DDS in a Component-Based Architecture

Protima Banerjee
protima.x.banerjee@lmco.com

Lockheed-Martin Mission Systems & Training

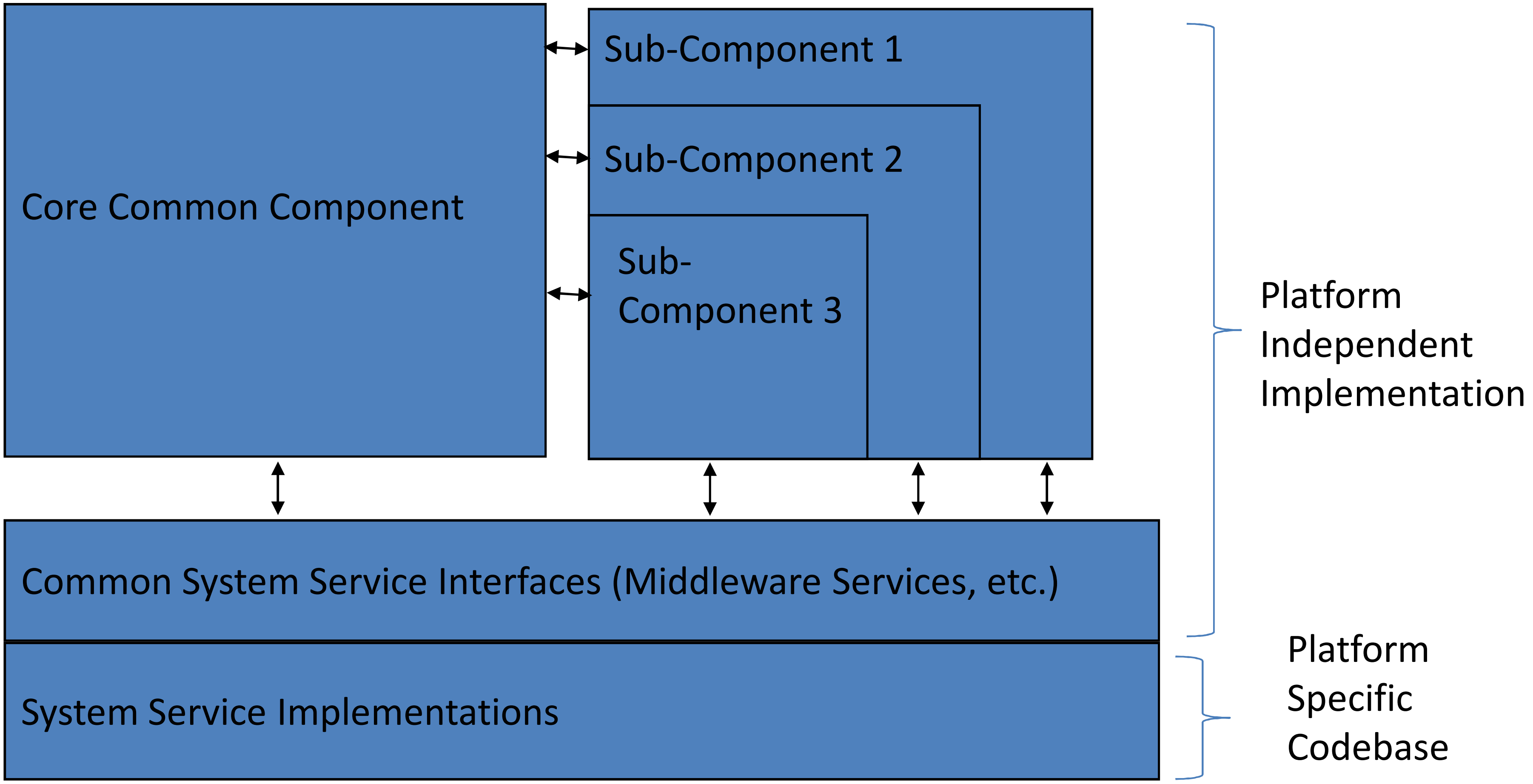
March 2013

Session #

Motivation: Component-Based Architectures

- A Component-based architecture consists of:
  - A set of common core software components
  - A common data model showing data shared between components
  - A set of common service interfaces used by components to access platform-specific functions
- A Component-based architecture allows for:
  - An open development model that allows many developers to provide combat system products
  - Incremental capability development
  - Incremental capability upgrades
  - Rapid technology insertion and more effective transition of R&D products by a system integrator
- Example: Navy’s Software Product Line Architecture

