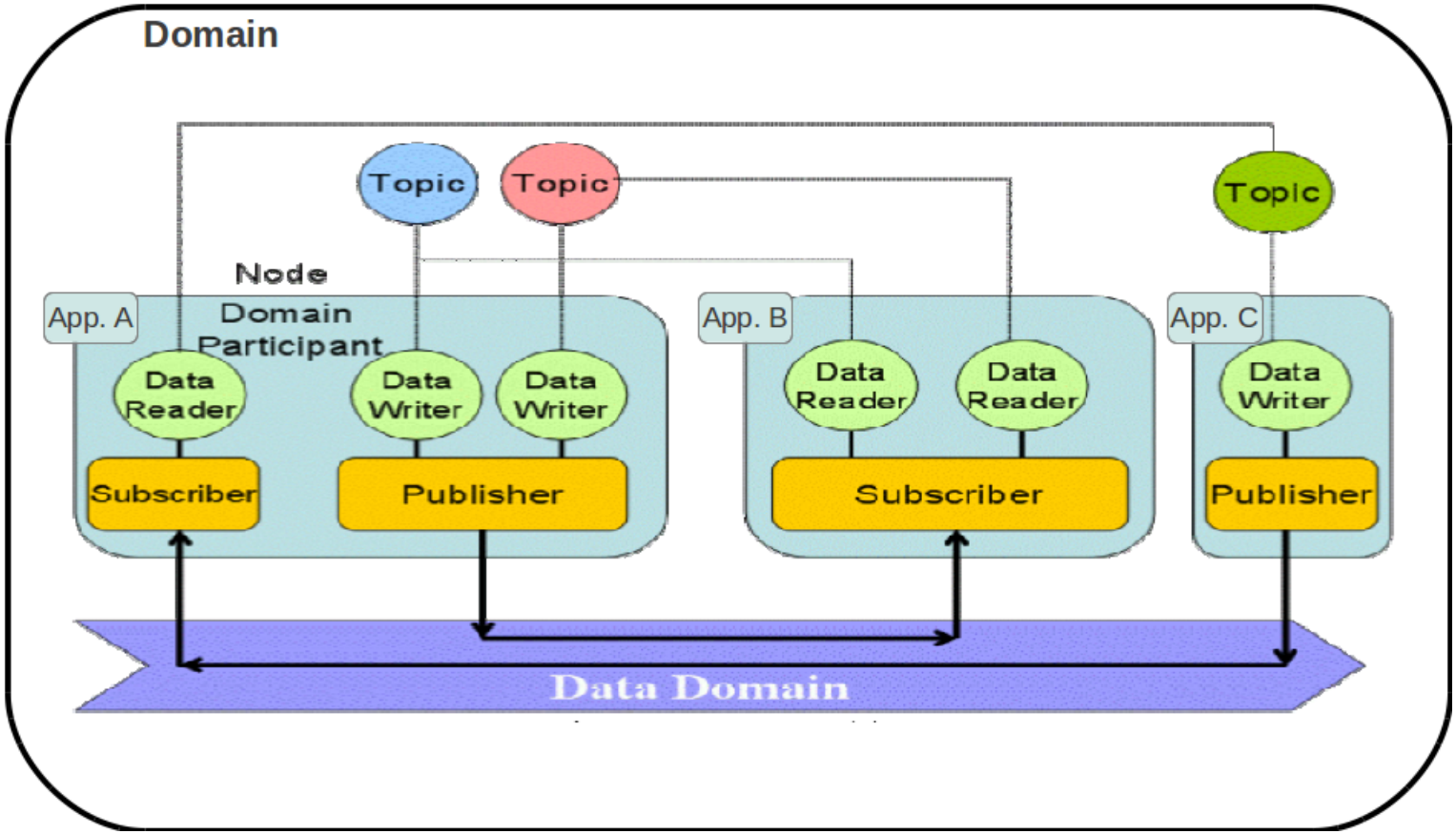
(Pardo-Castellote et al., 2005 [3])

