electronics

Article

Formalizing the Semantics of DDS QoS Policies for Improved Communications in Distributed Smart Grid Applications

Alaa Alaerjan

College of Computer and Information Sciences, Jouf University, Sakaka 72341, Saudi Arabia; asalaerjan@ju.edu.sa

Abstract: Quality communication is a major challenges in large-scale and distributed smart grid applications. Several protocols and middleware have been proposed to address communication quality issues in those applications. DDS is a standard data-centric middleware for publish/subscribe communication. It has been proposed for smart grid to address both connectivity and communication quality issues. DDS provides multiple quality of service (QoS) policies to address reliability, latency, and data availability. One of the main challenges in adopting the standard in smart grids is the complexity of adopting and tailoring its QoS policies. This is because those policies are described informally introducing ambiguities, which hinders the precise implementation of DDS. To address this, we formalize the descriptions of DDS QoS policies using the object constraint language (OCL). We also clearly defined the design structural relations among DDS entities and QoS policies. In the process, we analyzed the dependencies among QoS policies and we built clear and concise structural relations. We then proposed feature modeling and a management layer to facilitate QoS tuning and to reduce development and configuration complexity. We implemented the proposed approach in a simulated power consumption domain. The results show that the approach improves the development process. They also show that the approach significantly improves the performance of DDS-enabled applications.

Keywords: communication; data centric; DDS; distributed applications; OCL; QoS; smart grid

Citation: Alaerjan, A. Formalizing the Semantics of DDS QoS Policies for Improved Communications in Distributed Smart Grid Applications. *Electronics* **2023**, *12*, 2246. https://doi.org/10.3390/electronics12102246

Academic Editors: Imed Ben Dhaou, Ahmed Abdelgawad and Hannu Tenhunen

Received: 23 April 2023
Revised: 9 May 2023
Accepted: 12 May 2023
Published: 15 May 2023

1. Introduction

Distributed systems such as smart grids and renewable energy resources require high quality data communication [1,2]. Traditionally, the development of those systems relies on static distribution models such as the Remote Procedure Calls (RPC) [3]. However, given the technical advances in recent years, there has been a growing interest in adopting dynamic models to cope up with communication quality requirements. Specifically, communication models that are based on the publish/subscribe paradigm have become appealing to large-scale distributed systems due to their dynamism and scalability [4,5].

Smart grids and renewable energy resources have emerged in the last decade due to operational and environmental challenges (e.g., blackouts, greenhouse gas emissions) [6–8]. A smart grid depends on connectivity to overcome the issues in traditional power grids [9]. However, controlling the communication system is one of the main challenges in smart grids [10]. This is due to aspects such as the heterogeneity of domains, equipments, and systems [11]. Another critical challenge is managing the quality of the communication system in smart grids. This is because several applications within a smart grid impose rigid quality requirements, which makes quality communication one of the key design requirements [12]. Significant efforts have been made by researchers and industry to develop robust communication systems for smart grids [10]. Most of those studies proposed adopting flexible communication model (e.g., publish/subscribe, data centric) to cope with the requirements of different distributed applications in smart grids.

Several publish/subscribe protocols have been proposed for possible adoption in smart grids. Among them is the Data Distribution Service (DDS) [13], which has attracted significant attention. DDS is a standardized protocol based on the publish/subscribe communication paradigm. DDS aims at providing efficient and reliable communication for distributed systems. Multiple studies have proposed DDS as a potential solution for smart grids' communication issues [14]. DDS has also been adopted in other distributed systems, such as cyber-physical systems [15] and combat systems [16].

DDS enables reliable data communication by providing a wide range of QoS policies, which allows rigorous agreements on data exchange among distributed applications. The model also provides a flexible discovery mechanism, allowing communicating entities to be added and removed dynamically [17] without pre-configuration. DDS is brokerless, which means it does not have a single point of failure. The standard defines 22 QoS policies, each addressing a different quality aspect, such as data availability and reliability. DDS supports different transport protocols, such as TCP and UDP via wired or wireless communication [18]. However, one of the main challenges to adopting DDS in distributed environments is the complexity of its QoS policies [19]. To illustrate, the QoS policies in DDS standard are described informally, which introduces ambiguities and inconsistency among different implementations. Such ambiguities make it difficult to analyze dependencies among different policies, which is critical for the consistency of QoS agreements. Such an ambiguity deteriorates the adoption of DDS in different domains in smart grids. It also hinders the development process, which makes DDS complex to implement.

To address the above issues, we present a formalization approach to manage DDS QoS policies using the Object Constraint Language (OCL) [20]. We first categorize the QoS policies based on their objectives. Then, we localize dependencies and define a class diagram for each category to identify the relationships between DDS entities and QoS policies. Then, we analyze dependencies among QoS policies in the category using the class diagrams. Based on the analysis of class diagrams, we provide formal definitions of QoS policies using OCL. We demonstrate how the rigorous formalization of QoS policies helps the implementation of DDS. Additionally, we model the QoS policies using feature modeling to improve the efficiency of DDS development. Based on that, we propose a QoS management layer to facilitate implementing DDS in smart grid domains. Finally, we develop several use case experiments to simulate DDS communication in one domain in a smart grid.

The rest of this paper is structured as follows. Section 2 describes the related research on modeling different aspects of DDS including the QoS policies. It also describes some of the research done on adopting DDS in smart grids. Section 3 provides an overview of DDS, in terms of entities and layers. It also presents the formalization of QoS policies using OCL. Section 4 describes the proposed feature model to facilitate adopting DDS in a smart grid. In this section, we also describe the proposed QoS management layer. Section 5 demonstrates the implementation and the validation of the proposed research. Section 6 concludes the study with a discussion about future research.

2. Related Research

There is little research on formalizing QoS policies of DDS. In a modeling perspective, the formalization of QoS policies is part of DDS modeling. This section provides a summary of existing research on DDS modeling. We also discuss some of the research on adopting DDS to smart grids.

Perez and Gutierrez [19] studied the use of DDS QoS policies to support the quality requirements of real-time systems on an end-to-end basis. In particular, they focus on modeling quality policies for data delivery and timeliness using Modeling and Analysis of Real-Time Embedded Systems (MARTE) [21]. In their study, the importance of modeling the QoS policies in DDS is thoroughly described. They show that not only can the development process be improved, but also the quality of communication.

The study by Wang et al. [16] proposed tailoring the discovery mechanism of DDS for a warfare system based on the properties and requirements of the system. The tailored model interacts with other layers in DDS including the layers adjacent to the discovery model and other distant layers, such as the interface layer. Beckmann and Dedi [22] proposed a model-driven approach for customizing DDS to support Wireless Sensor Networks (WSNs). They introduced a layer underlying DCPS in DDS to ease the implementation of DDS for WSNs. In our previous study [23], we defined the missing behaviors of DCPS to improve the level of the completeness of the DDS standard using UML [24]. The improved model is used as the base for designing configurable DDS to support the devices that have limited computing resources.

With regard to adopting DDS in smart grids, multiple studies [25,26] have already established this research track. For instance, the study by Youssef et al. [25] proposed the adoption of DDS to the communication system in smart grids. The authors justified their study by the fact that DDS improves reliability and addresses latency requirements. In their study, the authors demonstrated their approach by implementing DDS in a simulated environment. The results of their research show that DDS can meet both latency and reliability requirements. Reference [26] also proposed adopting DDS in smart grids. In this study, the authors showed a use case by adopting DDS in a simulated micro-grid environment. The results of this research demonstrate the ability of DDS to support large-scale data sharing with imposed quality requirements.

Compared to the above, this study aims at facilitating the development of DDS in the context of smart grids. In particular, the key contribution points can be summarized as follows: (a) formalizing the semantics of the QoS policies in the DDS standard; (b) adopting feature modeling to improve the understandability of QoS policies and to ease the implementation of DDS; (c) proposing a QoS management layer based on the predictive configuration approach to enhance the performance of DDS.

3. Formalizing QoS Policies Using Dependency Analysis and Structural Relations

This section describes the proposed techniques for formalizing the semantics of DDS QoS policies. The process involves two main techniques, including dependency analysis and structural relations definitions.

3.1. Background

DDS [13] is a standard middleware publish/subscribe protocol. It is designed to support quality communication in large-scale data-centric distributed systems. DDS is structured in two layers. The upper layer is the Data Local Reconstruction Layer (DLRL), and the lower layer is the Data Centric Publish-Subscribe (DCPS). DLRL provides an interface to applications to integrate with DDS, while DCPS defines the core DDS entities and functionalities. In implementation, DLRL is an optional layer residing on top of DCPS. On the other hand, DCPS is the core required layer, since it consists of the main entities in DDS.

