

Webinar begins at 2:05PM, London Time

Dr. Angelo Corsaro [angelo.corsaro@prismtech.com]

OpenSplice DDS Product Marketing Manager, PrismTech

Angelo co-chairs the OMG Data Distribution Service (DDS) Special Interest Group and the Real-Time Embedded and Specialized Services (RTESS) Task Force. He is a well known figure in the distributed real-time and embedded systems middleware community and has a wealth of experience in hard real-time embedded systems, large-scale and very large-scale distributed systems, such as defense, aerospace, homeland security and transportation systems. Prior to joining PrismTech, he worked for the SELEX-SI CTO Directorate, a FINMECCANICA company, where his responsibilities included mapping business requirements to technology capabilities, strategic standardization and technology innovation.



Hans van't Hag [hans.vanthag@prismtech.com]

OpenSplice DDS Product Manager, PrismTech

Hans has extensive experience in applying an information approach towards mission-critical and real-time net-centric systems. He is a co-author of the OMG DDS specification and has presented numerous papers on DDS and publish subscribe middleware technologies. Prior to joining PrismTech he worked for 23 years at Thales Naval Netherlands (TNN) where he was responsible as Product Manager for the development of the data-centric real-time middleware (SPLICE) as applied in TNN's TACTICOS combat system in service with 15 Navies worldwide.

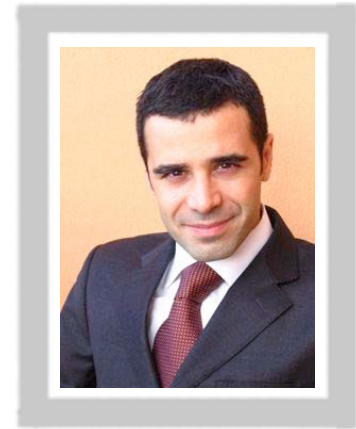




Taming the Data Centric Design with OpenSplice DDS and OpenSplice Powertools

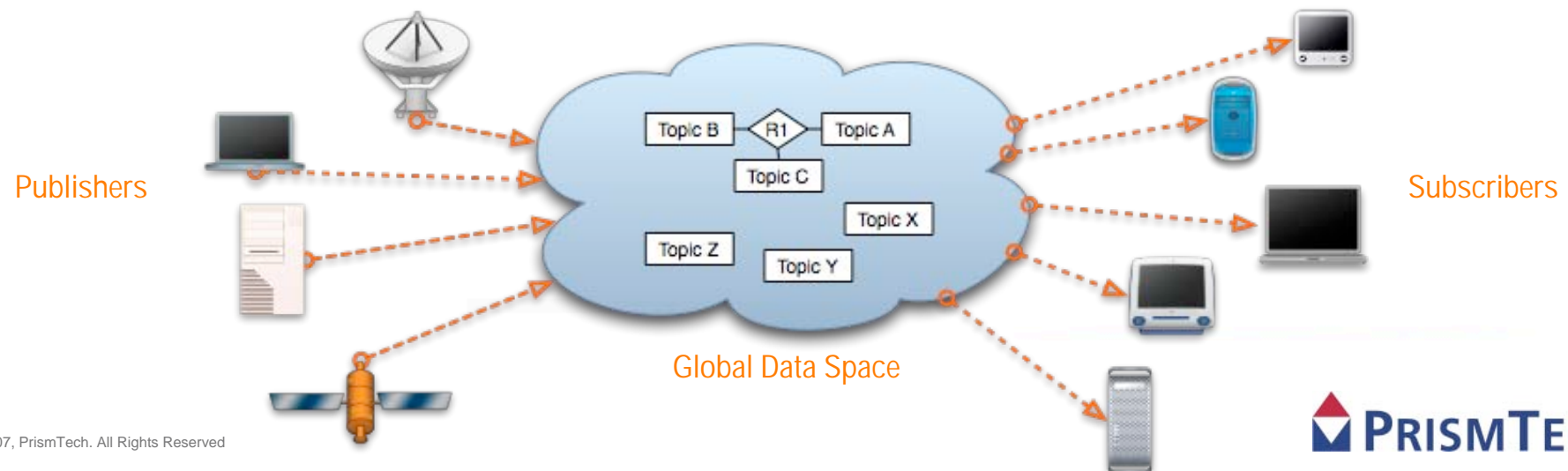


- ▶ **OpenSplice DDS Overview**
- ▶ **Information modeling**
- ▶ **DDS by Example**
- ▶ **Modeling the Example**
- ▶ **OpenSplice Powertools™**
- ▶ **Whats next**



Dr. Angelo Corsaro

- ▶ **An High Performance Real-Time Data-Centric Publish/Subscribe Middleware**
 - ▶ *The right data, at the right place, at the right time -- all the time!*
 - ▶ *Fully distributed, high performance, highly scalable, and high availability architecture*
- ▶ **Unparalleled support for Data-Centric & real-time Publish/Subscribe features**
 - ▶ *Content based subscriptions, queries and filters, DLRL*
 - ▶ *Fine grained tuning of resource usage and data delivery and availability QoS*
 - ▶ *Optimal networking and computing resources usage*
- ▶ **Loosely coupled**
 - ▶ *Plug and Play Architecture with Dynamic Discovery*
 - ▶ *Time and Space Decoupling*
- ▶ **Open Standard,**
 - ▶ *Complies with the full profile of the OMG DDS v1.2*



OpenSplice DDS is compliant with the full profile specified in the OMG DDS Specification v1.2



Object Model Profile

**Data Local Reconstruction Layer
(DLRL)**

Ownership

Persistence

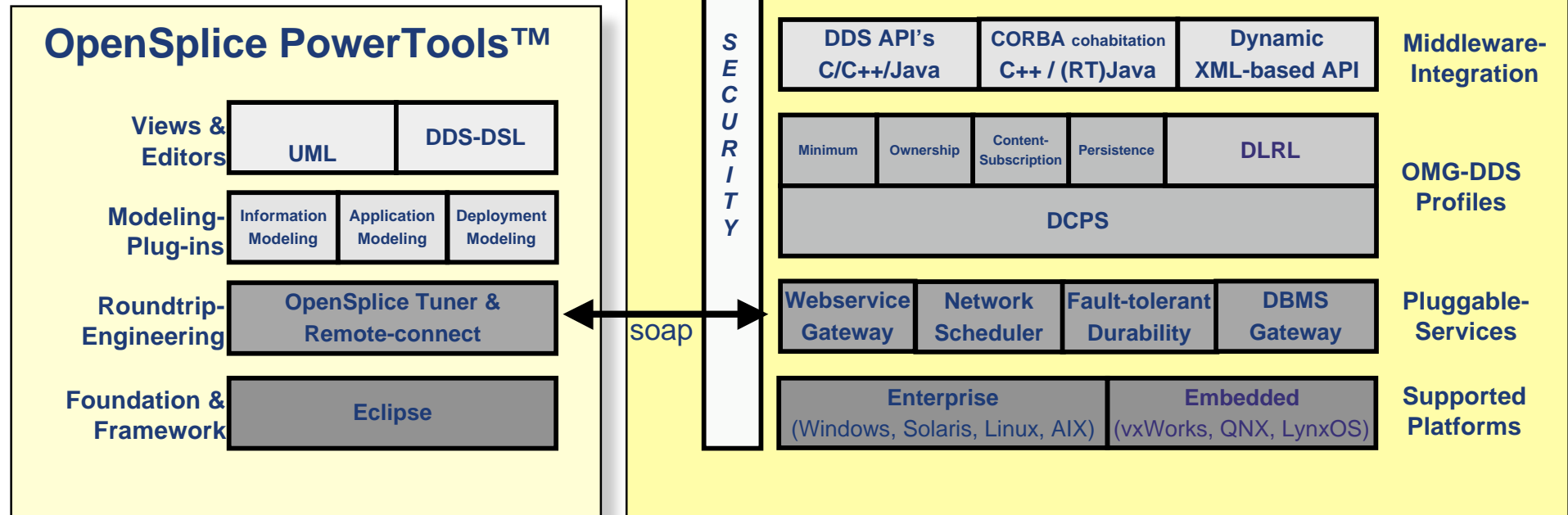
Content-Subscription

Minimum Profile

**Data Centric Publish Subscribe
(DCPS)**

Deployment

Development



OpenSplice™ DDS

OpenSplice™ DDS

Functionalities

- ▶ Full OMG-DDS specification coverage *(DCPS and DLRL)*
- ▶ Provision of a true 'fault-tolerant information backbone' *(content-aware and FT-durability)*
- ▶ Wide Cohabitation and Connectivity with other Technologies *(Corba, RT-Java, DBMS, SOAP, XML)*
- ▶ Availability of (remote) deployment tools *(Tuner™ offering total & remote control)*
- ▶ Support for Information/application/deployment modeling *(DCPS/DLRL-specific roundtrip development)*

Performance

- ▶ **Scalability** w.r.t. number of applications as well as computing nodes and topics
- ▶ **Real-time determinism** by urgency (latency-budget) & importance (priority) based network-scheduling
- ▶ **Fault-tolerance** by FT-durability and reliable network-service shielding faulty applications from the network

Pedigree

- ▶ **Maturity.** Product proven, fielded, In service in 15 Navies world-wide
- ▶ **Fractal Architecture.** Large-scale, real-time, fault-tolerant, embedded, all in one system!
- ▶ **High Standard of Quality Assurance.** Process/procedures, QA-artefacts and regression testing w.r.t. number of applications as well as computing nodes and topics

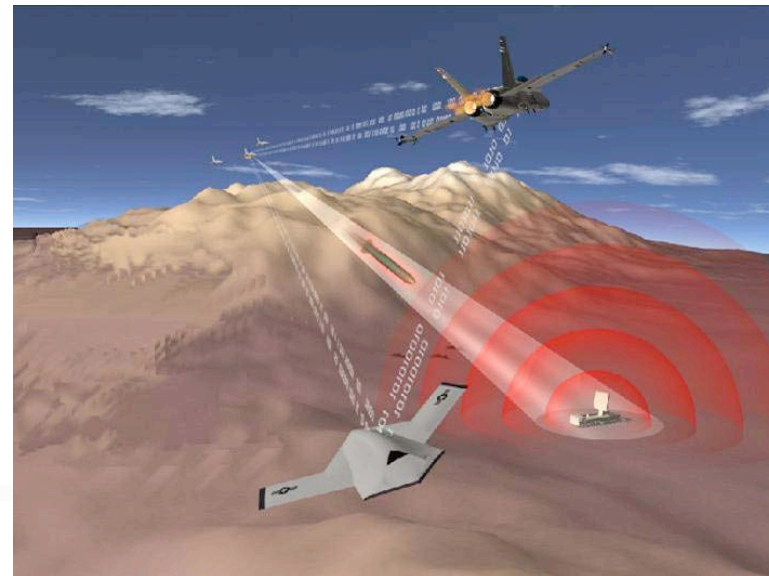
- ▶ OpenSplice DDS Overview
- ▶ Information modeling
- ▶ DDS by Example
- ▶ Modeling the Example
- ▶ OpenSplice Powertools™
- ▶ Whats next



Hans van't Hag

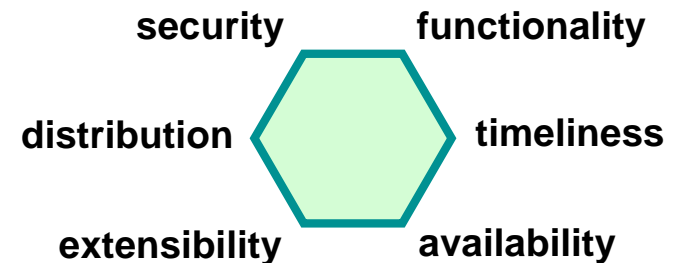
More Complex Systems and Requirements

11



Problem: engineering (-cost) of distributed systems

- ▶ too complex
- ▶ not reactive
- ▶ not future-proof
- ▶ not fault tolerant



Because 'multi-dimensional engineering' is needed:

What about the current 'state-of-the-art'?

- ▶ architectures: client/server, message-passing, SOA
- ▶ most efforts fall short in a number of dimensions:
- ▶ typically:
 - ▶ limited RT performance (high-volume & low-latency balance)
 - ▶ exploding complexity (dependencies in many dimensions)
 - ▶ costly evolution (impact of changes & extensions)

System design

- ▶ *provide a stable basis to operate upon by applications*
- ▶ *enhance component autonomy*
- ▶ *allow transparent and global QoS assurance*

System development

- ▶ *reduce complexity and enhance re-usability*
- ▶ *provide shared/guaranteed properties*
- ▶ *small learning effort and flat learning curve*

System integration

- ▶ *support effortless component integration*
- ▶ *provide easy monitor & control*
- ▶ *shift ratio between design and integration effort*

System deployment

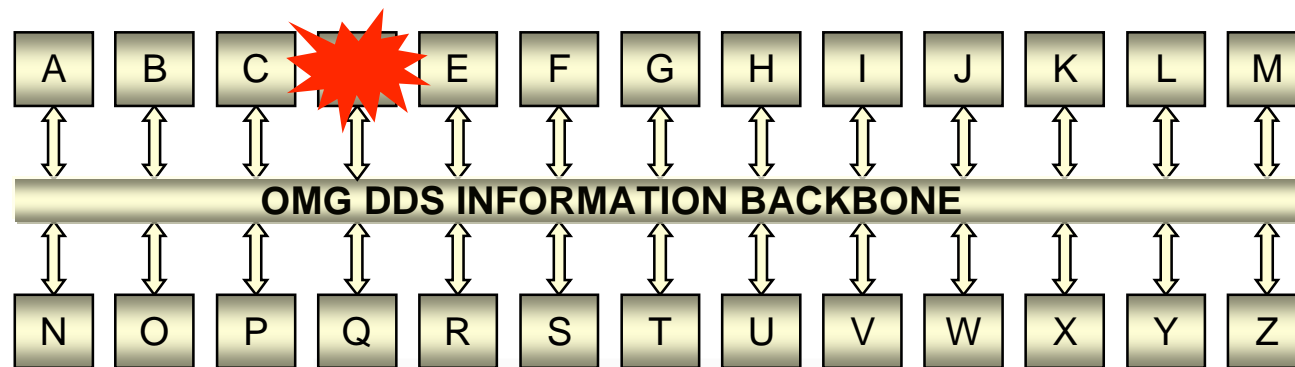
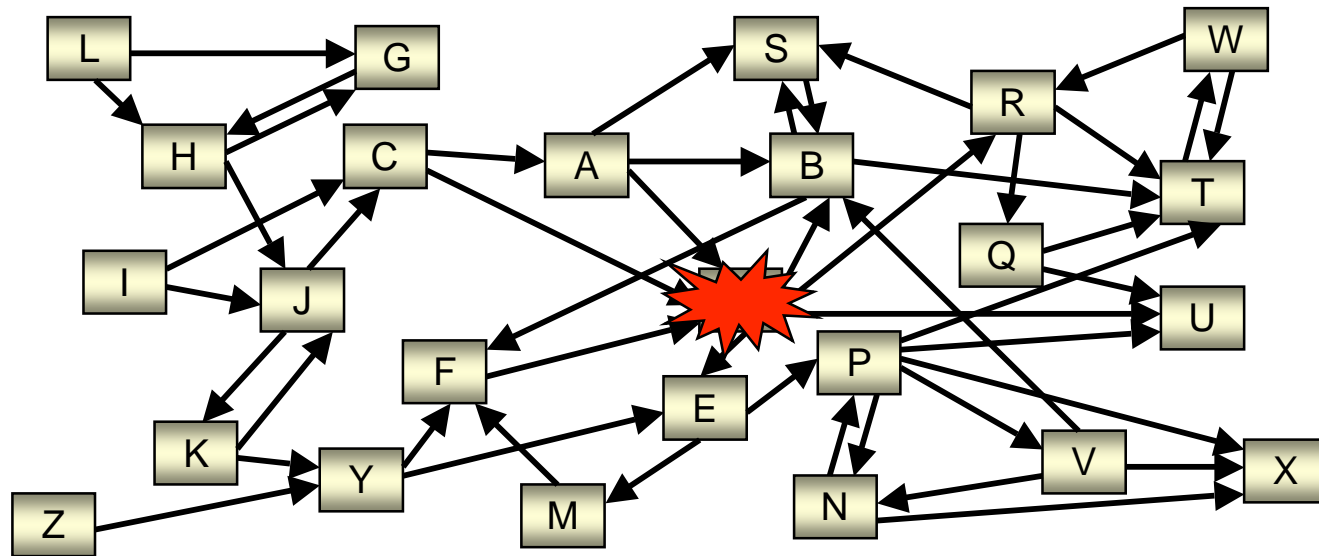
- ▶ *guaranty QoS for reliability, latency and persistency*
- ▶ *allow applications to join the system at any time*
- ▶ *allow runtime migration/restart of applications*

