energies

MDPI

Article

Configurable DDS as Uniform Middleware for Data Communication in Smart Grids

Alaa Alaerjan [1], Dae-Kyoo Kim [2,*], Hua Ming [2] and Hwimin Kim [2]

[1] Department of Computer Science, Jouf University, Sakaka 72388, Saudi Arabia; asalaerjan@ju.edu.sa
[2] Department of Computer Science and Engineering, Oakland University, Rochester, MI 48309, USA; ming@oakland.edu (H.M.); hwiminkim@oakland.edu (H.K.)
* Correspondence: kim2@oakland.edu; Tel.: +1-248-370-2863

Received: 16 January 2020; Accepted: 7 April 2020; Published: 10 April 2020

Abstract: Data Distribution Service (DDS) has emerged as a potential solution for data communication challenges in smart grids. DDS is designed to support quality communication for large scale real-time systems through a wide range of QoS policies. However, a smart grid involves various types of communication applications running on different computing environments. Some environments have limited computing resources such as small memory and low performance, which makes it difficult to accommodate DDS. In this paper, we present a feature-based approach for tailoring DDS to configure lightweight DDS by selecting only the necessary features for the application in consideration of the resource constraints of its running environment. This allows DDS to serve as a uniform communication middleware across the smart grid, which is critical for interoperability. We analyze DDS in terms of features and design them using Unified Modeling Language (UML) and Object Constraint Language (OCL) based on inheritance and overriding. We define a formal notion of feature composition to build DDS configurations. We implemented the approach in OpenDDS and demonstrate its application to different application environments. We also experimented the approach for the efficiency of configured DDS in terms of resource utilization. The results show that configured DDS is viable for efficient and quality data communication for applications that run on an environment with limited computing capability.

Keywords: communication; configuration; DDS; feature; publish–subscribe; smart grid

1. Introduction

Smart grids have emerged as the next generation of power grids for improved efficiency, reliability, and flexibility of power production and consumption. Unlike the traditional power grid, A smart grid is data-centric involving significant data communication among various types of applications running on different computing environments across the grid [1,2]. Some environments have ample computing resources (e.g., memory, CPU clock speed) which allow heavy-weight applications to run, while other environments have limited computing capability which can accommodate only small and light-weight applications. In the current practice, there has not been a widely accepted communication platform for smart grids. Existing protocols such as Distributed Network Protocol (DNP) [3] and Modbus [4] in the traditional power grid are not suitable for smart grids due to the lack of support for a large amount of data and quality communication. DNP involves 50%~80% of processing delay in power devices over TCP/IP [5] and Modbus is able to support only simple applications (e.g., programmable logic controllers) within the same network.

Smart grid deployment is the modernization process of the power grid with new power resources, technologies, and devices for efficient management of energy. The core of the modernization is data communication between end-users and the grid such as home-to-grid (H2G), building-to-grid (B2G),

industry-to-grid (I2G), and vehicle-to-grid (V2G) [1]. There have been efforts (e.g., Smart Energy Profile 2.0 [6]) for standardizing smart energy management for businesses and homes by sharing IP-based information and control. However, as the above types of communication increasingly involve various kinds of devices and applications, a more scalable and flexible communication paradigm such as publish–subscribe communication is needed. There has been active work on adopting publish–subscribe communication for smart grids (e.g., [7–9]). The existing work proposes various models for different domains in a smart grid, which leads to interoperability issues. The dominant communication paradigm in the traditional grid is client-server communication (e.g., IEC 61850 [10]) with limited data exchanges. As the grid becomes modernized over time, increased data communication is inevitable and alternative communication paradigms should be considered. Per the study by Petersen et al. [11], publish–subscribe communication outperforms other communication paradigms for smart grids.

Data Distribution Service (DDS) [12], which is a publish–subscribe communication standard by Object Management Group (OMG), has emerged as a potential solution to address the communication challenges in smart grids [13]. DDS is capable of supporting large scale real-time systems with a wide range of quality of service (QoS) policies. It also supports TCP, UDP, and shared memory over different network configurations (e.g., LAN, WAN) via various wired/wireless communication technologies (e.g., Ethernet, 4G, Wi-Fi). However, DDS requires high computing resources to run and the application environments that have limited computing resources are not capable of accommodating DDS [13–15], which can cause interoperability issues if different communication protocols are adopted for those applications.

In this paper, we present an approach for tailoring DDS for light-weight DDS per the computing capability of application environments so that DDS can serve as a uniform communication platform across a smart grid, which facilitates interoperability. The approach is based on the feature modeling of DDS. DDS is analyzed and designed in terms of features based on inheritance and overriding. When an application is developed, only the features that are needed for the application can be chosen in consideration of the computing resources of the running environment. This allows one to configure light-weight DDS by selecting only the necessary features for the application in consideration of the resource constraints of its running environment. In this way, DDS can be adopted by various applications across the smart grid, serving as a uniform communication platform. We use the Unified Modeling Language (UML) [16] to design DDS features as it is used as the base modeling language in DDS. We also adopt the Object Constraint Language (OCL) [17] to precisely specify the quality of service (QoS) policies in DDS. We implemented the approach in OpenDDS [18] and demonstrate its application to four different application environments. We experimented with the viability of the approach by measuring the memory use and performance of the configured DDS in each environment. The results show that the configured DDS runs efficiently while satisfying the quality requirements of grid communication.

The remainder of the paper is organized as follows. Section 2 outlines related research on adopting DDS to smart grids. Section 3 gives an overview of DDS and smart grid communication. Section 4 describes design principles and modeling DDS features. Section 5 presents feature composition. Section 6 demonstrates case studies of applying the approach to four different application environments and their implementation in OpenDDS. Section 7 evaluates the implementations for resource utilization in terms of memory use and CPU utilization. Section 8 gives a brief discussion on cybersecurity in smart grids. Section 9 concludes the paper with a discussion of future work.

2. Related Work

There exists some work (e.g., [19–22]) on using DDS to address communication requirements in smart grids or related fields such as real-time systems and wireless sensor networks.

The work by Youssef et al. [19] proposes a DDS-based communication system to address reliability and latency requirements of different types of smart grid applications. They demonstrated that DDS is

flexible enough to support the heterogeneity of applications with high reliability and low latency even at high transmission rates.

Shi et al. [20] present an implementation of DDS for exchanging monitoring data in a micro-grid. The purpose of their work is to demonstrate the capabilities of DDS to support data sharing within a micro-grid. They conclude that DDS is suitable for lightweight communication in micro-grids.

Perez and Gutierrez [21] studied the viability of adopting DDS to real-time systems. They modeled the performance of DDS using an end-to-end flow model with focus on reliability and latency. They describe QoS policies using Analysis of Real-Time Embedded Systems (MARTE) [23]. Their experiments show that DDS is suitable for real-time systems with support for high reliability.

Beckmann and Dedi [22] proposed a three-layer architecture for DDS to support wireless sensor networks (WANs). The architecture defines DDS in terms of the API layer, the platform-independent layer, and the platform-specific layer to facilitate the development of DDS applications and ease the porting process between WAN platforms. While their work is based on their own layers which are not part of DDS, our work is based on the Data-Centric Publish–Subscribe (DCPS) layer which is already defined in DDS. This facilitates the development of DDS applications in conformance to the DDS standard.

3. Background

In this section, we give an overview of DDS's structure and smart grid communication in terms of involved devices providing computing environments to smart grid applications and their computing capabilities.

3.1. Data Distribution Service

DDS is a data-centric publish–subscribe protocol designed for large scale and real-time systems in the IoT domain. It supports multicasting, peer-to-peer communication, and dynamic discovery with a wide range of QoS policies for quality communication. DDS facilitates data interoperability by allowing topics to be represented in different syntaxes such as OMG Interface Definition Language (IDL) or Extensible Markup Language (XML). DDS is also compatible with both TCP and UDP for the transport protocol.

DDS consists of Data Local Reconstruction Layer (DLRL) and DCPS. DLRL, which is optional, outlines how an application should interface to DCPS. DCPS, which is mandatory, resides below DLRL and enables DDS components to communicate with each other. DCPS defines the following entities—domain participant, publisher, data writer, subscriber, data reader, topic, and QoS policy. A domain participant represents an application participating in the data domain. It is responsible for creating publishers, subscribers, and topics of its own. An application may have multiple publishers and subscribers. A publisher publishes data in the network. It is associated with a set of data writers which are responsible for writing data to be published. Once data is written, the data writer notifies its publisher the availability of the data. It serves as an interface between the application and the publisher. A subscriber receives published data. It is associated with a set of data readers which are responsible for reading in received data. Once data is received, the subscriber notifies its associated data reader which makes the data available to the application. The data reader serves as an interface between the application and its subscriber. A topic represents a data object of name and type. In order for a publisher and a subscriber to communication each other, the topic of the publisher must match the topic of the subscriber. The same topic may be associated with multiple publishers and subscribers. A topic may have multiple instances, each having a unique key. A QoS policy specifies a requirement for quality communication. DDS defines 22 different QoS policies and an individual entity can specify its own set of QoS policies. Depending on the type of entities, applicable QoS policies may differ. For example, the durability policy can be applied to topic, but not applicable to publisher and subscriber.

Figure 1 shows the structure of DCPS and data communication in a data domain. Two domain participants DP-1 and DP-2 communicate with each other for topics A, B, and C. Each participant contains a pair of a publisher and a subscriber. The publisher in DP-1 is associated with one data writer responsible for writing the topic A which is subscribed by DP-2 through its subscriber that has only one data reader.

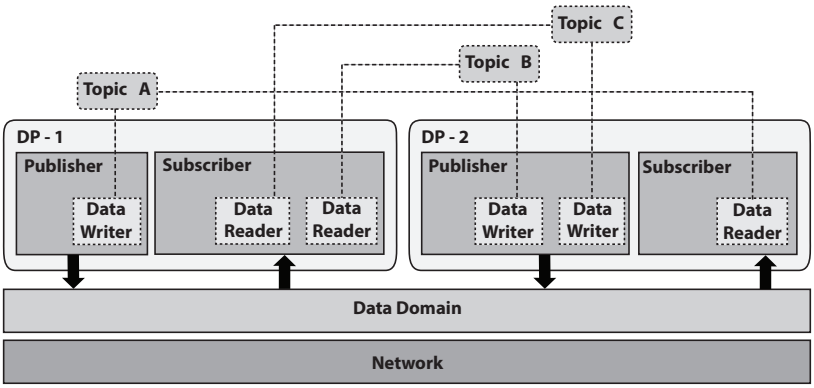


Figure 1. Data-Centric Publish–Subscribe (DCPS) structure.

3.2. Smart Grid Communication

The smart grid domain involves four sub-domains [24]—power generation, power transmission, power distribution, and power consumption. These domains involve various types of communication devices providing computing environments to smart grid applications. These devices have different computing resources. Table 1 shows several types of devices per domain. The following acronyms are used in the table—ACC: Automation Control Computer, AIC: Automation Industrial Computer, DAQC: Data Acquisition Computer. HAM: Home Automation Module, HMI: Human Machine Interface, PFD: Protection Field Device, PLC: Programmable Logic Controller, PMU: Phasor Measurement Units, SC: Substation Computer, WEM: Wireless Energy Monitor, and WSN: Wireless Sensor Node.

Based on our survey, we categorized the devices into limited devices and capable devices in terms of memory and CPU clock speed. Devices with 2 GB or more memory and 1.0 GHz or higher CPU speed are categorized as capable and otherwise categorized as limited. For example, a Raspberry Pi which is used in the Distribution domain as a WSN with 0.5 GB memory and 0.7 GHz CPU speed is categorized as a limited device, while a 6135A/PMUCAL which is used in the transmission domain as an IED with 2 GB memory and 2 GHz CPU speed is categorized as a capable device.

A key success factor in smart grids is smooth and uninterrupted data communication across different domains in a smart grid [24]. In the traditional power grid, different domains use different communication protocols, which hinders the communication between domains and consequently compromises interoperability [25]. To make it worse, these protocols are not designed for heavy data communication. This has been a great barrier in moving toward smart grids. An ideal solution for this problem is to adopt a uniform communication protocol that can serve all the domains in a smart grid. Such a protocol should be flexible enough to be accommodated by not only capable devices, but also limited devices. It should be also able to support quality requirements (e.g., reliability, latency) which is critical in smart grid communication.