High Level Component Model


Core Common Component
Sub-Component 1
Sub-Component 2
Sub-Component 3
Common System Service Interfaces (Middleware Services, etc.)
System Service Implementations
Platform Independent Implementation
Platform Specific Codebase

Components can be composed of sub-components which may be auto-nomous or semi-autonomous.

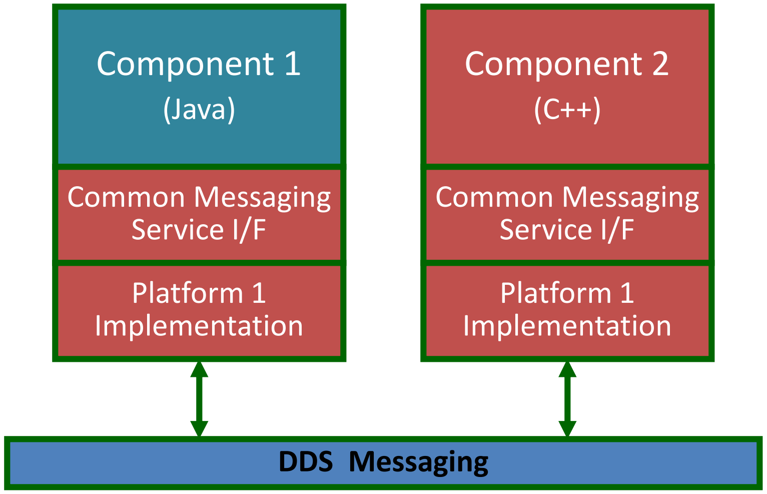
Architectural Considerations

- Component-based systems are designed with these goals:
  - Standards-based, components decoupled from one another
  - Make use of COTS where applicable
  - Robust to expected failures
  - Provide redundancy where applicable
- Inter-component communication is key
  - Common Data Model across components provides data structures
  - Common Messaging Service Interface provides the messaging mechanism
  - Examples of a Messaging Service implementation may be:
    - The OMG Data Distribution Service (DDS) a middleware standard from the Object Management Group (OMG)
    - Java Messaging Service (JMS)
    - Common Object Request Broker Architecture (CORBA)

Messaging is fundamental to Component-based systems!

Components can be.

Messaging Between Components



Component 1 (Java)

Common Messaging Service I/F

Platform 1 Implementation

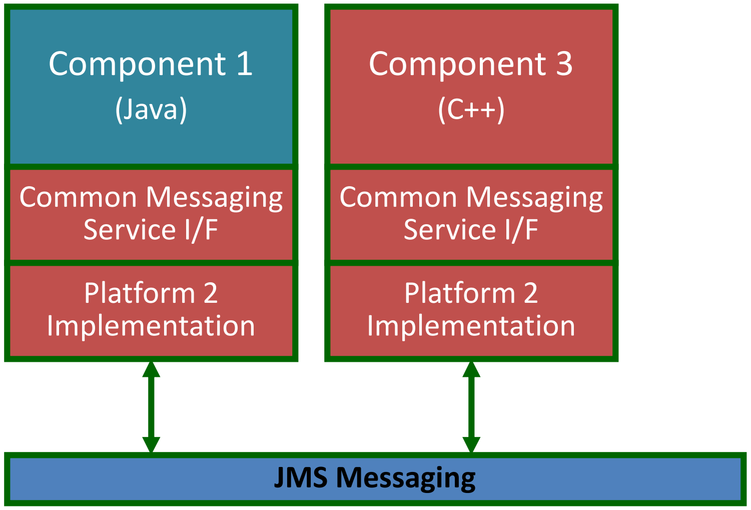
Component 2 (C++)

Common Messaging Service I/F

Platform 1 Implementation

DDS Messaging

On Platform 1, Component 1 communicates with Component 2 using DDS as the underlying transport.