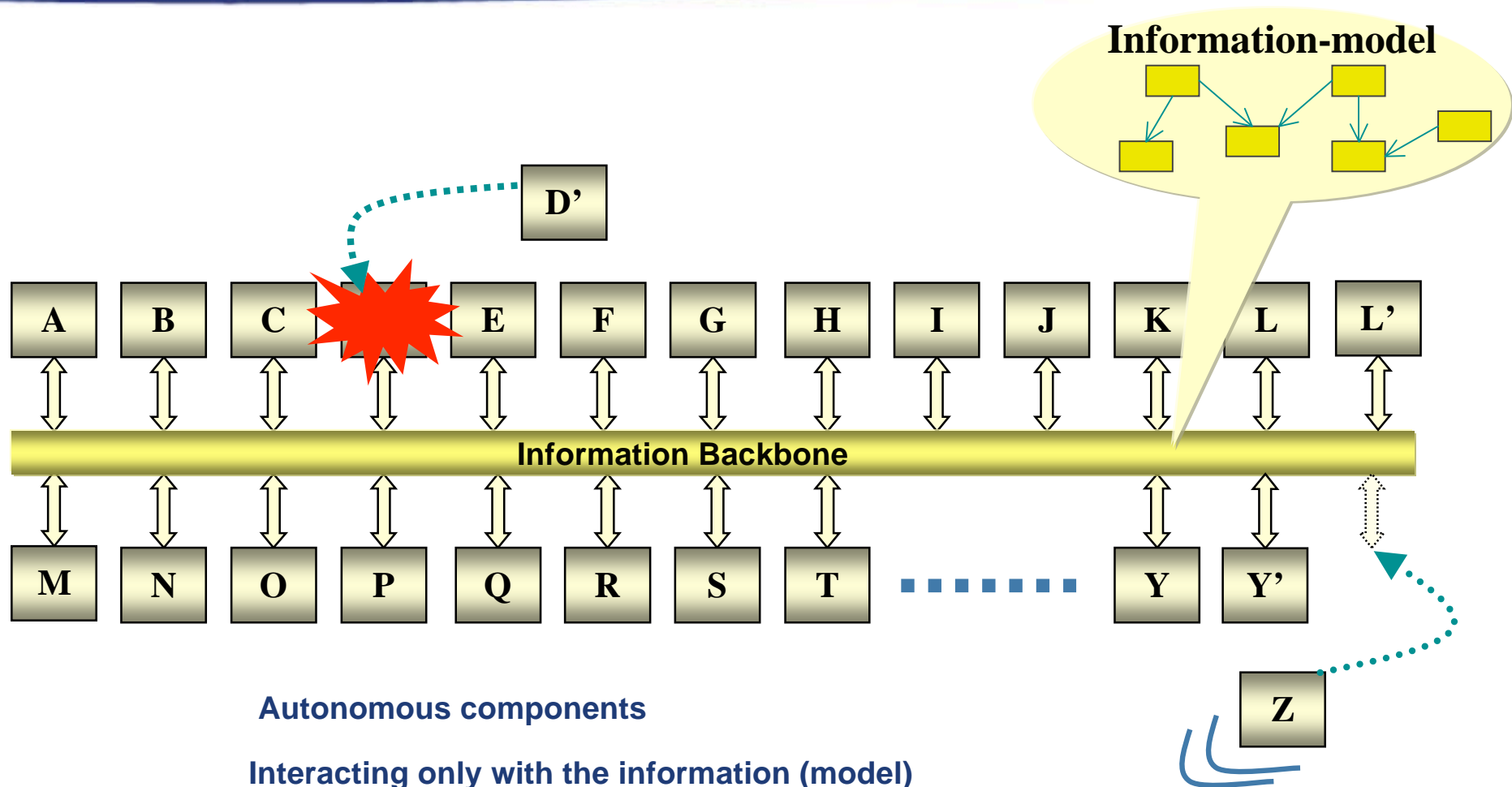
System maintenance & evolution

- ▶ *allow runtime replacement and evolutionary upgrading*
- ▶ *support for logging & replay of information*
- ▶ *provide future-proof, re-usable, robust and scalable system*

Data centric systems: A mind shift

14





Autonomous components

Interacting only with the information (model)

Spontaneous: **Z**, **Self-healing:** **D'**

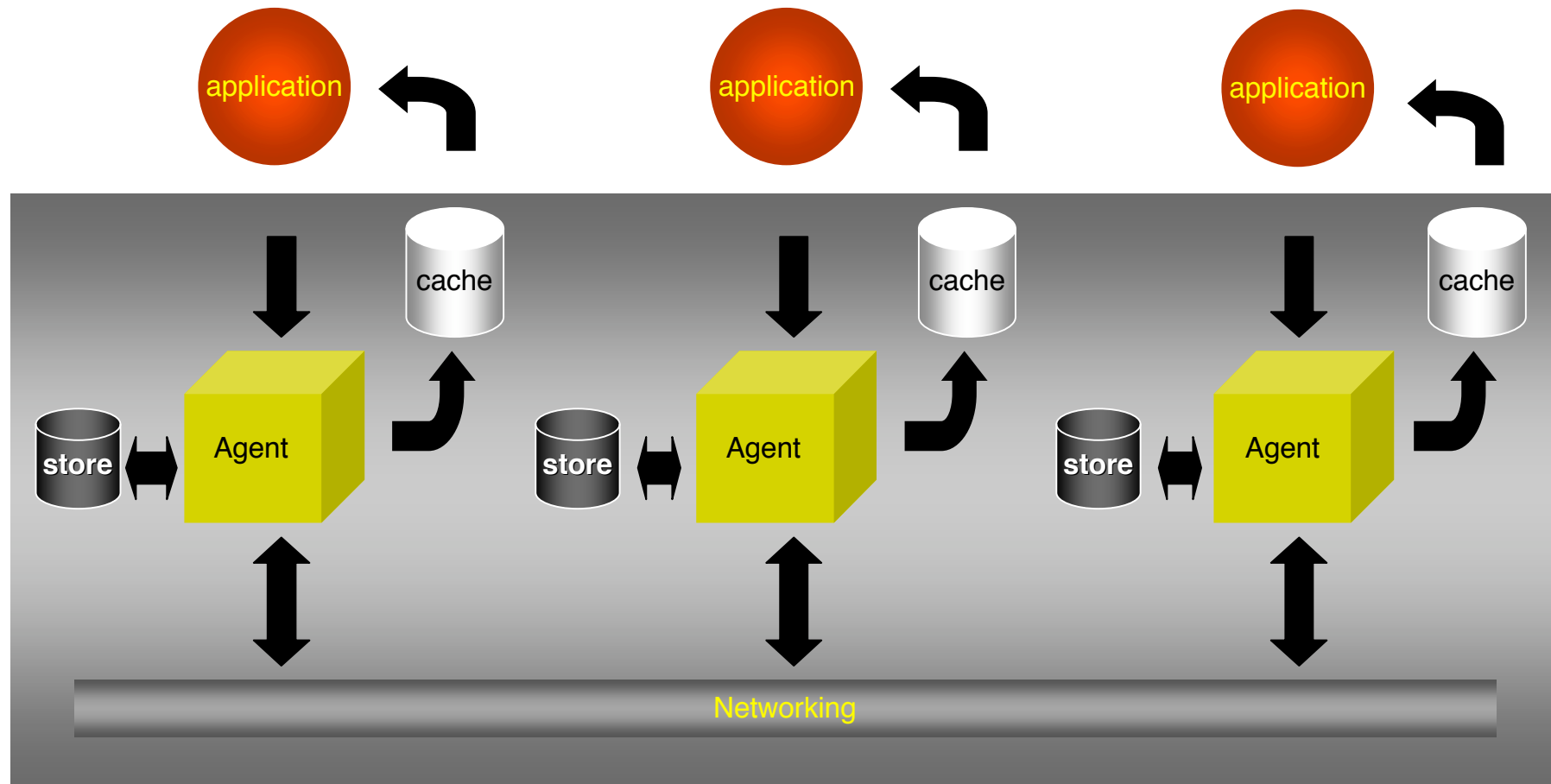
Redundant & Replicated: **L'**, **Y'**

QOS-driven Data Distribution Service (urgency, importance, durability):

DDS

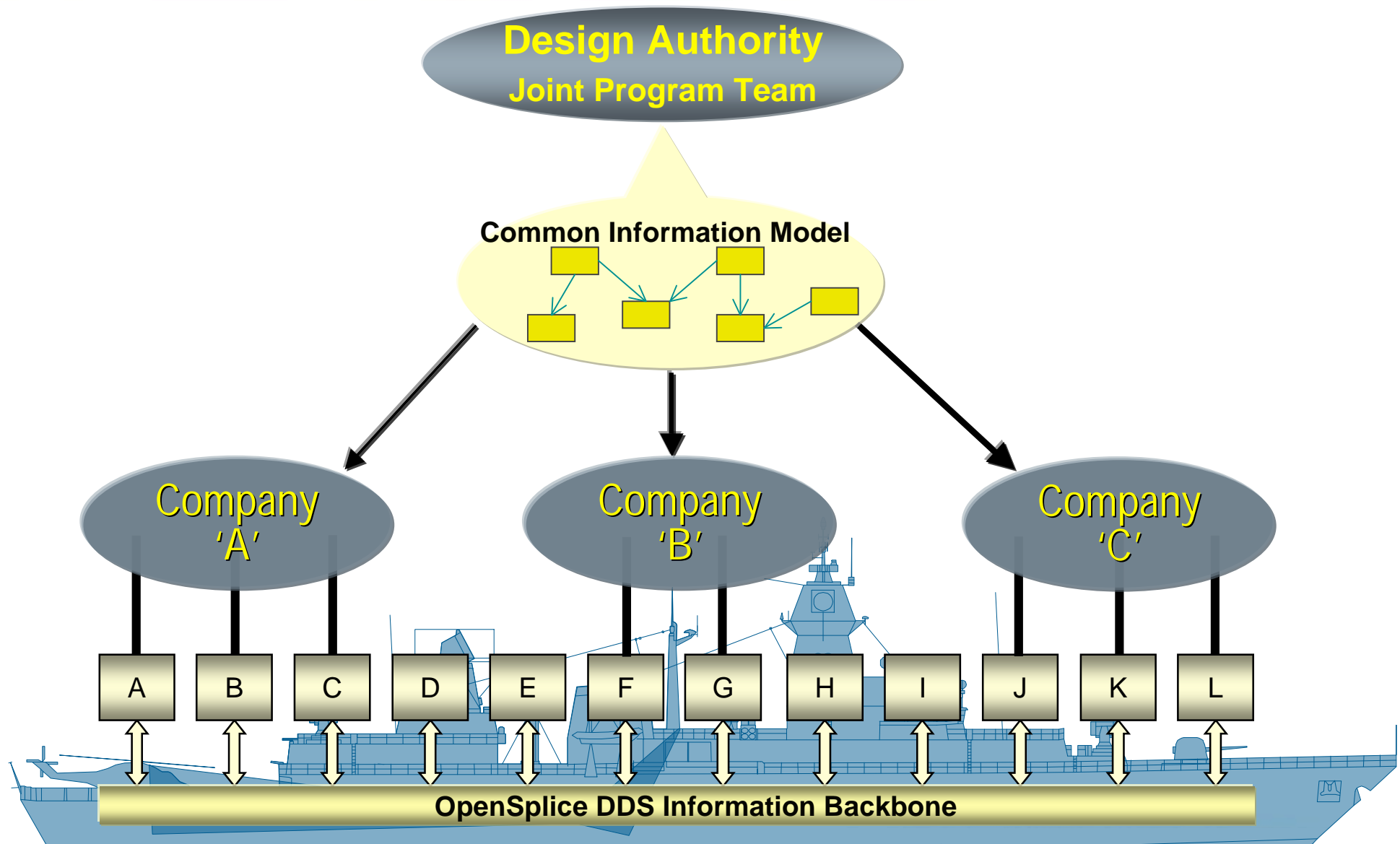
Self-Healing: Fault-tolerant persistence

16



EXAMPLE: Coalition-based development (German F124 frigate)

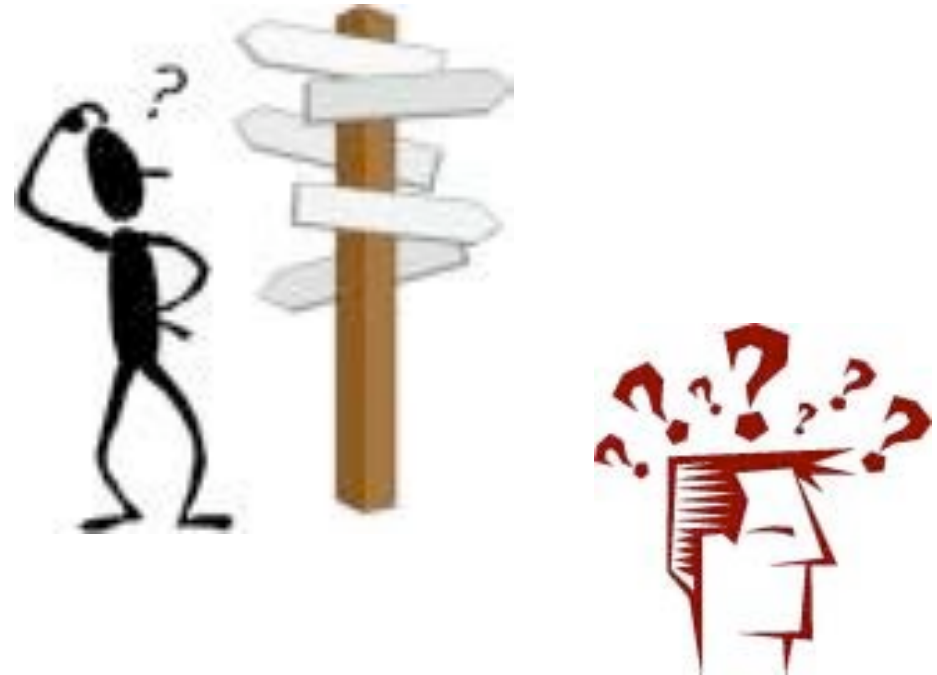
17



Complex Publish Subscribe Systems

18

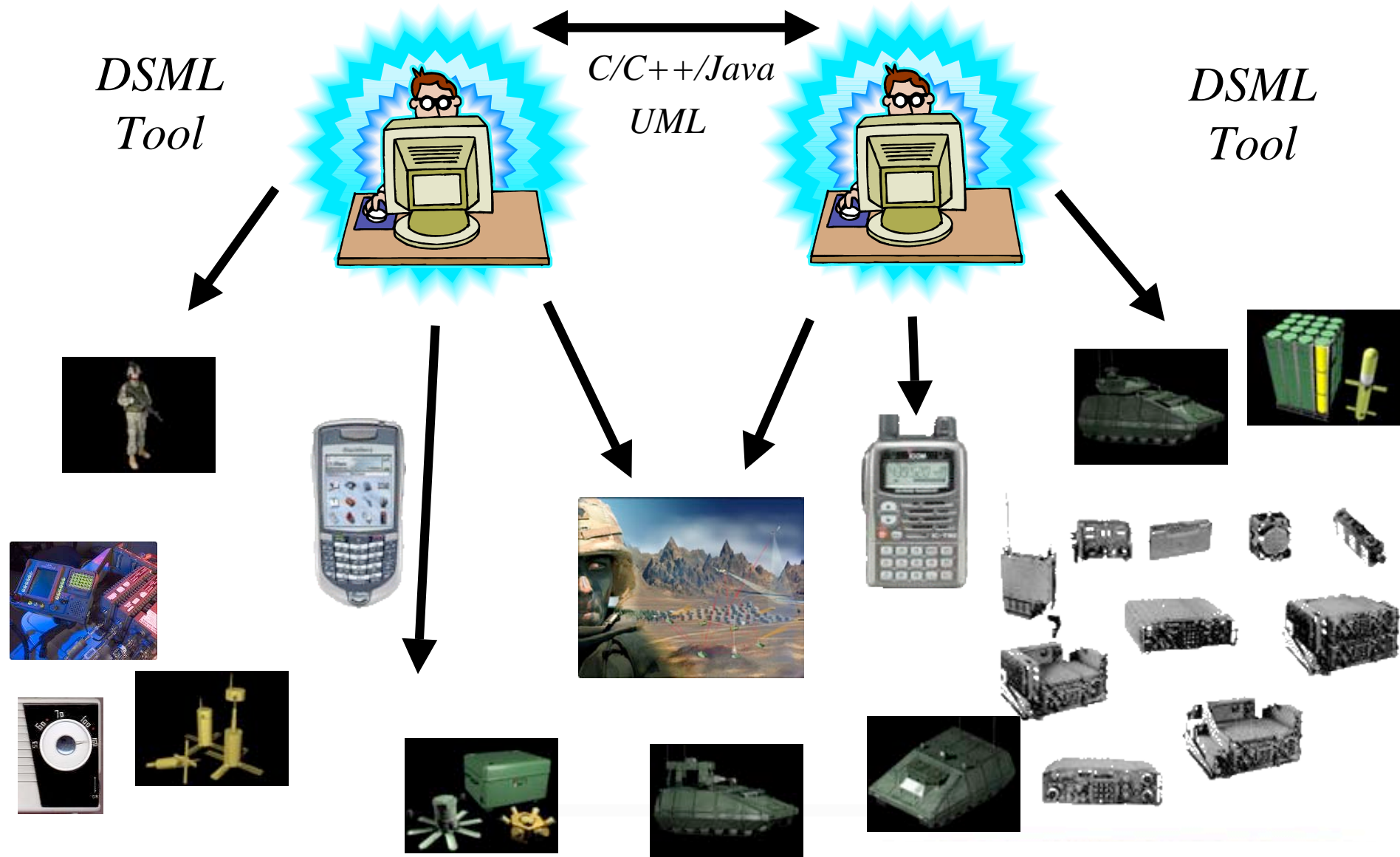
Object Oriented
Component Based
Multithreaded/MultiProcess
Real-time
Embedded
C++/Java/Ada/VHDL
Platform Independent
High Performance
Heterogeneous
Distributed
Vital
Secure
Fault Tolerant
Portable
Standardized
Declarative
Imperative
Dynamic



So what's the big deal?
Each one *by itself* is difficult,
let alone doing them all *at the*
same time

Providing sufficient tools to do the job

19



Domain Specificity

20



*Domain
Independent*



*Domain Specific -
Tools*

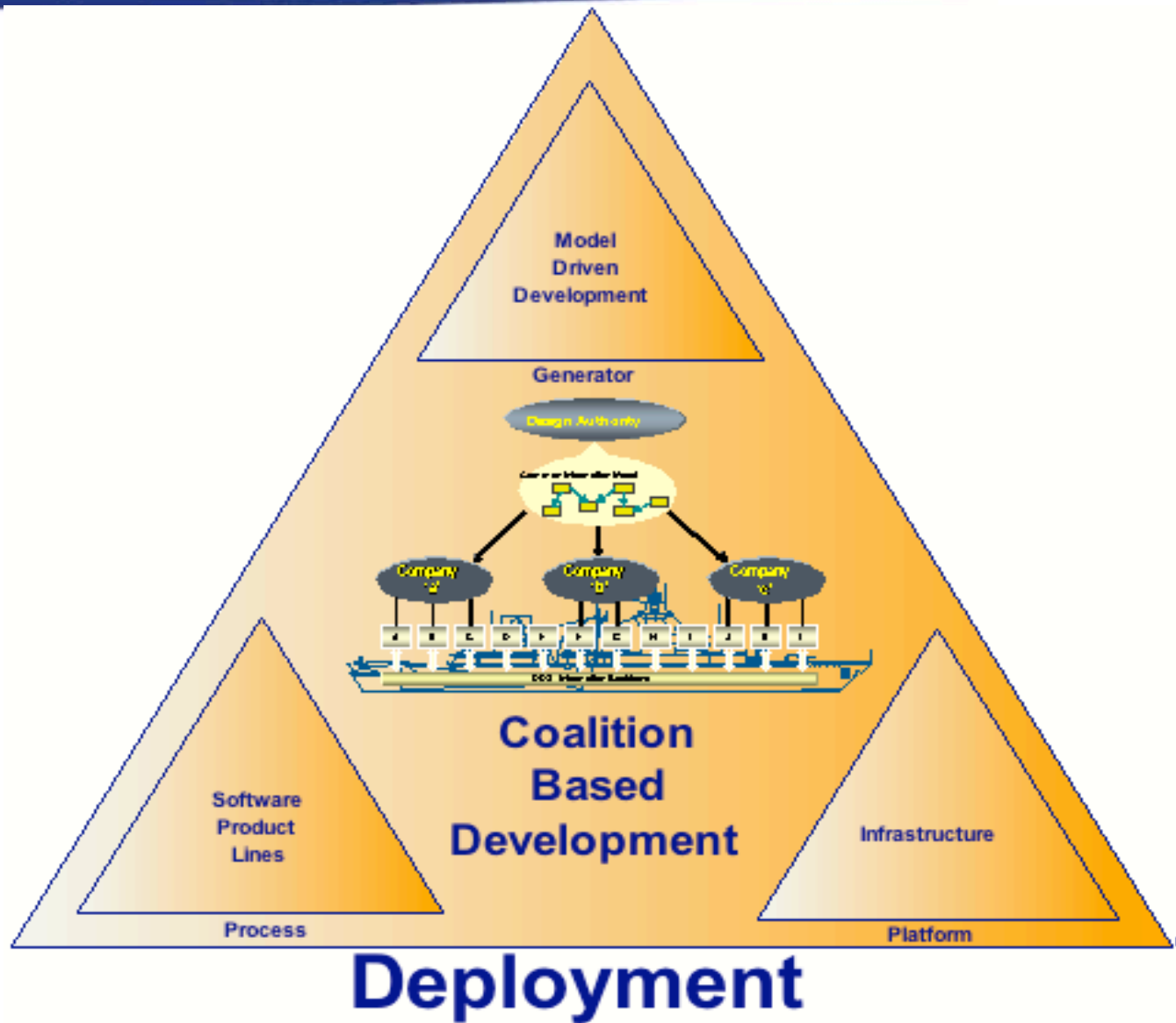


*The task
at hand*



The “Holy Grail” ??.....