The DCPS consists of the multiple entities. For instance, the *Domain Participant* is the entity that represents the participation of an application in a data domain. This entity is responsible for creating other DDS entities such as publishers, subscribers, and topics. The *Publisher* is the entity responsible for disseminating data in DDS and it includes a *Data Writer*. The *Subscriber* is the entity responsible for receiving data and it includes a *Data Reader*. A *Topic* is the entity that represents data objects that are communicated among participants. The *QosPolicy* is the abstract entity that is the root for all DDS QoS policies.

3.2. Modeling QoS Entities

The DDS standard defines 22 QoS policies to support quality data communication. To form comprehensive understanding of those policies, we categorize them into five types as follows:

1. Service configuration, which is concerned with configuring DDS and define its runtime environment;
2. Data delivery, which is concerned with delivering and presenting data samples to remote applications;

3. Data availability, which is concerned with controlling the availability of data in DDS;
4. Data timeliness, which is concerned with distributing data based on the defined time constraints;
5. Resource control, which is concerned with controlling the computing resources such as the memory.

Figure 1 shows the categorization with relationships among QoS policies. In the figure, each policy is defined as a class inheriting the abstract *QosPolicy* class. We formalize each QoS policy of the categories using OCL expressions based on the class diagrams. We first define a class diagram for each category. The class diagram is then used as the base for describing OCL expressions for the QoS policies involved in the category. We use the term *service* interchangeably with DDS and the *application* term refers to an application using DDS. Due to the large number of QoS policies and limitations on paper's size, we choose one representative QoS policy for each category to describe and apply the proposed approach. Consequently, the approach in this study can be generalized to all DDS QoS policies.

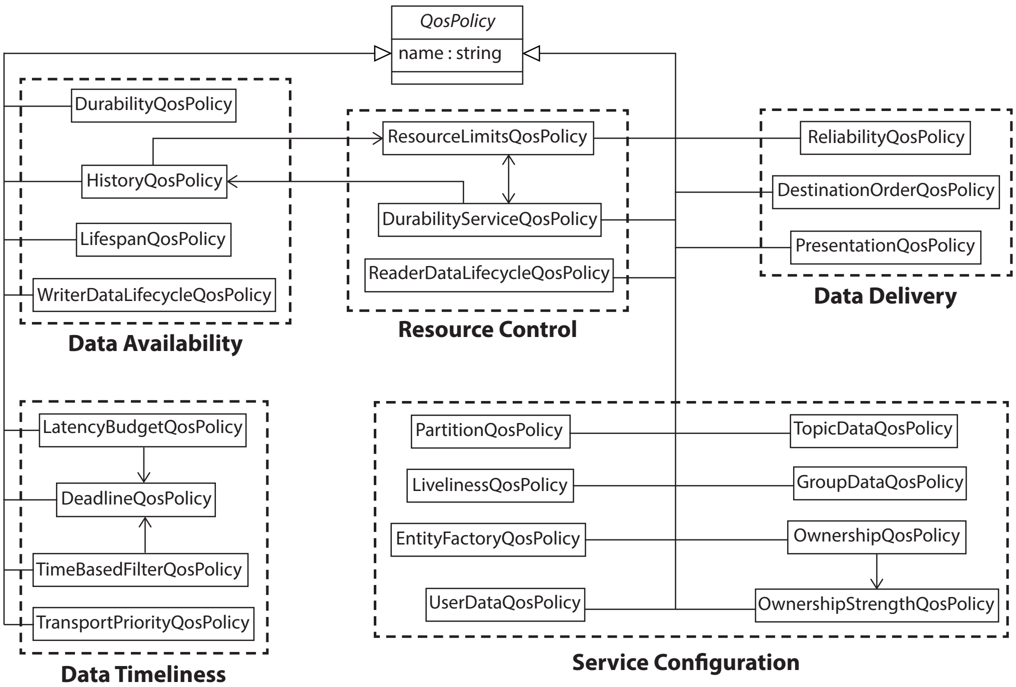


Figure 1. Different categories of DDS QoS policies.

3.2.1. Service Configuration

This category is concerned with the QoS policies that are used for configuring the service and define the DDS runtime environment. It involves *PartitionQosPolicy*, *LivelinessQosPolicy*, *EntityFactoryQosPolicy*, *UserDataQosPolicy*, *TopicDataQosPolicy*, *GroupDataQosPolicy*, *OwnershipQosPolicy*, and *OwnershipStrengthQosPolicy*. These policies are based on the class diagram shown in Figure 2. The class diagram contains classes corresponding to the involved policies and they are related to the *DomainParticipants*, *Entity*, *TopicDataQosPolicy*, *Publisher*, *Subscriber*, *Topic*, *DataReader*, and *DataWriter*. It is worth mentioning that in the following class diagrams the number "1" and the symbol "*" denote multiplicity, which defines cardinality among the objects.

User_Data

This policy is used to attach additional information about the application, such as security and access credentials. The information may be specific to the components (e.g., data writers and data readers) of the application under consideration. Such information can be used by a remote application for their own purposes, such as authentication. The following shows an OCL expression for the policy, where the symbol ˆ denotes the *hasSent* operation, which results in "true" if a message of the concerned operation is sent.

```
context UserDataQosPolicy inv:
 if pub.pubUserData.value =
 sub.subUserData.value
 then pub^get_discovered_participant_data()
 else pub^ignore_participant()
 endif
```

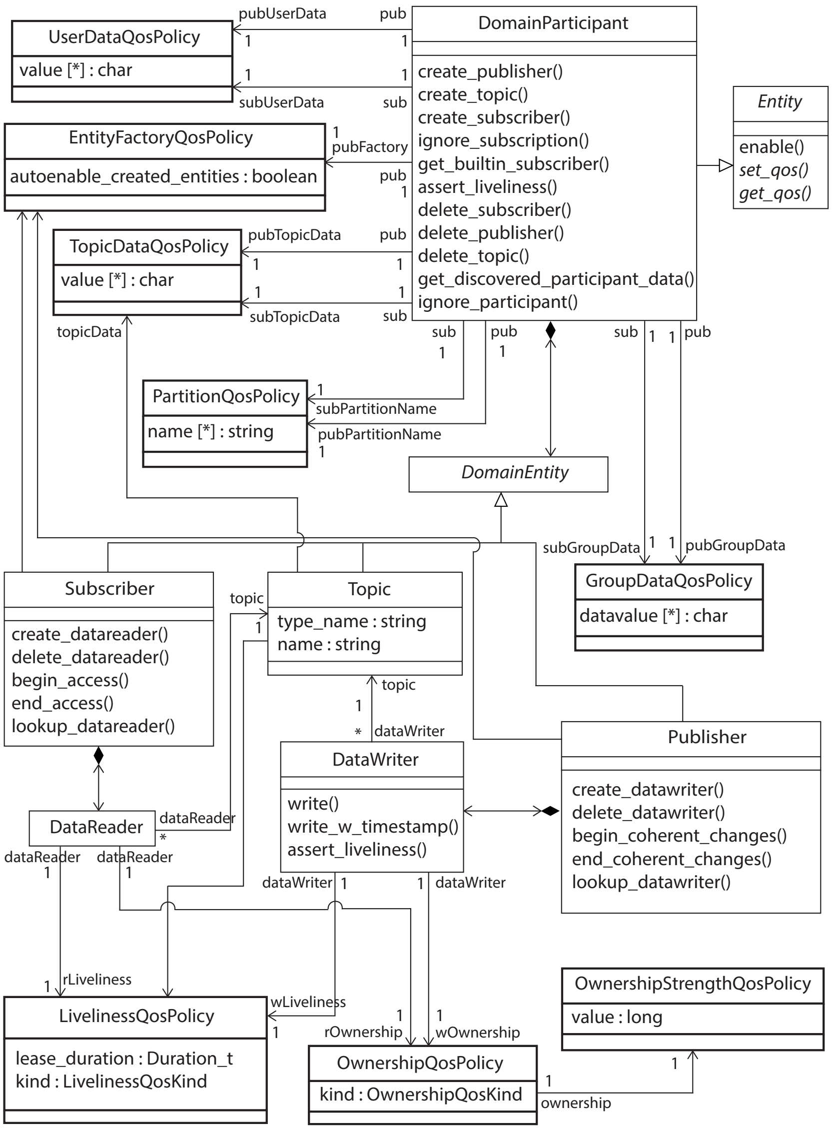


Figure 2. Class diagram of the service configuration.

The expression describes that if the publishing participant and the subscribing participant have the same user data, then the publishing participant accesses the data for communication. If not, the subscribing participant is ignored. Note that the way how the value of *UserDataQosPolicy* is used in the expression may vary depending on the agreement between the publisher and the subscriber. For instance, instead of the equality, non-equal relational operations such as "<" or ">" may be used if agreed by the publisher and subscriber.

3.2.2. Data Delivery

This category describes the set of QoS policies that can be used to define how the DDS should deliver and present data samples to remote applications. Figure 3 shows the class diagram of DDS entities that are related to the QoS policies in this category.

