Message Oriented Middleware with QoS Support for Smart Grids

Abdel Rahman Alkhawaja, Luis Lino Ferreira, Michele Albano

CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto
R. Dr. António Bernardino de Almeida, 431
4200-072 Porto / Portugal
{abdel,llf,mialb}@isep.ipp.pt

Abstract. The rapid growth on the adoption of smart grids technologies is enabling the improvement of efficiency, reliability and security on energy distribution and consumption. The efficiency increase comes from the overall monitoring of the electricity network, and from the capability of acting upon loads in order to better adapt to overall and local energy production from traditional sources and renewables. To guarantee that these energy sources can be effectively used, smart grid systems must be able to react quickly and predictably, adapting to changing supply, by controlling loads and energy storage. Many applications have been identified and developed to optimize power grid systems, and these applications rely on a solid communications network that is secure, highly scalable, and always available. Thus, any communication infrastructure for smart grids should support its potential of producing high quantities of real-time data, with the goal of reacting to state changes by actuating on devices in real-time, while providing Quality of Service (QoS) guarantees to the communications. These functionalities can be supported by a Message-Oriented Middleware, which allows interconnecting houses and controlling applications in a distributed environment. Therefore, in this paper we survey and analyze existing middleware solutions for the support of distributed scalable large-scale applications with QoS requirements that are structured on top of a Message oriented Middleware.

1 Introduction

The rapid growth on the adoption of smart grids technologies is enabling the improvement of efficiency, reliability and security on energy distribution and consumption. The efficiency increase comes from the overall monitoring of the electricity network, the capability of acting upon loads in order to better adapt to overall and local energy production from traditional and renewable sources. The increase in reliability and security arises from the capability of real-time monitoring and actuation over the network.

An intrinsic problem of this scenario is that energy produced from renewable sources is affected by fluctuating weather factors. To guarantee that these energy sources can

be effectively used, smart grid systems must be able to react quickly to changing supply, by controlling power loads and energy storage.
In smart grid systems, distributed applications need to interact with each other to exchange data among different platforms with different types of information systems. The management of these systems is an important aspect especially with the increasing number of users, applications, services and information sources.
Many applications have been identified and developed to optimize power grid systems. Typically, these applications interact with each other and with the surrounding environment, which is composed by the energy-consuming appliances of Home Area Networks (HAN), the energy-producing devices that harvest energy from renewable sources, and the energy market that enables the user to buy and sell energy to reach an equilibrium that can power appliances with minimal energy waste. The changes in the environment can happen at a high rate, and some of the events are more important than others, hence these applications rely on a reliable communications network equipped with QoS characteristics, like security, scalability, real-time operation and high availability. Such a communication network must support high quantities of real-time data, and actuating on devices in real-time [2].
The European ENCOURAGE [21] project, which is the driving force behind this work, addresses the development of adequate technologies for the optimization of energy consumption and production in buildings and houses. The ENCOURAGE platform will be capable of handling thousands of homes, each with tens of devices that can be controlled cooperatively. These devices range from appliances whose loads are controlled by simple on/off switches, to sophisticated energy producing equipment. In such a system communications can be structured upon a Message Oriented Middleware, which facilitates the integration of highly heterogeneous systems, e.g., connecting home gateways to higher level applications. The main idea behind its use is to simplify distributing applications across heterogeneous operating systems, programming language, computer architectures, networking protocols, and at the same time reducing the complexity on the interconnection functionalities and providing a high level of scalability. In our work, we want to base this layer on a Message Oriented Middleware (MOM) [19]. Examples of such technologies are RabbitMQ [4], Data Distribution Service (DDS) [6] and the Extensible Messaging and Presence Protocol (XMPP) [5].
In this paper we provide a survey of existing Middleware solutions for the support of such systems in Section 2; Section 3 surveys some relevant MOM solutions which can support the requirements of smart grid applications; the QoS support offered by the MOM solutions are surveyed in Section 3 and are evaluated in Section 4; finally, in section 5 we draw some conclusions on the topic at hand.