21



- ▶ OpenSplice DDS Overview
- ▶ Information modeling
- ▶ **DDS by Example**
- ▶ Modeling the Example
- ▶ OpenSplice Powertools™
- ▶ Whats next

Introducing The Example

23



SENSOR PROCESS

- ▶ Optical sensor
- ▶ Scans the environment
- ▶ Produces 'Tracks'
- ▶ Position of 'objects'
- ▶ Reports '**pointTrack**'



IDENTIFICATION PROCESS

- ▶ Classifies Tracks
- ▶ Determines their identity
- ▶ Analyses the trajectories
- ▶ Deduces hostile intent
- ▶ Reports '**trackState**'



DISPLAY PROCESS

- ▶ Displays track info
- ▶ Both Position & Identity
- ▶ Raises alerts
- ▶ Requires '**pointTrack**'
- ▶ Requires '**trackState**'

Importance of the information model

- ▶ In Data-centric systems, it's a key-asset for customer-interaction and system-design
- ▶ In a DDS-based system, it's actually THE (only) interface between an application and the rest of the system

Complexity reduction: Process Autonomy

- ▶ **Sensor Process:**
 - ▶ only knows about observed object-locations which it should publish at 'his' rates
 - ▶ Shouldn't bother about 'classifications' or any other subscriber to its information
- ▶ **Classification/Display Process:**
 - ▶ Also autonomous applications without dependencies

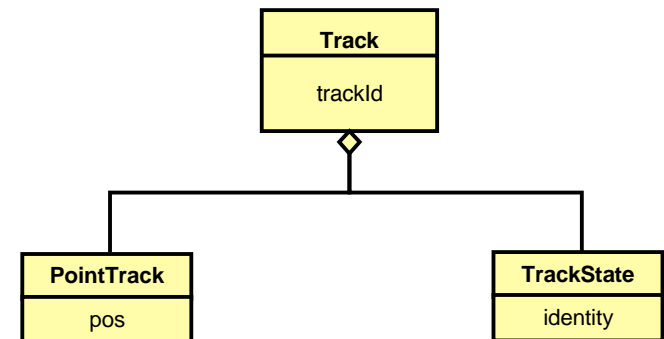
Assuring Non-Functional properties: Performance & Fault-tolerance

- ▶ **Performance**
 - ▶ Periodic measurements (pointTrack objects): *best-effort* delivery, *volatile* persistence
 - ▶ A-periodic state-data (trackState objects): *reliable* delivery, *transient* persistence
- ▶ **Fault tolerance**
 - ▶ **Sensor:** replicated sensors shouldn't increase system complexity
 - ▶ **Identification:** state should be preserved to allow quick re-start after any errors
 - ▶ **Display:** should be able to 'join' the system at any time

Information modeled as “TOPICS”

Each TOPIC has an associated *name* and *data type*

- ▶ Type in IDL
- ▶ ‘Key’ fields for unique identification
- ▶ Relational Data Model (keys)



Topics can be annotated with QoS policies

- ▶ Driving system-wide behavior w.r.t. delivery, durability, priority, urgency, etc.
- ▶ Topic-level QoS policies can be used as defaults for Readers/Writers

Topic “*PointTrack*”

```
Struct PointTrackType {
    long trackId;

    Position pos;
}
```

Topic “*TrackState*”

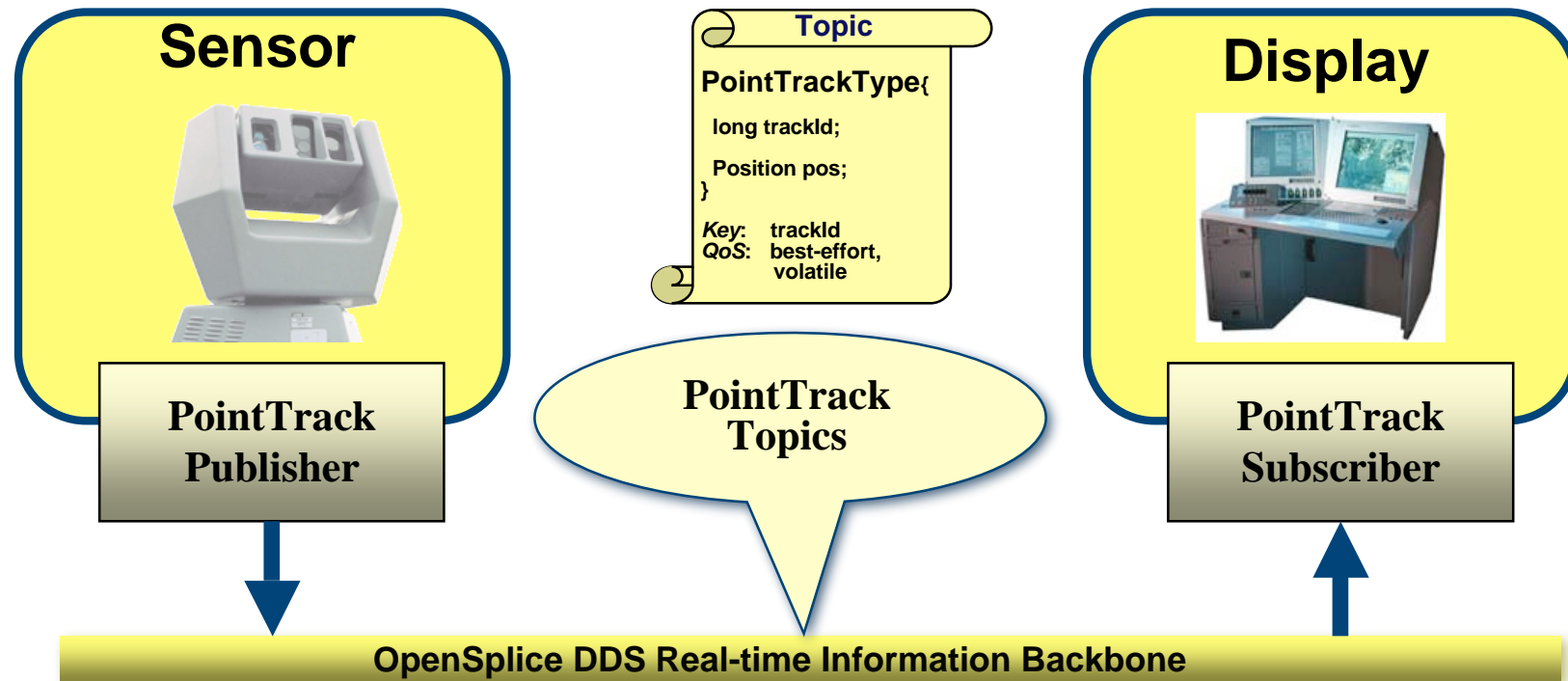
```
Struct TrackStateType {
    long trackId;

    Id identity;
}
```

Key fields

Utilizing the 'minimum profile' features

26



System Requirements:

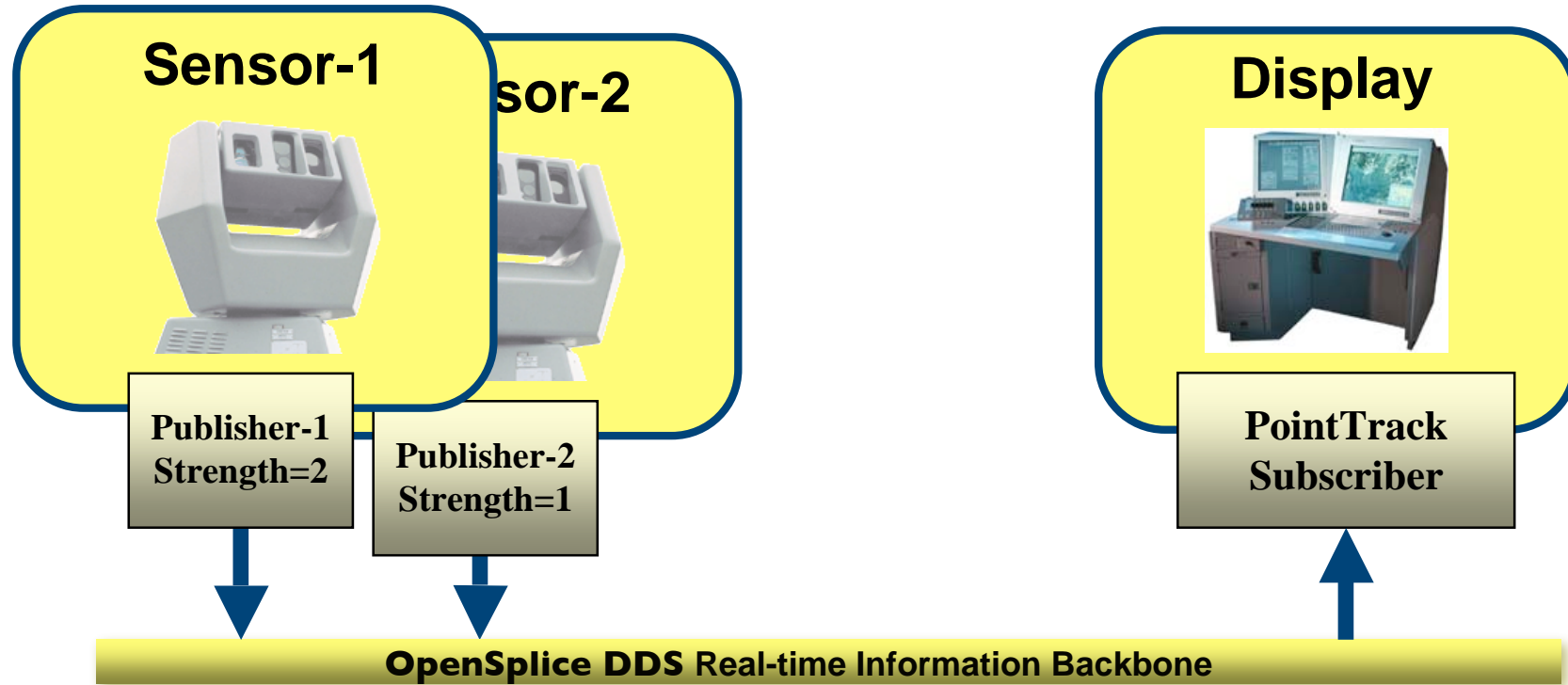
- ▶ Autonomous applications
 - ▶ decoupled in space & time
- ▶ Easy integrate-able
- ▶ Re-usable

Information Model impact (Sensor example)

- ▶ *PointTrack* Topic as 'only' interface of the sensor
 - ▶ Type: position of each identified 'Track'
 - ▶ Keys: TrackId as key-field
 - ▶ QoS: *Best-effort* delivery, *Volatile* persistence

Utilizing the 'ownership profile' features

27



System Requirements

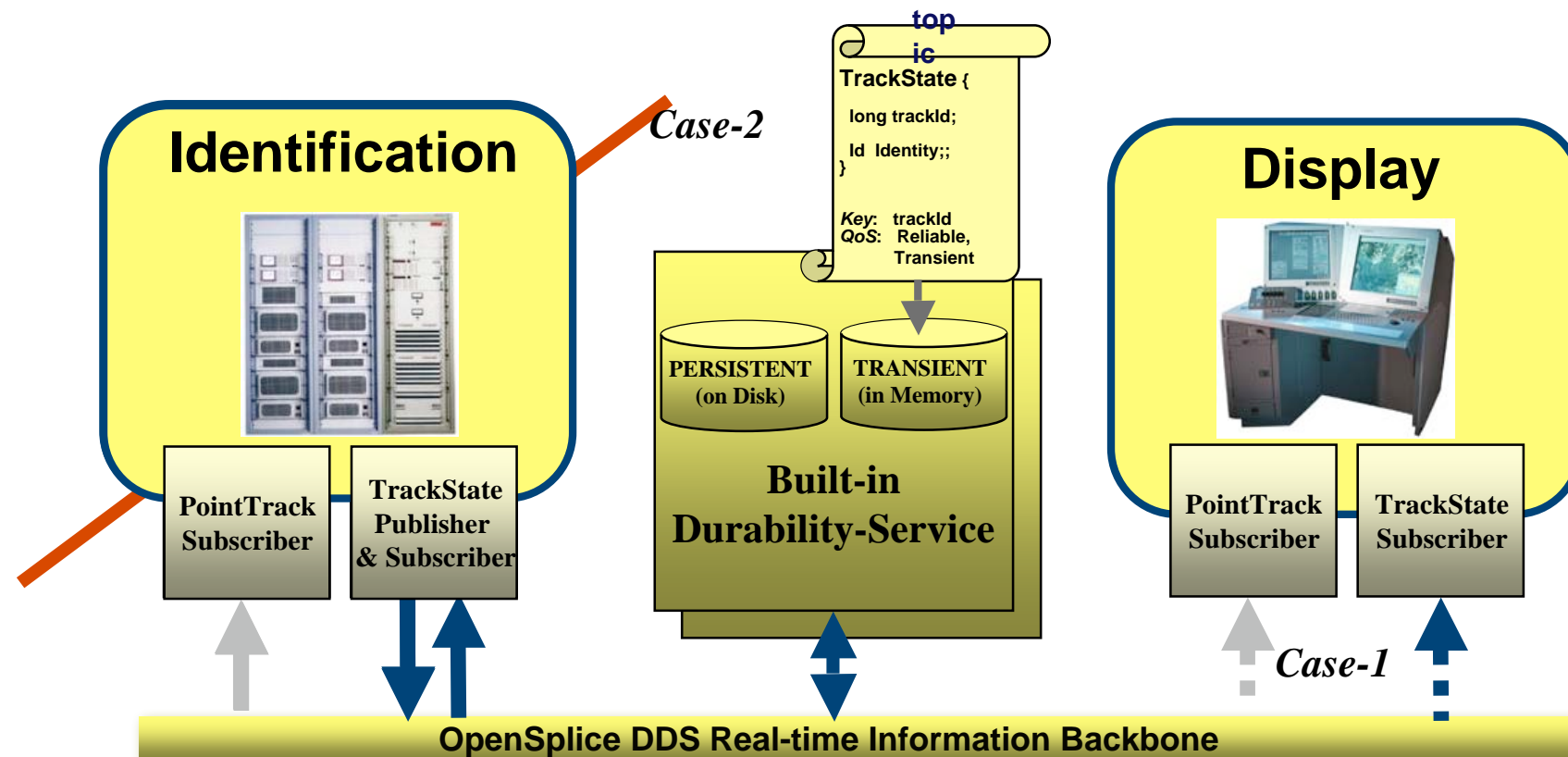
- ▶ Replicated sensors to increase fault-tolerance
- ▶ Shall not increase sensor or system complexity
- ▶ Tolerable to assign each replica a 'strength'

Modeling impact

- ▶ Transparent Fault-tolerance by replication
 - ▶ Only highest-strength data will be received
- ▶ Alternative solution = content-awareness
 - ▶ e.g. query for highest quality (data-attributes)

Utilizing the 'Persistence Profile' features

28



System Requirements

- ▶ **Case-1:** *late-joining* of Display process
 - ▶ Previously produced TrackStates must be available
- ▶ **Case-2:** *restart* of failed Classification process
 - ▶ Internal state (already classified tracks)