Table 1. Devices in smart grid domains.

Domain	Device Type	Device Example	Memory	CPU	Capability
Consumption	Smart Home HMI	EX2N-43H	0.2 GB	0.64 GHz	Limited
	HAM	Boardcon EM210	0.5 GB	0.8 GHz	
	WEM	Raspberry Pi2	1 GB	0.9 GHz	
Distribution	PLC	EX2N-100H	0.2 GB	0.72 GHz	Limited
	WSN	Raspberry Pi	0.5 GB	0.7 GHz	
	PFD	Protection IED 670	1 GB	0.9 GHz	
Transmission	IED/PMU	6135A/PMUCAL	2 GB	2.0 GHz	Capable
	SC	SEL-3355	16 GB	2.8 GHz	
	ACC	AIS Box PC	16 GB	3.3 GHz	
Generation	HMI	RXi Industrial	4 GB	1.0 GHz	Capable
	AIC	IPC-4	4 GB	2.16 GHz	
	DAQC	DA-820	16 GB	3.1 GHz	

4. Modeling DDS for Smart Grid

In this section, we design DDS in terms of features using feature modeling [26]. A feature is defined as a functional unit of integration where one feature can be integrated with another. This allows DDS to be tailored by selecting and integrating only the necessary features for a specific application, so that DDS can be adopted even for the applications that run on resource-constrained devices. Given that, a configuration is defined as an integration of a set of features and it shall henceforth be used interchangeably with configured DDS and trailed DDS. In this work, we focus on DCPS which is the core layer of DDS.

4.1. Modeling Principles: Inheritance and Overriding

We use UML class diagrams and sequence diagrams to design DDS features. UML [16] is chosen as it is used as the base notation in the DDS standard [12]. We design DDS features based on inheritance and overriding which are defined as follows.

Inheritance

Let *mathcalC* be the universal set of classes and *mathcalR* be the universal set of relationships. For a given feature f, $class(f) \subset mathcalC$ denotes the set of the classes of f and $rel(f) \subset mathcalR$ signifies the set of the relationships of f. Then, the structural properties of f denoted as $cd(f)$ are defined as $class(f) \cup rel(f)$ and the behavioral properties of f denoted as $sd(f)$ are defined as the set of the sequence diagrams of f.

Definition 1. *A child feature cf inherits its parent feature pf iff*

I1. $\forall c' : class(pf) \exists c'' : class(cf) \bullet c' = c''$;
I2. $\forall r' : rel(pf) \exists c'' : rel(cf) \bullet r' = r'' \wedge \forall e' : end(r') \exists e'' : end(r'') \bullet e' = e''$ *where* $end(r)$ *is the set of the ends of a relationship* r;
I3. $\forall s' : sd(pf) \exists s'' : sd(cf) \bullet s' = s''$.

Overriding

Let $name(e)$ be the name of an element e.

Definition 2. *A child feature cf overrides its parent feature pf iff*

$O1.\ \exists e' : cd(pf), e'' : cd(cf) \bullet name(e') = name(e'');$
$O2.\ \exists s' : sd(pf), s'' : sd(cf) \bullet name(s') = name(s'').$

Overriding must not cause any conflict with non-overridden entities. The syntactic notion of conflict observes the abstract syntax of UML [16]. The semantic notion of conflict depends on the context of the feature under consideration.

Figure 2 shows the feature model of DCPS. The model defines DCPS in terms of the publication and subscription features. The filled triangle underneath the DCPS node denotes that the two features can be selected inclusively. The publication feature can be either QoS-based or simple, but not both. The exclusive selection is denoted by the empty triangle beneath the publication node. The QoS_Based feature has two mandatory features—publication QoS and common QoS which are denoted by the filled circle on the nodes. the publication QoS feature include QoS policies that are specific to publication such as DurabilityService and WriterDataLifecycle. The common QoS feature provides QoS policies that are applicable to both publication and subscription. They include the deadline, ResourceLimits, history, durability, and reliability. The dashed arrows represent consistent dependencies. That is, the use of the history and reliability features must be consistent with the ResourceLimits feature. The notion of consistency is further described in Section 4.2.3. Similar to publication, subscription can be also either simple or QoS-based. The simple feature provides the basic subscription function, while the QoS_Based feature allows QoS policies to be enforced in subscription. The QoS_Based feature involves two mandatory features—common QoS and subscription QoS and two alternative features—listener and condition. The subscription QoS feature provides the QoS policies that are specific to subscription such as TimeBasedFilter and ReaderDataLifeCycle. The listener feature allows one to decide if data should be received by the subscriber listener or the data reader listener or both. The condition feature specifies a condition for data to be read.

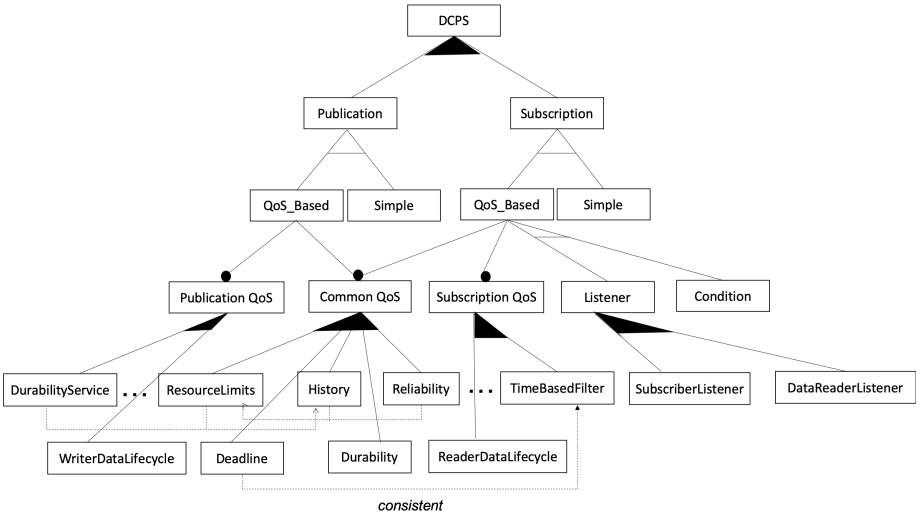


Figure 2. Feature model of DCPS.

4.2. Publication Features

The publication feature is concerned with publishing data on a specific topic. Data can be published in a simple way without concerning QoS or with QoS policies for quality communication. The former is captured by the simple feature and the latter is captured by the QoS_Based feature.

4.2.1. Simple Publication Feature

The simple feature provides basic functions for publishing data without QoS. Only the minimal computing resources are needed to accommodate this feature, which makes it suitable for limited devices. Figure 3 shows the structure and publish behavior of the simple feature. The class diagram

involves the application, publisher, DataWriter, and topic classes which are described in Section 3.1. The one-to-many relationships between the classes are specified by multiplicities on association ends. The sequence diagram describes the publish behavior. The application calls the write operation on a specific data writer to write data being published and once the data is written and ready to published, the data writer calls the publish_data() operation on the publisher associated with the data writer to publish the date.

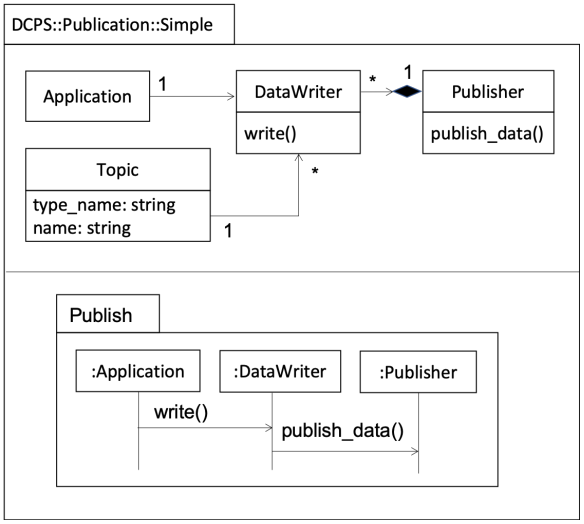


Figure 3. Simple publication feature.

4.2.2. QoS_Based Publication Feature

The QoS_Based feature supports QoS policies in publication for quality communication. Figure 4 shows the structure and setting QoS policy behavior of the QoS_Based feature. In addition to the classes in the simple feature, this feature also includes the abstract QosPolicy class which serves as an anchor point for inheritance in its sub-features. The Set QoS Policy for DataWriter sequence diagram specifies the behavior of setting a QoS policy for a data writer. The application designates a specific QoS policy of its interests and sets it in the data writer. After successful setting, the data writer acknowledges back to the application. The Set QoS Policy for Publisher sequence diagrams can be explained similarly.

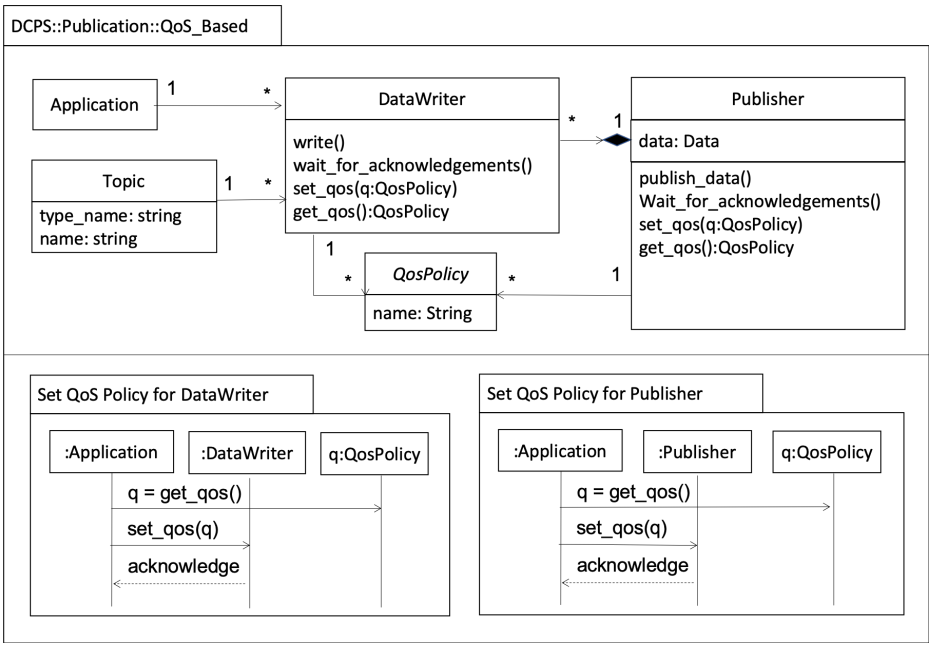


Figure 4. QoS_Based publication feature.

4.2.3. Common QoS Feature

This feature provides the QoS policies that are common to both publication and subscription. They include Durability, History, ResourceLimits, Reliability, DestinationOrder, Presentation, LatencyBudget, Deadline, Partition, Liveliness, EntityFactory, UserData, TopicData, GroupData, and Ownership. In this work, we focus on ResourceLimits, History, Deadline, Durability, and Reliability which are commonly used. QoS features might have dependencies on each other. For example, the History feature requires the use of the ResourceLimits feature as specified in Figure 2. We use OCL to specify the semantics of QoS features. OCL is chosen for precise definition.

ResourceLimits Feature

This feature is used for specifying the maximum number of data samples that can be managed by a single data writer or a data reader. Data samples can be queued until the specified number. Figure 5 shows the structure of the feature. The feature involves ResourceLimitsQosPolicy and QosPolicy classes. The QosPolicy class refers to the same class in Figure 4 which is denoted by « and ». This allows inheritance of the relationships of the QosPolicy class in Figure 4 per the inheritance principle in Section 4.1. This enables use of OCL to define the semantics of this feature.

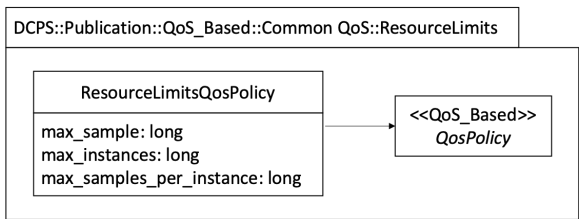


Figure 5. ResourceLimits feature.