2 Middleware Solutions

Developing distributed applications and embedded systems is a complex task due to the heterogeneity of the applications, operating systems and programming languages that must be orchestrated together. Additionally, the response to the dynamic changes

such as hardware failure, dynamic network environment, and security attacks are hard to accomplish without the right tools and techniques [8]. Moreover, in complex distributed application such as smart grids, there are several functional modules, such as Geographical Information System (GIS), billing and metering systems, all of which need to exchange data between each other in real-time. To ease the development of complex applications, their communications are usually supported by a middleware layer. Fig. 1 illustrates how a middleware layer simplifies the transition from a communication paradigm based on direct application-to-application connections, to one based on a middleware communication bus. The approach based on direct interaction between components of the architecture leads to a large number of use cases and huge complexity of the system. Moreover, direct communication puts into the picture many specifications, some of which are work in progress by normalization bodies. Apart from fighting the resulting high complexity of such a system by using a lower number of standards in the middleware, Figure 1 makes it explicit that by the use of middleware services applications do not need to know each other's address and/or identity. With a middleware communication bus, applications send their messages to the middleware, which then distribute the messages between consuming applications. Furthermore, and also very important, the middleware concurs to the adoption of a same communication protocol for the application, further easing the communication activities.

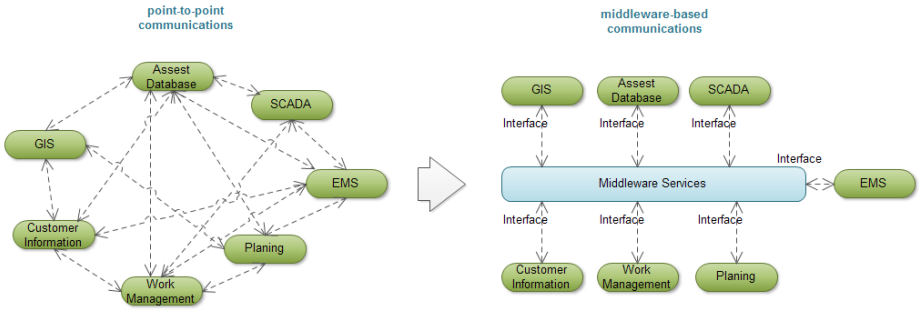

Fig. 1. Transition between point-to-point to middleware-based communications

A middleware can also provide some degrees of abstraction from the complexity and heterogeneity of the underlying communication networks, operating systems, programming languages and management of distributed applications, by providing an API that encapsulates the access to the underlying mechanisms.

In this section we briefly review the main characteristics of different categories of middleware and identify the reasons why Message Oriented Middleware (MOM) is the most adequate category to support smart grid applications.

2.1 Middleware Technologies

Middleware for distributed systems can be classified into four main categories: Remote Procedure Call (RPC), Transaction-Oriented Middleware (TOM), Object Oriented/Component middleware (OOCM) and Message Oriented Middleware (MOM). Fig. 2 displays these categories.

A Remote Procedure Call (RPC) middleware provides functionalities and infrastructure to call procedures on remote systems. It allows a computer program to run code on a remote machine without having to worry about the communication details between them. Any API call that involves RPC interface is synchronous to the user, since it waits until the server returns a response; even if it is possible to encapsulate the API call in a multi-threaded workflow to asynchronies it, the resulting system would be not scalable and with a low fault-tolerance capability [9]. The RPC model is particularly adequate for distributed systems based on client-server model.

A Transaction-Oriented Middleware (TOM) is used to ensure the correctness of transaction operations in a distributed environment. It is primarily used in architectures where the main components are database applications [15]. TOM supports synchronous and asynchronous communication among heterogeneous hosts, and it eases integration between servers and database management systems. Transactional middleware experiences a number of disadvantages such as significant overhead in managing the transactions, and, according to [10] the QoS guarantees they provide are often unnecessary or undesirable for most applications.