The DDS infrastructure: DLRL and DCPS



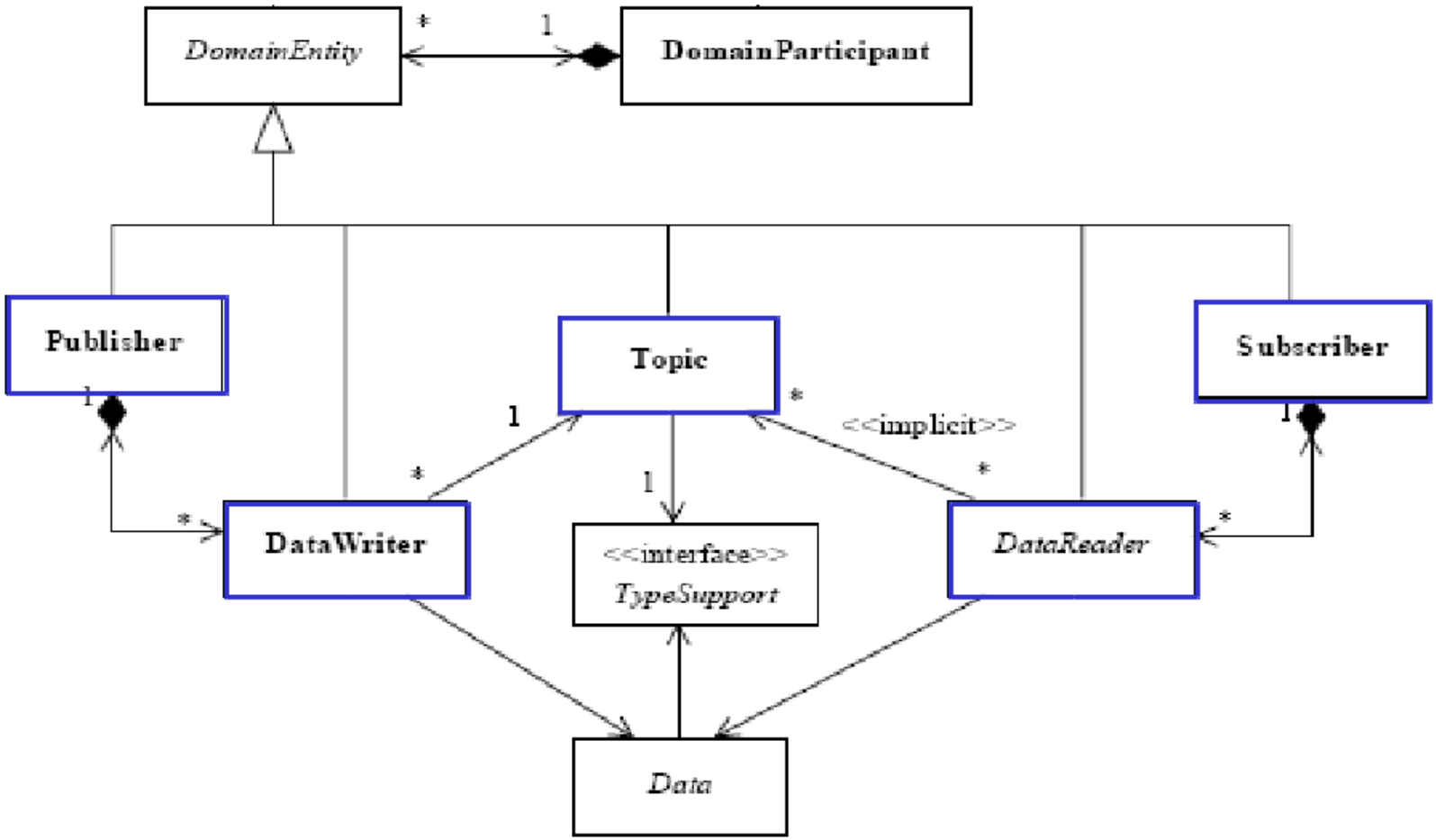
(Pardo-Castellote et al., 2005 [3])

DCPS entities: Graphical layout



(Pardo-Castellote et al., 2005 [3])

DCPS entities: UML layout

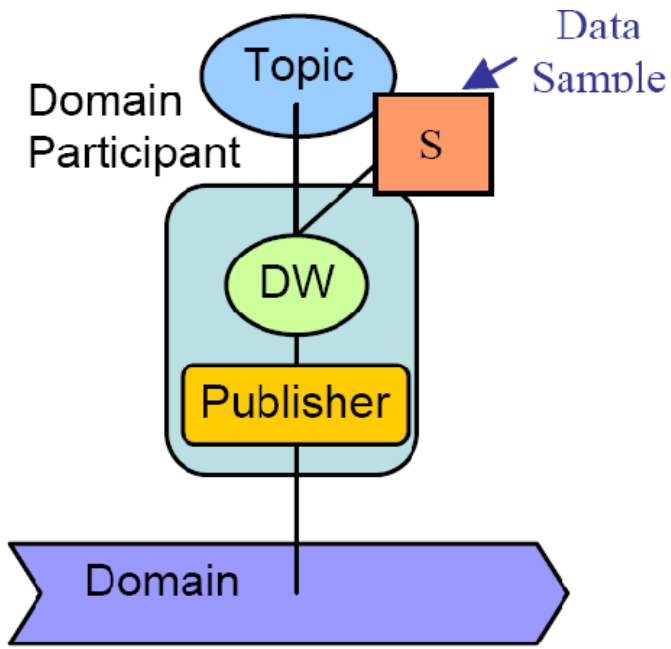


Publication model

A **Publisher** is an object responsible for data distribution

- DataWriter object is a typed accessor to the publisher
- DataWriters send data-objects of a given type