The ResourceLimitsQosPolicy class defines three attributes—max_samples, max_instances, and max_samples_per_instance. The max_samples attribute specifies the total number of data samples, the max_instances attribute sets the maximum number of topic instances, and the max_samples_per_instance attribute is used to constrain the maximum number of data samples per instance, which must be less than or equal to max_samples. Alternatively, the constant

LENTH_UNLIMITED may be used to indicate the absence of the limit. The above constraints are specified in OCL as follows.

```
context ResourceLimitsQosPolicy inv:
    max_samples_per_instance = LENGTH_UNLIMITED or
    max_samples_per_instance ≤ max_samples
```

This feature is used together with the History feature to keep historical data. To be consistent with the History feature, the value of the max_samples_per_instance attribute must be greater than or equal to the value of the depth attribute in the History feature.

Durability Feature

This feature is used to specify whether a data writer should keep or discard data samples after publishing, which enhances the decoupling between data writers and data readers. The kept data can be made available to late-joining data readers in the network even after the data samples have been published and delivered to interested readers. Data samples may be kept in either memory (while the data write is alive) or a persistent storage. Figure 6 shows the structure of the feature. The feature involves the DurabilityQosPolicy class for setting durability. There are four durability settings.

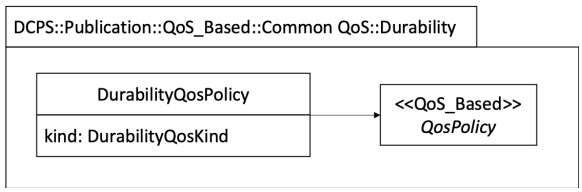


Figure 6. Durability feature.

- VOLATILE: This setting is used to discard data samples after they have been sent to all known subscribers.
- TRANSIENT LOCAL: This setting requires data readers to receive all the data samples kept in the data writer's history.
- TRANSIENT: This setting indicates that data samples outlive and last as long as the data writer is alive, which means that they are kept in a memory. Data readers associated with a TRANSIENT writer should receive all cached samples.
- PERSISTENT: This setting provides the same functionality as the TRANSIENT setting, but the cached data samples with this setting are stored in a persistent storage. This means that data samples outlive even if a data writer is no longer alive.

They are ordered as follows: VOLATILE < TRANSIENT LOCAL < TRANSIENT < PERSISTENT. In order for a data writer and a data reader to communicate with each other, the data reader's setting must be less than or equal to the data writer's setting. The durability is set through the kind attribute in the DurabilityQosPolicy class. The following OCL expression specifies the durability constraint.

```
context DurabilityQosPolicy inv:
    DataWriter->(dw|dw.topic.DataReader ->
    forAll(dr|dr.DurabilityQosPolicy.kind ≤
    dw.DurabilityQosPolicy.kind))
```

Deadline Feature

This feature is used to control how frequently data samples should be written by a data writer. This information is critical to data readers to expect how frequently data would be received. In order

to give sufficient time for data to be received, the frequency of data readers must be greater than or equal to that of the data writer. Figure 7 show the structure of the feature.

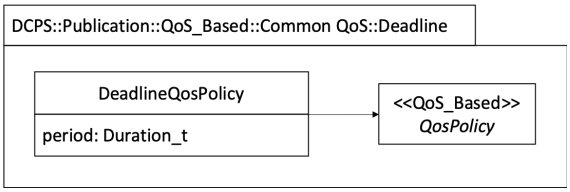


Figure 7. Deadline feature.

The sampling frequency is specified in the period attribute in the DeadlineQosPolicy class in this feature. The following OCL expression specifies the requirement.

```
context DeadlineQosPolicy inv:
    DataWriter->(dw|dw.Topic.DataReader->
    forAll(dr|dr.DeadlineQosPolicy.period ≥
    dw.DeadlineQosPolicy.period)
```

It should be noted that the frequency of data readers must be consistent with the TimeBasedFilter feature which specifies the required minimum time duration between data samples. In order for a data reader to be consistent with the feature, the frequency of the data reader must be greater than or equal to the minimum time interval of data sample required by the feature. This is specified in OCL as follows.

```
context DataReader inv:
    DeadlineQosPolicy.period ≥
    TimeBasedFilterQosPolicy.minimum_separation
```

History Feature

This feature specifies how many data samples should be kept in a data writer. Data may be kept until they are retrieved by the publisher or delivered to all interested data readers. In this way, data can remain intact even if it keeps changing before they are communicated. This feature is different from the Durability feature in that it deals with only the data samples on one side (e.g., data writer), while the Durability feature deals with data samples on both sides of data writer and data reader. Figure 8 show the structure of the feature.

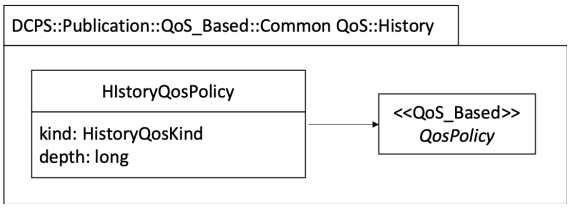


Figure 8. History feature.

The feature can be set to either KEEP_LAST or KEEP_ALL. The KEEP_LAST setting keeps a certain number of last samples as specified in the depth attribute in the HistoryQosPolicy class. On the other hand, the KEEP_ALL setting requires all samples to be kept until delivery. The setting is specified through the kind attribute in the HistoryQosPolicy class. Note that this feature should be consistent with the ResourceLimits feature in that the depth attribute must be less than or equal to the max_samples_per_instance attribute in the ResourceLimits feature. This is specified in OCL as follows.

```
context HistoryQosPolicy inv:
    kind = KEEP_LAST implies
    (depth ≤ ResourceLimitsQosPolicy.max_samples_per_instance) and
    kind = KEEP_ALL implies
    (ResourceLimitsQosPolicy.max_samples_per_instance =
    LENGTH_UNLIMITED)
```

Reliability Feature

This feature is concerned with reliable data delivery. It can be set to either BEST_EFFORT or RELIABLE. The BEST_EFFORT setting enables the publisher to make the best effort to deliver data, but does not guarantee the delivery. This setting is useful for the applications that publish data periodically or when latency is more concerned than reliability (e.g., sending visual data [27]). The behavior of the BEST_EFFORT setting is similar to the simple feature except that it may involve multiple data writers. The RELIABLE setting guarantees the delivery of data using acknowledgments. The publisher keeps publishing data until the receipt of the data is confirmed by the subscriber. Figure 9 shows the structure and the behaviors of the RELIABLE setting.

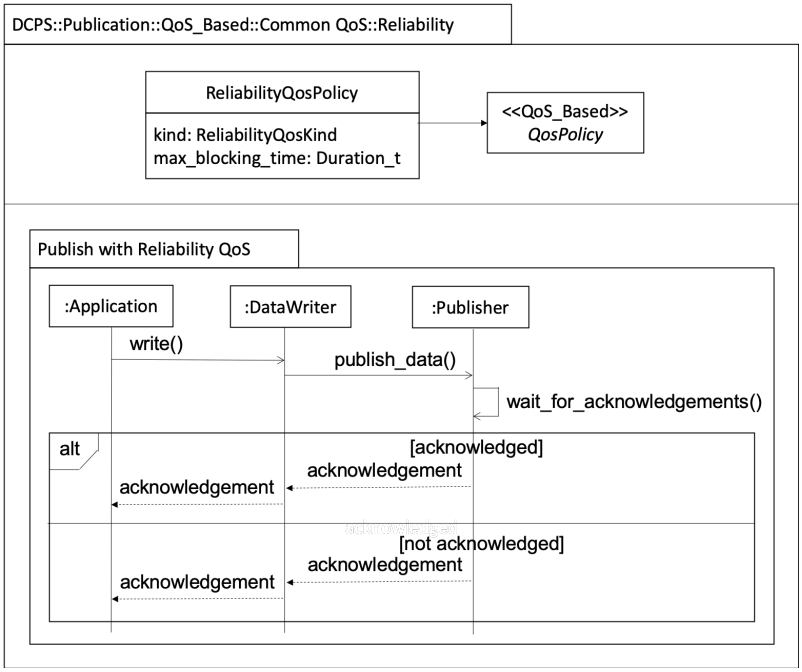


Figure 9. Reliability feature.

The wait_for_acknowledgments() operation waits for the acknowledgment of successful data receipt by all interested data readers. While waiting, the operation blocks data writers and publishers from writing and publishing new data until the receipt of published data has been confirmed. How long they should be blocked is specified in the max_blocking_time attribute in the ReliabilityQosPolicy class. The operation expires when the maximum blocking time has elapsed without a response, which is specified by the not acknowledged case in the alt fragment. This setting is useful for the applications that require high reliability such as energy management system (EMS) in substation automation [28]. Note that the RELIABLE setting is considered as greater than the BEST_EFFORT setting in comparison. In order for a data writer and a data reader to communicate under this feature, the reliability of the data writer must be greater than or equal to that of the data reader. This is specified in the below.

```
context ReliabilityQosPolicy inv:
    dataWriter -> (dw| dw.Topic.DataReader ->
    forAll(r|r.ReliabilityQosPolicy.kind ≤
    ReliabilityQosPolicy.kind))
```

This feature must be consistent with the ResourceLimits feature in that the RELIABLE setting requires LENGTH_UNLIMITED set in the max_samples_per_instance attribute in the ResourceLimits feature. This is specified in the below.

```
context ReliabilityQosPolicy inv:
    kind = RELIABLE implies
    ResourceLimitsQosPolicy.max_samples_per_instance =
    LENGTH_UNLIMITED
```

4.3. *Publication QoS Feature*

This feature provides the QoS policies that are specific to publication. They include lifespan, WriterDataLifecycle, DurabilityService, TransportPriority, and OwnershipStrength. In this work, we focus on DurabilityService and WriterDataLifecycle.

DurabilityService Feature

This feature is used to control the deletion of data samples from the data writer cache depending on the history setting and resource limit setting, which creates dependencies on the history and ResourceLimits features. Figure 10 shows the structure of the feature.

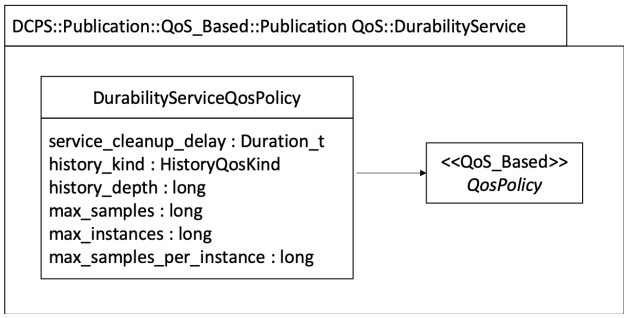


Figure 10. DurabilityService feature.

The time duration for deletion is set in the service_cleanup_delay attribute in the DurabilityServiceQosPolicy class. When the attribute is set for a specific duration with the history policy set to keep the last history, this feature requires the depth of the history be greater than zero and less than or equal to the max samples and max samples per instance in the resource limit policy. This is specified by the following OCL expression.

```
context DurabilityServiceQosPolicy inv:
    service_cleanup_delay > 0 and HistoryQosPolicy
    .kind = KEEP_LAST_HISTORY_QOS implies
    history_depth > 0 and
    ResourceLimitsQosPolicy.max_samples ≥
    history_depth and
    ResourceLimitsQosPolicy.max_samples_per_instance ≥
    history_depth
```

WriterDataLifecycle Feature

This feature is used to control the lifecycle of a topic instance managed by a data writer. Using this feature, the data writer can decide if unregistered instances should be disposed or kept. Figure 11 shows the structure of the feature. In the figure, the DataWriter class overrides the same class in Figure 4 per the overriding principle in Definition 2.

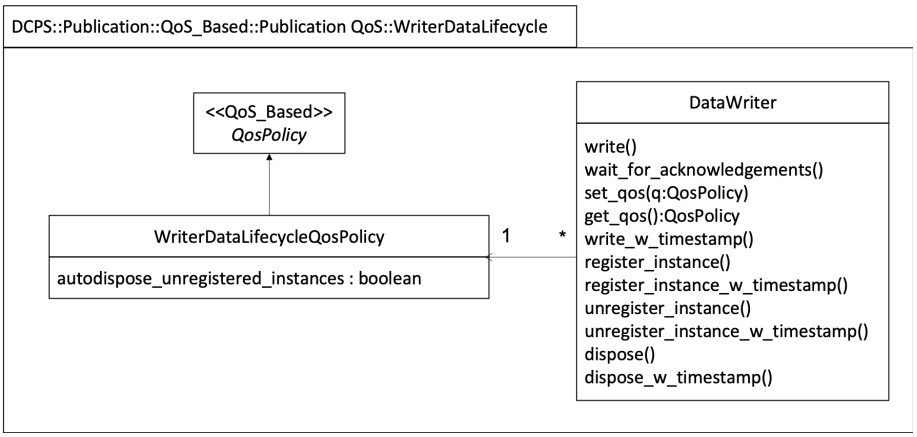


Figure 11. WriterDataLifecycle feature.

