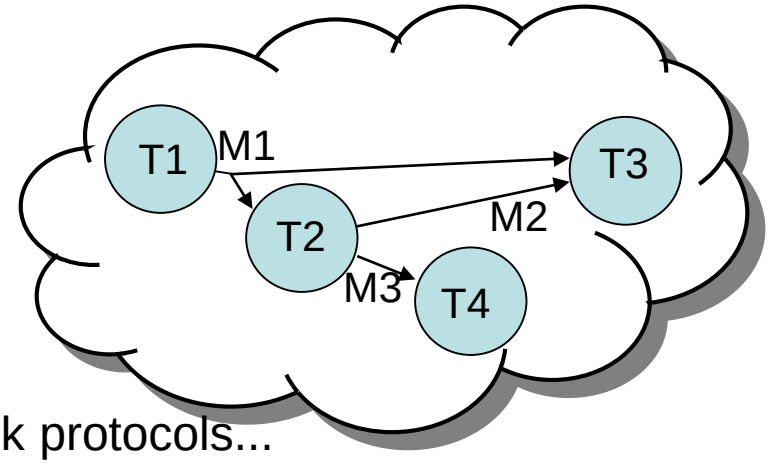
Distributed Systems
MIEEC, Fall 2018

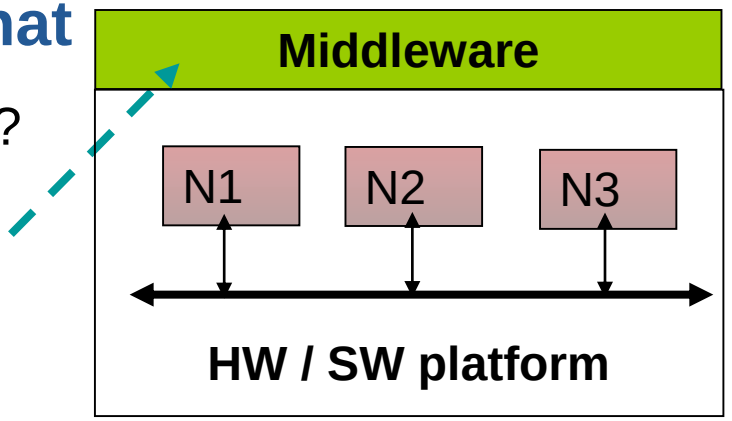
Middleware

Luis Almeida
Faculty of Engineering – **University of Porto**
Portugal

Abstracting away platform details

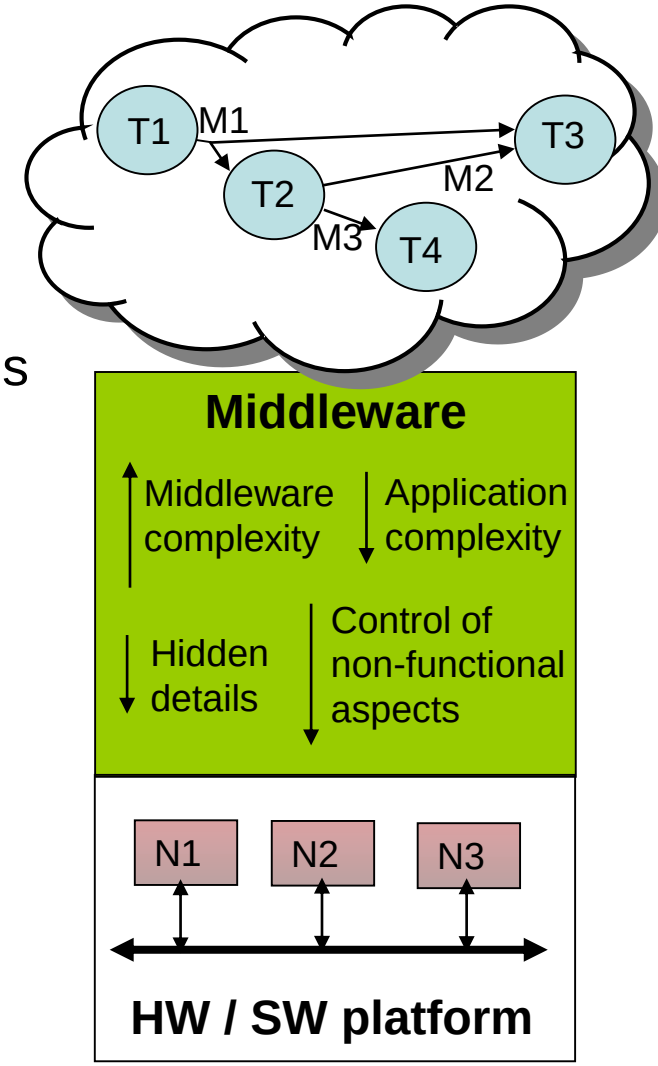
- **Applications are executed on HW / SW platforms**
  - Computing and communication
- **Implying many idiosyncrasies**
  - Dependence on HW / SW features
    - Processors, OSs, languages, network protocols...
- **How to develop applications that**
  - Are agnostic to such idiosyncrasies?
  - Still delivering their services and exhibiting the desired properties...
  - And executing on a distributed platform...