Publication model: Graphical layout



(Pardo-Castellote et al., 2005 [3])

Subscription model

A **Subscriber** is an object responsible for receiving published data

- DataReader object is a typed accessor to the subscriber
- DataReaders receive data-objects of a given type

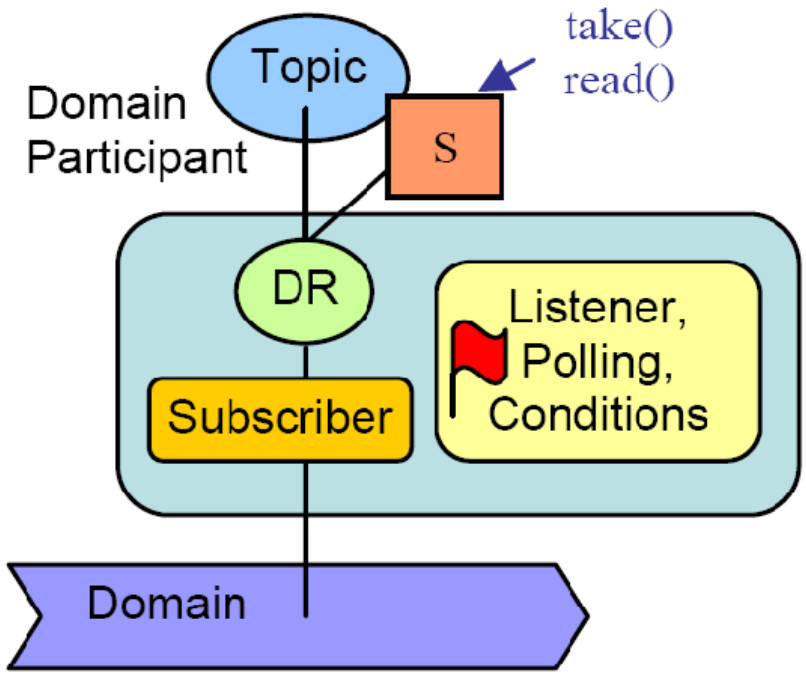
Three methods for receiving data

- **Listener** callback: Called by DDS when data is received
- **Polling**: Application polls DataReader for available data
- **Conditions** and WaitSets
  - The application waits until a specified condition is met
  - When met: Access the data from the DataReader

Accessing data by calling `take()` or `read()`

- `take()`: removes the data
- `read()`: reads but does not remove data

Subscription model: Graphical layout



(Pardo-Castellote et al., 2005 [3])

Topics: In general

- Logical connection point between publishers and subscribers
- Publisher/subscriber topics much match for communication
- A Topic consists of a **name** and a **type**
- **Name** is a string **uniquely** identifying the topic in a **domain**
- **Type** is the definition of the data contained in the Topic
- Types are defined with Interface Definition Language (IDL)
- IDL is an OMG standard for defining object/data interfaces