Component 1 (Java)

Common Messaging Service I/F

Platform 2 Implementation

Component 3 (C++)

Common Messaging Service I/F

Platform 2 Implementation

JMS Messaging

On Platform 2, Component 1 communicates with Component 3 using JMS as the underlying transport.

Component does not need to be aware of underlying transport, but the underlying transport can provide features to facilitate component interactions.

Messaging in a Component Based System

- Propose that there are four aspects that characterize inter-component communications:
  - Message Data Definition – Syntactic Definition
    - Shows data field names, data types
  - Message Data Definition – Semantic Defintion
    - Shows data relationships including inheritance, aggregation, etc.
  - Messaging Behavior – Syntactic
    - Shows which messages are sent and received by which components
    - Shows which transports are used to communication messages
  - Messaging Behavior – Semantic
    - Shows message transmission characteristics
    - Provides knowledge about the message flows in a system

Not all middleware implementations provide all of these!

Why is Messaging Behavior Important?

- A system integrator needs to understand all four characteristics of a message exchange:
  - Data model and data relationships
  - Physical aspects of messaging behavior:
    - Eg. What transports and interfaces are being used for the exchange?
    - What network resources are being used and how intensively?
  - Semantic aspects of messaging behavior:
    - Eg. Is the message transmitted reliably or best-effort?
    - Does the message have a periodicity?
    - Will old messages be re-transmitted to new subscribers that come up after the system has been running?
    - Is an error condition indicated if a message is not received after a certain amount of time?

A robust messaging service should include parameters that define behavior.

The Data Distribution Service (DDS)

- What is DDS?
  - DDS is an OMG standard for decentralized publish / subscribe messaging
    - Recommended by the Naval Open Architecture Computing Environment (OACE) and Net-Centric Solutions for Interoperability (NESI)
- DDS provides the following key capabilities for component-based development:
  - Interoperability across vendors
  - Data domains and partitions to segregate specific component-interactions
  - Support for a variety of underlying transports to allow network-level tuning
  - Upcoming Security features to support secure interactions between specific components

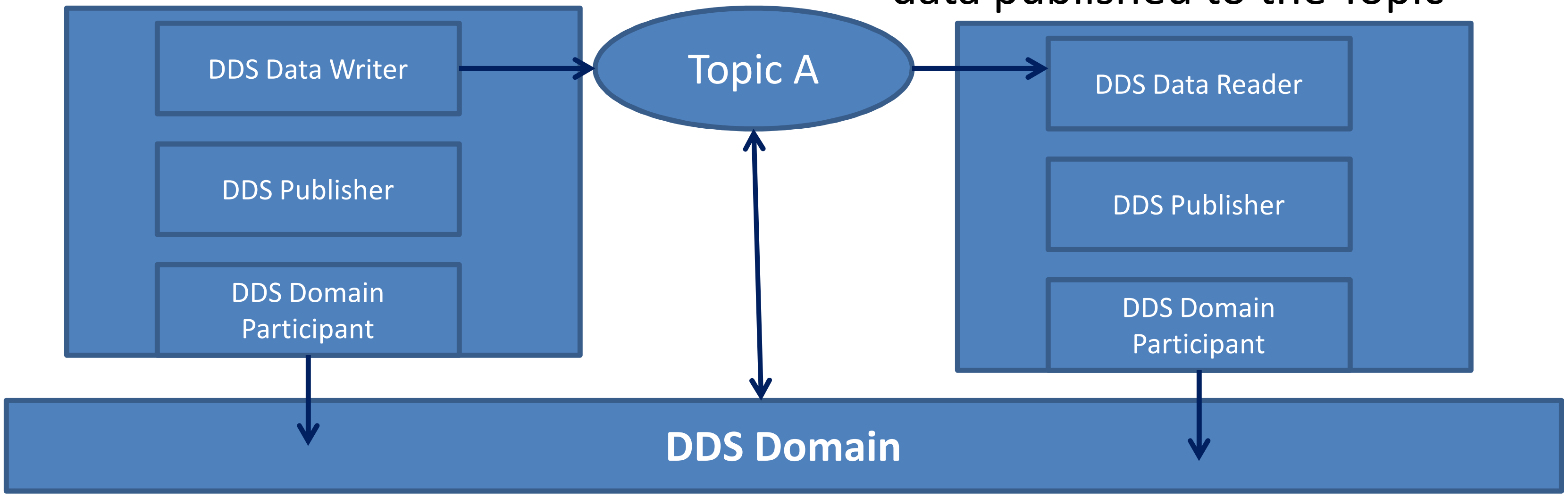
The Data Distribution Service (DDS)