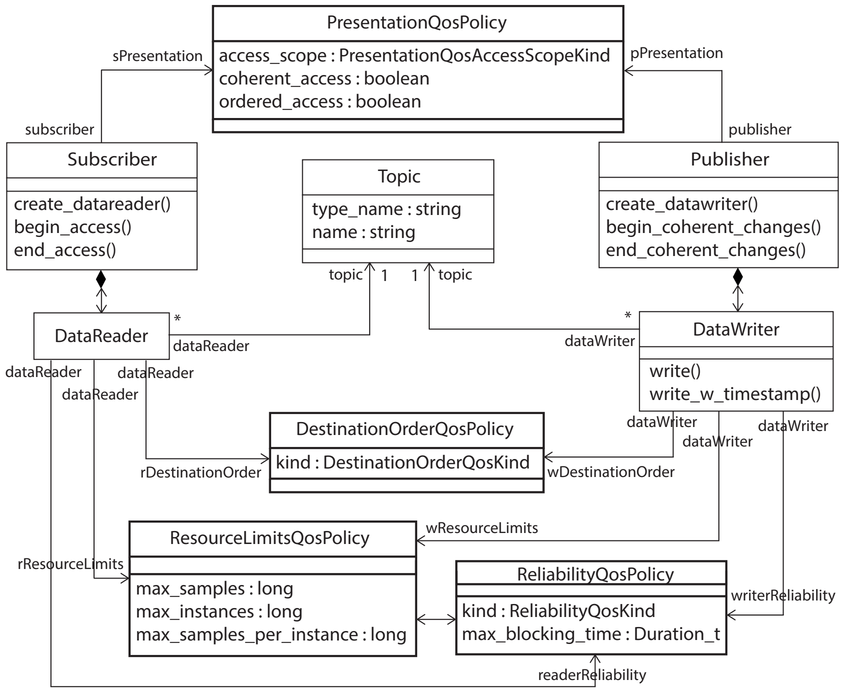


Figure 3. Class diagram of data delivery QoS policies.

Reliability

This QoS policy is concerned with the reliability level of data delivery. It is used to control how data writers and readers should treat data samples. The policy applies to both the data writers and data readers under the same topic. It offers —*BEST_EFFORT* and *RELIABLE*. These settings control the reliability of data delivery. The *BEST_EFFORT* setting allows DDS to perform best effort delivery when communicating data. This means data delivery is not guaranteed. The *RELIABLE* setting ensures data delivery to all interested data readers. Reliable delivery is achieved by resending data until data are received and confirmed by all data readers via acknowledgements.

The attribute *kind* in the *ReliabilityQosPolicy* class is used to set the reliability type. If the reliability is set to *RELIABLE* setting, then the service blocks the data writer from writing further data sample until the receipt of the previous sample has been confirmed. The *Reliability* policy is dependent on the *ResourceLimits* policy. This is because in case of the *RELIABLE* setting, the limits specified by the *ResourceLimits* policy can control the number of samples and the maximum blocking time of this policy. The *RELIABLE* setting is considered to be higher and it overrides the *BEST_EFFORT* setting. To associate a data writer and a data reader, the offered *kind* must be higher than or equal to the requested *kind*, as illustrated by the following OCL constraint.

```
context ReliabilityQosPolicy inv:
dataWriter.topic.dataReader ->
forAll(dr|dr.readerReliability
.kind <= writerReliability.kind) and
writerReliability.kind = RELIABLE implies
wResourceLimits.max_samples_per_instance
= LENGTH_UNLIMITED
```

The expression specifies that in order to associate readers with a writer, the *kind* of the readers must be less than or equal to the *kind* of a writer. Additionally, if the *kind* of the writer is set to *RELIABLE*, then the *max_samples_per_instance* must be set to *LENGH_UNLIMITED*, specifying that there is no limit on the length of the data sample.

3.2.3. Data Availability

This category describes the set of QoS policies that can be used to define how the service should control the availability of data. The QoS policies under this category control aspects, such as queuing and storing data. Figure 4 shows the class diagram of DDS entities that are related to the QoS policies in this category.

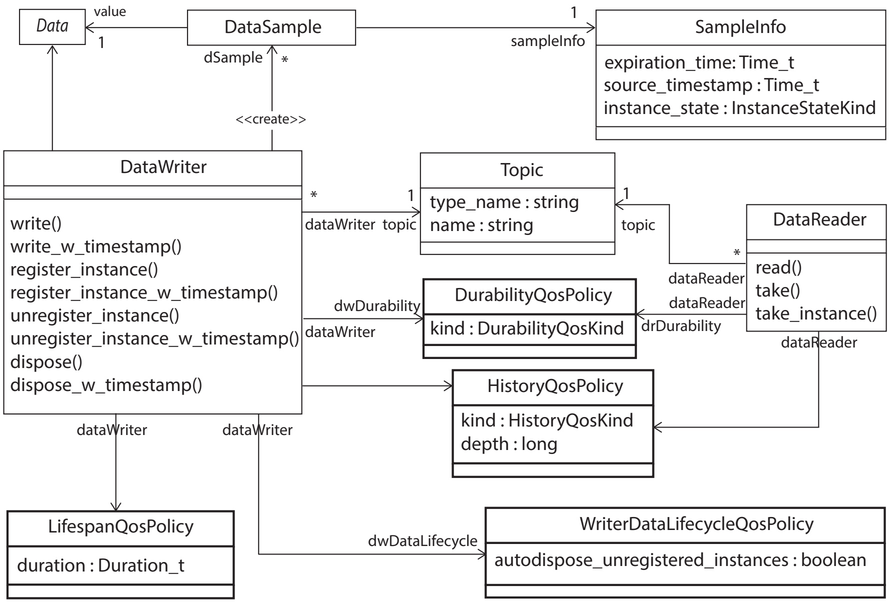


Figure 4. Class diagram of data availability QoS policies.

History

This QoS policy is used to control the behavior of the service when the value of a topic instance keeps changing before they are communicated to all interested data readers. To illustrate, it specifies the maximum number of data samples to be kept in an entity. Different settings can be used with this policy. For example, data values can be kept until a publisher retrieves and delivers them to interested data readers. Regarding the number of kept data samples, this policy provides two settings, which are *KEEP_LAST* and *KEEP_ALL*.

The attribute *kind* in the class *HistoryQosPolicy* is used to specify one of settings of this policy. The *KEEP_LAST* setting allows the service to keep the latest values of the topic instance and it will discard older values. If this setting is chosen, the attribute *depth* can be used to specify the maximum number of data values. On the other hand, the *KEEP_ALL* setting allows the service to keep all data values until they are delivered to all interested data readers. This QoS policy depends on the *ResourceLimits* policy. Therefore, the setting of the policy should be consistent with the setting of the *ResourceLimits* policy. To illustrate, the attribute *depth* of this policy should be consistent with the attribute *max_samples_per_instance* of the *ResourceLimits* policy. For consistency, the *max_samples_per_instance* attribute must be greater than or equal to *depth*. The following OCL expression describes the constraint.

```
context HistoryQosPolicy inv:
participant.kind = KEEPLAST implies
depth =< ResourceLimits.
max_samples_per_instance and
depth = maximumNumber
```

The expression specifies that if the policy is set to *KEEP_LAST*, then the *depth* must be less than or equal to the *max_samples_per_instance* attribute in the *ResourceLimits* policy.

3.2.4. Data Timeliness

This category describes the set of QoS policies that can be used to define the latency of the distributed data. The policies under this category define latency at different levels. For example, some define latency at the DDS level (e.g., *Deadline*), while others define latency at the transport layer (*Transport_Priority*). Figure 5 shows the class diagram of DDS entities that are related to the QoS policies in this category.

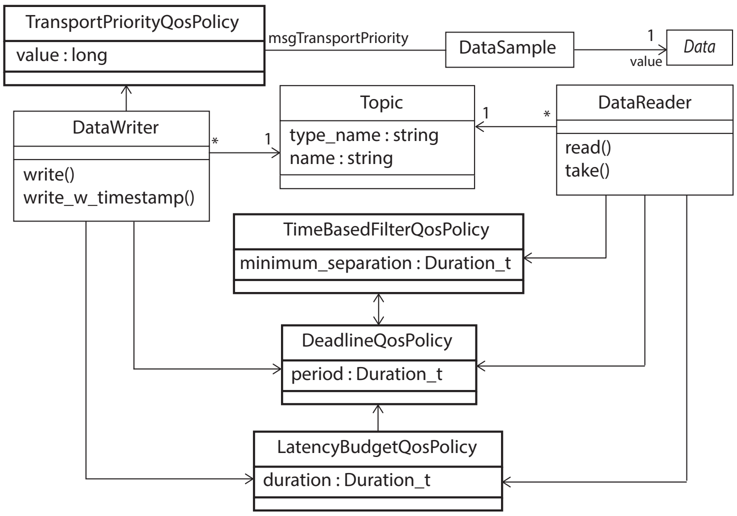


Figure 5. Class diagram of data timeliness QoS policies.