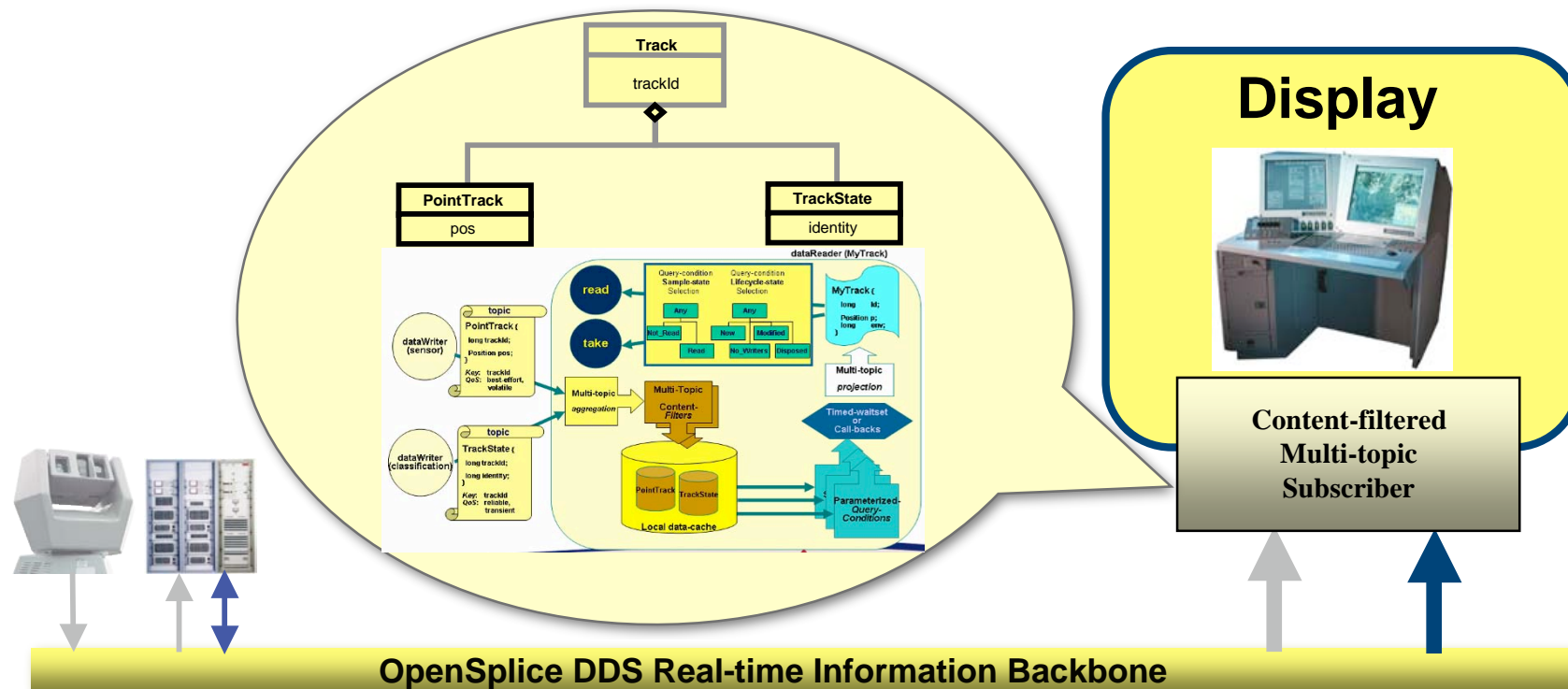
must be available upon restart

Information Modeling Impact

- ▶ Information *durability* QoS
 - ▶ Volatile: data only available at production time
 - ▶ Transient: FT-availability for late-joining app's
 - ▶ Persistent: data persisted on (replicated) disks
- ▶ Information *reliability* QoS: resends of missed data

Utilizing the 'content subscription profile' features

29



Display Requirements

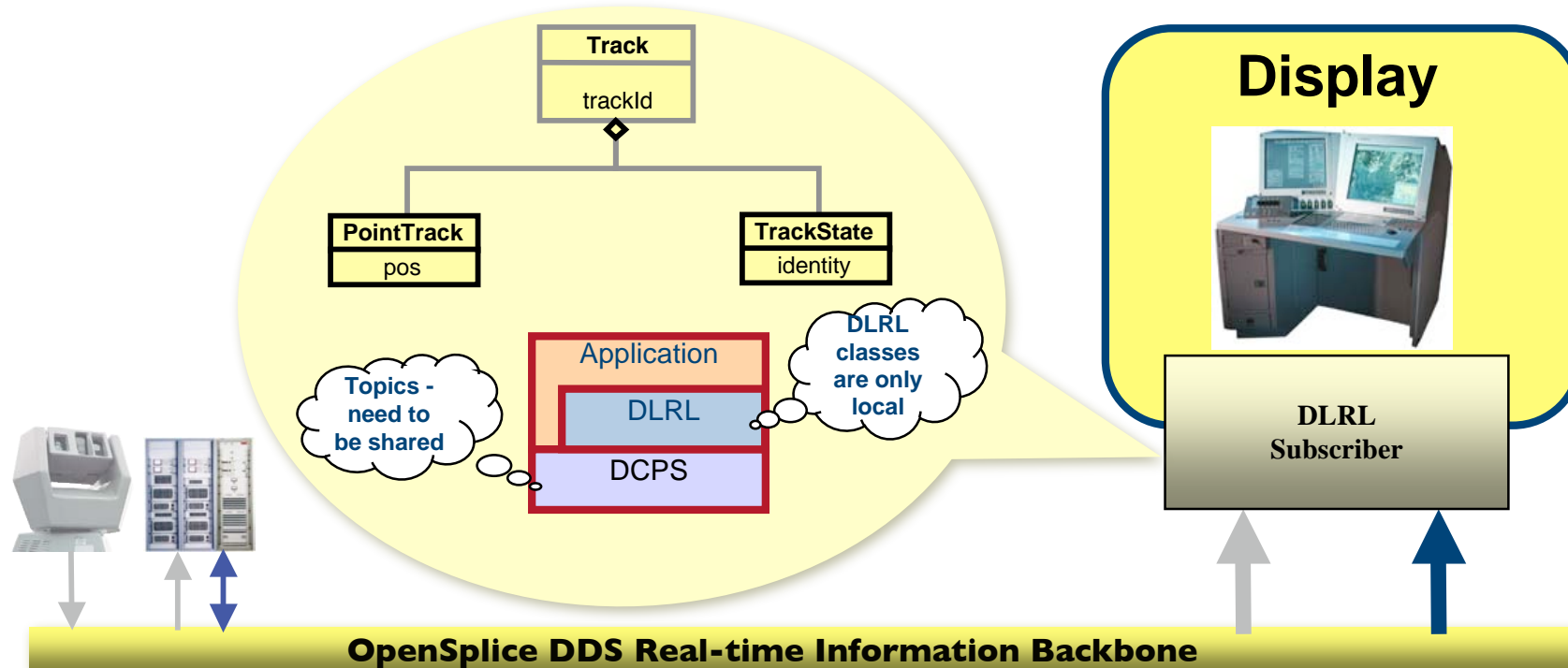
- ▶ Use 'content-aware' information backbone
 - ▶ To reduce application complexity
 - ▶ To increase system performance

Information Modeling Impact

- ▶ SQL-like filtering and/or querying on topic-attributes
 - ▶ Application can express fine-grained interest
 - ▶ Only relevant information needs to be processed

Utilizing the 'DLRL profile' features

30



System Requirements

- ▶ Reduced application complexity
 - ▶ Intuitive OO-interface (navigation)
 - ▶ Automatic change-management

Information Modeling impact

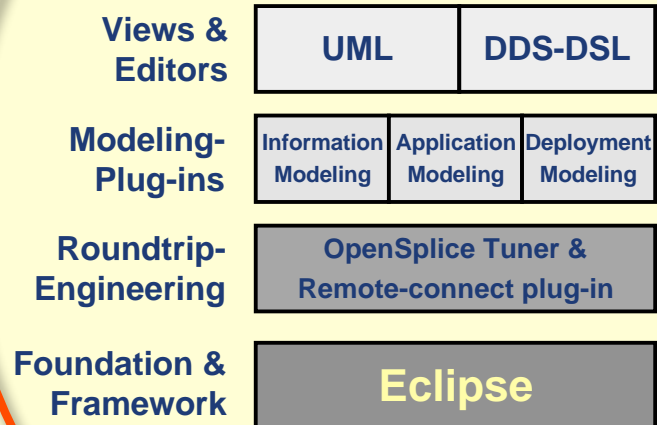
- ▶ Model an 'Track' object model on top of the DCPS topics
 - ▶ DLRL objects have user-defined assessor methods
 - ▶ DLRL objects have fine-grained listener mechanism
 - ▶ DCPS-topics will still be distributed

- ▶ **OpenSplice DDS Overview**
- ▶ **Information modeling**
- ▶ **DDS by Example**
- ▶ **Modeling the Example**
- ▶ **OpenSplice Powertools™**
- ▶ **Whats next**

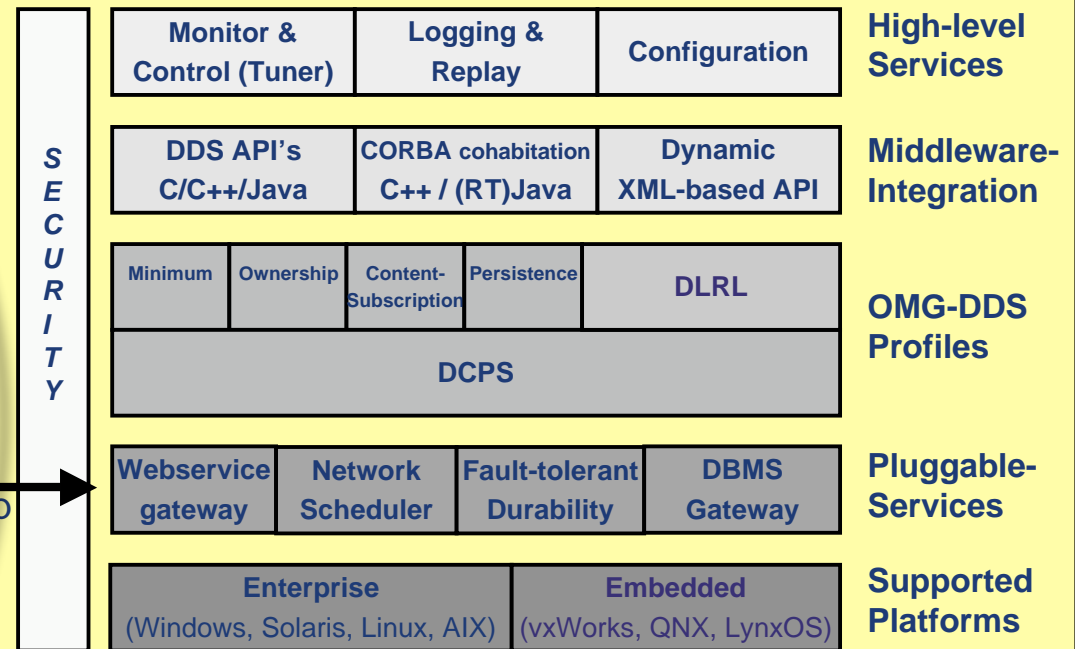
Deployment

Development

OpenSplice PowerTools™

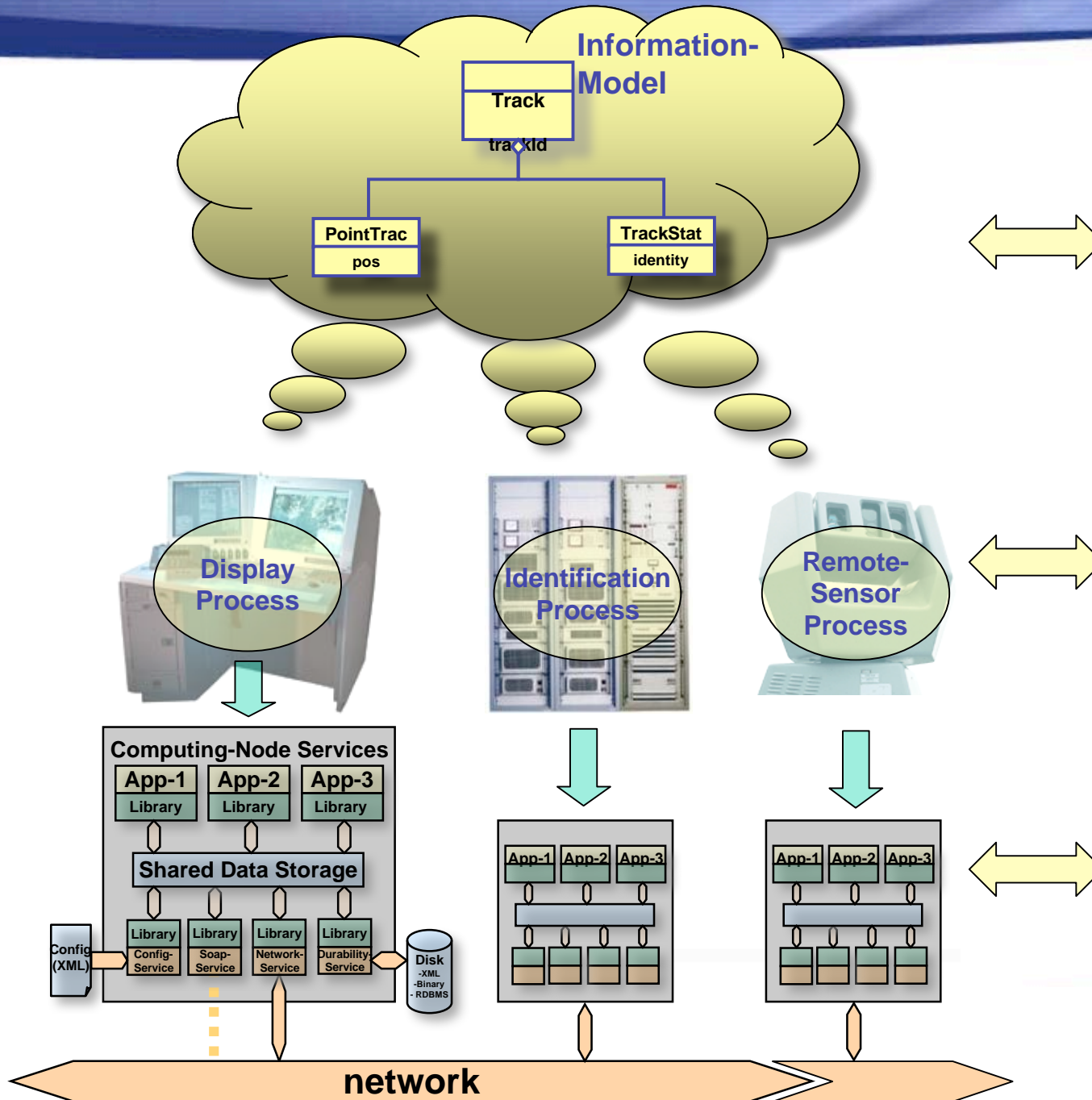


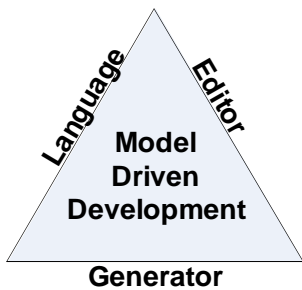
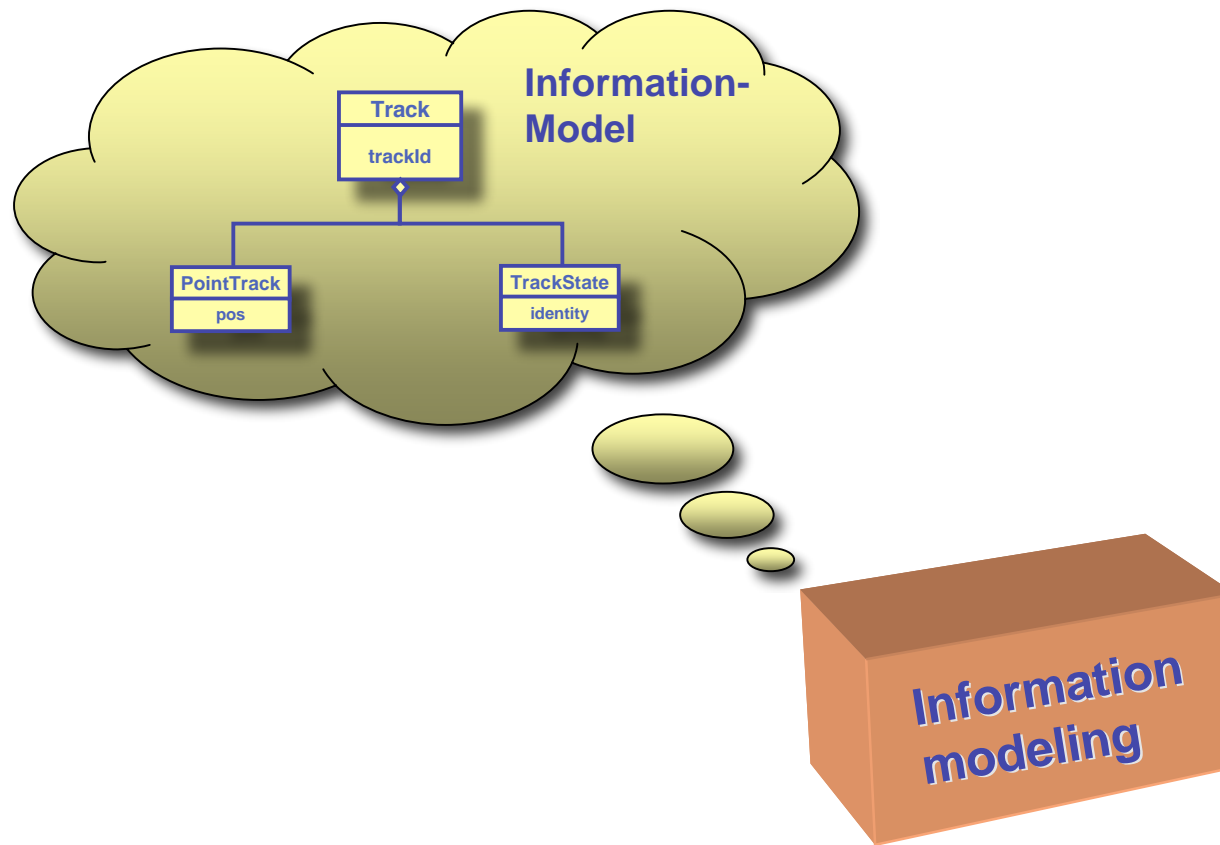
OpenSplice™ OMG-DDS product-line



