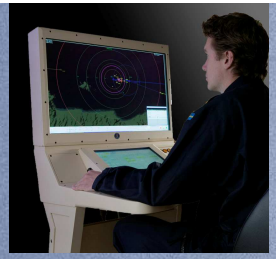
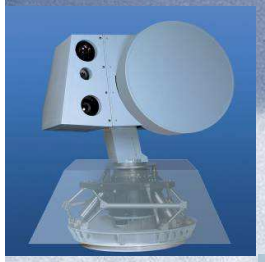




# Model Driven DDS with Java

*using OMG EMOF and OSGi*

Magnus Rundlöf
Lars Millberg

Arlington, VA, 2008



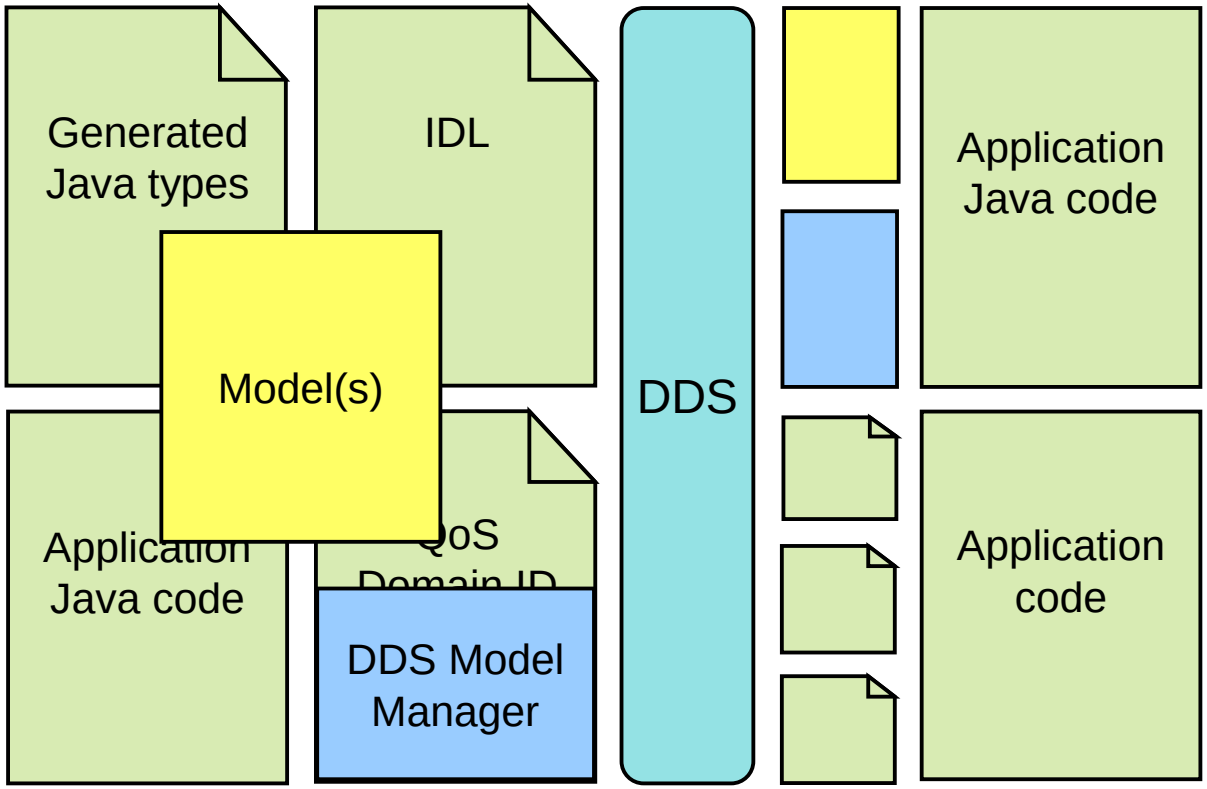


SAAB TECHNOLOGIES SAAB

# Who are we?

- Lars Millberg and Magnus Rundlöf
- Saab AB, Saab Systems, Naval Systems Division Sweden
- Been working together for 1.5 years in an internal R&D project
- Worked a lot with open standards and model driven development