Deadline

This QoS policy can be used to specify the frequency of writing data samples by a data writer. The rate that is set by the data writer represents the minimum required frequency of a data reader. Figure 5 shows the class diagram of this policy. The *period* attribute specifies the writing frequency. It is necessary that the writing frequency at the data writer side must be less than or equal to the period that is specified by the data reader. This is specified by the following OCL expression.

```
context DeadlineQosPolicy inv:
    participant.Topic.DataWriter ->
    forAll(dReader|dReader.Deadline.period
    >= dWriter.Deadline.period)
```

On the data reader, the *Deadline* policy should be consistent with the *TimeBasedFilter*. This means that the frequency of receiving data on a data reader must be greater than or equal to the time interval between data samples specified by the *minimum_separation* attribute in the *TimeBasedFilterQosPolicy* class, as specified by the following OCL expression.

```
context DataReader inv:
    participant.Deadline.period >=
     participant.minimum_separation.
    TimeBasedFilter
```

3.2.5. Resource Control

This category describes the set of QoS policies that can be used to control the computing resources (i.e., memory). The policies under this category define memory related constraints.

Figure 6 shows the class diagram of DDS entities that are related to the QoS policies in this category.

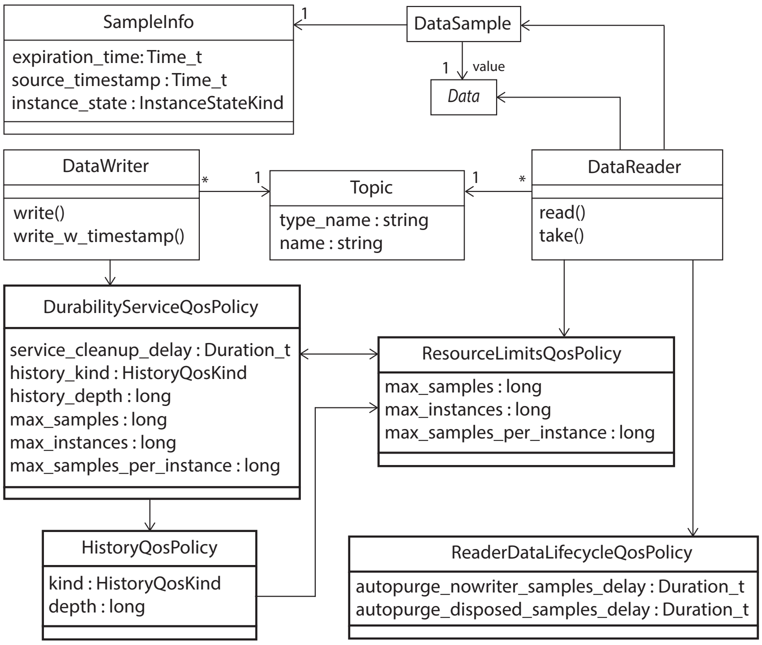


Figure 6. Class diagram of resource control QoS policies.

Durability_Service

This QoS policy is used in conjunction with the *Durability* QoS policy to specify the behavior of deleting data samples from a data writer cache. It controls the deletion behavior when durability is set to *TRANSIENT* or *PERSISTENT*. The policy can also provide a way to specify the settings of the *History* and *ResourceLimits* policies, since they have similar characteristics. The setting of these three policies should be compatible. The following OCL expression describes some constraints based on the properties of the above three QoS policies.

```
context DurabilityServiceQosPolicy inv:
    self.service_cleanup_delay > 0 and
    self.History.kind =
    KEEP_LAST_HISTORY_QOS implies
    self.history_depth > 0 and
     self.ResourceLimits.
    max_samples >= self.history_depth and
     ResourceLimits.
    max_samples_per_instance >=
    self.history_depth
```

The constraints are defined on the *DurabilityServiceQosPolicy* class, where the attribute *service_cleanup_delay* is used to specify the time duration for deleting samples from a data writer. The expression specifies that if the *service_cleanup_delay* is set for a specific duration and it is set to keep the last history, then the depth in the history policy must be greater than zero. It also specifies that the maximum samples to be managed by a data writer (i.e., specified by *ResourceLimitsQosPolicy* class) should be greater than or equal to the *depth* specified by the *HistoryQosPolicy* class.

4. Modeling QoS Features and the Management Layer

Managing the QoS policies of DDS is a complex task, especially in a distributed environment such as a smart grid and with renewable energy resources [27–30]. This is due to several factors, such as device heterogeneity and domain requirements [31]. To simplify the implementation of DDS QoS policies in smart grid domains, we adopt feature modeling [32], where we describe QoS attributes in terms of features. We also propose a QoS management layer in which we control QoS integrity, tuning, and reconfiguration. The rest of this section describes the feature modeling and the QoS management layer based on the defined structural relations in Section 3.

4.1. Feature Modeling

A feature can be described as an integration of one or multiple attributes to represent a functional unit. Modeling the QoS policies into features allows an application to tailor the nonfunctional requirements of DDS by selecting and integrating necessary features based on the requirements of the application. Consequently, multiple benefits can be achieved to improve the performance of an application. For instance, realizing the number of features and the required resources for each feature allows for adopting DDS even with constrained devices. Additionally, understanding features composition allows for selecting only required QoS attributes that serve the application's main objectives. In fact, designing DDS QoS policies into features improves the understanding of the behavior of each QoS policy and its relationship with other policies.

Figure 7 illustrates the proposed feature model of DDS QoS policies based on the categorization defined in Figure 1. The model defines the above five Categorize (i.e., *Data Availability*, *Resource Control*, *Data Delivery Service Configuration*, and *Data Timeliness*). Each feature is composed of multiple child features. The filled triangle under each main feature represents inclusive selection. This means multiple features can be chosen together. Exclusive selection is denoted using the empty triangle (e.g., the *kind* feature can be either *automatic* or *manual*). The filled circles represent mandatory features, while the empty circle denotes optional features. The dashed line can be used to represent dependency among features. Its worth noting that due to limitation on paper size, the feature model in Figure 7 has been reduced in size. The set of features under each category are represented in Figure 1.

Based on the above-defined feature model, we use FeatureIDE [33] to implement the defined models in Section 3. FeatureIDE is a tool that can be used in a form of plug-in with the Eclipse integrated development environment (IDE). It is considered a feature-oriented development tool. Figure 8 shows a partial implementation of the feature model that is defined in Figure 7. In this implementation, two feature groups are considered, which are *DataDelivery* and *ResourceControl*. In this study, the QoS policies are designed based on inheritance and overriding. Inheritance represents the relationship between a parent feature and a child feature, and it is defined below.

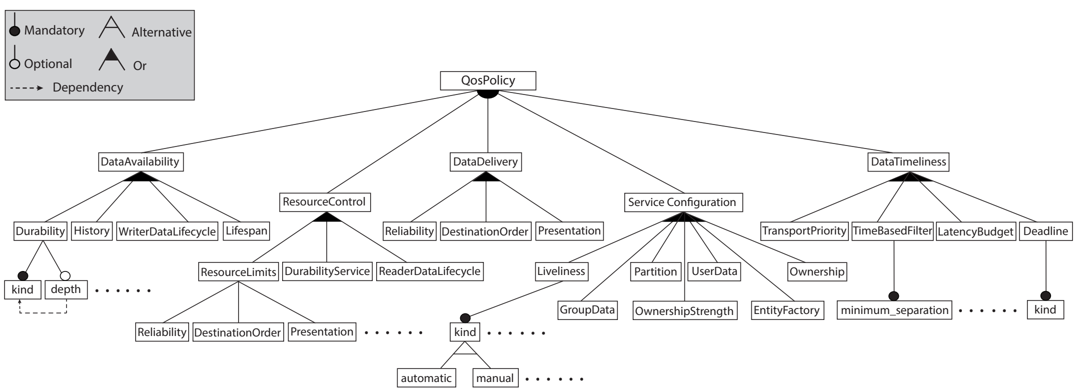


Figure 7. Feature model of DDS QoS policies based on defined categories.

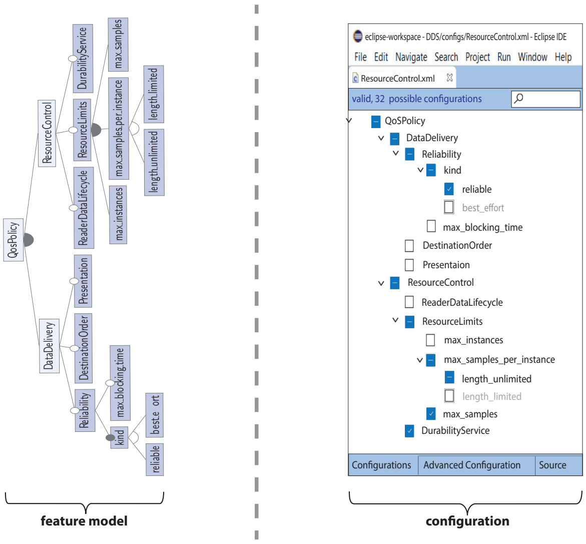


Figure 8. Implementation of *DataDelivery* and *ResourceControl* in FeatureIDE.