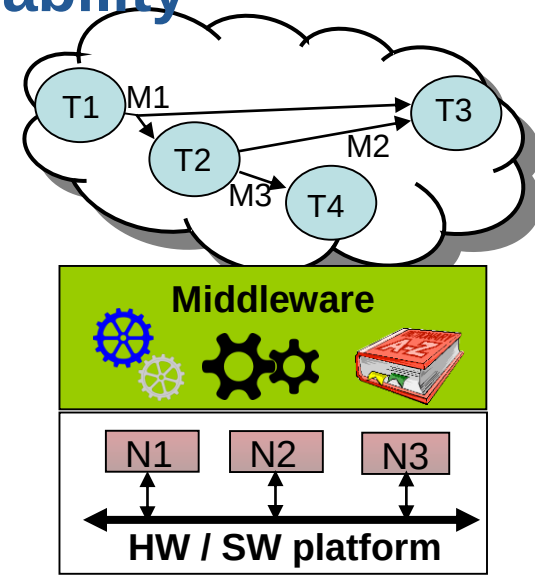
Middleware

- **The middleware is a SW layer that**
  - Hides unnecessary platform details
  - Simplifies development, adds new services
- **But it implies trade-offs**
  - The HW/ SW platform has a profound impact on non-functional properties
    - timing, performance, dependability...
  - The simpler it is to develop applications the more complex the middleware is
    - And vice-versa



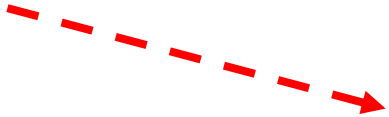
Typical middleware requirements

- **Simplify application development**
  - High level abstractions and simple interfaces
    - Hiding heterogeneity and low-level communication
- **Support communications and interoperability**
  - Integration of modules from different sources
    - Automatic discovery and configuration
- **Provide efficient resource utilization**
  - Processors, networks, memory…
- **Offer typically required services**
  - Synchronization, filters, control…
- **Support integration of devices with low resources**
  - Connection to simple embedded devices



Middleware issues

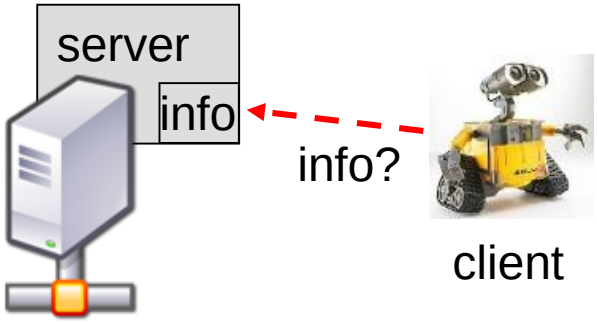
- **Some relevant aspects**
  - **Transparency** wrt Distribution, OS, Languages...
  - Support for different programming paradigms (SO, OO, CB...)
  - **Cooperation model**
    - Client-Server
    - Producer-Consumer
    - Producer-Distributor-Consumer
    - Publisher-Subscriber
    - Shared memory (RTDB, Blackboard...)
    - Peer-to-peer



How the nodes exchange information within a distributed application

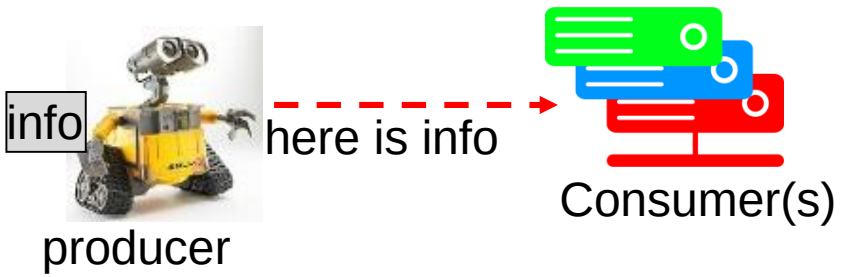
Cooperation models

- **Client-Server**
  - **Servers hold** information. **Clients request** it
    - Transactions **triggered by the receivers** (clients)
    - Typically based on **unicast** transmission (one to one comm.)
  - Transactions can be
    - **synchronous** (client blocks until server answers)
      - Attention: **communication** time is **inside** the **computing loop**…
    - **asynchronous** (client follows execution after issuing the request)
  - Requires **naming service**
  - Adequate for sporadic use of the data
  - Technologies: **RPC, RMI, CORBA, ROS**



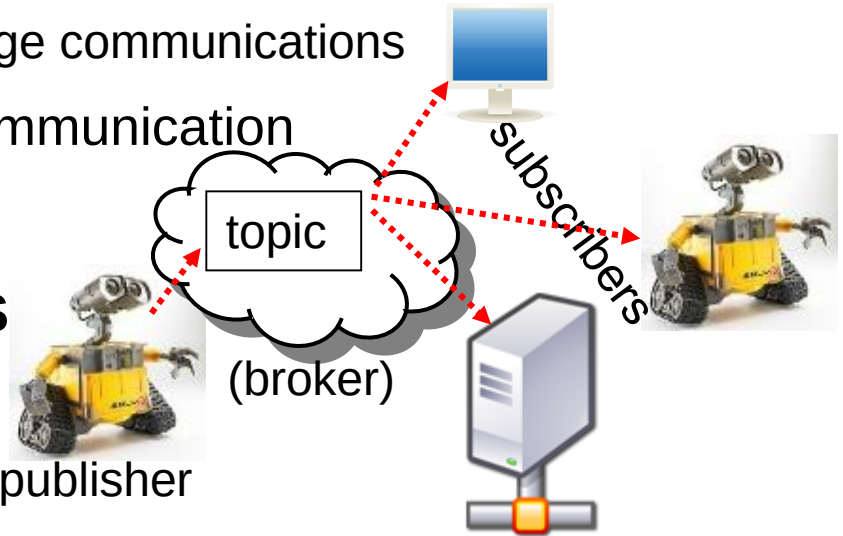
Cooperation models

- **Producer-Consumer**
  - **Producers disseminate** information. **Consumers use** it
    - Transactions **triggered by the senders** (producers).
    - Based on **broadcast** transmission (each message is received by all)
      - **Anonymous asynchronous** communication,
      - Attention: Security constraints?
  - **Communications** time is **outside** the **computing loop**
  - Adequate to regular state dissemination
  - Technologies: **CANopen**



