The Data Distribution Service

The Communication Middleware Fabric for Scalable and Extensible Systems-of-Systems

Angelo Corsaro, PhD
angelo@icorsaro.net
PrismTech

Douglas C. Schmidt, PhD
d.schmidt@vanderbilt.edu
Vanderbilt University

1 Introduction

During the past several decades techniques and technologies have emerged to design and implement distributed systems effectively. A remaining challenge, however, to devising techniques and technologies that will help design and implement SoSs. SoSs present some unique challenges when compared to traditional systems since their scale, heterogeneity, extensibility, and evolvability requirements are unprecedented compared to traditional systems [9].

For instance, in Systems-of-Systems (SoS), such as the one depicted in Figure 1, the compu-

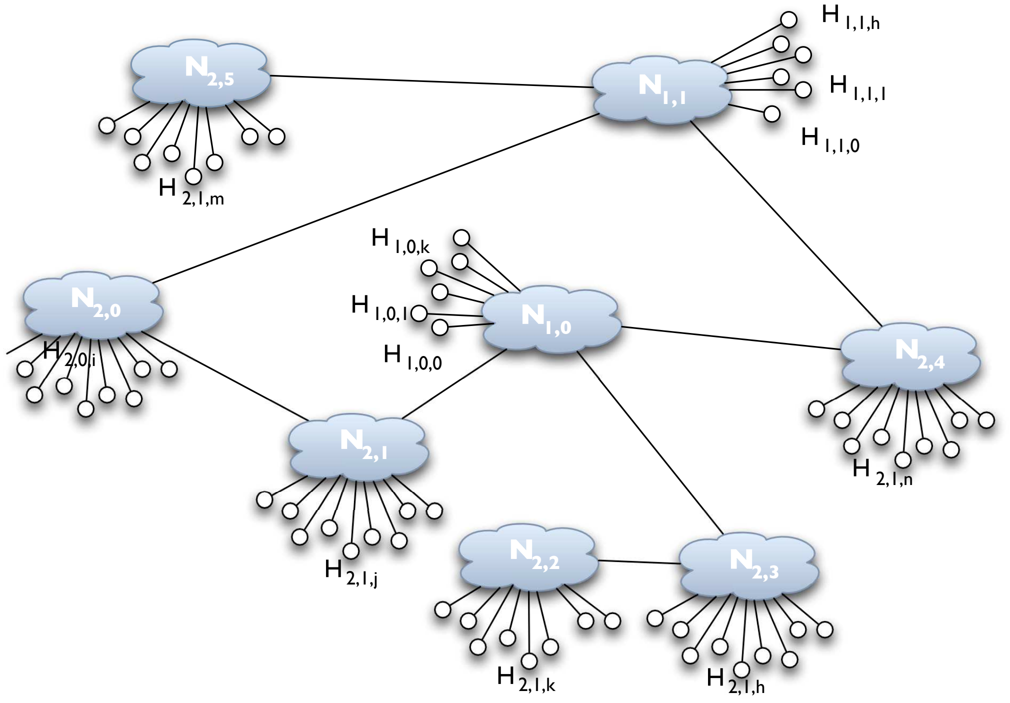

Figure 1: A System-of-Systems Architecture.

tational and communication resources involved are highly heterogeneous, which yields situations where high-end systems connected to high-speed networks must cooperate with embedded devices or resource-constrained edge systems connected over bandwidth-limited links. Moreover, in SoS it is common to find multiple administrative entities that manage the different parts, so upgrading the system must be incremental and never require a full redeployment of the whole SoS. In addition, SoS are often characterized by high degrees of dynamism and thus must be enable subsystems and devices dynamically joining and leaving the federation of system elements.

The Object Management Group (OMG) Data Distribution Service for Real-Time Systems (DDS) [5] is a standard for data-centric Publish/Subscribe (P/S) introduced in 2004 to address the challenges faced by important mission-critical systems and systems-of-systems. As described in the reminder of this Chapter, DDS addresses all the key challenges posed by SoS outlined above. As a result, it provides the most natural choice as the communication middleware fabrice for developing scalable and extensible SoS.

Since its inception DDS has experienced a swift adoption in several different domains. The reason for this successful adoption stems largely from its following characteristics:

1. DDS has been designed to scale up and down, allowing deployments that range from resource-constrained embedded systems to large-scale systems-of-systems.

2. DDS is equipped with a powerful set of QoS policies that allow applications fine-grain control over key data distribution properties, such as data availability, timeliness, resource consumption, and usage.

3. DDS is equipped with a powerful type system that provides end-to-end type safety for built-in and user-defined types, as well as type-safe extensibility.

As a result of these characteristics, the OMG DDS standard is the most advanced data distribution technology and is a key building-block for many existing and upcoming SoSs.

The remainder of this chapter presents an in-depth introduction to DDS, as well a set of guidelines on how to apply this technology to architect scalable and efficient SoSs. The chapter concludes with a preview of forthcoming DDS innovations.

2 Overview of the OMG Data Distribution Service (DDS)

P/S is a paradigm for one-to-many communication that provides anonymous, decoupled, and asynchronous communication between producers of data–the publishers–and consumers of data–the subscribers. This paradigm is at the foundation of many technologies used today to develop and integrate distributed applications (such as social application, e-services, financial trading, etc.), while ensure the composed parts of the applications remain loosely coupled and independently evolvable.

Different implementations of the P/S abstraction have emerged to support the needs of different application domains. DDS [5] is an OMG P/S standard that enables scalable, real-time, dependable and high performance data exchanges between publishers and subscribers. DDS addresses the needs of mission- and business-critical applications, such as, financial trading, air traffic control and management, defense, aerospace, smart grids, and complex supervisory and telemetry systems. That key challenges addressed by DDS are to provide a P/S technology in which data exchanged between publishers and subscribers are:

- **Real-time**, meaning that the right information is delivered at the right place at the right time–all the time. Failing to deliver key information within the required deadlines can lead to life-, mission- or business-threatening situations. For instance in a financial trading 1ms can make the difference between losing or gaining $1M. Likewise, in a supervisory applications for power-grids, failing to meet deadlines under an overload situation could lead to severe blackout, such as the one experienced by the northeastern US and Canada in 2003 [1].