OpenSplice DDS PowerTools™ MDE Suite

33



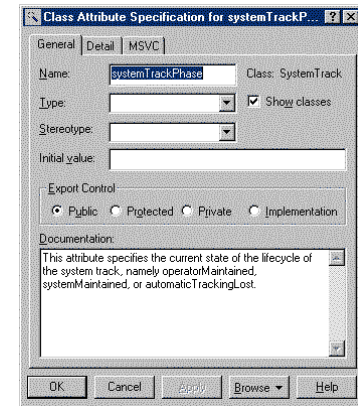


Information modeling: Scope & Purpose

35

Language

- ▶ Define the 'spoken language' in the system
 - ▶ Requirements phase: to talk with the customer
 - ▶ Development phase: to talk between components
- ▶ DDS-Domain specific 'Topics' (Type + keys + QoS)

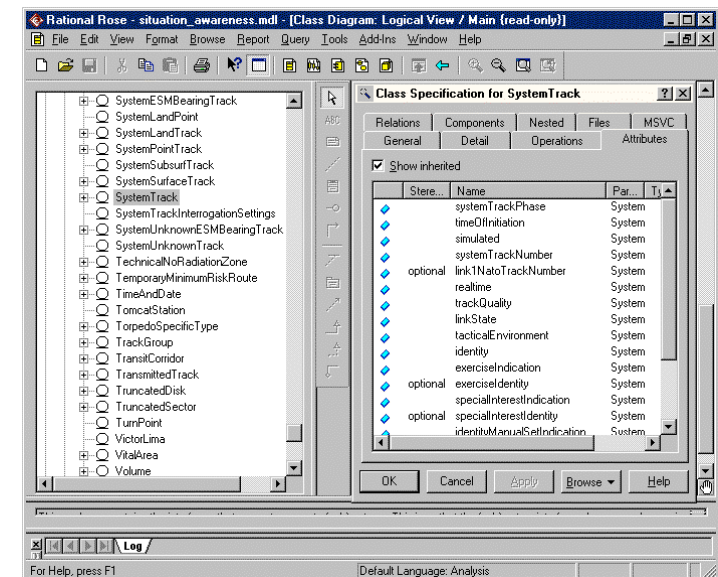


Editor

- ▶ Semantics: UML ('annotated')
- ▶ Syntax: IDL (can be imported)
- ▶ Behaviour: QoS (reliability, urgency, durability)

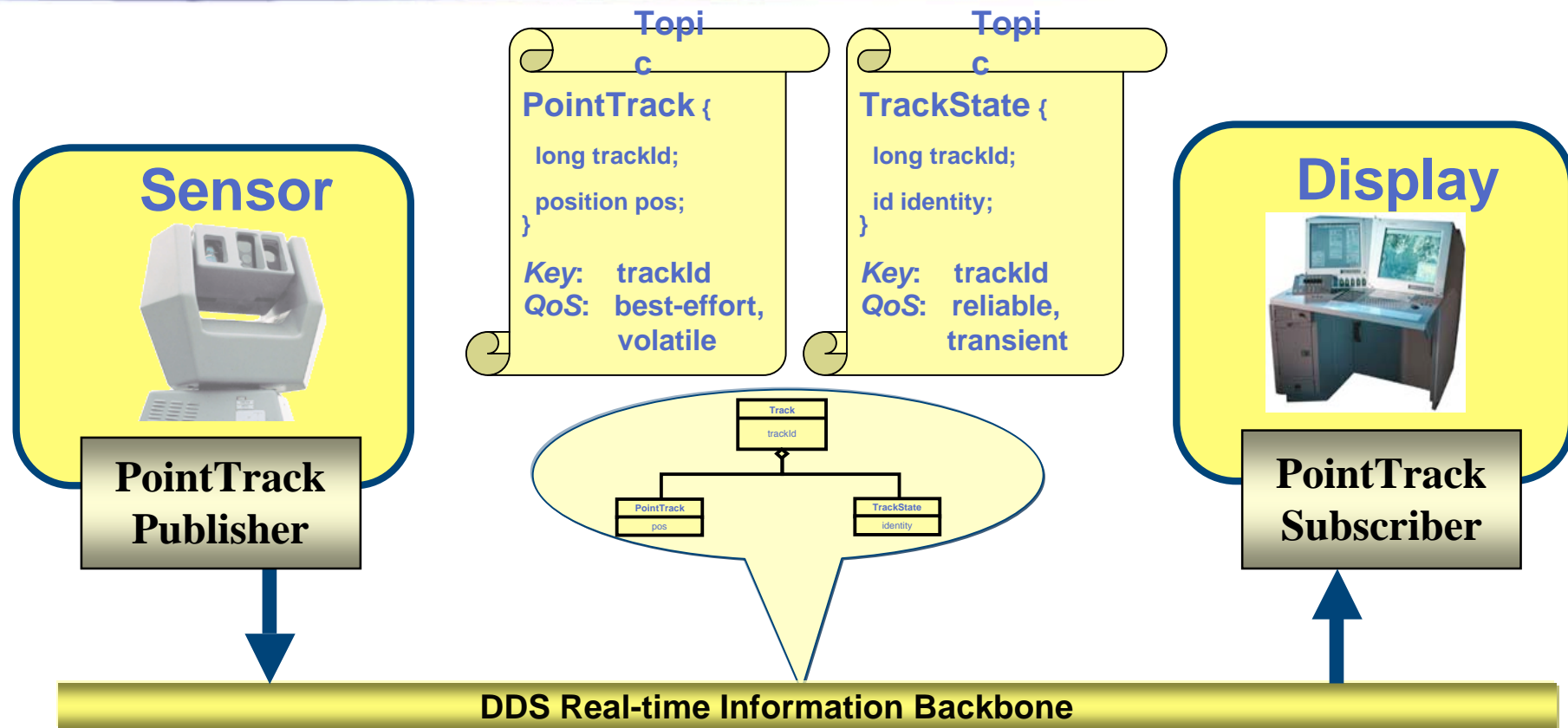
Generator

- ▶ Types (IDL) from UML (or direct IDL-import)
- ▶ Topics from Types
- ▶ DDS-entities
 - ▶ Topic-creation
 - ▶ Topic QoS-setup
 - ▶ Generated typed readers/writers



Information modeling (example)

36

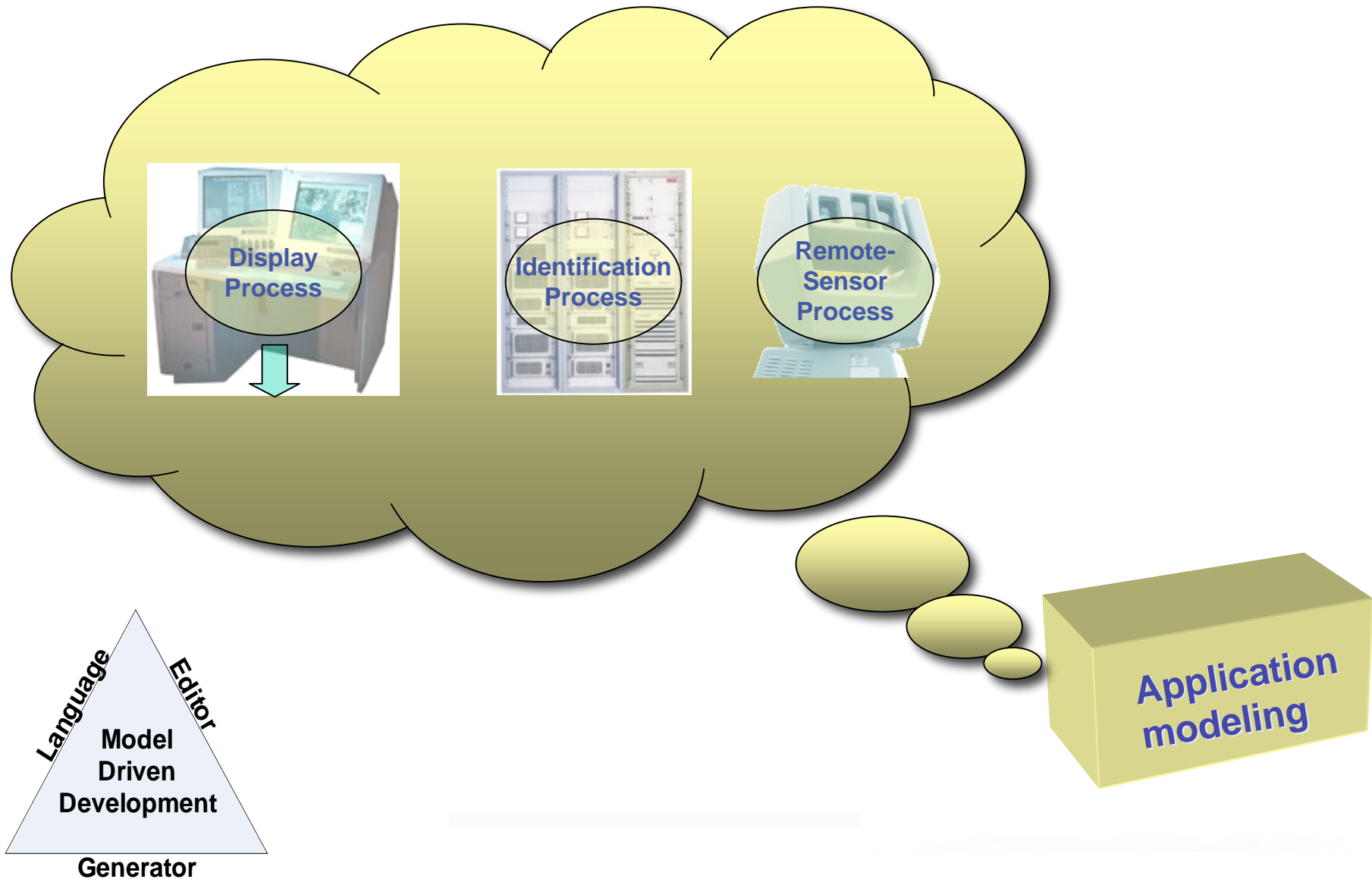


Task: Information modeling

- ▶ Model the 'PointTrack' and 'TrackState' topics
- ▶ Model the system-wide behaviour of topics (QoS)
- ▶ Model the relationships between the Topics (keys)
- ▶ Separate these concerns from applications

MDE: Features / Advantages

- ▶ Graphical modeling of structure and QoS (intuitive, fast)
- ▶ DDS-entity code generation (topics, interfaces)
- ▶ Allowing for direct utilization (by applications or tool)
- ▶ Documented packages of re-usable topic-sets



Language

- ▶ Processing-language
- ▶ DDS 'keywords'
 - ▶ publishers/writers, subscribers/readers,...
 - ▶ Content-filters, queries, waitsets, listeners,...

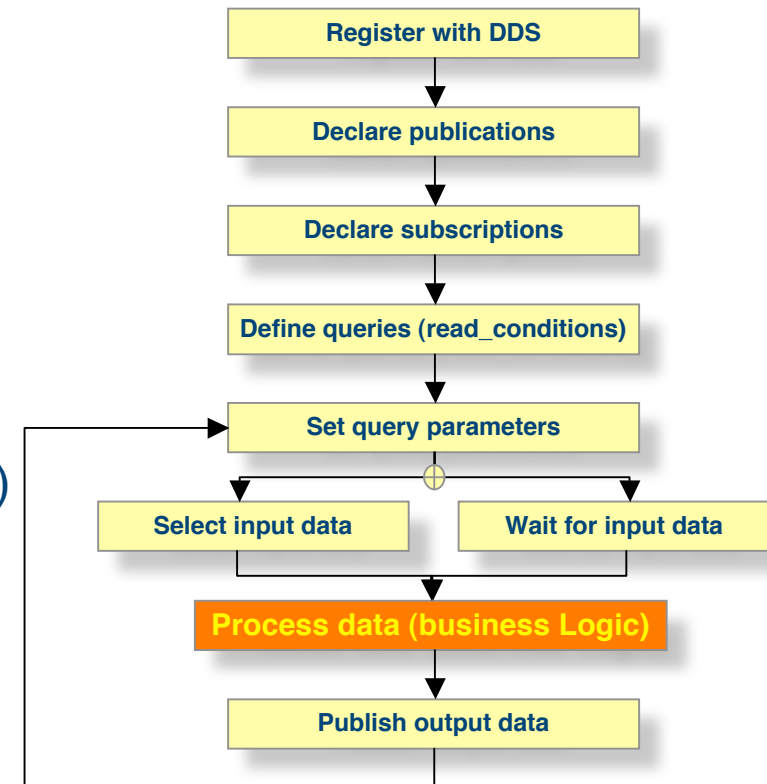
Editor

- ▶ Templates & Patterns (loop, MVC, ...)
- ▶ Application-level QoS modeling (history, filters)
- ▶ Process modeling (waitsets, listeners)

Generator

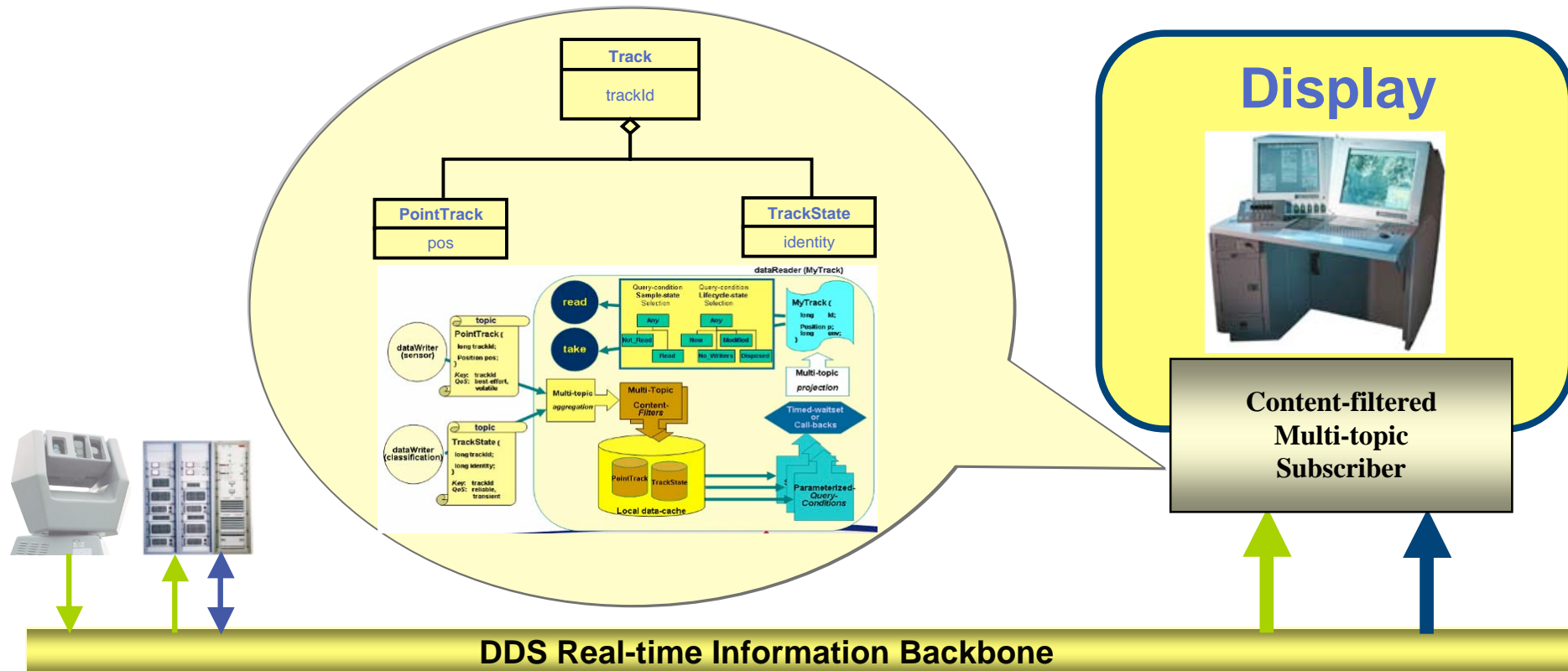
- ▶ Application-framework from templates
- ▶ DDS entities from (graphical) model
- ▶ Annotated application- and test-code

Simple DDS-Loop Pattern



Application modeling (example)

