

ITONK / TIMICO

Data distribution service for real-time systems (DDS)

Christian Fischer Pedersen

Assistant Professor, PhD

cfp@iha.dk

Electrical and Computer Engineering
Aarhus University

April 30, 2013

Outline

Middleware	ONK
Publish/Subscribe	ONK
DDS I/II	ONK/MICO
Connex I/II: Shapes and Java example	ONK/MICO
DDS II/II	MICO
Connex II/II	MICO
Theory	MICO
References	

Outline

Middleware	ONK
Publish/Subscribe	ONK
DDS I/II	ONK/MICO
Connex I/II: Shapes and Java example	ONK/MICO
DDS II/II	MICO
Connex II/II	MICO
Theory	MICO
References	

Distributed software layers

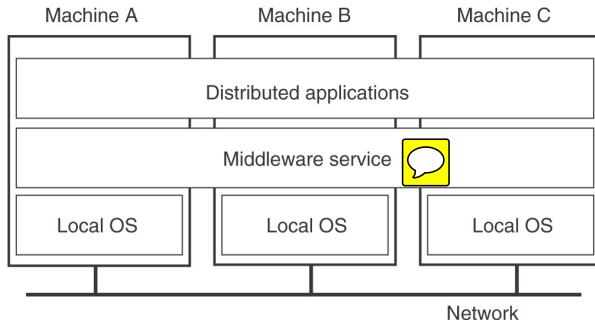


Figure: Courtesy of (Tanenbaum, 1995 [5])

Distributed system: Loose definitions

Definition (Tanenbaum, 1995 [5])

A distributed system is a collection of independent computers that appear to the users of the system as a single computer.

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

System in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.

Definition (Wikipedia, 2013)

A distributed system consists of multiple networked computers that interact with each other in order to achieve a common goal.

Distributed application: Loose definition

Definition

In a distributed application a number of parts are executed – most often concurrently - on one or more network nodes.

Process

- ▶ Instance of executing application
- ▶ May consist of multiple (concurrent) threads of execution

Challenges compared to sequential applications, e.g.

- ▶ Inter-process synchronization and interaction mechanisms
- ▶ Middleware provides support for these challenges

Middleware: 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

Homogenizes by abstracting over


- ▶ 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

Use middleware if...

- ▶ Hardware and underlying system software will support it, or can be modified to support it 
- ▶ If it simplifies design and implementation
- ▶ It supports needed configurability, portability, maintainability, etc. of the software

Do not use middleware if...

- ▶ It complicates design and implementation
- ▶ Introduces, e.g. extra costs, extra development time, risk, and stress into the project
- ▶ Introduces too much overhead on hardware, e.g. RAM, CPU, battery, **middleware is just software like any other**

Replace current middleware if...

- ▶ Better and/or cheaper technology has emerged on the market
- ▶ New project requirements are addressed better with other technology
- ▶ The middleware is being bypassed and/or not being used for the intended purposes, e.g. portability, performance, stability

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

- ▶ Not **one** middleware solution that 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, i.e.
 - ▶ 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)

Some middleware standards 1/6

Table 3.1: Examples of Real-world Standards Organizations and Middleware Standards in Embedded Systems Market

Standard type	Standard	General description
Aerospace and Defense	Aerospace Industries, Association of America, Inc. (AIA/NAS)	Association representing the nation's major aerospace and defense manufacturers, helping to establish industry goals, strategies, and standards. Related to national and homeland security, civil aviation, and space (www.aia-aerospace.org)
	ARINC (Avionics Application Standard Software Interface)	ARINC standards specify air transport avionics equipment and systems used by commercial and military aircraft worldwide (www.arinc.com)
	DOD (Department of Defense) – JTA (Joint Technical Architecture)	DOD initiative that supports the smooth flow of information via standards, necessary to achieve military interoperability (www.disa.mil)
	Multiple Independent Levels of Security/Safety (MILS)	Middleware framework for creating security-related and safety critical embedded systems
	SAE (Society of Automotive Engineers)	Defining aerospace standards, reports, and recommended practices (www.sae.org)
Automotive	Federal Motor Vehicle Safety Standards (FMVSS)	The Code of Federal Regulations are regulations issued by various agencies within the US Federal government (http://www.nhtsa.dot.gov/cars/rules/standards/)
	Ford Standards	From the engineering material specifications and laboratory test methods volumes, the approved source list collection, global manufacturing standards, non-production material specifications, and the engineering material specs and lab test methods handbook (www.ihs.com/standards/index.html)
	GM Global	Used in the design, manufacturing, quality control, and assembly of General Motors automobiles (www.ihs.com/standards/index.html)
	ISO/TS 16949 – The Harmonized Standard for the Automotive Supply Chain	Developed by the International Automotive Task Force (IATF), based on ISO9000, AVSQ (Italy), EAQF (France), QS-9000 (USA), and VDA6.1 (Germany), for example (www.iaob.org)
	Jaguar Procedures and Standards Collection	Contains Jaguar standards including Jaguar-Test Procedures Collection, Jaguar-Engine and Fastener Standards Collection, for example (www.ihs.com/standards/index.html)
Commercial and Home Office Automation	ANSI/AIM BC3-1995, Uniform Symbology Specification for Bar Codes	Specifies encoding general purpose all-numeric types of data, reference decode algorithm, and optional character calculation. This standard is intended to be identical to the CEN (commission for European normalization) specification (www.aimglobal.org/standards/)
	IEEE Std 1284.1-1997 IEEE Standard for Information Technology Transport Independent Printer/System Interface (TIP/SI)	Standard defining a protocol for printer manufacturers, software developers, and computer vendors that defines how data should be exchanged between printers and other devices (www.ieee.org)

Figure: Courtesy of (Noergaard, 2011 [3])

Some middleware standards 2/6

Table 3.1: Examples of Real-world Standards Organizations and Middleware Standards in Embedded Systems Market *continued*