# Introduction


Generated
Java types
IDL
Model(s)
DDS
Application
Java code
Application
Java code
DDS Model
Manager
Application
code

# Model Driven DDS

MDDS

# Prerequisite: EMF

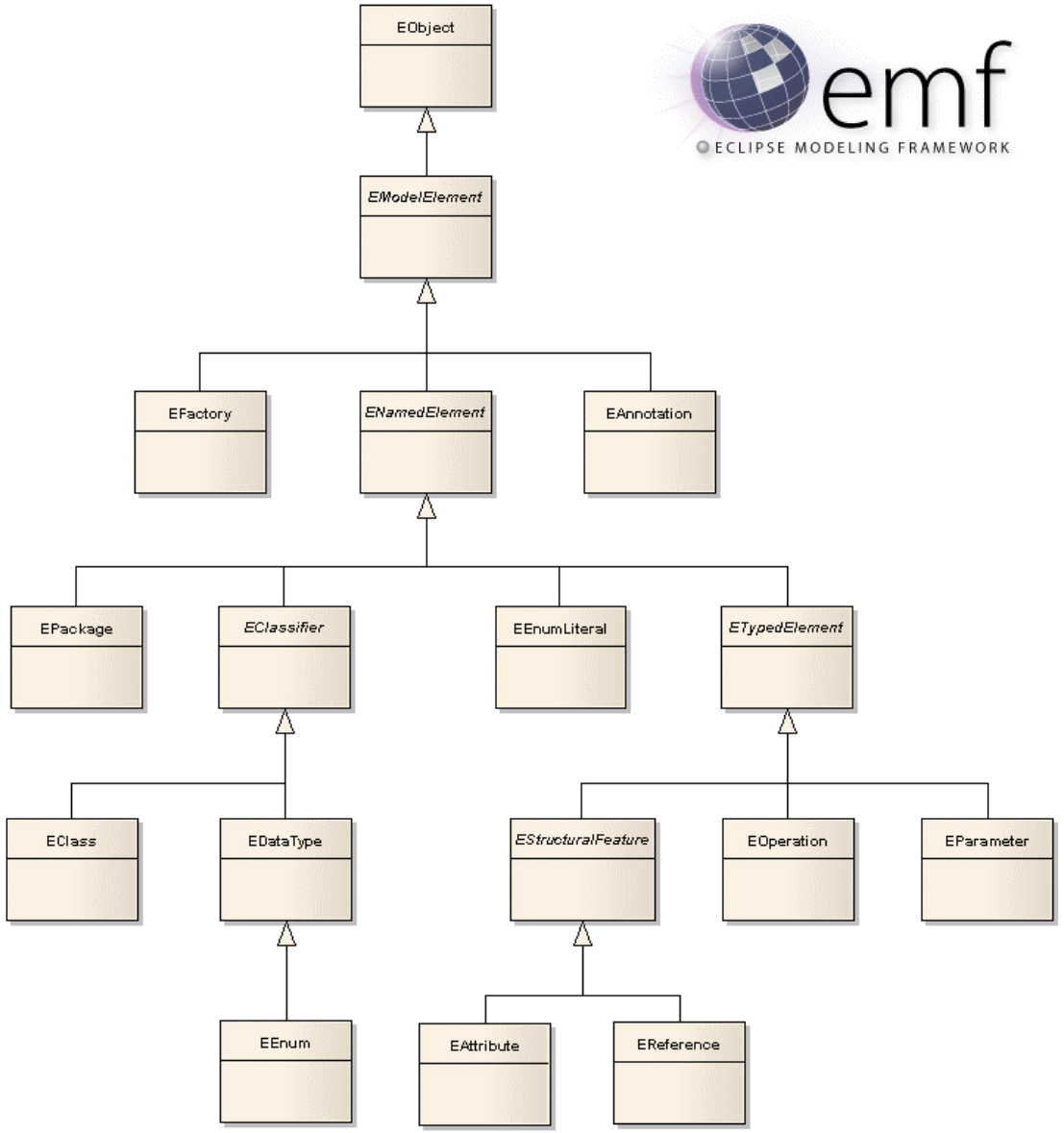
# Eclipse Modeling Framework


- A tool for domain-specific modeling
- Unifies three major technologies in modern software engineering
- A Java framework and code generation facility for building applications from structured (formal) data models



# The Ecore Model

- The modeling language of EMF
- Almost equivalent to OMG EMOF
- Essentially a simplified subset of the UML Class Diagram package



# EMF and MDDS


- MDDS heavily relies on EMF
- Three Ecore models constitute the MDDS Meta models
  - Data Model
  - QoS Model
  - Interface Model
- Separated this way to enable reuse of data types and QoS settings
- The major part of code in MDDS is generated by EMF, including the Eclipse editors for creating and editing instances of the MDDS Meta classes

To the limits!