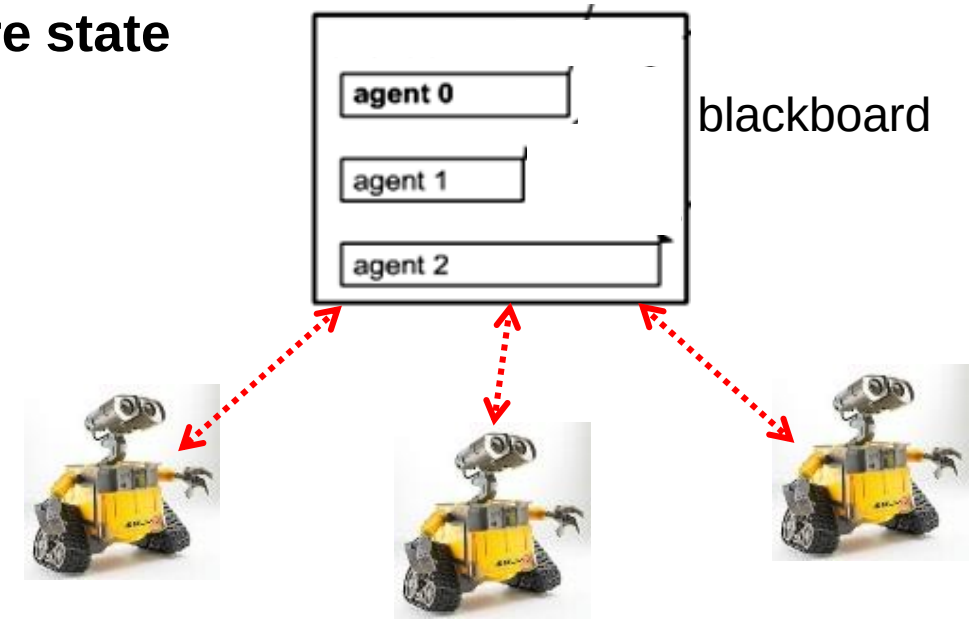
Cooperation models

- **Publisher-Subscriber**
  - Concept of **group communication** around data entities (topics)
  - Nodes must adhere to groups either as
    - **publisher** (produces information) or
    - **subscriber** (consumes information)
  - Transactions are **triggered by the publisher** of a group and disseminated among the **respective subscribers**, only (multicast)
    - Typically uses a broker to manage communications
  - **Anonymous asynchronous** communication
  - Regular and sporadic data
  - Technologies: **RTPS, DDS, ROS MQTT, OM2M**



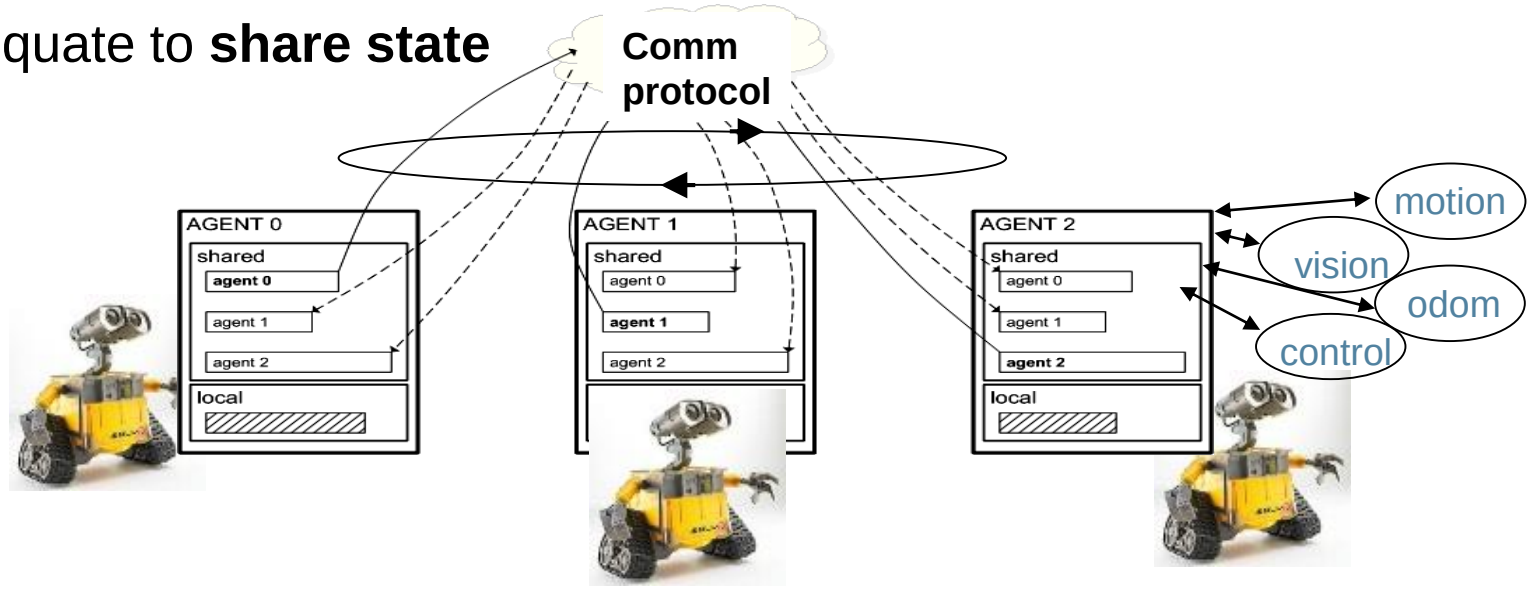
Cooperation models

- **Shared memory – Blackboard**
  - Communicating processes **read / write from a common area**
    - This common area (**Blackboard**) may reside in a different computer
    - Attention: **Communication** time **inside** the **computing loop**
  - **Anonymous asynchronous** communication
  - Adequate to **share state**



Cooperation models

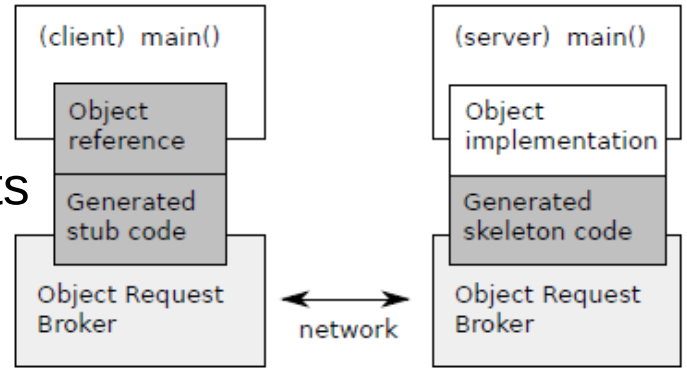
- **Shared memory – Real-Time Database (RTDB)**
  - Communicating processes **read / write from a common area**
    - Common area is **replicated** in all agents providing **local data access**
      - **Real-time Database (RTDB)**
    - **Communications** time **outside** the **computing loop**
  - **Anonymous asynchronous** communication
  - Adequate to **share state**