Definition 1. *The $cfeature$ and the $pfeature$ relationship*

$$\forall cls' : class(pfeature) \exists cls'' : class(cfeature) \bullet cls' = cls'';$$
$$\forall r' : relation(pfeature) \exists cls'' : rel(cfeature) \bullet r' = r'' \wedge \forall entity' :$$
$$end(r') \exists entity'' : end(r'') \bullet entity' = entity'';$$

Definition 2. *The $cfeature$ overrides $pfeature$ iff*

$$\exists element' : sd(pf), overrides'' : sd(cf) \bullet name(overrides') = name(overrides'');$$
$$\exists b' : bd(pfeature), b'' : bd(cfeature) \bullet name(b') = name(b'');$$

Definition 1 specifies when a child feature inherits a parent feature. In the definition, $end(r)$ represents the set of the ends in a given relationship r. On the other hand, Definition 2 defines the override process. It specifies when a child feature overrides a parent feature. The *element* represents a name of an entity. Additionally, sd represents the structural properties the feature, which is defined as $class(f) \cup relations(f)$. The bd represents the behavioral properties of a feature.

4.2. DDS QoS Management Layer

Managing the QoS policies in DDS is a complex task due to several factors. For instance, different QoS policies are tightly coupled, which means controlling a policy might affect other policies [34]. Additionally, understanding the behavior of each policy is a key aspect in adopting any QoS policy [35]. Furthermore, since the QoS policies in DDS are designed to address different concerns (e.g., reliability, latency), their control needs to be established at different layers (e.g., application, transport, network). In fact, adopting one QoS policy requires dealing with multiple DDS entities. Thus, we propose a DDS QoS management layer (QoSML). QoSML is designed to manage the QoS policies to ease the development of DDS. QoSML is based on two software design principles, which are modularity and extendibility. The layer serves multiple purposes including improving the efficiency of hardware resources (i.e., CPU, response time) and to reduce QoS configuration complexity.

One of the major functionalities of QoSML is managing QoS dependency. This is accomplished through defining the structural relationships among DDS QoS policies. QoSML is also used to define and implement the feature control approach that is provided in Section 3. To illustrate, the layer is used to hold QoS management data. These data are defined and controlled in the layer in a form of metadata. Two metadata types are defined. The first, is the metadata that defines the dependency and the structural relations among the QoS policies. This type works as a descriptive file that combines DDS entities and QoS policies and defines their relationships and constraints. The second type is the metadata that defines the feature modeling configurations. The latter works as a configuration file to facilitate QoS agreements in each communication scenario. Configuration metadata are generated from FeatureIDE in of XML files. QoSML is designed to be a part of a DDS layer, as shown in Figure 9. This means that the application layer has direct access to QoSML.

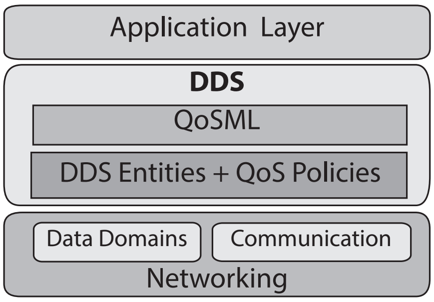


Figure 9. Different layers in DDS and placement of QoSML.

As mentioned above, QoSML is proposed to control QoS using metadata and configuration files. In QoSML, QoS control is performed when the application is idle to improve the efficiency of hardware resources. This means that the layer adopts a predictive metadata reconfiguration mechanism. Algorithm 1 shows the process of predicting metadata update at a publisher or subscriber. The process starts when a publisher/subscriber receives data interest request. This request is generated and exchanged between applications in the form of topic publication/subscription. Metadate files are updated regularly based on a preconfigured frequency threshold to ensure the consistency of QoS among communicating applications. The threshold can be set based on the frequency of data exchanged or other factors, such as the number of new publishing/subscribing applications in a data domain.

Algorithm 1 Updating dependency and features metadata

procedure MDU(**Input**: participant ID, **Output**: new metadata)
    while participant is on **do**
    if participant is new **then**
        create metadata files for participant
    else
        if QoS policies requirements changed **then**
            update QoS policies dependency metadata
            update feature configuration metadata
            calculate MUR_t at time t
            if $MUR_t < UTF$ **then**
                change metadata
                change metadata UTF to new UTF_t
            else
                keep old UTF
            end if
        end if
    end if
end procedure

The algorithm takes as input the ID of the new participant and compares it with the set of preexisting participant IDs. If it is a new participant (i.e., not registered before), a structural relations metadata file is generated and stored. Additionally, configuration metadata based on selected QoS policies is generated and stored. If the participant has been previously registered with the publisher/subscriber, the algorithm looks the pre-generated metadata files and match them based on the participant ID. If the metadata update request (MUR) at a given time (t) is less than the update threshold frequency (UTF), we change the frequency value (i.e., UTF) to the new update rate (i.e., MUR). Then, we update the metadata files for the participant. Otherwise, we keep the old update frequency rate and discard the update request. The goal of this prediction algorithm is to improve the response time and optimize computing resources during communication runtime. In fact, this model can be used to measure the precision bound of the update threshold, which can contribute to improving the overall communication in DDS.

5. Use Case and Validation

This section presents an overview of the experiments' setup. It describes the simulated domain where DDS is implemented in the consumption domain in a smart grid to monitor energy consumption. It also describes the implementation of QoSML. Finally, the section provides performance results based on the defined use case.

5.1. Experiment Components and Setup

To test the proposed approach, we build an environment to simulate sensor communication in the smart grid consumption domain. The exchanged data in the communication scenarios represent metering and sensing data. The exchanged data are distributed to multiple interested entities (e.g., grid operators, utility centers). The data are used for different purposes, such as estimating power demand and evaluating power utilization. Figure 10 shows different DDS communication scenarios. In the figure, it can be seen that one device can be connected to multiple others. This means that one type of data (e.g., sensor reading, meter reading) might be required by multiple interested entities. This is typical in a smart grid environment, since the main concept is to connect multiple applications with each other to maximize connectivity and enhance domain knowledge [28,36–38]. The figure also shows that there are two types of DDS-based communications, i.e., wired and wireless. In the wired setting, sensor devices are connected to each other via Ethernet. On the other hand, Wi-Fi is used for wireless sensor communication.

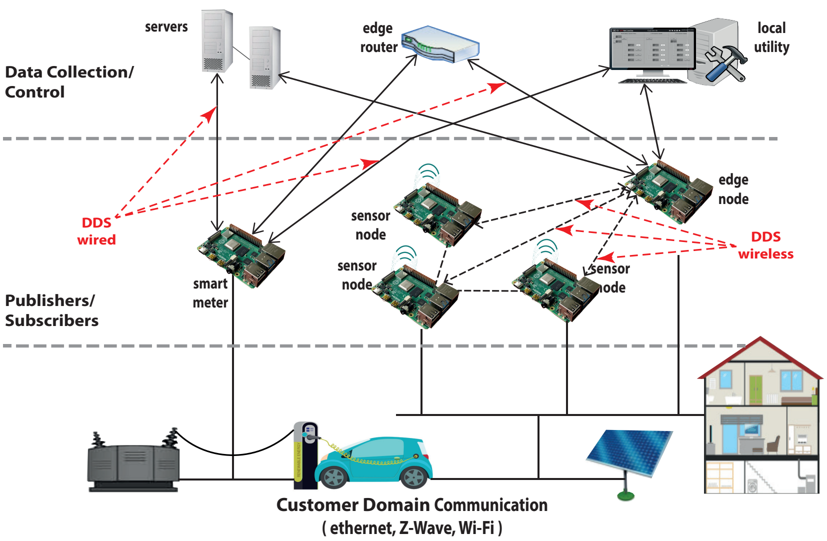


Figure 10. Communication scenarios in consumption domain using DDS.

In the experiment, multiple devices with different hardware capabilities are used. The hardware components are based on Raspberry Pi 4 with different memory size, ranging from 1GB–4GB. The devices communicate to simulate sensors that exchange equipment temperature and microelectromechanical pressure reading. Additionally, other devices communicate to simulate smart meters that exchange data with utilities. All the communication in the experiments is based on DDS. The devices are configured to act as publishers, subscribers, or both. Sensor nodes communicate their data wirelessly using IEEE 802.15.4 standard [39]. On the other hand, edge nodes and smart meters communicate their data based on Ethernet connection. As seen from the figure, the data flow in a bidirectional manner, which means publisher nodes can also act as subscribers and vice versa.