An Object-Oriented/Component Middleware (OOCM) is based on object-oriented programming models and supports distributed object request. OOCM is an extension of Remote Procedure Calls (RPC), and it adds several features that emerge from object-oriented programming languages, such as object references, inheritance and exceptions. These added features make OCMM flexible and very powerful [16]. Object-oriented middleware has synchronous requests as its default form of interaction, however many systems include support for asynchronous communications as well. One of the main disadvantages of object-oriented middleware is the limited scalability [11].

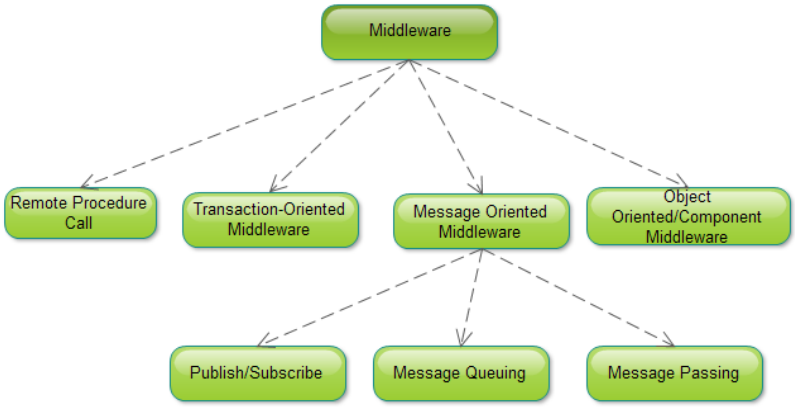

Fig. 2. Categories of Middleware

2.2 Message Oriented Middleware

A Message Oriented Middleware (MOM) is a family of software that allows message passing across applications on distributed systems. This is a large category that includes message passing, message queuing and message publish/subscribe. A MOM provides several features such as: i) asynchronous and synchronous communication mechanisms; ii) data format transformation (i.e. a MOM can change the format of the data contained in the messages to fit the receiving application [12]); iii) loose coupling among applications; iv) parallel processing of messages; v) several levels of Quality of Service support.

A MOM usually supports one or more among three different communication paradigms: i) message passing (direct communication between applications); ii) message queuing (indirect communication via a queue); iii) Publish/Subscribe mediated interaction, where messages are published to "topics" and then subscribers receive all messages published to the topics they subscribed to.

For the particular requirements of smart grid applications, as it will be seen in the rest of this section, the most interesting communication paradigm is the one supported by Publish Subscribe Message Oriented Middleware (PSMOM), which provides asynchronous and highly scalable many-to-many communication model [13]. In this scheme, senders and receivers of messages interact through an intermediary, the PSMOM. The sender of the message, called publisher, is not aware of the identity of recipients (subscribers), and it publishes its messages to the PSMOM. Subscribers are enabled to receive the messages from the PSMOM by performing subscriptions of the information they are interested in.

The publish/subscribe scheme provides systems decoupled in terms of space, time and synchronization. Space decoupling means that publisher and subscriber do not need to be aware of each other's location or identities. Time decoupling means that publisher and subscriber do not need to be online and actively collaborating in the interaction at the same time. Synchronization decoupling allows asynchronous notification of subscribers by using event services callbacks.

The filtering of messages for subscribers can be based on two approaches: topic-based and content-based [13]. In a topic-based scheme, publisher labels each message as pertaining to a certain topic, and then the messages are published as part of the topics. Subscribers will receive all the messages published to the topics they are interested in, and to which they have subscribed to. In content-based scheme, messages are sent to a subscriber based on the content of those messages; hence it represents an automatic labeling system. Subscribers will receive all the messages that match the constraints defined by them.

Fig. 3 shows four applications that connect to a MOM broker. The Publisher application sends messages to Topic 1 and Topic 2. The topics operate as relaying systems, forwarding these messages only to subscribers based on their subscriptions interests. In this case, Subscriber A subscribes to Topic 1, Subscriber B subscribes to Topic 1 and Topic 2, and Subscriber C subscribes to topic 2.