Middleware technologies

Middleware technologies

- **CORBA – Common Object Request Broker Architecture**
  - www.corba.org
  - Open specification proposed by OMG
  - **Purpose:** Clients use remote objects as if they were local
- **Main features**
  - Interoperability between languages and platforms
    - Windows, Linux, Unix, MacOS, QNX, VxWorks, …
    - Ethernet, CAN, Internet, ...
    - Ada, C, C++, Java, Python, …
  - Multiple vendors & open-source products



Middleware technologies

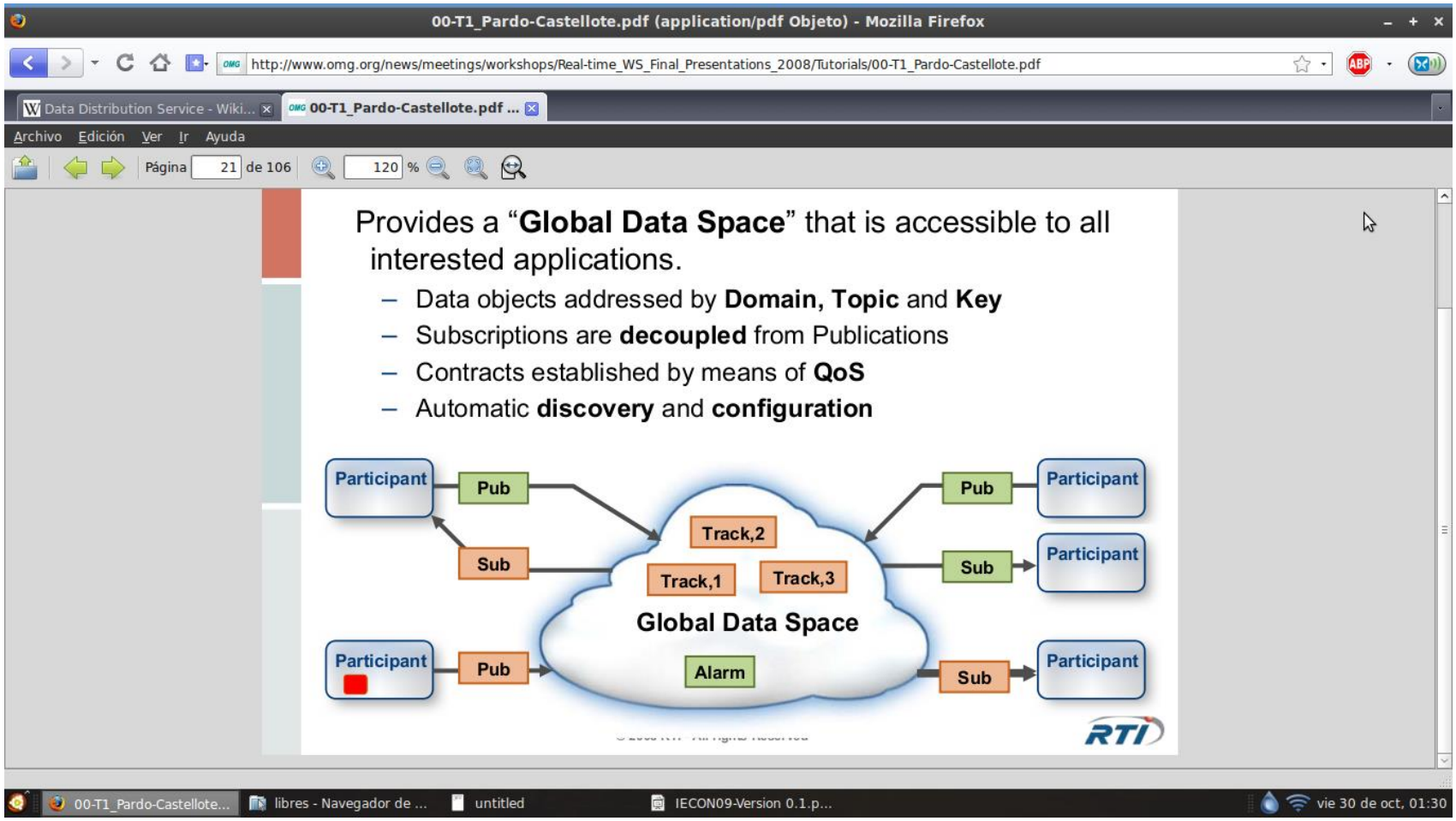
- **CORBA implementations / profiles used in robotics**
  - **RT-CORBA:** Support for applications with end-to-end timing constraints
  - **CORBA/e:** For embedded devices (Minimum CORBA and Micro CORBA)
  - **RTC – Robotics Technology Component**
    - Component model, with structural and behavior features typical in robotics
  - **TAO:** Open source, QoS support for real-time and embedded systems
  - **MIRO:** (TAO) sensor/actuator services as network transparent CORBA objects
  - **RT-Middleware:**
    - Component model with real-time functional elements – RT-Components
    - Applications designed in UML aggregating RT-Components

Middleware technologies

- **DDS – Data Distribution Service**
  - portals.omg.org/dds
  - Open specification proposed by OMG
  - **Purpose:** provide a Publisher-Subscriber data-centric model for distributed real-time applications
- **Main features**
  - Anonymous communication with asynchronous channels
    - Platform independence
    - Handles addressing, delivery, control flow
  - Global distributed database of **Topics**
    - Unique names, abstract data type, QoS parameters
    - **Signals**, **Streams** and **States**