An instance becomes unregistered when the data writer unregisters it using the unregister_instance() operation in the DataWriter class or marks it as the final instance (no further changes to be made on data) using the unregister_instance_w_timestamp() operation. After unregistering, the autodispose _unregistered_instances attribute in the WriterDataLifecycleQosPolicy class is set to TRUE indicating that the topic instance has been disposed. It might be desirable not to dispose of the unregistered instance in case the data writer wants to re-register the instance in the future. In such a case, the attribute is set to FALSE. This is specified by the following OCL expression where " ^ " denotes the *hasSent* operation.

```
context WriterDataLifecycleQosPolicy inv:
    DataWriter^unregister_instance() and
    DataWriter^dispose() implies
     DataWriter.WriterDataLifecycleQosPolicy
    .autodispose_unregistered_instances = TRUE
```

Note that if a data writer is deleted, all of its associated topic instances are unregistered by the service.

4.4. Subscription Features

The subscription feature is used to subscribe data on a specific topic. Similar to the publication feature, the subscription feature is refined into the simple feature and the QoS_Based feature. The simple feature provides a simple subscription function without QoS, while the QoS_Based feature supports QoS in subscription.

4.4.1. Simple Subscription Feature

The simple feature provides the basic function for subscribing data without QoS. It requires only the minimal memory and CPU loads for subscription, which is suitable for applications running on limited devices (e.g., wireless energy monitors). Figure 12 shows the structure and subscribe behavior of the simple subscription feature. The class diagram involves the application, DataReader, subscriber, and topic classes. The sequence diagram specifies that upon data arrival, the subscriber informs the

availability of the data to the data reader and the data reader reads in the data and makes it available to the application. The behaviors assume that the subscriber has already registered its interests in a particular topic during discovery.

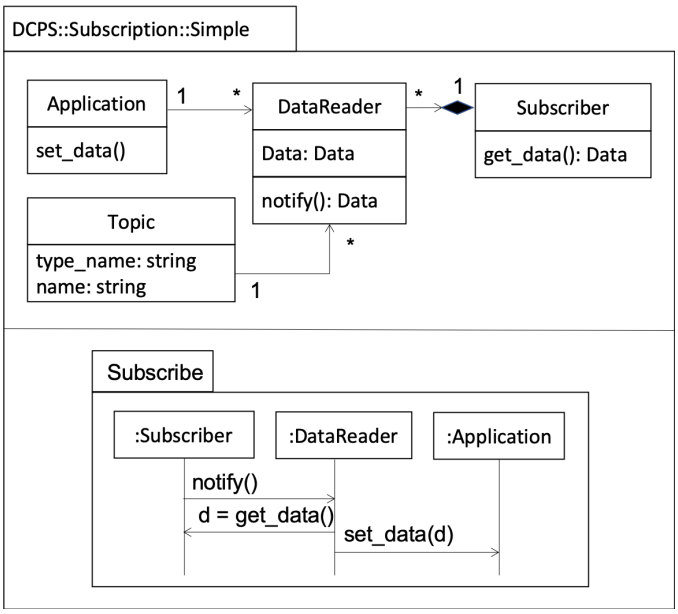


Figure 12. Simple subscription feature.

4.4.2. QoS_Based Subscription Feature

The QoS_Based feature supports QoS policies in data subscription. This feature is refined into common QoS, subscription QoS, condition, and listener. The common QoS feature is shared with publication and has been described in Section 4.2.3. The subscription QoS feature provides the QoS features that are specific to subscription. The condition and listener features address how data should be received under QoS policies. The condition feature requires a certain condition to be satisfied in order to receive data, while the listener feature constantly monitors data arrival. Only either the condition feature or the listener feature can be selected.

Figure 13 shows the structure and the subscribe behavior of the QoS_Based feature. In addition to the classes in the simple feature, the feature involves the QosPolicy class. The abstract QosPolicy class serves as a link to concrete QoS features by the inheritance principles in Definition 1. The sequence diagram specifies the behavior of setting a QoS policy for a data reader. Subscription starts after setting the QoS policy. Similar behaviors are defined for a subscriber.

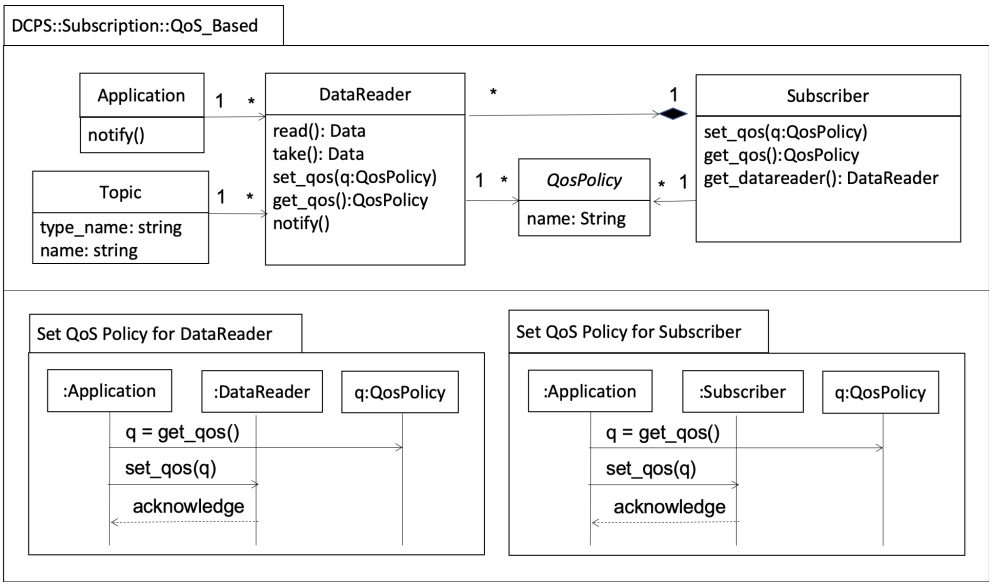


Figure 13. QoS_Based subscription feature.

4.4.3. Subscription QoS Feature

This feature provides the QoS features that are specific to subscription. They include TimeBasedFilter and ReaderDataLifecycle.

TimeBasedFilter Feature

This feature is used to control how often a data reader should receive data samples by setting the minimum time duration between data samples. This is specified in the minimum_separation attribute of the TimeBasedFilterQosPolicy in Figure 14. This allows a data reader to filter data samples. This feature is useful for applications running on limited devices that cannot accommodate all data samples.

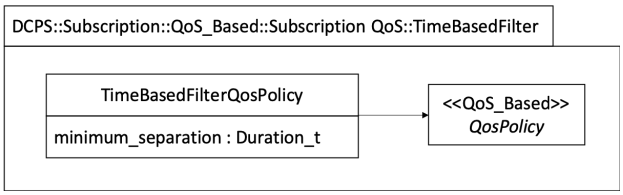


Figure 14. TimeBasedFilter feature.

This feature must be consistent with the deadline feature as they both control the frequency of receiving data. They are consistent if minimum_separation is less than or equal to the period value of the deadline policy. This is specified in the following.

```
context TimeBasedFilterQosPolicy inv:
    minimum_separation ≤ DeadlineQosPolicy.period
```

ReaderDataLifecycle Feature

This feature is used to control the lifecycle of a topic instance managed by a data reader. A data reader maintains certain information about the data samples that have not been consumed by the application or there exists an alive data writer associated with the data samples. The information and data samples are removed when there exists no longer data writer associated with the topic instance (i.e., the topic instance has been disposed of by data writers) or the data samples of the topic instance have been all consumed by the application (through the take() operation). After the removal, the data

reader reclaims the resources (e.g., memory) that are used to hold the removed information and data samples, so that they can be used for another topic instance. This feature applies to only data readers. Figure 15 shows the structure of the feature.

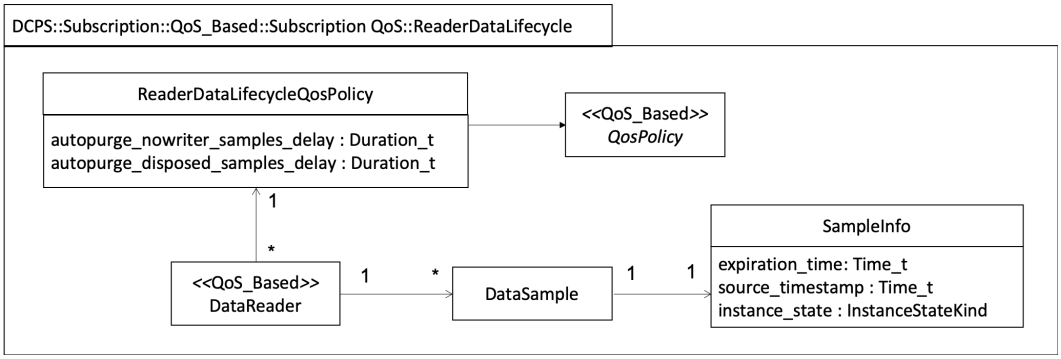


Figure 15. ReaderDataLifecycle feature.

The data reader uses the instance_state attribute in the SampleInfo class to determine the state of the data sample. Resources are reclaimed when the attribute is set to NOT_ALIVE_NO_WRITERS indicating the topic instance not alive with no associated data writer or NOT_ALIVE_DISPOSED indicating the topic instance disposed of by the data writer that wrote the data. The autopurge_nowriter_samples_delay attribute is used to specify the maximum duration that the data reader can wait before reclaiming resources for the NOT_ALIVE_NO_WRITERS state. The autopurge_disposed_samples_delay attribute is used to control how long the data reader should wait before reclaiming resources for the NOT_ALIVE_DISPOSED state. Once the specified duration of the attributes has elapsed, all the internal information regarding the data samples is deleted and resource reclaiming starts. This is specified as follows.

```
context ReaderDataLifecycleQosPolicy inv:
    (DataReader^take() and
    is_elapsed(autopurge_nowriter_samples_delay))implies
    DataReader.DataSample.SampleInfo.instance_state =
    NOT_ALIVE_NO_WRITERS or
    DataReader.DataSample.SampleInfo.instance_state =
    NOT_ALIVE_DISPOSED
```

4.4.4. Condition Feature

A condition may be defined for reading data. For example, the application may specify that in order to read data, two new data samples must be received. The feature is enforced by attaching a read condition to a WaitSet object representing a waiting for data. Figure 16 shows the structure and the subscription behaviors with a condition. The feature involves the ReadCondition class representing conditions and the WaitSet class capturing waiting locks. It also includes the Application and DataReader classes which are from the QoS_Based subscription feature to inherit other necessary classes and relationships to support this feature. The sequence diagrams describe that when a condition is defined, the application is put on waiting using a WaitSet object. When data is received, the subscriber makes the data available to the data reader, which triggers the read condition releasing the WaitSet object.