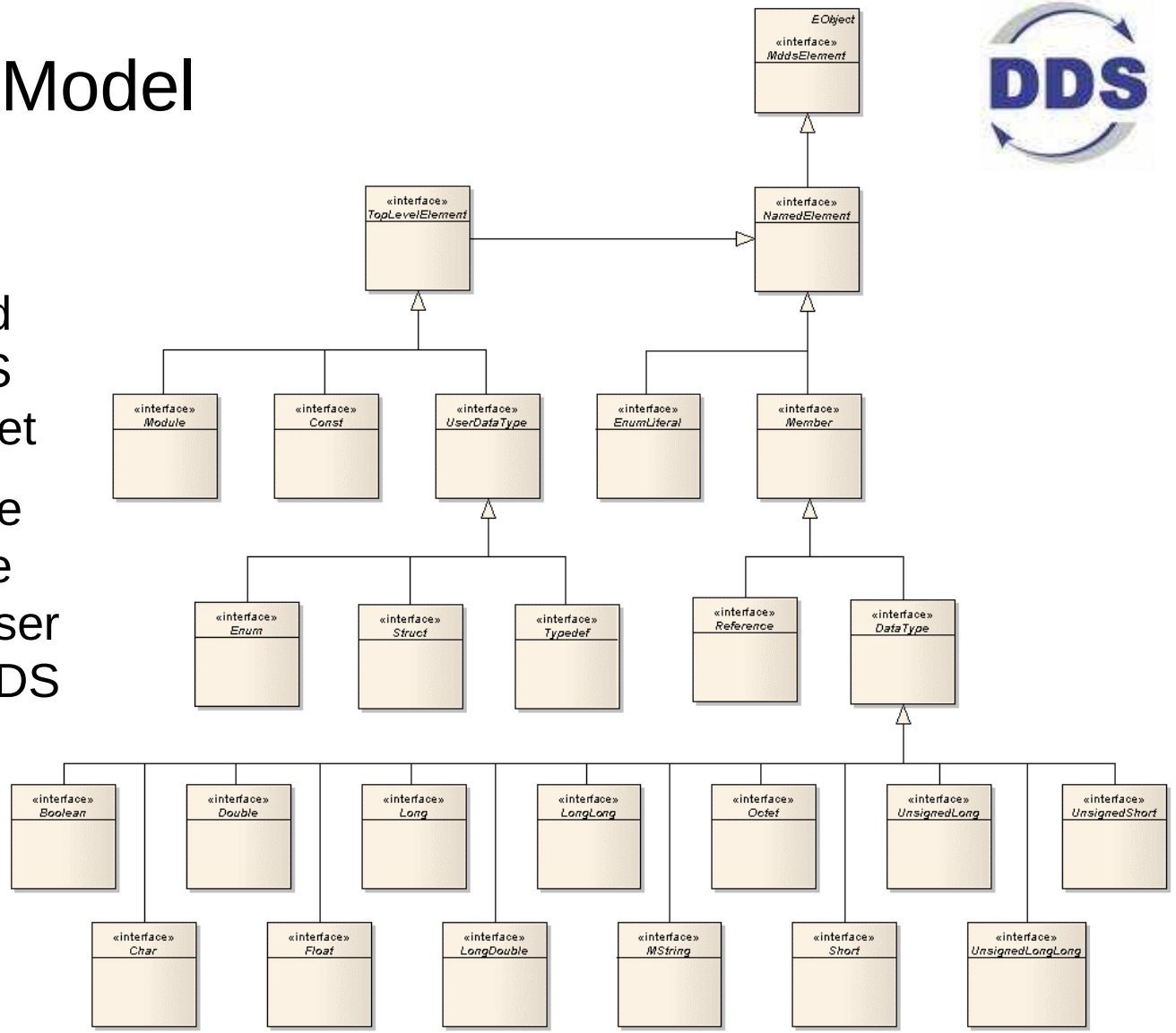
# EMF and MDDS


- MDDS heavily relies on EMF
- Three Ecore models constitute the MDDS Meta models
  - Data Model
  - QoS Model
  - Interface Model
- Separated this way to enable reuse of data types and QoS settings
- The major part of code in MDDS is generated by