Topics: An example

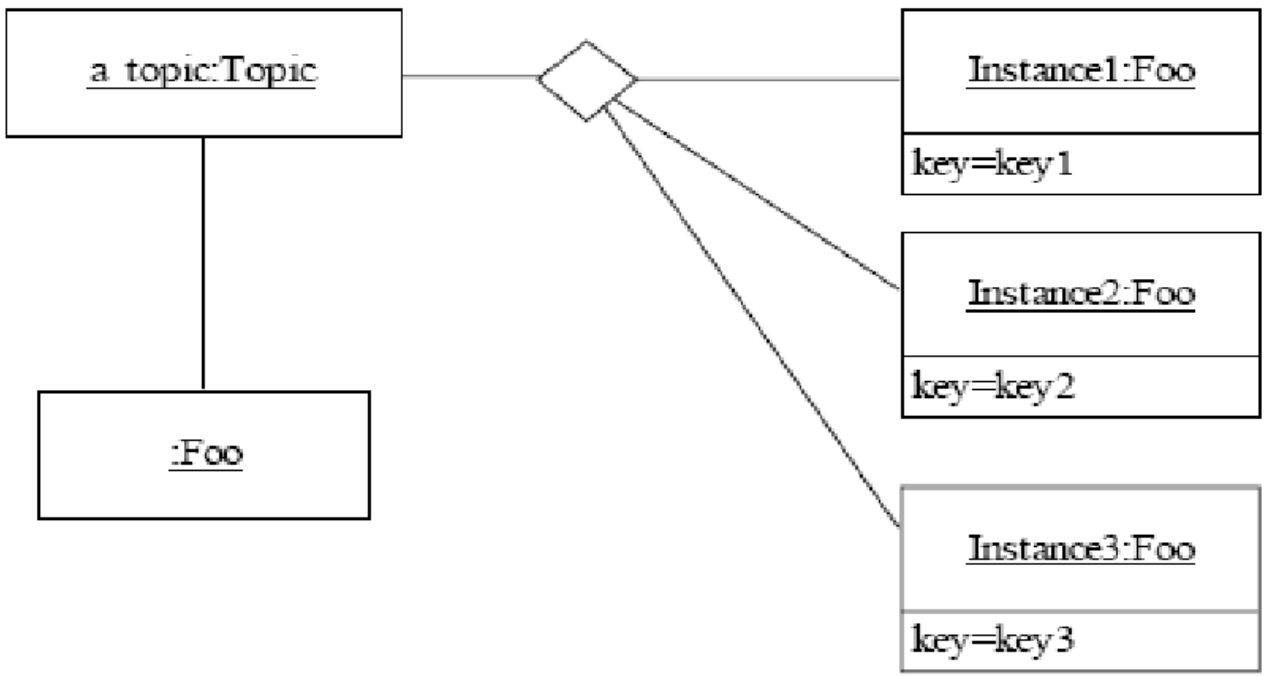
- In a Topic type definition data-elements can be set to **key**
- The **key** can be used for filtering incoming data
- Keys support scalability, e.g. multiple temp. sensor nodes
- No need for individual Topics with different names for each temp. sensor node when the type is the same
- With keys: Only one Topic needed

Example (Topic Type)

```
struct Temperature{
  float data;
  unsigned long sensorId; // key
};
```

Topics: Instances and keys

- A **Topic** corresponds to a single data type
- **Keys** are for filtering incoming data
- Without keys:
  Create individual **Topics** with same **Types** for **each** publisher



Topics: Content filtering

A **ContentFilteredTopic** allows to declare a filter expression

- Data samples of specified Topic will be filtered

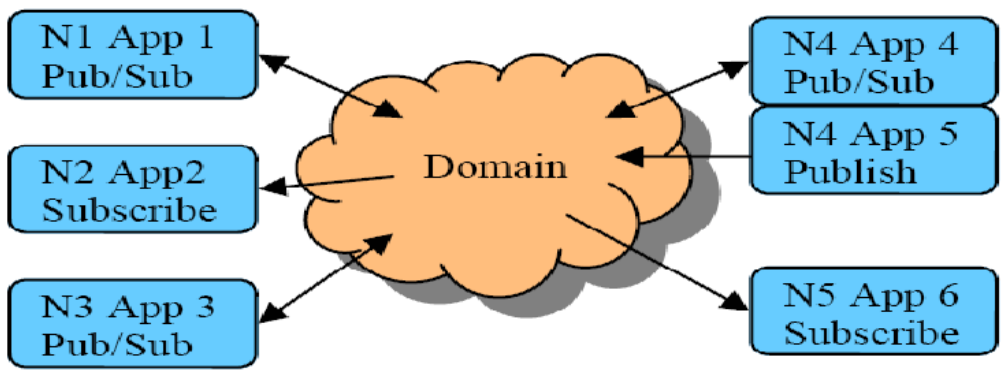
Temperature sensor example

- Filters on temperature value
- E.g. subscribers only receive and process data when a temperature exceeds a specific limit
- Hence, reduce information overload for subscribers

Single domain system

Domain: Binds individual applications together for communication

- DataWriters and DataReaders with same data types will communicate within domain
- In this domain: Six applications on five nodes