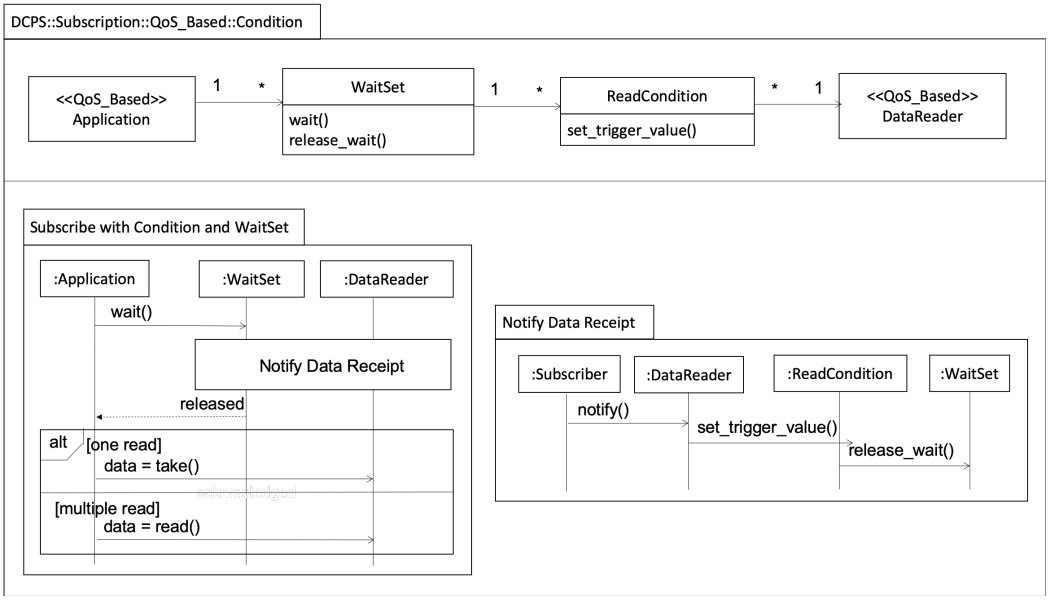


Figure 16. Condition feature.

4.4.5. Listener Feature

Instead of using a condition, a listener can be used for acknowledging the receipt of data. There are two types of listeners—subscriber-listeners and data reader-listener. Data may be also received by both a subscriber and a data reader in which case both a subscriber listener and a data reader listener are used together. In that case, the subscriber listener overrides the data reader listener.

SubscriberListener Feature

This feature uses subscriber-listeners to monitor data arrival when data is received by subscribers. Figure 17 shows the structure and the behaviors of subscribing data using a subscriber-listener. Once data has arrived, the subscriber makes the data available to the data reader designated to the specific topic of the data and informs the availability to the listener through the on_data_on_reader() operation in the SubscriberListener class. The listener then identifies which data reader contains the data using the get_datareaders() operation and notifies the application about the availability of the data. The application reads in the data from the data reader through either the read() or take() operation, which is captured in the alt fragment in the sequence diagram. The read() operation accesses the data as many times as needed without deleting it, while the take() operation allows accessing the data only once and deletes it after accessing.

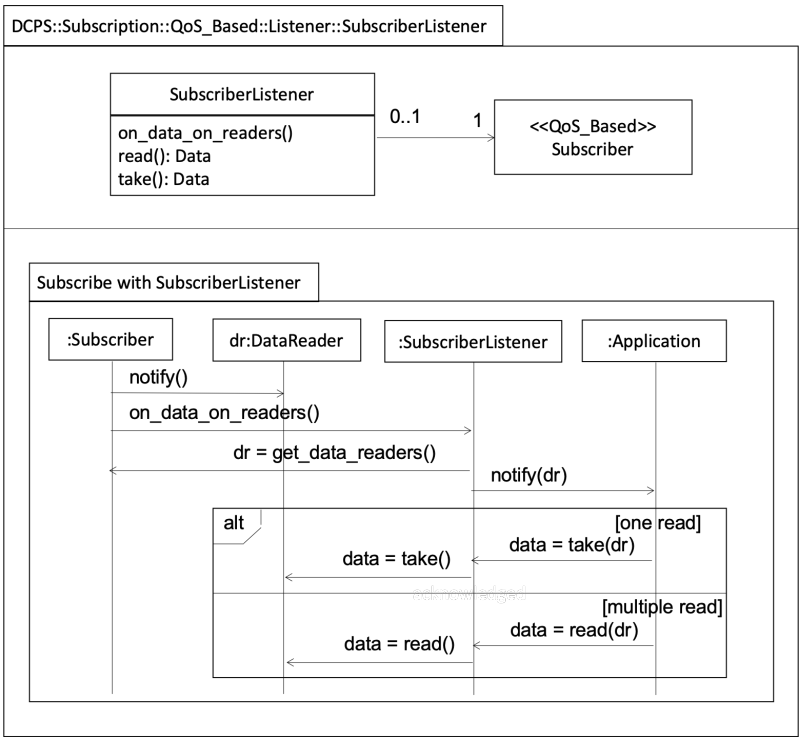


Figure 17. SubscriberListener feature.

DataReaderListener Feature

This feature uses data listeners to monitor data arrival when data is received by data readers. Figure 18 shows the structure and the behaviors of subscribing data using a data reader listener. Unlike the SubscriberListener feature where the listener has to inquire the subscriber about the data reader where data was received, the data reader listener already knows which data reader had received the data using the on_data_available() operation. The data reader listener notifies the availability of the data to the application for consumption.

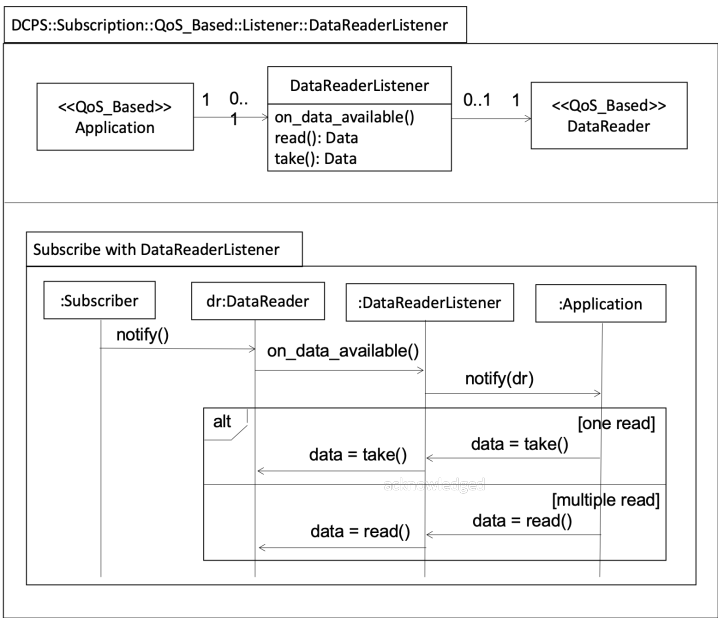


Figure 18. DataReaderListener feature.

5. Feature Composition

The feature modeling in Section 4 allows one to configure light-weight DDS by selecting only the necessary features for the application in consideration of its running environment. In this way, DDS can be adopted for various applications across a smart grid as a uniform communication platform. The configurations that consist of two or more features are subject to composition. Features are composed in terms of class diagrams and sequence diagrams. Note that composition is carried only for end-node features. Features in a hierarchy are subject to observe the inheritance and overriding principles in Section 4.1.

5.1. Class Diagram Composition

Class diagram composition is carried out in terms of class composition and relationship composition. We use P_c to denote the set of the properties of class c and CD_f to denote the class diagram of feature f.

Definition 3. *The composition of two classes c_a in CD_{f_a} and c_b in CD_{f_b} is a class c_c such that*

c1. $Inv(c_c) \Rightarrow Inv(c_a) \wedge Inv(c_b)$;
c2. $\forall p_a \in P_{c_a} \bullet [\forall p_b \in P_{c_b} \bullet p_a \neq p_b \Rightarrow p_a \in P_{c_c}]$;
c3. $\forall p_b \in P_{c_b} \bullet [\forall p_a \in P_{(c_a)} \bullet p_a \neq p_b \Rightarrow p_b \in P_{c_c}]$;
c4. $\exists p_c \in p_a \oplus p_b \bullet p_c \in P_{c_c}$ *if* $p_a \in P_{c_a}$, $p_b \in P_{c_b}$ *and* $p_a = p_b$.

$c1$ ensures class invariants preserved in the composed class. Other items ensure that the composed class includes both matching and non-matching attributes and operations.

A relationship $r_{f_a} : c^1_{f_a} \times c^2_{f_a}$ from CD_{f_a} is composed with a relationship $r_{f_b} : c^1_{f_b} \times c^2_{f_b}$ from CD_{f_b} if $r_{f_a} = r_{f_b}$. The composition r_{f_c} of r_{f_a} and r_{f_b} is a relationship $r_{f_c} : c^1_{f_c} \times c^2_{f_c}$ such that the bounds of r_{f_c} at the end $c^i_{f_c}$ is the intersection of the bounds of r_{f_a} at the end of $c^i_{f_a}$ and the bounds of r_{f_b} at the end of $c^i_{f_b}$. This ensures that the resulting end has the maximal bound interval that conforms to the end of both r_{f_a} and r_{f_b}.

Definition 4. *Let $\mathcal{E}(CD)$ be the set of classes and relationships of class diagram CD. A composition of two class diagrams CD_{f_a} and CD_{f_b} is a class diagram $CD_{f_c} \in CD_{f_a} \oplus CD_{f_b}$ such that*

d1. $\forall e_{f_a} \in \mathcal{E}(CD_{f_a}) \bullet [\forall e_{f_b} \in \mathcal{E}(CD_{f_b}) \bullet e_{f_a} \neq e_{f_b} \Rightarrow e_{f_a} \in \mathcal{E}(CD_{f_c})]$;
d2. $\forall e_{f_b} \in \mathcal{E}(CD_{f_b}) \bullet [\forall e_{f_a} \in \mathcal{E}(CD_{f_a}) \bullet e_{f_a} \neq e_{f_b} \Rightarrow e_{f_b} \in \mathcal{E}(CD_{f_c})]$;
d3. $\forall e_{f_a} \in \mathcal{E}(CD_{f_a}) \bullet \forall e_{f_b} \in \mathcal{E}(CD_{f_b}) \bullet e_{f_a} = e_{f_b} \Rightarrow \exists e_{f_c} \in \mathcal{E}(CD_{f_c} \bullet e_{f_c} \in e_{f_a} \oplus e_{f_b})$.

5.2. Sequence Diagram Composition

A feature may have several sequence diagrams defined for different behaviors. An operation on sequence diagrams $\oplus$ is a composition operation if each trace of $SD_1 \oplus SD_2$ can be obtained by interleaving a trace of SD_1 and a trace of SD_2 and all traces of SD_1 and SD_2 are used. The interleave of two traces of events is the set of traces obtained by interleaving the two traces in all possible ways.

Definition 5. *Let the set of traces of sequence diagram SD be denoted as $\mathcal{T}(SD)$. An operation $\oplus$ on sequence diagrams is a composition operation iff*

s1. $\forall t \in \mathcal{T}(SD_i) \bullet \exists t' \in \mathcal{T}(SD_1 \oplus SD_2) \bullet (t \rhd t')$ *for* $i = 1, 2$ *where* $t \rhd t'$ *denotes that t is a sub-sequence of t';*
s2. $\forall t' \in \mathcal{T}(SD_1 \oplus SD_2) \bullet \exists t_1 \in \mathcal{T}(SD_1) \bullet \exists t_2 \in \mathcal{T}(SD_2) \bullet t' \in (t_1 \,|||\, t_2)$ *where $t_1 \,|||\, t_2$ is the set of traces obtained from interleaving t_1 and t_2 in all possible ways [29].*

Given that, the composition of two features on sequence diagrams are defined as follows.

Definition 6. *Let SD_f be the set of the sequence diagrams of feature f. The composition of the sequence diagrams SD_{f_a} and SD_{f_b} is sequence diagrams SD_{f_c} such that*

q1. $\forall s_a \in SD_{f_a} \bullet [\forall s_b \in SD_{f_b} \bullet s_a \neq s_b \Rightarrow s_a \in SD_{f_c}]$;
q2. $\forall s_b \in SD_{f_b} \bullet [\forall s_a \in SD_{f_a} \bullet s_a \neq s_b \Rightarrow s_b \in SD_{f_c}]$;
q3. $\forall s_b \in SD_{f_b} \bullet [\forall s_a \in SD_{f_a} \bullet s_a = s_b \Rightarrow s_a \oplus s_b \in SD_{f_c}]$.

6. Case Studies

We demonstrate the approach by applying it to case studies. They are designed to demonstrate the configurability of DDS by selecting features to develop four configurations which include cf1) Simple publication feature, cf2) ResourceLimits, Durability, Reliability, and History publication features, cf3) Simple subscription feature, and cf4) ResourceLimits, Deadline, Reliability, History, and DataReaderListener subscription features. Figure 19 shows the four configurations. From a functional perspective, cf1 and cf2 are built for publishing applications, while cf3 and cf4 are for subscribing applications. From a computing perspective, cf1 and cf3 are designed to support simple applications (e.g., HAM, WEM, PLC) running on limited devices, while cf2 and cf4 support capable applications (e.g., IED, HMI, DAQC) running on capable devices.