Standard type	Standard	General description
Commercial and Home Office Automation	Postscript	Major printer manufacturers make their printers to support postscript printing and imaging standard (www.adobe.com)
Consumer Electronics	ARIB-BML (Association of Radio Industries and Business of Japan)	Responsible for establishing standards in the telecommunications and broadcast arena in Japan ⁵ (http://www.arib.or.jp/english/)
	ATSC (Advanced Television Standards Committee) DASE (Digital TV Application Software Environment)	Defines middleware that allows programming content and applications to run on DTV receivers. This environment provides content creators the specifications necessary to ensure that their applications and data will run uniformly on all hardware platforms and operating systems for receivers ⁶ (www.atsc.org)
	ATVEF (Advanced Television Enhancement Forum) – SMPTE (Society of Motion Picture and Television Engineers) DDE-1	The Advanced Television Enhancement Forum (ATVEF) is a cross-industry group that created an enhanced content specification defining fundamentals necessary to enable creation of HTML-enhanced television content. The ATVEF specification for enhanced television programming delivers enhanced TV programming over both analog and digital video systems using terrestrial, cable, satellite and Internet networks ⁷ (http://www.atvef.com/)
	CEA (Consumer Electronics Association)	An association for the CE industry that develops essential industry standards and technical specifications to enable interoperability between new products and existing devices ⁸ –Audio and Video Systems Committee –Television Data Systems Subcommittee –DTV Interface Subcommittee –Antennas Committee –Mobile Electronics Committee –Home Network Committee –HCS1 Subcommittee –Cable Compatibility Committee –Automatic Data Capture Committee (www.cea.org)
	DTVIA (Digital Television Alliance of China)	An organization made up of broadcasting academics, research organizations, and TV manufacturers targeting technology and standards within the TV industry in China (http://www.dtvia.org.cn/)
	DVB (Digital Video Broadcasting) – MHP (Multimedia Home Platform)	The collective name for a compatible set of Java-based open middleware specifications developed by the DVB Project, designed to work across all DVB transmission technologies (see www.mhp.org)
	GEM (Globally Executable MHP)	A core of MHP APIs, where the DVB-transmission-specific elements were removed. This allows other content delivery platforms that use other transmission systems to adopt MHP middleware (see www.mhp.org)

Figure: Courtesy of (Noergaard, 2011 [3])

Some middleware standards 3/6

Table 3.1: Examples of Real-world Standards Organizations and Middleware Standards in Embedded Systems Market *continued*

Standard type	Standard	General description
Consumer Electronics	HAVi (Home Audio Video Initiative)	Digital AV home networking software specification for seamless interoperability among home entertainment products. HAVi has been designed to meet the particular demands of digital audio and video by defining an operating-system-neutral middleware that manages multidirectional AV streams, event schedules, and registries, while providing APIs for the creation of a new generation of software applications ¹ (www.havi.org)
	ISO/IEC 16500 DAVIC (Digital Audio Visual Council)	Open interfaces and protocols that maximize interoperability, not only across geographical boundaries but also across diverse of interactive digital audio-visual applications and services (www.davic.org)
	JavaTV	Java-based API for developing interactive TV applications within digital television receivers. Functionality provided via the JavaTV API includes audio/video streaming, conditional access, access to in-band/out-of-band data channels, access to service information, tuner control for channel changing, on-screen graphics control, media synchronization, and control of the application life-cycle, for example ² (see java.sun.com)
	MicrosoftTV	Interactive TV systems software layer that contains middleware that provides a standard which combines analog TV, digital TV, and internet functionality (http://www.microsoft.com/tv/default.msp)
	OCAP (OpenCable Application Forum)	System software, middleware layer that provides a standard that allows for application portability over different platforms. OCAP is built on the DVB-MHP Java-based standard, with some modifications and enhancements to MHP (www.opencable.com)
	OpenTV	DVB compliant system software, middleware standard and software for interactive digital television receivers. Based on the DVB-MHP specification with additional available enhancements (www.opentv.com)
	OSGi (Open Services Gateway Initiative)	OSGi provides <i>Universal Middleware</i> for service-oriented, component-based environments across a range of markets (www.osgi.org)

Figure: Courtesy of (Noergaard, 2011 [3])

Some middleware standards 4/6

Table 3.1: Examples of Real-world Standards Organizations and Middleware Standards in Embedded Systems Market *continued*

Standard type	Standard	General description
Energy and Oil	AWEA (American Wind Energy Association)	Organization that develops standards for the USA wind turbine market (www.awea.org)
	International Electrotechnical Commission (IEC)	One of the world's leading organizations that prepares and publishes international standards for all electrical, electronic and related technologies – such as in the wind turbine generator arena (www.iec.ch)
	International Standards Organization (ISO)	One of the world's leading organizations that prepares and publishes international standards for energy and oil systems – such as in the nuclear energy arena (www.iso.org)
Industrial Automation and Control	International Electrotechnical Commission (IEC)	One of the world's leading organizations that prepares and publishes international standards for all electrical, electronic and related technologies – including in industrial machinery and robotics (www.iec.ch)
	International Standards Organization (ISO)	One of the world's leading organizations that prepares and publishes international standards for energy and oil systems – including in industrial machinery and robotics (www.iso.org)
	Object Management Group (OMG)	An international, open membership consortium developing middleware standards and profiles that are based on the Common Object Request Broker Architecture (CORBA™) and support a wide variety of industries, including for the field of robotics via the OMG Robotics Domain Special Interest Group (DSIG) (www.omg.org)
Medical	Department of Commerce, USA – Office of Microelectronics, Medical Equipment and Instrumentation	Website that lists the medical device regulatory requirements for various countries (www.ita.doc.gov/td/mdequip/regulations.html)
	Digital Imaging and Communications in Medicine (DICOM)	Standard for transferring images and data between devices used in the medical industry (medical.nema.org)
	Food and Drug Administration (FDA) USA	Among other standards, includes US government standards for medical devices, including class I non-life sustaining, class II more complex non-life sustaining, and class III life sustaining and life support devices (www.fda.gov)
	IEEE1073 Medical Device Communications	Standard for medical device communication for plug-and-play interoperability for point-of-care/acute care environments (www.ieee1073.org)
	Medical Devices Directive (EU)	Standards for medical devices for EU states for various classes of devices (europa.eu.int)

Figure: Courtesy of (Noergaard, 2011 [3])

Some middleware standards 5/6

Table 3.1: Examples of Real-world Standards Organizations and Middleware Standards in Embedded Systems Market *continued*