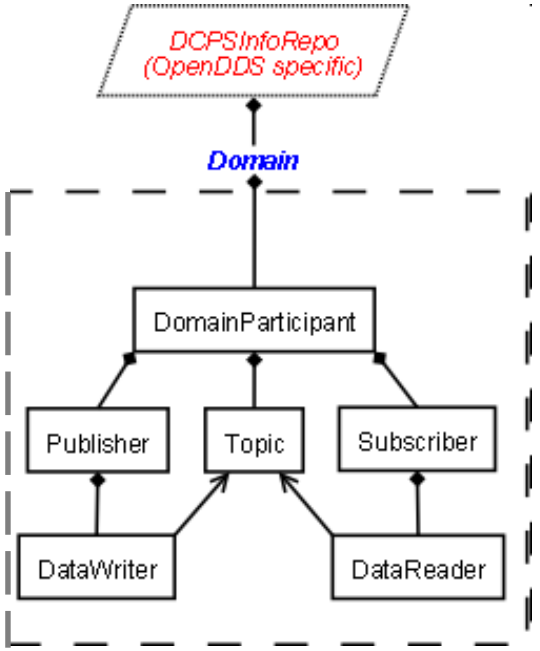
Middleware technologies

- **DDS – Global Data Space (GDS)**



Middleware technologies

- **DDS implementations**
  - **Connext DDS**
    - most complete implementation (commercial, by RTI)
  - **OpenDDS**
    - Open source implementation by OMG
  - **RTPS**
    - Real-Time Publisher-Subscriber protocol
    - Provides interoperability to DDS
      - RT communication over IP (UDP/IP),
      - Fault-tolerance, Extensibility, Plug&play,
      - Configurability, Modularity, Scalability,
      - Type-safety
    - ORTE - Open source RTPS implementation



Middleware technologies

- **SOAP – Simple Object Access Protocol**
  - www.w3.org/TR/soap
  - Open specification proposed by W3C
  - **Purpose:** exchange structured and typed information based on XML
    - XML-RPC
- **Main features**
  - Stateless, asynchronous messaging system
  - Agnostic to application semantics
  - Modular packaging model and encoding mechanism
    - **Envelope:** definition of what, who and whether optional/mandatory
    - **Encoding rules:** serialization mechanism
    - **RPC representation:** convention to represent RPCs and responses

Middleware technologies

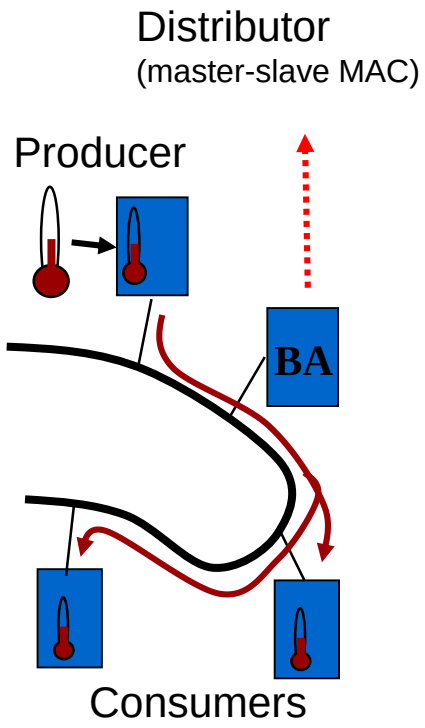
- **SOAP implementations / profiles used in robotics**
  - **ROS: Robot Operating System**
    - Hardware abstraction, low-level device control, commonly used functions, message-passing, package management
    - **Client-Server** and **Publisher-Subscriber** (ROS Topics) models
    - Framework with contributed packages: SLAM, planning, perception, …
    - Free and Open-source

    Goals:

    - **Peer-to-peer** model supported on a *name service*
    - Focused **tools-based** approach, e.g. get/set configuration parameters, visualize the peer-to-peer connection topology, measure bandwidth utilization
    - **Language neutral** with specification at the messaging layer and peer-to-peer connection, negotiation and configuration in XML-RPC

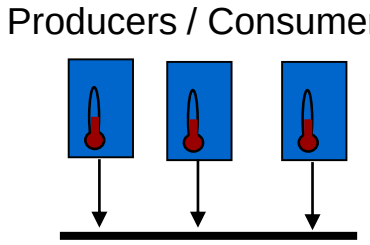
Middleware technologies

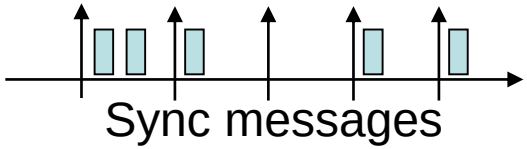
- **WorldFIP**
  - MPS – Messagerie Periodique e Sporadique
  - *Producer-Distributor-Consumer* middleware
  - Concept of **Network Variable**
    - Distributed entity
      (several local copies coexist in different nodes)
    - Can be periodic or aperiodic
    - Local copies of periodic variables
      are automatically refreshed by the network
    - Local copies of aperiodic variables
      are refreshed by the network upon explicit request



Middleware technologies

- **CANopen – a middleware for CAN**
  - **Process data** shared with the **producer/consumer** model
  - Standardized **device description** and **methods**
    - data, parameters, functions, programs
  - Standardized services for **device monitoring**
    - e.g. membership functions based on heartbeats
  - **System services**:
    - synchronization message,
    - central time-stamp message
      - e.g. synchronous data acquisition
  - **Emergency messages**
  - Adopted by other protocols (e.g. Ethernet POWERLINK, EtherCAT)