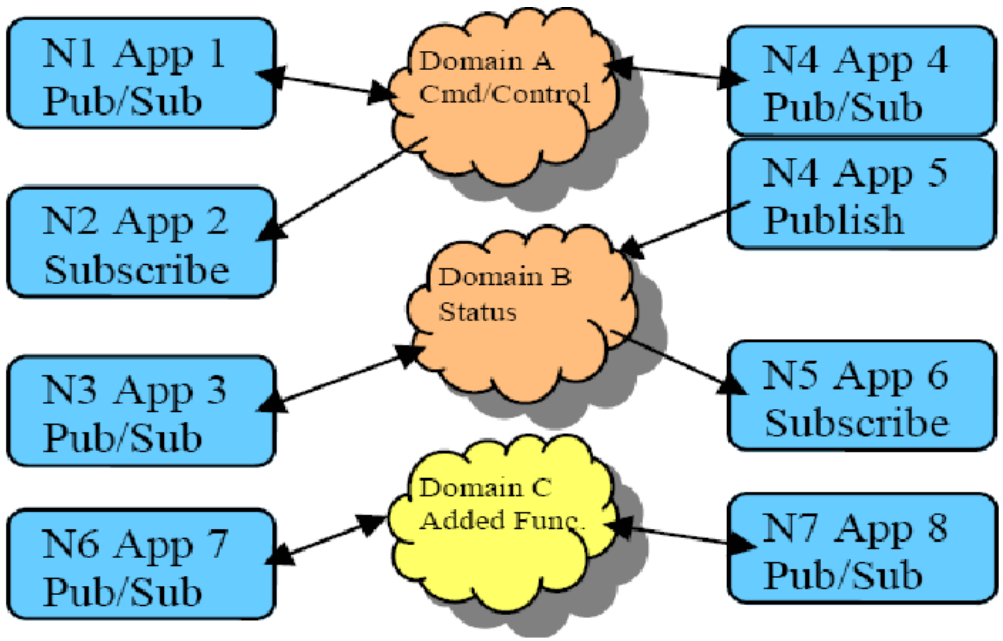


(Pardo-Castellote et al., 2005 [3])

Multiple domains system

Domain: Binds individual applications together for communication

- DataWriters and DataReaders with same data types will communicate within domain – **not across** domains
- For data isolation, e.g. one domain per functional area



(Pardo-Castellote et al., 2005 [3])

Outline

Simple Discovery Protocol: The challenge

The challenge

- DDS tracks presence of all **participants** and **endpoints**
- For **DDS** and **applications** to be able to react on discovery

Simple Discovery Protocol (SDP)

1. Simple **Participant** Discovery Protocol (SPDP)
   To discover new Participants in the same Domain
2. **Endpoint** Discovery Protocol (SEDP)
   To exchange Endpoint information between Participants

Simple Discovery Protocol: Overall mechanics

SDP uses DDS pub/sub for discovery purpose

- For DDS itself and the application to get presence information
- Built-in Topics with predefined name and data type
- Built-in DataReader/Writer objects associated with the Topics
- Predefined - but tweakable - QoS policies for SDP entities

Simple Discovery Protocol: Auto-Magic

For **each** DomainParticipant

- **6 objects** automatically created for discovery purposes

First 2 objects: Simple Participant Discovery Protocol

- To send/receive participant DATA messages to find remote DomainParticipants
- This phase uses best-effort communications.

Last 4 objects: Endpoint Discovery Protocol

- To learn about each other's DataWriters and DataReaders
- This phase uses reliable communications