- Our focus here: DDS and Messaging Behavior
  - DDS provides the system integrator the ability to control messaging behavior between components at a detailed level.
  - DDS is unique in this regard.
- DDS Quality of Service
  - DDS provides twenty-six Quality of Service (QoS) parameters assignable to all communications entities
  - Semantics of message behavior can be defined in a standards-based way
  - Is well-suited for providing the under-lying transport for a component-based architecture for this reason

Background: DDS Topic Based Publish/ Subscribe

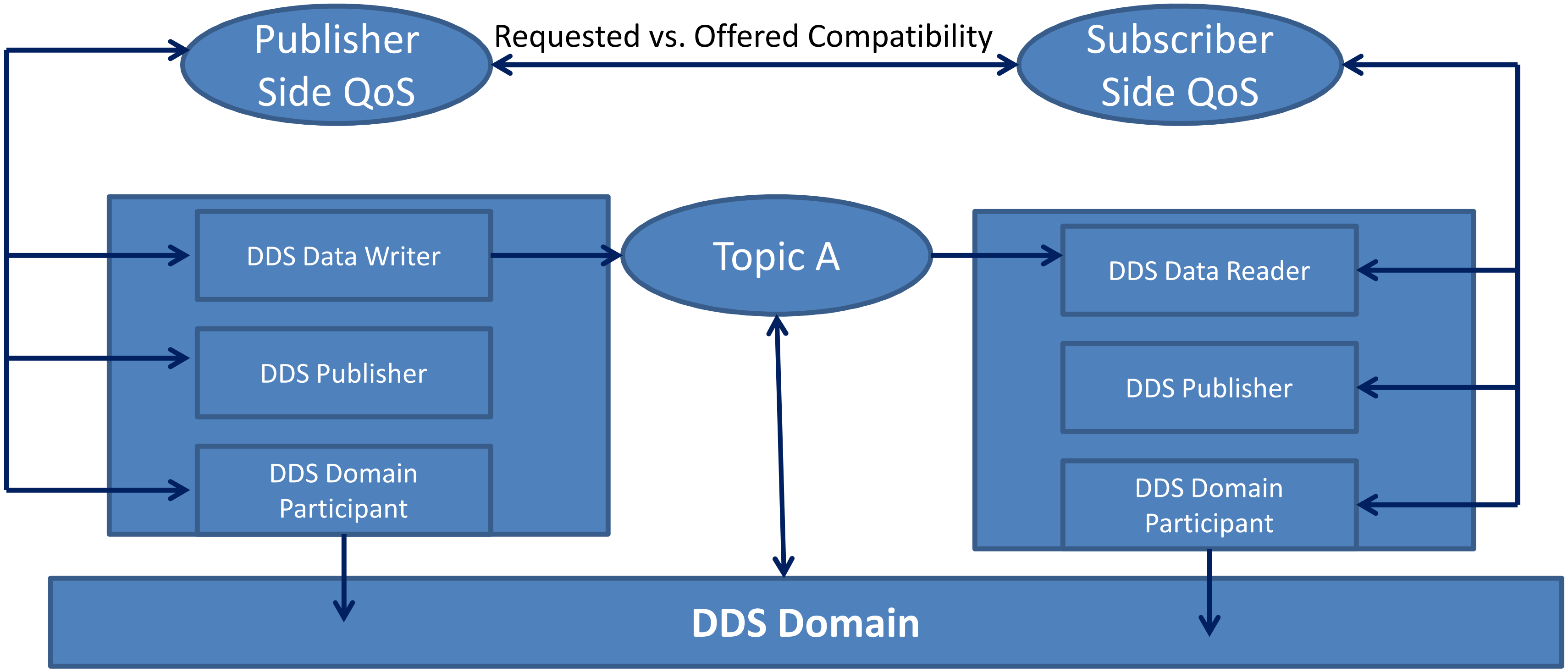
- For a given message exchange, the publishing application creates a Domain Participant, Publisher and Data Writer
- The Data Writer is bound to a Topic
- The Data Writer writes data to a Topic