5.2. QoS Features Analysis

Given the multiple categorizations of the QoS policies, we select *DataDelivery* and *ResourceControl* to validate the approach. We perform the dependency analysis and structural relation with the defined feature model of the QoS policies. Figure 8 shows the implemented features using FeatureIDE. Given this implementation, we rigorously test the above definitions (i.e., Definitions 1 and 2) against the possible QoS configurations. The purpose of this test is to discover any conflict in feature compositions. We observe any invalid configurations in these feature groups. Invalid composition results in an unexpected or wrong behavior. This may result in communication hindering or even communication failure. The overall purpose of the feature analysis test is to observe the impact of the rigorous formalization of DDS QoS policies.

Table 1 shows seven feature composition tests. The first column in the table represents the group number, which is used to define the selected features. The second column shows the composed features. The third illustrates the possible configuration conflicts. The latter represents the internal invalid behavior that may arise from composing the specified QoS features in the group. The number shown represents the average number of invalid conflicts. This means that the actual QoS conflicts may be higher than the represented number in front of each group. The table shows that composing one group of QoS features may result in significant errors if not performed correctly. For instance, consider the case with *Group 1*, where four features (*Reliability*, *ResourceLimits*, *Presentation*, and *DestinationOrder*) are composed. If this composition is performed incorrectly, 32 errors may arise, causing unexpected behavior at runtime. The number of errors depends on the relationships and the dependencies among the QoS features. The results in the table show only a subset of DDS QoS policies. These experiments show the importance of defining the dependency and structural relations that is presented in this study. In fact, without the proposed formal definition, different implementations of DDS may not be compatible and runtime errors are most likely to happen.

Table 1. Composed features of DataDelivery and ResourceControl.

Group #	Composed QoS Features	Possible QoS Conflicts
1	*Reliability* & *ResourceLimits* & *Presentation* & *DestinationOrder*	32
2	*Reliability* & *DestinationOrder* & *ReaderDataLifecycle*	8
3	*Reliability* & *DurabilityService* & *ReaderDataLifecycle*	16
4	*Reliability* & *DurabilityService* & *ResourceLimits* & *Presentation*	48
5	*DestinationOrder* & *ResourceLimits* & *Presentation*	16
6	*Reliability* & *DurabilityService* & *Presentation*	12
7	*Reliability* & *ReaderDataLifecycle* & *ResourceLimits*	12

5.3. Resource and Performance Testing

Figure 11 illustrates the different layers that are implemented on each sensor node in this performance testing. In the figure, the application layer is used to perform data sensing management and aggregation. QoSML is used to handle feature configuration, QoS tuning management, and on demand re-configuration. Based on the chosen configurations, metadata files are pre-generated before communication takes place according to the process described in Algorithm 1. Metadata files are totally generated, stored, and managed solely on this layer. This layer is designed to work on conjunction with DDS. This means that an application has direct access to QoSML to perform configuration/re-configuration. Consequently, this improves processing performance and response time, since the layer generates and stores QoS metadata in advance. The figure also shows the DDS layer, where the core functional and non-functional components are implemented.

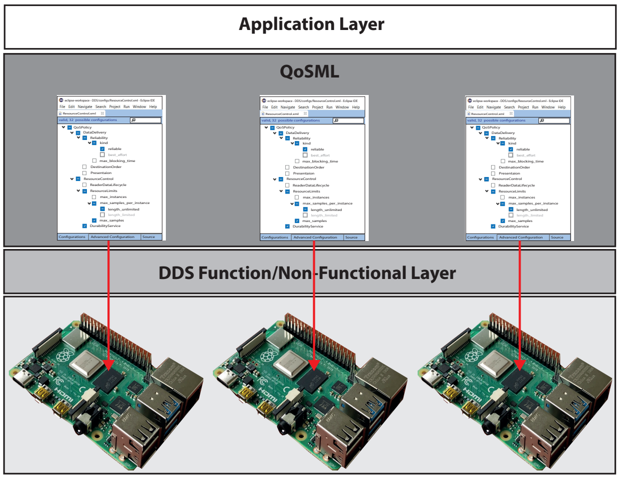
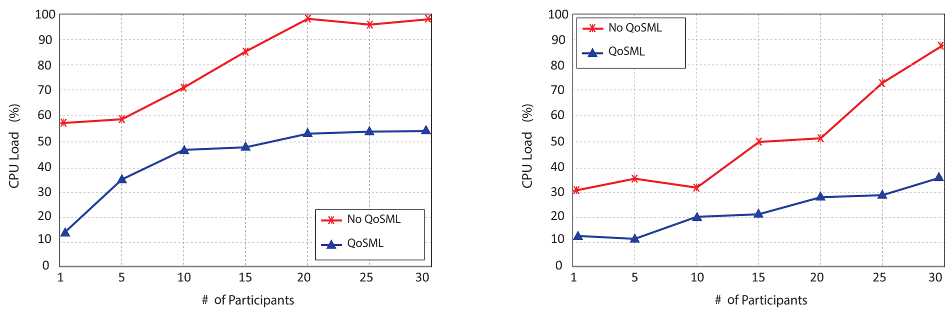


Figure 11. Structure of the implemented approach on sensor nodes.

Figure 12 shows CPU loads on the DDS publisher based on wireless communication (i.e., Wi-Fi). The figure shows CPU loads on the publisher when sending data to multiple participants, ranging from 1 to 30. Graph (a) illustrates the performance when the publisher is set to communicate data based on *Reliable* setting. The CPU loads are measured with and without QoSML. It can be seen from the graph that CPU loads are significantly improved when QoSML is used. For instance, with the largest number (i.e., 30) of interested subscribers, the CPU load with QoSML is only 65% compared to 100% when QoSML is not used. Similar performance results are shown in graph (b) when the publisher is set to communicate data based on the *Best Effort* setting.

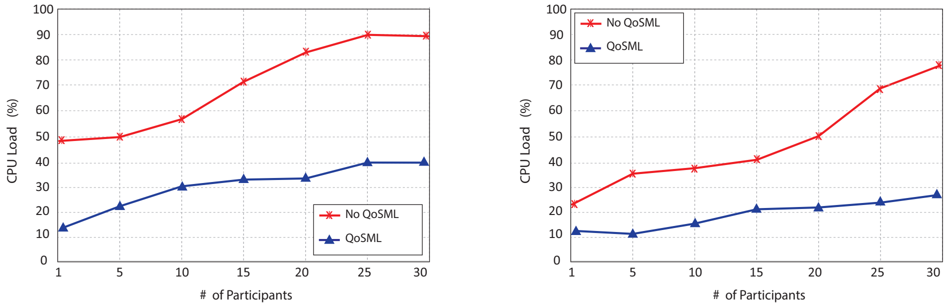


(**a**) Reliable setting (**b**) Best effort setting

Figure 12. CPU load on the DDS publisher (wireless communication).

From both graphs in Figure 12, it can be seen that the CPU loads with the *Reliable* setting are higher than the loads with *Best Effort* setting. This is because the *Reliable* setting is computation dependent, which results in higher CPU consumption. Thus, CPU performance is improved with QoSML, since the layer provides pre-configuration services using stored QoS metadata. This significantly lowers CPU loads and improves response time, since most of the reconfiguration and metadata management work is done IN advance.

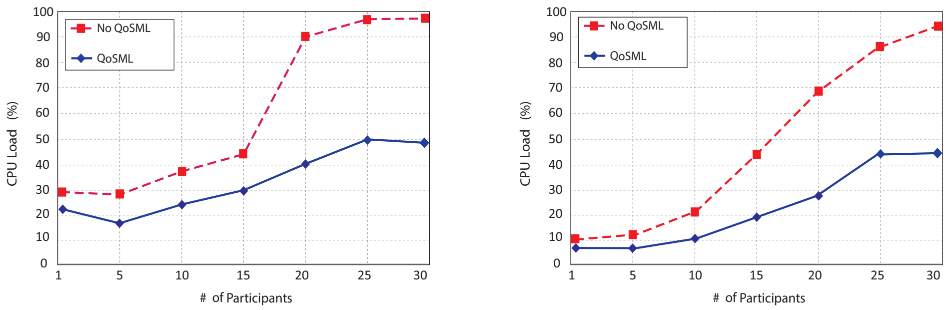
Figure 13 illustrates CPU loads of the publisher when DDS is set to communicate based on Ethernet. As the above, the publisher sends data to the same range of interested participants. Graph (a) shows THE publisher's CPU loads with the *Reliable* setting. On the other hand, graph (b) illustrates the loads with the *Best Effort* setting. Similar to the above, QoSML significantly contributes to lowering the CPU loads in the publisher device with both settings. For example, with the largest interested subscribers and with the *Reliable* setting, the highest CPU load with 30 participant is 89% when QoSML is not adopted. This is compared to only 40% of CPU load when QoSML is adopted. From both Figures 12 and 13, it can be seen that when the number of interested subscriber increases, the CPU loads increase as well. It can also be seen that QoSML contributes to reducing CPU loads by half of the amount when QoSML is not implemented.