- **Dependable**, thus ensuring availability, reliability, safety and integrity in spite of hardware and software failures. For instance, the lives of thousands of air travelers depend on the reliable functioning of an air traffic control and management system. These systems must ensure 99.999% availability and ensure that critical data is delivered reliably, regardless of experienced failures.

- **High-performance**, which necessitates the ability to distribute very high volumes of data with very low latencies. As an example, financial auto-trading applications must handle millions of messages per second, each delivered relialy with minimal latency, e.g., on the order of tens of microseconds.

2.1 Components in the DDS Standard

The components in the OMG DDS standards family are shown in Figure 2 and consist of the DDS v1.2 API [5] and the Data Distribution Service Interoperability Wire Protocol (DDSI) [6]. The DDS API standard ensures source code portability across different vendor implementations, while the DDSI Standard ensures on the wire interoperability across DDS implementations from different vendors. The DDS API standard shown in Figure 2 also defines several profiles that enhance real-time P/S with content filtering, persistence, automatic fail-over, and transparent integration into object oriented languages.

The DDS standard was formally adopted by the OMG in 2004. It quickly became the established P/S technology for distributing high volumes of data dependably and with predictable low latencies in applications such as radar processors, flying and land drones, combat management systems, air traffic control and management, high performance telemetry, supervisory control and data acquisition systems, and automated stocks and options trading. Along with wide commercial adoption, the DDS standard has been mandated as the technology for real-time data distribution by organization worldwide, including the US Navy, the Department of Defence (DoD) Information-Technology Standards Registry (DISR) the UK Mistry of Defence (MoD), the Military Vehicle Association (MILVA), and EUROCAE–the European organization that regulates standards in Air Traffic Control and Management.

2.2 Key DDS Architectural Concepts and Entities

Below we summarize the key architectural concepts and entities in DDS.

2.2.1 Global Data Space

The key abstraction at the foundation of DDS is a fully distributed Global Data Space (GDS) (see Figure 3). The DDS specification requires a fully distributed implementation of the GDS to avoid single points of failure or single points of contention. Publishers and Subscribers can join or leave the GDS at any point in time as they are dynamically discovered. The dynamic discovery of Publisher and Subscribers is performed by the GDS and does not rely on any kind of centralized

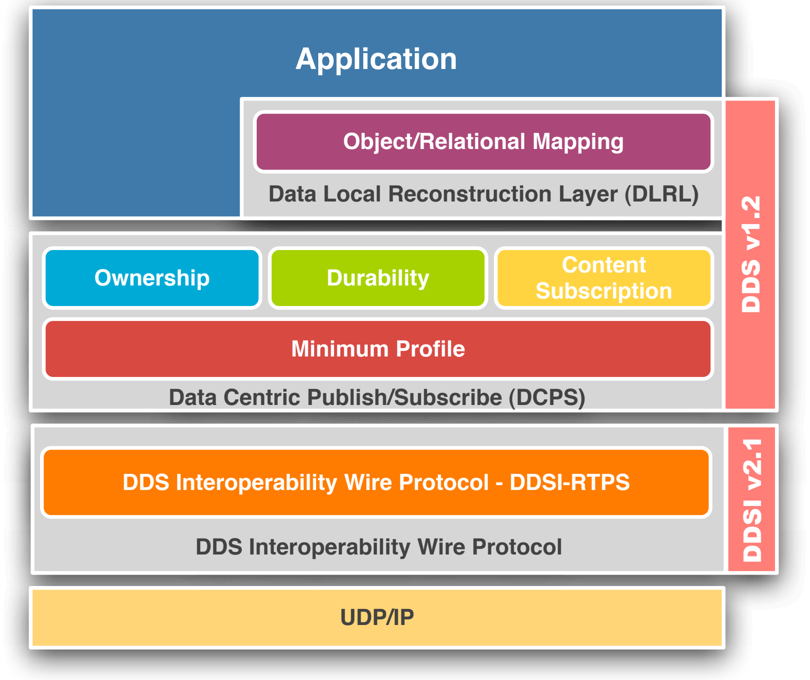
Application
Object/Relational Mapping
Data Local Reconstruction Layer (DLRL)
Ownership
Durability
Content Subscription
Minimum Profile
Data Centric Publish/Subscribe (DCPS)
DDS v1.2
DDS Interoperability Wire Protocol - DDSI-RTPS
DDS Interoperability Wire Protocol
DDSI v2.1
UDP/IP

Figure 2: The DDS Standard.

registry, such as those found in other P/S technologies like the Java Message Service (JMS) [8]. The GDS also discovers application defined data types and propagates them as part of the discovery process.

Since DDS provides a GDS equipped with dynamic discovery, there is no need for applications to configure anything explicitly when a system is deployed. Applications will be automatically discovered and data will begin to flow. Moreover, since the GDS is fully distributed the crash of one server will not induce unknown consequences on the system availability, i.e., in DDS there is no single point of failure and the system as a whole will continue to run even if applications crash/restart or connect/disconnect.

2.2.2 Topic

The information that populates the GDS is defined by means of DDS *Topics*. A *topic* is identified by a unique name, a data type, and a collection of Quality of Service (QoS) policies. The unique name provides a mean of uniquely referring to given topics, the data type defines the type of the stream of data, and the QoS captures all the non-functional aspect of the information, such as its temporal properties or its availability.