Standard type	Standard	General description
Networking and Communication	Cellular	Networking standards implemented for cellular phones (www.cdg.org and www.tiaonline.org)
	IP (Internet Protocol)	OSI Network layer protocol implemented within various network devices based on RFC 791 (www.faqs.org/rfcs)
	TCP (Transport Control Protocol)	OSI Transport layer protocol implemented within various network devices based on RFC 793 (www.faqs.org/rfcs)
	Bluetooth	Standards developed by the Bluetooth Special Interest Group (SIG) which allows for developing applications and services that are interactive via interoperable radio modules and data communication protocols (www.bluetooth.org)
	UDP (User Datagram Protocol)	OSI Transport layer protocol implemented within various network devices based on RFC 768 (www.faqs.org/rfcs)
	HTTP (Hypertext Transfer Protocol)	A WWW (world wide web) standard defined via a number of RFC (request for comments), such as RFC2616, 2016, 2069 to name a few (www.w3c.org/Protocols/Specs.html)
	DCE (Distributed Computing Environment)	Defined by the Open Group, the Distributed Computing Environment is a framework that includes RPC (remote procedure call), various services (naming, time, authentication), and a file system to name a few (http://www.opengroup.org/dce/)
	SOAP (Simple Object Access Protocol)	WWW Consortium specification that defines an XML-based networking protocol for exchange of information in a decentralized, distributed environment (http://www.w3.org/TR/soap/)

Figure: Courtesy of (Noergaard, 2011 [3])

Some middleware standards 6/6

Table 3.1: Examples of Real-world Standards Organizations and Middleware Standards in Embedded Systems Market *continued*

Standard type	Standard	General description
General Purpose	Networking and Communication Standards	TCP, Bluetooth, IP, etc.
	C# and .NET Compact Framework	Microsoft-based standard and middleware system for portable application development. Evolution of COM (www.microsoft.com)
	HTML (Hyper Text Markup Language)	A WWW (world wide web) standard for a scripting language processed by an interpreter on the device (www.w3c.org)
	Java and the Java Virtual Machine	Various standards and middleware systems from Sun Microsystems targeted for application development in different types of embedded devices (java.sun.com) Personal Java (pjava) Embedded Java, Java 2 Micro Edition (J2ME) The Real Time Specification for Java From J Consortium Real Time Core Specification
	SSL (Secure Socket Layer) 128-bit encryption	Security standard providing data encryption, server authentication, and message integrity, for example for a TCP/IP-based device (wp.netscape.com)
	Filesystem Hierarchy Standard	Standard that defines a file system directory structure hierarchy (http://www.linuxfoundation.org/)
	COM (Component Object Model)	Originally from Microsoft, a standard that allows for interprocess communication and dynamic object creation independent of underlying hardware and system software
	DCOM (Distributed COM)	Based on DCE-RPC and COM, that allows for interprocess communication and dynamic object creation across networked devices

Figure: Courtesy of (Noergaard, 2011 [3])

A multitude of standards exist

It is impossible to keep track of them all

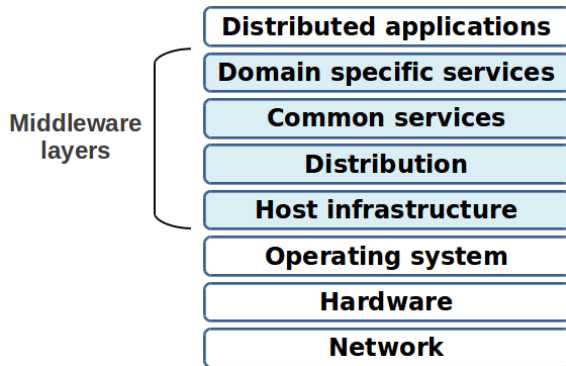
- ▶ Know a few within your area of expertise
- ▶ Learn **classes** of middleware
- ▶ When you need to find and investigate a **specific** technology, you know about the defining characteristics of the **class**

Middleware characteristics



Characteristic	Description
Adaptive	Makes applications adaptable to changing availability of systems resources
Flexibility and scalability	Makes applications configurable in terms of scalable functionality
Security	Can allow applications authorized access to resources
Portability	Makes applications able to run in different hardware/software environments, i.e. "write once, run-anywhere"
Connectivity and intercommunication	Makes applications able to transparently communicate with other remote applications through standardized interface

Middleware seen as layered software



Layer 1: Host infrastructure

- ▶ Encapsulates and enhances native OS mechanisms to create reusable network programming components
- ▶ The components abstract away many tedious and error-prone aspects of low-level OS APIs
- ▶ Examples
 - ▶ Java Virtual Machine (JVM)
 - ▶ Common Language Runtime (CLR)
 - ▶ Adaptive Communication Environment (ACE)

Layer 2: Distribution

- ▶ Distributed programming models whose reusable APIs and components automate and extend native OS capabilities
- ▶ Avoids hard-coding client-server application dependencies on object location, language, OS, protocols, and hardware
- ▶ Write distributed applications similar to stand-alone applications
- ▶ Examples (main functionality resides here)
 - ▶ OMG real-time common object request broker arch. (CORBA)
 - ▶ OMG real-time data-distribution service (DDS)
 - ▶ SUN remote method invocation (RMI)
 - ▶ W3C simple object access protocol (SOAP)

Layer 3: Common services

- ▶ Domain-independent services for business logic
- ▶ Functionality for recurring distributed system capabilities
 - ▶ Transactional behavior
 - ▶ Authentication, authorization and security
 - ▶ Database connection pooling and concurrency control
 - ▶ Active replication
 - ▶ Dynamic resource management
 - ▶ Fault tolerance, load balancing, real-time scheduling

Examples

- ▶ CORBA Component Model And Object Services
- ▶ Sun J2EE
- ▶ Microsoft .NET
- ▶ W3C Web Services

Layer 4: Domain specific services

Tailored to the requirements of particular domains, e.g.

- ▶ Telecom
- ▶ E-commerce
- ▶ Health care
- ▶ Process automation
- ▶ Aerospace

Examples

- ▶ CORBA domain interfaces
- ▶ Siemens MED Syngo: Common software platform for distributed electronic medical systems

Middleware decoupling

Middleware **decouples** 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

Remote procedure call and derivatives 1/2

- ▶ Call services on other nodes in network
- ▶ Function call look local even if non-local
- ▶ No time decoupling on consumer side
- ▶ No space decoupling: prod./consum. has ref. to each other
- ▶ One-to-one semantics not effective for broadcast
- ▶ Applied in, e.g. Java RMI, CORBA, MS DCOM

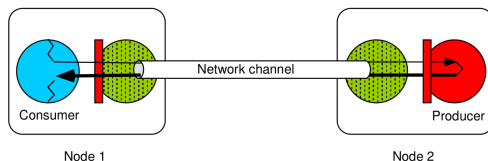


Figure: Courtesy of (Eugster et al., 2003 [2])