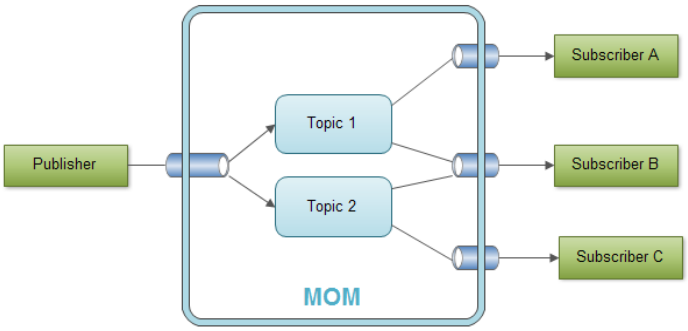

Fig. 3. Subscription to topics controls the message types that reach each subscriber

3 Message-Oriented Middleware Technologies

This section presents three MOM systems that can be considered placeholders for the categories under which current solutions on the market fall. Moreover, we believe that the solutions we present are the best representatives for the categories at hand, since they are the most mature implementation of their approaches.

3.1 Data Distribution Service (DDS).

The Data Distribution Service for Real-Time Systems (DDS) standard has been defined by the OMG organization. DDS has been designed with an emphasis on high-performance and predictability, but also to be very efficient on the use of resources. High performance is ensured by a lightweight architecture, predictability is ensured by its capabilities to reserve resources by enforcing QoS on the communications.
DDS is based on the Data Centric Publish-Subscribe (DCPS) model, which bases its operation on a global access space that is reachable by distributed nodes. Publishers are applications that write information to the data space, while consumers are applications that are able to read on the data space. Consumers are registered on the middleware, thus when data is created or modified by the publishers, DDS takes care of propagating the related changes to the data to all subscribers.
DDS uses a data model based on specific structures, which are identified by a topic and a type. The topic provides an identifier that uniquely identifies a data item within the global data space. The type provides structural information, needed to inform the middleware on how to manipulate the data and also allows the middleware to provide type safety. This feature is particularly important since it is responsible for the DDS high performance level.
DDS provides a set of complex QoS policies that provide guaranteed data delivery, real-time performance, bandwidth, redundancy and data persistence. DDS is a standard for both Application Programming Interface (API) and Wire Protocol, since it defines the binary encoding for both protocol messages and data-payload.

DDS targets high performance systems, providing high throughputs, very low latencies and real-time determinism, as proven by the results in [3]. DDS provides data structural message persistence, meaning that if an application reboots it will receive all the data changes to the data it subscribed to, which causes less traffic than receiving all the messages that changed the data over time. DDS is meant to be highly scalable, since the DDS server can extend over several computers. Moreover, DDS offers QoS policies to prioritize messages, and to guarantee QoS properties in the communication (bandwidth usage, delivery semantics, etc).

3.2 Extensible Messaging and Presence Protocol (XMPP).

The Extensible Messaging and Presence Protocol (XMPP) is an open eXtensible Markup Language (XML) protocol specified by the Internet Engineering Task Force (IETF) for near-real-time messaging, presence, and request-response services [5]. This technology was originally developed for instant messaging applications, and it has been extended to support other application domains, like voice and video communications. But in fact, it can serve the purpose of delivering data items from a large number of connected devices to higher level applications, being, at the same time, scalable and easy to implement. XMPP includes functionalities for XML streaming, encryption using Transport Layer Security (TLS), authentication based on the Simple Authentication and Security Layer (SASL), unicode support, and information about publishers' or subscribers' presence in the network.

XMPP technology is based on the client/server paradigm where clients are interconnected through servers. Therefore, when two applications want to communicate, they connect to a XMPP server and start exchanging XMPP messages, which are always routed through the servers. Such kind of communications can be encrypted and the clients are also able to run an authentication protocol to connect with the server(s). The XMMP clients are identified by a unique identifier called Jabber Identification (JID). XMPP servers are also capable of connecting between each other for scalability proposes.