Topic types. DDS Topics can be specified using several different syntaxes, such as Interface Definition Language (IDL), eXtensible Markup Langauge (XML), Unified Modeling Langauge (UML), and annotated Java. For instance, Listing 1 shows a type declaration for an hypothetical temper-

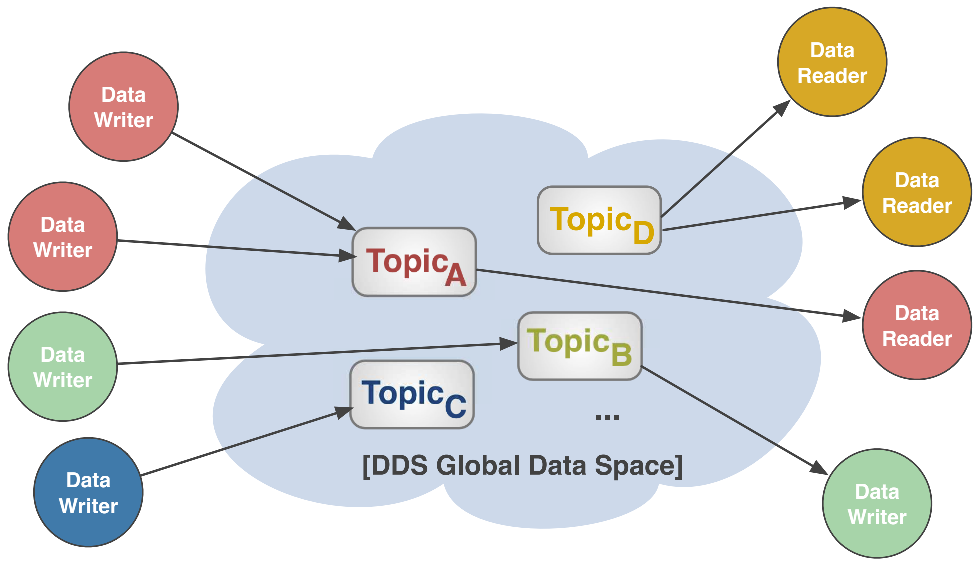

Figure 3: The DDS Global Data Space.

ature sensor topic-type. Some of the attributes of a topic-type can be marked as representing the key of the type.

Listing 1: Topic type declaration for an hypothetical temperature sensor

```
struct TempSensorType {
  @Key
  short id;
  float temp;
  float hum;
};
```

The key allows DDS to deal with specific instances of a given topic. For instance, the topic type declaration in Listing 1 defines the `id` attributes as being the key of the type. Each unique `id` value therefore identifies a specific topic instance for which DDS will manage the entire life-cycle, which allows an application to identify the specific source of data, such as the specific physical sensor whose `id=5`. Figure 4 provides a visual representation of the relationship existing between a topic, its instances, and the associated data streams.

Topic QoS. The Topic QoS provides a mechanism to express relevant non-functional properties of a topic. Section 4 presents a detailed description of the DDS QoS model, but at this point we simply mention the ability of defining QoS for topics to capture the key non-functional invariant of the system and make them explicit and visible.

Content Filters. DDS supports defining content filters over a specific topic. These content filters are defined by instantiating a `ContentFilteredTopic` for an existing topic and providing a filter expression. The filter expression follows the same syntax of the WHERE clause on a SQL statement and can operate on any of the topic type attributes. For instance, a filter expression for a temperature sensor topic could be `"id = 101 AND (temp > 35 OR hum > 65)"`.

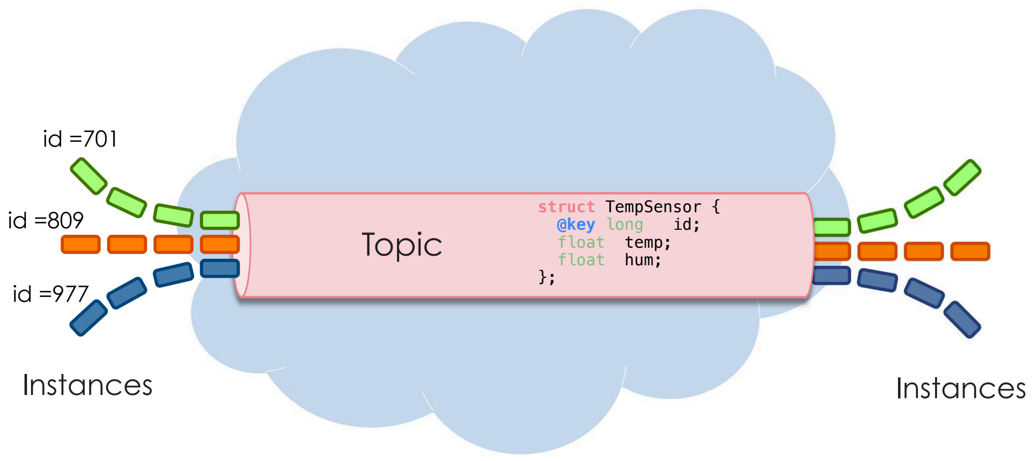

Figure 4: Topic Instances and Data Streams.

2.2.3 DataWriters and DataReaders

Since a topic defines the subjects produced and consumed, DDS provides two abstractions for writing and reading these topics: *DataWriters*s and *DataReader*s, respectively. Both DataReaders and DataWriters are strongly typed and are defined for a specific topic and topic type.

2.2.4 Publishers, Subscribers and Partitions

DDS also defines *Publishers* and *Subscribers*, which group together sets of DataReaders and DataWriters and perform some coordinated actions over them, as well as manage the communication session. DDS also supports the concept of a *Partition* which can be used to organize data flows within a GDS to decompose un-related sets of topics.

2.3 Example of Applying DDS

Now that we presented the key architectural concepts and entities in DDS, we will show the anatomy of a simple DDS application. Listing 2 shows the steps required to join a DDS domain, to define a publisher and a topic on the given domain, and then create a data writer and publish a sample.

Listing 2: A simple application writing DDS samples.

```

// -- Joining Domain by creating a DomainParticipant
int32_t domainID = 0;
DomainParticipant dp(domainID);

// -- Get write acces to a domain by creating a Publisher
Publisher pub(dp);

// -- Register Topic
Topic<TempSensor> topic(dp, "TemSensorTopic");

// -- Create DataWriter
DataWriter<TempSensor> dw(pub,topic);
```

```

// -- Writer a sample
dw <<  TempSensor(701, 25, 67);
```

Note that no QoS has been specified for any of the DDS entities defined in this code fragment, so the behavior will be the default QoS.