Remote procedure call and derivatives 2/2

- Two attempts to reduce coupling

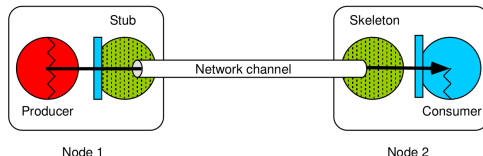


Figure: Asynchronous invocation/fire-and-forget: The producer does not expect reply, e.g. CORBA oneway. Courtesy of (Eugster et al., 2003 [2])

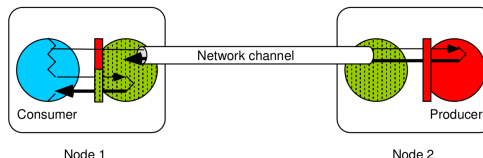


Figure: Future invocation: Producer not blocked and can access reply later when available via handle. Courtesy of (Eugster et al., 2003 [2])

Object request brokers



- ▶ Synchronous
- ▶ Applications send objects to broker
- ▶ Service is requested
- ▶ Broker routes requests to service provider
- ▶ Broker returns object to requestor
- ▶ Make an object method **look local** even if **non-local**
- ▶ Interface Description Language (IDL) to describe the data to be transmitted on remote calls
- ▶ Cf. e.g. CORBA, Java RMI

Notifications

- ▶ Subscribers asynchronously notify publishers w/ callback ref.
- ▶ Publishers asynchronously notify subscribers via callback
- ▶ Flow decoupling
- ▶ Coupled in time and space

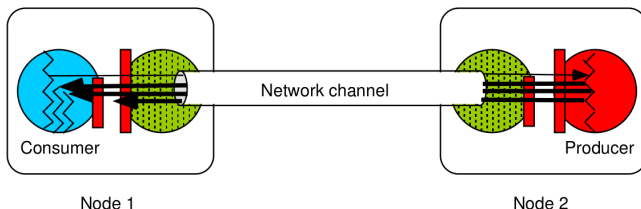


Figure: Courtesy of (Eugster et al., 2003 [2])

SQL orientation

- ▶ Time coupling, i.e. synchronous
- ▶ Applications send generic SQL requests to DBs
- ▶ Requests are translated to DB's native language, e.g. SQL
- ▶ DB returns information

Shared space

- ▶ Distributed **shared memory**, e.g. tuple space
- ▶ Communication between hosts via shared space/memory
- ▶ Insertion and removal of tuples to/from tuple-space
- ▶ Time decoupling: After producing, consuming anytime
- ▶ Space decoupling: Prod./consum. do not know each other
- ▶ **Flow coupled**: In main flow on consumer side

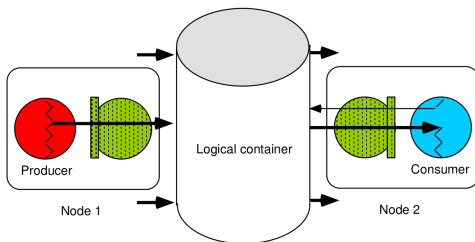
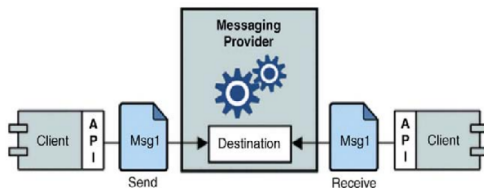


Figure: Courtesy of (Eugster et al., 2003 [2])

Message orientation

- ▶ Asynchronous
- ▶ Applications send messages
- ▶ Messages are queued and ordered
- ▶ Other applications read the queue
- ▶ Actions are taken
- ▶ Example: Java message service, SOAP



Message passing

Time decoupling

- ▶ Time decoupling for producer
- ▶ No time decoupling for consumer

No space decoupling

- ▶ Destination of message is explicitly specified
- ▶ Coupled to the socket abstraction

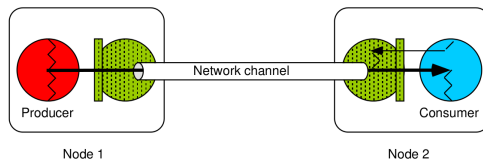


Figure: Courtesy of (Eugster et al., 2003 [2])

Message queuing

- ▶ Insertion and removal of messages
- ▶ Similar to *Shared Space*, but w/ transactional timing and ordering
- ▶ Messages are shared in FIFO queue
- ▶ Producers append, consumers dequeue
- ▶ Time and space decoupling achieved
- ▶ No flow decoupling on consumer side

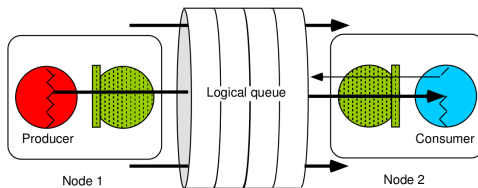


Figure: Courtesy of (Eugster et al., 2003 [2])

Publish/Subscribe

- ▶ Publishers publish messages to broker
 - ▶ Messages are categorized
 - ▶ Not send to specific subscriber(s)
- ▶ Subscribers receive messages from broker
 - ▶ Registered for particular message categories
 - ▶ Do not know the specific publisher(s)
 - ▶ Filtering is based on: **topic, content, or type**
- ▶ Decouples publisher and subscriber: **time, space and flow**
 - ▶ Loose coupling for large scale and dynamic networks

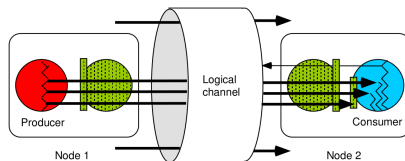


Figure: Courtesy of (Eugster et al., 2003 [2])

Decoupling interaction paradigms

Abstraction	Space de-coupling	Time de-coupling	Flow de-coupling
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

Figure: Courtesy of (Eugster et al., 2003 [2])

Outline

Middleware	ONK
Publish/Subscribe	ONK
DDS I/II	ONK/MICO
Connex I/II: Shapes and Java example	ONK/MICO
DDS II/II	MICO
Connex II/II	MICO
Theory	MICO
References	

Basic object based Publish/Subscribe system

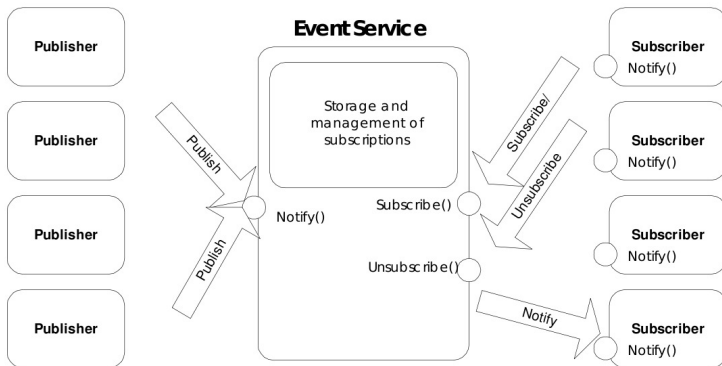


Figure: Courtesy of (Eugster et al., 2003 [2])