Simple Discovery Protocol: Graphical layout

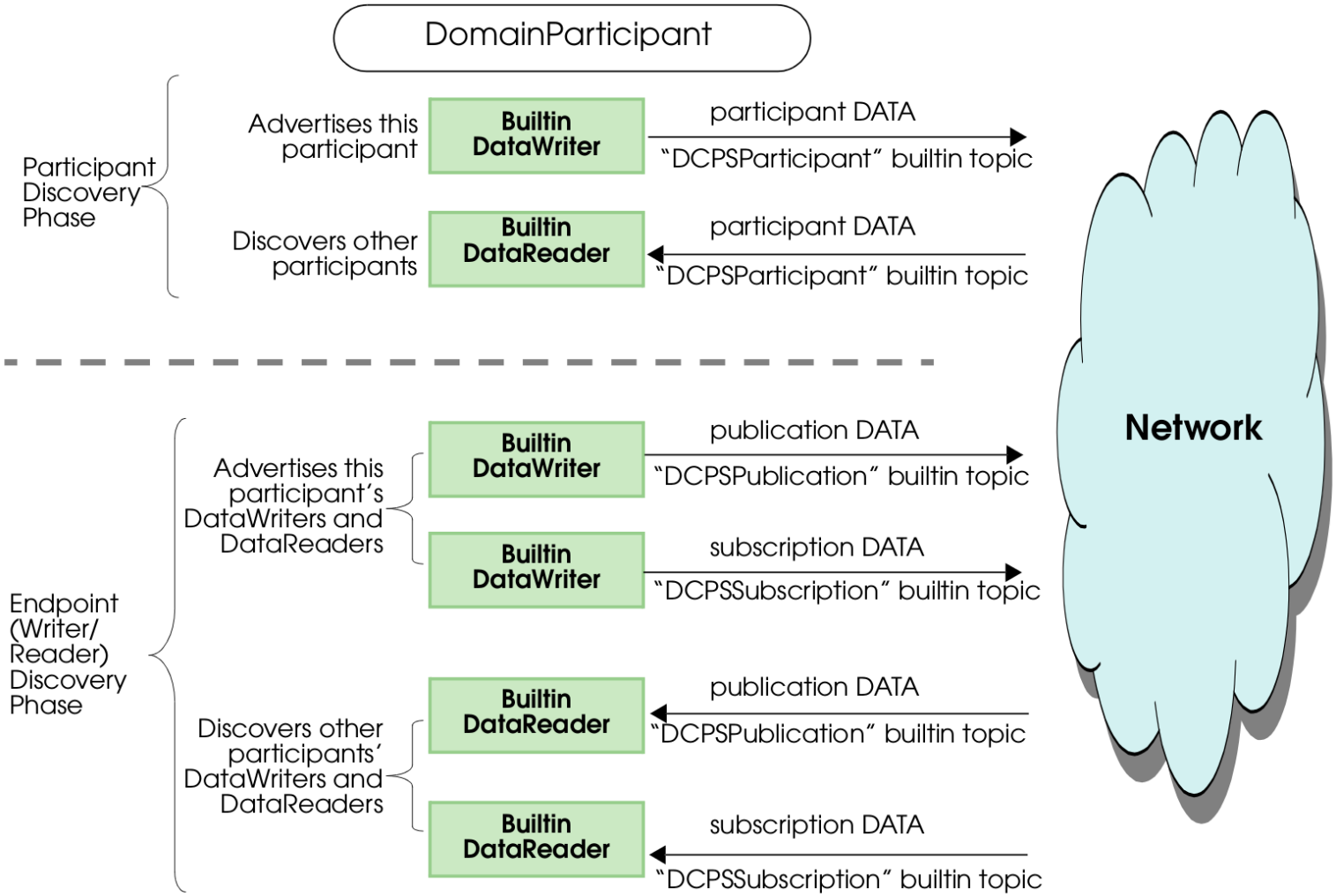


Figure : Built-in Writers and Readers for the discovery protocol SDP

(RTI DDS Users' Manual v. 4.5)

DDS QoS parameters

1. Deadline
2. Destination order
3. Durability
4. Entity factory
5. Group data
6. History
7. Latency budget
8. Lifespan
9. Liveliness
10. Ownership
11. Ownership strength
12. Partition
13. Presentation
14. Reader data lifecycle
15. Reliability
16. Resource limits
17. Time-based filter
18. Topic data
19. Transport priority
20. User data
21. Writer data lifecycle

QoS parameter example

Durability QoS

- Whether DDS will make past data samples available for newly joining DataReaders
- Volatile: The system does not keep any past data samples
- Transient: The system keeps a certain no. of samples
- Persistent: The system will keep all past samples, e.g. on disk

Persistence Service in RTI's Connext to persist in files and DBs

Outline

RTI Connext

- RTI (Real-Time Innovations): US based company
- RTI Connext: **Closed** source impl. of OMG's **open** DDS std.
- OMG: Object Management Group
- Connext covers heterogeneity
  - C, C++, .Net, Java, Ada
  - Windows, Linux, Solaris, Unix, IBM AIX, ...
  - VxWorks, Integrity, LynxOS, and other RTOSs
- These slides look at **Connext** from a **Ubuntu** perspective

RTI Connext: Download and install

- Download a version of Connext Prof. from RTI's website
- Get license file, `rti_license.dat`, from me or trial from web
- `sudo [sh] ./RTI_Connext_Professional_Edition-*.sh`
- Install in, e.g. `/opt/RTI`
- Follow the on-screen installation instructions
- Move license file to installation root, e.g.`/opt/RTI/`
- Copy license file to `/opt/RTI/ndds.5.1.0/` (check ver. no.)
- I need the license file in both places sometimes... hmm...

Do not distribute AU's licence file!

RTI Connext: Setup environment

Add to /etc/environment

```
NDDSHOME="/opt/RTI/ndds.5.1.0"
PATH="[current paths]:$NDDSHOME/scripts"
RTI_LICENSE_FILE="$NDDSHOME/rti_license.dat"
```

Refresh environment variables

- Logout and login

RTI Connext: Check installation

Run the `rtilauncher` in `/opt/RTI/RTI_Launcher*/[sub-dirs]/`

- Is the license file accepted?
- Does the RTI Launcher start up?
- Are all menu items colored or grayed out?