Listing 3: A simple application reading DDS samples.

```
// -- Joing Domain by creating a DomainParticipant
int32_t domainID = 0;
DomainParticipant dp(domainID);

// -- Get write acces to a domain by creating a Subscriber
Subscriber sub(dp);

// -- Register Topic
Topic<TempSensor> topic(dp, "TemSensorTopic");

// -- Create DataReader
DataReader<TempSensor> dr(pub,topic);


std::vector<TempSensor> data(MAX_LEN);
std::vector<SampleInfo> info(MAX_LEN);
// -- Rear a samples
dr.read(data.begin(), info.being(), MAX_LEN)
```

Listing 3 shows the steps required to read the data samples published on a given topic. This code fragment shows the application proactively reading data. DDS also provides a notification mechanism that informs a datareader via a callback when new data is available, as well as an operation for waiting for new data to become available.

3 The DDS Type System

Strong typing plays a key role in developing software systems that are easier to extend, less expensive to maintain, and in which runtime errors are limited to those computational aspects that either cannot be decided at compile-time or that cannot be detected at compile-time by the type system. Since DDS was designed to target mission- and business-critical systems where safety, extensibility, and maintainability are critically important, DDS adopted a strong and statically typed system to define type properties of a topic. This section provides an overview of the DDS type system, as defined in [7].

3.1 Structural Polymorphic Types System

DDS provides a *polymorphic structural type system* [3, 2, 7], which means that the type system not only supports polymorphism, but also bases its sub-typing on the *structure* of a type, as opposed than its *name*. For example, consider the types declared in Listing 4.

Listing 4: Nominal vs. Structural Subtypes

```
struct Coord2D {
  int x;
  int y;
};

struct Coord3d : Coord2D {
  int z;
};

struct Coord {
  int x;
  int y;
  int z;
};
```

In a traditional nominal polymorphic type system, the `Coord3D` would be a subtype of `Coord2D`, which would be expressed by writing $Coord3D <: Coord2D$. In a nominal type system, however, there would be no relationship between the `Coord2D`/`Coord3D` with the type `Coord`. Conversely, in a polymorphic structural type system like DDS the type `Coord` is a subtype of the type `Coord2D`—thus $Coord <: Coord2D$ and it is structurally the same type as the type `Coord3D`.

The main advantage of a polymorphic structural type system over nominal type systems is that the former considers the structure of the type as opposed to the name to determine sub-types relationships. As a result, polymorphic structural types systems are more flexible and well-suited for SoS. In particular, types in SoS often need to evolve incrementally to provide a new functionality to most subsystems and systems, *without* requiring a complete redeploy of the entire SoS.

In the example above the type `Coord` was a monotonic extension of the type `Coord2D` since it was added some new attributes "at the end." The DDS type system can handle both attribute reording and attribute removal with equal alacrity.

3.2 Type Annotations

The DDS type systems supports an annotation system very similar to that available in Java [4]. It also defines a set of built-in annotations that can be used to control the extensibility of a type, as well as the properties of the attributes of a given type. Some important built-in annotations are described below:

- The `@ID` annotation can be used to assign a global unique ID to the data members of a type. This ID is used to deal efficiently with attributes reordering.

- The `@Key` annotation can be used to identify the type members that constitute the key of the topic type.

- The `@Optional` annotations can be used to express that an attribute is optional and might not be set by the sender. DDS provides specific accessors for optional attributes that can be used to safely check whether the attribute is set or not. In addition, to save bandwidth, DDS will not send optional attributes for which a value has not been provided.

- The `@Shared` annotation can be used to specify that the attribute must be referenced through a pointer. This annotations helps avoid situations when large data-structures (such as images

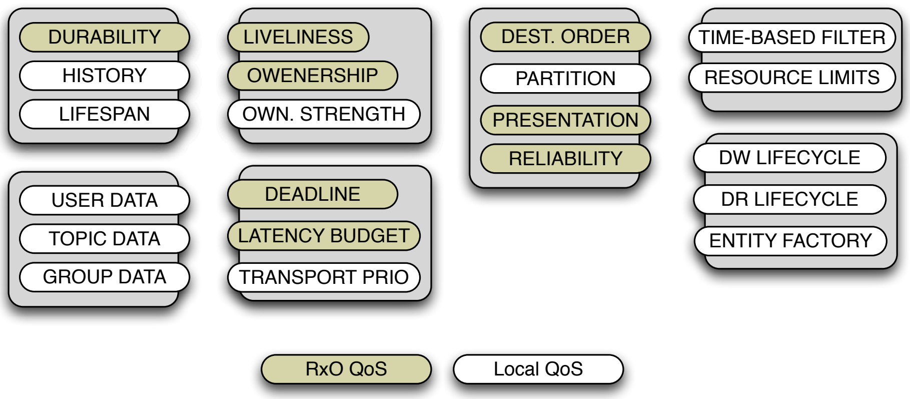

Figure 5: DDS QoS Policies.

or large arrays) would be allocated contiguously with the topic type, which may be undesirable in resource-contrained embedded systems.

- The `@Extensibility` annotation can be used to control the level of extensibility allowed for a given topic type. The possible values for this annotation are (1) `Final` to express the fact that the type is sealed and cannot be evolved – as a result this type cannot be substituted by any other type that is structurally its subtype, (2) `Extensible` to express that only monotonic extensibility should be considered for a type, and (3) `mutable` to express that the most generic structural subtype rules for a type should be applied when considering subtypes.

In summary, the DDS type system provides all the advantages of a strongly-typed system, together with the flexibility of structural type systems. This combinations supports key requirements of a SoS since it preserves types end-to-end, while providing type-safe extensibility and incremental system evolution.

4 The DDS QoS Model

DDS provides applications with explicit control over a wide set of non-functional properties, such as data availability, data delivery, data timeliness and resource usage through a rich set of QoS policies – Figure 5 shows the full list of available QoS. The control provided by these policies over the key non-functional properties of data is important for traditional systems and indispensable for SoS. Each DDS entity (such as a topic, data reader, and data writer) can apply a subset of available QoS policies. The policies that control and end-to-end property are considered as part of the subscription matching. DDS uses a request vs. offered QoS matching approach, as shown in Figure **??** in which a data reader matches a data writer if and only if the QoS it is requesting for the given topic does not exceed (e.g., is no more stringent) than the QoS with which the data is produced by the data writer.