(**a**) Reliable setting (**b**) Best effort setting

Figure 13. CPU load on the DDS publisher (wired communication).

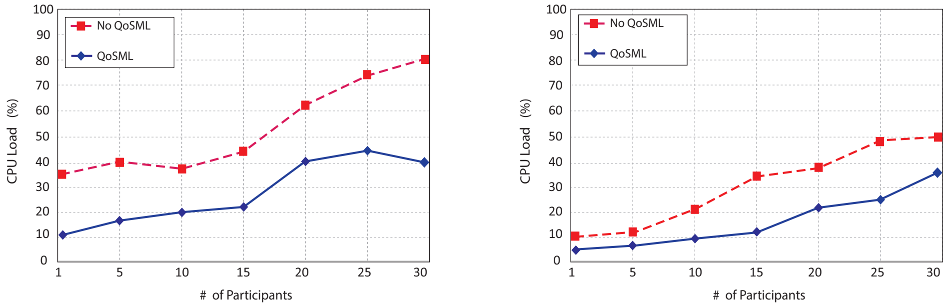
Figure 14 shows the CPU loads of DDS subscriber based on Wi-Fi connectivity. Graph (a) shows CPU loads on the subscriber based on the *Reliable* setting. It illustrates CPU loads with and without QoSML. Improved CPU loads are seen in the graph when QoSML is adopted. For example, with the largest number of participants (i.e., data received from multiple publishers), the CPU load with QoSML is only 49%. This is compared to 100% when QoSML is not used. Graph (b) shows similar performance when the subscriber is set to *Best Effort*. In fact, QoSML reduces CPU loads by using metadata, which does not require the subscriber to check the QoS setting upon receiving new data.



(**a**) Reliable setting (**b**) Best effort setting

Figure 14. CPU load on DDS subscriber (wireless communication).

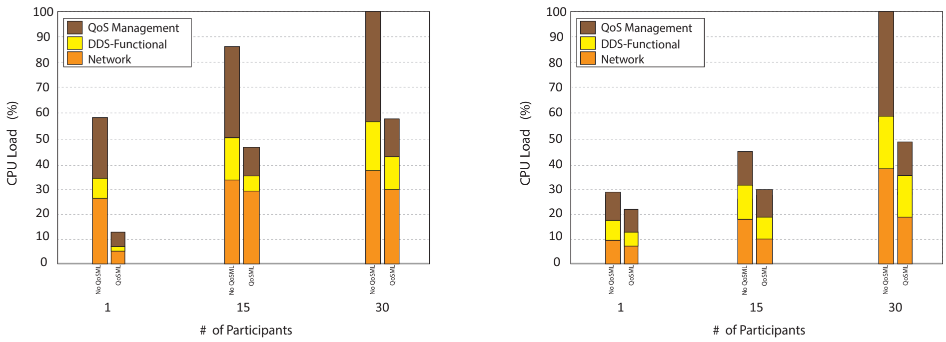
Figure 15 shows the CPU loads on the subscriber when it is set to communicate data based on Ethernet. Similar to the above experiments, the figure shows two settings, which are *Reliable* and *Best Effort*. Additionally, for each setting, the CPU loads are measured with and without QoSML. As illustrated in the figure, QoSML reduces CPU loads and improves response time. From both Figures 14 and 15, it can be seen that CPU loads increase when Wi-Fi connection is used compared to Ethernet. Additionally, it is seen that the CPU loads with Wi-Fi communication increase rapidly as the number of participants increases. On the other hand, with Ethernet connection, CPU loads increase but not as rapid as with Wi-Fi. This means that with Ethernet, we observe a linear increase. This is because Ethernet communication does not require extensive reliability management, which is due to the nature of Ethernet. Overall, QoSML contributes to reducing CPU loads, as it uses prediction, metadata management, and pre-configuration.



(**a**) Reliable setting (**b**) Best effort setting

Figure 15. CPU load on DDS subscriber (wired communication).

To further test the approach, we study the impact of the proposed QoSML on reducing CPU loads to improve communication efficiency. We analyze the highest CPU loads. Those loads happen to be for the *Reliable* settings with both publisher and subscriber for both wired and wireless communication. Figure 16 shows the CPU load breakdown with wireless communication. In the figure, the loads are shown for three sampling groups based on the number of participants—when data are sent/received from 1, 15, and 30 participants. For each group, CPU loads are observed with and without the utilization of QoSML. Based on the observation, CPU loads are categorized into three main tasks. The first is the task related to the functional performance of DDS (e.g., sending/receiving/sorting data). The second is the task related to QoS management (e.g., setting, tuning). The third is the task related to network management.

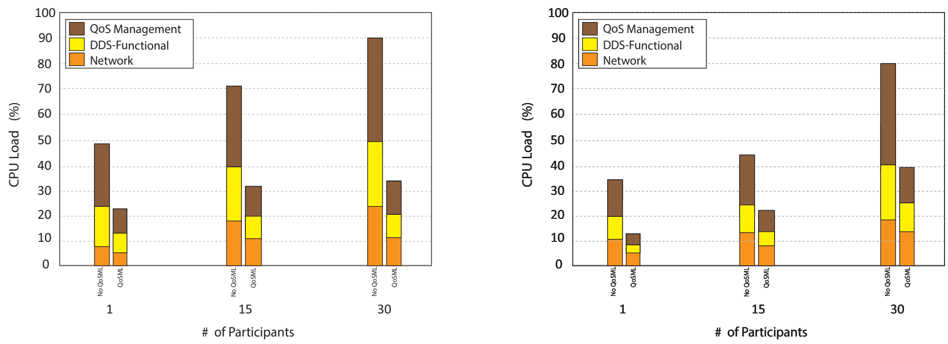


(**a**) Reliable publisher (**b**) Reliable subscriber

Figure 16. Illustration of CPU load (wireless communication).

Graph (a) in Figure 16 shows the illustration of the CPU load breakdown with the *Reliable* setting for the publisher. It shows that when DDS is implemented with no QoSML, the CPU loads are significantly higher than the loads with QoSML. It also shows that QoS management task amounts to about 50% of the overall CPU load for each group. Additionally, it can be seen that the rise of the CPU loads for the QoS management task has a direct proportional relationship with the network task. This can be justified since reliable communication in DDS is a network-oriented task. Graph (b) shows the illustration for the CPU loads for the reliable subscriber. Similar behavior and breakdown is observed. From both figures, we can conclude that the utilization of QoSML reduces CPU loads, since it manages QoS through pre-configuration and stored QoS metadata, which does not require the publisher/subscriber to check QoS compliance each time data are sent or received.

Figure 17 shows the CPU breakdown based on wired communication. Similarly, in this test, the highest CPU load have been chosen. These CPU loads represent the *Reliable* setting for both publisher and subscriber. Graph (a) illustrates the CPU loads breakdown for the publisher. The graph shows that the proposed QoSML reduces CPU loads by about 55% on average. The graph also shows that the QoS management represents the highest CPU load of the overall load when QoSML is not adopted. This is because at each communication attempt (i.e., sending/receiving data), the QoS policies need to be validated, which results in extra CPU loads. Graph (b) shows similar CPU load breakdown behavior. It also shows that when QoSML is adopted, it significantly reduces the task related to QoS management.



(**a**) Reliable publisher (**b**) Reliable subscriber

Figure 17. Illustration of the CPU load on wired communication.

6. Conclusions

In this study, we have presented modeling DDS QoS policies to improve connectivity in distributed smart grid applications. We first formalize the QoS policies in DDS with the objective of improving the completeness of DDS specification. In the formalization process, we categorize the QoS policies into five different groups based on the semantics and the purpose of each policy. We then provide a dependency analysis to illustrate the structural relation among the QoS policies. Given the dependency, we formally describe the constraints imposed by the policies using OCL. Furthermore, we explain the semantics of some policies by providing utilization scenarios. We also propose QoS feature modeling to facilitate DDS development and to ease implementation. Thus, we propose a QoS management layer with the predictive configuration model to improve the efficiency of DDS-enabled applications. We provided use cases where we implement the approach in power consumption applications. The results show that the proposed approach facilitates the development of QoS policies and improves resource utilization. For future research, we plan to study the interaction behaviors of each DDS entity based on the utilized QoS policies. We also plan to generalize the approach so it can be adopted in operation and control applications in smart grids.

Author Contributions: Conceptualization: A.A.; methodology: A.A.; validation: A.A.; formal analysis: A.A.; writing—original draft preparation: A.A.; writing—review and editing: A.A.; supervision: A.A. All authors have read and agreed to the published version of the manuscript.