RTI Connext: Common problems

Q1 Error while loading shared libraries `libtiff.so.3`

A1 Given that your latest libtiff library is `libtiff5`:

```
sudo ln -s
/usr/lib/x86_64-linux-gnu/libtiff.so.5
/usr/lib/x86_64-linux-gnu/libtiff.so.3
```

Q2 Error while loading shared libraries `libjpeg.so.62`

A2 Given that your latest libjpeg library is `libjpeg8`:

```
sudo ln -s
/usr/lib/x86_64-linux-gnu/libjpeg.so.8
/usr/lib/x86_64-linux-gnu/libjpeg.so.62
```

Learning by the Shapes Demo

- Run `rtilauncher` in `/opt/RTI/RTI_Launcher*/scripts/`
- Goto the `Utilities` tab and choose `Shapes Demo`

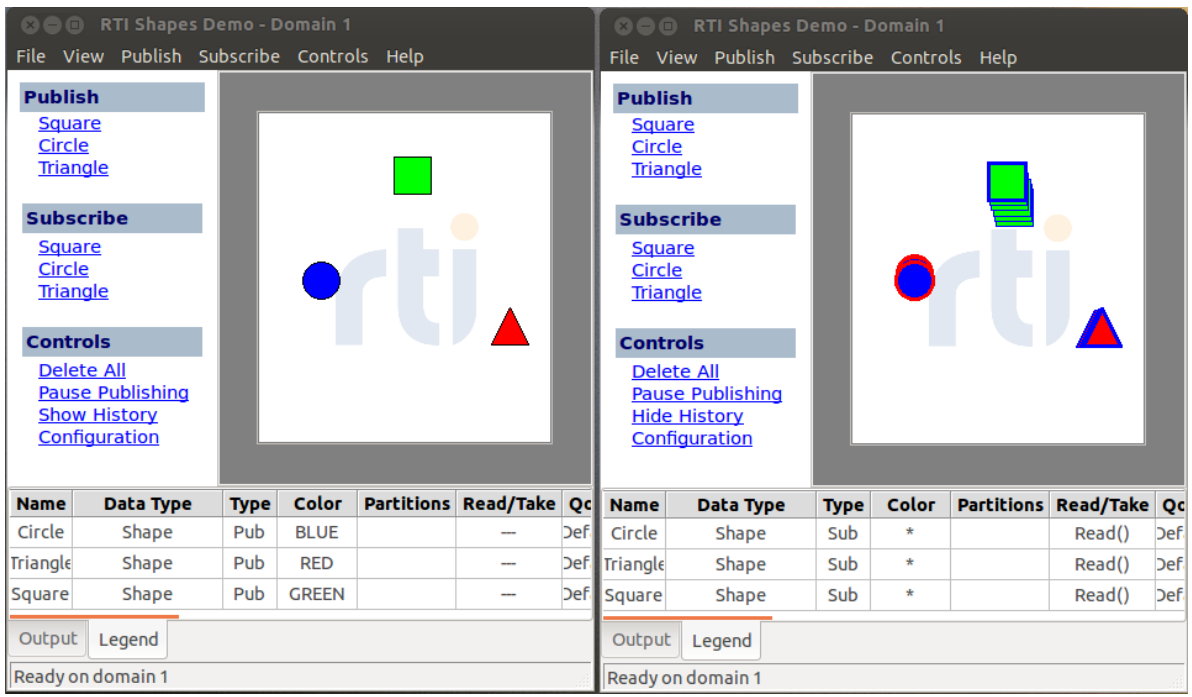


Figure : Left: Publishers. Right: Subscribers.

Shapes experiment

Basic publish/subscribe

- Start two instances of the Shapes Demo with
  - Data type: Shape
  - Domain: 0
  - Profile: Default::Default
- Start a publisher and a subscriber
  - Select same geometry for publisher and subscriber
  - For the rest: use default settings
- Catch and release the publisher shape with your mouse
  - Are changes in speed/orientation mirrored in the subscriber?
- Turn history on/off in subscriber's Controls menu
  - Notice how the 6 historical samples are turned on/off