DDS subscriptions are matched against the topic type and name, as well as against the QoS being offered/requested by data writers and readers. This DDS matching mechanism ensures that

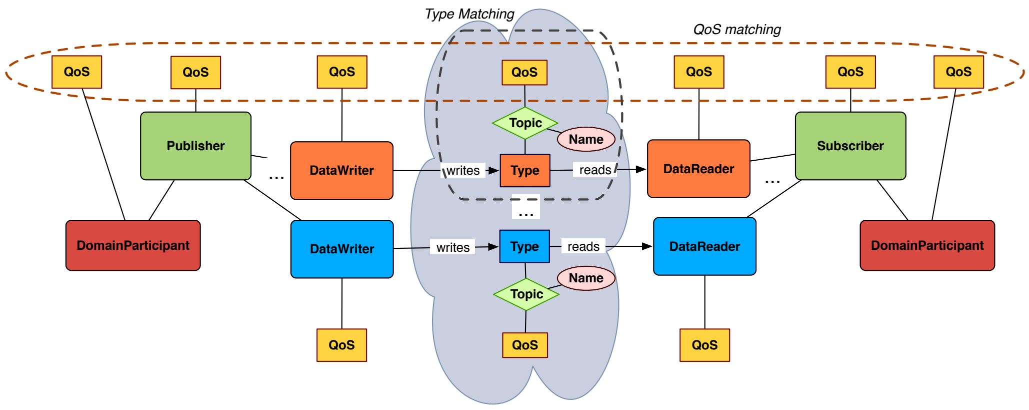

Figure 6: DDS Request vs. Offered QoS Model.

(1) types are preserved end-to-end due to the topic type matching and (2) end-to-end QoS invariants are also preserved.

The reminder of this section describes the most important QoS policies in DDS.

4.1 Data availability

DDS provides the following QoS policies that control the availability of data to domain participants:

- The `DURABILITY` QoS policy controls the lifetime of the data written to the global data space in a DDS domain. Supported durability levels include (1) `VOLATILE`, which specifies that once data is published it is not maintained by DDS for delivery to late joining applications, (2) `TRANSIENT_LOCAL`, which specifies that publishers store data locally so that late joining subscribers get the last published item if a publisher is still alive, (3) `TRANSIENT`, which ensures that the GDS maintains the information outside the local scope of any publishers for use by late joining subscribers, and (4) `PERSISTENT`, which ensures that the GDS stores the information persistently so to make it available to late joiners even after the shutdown and restart of the whole system. Durability is achieved by relying on a durability service whose properties are configured by means of the `DURABILITY_SERVICE` QoS of non-volatile topics.

- The `LIFESPAN` QoS policy controls the interval of time during which a data sample is valid. The default value is infinite, with alternative values being the time-span for which the data can be considered valid.

- The `HISTORY` QoS policy controls the number of data samples (i.e., subsequent writes of the same topic) that must be stored for readers or writers. Possible values are the last sample, the last n samples, or all samples.

These DDS data availability QoS policies decouple applications in time and space. They also enable these applications to cooperate in highly dynamic environments characterized by continuous joining and leaving of publisher/subscribers. Such properties are particularly relevant in SoS since they increase the decoupling of the component parts.

4.2 Data delivery

DDS provides the following QoS policies that control how data is delivered and how publishers can claim exclusive rights on data updates:

- The `PRESENTATION` QoS policy gives control on how changes to the information model are presented to subscribers. This QoS gives control on the ordering as well as the coherency of data updates. The scope at which it is applied is defined by the access scope, which can be one of `INSTANCE`, `TOPIC`, or `GROUP` level.
- The `RELIABILITY` QoS policy controls the level of reliability associated with data diffusion. Possible choices are `RELIABLE` and `BEST_EFFORT` distribution.
- The `PARTITION` QoS policy gives control over the association between DDS partitions (represented by a string name) and a specific instance of a publisher/subscriber. This association provides DDS implementations with an abstraction that allow to segregate traffic generated by different partitions, thereby improving overall system scalability and performance.
- The `DESTINATION_ORDER` QoS policy controls the order of changes made by publishers to some instance of a given topic. DDS allows the ordering of different changes according to source or destination timestamps.
- The `OWNERSHIP` QoS policy controls which writer "owns" the write-access to a topic when there are multiple writers and ownership is `EXCLUSIVE`. Only the writer with the highest `OWNERSHIP_STRENGTH` can publish the data. If the `OWNERSHIP` QoS policy value is shared, multiple writers can concurrently update a topic. `OWNERSHIP` thus helps to manage replicated publishers of the same data.

These DDS data delivery QoS policies control the reliability and availability of data, thereby allowing the delivery of the right data to the right place at the right time. More elaborate ways of selecting the right data are offered by the DDS content-awareness profile, which allows applications to select information of interest based upon their content. These QoS policies are particularly useful in SoS since they can be used to finely tune how—and to whom—data is delivered, thus limiting not only the amount of resources used, but also minimizing the level of interference by independent data streams.

4.3 Data timeliness

DDS provides the following QoS policies to control the timeliness properties of distributed data:

- The `DEADLINE` QoS policy allows applications to define the maximum inter-arrival time for data. DDS can be configured to automatically notify applications when deadlines are missed.
- The `LATENCY_BUDGET` QoS policy provides a means for applications to inform DDS of the urgency associated with transmitted data. The latency budget specifies the time period within which DDS must distribute the information. This time period starts from the moment the data is written by a publisher until it is available in the subscribers data-cache ready for use by reader(s).

- The `TRANSPORT_PRIORITY` QoS policy allows applications to control the importance associated with a topic or with a topic instance, thus allowing a DDS implementation to prioritize more important data relative to less important data. These QoS policies help ensure that mission-critical information needed to reconstruct the shared operational picture is delivered in a timely manner.