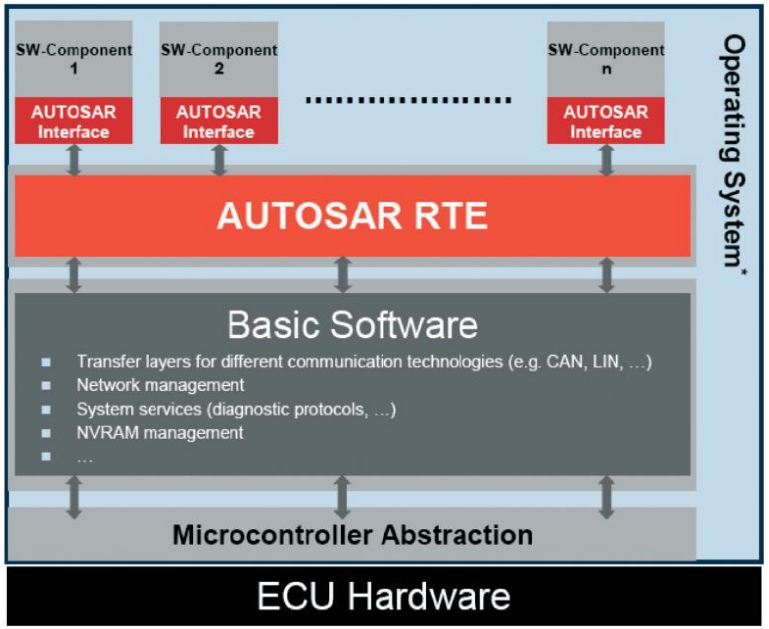


Middleware technologies

- **CANopen**
  - Process Data Objects (PDO)
    - Carry actual **application data**;
      - broadcast, **producer/consumer** cooperation model, unacknowledged
    - Asynchronous PDOs (event-triggered)
    - Synchronous PDOs (time-triggered based on Sync Object)
  - Service data objects (SDO) - device configuration
    - Read/write device Object Dictionary (OD) entries
    - Synchronous client/server model
  - Network management (NMT)
    - Node monitoring
    - Control their communication state

Middleware technologies

- **AUTOSAR**
  - Proposed by a consortium of automotive industries
  - Aims at separating **functionality** from **execution HW**
    - Soft AUTOSAR components
  - Improve efficiency in using system resources
    - Reduce **number of active components** and costs
    - Manage complexity
  - Give **more design freedom to the OEM** wrt subsystem providers
  - **Similar trends** in
    - avionics (IMA)
    - industrial automation (IEC 61499)



Middleware technologies - Wrap up

- **Different middleware models/paradigms imply**
  - Different communication **load**
    - CS, Blackboard: unicast request-response
    - PS, RTDB: multicast, one way
  - Different communication **pattern**
    - CS, PS – normally events
    - PC, PS, shared memory – normally periodic
  - Different level of **openness**
    - CS, Blackboard – directed unicasts
    - PS, RTDB – anonymous multicast
    - PC – broadcast

DDS – Data Distribution Service (OMG)

Adapted from a presentation made by

Isidro Calvo

Universidad del Pais Vasco, Spain

DDS

Introduction

- **Data Distribution Service for Real-time Systems**
  - Middleware based on the **Publisher / Subscriber** cooperation model
  - Provides one-to-many (anonymous), decoupled and asynchronous communication channels between data producers (Publishers) and their consumers (Subscribers)
  - Uses a **data-centric** programming model
- **High efficiency:**
  - Adequate to distribute large data volumes with low latency
  - Targets real-time distributed applications
- **Specification managed by the "Object Management Group" (OMG)**
  - Such as CORBA, UML, XML, …
  - Last open specification at the time of this talk: **DDS v1.2**
- **Used in:**
  - Time-critical applications (e.g., air traffic control, SCADA systems, telemetry, etc.)

DDS

Comparing with other technologies

- *wrt* **JMS (*Java Messaging Service*)**
  - JMS standardizes the API, only
- *wrt* **AMQP (*Advanced Messaging Queueing Protocol*)**
  - AMQP standardizes the network protocol, only

- **DDS**
  - Standardizes both, API and network protocol
  - Uses less layers than other middlewares: higher efficiency
  - Supports many configuration parameters, particularly for QoS control
  - Designed to be scalable, efficient and predictable
  - Assures interoperability with products of different makers

DDS

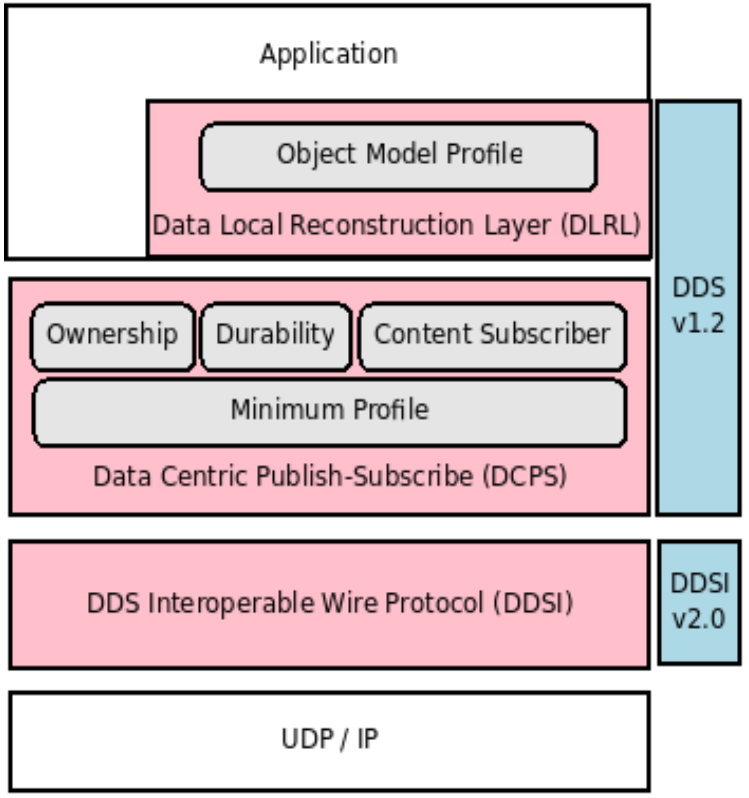
Other features

- DDS is **totally distributed**
  - Does not need a broker to connect data publishers with data subscribers
    - depends on implementations, some still use a broker (openDDS), other (ConnextDDS) allow topics to be managed from and reside in different nodes, eg, the Publisher itself
- DDS associates **user defined data types** to ***Topics***
  - Flexible data semantics
  - Allows single or continued (recurrent) access to the respective data
- DDS implementations provide **low access latency** with **low jitter**