Decoupling by Publish/Subscribe

- ▶ The event-service (broker) 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

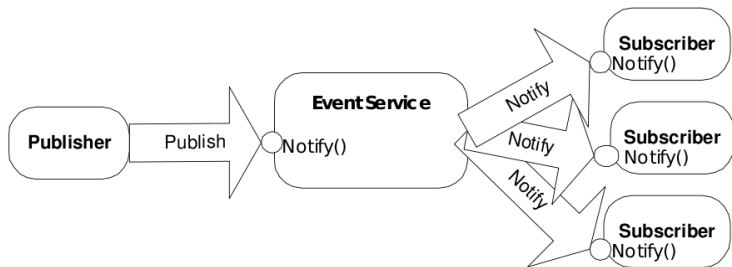


Figure: Courtesy of (Eugster et al., 2003 [2])

Time decoupling

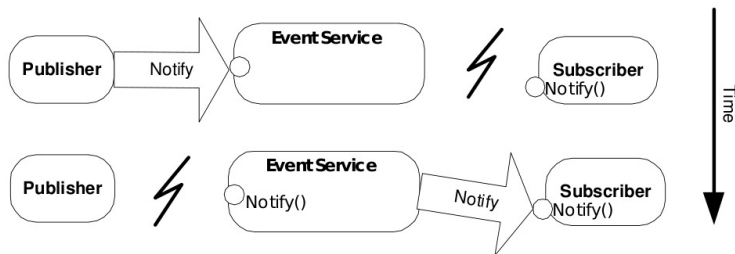


Figure: Courtesy of (Eugster et al., 2003 [2])

Flow decoupling

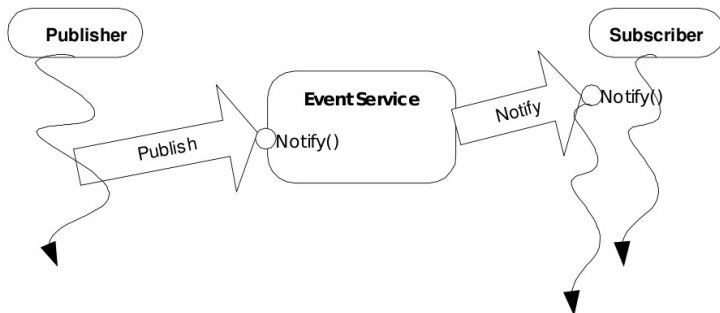


Figure: Courtesy of (Eugster et al., 2003 [2])

Space, time, and flow decoupling in P/S

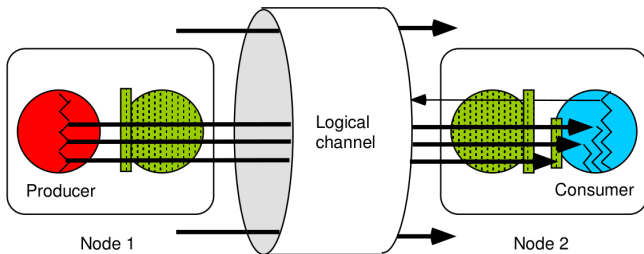


Figure: Courtesy of (Eugster et al., 2003 [2])

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 (aka. 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

Middleware ONK

Publish/Subscribe ONK

DDS I/II ONK/MICO

Connex I/II: Shapes and Java example ONK/MICO

DDS II/II MICO

Connex II/II MICO

Theory MICO

References

About the data distribution service for RT systems

Data distribution service for real-time systems (DDS)

- ▶ Specification of a Publish/Subscribe (P/S) middleware for distributed real-time (RT) systems

DDS supports elements from all three subscription models


- ▶ Topic based
- ▶ Content based
- ▶ Type based

See the P/S subscription models in the paper

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

Background of DDS

Two major proprietary DDS solutions available for several years

- ▶ NDDS from Real-Time Innovations 
- ▶ Splice from Thales – now OpenSplice DDS from Prismtech

Teamed together in 2004 to create the DDS specification

- ▶ Approved by Object Management Group, cf. www.omg.org
- ▶ The standards body that maintains the DDS specification
- ▶ OMG maintains also, e.g. UML and CORBA

DDS specifies a data-centric publish-subscribe (DCPS) model

- ▶ E.g. an **augmentation** to 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

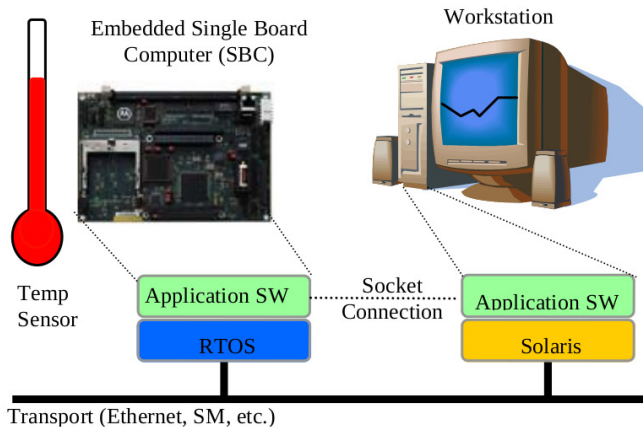


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Simple distributed application with DDS

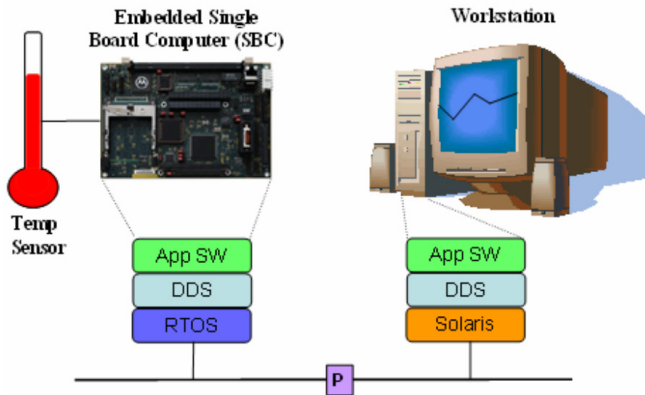


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Advantages of DDS

- ▶ Based on a simple P/S communication paradigm
- ▶ Flexible and adaptable architecture
 - ▶ Supports auto-discovery of **new** or **stale** applications
- ▶ Low overhead on memory and CPU cycles
- ▶ Deterministic data delivery
 - ▶ Underlying transport, e.g. Ethernet, does not automatically become deterministic
- ▶ Dynamically scalable
- ▶ Efficient use of transport bandwidth
- ▶ Supports one-to-one, one-to-many, many-to-one and many-to-many communications
- ▶ Large number of configuration parameters, e.g. for QoS