These DDS data timeliness QoS policies provide control over the temporal properties of data. Such properties are particularly relevant in SoS since they can be used to define and control the temporal aspects of various subsystem data exchanges, while ensuring that bandwidth is exploited optimally.

4.4 Resources

DDS defines the following QoS policies to control the network and computing resources that are essential to meet data dissemination requirements:

- The `TIME_BASED_FILTER` QoS policy allows applications to specify the minimum inter-arrival time between data samples, thereby expressing their capability to consume information at a maximum rate. Samples that are produced at a faster pace are not delivered. This policy helps a DDS implementation optimize network bandwidth, memory, and processing power for subscribers that are connected over limited bandwidth networks or which have limited computing capabilities.

- The `RESOURCE_LIMITS` QoS policy allows applications to control the maximum available storage to hold topic instances and related number of historical samples DDSs QoS policies support the various elements and operating scenarios that constitute net-centric mission-critical information management. By controlling these QoS policies it is possible to scale DDS from low-end embedded systems connected with narrow and noisy radio links, to high-end servers connected to high-speed fiber-optic networks.

These DDS resource QoS policies provide control over the local and end-to-end resources, such as memory and network bandwidth. Such properties are particularly relevant in SoS since they are characterized by largely heterogeneous subsystems, devices, and network connections that often require down-sampling, as well as overall controlled limit on the amount of resources used.

4.5 Configuration

The QoS policies described above, provide control over the most important aspects of data delivery, availability, timeliness, and resource usage. DDS also supports the definition and distribution of user specified bootstrapping information via the following QoS policies:

- The `USER_DATA` QoS policy allows applications to associate a sequence of octets to domain participant, data readers and data writers. This data is then distributed by means of a built-in topic. This QoS policy is commonly used to distribute security credentials.

- The `TOPIC_DATA` QoS policy allows applications to associate a sequence of octet with a topic. This bootstrapping information is distributed by means of a built-in topic. A common use of this QoS policy is to extend topics with additional information, or meta-information, such as IDL type-codes or XML schemas.

- The `GROUP_DATA` QoS policy allows applications to associate a sequence of octets with publishers and subscribers–this bootstrapping information is distributed by means built-in topics. A typical use of this information is to allow additional application control over subscriptions matching.

These DDS configuration QoS policies provide useful a mechanism for bootstrapping and configuring applications that run in SoS. This mechanism is particularly relevant in SoS since it provides a fully distributed means of providing configuration information.

5 Guidelines for Building System of Systems with DDS

This section presents a systematic method for building scalable, extensible, and efficient SoS by integrating them through DDS. This method has been used successfully in many SoS and has shown over time its value on addressing the key requirements faced when architecting a SoS, including (1) achieving the right level of scalability and extensibility while maintaining loose coupling and (2) minimizing resource usage.

The method described below will be presented as a series of steps, which are applied incrementally—and often iteratively–to achieve the appropriate SoS design. With time and experience the number of iterations required will diminish. It is advisable, however, to iterate two to three times through the steps described below when first applying the method. A visual representation of the steps involved in this approach is outlined in Figure 7.

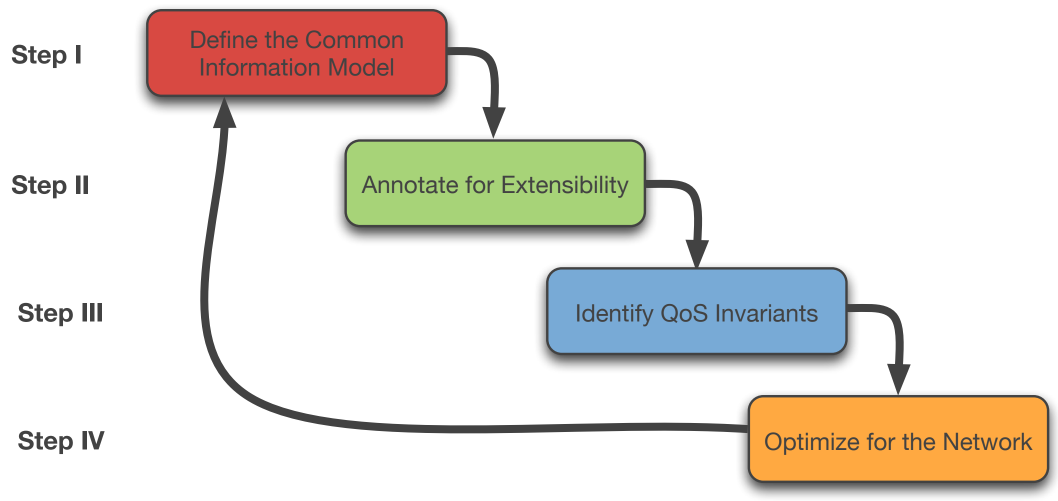

Figure 7: Visual representation of the steps involved in the common information model approach.

5.1 Step I: Define the Common Information Model

Integrating systems to form a SoS involves interconnecting these systems in a meaningful way. A conventional approach is to integrate systems in a pairwise manner, thus leading to a star-like system topology depicted in Figure 8(a). This conventional approach to system integration has several shortcomings, however, because (1) is not scalable since it requires integrating with $n - 1$

systems, (2) it is not resource efficient since it duplicates information, and (3) it is hard to extend since each change is usually propagated on the $n - 1$ point-to-point integrations.