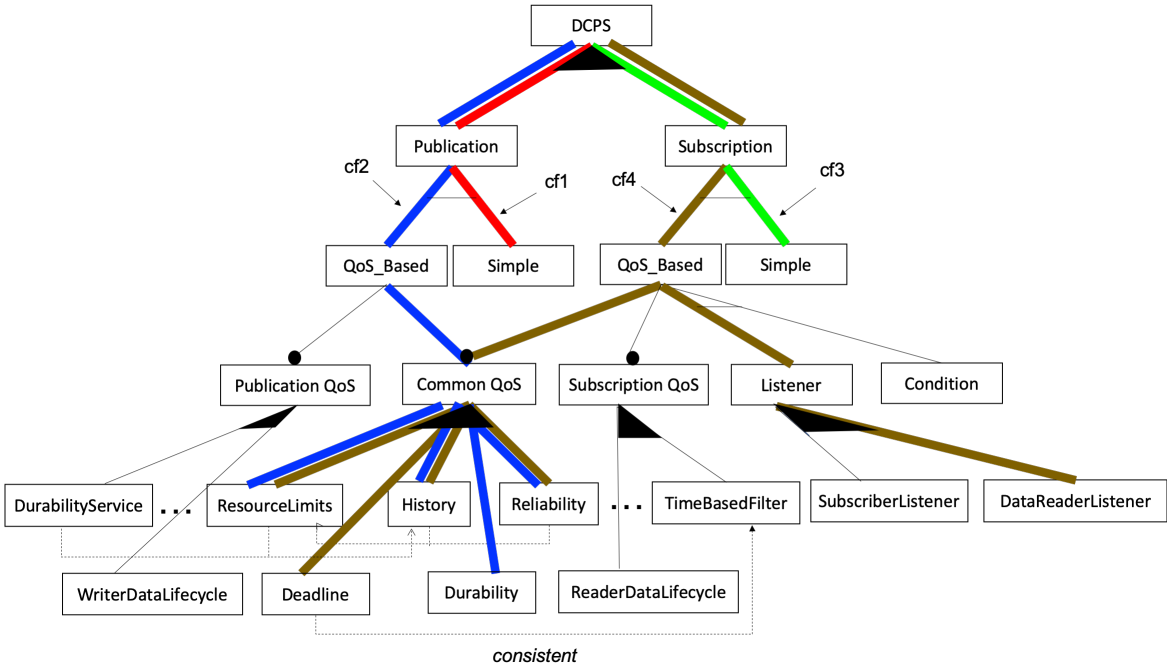


Figure 19. Four configurations.

6.1. Building Configurations

We first build the four configurations by composing the selected features based on the composition principles in Section 5. The configurations cf1 and cf3 involve only one feature and thus, no composition is needed and the same design as the feature is used. cf2 is concerned with publication involving four features—ResourceLimits, Durability, Reliability, and History. The composition of these features results in the design in Figure 20. In the figure, the Application, DataWriter, Publisher, Topic, and QosPolicy classes are added by inheritance from the QoS_Based publication feature. Other classes are added by the four features per d1 and d2 in Definition 4. The sequence diagrams are added by s1 in Definition 5.

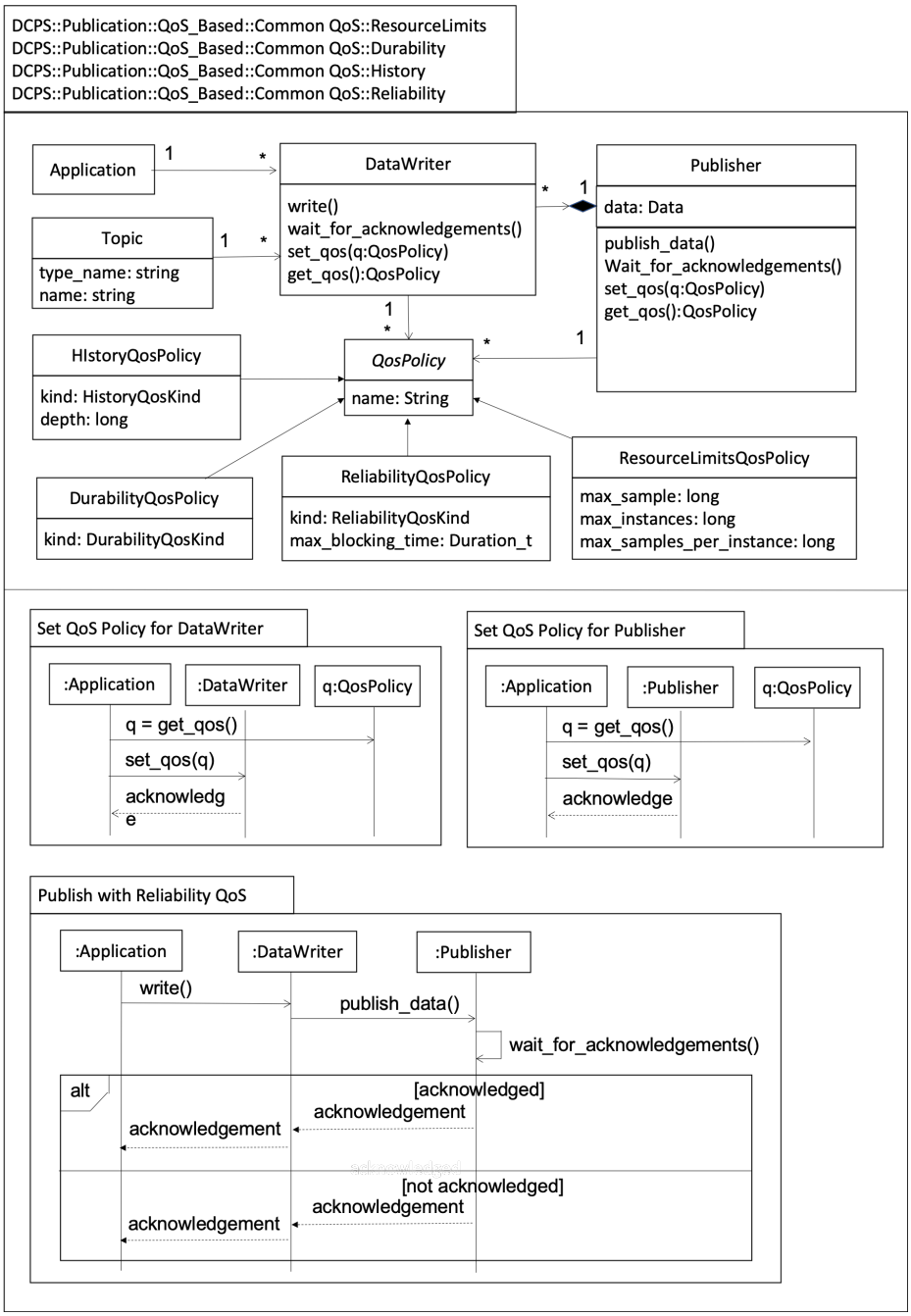


Figure 20. The cf2 configuration.

cf4 is concerned with subscription involving five features—ResourceLimits, Deadline, Reliability, History, and DataReaderListener. The composition of these features results in the design in Figure 21. In the figure, the Application, DataReader, Subscriber, Topic, and QosPolicy classes are added by inheritance from the QoS_Based subscription feature. Other classes are added by the five features per d1 and d2 in Definition 4. The sequence diagrams are added by s1 in Definition 5.

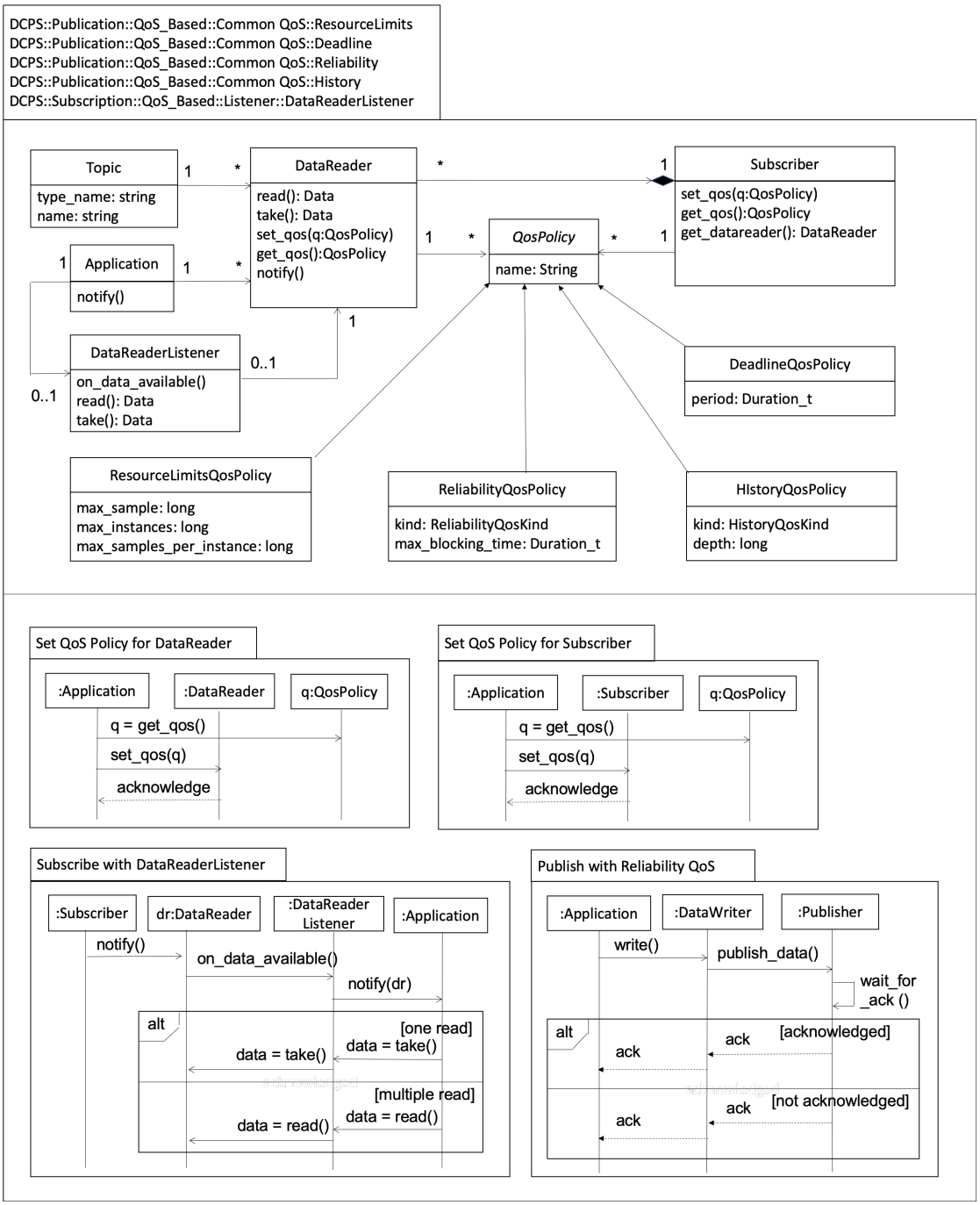


Figure 21. The cf4 configuration.

The designs developed in the above are used to tailor DDS for four different applications—a simple publishing application, a capable publishing application, a simple subscribing application, and a capable subscribing application.

6.2. Implementation

In this section, we demonstrate the implementation of the designs built in Section 6.1. We use OpenDDS [18] an open-source implementation of DDS as the base tool. Listing 1 shows a code fragment of cf2's implementation for setting QoS policies. In the listing, line 2 and 3 specify that the data writer is configured for QoS policies. Line 5 and 6 describe the Reliability policy requiring the data writer to guarantee the delivery of data samples. Line 7 sets the History policy for all data samples

to be held by the data writer until they are retrieved by the publisher and successfully delivered to interested subscribers. Line 8 specifies the Durability policy requiring that data samples must outlive the data writer. Line 9 describes the ResourceLimits policy setting the maximum number of data samples that the data writer can hold to 100. Line 11 to 13 create a data reader with a topic name (i.e., controlTopic) and the above QoS settings.