- The subscribing application creates a Domain Participant, Subscriber and Data Reader
- The Data Reader is bound to a Topic
- The Data Reader receives the data published to the Topic



Background: DDS Quality of Service

- QoS Policies assignable to all entities in the DDS message exchange
- Allows for a very granular definition of messaging behavior for specific application threads and timelines



Behavioral DDS QoS Policies that Support Component Interactions

- Deadline
  - At least one message must be received within a specified time period
- Destination Order
  - Received messages can be delivered either by send or receipt timestamp order
- Durability
  - Messages are re-transmitted to late-joining subscribers
- History
  - Up to N (possibly infinite) messages are retained in a local queue.
- Lifespan
  - The “shelf-life” of a message. Old messages are discarded by the system.
- Presentation
  - Message ordering by a logical sequence number. Topics can be grouped and the ordering occur within a logical topic group.
- Reliability
  - Should messages be sent reliably or best-effort?
- Time-Based Filter
  - Can only a sub-set of messages within a specified time window be considered useful?
- User Data, Group Data, Topic Data
  - Allow publisher/subscriber authentication policies to be put in place

Challenge: Managing Messaging Behavior

- Challenges for the system integrator:
  - Messaging behavior is likely to change frequently over time as systems evolve
    - Behavior may change even when message structures stay consistent
  - The same message may participate in multiple exchanges, each of which has a unique behavior
    - For example, a message may have both reliable and unreliable subscribers.
  - Messaging behavior should be communicated in a concise way to component developers, in a manner than can be interpreted easily into code.
    - Automated generation of QoS XML Profiles
  - Messaging behavior must be consistent between components otherwise communications at the system level will fail

Calls for Messaging Behavioral Model to be overlaid on a Messaging Data Model.

Approach: Add to Existing UML/SysML Models

- Benefits:
  - Behavior and data could be captured in a single repository
  - Accessible using a single toolset

- Considerations:
  - Existing UML/SysML paradigm may not be appropriate for messaging behavioral semantics
    - Explore DDS Profile for UML
  - Extraction of the behavioral semantics is also required
    - Ideally, would like the behavioral model to produce an XML configuration that could then be provided to software
    - Code generation would also be ideal
  - Configuration management
    - Easy for messaging behaviors to get out of sync with the model unless the process is tightly controlled

DDS Profile for UML is a standards-based approach to modeling DDS QoS.

DDS in Component Based Systems: Some Other Thoughts

- Testability is critical to message exchanges between components:
  - Vendor-specific tools are currently available:
    - For example, RTI provides a DDS Monitoring, Analyzing and Recording capabilities
    - However, integration tools that are not tied to a specific DDS implementation are desirable
  - Trouble-shooting message exchange problems between vendors limited to tools at the wire protocol level
    - Wireshark DDS Dissector
    - Wire level data is difficult to analyze; difficult to collect and retain for long periods of time

Focus on this area would have a huge return on investment for system integrators.

Conclusion

- A Component-based architectures allow for:
  - Incremental capability development
  - Incremental capability upgrades
  - Rapid technology insertion
- DDS Middleware is critical to component-based architectures
  - Provides a standardized means of defining both message data types as well as message exchange behaviors
  - Model-based definition of message behavior is a current challenge
  - System integrators would benefit from industry focus in the area of vendor-independent DDS tools

Questions?