The DDS infrastructure

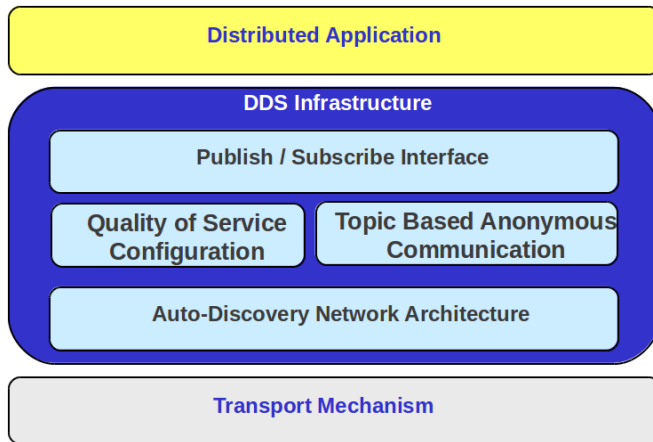


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Data-centricity

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

- ▶ Rate of publications
- ▶ Rate of subscriptions
- ▶ How long the data is valid
- ▶ Etc...

QoS enables

- ▶ Custom tailored communication mechanism based on application requirements

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...

Real-time application requirements to DDS

Predictable distribution of data with minimal overhead

- ▶ Requires the ability to control QoS properties that effect predictability, overhead and resource utilization

Flexible and robust **scalability**

- ▶ To hundreds or thousands of publishers and subscribers

A DDS application: Bluefin Robotic's UUV

Typical payloads include

- ▶ Side-scan radar
- ▶ Synthetic aperture sonar
- ▶ Hydrophone arrays

Other sensors include

- ▶ Doppler velocity logs
- ▶ Acoustic Doppler current profilers
- ▶ Conductivity
- ▶ Temperature
- ▶ Fluorometer
- ▶ GPS

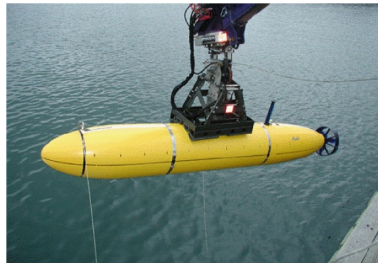


Figure: Unmanned underwater vehicle (UUV). Courtesy of Real-Time Innovations

OMG's DDS specification structure

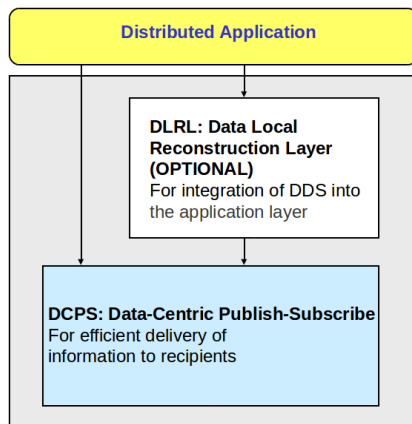


Figure: The two main DDS components: DLRL and DCPS

DLRL: Data Local Reconstruction Layer

- ▶ Optional layer
- ▶ Interface to DCPS data
- ▶ Automatically reconstructs the data locally from the updates
- ▶ Allows the application to access the data as if local
- ▶ The middleware not only propagates the information to all interested subscribers but also updates a local copy of the data

DCPS: Data Centric Publish Subscribe

- ▶ Allow the middleware to pre-allocate resources to avoid unpredictable dynamic resource allocation
- ▶ Avoid unbounded or hard-to-predict resources
- ▶ Efficient in its use of resources, i.e. low overhead
- ▶ Minimize the need to make copies of the data

DCPS: Advantages with typed interfaces

Simplicity

- ▶ Developer manipulates constructs that naturally represent data

Safety

- ▶ Verification can be performed at compile time

Efficiency

- ▶ The execution code can rely on the knowledge of the exact data type e.g. to pre-allocate resources

DDS (DCPS) entities

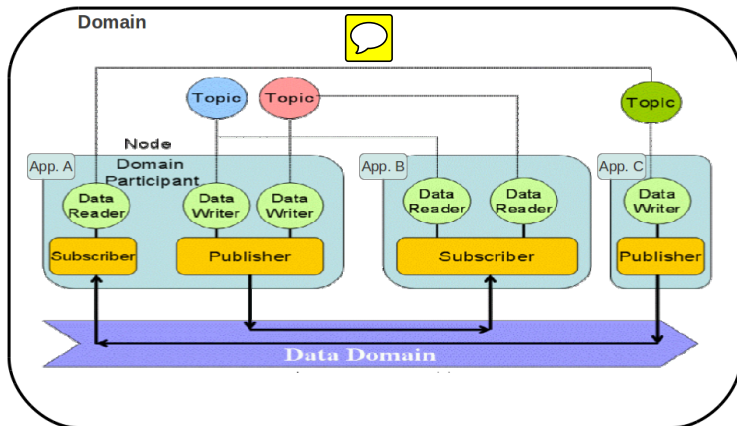


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Responsibilities of publisher and subscriber

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

- ▶ A **DataWriter** object acts as a **typed accessor** to the publisher
- ▶ The **application** use the DataWriter to communicate a data-object of a given type

A **Subscriber** is an object responsible for receiving published data and making it available to the application

- ▶ To access the received data the **application** must use a **typed DataReader**

Topics

- ▶ Connection point between publishers and subscribers
- ▶ Publisher/subscriber Topics must 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