XMPP specifies three different communication snippets called XML stanzas: message, presence, and iq (Info/Query). The message stanza is a method sent from one entity to another for negotiating an XML stream.

In this protocol, all messages are encoded using the XML format. In addition to the basic functionalities, XMPP also defines protocols for multi-user chat, publish/subscribe, HTTP bindings and other. In contrast to DDS, XMPP is considered to be a standard only for Wire protocol, i.e. XMPP is agnostic in relation to the data being transferred.

3.3 AMQP (RabbitMQ)

RabbitMQ is an open source messaging broker based on the Advance Messaging Queue Protocol (AMQP) standard [7], which has been written in Erlang. AMQP defines both a wire protocol, and a protocol model that specifies the semantics for

AMQP implementation: by abiding to these guidelines, AMQP implementations will be interoperable with other AMQP implementations.
AMQP divides the brokering task between exchanges and message queues, where the first is basically similar to a router that accepts incoming messages, and, based on a set of rules or criteria, decides which queues to route the messages to. Note that this module does not store messages. A message queue stores messages and sends them to message consumers.
This protocol clearly differentiates form the other ones due to the provision of the chain of responsibility pattern, where each processor acts on the message along the way, perhaps adding to it, modifying its form, rejecting it, or simply passing it through to the next processor. It basically enhances system flexibility by letting developers to separate and to combine orthogonal functionality as needed. Another second important aspect is that AMQP enables the broker to make routing decisions that are usually left to the application.
The AMQP protocol defines the wire protocol, usually based on TCP/IP, where each frame contains a channel number. A last characteristic of AMQP is that it is a binary protocol, thus more efficient than text-based ones.

4 QoS Support on Message Oriented Middlewares

The capabilities of a MOM in relation to QoS play a critical role in the overall system performance, especially with the increasing demand from applications that require soft real-time publish/subscribe services. Applications within distributed systems are characterized by real-time information that flows from sensors to applications and from applications to actuators. Ensuring real-time data distribution is fundamental for maintaining the correctness and safety of such systems. In the remainder of this section we analyze the support by MOM protocols described in Section 3, in relation to 4 QoS metrics: latency/jitter, bandwidth, delivery semantics and message priority and ordering [14, 17].
Latency is defined as the travel time of a message from its source to its destination. A concept related to latency, and which constitutes an important metric for a distributed paradigm, is the **jitter**, which is a measure of the variability over time of the message latency; a system with high jitter can be considered as unreliable. The end-to-end latency between two distributed applications depends on the number of broker hops in the path, the destination and source loads, the network protocol, and the network condition. Furthermore, controlling jitter is important for real time systems.
The latency and jitter can be explained from several aspects, network point of view, publisher point of view and subscriber point of view. As far as network is concerned, the latency regards the time that the message spends while being transported over the network. From the publisher side, the delivery time of these messages is critical since some messages are useful only for a certain amount of time, and after they become outdated. From the subscriber point of view, the metric is important when the subscriber needs to receive the messages from publishers within a certain time.

There are several aspects that should be taken into account in publish/subscribe systems' latency. First, the topology gives an assessment of how many brokers exist in the path between publisher and subscriber and what is the impact of the MOM layers in this delay. Second, the travel time among different hops can vary, depending on how the overlay network routes messages among nodes [18]. Third, the time message spends in each broker to be processed must be accounted, and that time depends on the broker's load.

The DDS protocol ensures very low message latency and more limited jitter by using fewer layers in comparison with other middleware technologies. DDS supports a number of QoS policies, like guarantees on the maximum latency for data delivery, latency budget, reliability of data delivery, priority of data delivery, and deadline policy; this set of QoS policies can reduce latency and jitter significantly. Latency-budget policy defines the maximum acceptable delay from the time the data is written until the data is received by a subscriber application. Deadline policy specifies the maximum inter-arrival time between messages, and it defines the maximum duration that a "Data Reader" expects to elapse between the change of a value, and the update of the values contained in each subscriber's instance.