39

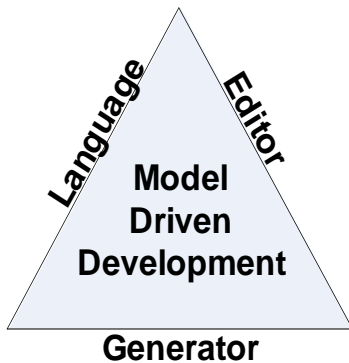
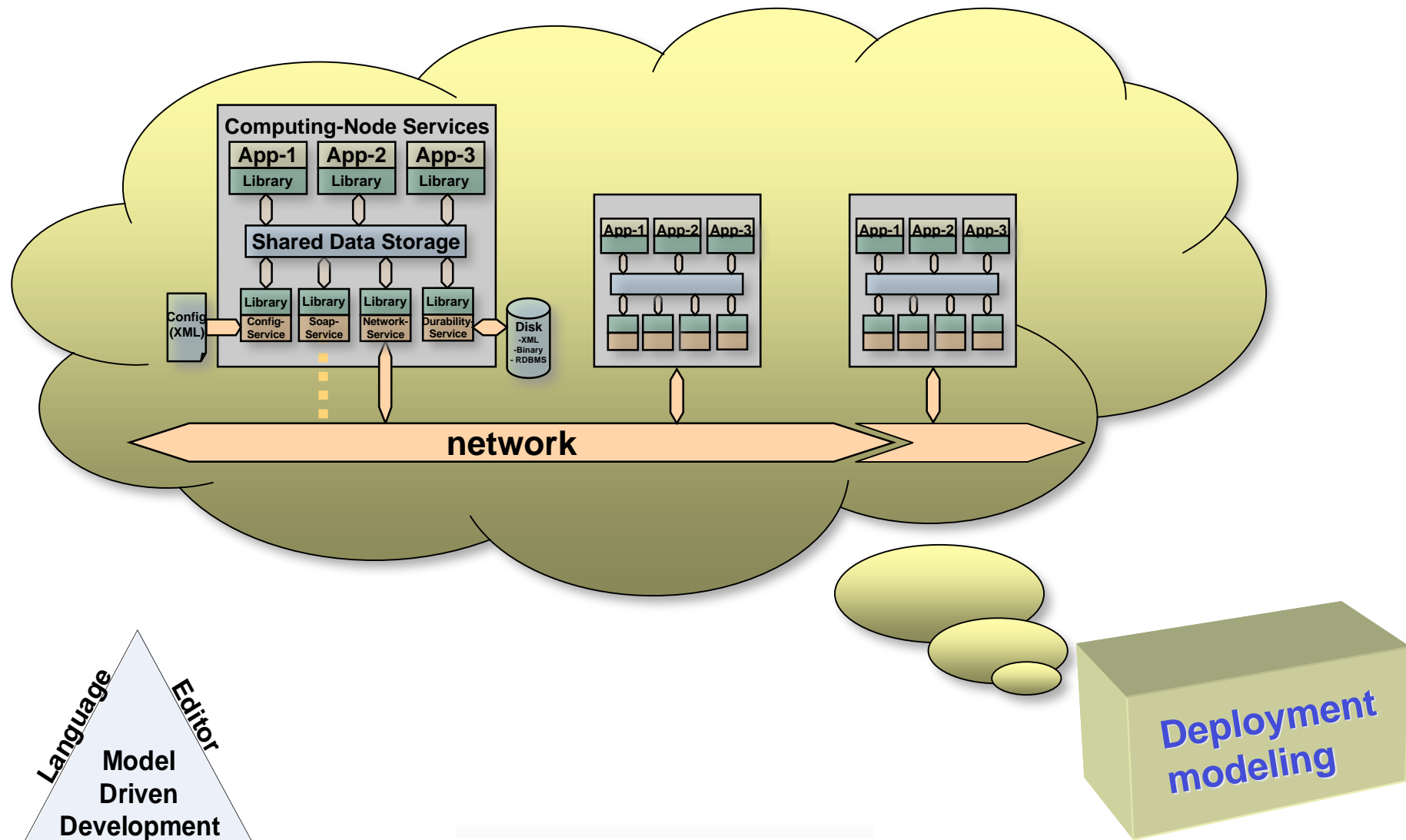


Task: Display Process modeling

- ▶ Model 'aggregate interest'
 - ▶ 'multi-topic' or 'DLRL-object' with local 'QoS'
- ▶ Model display process (periodic loop)
 - ▶ Handle first-appearances of hostile tracks with prio

MDE: Features / Advantages

- ▶ Import information-model
- ▶ Provide Application-framework (from 'loop' template)
- ▶ Model DLRL-objects, Multi-Topics, Queries, Waitsets,...
- ▶ Developer can concentrate on 'business-logic' (GUI)



Deployment modeling: Scope & Purpose

41

Language

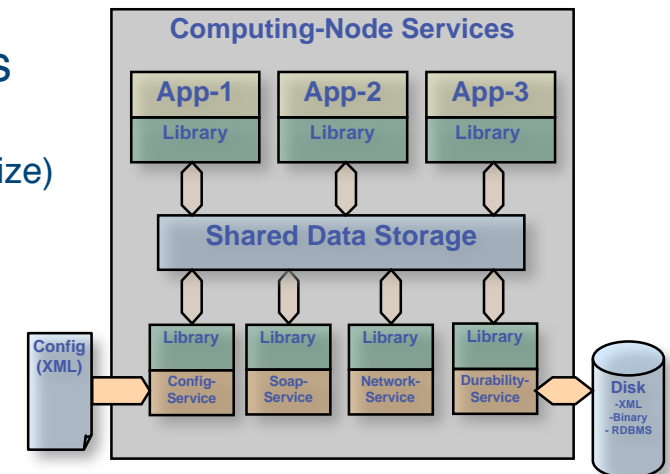
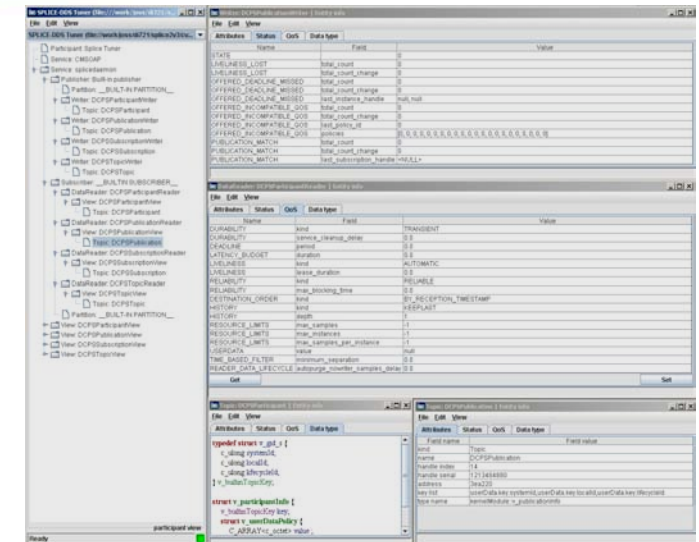
- ▶ Deployment-language
- ▶ DDS 'keywords'
 - ▶ Participants, Partitions, Resource limits,...
 - ▶ Latency-budget, Transport-priority,...

Editor

- ▶ Configure 'physical' DDS system
 - ▶ Networking, Durability, Resource-limits, ...
- ▶ Map dynamic DDS Policies to deployment properties
 - ▶ DDS 'partitions' to *Network-partitions* (multicast groups)
 - ▶ DDS 'transport-priority' to *Network-channels* (OS-prio/diffserv/burstsize)

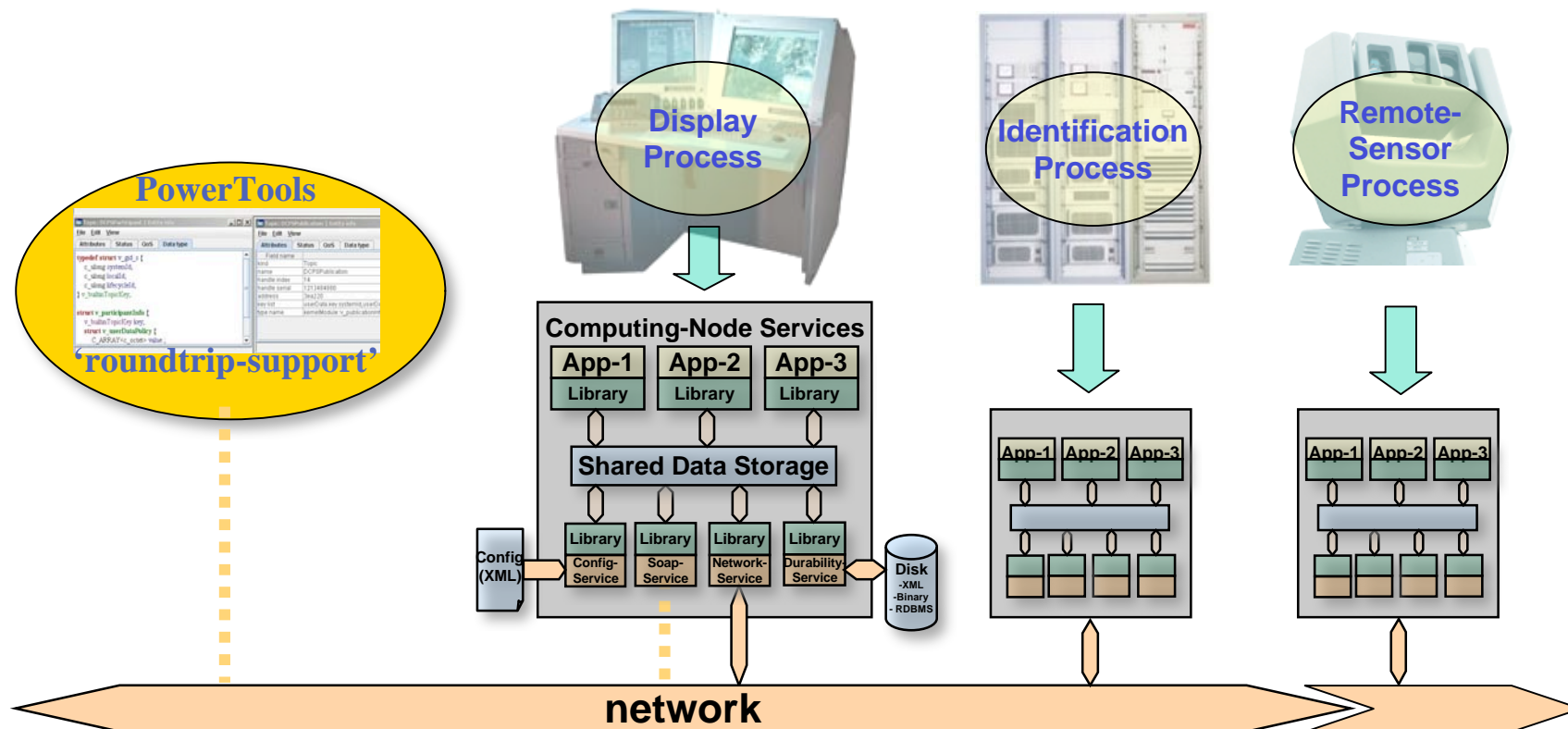
'Generator' (Control & Monitoring)

- ▶ (remote) set-up, configure and monitor DDS-system
- ▶ Deploy information-models (create readers/writers)
- ▶ Deploy applications (monitor, QoS-tuning)
- ▶ Deploy systems (logging/replay, QoS tuning,...)



Deployment modeling (example)

42



Task: Deployment modeling ('Control & Monitoring')

- ▶ Configure Durability-service (TRANSIENT TrackState topic)
- ▶ Configure Network-channels (priority, reactivity)
- ▶ Configure Resources (resource-limits)
- ▶ Remote control & Monitoring of deployed system

MDE: Features / Advantages

- ▶ Round-trip engineering
 - ▶ Deploy & tune models (info & application)
- ▶ Control & Monitor deployed system
 - ▶ Using the domain specific 'DDS-entity language'

Modeling the Example

43

OpenSplice Design - ::Battleship_Global_Design - OpenSplice DDS Power Tools

File Edit View Navigate Search Project Run Window Help

The screenshot displays the OpenSplice Design IDE interface. The **Project Explorer** on the left shows a project structure with folders like `bin`, `idl`, `model`, and `Sensor`. A context menu is open over the `Sensor` folder, listing actions such as `New Listener`, `Add Domain Participant`, `Rename`, `Delete`, `Refresh Content`, and `Generate Application Code`. The `Sensor` folder is expanded, showing sub-elements like `PointTrackAirDataW`, `PointTrackSeaDataW`, and `PointTrack`. The `PointTrack` element is further expanded, showing `padding`, `position`, and `trackId`. The `QoS` section is also visible.

The main workspace shows a diagram of DDS components. It includes four main components: `Sensor`, `Identification`, `Display`, and `Battlefield`. Each component has a `Participant` sub-component. The `Sensor` participant contains a `Publisher` and a `Subscriber`. The `Identification` participant contains a `Publisher` and a `Subscriber`. The `Display` participant contains a `Subscriber`. The `Battlefield` participant contains a `Publisher`. Arrows indicate data flow between these components, specifically from `PointTrack_topic` and `TrackState_topic` to the `Subscriber` in the `Sensor` and `Identification` participants, and from `ObjectLocation_topic` to the `Subscriber` in the `Battlefield` participant.

The **Properties** window at the bottom shows the `Sensor` component selected. The `Advanced` tab is active, displaying a table of properties:

Property	Value
Domain Participants	
Listeners	
Name	::Sensor::Sensor
Type	APPLICATION

The "Battleship" example 'at work'

Load Average Last Update

Current Memory (heap) Update Frequency (sec)

Current Memory (Non-heap)

Battlefield Objects

ID	X	X Velocity	Y	Y Velocity	Height
454	27	-1	115	0	55
460	18	-2	21	-1	18
461	118	-1	21	0	83
462	123	-1	45	-1	99
463	48	-2	106	0	28
466	122	0	138	-1	0
467	152	-1	165	0	32
468	155	-1	11	0	0
470	43	1	146	0	90
471	98	1	86	2	0

Battlefield Settings

Number of tracked objects Update frequency (ms)

ObjectLocation Data Writer

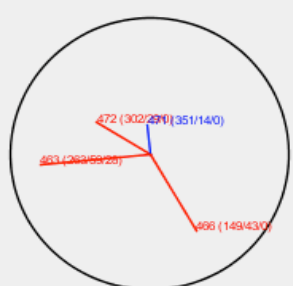
Latency Budget Transport Priority

Load Average Last Update

Current Memory (heap) Update Frequency (sec)

Current Memory (Non-heap)

Sensor



Info

ID	Dir	Dist	Ang
463	263	59	28
466	149	43	0
471	351	14	0
472	302	29	0

Target Selection

☐ Unknown
☐ Hostile
☐ Friendly
☒ All

PointTrack Data Reader

Latency Budget

TrackState Data Reader

Latency Budget

Summary

Total	No Position	Unknown (Blue)	Friendly (Green)	Hostile (Red)

Sensor

Performance

Load Average Last Update

Current Memory (heap) Update Frequency (sec)

Current Memory (Non-heap)

Detected Objects

Mon Sep 17 16:24:01 CEST 2007 -- ID: 472, DIST: 30.81, DIR: 305.75, ANG: 0.00
Mon Sep 17 16:24:01 CEST 2007 -- ID: 463, DIST: 59.36, DIR: 263.42, ANG: 28.14
Mon Sep 17 16:24:01 CEST 2007 -- ID: 466, DIST: 43.91, DIR: 149.93, ANG: 0.00
Mon Sep 17 16:24:01 CEST 2007 -- ID: 471, DIST: 14.14, DIR: 351.87, ANG: 0.00
Mon Sep 17 16:24:01 CEST 2007 -- ID: 472, DIST: 29.68, DIR: 302.62, ANG: 0.00

ObjectLocation Data Reader

Latency Budget

Ownership Strength

1 5 10

PointTrack DataWriter (Airborne Objects)

Latency Budget Transport Priority

PointTrack DataWriter (Sea-level Objects)

Latency Budget Transport Priority

Identification

Performance

Load Average Last Update

Current Memory (heap) Update Frequency (sec)

Current Memory (Non-heap)

Identified Objects

Identified Object Count: 4

ID	Classification	TrackState Handle
463	1	2757369004033
466	1	2731599200259
471	0	2851858284545
472	1	2877628088321

PointTrack Data Reader

Latency Budget

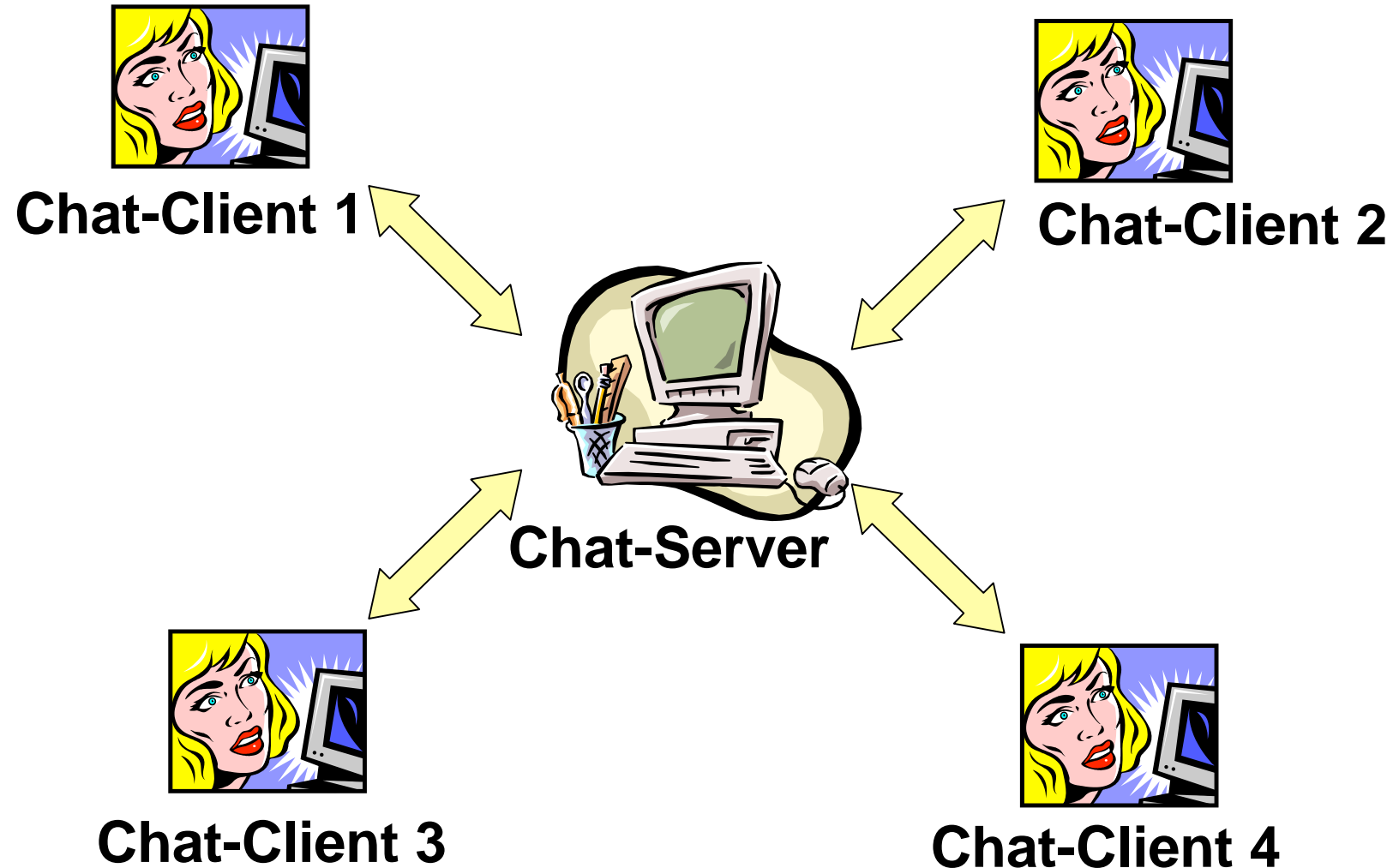
TrackState Data Writer

Latency Budget Transport Priority

- ▶ **OpenSplice DDS Overview**
- ▶ **Information modeling**
- ▶ **DDS by Example**
- ▶ **Modeling the Example**
- ▶ **OpenSplice Powertools™**
- ▶ **Whats next**

Chatroom Example – Traditional architecture

46



Typical sequence of events on a **traditional** Chat-application:

- **Connect** to the Chat-Server.
- **Transmit** your identity.
- **Download** the identities of the other chatters.
- **Receive** chat messages from other users.
 - These messages are **forwarded** to you by the server.
- **Write** your own chat-messages.
 - These messages are **forwarded** by the server to all the other users.