EMF, including the Eclipse editors for creating and editing instances of the MDDS Meta classes

The meta models…

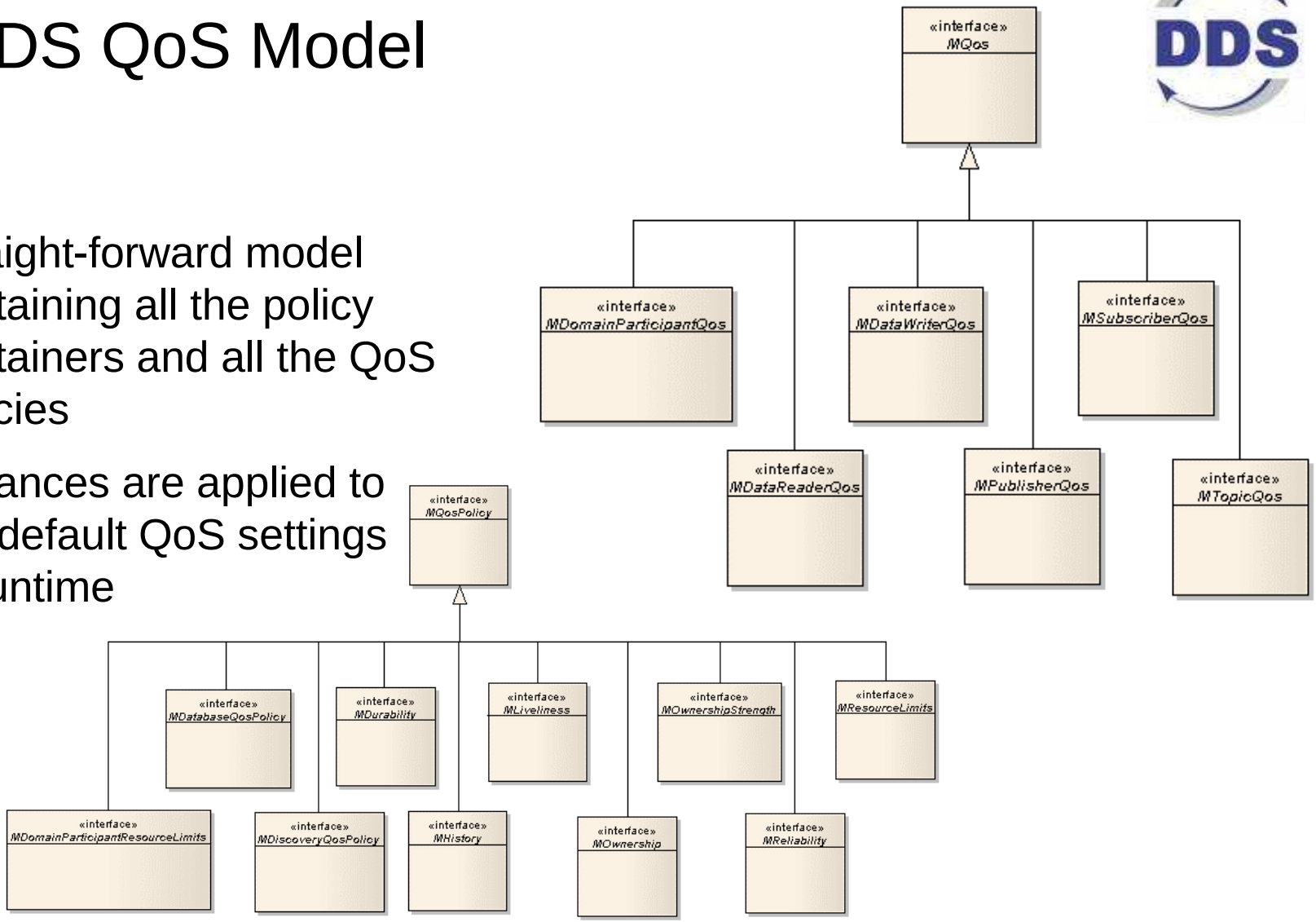
# MDDS Data Model

- A straight-forward model of the DDS specific IDL subset
- Instances of these model objects are used to specify user data types in MDDS