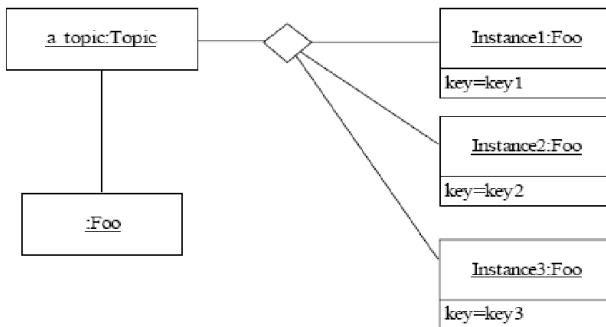
- ▶ In topic type definition data-element(s) can be set to **key**
- ▶ The key can be used for sorting incoming data
- ▶ Keys support scalability, e.g. multiple temp. sensor nodes
- ▶ No need for individual topics with different names for each temp. sensor node when the type is the same
- ▶ With keys: Only one Topic needed

Example (Topic Type)

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

Topic: Instances and keys

- ▶ A **Topic** corresponds to a single data type
- ▶ **Keys** are for sorting/filtering incoming data
- ▶ Without keys: Needed to create individual **Topics** with like **Types** for each publisher



Content filtered topics

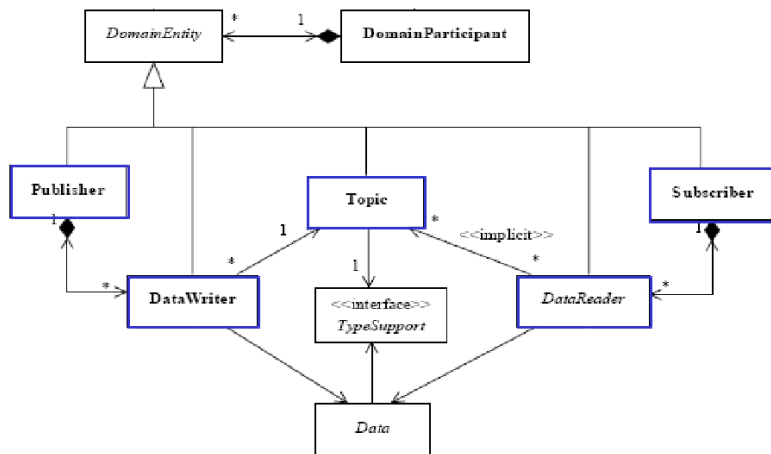
A **ContentFilteredTopic** allows to declare a filter expression

- ▶ Data samples of specified Topic will be filtered

Temperature sensor example

- ▶ Filters on temperature value
- ▶ Subscribers only receives and process data when a temperature exceeds a specific limit
- ▶ Hence, reduce information overload for subscribers

Overview: Core DDS concepts



Single domain system

Domain: Binds individual applications together for communication

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

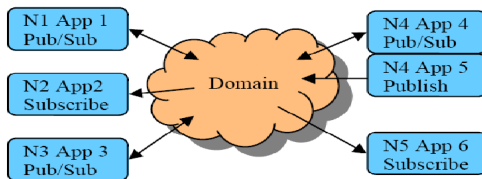


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Multiple domains system

Domain: Binds individual applications together for communication

- ▶ DataWriters and DataReaders w. like data types will communicate within a single domain – not across domains
- ▶ For data isolation, e.g. one domain per functional area
- ▶ Domain C: New functionality tested in isolation

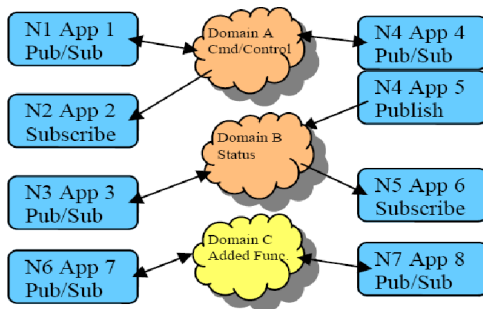


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Publication model

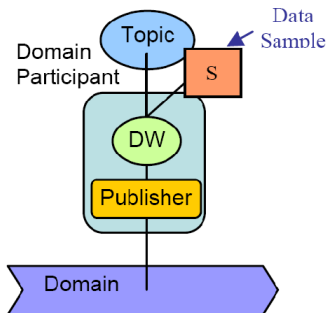


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Data readers and subscribers

Three methods for receiving data

- ▶ Listener callback routine
 - ▶ Called by DDS when data is received
- ▶ Polling the data reader
 - ▶ Query the DataReader to see if data is available
- ▶ 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

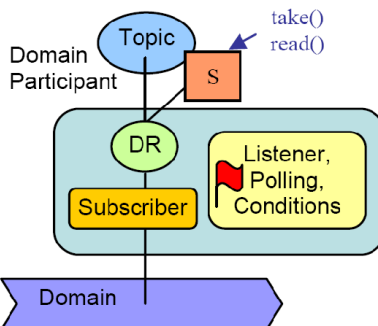


Figure: Courtesy of (Pardo-Castellote et al., 2005 [4])

Built-in topics

As part of its operation, the middleware must **discover** and possibly keep track of the presence of remote entities such as a new participant in the domain

- ▶ This info may also be important to the application, which may want to react to this discovery, or else access it on demand

To make this information accessible to the application, the DCPS specification introduces a set of **built-in topics** and corresponding **DataReader** objects that can then be used by the application

DDS QoS parameters

- ▶ Deadline
- ▶ Destination Order
- ▶ Durability
- ▶ Entity Factory
- ▶ Group Data
- ▶ History
- ▶ Latency Budget
- ▶ Lifespan
- ▶ Liveliness
- ▶ Ownership
- ▶ Ownership Strength
- ▶ Partition
- ▶ Presentation
- ▶ Reader Data Lifecycle
- ▶ Reliability
- ▶ Resource Limits
- ▶ Time-Based Filter
- ▶ Topic Data
- ▶ Transport Priority
- ▶ User Data
- ▶ Writer Data Lifecycle

Selected QoS parameters

Deadline indicates

- ▶ The publish frequency of DataWriter
- ▶ The read frequency of DataReader
- ▶ Often used in applications with periodical updates

Durability indicates

- ▶ 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

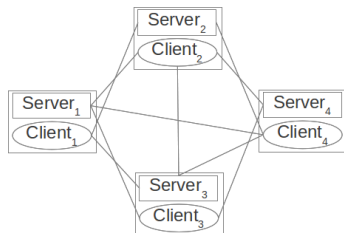
Resource Limits indicates

- ▶ Amount of local memory DDS can use, e.g. maximum number of instances per Topic
- ▶ Good for embedded systems w. limited memory