DDS

The DDS standard by OMG

- The DDS standard includes the **API DDS v1.2** and the **interoperability protocol DDSI v2.1**
  - The **DDS API** guarantees the source **code portability** between implementations of different makers
  - **DDSI (RTPS)** assures **interoperability** in the connection between the implementations of different makers



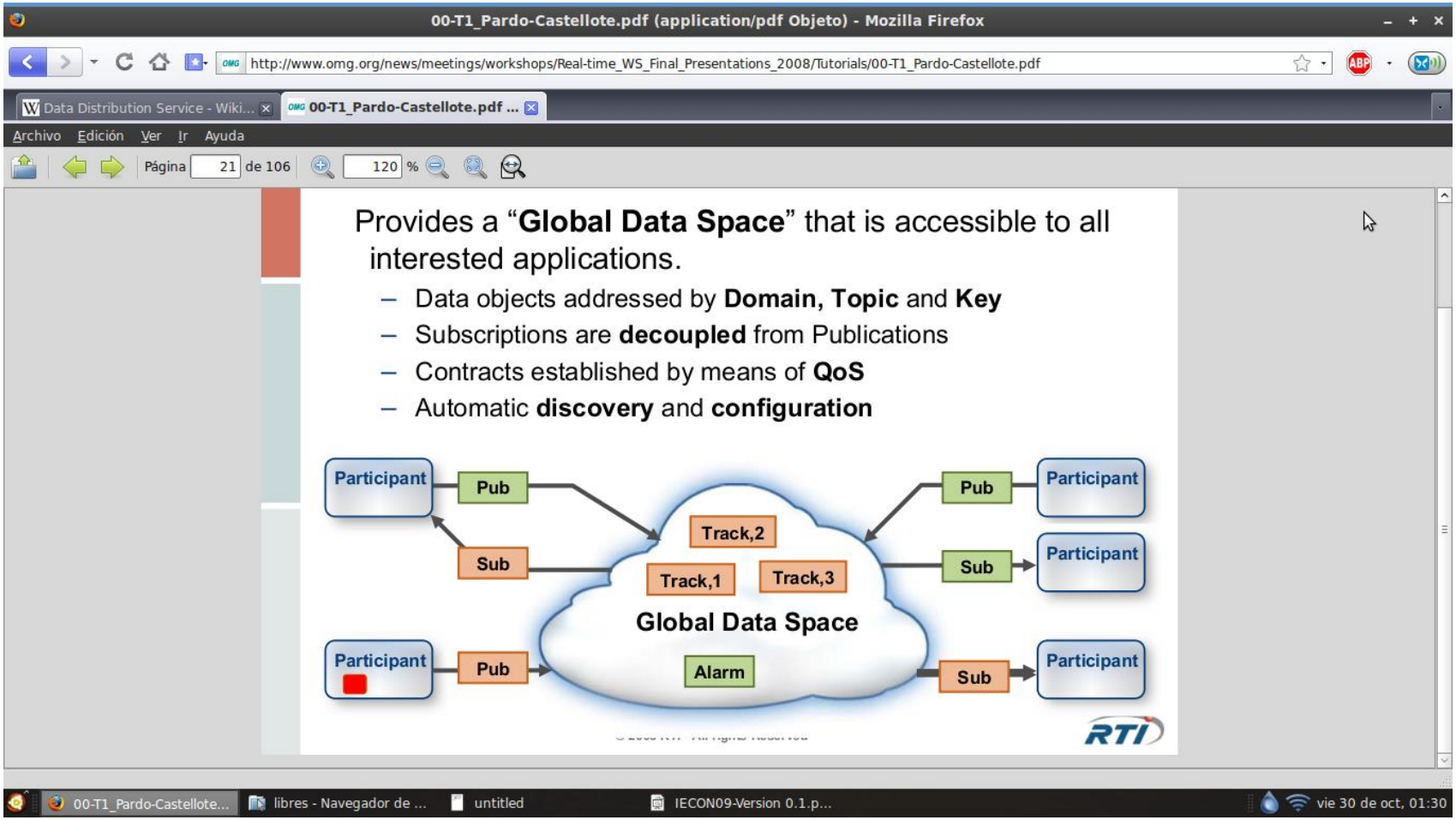
DDS

Publisher/Subscriber model

- DDS uses the concept of **Global Data Space (GDS)**
  - A distributed space (of Topics) accessible by all applications
- The specification requires the GDS to be **fully distributed**
  - Avoids a single point of failure or bottlenecks
- Publishers and subscribers can **connect to** or **disconnect from** the GDS at **any instant in time**
  - DDS uses dynamic discovery mechanisms
  - There is no centralized registry (e.g., as opposed to JMS)
- Distributed applications using DDS can **crash/reboot**, **connect/disconnect** at run-time
  - The distributed system will continue operating

DDS

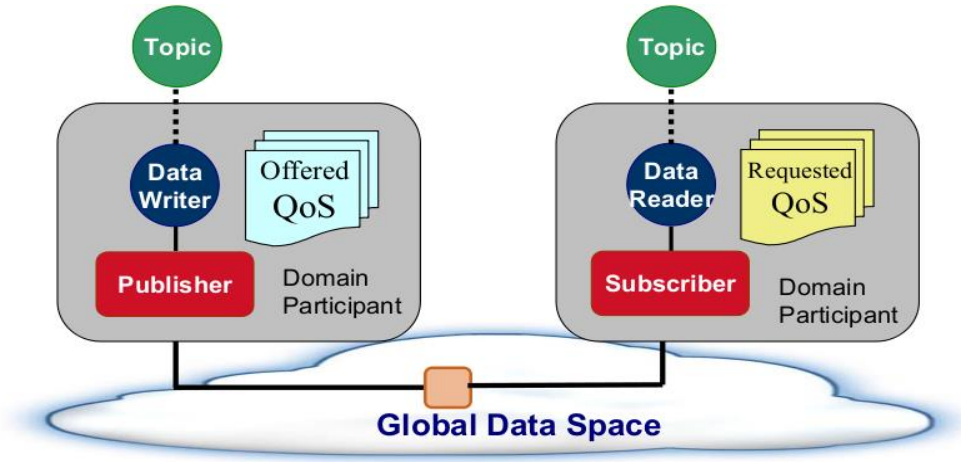
Global Data Space (GDS)