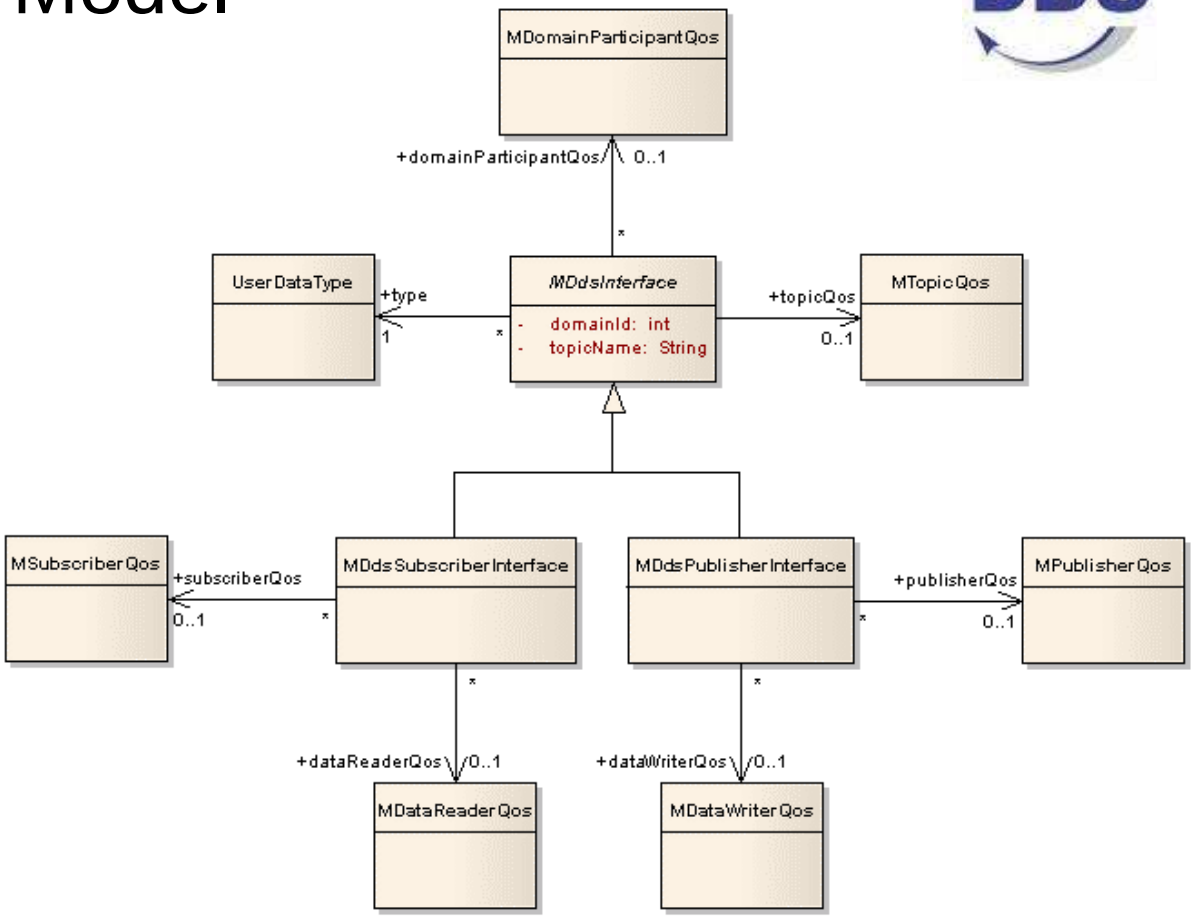
# MDDS QoS Model

- Straight-forward model containing all the policy containers and all the QoS policies
- Instances are applied to the default QoS settings in runtime