Funding: The authors extend their appreciation to the Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia for funding this research study through the project number 223202.

Conflicts of Interest: The authors declare that there was no disclosed possible conflict of interest relevant to the research.

References

1. Fang, X.; Misra, S.; Xue, G.; Yang, D. Smart Grid—The New and Improved Power Grid: A Survey. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 944–980. [CrossRef]
2. Olivares, D.E.; Mehrizi-Sani, A.; Etemadi, A.H.; Cañizares, C.A.; Iravani, R.; Kazerani, M.; Hajimiragha, A.H.; Gomis-Bellmunt, O.; Saeedifard, M.; Palma-Behnke, R.; et al. Trends in Microgrid Control. *IEEE Trans. Smart Grid* **2014**, *5*, 1905–1919. [CrossRef]
3. Kim, S.H.; Kim, J.S.; Maeng, S. Modeling and Evaluation of Serial Multicast Remote Procedure Calls (RPCs). *IEEE Commun. Lett.* **2009**, *13*, 283–285.

4. Kim, D.K.; Alaerjan, A.; Lu, L.; Yang, H.; Jang, H. Toward Interoperability of Smart Grids. *IEEE Commun. Mag.* **2017**, *55*, 204–210. [CrossRef]
5. Xylomenos, G.; Ververidis, C.N.; Siris, V.A.; Fotiou, N.; Tsilopoulos, C.; Vasilakos, X.; Katsaros, K.V.; Polyzos, G.C. A Survey of Information-Centric Networking Research. *IEEE Commun. Surv. Tutori.* **2014**, *16*, 1024–1049. [CrossRef]
6. Ekanayake, J.; Liyanage, K.; Wu, J.; Yokoyama, A.; Jenkins, N. *Smart Grid Technology and Application*; Wiley: Hoboken, NJ, USA, 2012.
7. Qazi, A.; Hussain, F.; Rahim, N.A.; Hardaker, G.; Alghazzawi, D.; Shaban, K.; Haruna, K. Towards Sustainable Energy: A Systematic Review of Renewable Energy Sources, Technologies, and Public Opinions. *IEEE Access* **2019**, *7*, 63837–63851. [CrossRef]
8. Tuballa, M.L.; Abundo, M.L. A review of the development of Smart Grid technologies. *Renew. Sustain. Energy Rev.* **2016**, *59*, 710–725. [CrossRef]
9. Sayed, K.; Gabbar, H. SCADA and Smart Energy Grid Control Automation. In *Smart Energy Grid Engineering*; Academic Press: Cambridge, MA, USA, 2017; pp. 481–514.
10. Mahmood, A.; Javaid, N.; Razzaq, S. A review of wireless communications for smart grid. *Renew. Sustain. Energy Rev.* **2015**, *41*, 248–260. [CrossRef]
11. Alaerjan, A. Model-Driven Interoperability Layer for Normalized Connectivity Across Smart Grid Domains. *IEEE Access* **2021**, *9*, 98639–98653. [CrossRef]
12. Su, Y.; Jiang, P.; Chen, H.; Deng, X. A QoS-Guaranteed and Congestion-Controlled SDN Routing Strategy for Smart Grid. *Appl. Sci.* **2022**, *12*, 7629. [CrossRef]
13. Object Management Group. Data Distribution Service (DDS). Technical Report. 2015. Number 2015-04-10. Available online: www.omg.org (accessed on 15 February 2023).
14. Youssef, T.; Hariri, M.; Elsayed, A.T.; Mohammed, O.A. A DDS-Based Energy Management Framework for Small Microgrid Operation and Control. *IEEE Trans. Ind. Inform.* **2017**, *14*, 958–968. [CrossRef]
15. Kang, W.; Kapitanova, K.; Son, S. RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems. *IEEE Trans. Ind. Inform.* **2012**, *8*, 393–405. [CrossRef]
16. Wang, N.; Schmidt, D. Toward an Adaptive Data Distribution Service for Dynamic Large-scale Network-Centric Operation and Warfare (NCOW) Systems. In Proceedings of the Proceedings of IEEE Military Communications Conference, San Diego, CA, USA, 16–19 November 2008; pp. 1–7.
17. Object Management Group. The Real-Time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification. Technical Report. Version 2.2. 2014. Available online: www.omg.org (accessed on 15 February 2023).
18. Esposito, C.; Ciampi, M. On Security in Publish/Subscribe Services: A Survey). *IEEE Commun. Surv. Tutori.* **2015**, *17*, 966–997. [CrossRef]
19. Perez, H.; Gutierrez, J. Modeling the QoS Parameters of DDS for Event-Driven Real-time Applications. *J. Syst. Softw.* **2015**, *104*, 126–140. [CrossRef]
20. Object Management Group. Object Constraint Language. Technical Report. Number 2014-02-03, Version 2.4. 2014. Available online: www.omg.org (accessed on 15 February 2023).
21. Object Management Group. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Technical Report. Number 2011-06-02, Version 1.1. 2011. Available online: www.omg.org (accessed on 15 February 2023).
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; pp. 115–120.
23. Alaerjan, A.; Kim, D. Modeling Functional Behaviors of DDS. In Proceedings of the 17th IEEE International Conference on Scalable Computing and Communications, San Francisco, CA, USA, 4–8 August 2017; pp. 1–7.
24. Object Management Group. OMG Unified Modeling Language. Technical Report. Number 2015-03-01, Version 2.5. 2015. Available online: www.omg.org (accessed on 20 January 2023).
25. Youssef, T.; Elsayed, A.; Mohammed, O. Data Distribution Service-Based Interoperability Framework for Smart Grid Testbed Infrastructure. *Energies* **2016**, *9*, 150. [CrossRef]
26. Shi, K.; Bi, Y.; Jiang, L. Middleware-based Implementation of Smart Microgrid Monitoring Using Data Distribution Service over IP Networks. In Proceedings of the 2014 49th International Universities Power Engineering Conference (UPEC), Cluj-Napoca, Romania, 2–5 September 2014; pp. 1–5.
27. Köksal, O.; Tekinerdogan, B. Obstacles in Data Distribution Service Middleware: A Systematic Review. *Future Gener. Comput. Syst.* **2017**, *68*, 191–210. [CrossRef]
28. Ma, R.; Chen, H.; Huang, Y.; Meng, W. Smart Grid Communication: Its Challenges and Opportunities. *IEEE Trans. Smart Grid* **2013**, *5*, 36–46. [CrossRef]
29. Petersen, B.; Bindner, H.; Poulsen, B.; You, S. Smart Grid Communication Middleware Comparison—Distributed Control Comparison for the Internet of Things. In Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems, Porto, Portugal, 22–24 April 2017; pp. 219–226.
30. Asbery, C.; Jiao, X.; Liao, Y. Implementation Guidance of Smart Grid Communication. In Proceedings of the 2016 North American Power Symposium (NAPS), Denver, CO, USA, 18–20 September 2016; pp. 1–6.
31. NIST. *Framework and Roadmap for Smart Grid Interoperability Standards*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2010.

32. 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.
33. Leich, T.; Apel, S.; Marnitz, L.; Saake, G. Tool Support for Feature-Oriented Software Development—FeatureIDE:An Eclipse-Based Approach. In Proceedings of the Workshop Eclipse Technology Exchange, San Diego, CA, USA, 16–17 October 2005; pp. 55–59.
34. Basem, A.M.; Ali, H. Data Distribution Service (DDS) based implementation of Smart grid devices using ANSI C12.19 standard. In Proceedings of the 12th International Conference on Future Networks and Communications, Fukuoka, Japan, 14–16 June 2017; pp. 394–401.
35. Pardo-Castellote, G. OMG Data Distribution Service: Real-Time Publish/Subscribe Becomes a Standard. *RTC Magazine* **2005**, *14*, 1–3.
36. Grammatikis, P.; Sarigiannidis, P. Securing the Smart Grid: A Comprehensive Compilation of Intrusion Detection and Prevention Systems. *IEEE Access* **2019**, *7*, 46595–46620. [CrossRef]
37. Martínez, J.F.; Rodríguez-Molina, J.; Castillejo, P.; De Diego, R. Middleware Architectures for the Smart Grid: Survey and Challenges in the Foreseeable Future. *Energies* **2013**, *6*, 3593–3620. [CrossRef]
38. Kim, D.; Lee, B.; Kim, S.; Yang, H.; Jang, H.; Hong, D.; Falk, H. QVT-Based Model Transformation to Support Unification of IEC 61850 and IEC 61970. *IEEE Trans. Power Deliv.* **2014**, *29*, 598–606. [CrossRef]
39. *IEEE Std 802.15.4-2011*; IEEE Standard for Local and Metropolitan Area Networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE: Piscataway, NJ, USA, 2011. [CrossRef]