DDS

Object model

- **The main components involved in DDS are:**
  - **Domain:** A logical partition of the GDS. Only the connected nodes will have access to ("will see") the data
  - **Domain Participant:** Entry point for the communication in a specific domain
  - **Topic:** the simplest description of the data to be published and subscribed
  - **Publisher (Subscriber):** object responsible for the actual dissemination of publications (actual reception of data from subscriptions)
  - **DataWriter (DataReader):** Entitity that declares the intention to publish (subscribe to) a Topic and is provided with *type-safe* operations to send (receive) data



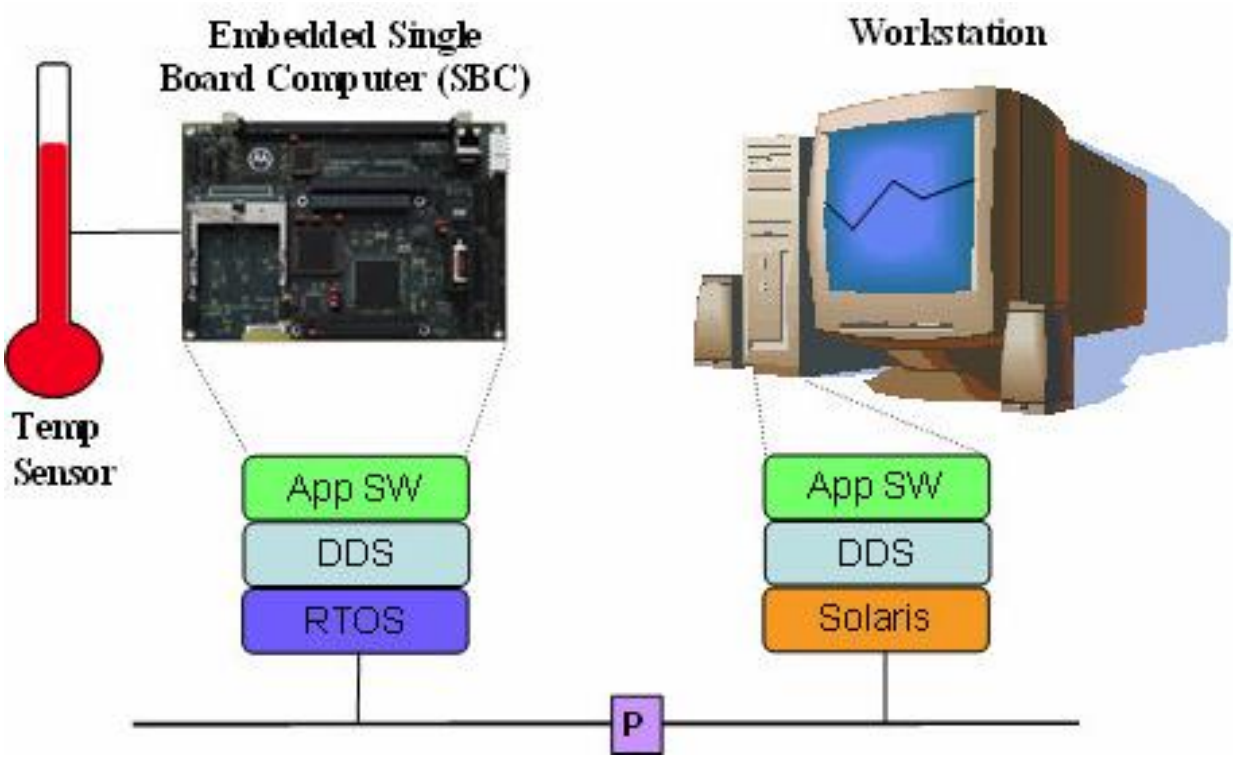
DDS

Topics

- A ***Topic*** is
  - **unit of information** that can be published or subscribed (shared)
- It includes these **fields**:
  - **Type**
  - Unique **name**
  - **QoS policies** to be applied
    - if unspecified, the default QoS policies will be used
- *Topics* are **represented** with
  - a subset of the OMG **description language IDL** (Interfac Descript Lang)

DDS

Application example



DDS

IDL of the previous example

```
enum TempScale {
   CELSIUM,
   KELVIN,
   FARENHEIT };
struct TempSensorType {
  short id;
  float temp;
  float hum;
  TempScale scale;
};
#pragma keylist TempSensorType id
```

DDS

QoS Parameters

- **Deadline** (T, DR, DW)
- **Destination Order** (T, DR)
- **Durability** (T, DR, DW)
- **Entity Factory** (DP, Pub, Sub)
- **Group Data** (Pub, Sub)
- **History** (T, DW, DR)
- **Latency Budget** (T, DR, DW)
- **Lifespan** (T, DW)
- **Liveliness** (T, DW, DR)
- **Ownership** (T)
- **Ownership Strength** (DW)
- **Partition** (Pub, Sub)
- **Presentation** (Pub, Sub)
- **Reader Data Lifecycle** (DR)
- **Reliability** (T, DW, DR)
- **Resource Limits** (T, DW, DR)
- **Time-Based Filter** (DR)
- **Topic Data** (T)
- **Transport Priority** (T, DW)
- **User Data** (T, DP, DR, DW)
- **Writer Data Lifecycle** (DW)

DDS

An on-line **tutorial** can be found here:

http://www.slideshare.net/Angelo.Corsaro/the-dds-tutorial-part-i