Shapes experiment

Multiple instances

This experiment picks up where experiment 1 left off

- In the subscriber's controls menu choose "delete all"
- Create a subscriber with History $= 1$
- Create new publisher w/ same geometry but different color
- Subscriber receives the new geometry's data automatically
  - Sub of a topic, i.e. geo., get **all** data sent for **all** topic instances
  - The new geo./color: **another instance** of the **topic**, i.e. geo.
  - Therefore, subscriber receives this new data **automatically**
- Start a new instance of the Shapes Demo
  - Start a new publisher **similar** to one of the others
- Notice the flickering result in the subscriber
  - 2 pubs updating **same** instance (color) of the **same** topic (geo)
  - Subscriber receives position data from **two** different publishers

Shapes experiment

Extensible types

- Start two instances of the Shapes Demo with
  - Data type: Shape Extended
  - Domain: 0
  - Profile: Default::Default
- Start a publisher and a subscriber
  - Select same geometry for publisher and subscriber
  - Set *Shape fill style* and *Rotation speed* for publisher
  - Othewise, use default settings throughout
- Start a new instance of the Shapes Demo with
  - **Data type: Shape**. Domain: 0. Profile: Default::Default
  - Extended data types can be read by regular subscribers
  - Sort of backward compatibility on the data types
- Try subscribing to both regular and extended data types simultaneously with **one** subscriber - it works!

Shapes experiment

Content-filtered topics

Content-filtered topic

- Filter data received by the subscriber
- Helps control network and CPU load
- Only data that are of interest to Sub is sent
- E.g. radar detection of planes: Only want to know location of planes with 20km radius

Start two instances of the Shapes Demo

- Start a publisher
- Start a subscriber w/ *Content filter topic*
- Move and scale the coordinate filter in the subscriber canvas

Shapes experiment

Lifespan

Lifespan QoS controls how long data samples are considered valid, i.e. prevent sending data that is considered too old.

- Create a publisher and subscriber
- Pub: History = 100. Lifespan = 1000 ms
- Sub: History = 100.
- Sub showing last 100 samples from publisher's history queue
- Samples disappear in sub whenever they timeout (lifespan)

Try pausing the publisher and see the effects on the subscriber

Shapes experiment

Reliability and durability

Late joining nodes receive data published prior to their joining

- Start two instances of Shapes Demo

Publisher and subscriber

- Transient-Local Durability, Reliability, and History = 200

The shadow of geometries seen in the subscriber canvas

- Sub showing last 200 samples from Pub history queue
- Shadow appears immediately as all 200 historic samples received in one go

Shapes experiment

Time-based filtering

If subscribers are located on systems, e.g. mobile unit, not able to cope with all the data that the publisher is capable of sending

- The subscriber can choose only to get some data samples

Create two instances of the Shapes demo

- Pub: Default settings
- Sub: History = 1. Time based filter = 1000ms

In this case the publisher is only sending data to the subscriber once a second according to the subscribers time based filtering

- Results in jumping motion of subscriber

Java example

Set up the environment I/II

Set up the environment on development machine

Add to /etc/environment

```
CLASSPATH="$NDDSHOME/class"
```

Refresh environment variables

- Logout and login

Check you have Java and Make installed

- `java -version`, `javac -version`, and `make -version`

Java example

Set up the environment II/II

Set up the path for dynamic loadable libraries

- Java needs this
- Since Ubuntu 9.04, `LD_LIBRARY_PATH` cannot be set in
  - `$HOME/.profile`, `/etc/profile`, nor `/etc/environment`

Add to $HOME/.bashrc in the end of the file

export LD_LIBRARY_PATH = /usr/local/RTI/ndds.5.0.0/lib/x64Linux2.6gcc4.4.5jdk

Check to see if **all variables** are set correctly

- `echo` $[VARIABLE_NAME]

Build and run the Java example application

Build the example application

- `$NDDSHOME/example/JAVA/Hello_simple/build.sh`

Start the subscribing application

- `$NDDSHOME/example/JAVA/Hello_simple/runSub.sh`

Start the publishing application

- `$NDDSHOME/example/JAVA/Hello_simple/runPub.sh`

Live demo

Outline

Links: DDS

1. Documentation on **rti.com** and **opendds.org**
2. OMG's **DDS portal**
3. Inspiration to distributed systems **scenarios**

References

[1] Coulouris, G., J. Dollimore, T. Kindberg, and G. Blair (2001). *Distributed systems: Concepts and design*. Pearson.

[2] Eugster, P., P. Felber, R. Guerraoui, and A. Kermarrec (2003). The many faces of publish/subscribe. *ACM Computing Surveys 35*(2), 114–131.

[3] Pardo-Castellote, G., B. Farabaugh, and R. Warren (2005, August). An introduction to DDS and data-centric communications. Technical report, Real-Time Innovations.

[4] Tanenbaum, A. (1995). *Distributed operating systems*. Prentice Hall.