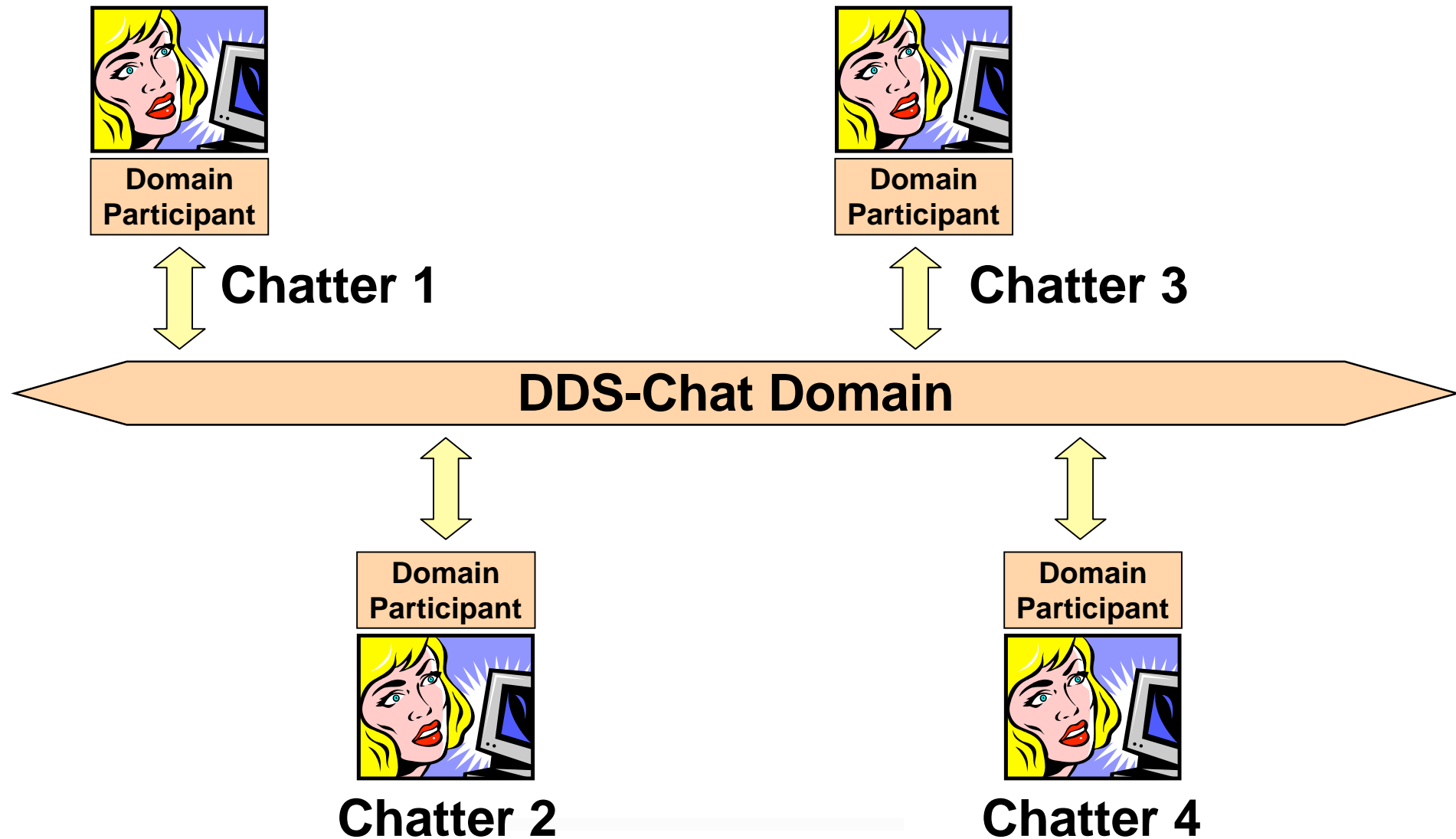


Typical sequence of events on a **DDS-based** Chat-application:

- **Participate** in the Chat-domain.
- **Publish** your identity.
- **Subscribe** yourself to the identities of all other chatters
- **Subscribe** yourself to all chat-messages in the Chat-domain.
 - All messages are delivered to you **directly** by their respective writers.
- **Publish** your own chat-messages.
 - Your messages are **directly** delivered to all the other interested users.

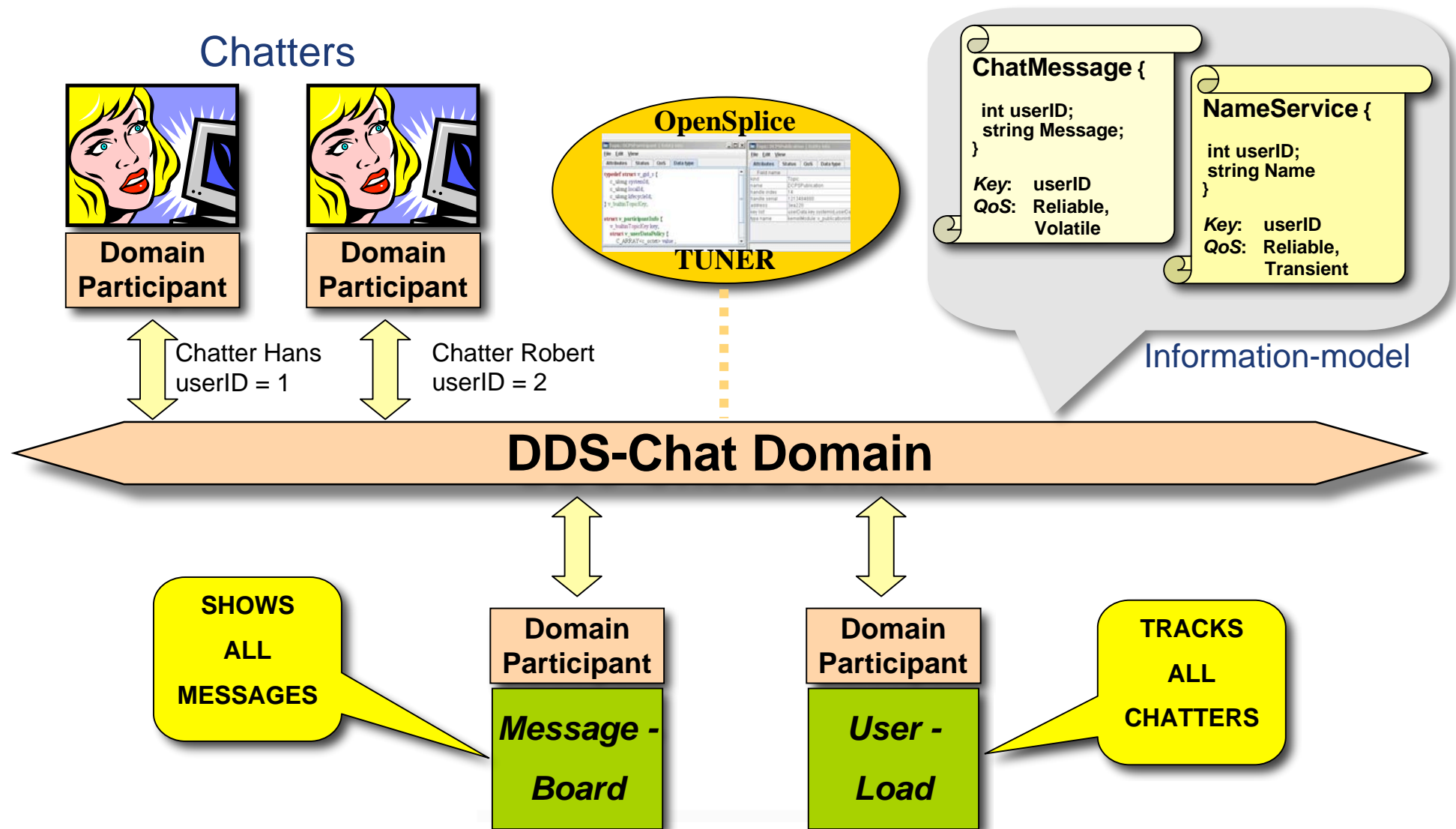
Chatroom Example – DDS based architecture

48

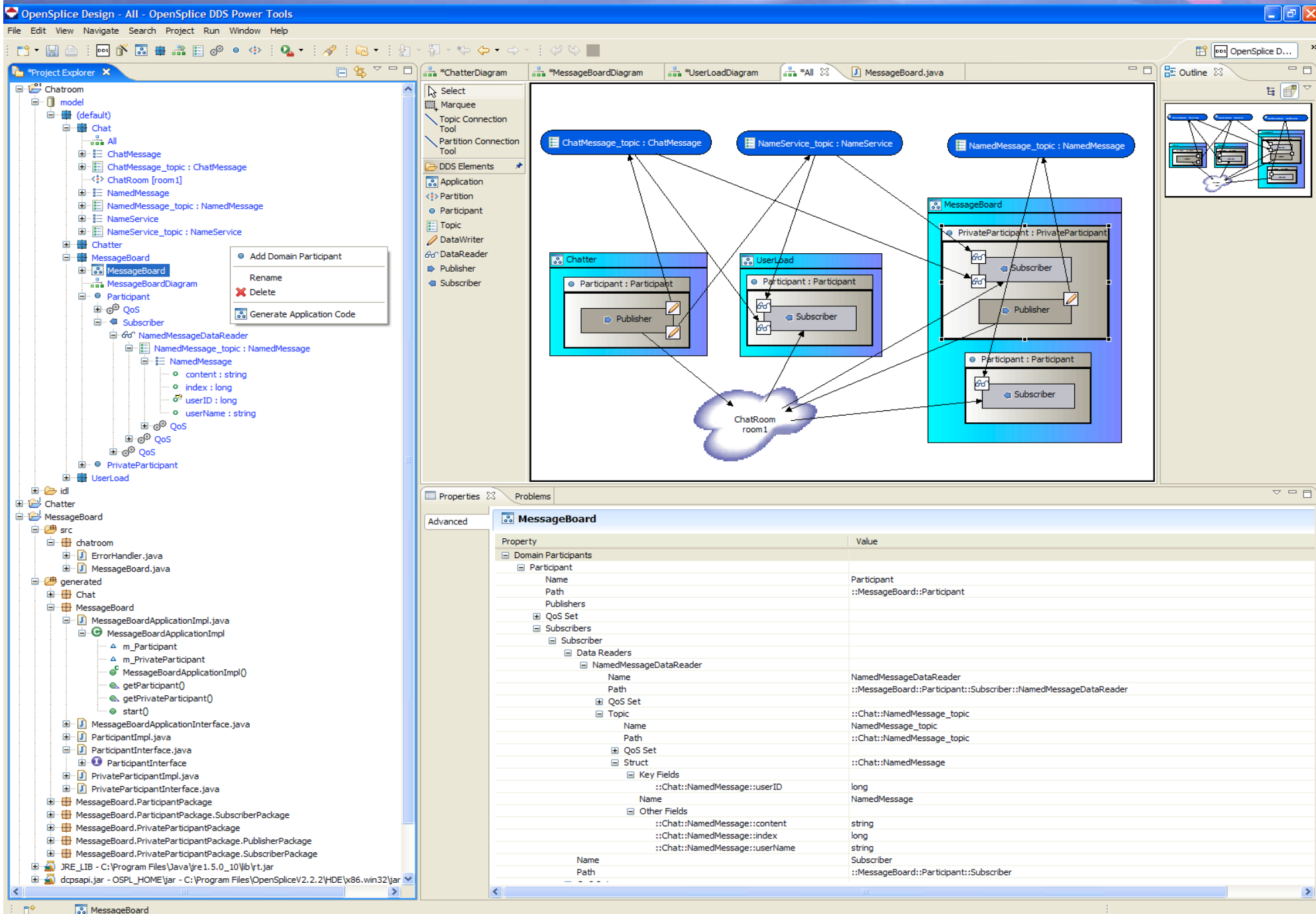


Chatroom Example – Applications

49



50



(51)

The screenshot displays the OpenSplice Design IDE with the following components:

- Project Explorer (Left):** Shows a hierarchical view of the project. The 'model' folder contains a 'Chatter' package, which includes a 'Participant' and a 'Publisher'. The 'Publisher' is associated with a 'NameService' topic, which has properties like 'name : string' and 'userID : long'. The 'Chatter' package also includes a 'QoS' set with various properties like 'Durability', 'History', 'Latency Budget', 'Lifespan', 'Liveliness', 'Ownership', 'Reliability', 'Resource Limits', 'Topic Data', and 'Transport Priority'.
- Diagram Area (Center):** Displays a UML-like diagram of the Chatter application. The 'Chatter' application contains a 'Participant' component, which in turn contains a 'Publisher' component. The 'Publisher' is connected to a cloud labeled 'ChatRoom room1'. The 'Publisher' also has two outgoing connections to external topics: 'NameService_topic : NameService' and 'ChatMessage_topic : ChatMessage'.
- Properties Panel (Bottom):** Shows the properties of the selected domain participants. The 'Domain Participants' section lists the 'Participant' and 'Publishers'. The 'Participant' properties include 'Name' (Participant) and 'Path' (::Chatter::Participant). The 'Publishers' section lists the 'Publisher' and its 'QoS Set'. The 'Publisher' properties include 'Name' (Publisher) and 'Path' (::Chatter::Participant::Publisher). The 'QoS Set' properties include 'Name' (Chatter) and 'Path' (::Chatter::Chatter).

The screenshot displays the OpenSplice Design IDE interface. The title bar reads "OpenSplice Design - Chatter.java - OpenSplice DDS Power Tools". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations, navigation, and execution.

The Project Explorer on the left shows a tree structure for the "Chatter" project. It includes a "model" folder with "Chat", "Chatter", "MessageBoard", and "UserLoad". Below this is an "idl" folder containing "Chat.idl". The "src" folder contains "chatroom", "Chatter.java", "Chatter", and "ErrorHandler.java". The "JRE_LIB" folder lists several JAR files. The "generated" folder contains a "Chat" package with "Chatter", "Chatter.ParticipantPackage", "PublisherImpl.java", "PublisherInterface.java", "Chatter.ParticipantPackage.PublisherPack", "ChatMessageDataWriterImpl.java", "ChatMessageDataWriterInterface.java", "NameServiceDataWriterImpl.java", and "NameServiceDataWriterInterface.java". The "idl" folder also contains "exported_ChatMessage.idl", "exported_ChatMessage.idl.log", "exported_Chat.idl", "exported_Chat.idl.log", "exported_NameService.idl", and "exported_NameService.idl.log".

The main editor window displays the "Chatter.java" file. The code is as follows:

```
ChatterName = "Chatter" + ownID;
}
else
{
    chatterName = "Chatter" + ownID;
}

/* Type-specific DDS entities */
ChatMessageDataWriter talker = chatterApp.getParticipant().getPublisher().getChatMessageDataWriter().getDataWriter();
NameServiceDataWriter nameServer = chatterApp.getParticipant().getPublisher().getNameServiceDataWriter().getDataWriter();

/* Initialize the NameServer attributes */
ns.userID = ownID;
ns.name = chatterName;

/* Write the user-information into the system (registering the instance implicitly) */
status = nameServer.write (ns, HANDLE_NIL.value);
ErrorHandler.checkStatus (status, "Chat.NameServiceDataWriter.write");

/* Initialize the chat messages */
msg.userID = ownID;
msg.index = 0;
msg.content = "Hi there, I will send you " + NUM_MSG + " messages.";
System.out.println ("Writing message: \"\" + msg.content + "\"");

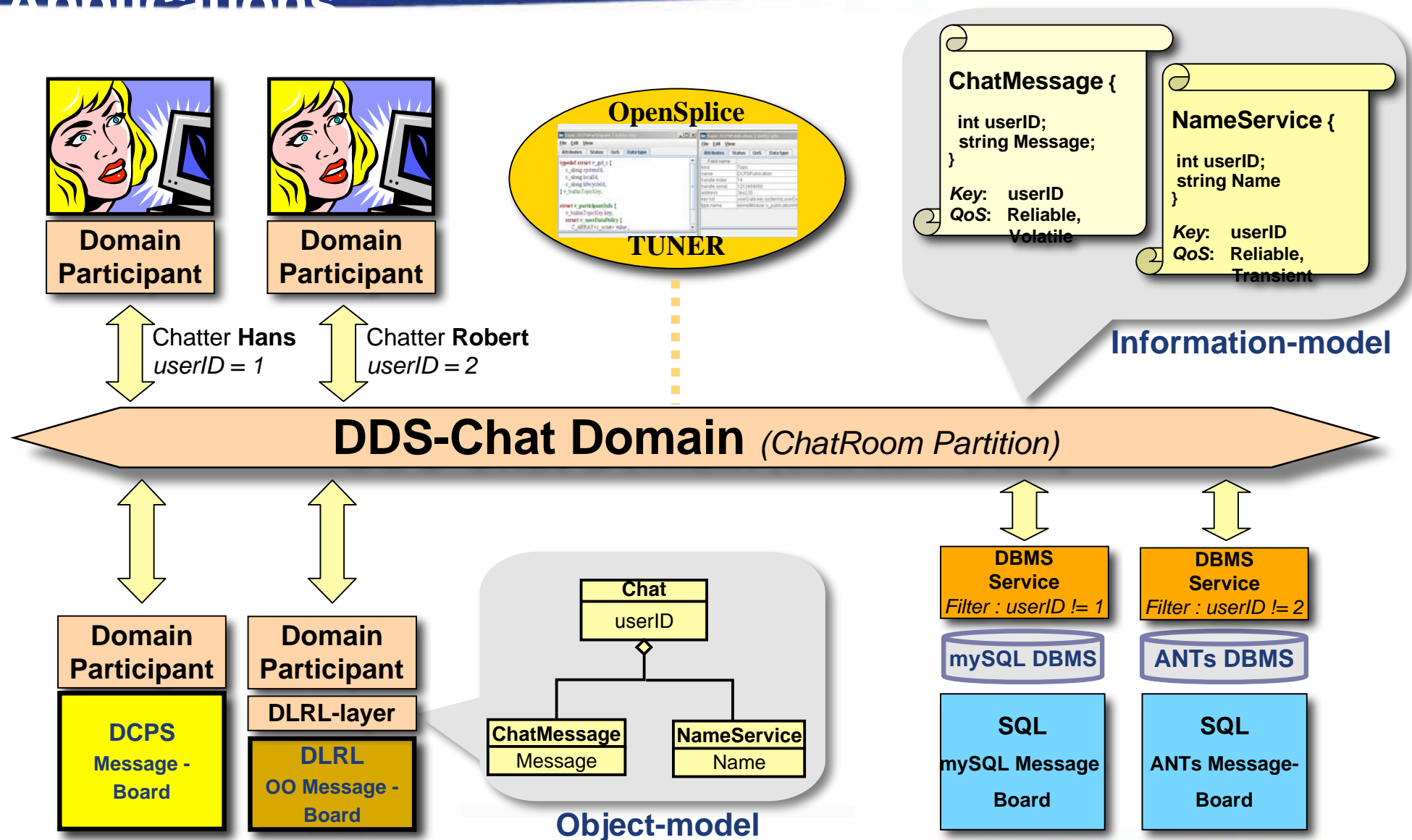
/* Register a chat message for this user (pre-allocating resources for it!!) */
userHandle = talker.register_instance (msg);

/* Write a message using the pre-generated instance handle */
status = talker.write (msg, userHandle);
ErrorHandler.checkStatus (status, "Chat.ChatMessageDataWriter.write");
```

The Properties and Problems panels at the bottom show "0 errors, 0 warnings, 0 infos". The bottom status bar indicates "Writable", "Smart Insert", and "73 : 33".