Listing 1: QoS setting for publication.

```
1   void Publisher::createDataWriter() {
2   DDS::DataWriterQos dw_qos;
3   publisher->get_default_datawriter_qos (dw_qos);
4
5      dw_qos.reliability.kind
6      = DDS::RELIABLE_RELIABILITY_QOS;
7      dw_qos.history.kind = DDS::KEEP_ALL_HISTORY_QOS;
8      dw_qos.durability.kind = TRANSIENT_DURABILITY_QOS;
9      dw_qos.resource_limits.max_samples = 100;
10
11  DDS::DataWriter_var dataWriter =
12  publisher->create_datawriter(controlTopic.in(),
13  dw_qos,0,OpenDDS::DCPS ::DEFAULT_STATUS_MASK); }
```

Listing 2 shows a code snippet of cf4's implementation for setting QoS policies. The listing is similar to Listing 1 except that line 2 creates a listener for the data reader and line 10 describes the Deadline policy for the data reader to read in data samples once every 500-millisecond.

Listing 2: QoS_Based setting for subscription.

```
1   void Subscriber::createDataReader() {
2   dr_listener = new DataReaderListenerImpl;
3   DDS::DataReaderQos dr_qos;
4   subscriber->get_default_datareader_qos (dr_qos);
5
6      dr_qos.reliability.kind
7       = DDS::RELIABLE_RELIABILITY_QOS;
8      dr_qos.history.kind = DDS::KEEP_ALL_HISTORY_QOS;
9      dr_qos.resource_limits.max_samples = 100;
10     dr_qos.deadline.period.sec = 0.5;
11
12  DDS::DataReader_var dataReader =
13  subscriber->create_datareader(voltageTopic.in(),
14  dr_qos, dr_listener.in (),::OpenDDS::DCPS
15  ::DEFAULT_STATUS_MASK); }
```

Backward compatibility is a key concern in the modernization of the electric grid [1,30]. The dominant communication paradigm in the traditional grid is client-server communication (e.g., IEC 61850 [10]). The proposed approach can be applied to the existing systems by extending the client-server communication with publish-subscribe features [31]. In this way, the existing systems may either continue to use the current communication method or adapt to the proposed technique by configuring the added publish-subscribe features.

7. Evaluation

In this section, we evaluate the effectiveness of the approach. An ideal environment for the evaluation is a real environment. However, as reported by DoE [32], the deployment of a smart grid is still ongoing (as part of the modernization of the electric grid) and there has not been a case where

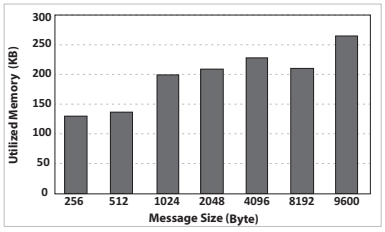
a smart grid is fully deployed. Given this, it is very difficult to find a real environment where the presented technique can be experimented. Even for those environments undergoing modernization, it is very difficult to access them as they usually belong to private sectors.

Alternatively, we evaluate the proposed techniques through a simulated environment. We load the four configurations into Raspberry Pi equipped with 0.5 GB memory and 0.7 GHz CPU and Raspberry Pi2 equipped with 1 GB memory and 0.9 GHz CPU. They are small devices which can barely accommodate a simple publisher or subscriber. We run cf1 and cf2 on Raspberry Pi simulating a publisher and observe their resource use in terms of memory and CPU performance and compare the results. cf1 is expected to use less resources than cf2 as cf1 does not use QoS policies. Similarly, we run cf3 and cf4 on Raspberry Pi2 simulating a subscriber and observe and compare their resource use. cf3 is expected to consume less resources than cf4. Note that the computing resources required by application functions and network module are not considered to avoid confusion.

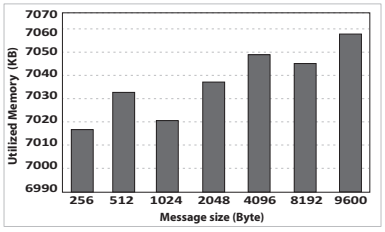
We also evaluate communication latency and reliability of configured DDS to ensure that the approach does not compromise communication quality. For this evaluation, we load cf2 and cf4 into 6135A/PMUCAL and DA-820 respectively.

7.1. Evaluating Publication Configurations

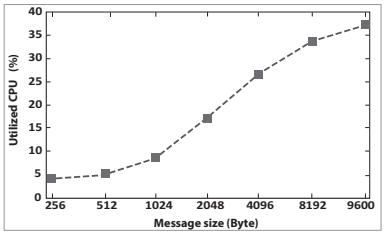
To evaluate publication configurations, we loaded cf1 and cf2 into the Raspberry Pi and compare their performance in terms of memory and CPU utilization. They were configured to publish 700 messages of 7 different sizes ranging from 256 to 9600 bytes at the rate of 10 messages/second (msg/s), which is typical for publish–subscribe protocols (e.g., GOOSE [33]) in the power domains. The experiment was conducted seven times for each configuration. Figure 22 shows the results. The graph (a) shows that cf1 took up 264.8 KB of memory at the most even for 9600-byte messages which are the largest. This accounts for only 0.05% of the device's memory (0.5 GB). On the other hand, the graph (b) shows that cf2 took up to 7057.4 KB of memory which accounts for 1.41% of the device's memory. While the percentage is still very low per the device's memory, it is a significant increase compared to cf1. With respect to CPU utilization, the graph (c) shows that cf1 used only 37% of the CPU for publishing 9600-byte messages, while cf2 requires up to 71% of the CPU for publishing the same messages as shown in the graph (d), which is almost double of cf1. This demonstrates that the configuration of publication features can make significant differences in required computing resources and DDS configured with the simple publication feature can be run on even small devices with decent performance.



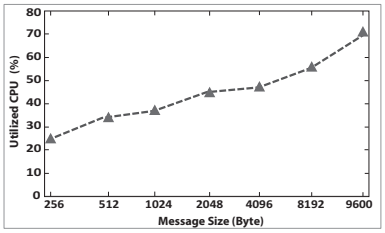
(**a**) Memory Used by cf1



(**b**) Memory Used by cf2



(**c**) CPU Used by cf1

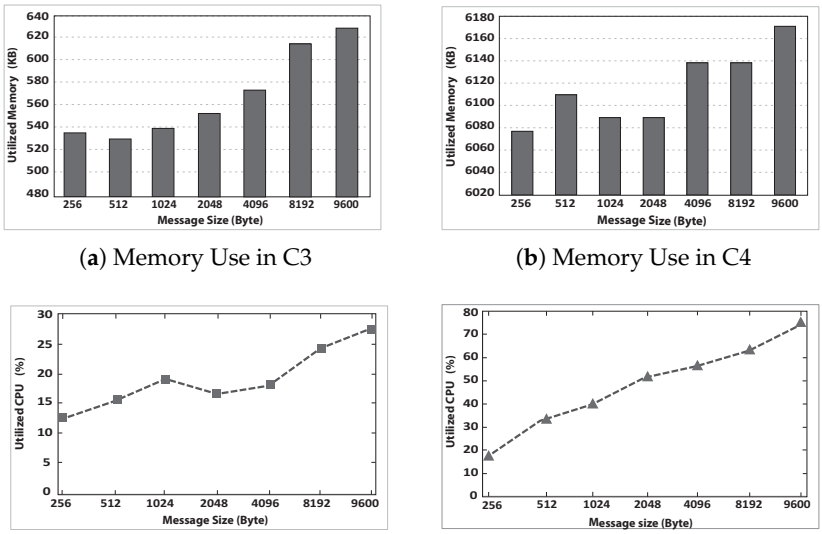


(**d**) CPU Used by cf2

Figure 22. Resource use of cf1 and cf2 on Raspberry Pi.

7.2. Evaluating Subscription Configurations

To evaluate subscription configurations, we loaded cf3 and cf4 into Raspberry Pi2 and compare their memory and CPU use. Same as the publication evaluation, 700 messages of 7 different sizes ranging from 256 to 9600 bytes are used to be subscribed at the rate of 10 msg/s. Each configuration was experimented with seven times. Figure 23 shows the results. The graph (a) shows that cf3 took up the maximum of 628.44 KB of memory for 9600-byte messages, which accounts for only 0.06% of the device's memory (1 GB). On the other hand, cf4 took up to 6171.08 KB of memory for 9600-byte messages, which accounts for 0.61% as shown in the graph (b). While this is still very low in consideration of the device's memory, it is ten times more than cf3, which is a significant increase. The graph (c) demonstrates that while cf3 maintains the CPU utilization below 28% for 9600-byte messages, cf4 uses up to 79% of the CPU as shown in the graph (d).



(**a**) Memory Use in C3 (**b**) Memory Use in C4

(**c**) CPU Use in C3 (**d**) CPU Use in C4

Figure 23. Resource use on Raspberry Pi2.

7.3. Quality Assessment

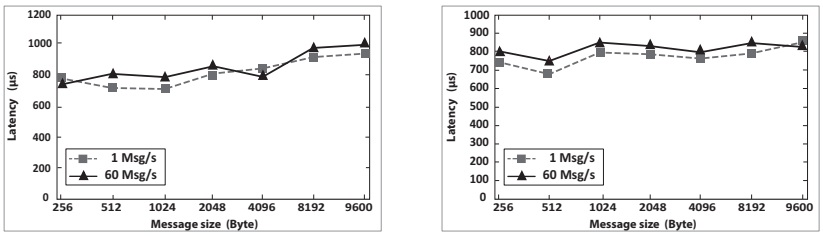
We also evaluated communication latency and reliability in communication between cf2 and cf4 which are connected via Ethernet with 100 Mbps network speed.

The cf2 is loaded into a 6135A/PMUCAL running PMU with 2 GB of memory and 2.0 GHz CPU. It is configured to publish messages at two different rates—1 msg/s and 60 msg/s which are the two end rates of the typical PMU range 1~60 msg/s for deadline policies [34]. 700 messages of seven different sizes were published at 1 msg/s and 4200 messages of the same set of sizes were published at 60 Msg/s. The messages published by cf2 are received by cf4, which is loaded into a DA-820 which has 16 GB of memory and 3.1 GHz CPU. It is configured to receive messages at the same rates as cf2. The messages are simulated to describe the measurements of voltage and current produced by a PMU. The structure of the messages is built based on IEEE C37.118.2 [35] for synchrophasor measurements for power systems. Figure 24 shows the content of a message. In the figure, the phasors section describes the phasor estimates, the analog values section describes sampled data such as control signal, and the digital status word section describes a status or a flag defined by the user to describe the state of the system.

```
Phasors
        Phasor #1: "Va.1",      122995.45V/_        0.00
        Phasor #2: "Vb.1",      122995.45V/_        0.00
        .... .. ....        ..........          ....
        .... .. ....        ..........          ....
Analog Values
        Analog value #1:        "Anlog1",    1000   (factor)
        Analog value #2:        "Anlog2",    2000   (factor)
        ..... ..... ..          ........  ....  ........
        ..... ..... ..          ........  ....  ........
Digital Status Word
        Digital status word  #1:  0xff00
        Digital status word  #2:  0xf000
        ..... ..... .... ..   .....
        ..... ..... .... ..   .....
```

Figure 24. The structure of Phasor Measurement Units (PMU) messages.

The graph in Figure 25a shows the results of measured latency of cf2 on 6135A/PMUCAL. The results show that the average latency for 9600-byte messages at 1 msg/s is measured as 827 µs and even at 60 msg/s, it remains under 871 µs, which is significantly lower than the required latency of 15~200 ms for PMU [34]. The graph in Figure 25b shows the measured latency of cf4 on DA-820. The results show that the average latency for 9600-byte messages at 1 msg/s is measured as 816 µs and even at 60 msg/s, it still remains below 803 µs which is significantly lower than the required latency of 15~200 ms for DAQC [27,28]. This experiment demonstrates that communication latency is maintained far satisfactorily in configured DDS.



(**a**) cf2 on PMUCAL (**b**) cf4 on DA-820

Figure 25. Communication latency.

We also measured the reliability of data delivery between cf2 and cf4. Every message published by cf2 was received successfully by cf4 at both rates under the experimental setting, which demonstrates 100% reliability. This well satisfies the 99.99 % reliability requirement of PMU and DAQC [27,28]. This shows that configured DDS does not compromise reliability.