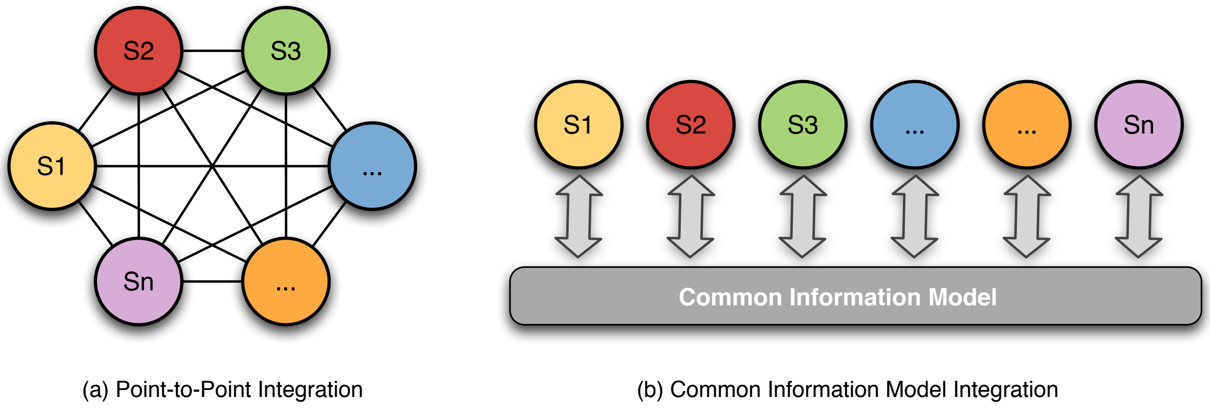
S2
S3
S1
...
Sn
...
(a) Point-to-Point Integration
S1
S2
S3
...
...
Sn
Common Information Model
(b) Common Information Model Integration

Figure 8: System of System Integration Styles.

An alternative approach to architect SoS involves focusing on a common set of abstractions—the *common information model*—used to represent all information that is necessary and relevant for the interworkings of the SoS. This approach, depicted in Figure 8(b), reduces—and in some cases eliminates—the integration effort since all systems communicate using the same protocol and type system. This approach also migrates some effort to the careful design of the common information model, which defines the data representations that establish the *lingua franca* for the SoS.

The first step required to build a common information model involves devising the data types that capture the state and the events relevant throughout the SoS. This data can then be normalized using one of the several forms defined in the database management systems literature [10]. After applying this first step, the information model should be free of common anomalies, such as the update or deletion anomalies, and should be efficient, in the sense that information duplication will have been removed via the normalization process. At this point, however, the information model may not be ideal for SoS with respect to aspects like evolvability, efficiency, and QoS.

5.2 Step II: Annotate for Extensibility

The second step of information model design should account for the extensibility requirements of each data type. Some data types must support evolutions, such as the ability of adding or removing attributes. Other data types must disallow extensions since they represent structural invariants, such as the type representing some physical structure of the system like the position and number of wheels. In either case, it is necessary to consciously decide what kind of extensibility is associated with each data type and use the annotations described in Section refSection:DDS:TypeSystem.

5.3 Step III: Identify QoS Invariants

The previous steps allow refinement of an information model so that it cleanly captures key traits of the SoS. At this point, however, the information model does not capture any non-functional requirements associated with various data types. The next step, therefore, involves identifying the least stringent set of QoS policies (see Section 4) that should be associated with each data type to meet SoS non-functional requirements.

Decorating the common information model with the proper QoS ensures data producers can only produce data with stronger guarantees, whereas data consumers can only ask to consume data with weaker guarantees. This rule ensures the QoS violations do not occur and that the SoS will work as expected.

5.4 Step IV: Optimize for the Network

After the common information model is decorated with QoS there is yet another steps to perform to address the fact that (1) a SoS is a distributed system, which requires awareness of network characteristics, and optimization of the used bandwidth as some of the subsystem or devices will often be connected through narrow-bandwidth links or will inherently have scarce computational and storage resources, and (2) DDS data is sent atomically, i.e., regardless of what changes occur in the (non-optional) fields of a data type when the entire data type is transmitted across the network.

These considerations requires additional scrutiny on the information model. In particular, it is necessary to identify all the data types that belong in one of the following cases:

- **Update frequency mix.** Each data type should be regarded with respect to the relative update frequency of its attributes. If there is a subset of attributes that are relatively static and another subset that changes relatively often, it is advisable to split the data type into two data types. The two types will share the key to allow correlation on the receiving side. This technique minimizes bandwidth utilization by limiting the amount of data sent over the network. For SoS that communicate over some low bandwidth links this technique significantly improve performance.

- **QoS mix.** Since QoS policies in DDS apply to the whole topic it is important to recognize that the `DURABILITY` or `RELIABILITY` QoS policy affects all attributes of the data type associated with the topic. In certain cases, however, some attributes will work fine with a weaker QoS setting. In such case, it is advisable to split data types into as many types as necessary to ensure that all attributes within a given data type share the same QoS, i.e., no attribute could select a weaker QoS without compromising correctness. This technique can improve both performance and resource utilization.

5.5 Step V: Iterate

The steps I to IV described above should be performed iteratively to ensure that (1) all key SoS concepts have been captured, (2) extensibility constraints have been handled, (3) the QoS policies properly capture non-functional invariants, and (4) the data model is efficient and scalable. With experience the number of iterations required will reduce, but you should typically apply these steps at least twice.

6 DDS: The Road Ahead

The DDS technology ecosystem is characterized by a lively and vibrant community that continues to innovate and extend the applicability of this powerful P/S technology. This section we summarize the state-of-the-art in DDS and then examine the DDS standards that will be forthcoming in the next year or so.

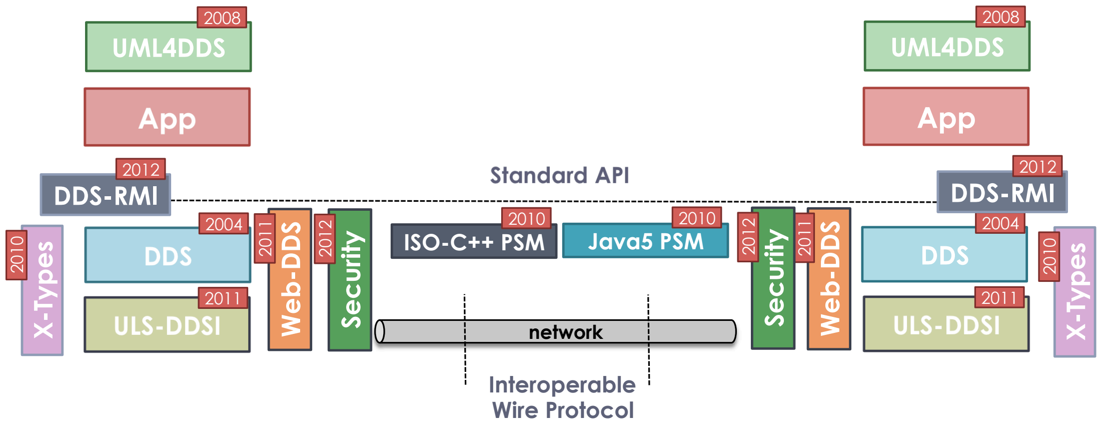

Figure 9: The DDS Standard Evolution.