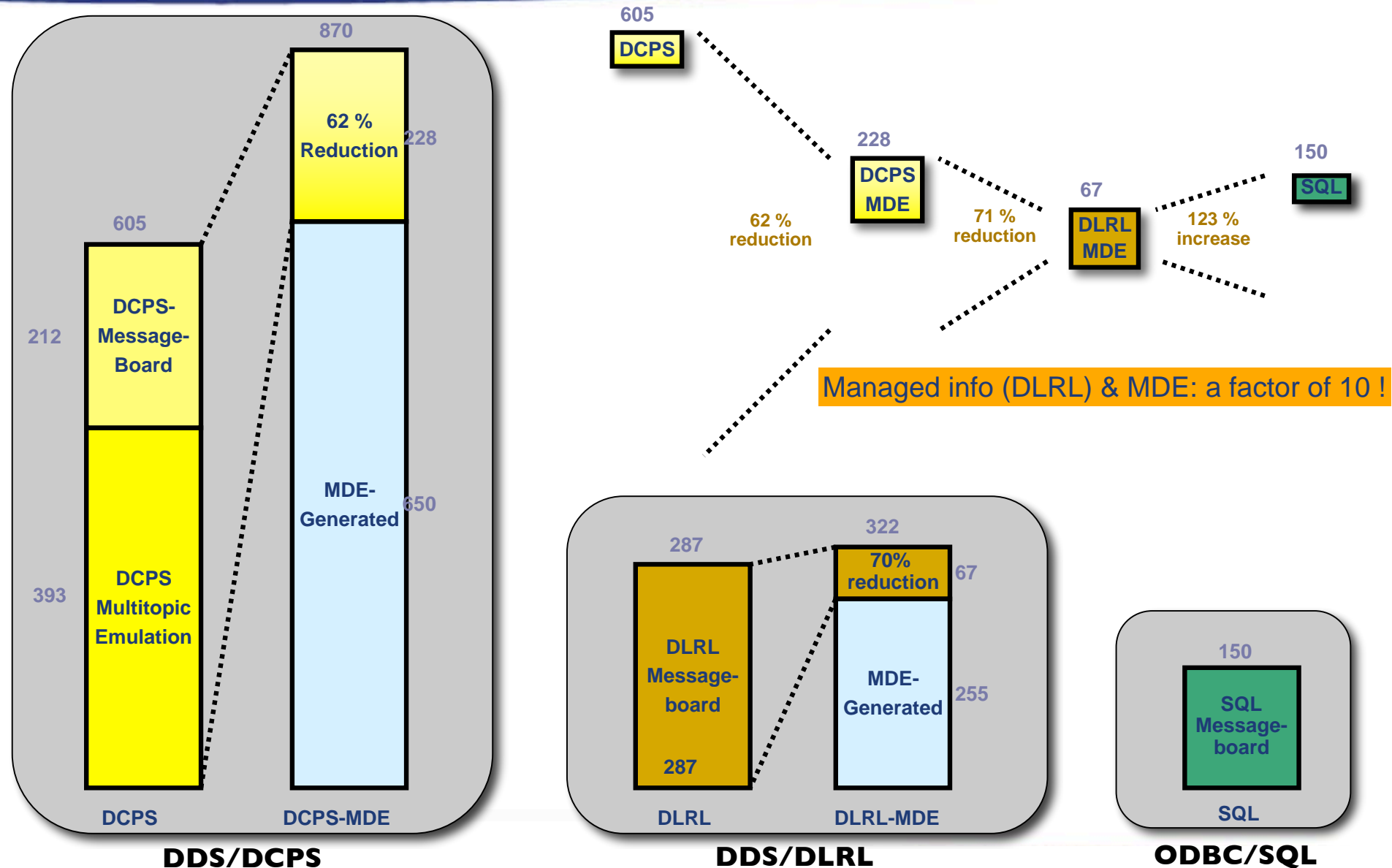
Chatroom Example – DCPS/DLRL & SQL Applications

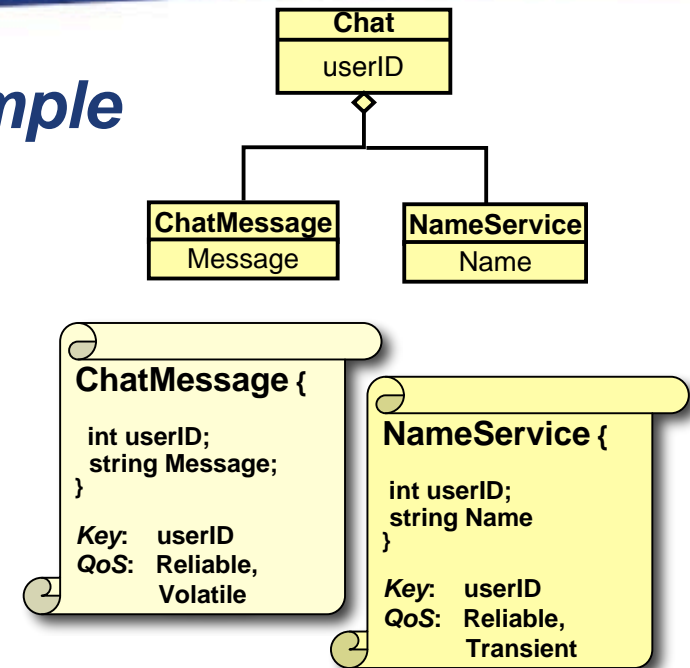
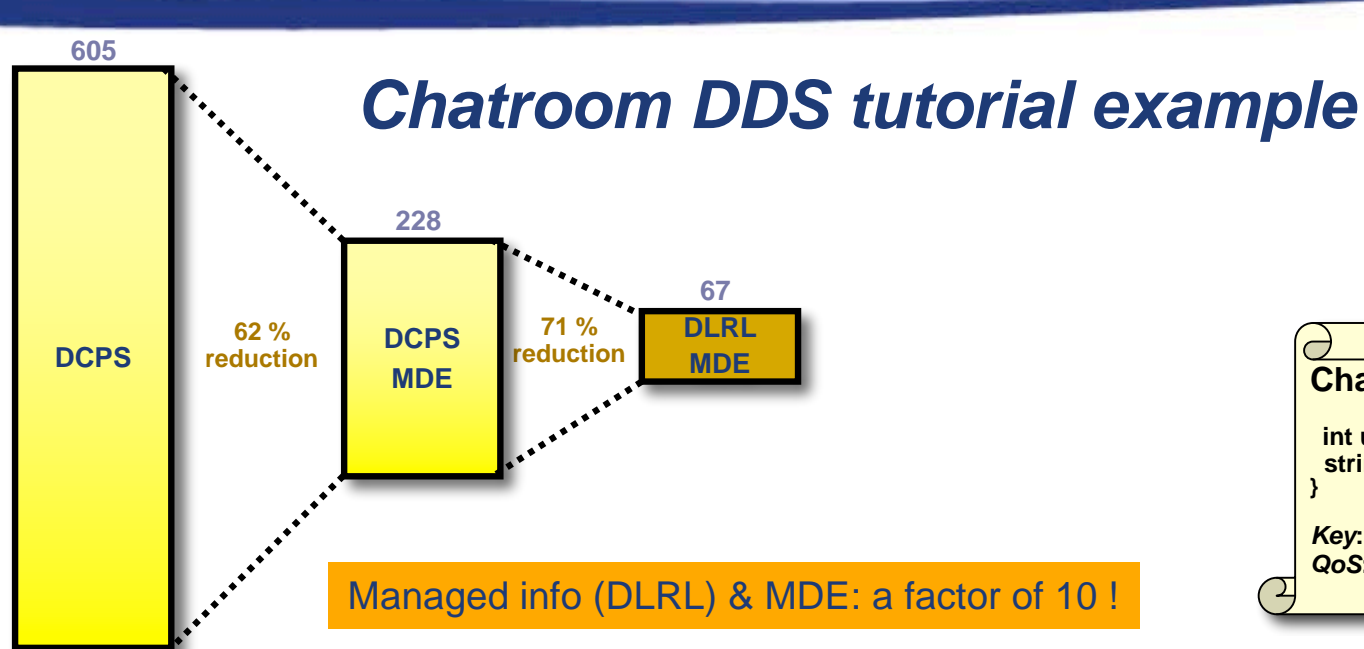
53



MessageBoard Example – Comparing architectures by size

54





OpenSplice DDS, PowerTools™ MDE-suite for modeling & code-generation

- Supports information/topic modeling: IDL-import and QoS annotation
- Supports application modeling: DDS entity/interaction modeling and code-generation
- Supports deployment modeling: OpenSplice Tuner™ for remote monitor & control (incl. QoS tuning)

OpenSplice DDS, DLRL API

- Object relationship-management greatly reducing application complexity
- Extensive selection and fine-grained listener mechanisms ease application design
- High-performance/low-overhead due to DLRL-support by DCPS-kernel in-memory OO-database

Application size

- Chatroom example (simple application) shows a **62% LOC reduction by MDE** another **71% LOC reduction by DLRL**

MDE tool-suite: Summary & Advantages

56

Complete modeling of system design cycle

- ▶ Information/application/deployment Modeling
- ▶ Context aware guidance / well defined steps
- ▶ **Fast, intuitive, correct**

Information Modeling

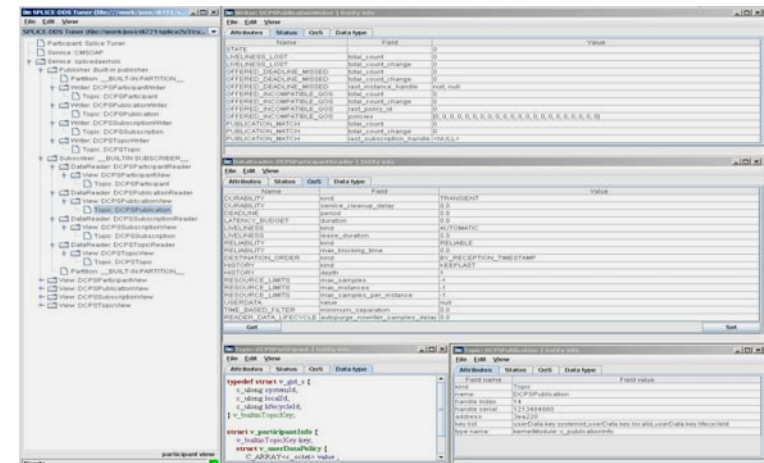
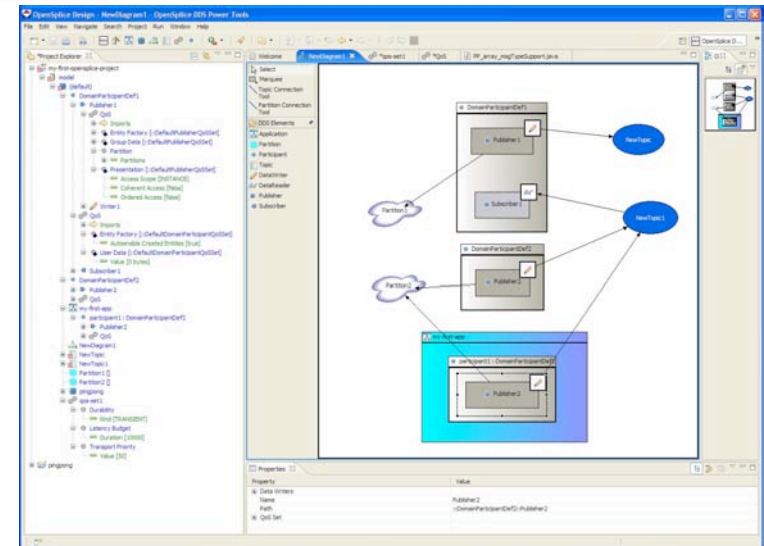
- ▶ Graphical system-wide information + QoS modeling
- ▶ DDS code-generation of topics and typed readers/writers
- ▶ **Documented packages of re-usable topic-sets**

Application Modeling

- ▶ Information-model (parts) import
- ▶ Graphical Application Modeling
- ▶ **Code-generation from patterns (listener/waitset/MVC)**

Deployment Modeling

- ▶ Modeling of DDS-configuration
- ▶ Service configuration (networking, durability)
- ▶ **Runtime control (& round-trip engineering) by OpenSplice Tuner™**



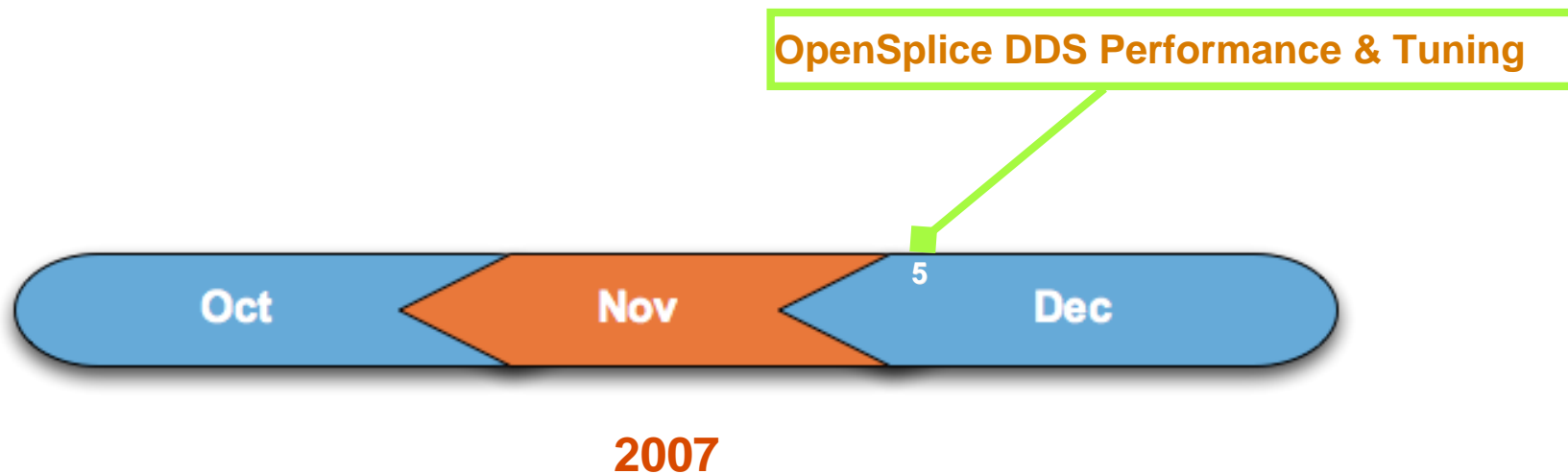
- ▶ **OpenSplice DDS Overview**
- ▶ **Information modeling**
- ▶ **DDS by Example**
- ▶ **Modeling the Example**
- ▶ **OpenSplice Powertools™**
- ▶ **Whats next**



Dr. Angelo Corsaro

Upcoming Webinars

58



Registration: <http://www.prismtech.com/section-item.asp?id=731&sid=29&sid2=15&sid3=289>

Architecture

- ▶ OpenSplice DDS™ uniquely implements the full OMG DDS v1.2 standard combining Real-time Pub/Sub with Data-centricity
- ▶ OpenSplice DDS™ can therefore significantly reduce system complexity and enhance component re-use

MDE Tool suite

- ▶ OpenSplice PowerTools™ optimally support data-centric system engineering
- ▶ OpenSplice PowerTools™ Promote a clear separation of concerns between information-modeling, application-modeling/code-generation and deployment
- ▶ Java/Eclipse based toolsuite for development, deployment as well as remote maintenance

OpenSplice DDS is the best solution available on the market for solving your data distribution problems!

OpenSplice™ | DDS

- ▶ OpenSpliceDDS Home Page
 - ▶ <http://www.prismtech.com/opensplice-dds/>

- ▶ For Information on OpenSplice DDS, or for evaluation licenses, contact:
 - ▶ opensplicedds@prismtech.com -or-
 - ▶ sales@prismtech.com

- ▶ OMG DDS Information
 - ▶ <http://www.dds-forum.org/>
 - ▶ <http://portals.omg.org/dds/>