8. Cybersecurity Consideration

A smart grid is a power grid supported by information technology (IT) and communication systems which are increasingly critical to the reliability of power systems [1]. As IT and communication sectors are increasingly coupled with the power grid, the complexity of power systems becomes high and the internal communication within the grid and the exposure of the grid to external networks increase. These vulnerabilities create a path for security threats such as network penetration, false data injection to the network, unauthorized access to power systems, data alteration on devices, and load redistribution. These attacks may not only impact on the efficiency and economics of the grid, but also threaten the continuity and quality of power supplies [36]. To protect the power grid from such attacks,

cybersecurity must be adopted. In order to adopt cybersecurity, the following challenges should be addressed.

- Lack of cybersecurity personnel. The power grid has its unique requirements for cyberseucirty [36,37] and there is little expertise available for smart grid cybersecurity. In order to cope with security threats in smart grids, security professionals who are specifically trained for the security challenges in the power domain should be available [1]. This includes the education about cybersecurity policy, procedures, and techniques as well as on the management, operational, and technical requirements that are necessary to secure power system resources.
- Lack of cybersecurity in legacy power components. Most legacy power systems in the traditional grid were developed with little consideration of cybersecurity [37]. For example, supervisory control and data acquisition (SCADA) systems have communications based on insecure protocols such as Modbus and DNP that do not have authentication and access control mechanisms integrated. This can be addressed by wrapping them to adapt to cybersecurity requirements. However, this requires changes in both hardware and software, which is not easy. Alternatively, they can be replaced gradually by newer models with advanced cybersecurity support. There have been growing efforts to improve the functionalities of power components for cybersecurity [38] and these efforts should continue to cope with cybersecurity concerns.
- Lack of privacy consideration. As "smart" devices (e.g., smart monitors, smart meters), which deal with end-user data, are introduced to the power domain, privacy issues arise. According to the study by Lisovich et al. [39], it is possible to monitor user behaviors through those devices even while the devices are not being used. Unauthorized access to such information may occur during transferring or storing data. Consequentially, this may lead the users to believe that they are under surveillance by the service providers, which may cause trust issues between the consumers and the power companies.

9. Conclusions

We have presented an approach for configuring light-weight DDS by selecting only the necessary features for the application, so that DDS can be adopted by not only large applications running on powerful devices, but also small application running on resource-constrained devices. In this way, DDS can be adopted as the uniform communication platform for various types of applications across a smart grid, which consequently enhances interoperability. The approach defines DDS in terms of features where a feature is a functional unit to be integrated with other features to build a configuration. In case studies, four configurations are developed and implemented on OpenDDS. They are evaluated under an experimental setting on the effectiveness of resource utilization and latency and reliability for quality communication. The results reveal that simple configurations use fewer computing resources, while QoS-based configurations use more computing resources. This demonstrates that depending on how DDS is configured, it can be accommodated by various types of devices with different computing capabilities across a smart grid. The experiments also show that DDS configurations do not compromise latency and reliability. We plan to extend the work to DLRL and RTPS to study the extent to which the discovery mechanism of DDS can be configured.

Author Contributions: Conceptualization, A.A. and D.-K.K.; methodology, A.A. and D.-K.K.; software, A.A.; validation, A.A. and D.-K.K.; formal analysis, A.A. and D.-K.K.; investigation, A.A. and D.-K.K.; resources, A.A.; data curation, A.A.; writing—original draft preparation, A.A.; writing—review and editing, D.K., H.M. and H.K.; visualization, A.A.; supervision, D.K. and H.M.; project administration, D.K.; funding acquisition, H.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. NIST. *NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0*; Technical Report 1108R2; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2012.
2. Asbery, C.; Jiao, X.; Liao, Y. Implementation Guidance of Smart Grid Communication. In Proceedings of the 48th North American Power Symposium, Denver, CO, USA, 18–20 September 2016.
3. DNP. *Distributed Network Protocol*; Technical Report. Available online: www.dnp.org (accessed on 17 February 2019).
4. Modbus. MODBUS over Serial Line Specification and Implementation Guide. Technical Report V1.02. 2006. Available online: http://modbus.org/ (accessed on 17 February 2019).
5. Lu, X.; Lu, Z.; Wang, W.; Ma, J. On Network Performance Evaluation toward the Smart Grid: A Case Study of DNP3 over TCP/IP. In Proceedings of the IEEE Global Telecommunications Conference, Houston, TX, USA, 5–9 December 2011.
6. Alliance, Z.; Appliance, H. *Smart Energy Profile 2 Application Protocol Standard*; Technical Report; ZigBee Alliance and HomePlug Appliance, 2013. Available online: http://www.csee.org.cn/Portal/zh-cn/Publications/atm/docs-13-0200-00-sep2-smart-energy-profile-2.pdf.pdf (accessed on 17 February 2019).
7. Ahmim, A.; Le, T.; Ososanya, E.; Haghani, S. Design and Implementation of a Home Automation System for Smart Grid Applications. In Proceedings of the IEEE International Conference on Consumer Electronics, Las Vegas, NV, USA, 7–11 January 2016.
8. Zu, X.; Bai, Y.; Yao, X. Data-Centric Publish-Subscribe Approach for Distributed Complex Event Processing Deployment in Smart Grid Internet of Things. In Proceedings of the 7th IEEE International Conference on Software Engineering and Service Science, Las Vegas, NV, USA, 26–28 August 2016.
9. Cao, Y.; Wang, N.; Kamel, G. A publish/subscribe communication framework for managing electric vehicle charging. In Proceedings of the International Conference on Connected Vehicles and Expo, Vienna, Austria, 3–7 November 2014.
10. IEC 61850. Communication Networks and System in Substation Automation, 2020. Available online: https://webstore.iec.ch/publication/6028 (accessed on 17 February 2019).
11. Petersen, B.; Bindner, H.; Poulsen, B.; You, S. Smart grid communication comparison: Distributed control middleware and serialization comparison for the Internet of Things. In Proceedings of the IEEE PES Innovative Smart Grid Technologies Conference Europe, Torino, Italy, 26–29 September 2017.
12. OMG. *Data Distribution Service (DDS)*; Technical Report 2015-04-10; Object Management Group: Needham, MA, USA, 2015. Available online: https://www.omg.org/spec/DDS/1.4/PDF (accessed on 17 February 2019).
13. Alaerjan, A.; Kim, D. Tailoring DDS to Smart Grids for Improved Communication and Control. In Proceedings of the 5th International Conference on Smart Cities and Green ICT Systems, Rome, Italy, 23–25 April 2016.
14. Alaerjan, A.; Kim, D. Adopting DDS to Smart Grids: Towards Reliable Data Communication. *Commun. Comput. Inf. Sci.* **2017**, *738*, 154–169.
15. Alaerjan, A.; Kim, D. Configuring DDS Features for Communicating Components in Smart Grids. In Proceedings of the 5th IEEE International Conference on Smart Energy Grid Engineering, Oshawa, ON, Canada, 14–17 August 2017.
16. OMG. *OMG Unified Modeling Language*; Technical Report 2017-12-05; Object Management Group: Needham, MA, USA, 2017. Available online: https://www.omg.org/spec/UML/2.5.1/PDF (accessed on 17 February 2019).
17. OMG. *Object Constraint Language*; Technical Report 2014-02-03; Object Management Group: Needham, MA, USA, 2014. Available online: https://www.omg.org/spec/OCL/2.4/PDF (accessed on 17 February 2019).
18. Object Computing Incorporated. OpenDDS Developer's Guide. Technical Report, 2017. Available online: www.objectcomputing.com (accessed on 17 February 2019).
19. Youssef, T.; Elsayed, A.; Mohammed, O. Data Distribution Service-Based Interoperability Framework for Smart Grid Testbed Infrastructure. In Proceedings of the IEEE 15th International Conference on Environment and Electrical Engineering, Rome, Italy, 10–13 June 2015.

20. Shi, K.; Bi, Y.; Jiang, L. Middleware-based Implementation of Smart Microgrid Monitoring Using Data Distribution Service over IP Networks. In Proceedings of the 49th International Universities Power Engineering Conference, Cluj-Napoca, Romania, 2–5 September 2014.
21. Perez, H.; Gutierrez, J. Modeling the QoS Parameters of DDS for Event-Driven Real-time Applications. *J. Syst. Softw.* **2015**, *104*, 126–140. [CrossRef]
22. Beckmann, K.; Dedi, O. sDDS: A portable data distribution service implementation for WSN and IoT platforms. In Proceedings of the 12th International Workshop on Intelligent Solutions in Embedded Systems, Ancona, Italy, 29–30 October 2015.
23. OMG. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*; Technical Report 2009-11-02; Object Management Group: Needham, MA, USA, 2009. Available online: https://www.omg.org/spec/MARTE/1.0/PDF (accessed on 17 February 2019).
24. NIST. *Framework and Roadmap for Smart Grid Interoperability Standards*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA. Available online: https://www.nist.gov/system/files/documents/public_affairs/releases/smartgrid_interoperability_final.pdf (accessed on on 17 February 2019).
25. Ma, R.; Chen, H.; Huang, Y.; Meng, W. Smart Grid Communication: Its Challenges and Opportunities. *IEEE Trans. Smart Grid* **2013**, *5*, 36–46. [CrossRef]
26. Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson, A. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*; Technical Report; Carnegie Mellon University: Pittsburgh, PA, USA, 1990.
27. United States Department of Energy. *Communication Requirements of Smart Grid Technologies*; Technical Report; US-DOE: Washington, DC, USA. Available online: https://www.energy.gov/ (accessed on 17 February 2019).
28. Sato, T.; Kammen, D.; Duan, B.; Macuha, A.; Zhou, Z.; Wu, J.; Tariq, M.; Asfw, S. *Smart Grid Standards Specifications, Requirements and Technologies*; Wiley: Hoboken, NJ, USA, 2015.
29. Störrle, H. Semantics of interactions in UML 2.0. In Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments, Auckland, New Zealand, 28–31 October 2003.
30. Ghatikar, G.; Bienert, R. Smart Grid Standards and Systems Interoperability: A Precedent with OpenADR. In Proceedings of the Grid-Interop, Phoenix, AZ, USA, 26 October 2011.
31. Kim, D.K.; Alaerjan, A.; Lu, L.; Yang, H.; Jang, H. Toward Interoperability of Smart Grids. *IEEE Commun. Mag.* **2017**, *55*, 204–210. [CrossRef]
32. United States Department of Energy. *Smart Grid System Report—2018 Report to Congress*; Technical Report; U. S. Department of Energy: Washington, DC, USA, 2018. Available online: https://www.energy.gov/oe/downloads/2018-smart-grid-system-report (accessed on 17 February 2019).
33. Almas, M.S.; Vanfretti, L. RT-HIL Implementation of the Hybrid Synchrophasor and GOOSE-Based Passive Islanding Schemes. *IEEE Trans. Power Deliv.* **2016**, *31*, 1299–1309. [CrossRef]
34. Huang, Z.; Kasztenny, B.; Madani, V.; Martin, K.; Meliopoulos, S.; Novosel, D.; Stenbakken, J. Performance Evaluation of Phasor Measurement Systems. In Proceedings of the IEEE Power & Energy Society General Meeting, Pittsburgh, PA, USA, 20–24 July 2008.
35. IEEE. *Standard for Synchrophasor Measurements for Power Systems (IEEE C37.118.2)*; Technical Report; IEEE: Piscataway, NJ, USA, 2011. Available online: www.ieee.org (accessed on 17 February 2019).
36. Li, Z.; Shahidehpour, M.; Aminifar, F. Cybersecurity in Distributed Power Systems. *Proc. IEEE* **2017**, *105*, 1367–1388. [CrossRef]
37. Grammatikis, P.; Sarigiannidis, P. Securing the Smart Grid: A Comprehensive Compilation of Intrusion Detection and Prevention Systems. *IEEE Access* **2019**, *7*, 46595–46620. [CrossRef]
38. Carvalho, R.; Saleem, D. Recommended Functionalities for Improving Cybersecurity of Distributed Energy Resources. In Proceedings of the 2019 Resilience Week, San Antonio, TX, USA, 4–7 November 2019.
39. Lisovich, M.; Mulligan, D. Inferring Personal Information from Demand-Response Systems. *IEEE Secur. Privacy* **2010**, *8*, 11–20. [CrossRef]