# MDDS Interface Model

- Two concrete interface types:
  - Subscriber interface
  - Publisher interface
- An instance contains all pieces of information needed to subscribe or publish on a topic



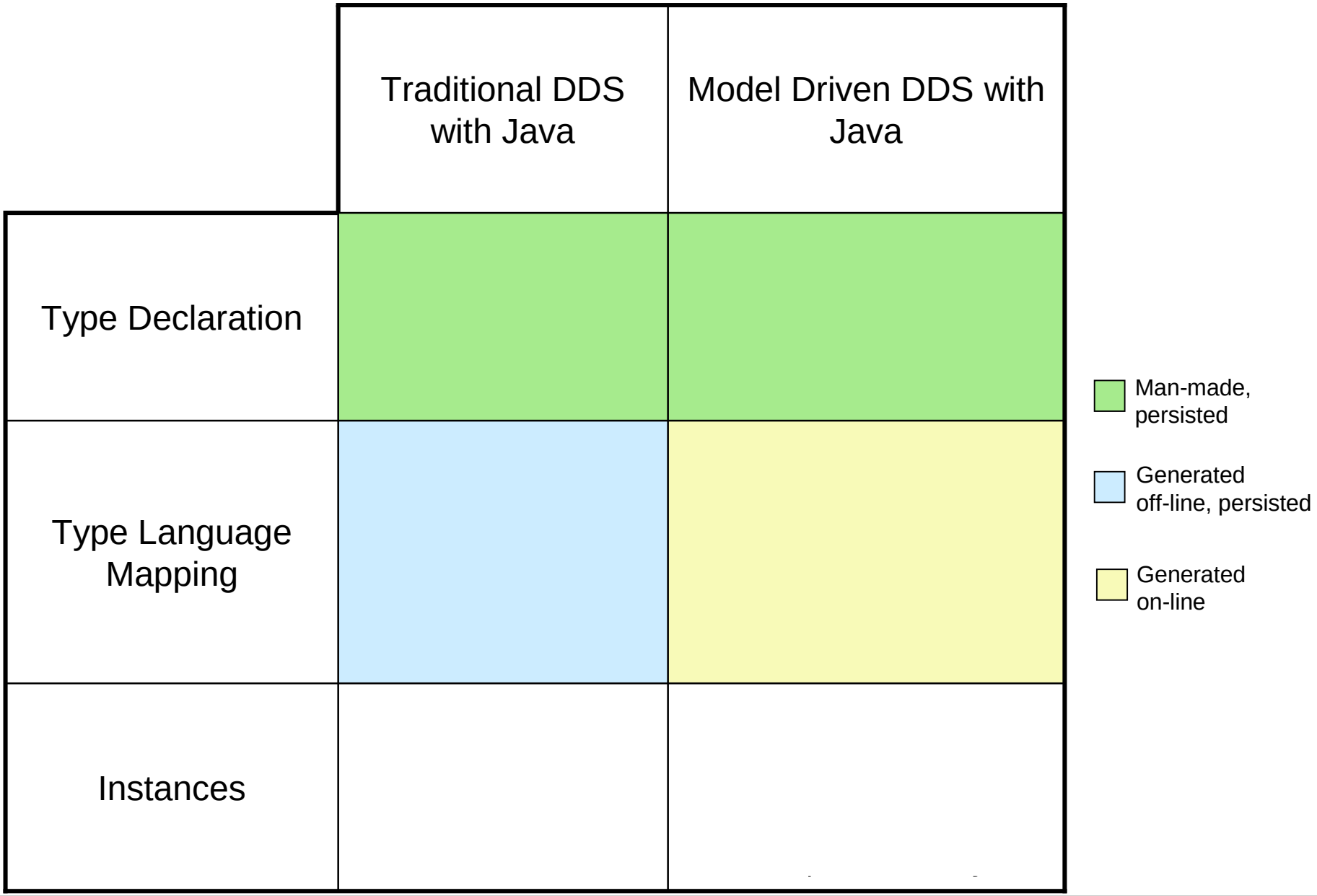
# Comparison to traditional DDS

| | Traditional DDS with Java | Model Driven DDS with Java |
|---|---|---|
| Type Declaration | | |
| Type Language Mapping | | |
| Instances | | |
| Support Classes | | |