XMPP protocol basically is a best effort protocol with some Extension Protocols (XEP) that support QoS functionalities. XEP – 0203 provides timestamp information regarding stored messages, which can be useful in case of later delivery, so that if a message is delayed, the original send time can be determined.

RabbitMQ uses a method called "pre-fetch" to determine how many messages will be sent before the customer acknowledges a message. The objective is to send message data in advance, to reduce latency [7].

All of the three systems perform fine with respect of this metric. Anyway, DDS makes efforts to cope with jitter by enforcing reliability in the communication.

Bandwidth: The overall bandwidth depends on the throughput of the broker and the size of each message. In this case publishers can identify the upper and lower bounds for the output stream and a subscriber can limit the maximum bandwidth used for receiving the messages.

DDS controls network bandwidth by using the time-based-filter policy, which defines the minimum inter-arrival time between messages. Also, DDS uses the resource-limit policy to control the amount of message buffering in the queues. Those policies lead to minimal waste of network bandwidth and potentially can provide high throughputs [3]. The time-based filter policy allows handling different production and consumption rates without overflowing consumers.

The XMPP XEP – 0138 defines an extension standard for negotiating compression of XML streams. The protocol supports a wide range of compression algorithms. It can reduce bandwidth usage up to 90% [5]. Jingle RTP Sessions (XEP – 0167) protocol enables applications to communicate through negotiated sessions that use the Real-time Transport Protocol (RTP) to exchange voice or video data. This kind of QoS protocol has implications on QoS guarantees for both latency and bandwidth.

In the RabbitMQ implementation of AMQP, channels provides a way to multiplex one robust TCP/IP connection into several lightweight connections, making efficient

use of the network port, and allowing the available bandwidth to be shared among concurrent activities.
All of the systems are good candidates for smart grids with respect of this metric.
Delivery semantics: Delivery semantics depend on two factors, network reliability and protection from duplicate messages. There are several levels of delivery semantics: i) best effort: this is the lowest level of delivery guarantee, which mean that no reliability guarantee and messages can be duplicated; ii) at most once: delivery guarantee ensures that the subscriber receives at most one message of an event type instance; iii) at least once: the subscriber receives at least one message from a specific event type; iv) exactly once: the subscriber receives a message from exactly one event type instance, and it receives each message only once.
DDS provides a reliability QoS policy that specifies two different data delivery guarantee modes. Those modes are "Reliable" and "Best Effort". Reliable guarantees mean that all messages in a "Data Writer" history will be delivered to the correspondent "Data Reader". Best effort indicates that a message is only sent once, and should the transmission fail, the message will be lost.
In XMPP, the Advanced Message Processing (AMP) (XEP – 0079) protocol allows publisher and subscriber to define additional delivery semantics for advanced processing of XMPP message stanzas, including reliable data transport.
RabbitMQ uses queues with guaranteed delivery to a single recipient. It uses different delivery modes, which specify if the message will need persistence. The messages that are indicated as persistent will be protected in case of server reboot by saving them in a persistent log file, and sent to each application that associates to the middleware, even if some time has elapsed from whence the message was published.
DDS is a clear winner as a basis for smart grids application with respect to this metric.
Message priority and ordering: message priority is not a subject related to topology design; instead, it impacts on the broker decisions on ordering and dropping messages. Publishers assign relative priorities to their messages, to control the broker's queues; subscribers allocate the relative priorities among their subscriptions.
A set of events can be delivered in different orders [1]: i) First In-First Out (FIFO), which is the default temporal message ordering, or Last In-First Out (LIFO); ii) random or unordered: if the messages aren't required to be subject to a given ordering policy, the ordering is considered to be either unordered or random; iii) causal ordering: it guarantees that the same causal relationship between sent and received messages are maintained, so that all the messages are delivered in the same order that they were produced; iv) total ordering: all messages are delivered in the same order to all subscribers; v) priority ordering: it is a way of ordering in which the publishers assigns priority among their messages while publishing them.
DDS provides QoS policies for both the definition of message transport priority, and to control the order of received messages. The transport priority QoS policies allow a DDS application to deliver messages with different priorities. The destination order QoS policies allow the subscriber to maintain a logical order for the same data instance among changes made by multiple publishers; this is achieved by using timestamps when a message is produced.