CORBA and DDS I

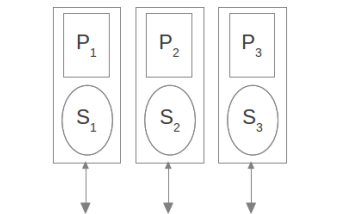
CORBA

- ▶ Distributed object model
- ▶ Remote method calls
- ▶ Connection-oriented



DDS

- ▶ Data distribution model
- ▶ Multicast transport
- ▶ Flexibility via QoS



CORBA and DDS II

- ▶ The only things DDS has in common with CORBA is that it uses a subset of IDL. Other than this, CORBA and DDS are two completely different standards and two completely different and complementary technologies
- ▶ CORBA and DDS address different needs. Complex systems often need both

Summary

- ▶ Data Distribution Service for real-time systems: an OMG specification
- ▶ Simple data communication model
- ▶ Enabling complex data patterns
- ▶ Topics allow endpoint nodes to be abstracted from each other
- ▶ Nodes can enter/leave the distributed application dynamically
- ▶ DDS provides an API for sending and receiving data
- ▶ Developers do not need to worry about net. programming

Outline

Middleware	ONK
Publish/Subscribe	ONK
DDS I/II	ONK/MICO
Connex I/II: Shapes and Java example	ONK/MICO
DDS II/II	MICO
Connex II/II	MICO
Theory	MICO
References	

RTI Connex DDS

- ▶ RTI (Real-Time Innovations): US based company
- ▶ RTI Connex: Closed source impl. of OMG's DDS standard
- ▶ OMG: Object Management Group
- ▶ Connex covers heterogeneity
 - ▶ C, C++, .Net, Java, Ada
 - ▶ Windows, Linux, Solaris, AIX & other enterprise class system
 - ▶ VxWorks, Integrity, LynxOS & other RT/embedded systems
- ▶ We will **only** look at Connex from a **Linux** perspective



Install RTI Connex

- ▶ Get a 32 or 64 bit Linux or Windows version from me
- ▶ Get a license file, `rti_license.dat`, from me
- ▶ **Do not distribute the SW and licence!**
- ▶ Current ver.: RTI Connex Professional Edition version 5.0.0
- ▶ E.g.: `sudo`
`./RTI_Connex_Professional_Edition-5.0.0-RHEL6_64_lic.sh`
- ▶ Install in, e.g. `/usr/local`
- ▶ Follow the on-screen installation instructions
- ▶ Copy the license file into the installation root, e.g.
`/usr/local/RTI/`

Check the installation

Run the `rtilauncher`

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

Learning by the Shapes Demo

RTI Shapes Demo - Domain 1

File View Publish Subscribe Controls Help

Publish

[Square](#)
[Circle](#)
[Triangle](#)

Subscribe

[Square](#)
[Circle](#)
[Triangle](#)

Controls

[Delete All](#)
[Pause Publishing](#)
[Show History](#)
[Configuration](#)

Name	Data Type	Type	Color	Partitions	Read/Take	QoS
Circle	Shape	Pub	BLUE		—	Def
Triangle	Shape	Pub	RED		—	Def
Square	Shape	Pub	GREEN		—	Def

Output Legend

Ready on domain 1

RTI Shapes Demo - Domain 1

File View Publish Subscribe Controls Help

Publish

[Square](#)
[Circle](#)
[Triangle](#)

Subscribe

[Square](#)
[Circle](#)
[Triangle](#)

Controls

[Delete All](#)
[Pause Publishing](#)
[Hide History](#)
[Configuration](#)

Name	Data Type	Type	Color	Partitions	Read/Take	QoS
Circle	Shape	Sub	*		Read()	Def
Triangle	Shape	Sub	*		Read()	Def
Square	Shape	Sub	*		Read()	Def

Output Legend

Ready on domain 1

Figure: Right: Publishers. Left: Subscribers.

Run the Shapes Demo

- ▶ Either `rtishapesdemo` or
- ▶ RTI Launcher | Utilities

Shapes experiment 1/7: 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
 - ▶ Otherwise, use default settings throughout
- ▶ 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 2/7: Multiple instances

This experiment picks up where experiment 1 left off

- ▶ In the sub's Controls menu: Delete all
- ▶ Create a subscriber with History = 1
- ▶ In pub canvas: Create new pub w/ same geo. but diff. color
- ▶ **Notice:** Sub receives 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 3/7: 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
 - ▶ Otherwise, 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 4/7: 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 5/7: 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 Pub's history queue
- ▶ Samples disappear in sub whenever they timeout (lifespan)

Try pausing the publisher and see the effects on the subscriber

Shapes experiment 6/7: Reliability and durability

Late joining nodes receive data that was 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 7/7: 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

```
PATH="/usr/local/RTI/ndds.5.0.0/scripts"  
NDDSHOME="/usr/local/RTI/ndds.5.0.0"  
CLASSPATH="$NDDSHOME/class"  
RTI_LICENSE_FILE="$NDDSHOME/rti_license.dat"
```

- ▶ Apply settings: Log out and in (not elegant but effective)

Check you have Java and GNU 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 (in EOF) \$HOME/.bashrc

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

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

- ▶ echo \${VARIABLE_NAME}

Java: Build and run the 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

Middleware	ONK
Publish/Subscribe	ONK
DDS I/II	ONK/MICO
Connex I/II: Shapes and Java example	ONK/MICO
DDS II/II	MICO
Connex II/II	MICO
Theory	MICO
References	

Outline

Middleware	ONK
Publish/Subscribe	ONK
DDS I/II	ONK/MICO
Connex I/II: Shapes and Java example	ONK/MICO
DDS II/II	MICO
Connex II/II	MICO
Theory	MICO
References	

Outline

Middleware ONK

Publish/Subscribe ONK

DDS I/II ONK/MICO

Connex I/II: Shapes and Java example ONK/MICO

DDS II/II MICO

Connex II/II MICO

Theory MICO

References

Outline

Middleware ONK

Publish/Subscribe ONK

DDS I/II ONK/MICO

Connex I/II: Shapes and Java example ONK/MICO

DDS II/II MICO

Connex II/II MICO

Theory MICO

References

Links: DDS

1. Documentation on [rti.com](https://realtime.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] Noergaard, T. (2011). *Demystifying embedded systems middleware*. Elsevier.
- [4] Pardo-Castellote, G., B. Farabaugh, and R. Warren (2005, August). An introduction to DDS and data-centric communications. Technical report, Real-Time Innovations.
- [5] Tanenbaum, A. (1995). *Distributed operating systems*. Prentice Hall.