6.1 State-of-the-Art for DDS

Figure 9 presents the whole family of standards belonging to the DDS ecosystem, including some of the upcoming extensions. As described in earlier sections, DDS supports QoS-enabled topic-based P/S [5] where topics are strongly typed and where a structural polymorphic type system is used to enable type-safe system extensibility and evolution [7]. DDS is equipped with a standard wire-protocol, DDSI [6], that provides native interoperability across vendor implementations of the standard.

Two new APIs were recently specified for DDS. One API defines a new mapping to ISO C++ and another defines a mapping to Java 5. Both APIs improve the productivity, safety and efficiency of DDS applications. As a result of these enhancements, DDS now provides the most dependable and productive standard-based communication middleware infrastructure for building mission- and business-critical SoS that require scalability, fault-tolerance, efficiency and performance.

6.2 Coming Next

There are three areas that will yield the following new DDS standards by 2012:

- **Web-enabled DDS.** This specification addresses the needs of SoS that must expose DDS data to Web/Internet applications. These capabilities will make it possible to expose DDS topics in a standardized manner to both RESTful and W3C web services. Web-enabled DDS will simplify the way in which SoS can bring mission-critical and real-time information to enterprise applications, as well as to browser-enabled devices, such as smart phones and tablets.

- **Secure DDS.** To date, DDS Security has been an area where each vendor has provided their own proprietary (and thus non-interoperable) solutions. With the increased adoption of DDS in SoS the need for an interoperable and extensible security infrastructure has become evident. As a result, work is progressing on an interoperable DDS security specification that will address many aspects of security, such as data confidentiality, integrity, assurance and availability. This specification is based on pluggable security modules that will allow vendors to provide a default interoperable set of behaviours and customers or vertical domains to develop their own customized security plugins.

- **Ultra Large Scale Systems (ULS) DDSI.** The DDS wire-protocol, knowns as DDSI, was designed by first optimizing for LAN deployments and then adding a series of mechanisms that vendors can use over WANs. The DDSI wire-protocol does not, however, take advantage of the latest research on dissemination and discovery protocols, such as encoding techniques, dynamic distribution trees, and distributed hash-tables. The ULS DDSI specification will thus extend the existing DDSI protocol to further improve its efficiency over WAN and improve the scalability on ULS deployments [9].

In summary, the DDS technology ecosystem continues to expand its applicability and support systems and SoS efficiently and effectively.

7 Concluding Remarks

This chapter has introduced the DDS standard and explained its core concepts in the context of meeting the requirements of SoS. As it has emerged from the use cases cited throughout the chapter—as well as from the set of features characterizing this technology—DDS is the ideal technology for integrating Systems-of-Systems. The main properties DDS-based SoS enjoy include:

- **Interoperability and portability.** DDS provides a standardized API and a standardize wire-protocol, thereby enabling portability and interoperability of applications across DDS implementations. These capabilities are essential for SoS since it is hard to mandate a single product be used for all systems and subsystems, but it is easier to mandate a standard.

- **Loose coupling.** DDS completely decouples publishers and subscribers in both *time*, e.g., data readers can receive data that was produced before they had joined the system, and *space*, e.g., through its dynamic discovery that requires no specific configuration—applications dynamically discover the data and topics of interest. These two properties minimize coupling between the constituent parts of SoS, thereby enabling them to scale up and down seamlessly.

- **Extensibility.** The DDS type system provides built-in support for system extensibility and evolution. Moreover, the system information model can be evolved dynamically in a type-safe manner, which helps ensure key quality assurance properties in SoS.

- **Scalability, efficiency, and timeliness.** The DDS architecture and the protocols used in its core where designed to ensure scalability, efficiency, and performance. In addition, the QoS policies available in DDS provide fine-grained control over the non-functional properties of a system, thereby allowing finely tuning and optimization of its scalability, efficiency, and timeliness.

In summary, DDS is a natural choice as the integration infrastructure for SoS, as evidenced by the many adoptions of DDS as the basis for current and next-generation SoS.

8 Acronyms

DDS	Data Distribution Service for Real-Time Systems	2
DDSI	Data Distribution Service Interoperability Wire Protocol	3
DISR	DoD Information-Technology Standards Registry	3
DoD	Department of Defence	3
GDS	Global Data Space	3
IDL	Interface Definition Language	4
JMS	the Java Message Service	4
MILVA	Military Vehicle Association	3
MoD	Mistry of Defence	3
OMG	Object Management Group	2
P/S	Publish/Subscribe	2
QoS	Quality of Service	4
SoS	Systems-of-Systems	1
ULS	Ultra Large Scale Systems	17
UML	Unified Modeling Langauge	4
XML	eXtensible Markup Langauge	4

References

[1] http://bit.ly/okmbhm, 2003.

[2] Luca Cardelli. Type systems. *ACM Comput. Surv.*, 28:263–264, 1996.

[3] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.

[4] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.

[5] Object Management Group. Data distribution service for real-time systems, 2004.

[6] Object Management Group. Data distribution service interoperability wire protocol, 2006.

[7] Object Management Group. Dynamic and extensible topic types, 2010.

[8] Sun Microsystems. The java message service specification v1.1, 2002.

[9] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems - The Software Challenge of the Future. Technical report, Software Engineering Institute, Carnegie Mellon, June 2006.

[10] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw Hill Higher Education, 3rd edition, 2002.