Man-made, persisted

Generated off-line, persisted

Generated on-line


Traditional DDS with Java
Model Driven DDS with Java
Type Declaration
Type Language Mapping
Instances
Man-made, persisted
Generated off-line, persisted
Generated on-line

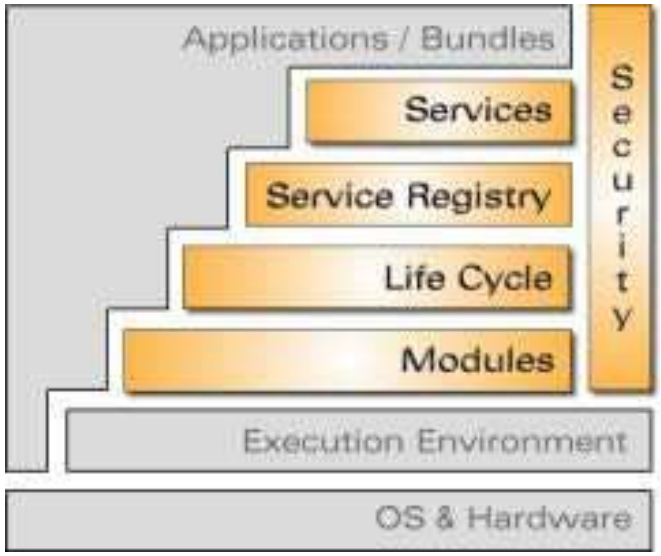
# The MDDS Manager

- Simple API
  - **`MDataReader createDataReader(MSubscriberInterface iface, MListener listener)`**
  - **`MDataWriter createDataWriter(MPublisherInterface iface)`**
- Resource sharing
  - Domain Participants, Subscribers and Publishers are shared wherever it is possible
- The MDDS Manager is implemented as an OSGi service

# Prerequisite: OSGi

# Open Services Gateway initiative

- Dynamic module system for the Java Programming Language
- Open standard – Open Architecture
- Component model, binary re-use possible
- Components are called *bundles*
- Loose coupling
- SOA within a Java Virtual Machine




- www.osgi.org

# MDDS and OSGi


- Again: The MDDS Manager is an OSGi service (implemented in a bundle)
- The component using an MDDS interface lives in another bundle, and asks the OSGi runtime for the MDDS Manager service

- Using OSGi we can reload a data model in a running system
- …except we haven't found a way to unregister a type. Vendor's limitation or spec requirement?

# Benefits of Model Driven DDS

- Complete description of an interface – domain id, QoS settings etc. – is kept in one place without a man in the middle
- Tuning QoS is easy
  - If types are kept unchanged this can be done in a running program, works beautifully with OSGi.
- IDL files and nice-to-look-at documentation can be generated from the models with little effort

# Todo’s

- Extend to other vendors; OpenSplice, others?
- Implement remaining IDL constructs
  - Unions, Value types
- Add remaining QoS policies to the QoS Model
- Wait-based data access
- Unregister types?
- Extend the editor
  - Context menus (e.g. Goto Type Definition, Goto Publisher QoS)
  - Refactoring
  - Validation

Demo

# Demo

- Using Saab's HMI Framework we have integrated a USB Missile Launcher
- Pre-conditions: a hacked launcher and an IDL for interfacing it

# Launcher IDL

(Namespace: com::saabgroup::enterprisebus::launcher)

```
struct LauncherStatus {
    ModuleId moduleId; //@key
    boolean allocated;
    boolean firing;
    boolean video;
    LauncherConfigurationVideo videoConfiguration;
};

struct LauncherConfigurationVideo {
    string host;
    short port;
    short maxFps;
}

struct LauncherControl {
    ModuleId moduleId;
    LauncherCommand command;
    LauncherCommandMove move;
    LauncherCommandSetVideo setVideo;
};
```

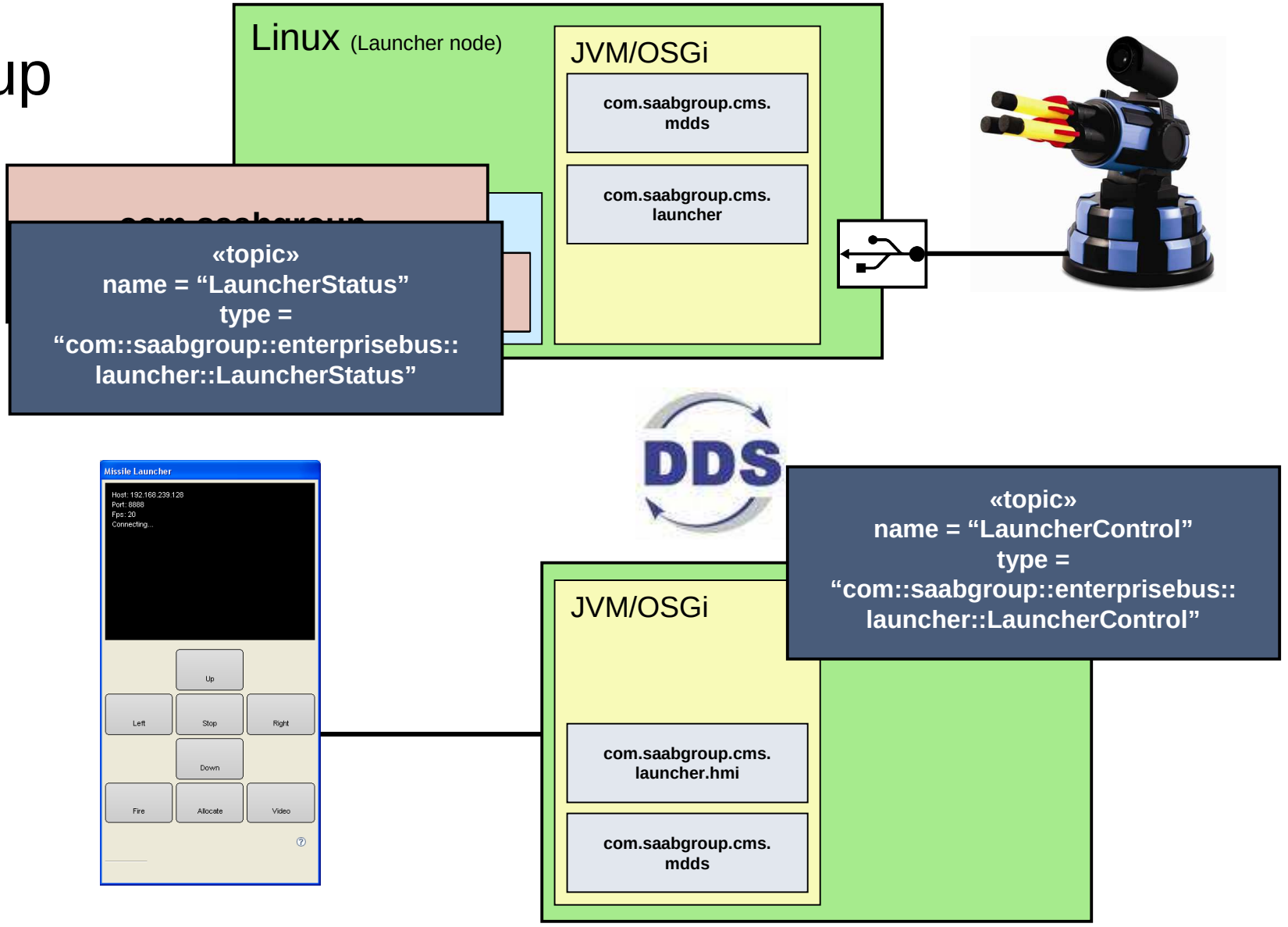
```
enum LauncherCommand {
    MOVE,
    FIRE,
    STOP,
    SET_VIDEO,
    ALLOCATE,
    DEALLOCATE
};

enum LauncherDirection {
    UP,
    DOWN,
    LEFT,
    RIGHT
};

struct LauncherCommandMove {
    LauncherDirection direction;
};

struct LauncherCommandSetVideo {
    boolean active;
};
```

# Setup




www.saabgroup.com