In XMPP, Resource Application Priority extension protocol (XEP – 0168) specifies priority for each connected resource. The protocol defines how the XMPP server assigns resources to be prioritized for any given application type.
The RabbitMQ ensures content ordering by exploiting the TCP/IP transport layer, which the AMQP is built on. Messages are delivered in the order in which they are sent. For prioritizing messages, a publisher can assign a value from 0 to 9 to each message, to specify a message priority that will cause the message to be transferred between queues before messages having lower priorities.
DDS has got the advantage of proposing different ways to cope with message priorities and ordering, hence it is the best candidate for smart grids as far as this metric is considered.

5 Conclusions

Ensuring real-time for bidirectional data flows in the middleware layer is a key requirement for system requiring high scalability, such as smart grids. In this paper, we described the main features of different categories of middleware, and identified reasons to prefer Message Oriented Middleware (MOM) over the other categories to support smart grid applications. To effectively take advantage of distributed communication through a MOM, a large-scale data infrastructure must employ a scalable real-time data middleware. The selection of the most adequate MOM technologies for Smart Grid middleware is not a trivial task. The choice depends on the grid requirements and the features they need, such as scalability, reliability, flexibility, security and real-time data.
This paper surveys several technologies that support the MOM communication paradigm. Our analysis led us to the conclusion that XMPP has not been developed in order to support real time constraints, since it mostly targets interactive web applications. The QoS capabilities of XMPP are also limited and are mostly supported by extensions to the protocol. DDS targets distributed real-time systems [20] and therefore it is capable of addressing very complex distributed applications, where QoS requirements have to be guaranteed. AMQP is used for high performance distributed system applications, and it is an open cloud messaging platform for real-time on a global scale. On the other hand, AMQP's focus is mostly on high performance and not on predictability. It is therefore possible to conclude that DDS is the most suitable technology for smart grid application with QoS requirements.

Acknowledgment

This work was supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by the EU ARTEMIS JU funding, within ENCOURAGE project, ref. ARTEMIS/0002/2010, JU grant nr. 269354.

References

1. S.P. Mahambre et al., “A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware”, IEEE Internet Computing Magazine, vol. 11, no. 4, Jul. 2007.
2. Jian Yin, Ian Gorton, “Scalable Data Middleware for Smart Grids”, Pacific Northwest National Laboratory, Proceedings of the Middleware 2011 Industry Track Workshop.
3. A. Cosaro, L. Querzoni, S. Scipioni, S. Piergiovanni and A. Virgillito, “Quality of Service in Publish/Subscribe Middleware”, Global Data Management , IOS Press,2006, pp.20- 20.
4. A. Videla, J. J.W. Williams, “RabbitMQ in Action: Distributed Messaging for Everyone” MEAP Edition, Manning Early Access Program, 2011.
5. P. Saint-Andre, K. Smith, and R. Tronc¸on, “XMPP: The Definitive Guide”, O’Reilly, 2009.
6. Object Management Group, Inc. (OMG), “Data Distribution Service for Real-Time Systems Specification”, Version 1.1, formal/05-12-04, December 2005.
7. AMQP Advanced Message Queuing Protocol, Protocol Specification, Version 0-9-1, 13 November 2008, http://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf.
8. S. M. Sadjadi, P. K. McKinley. “A survey of adaptive middleware”, Technical Report MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, December 2003.
9. K. Geihs, “Middleware challenges ahead”, IEEE Computer, June 2001, 34(6): 24--31.
10. V. Issarny, M. Caporuscio, N. Georgantas, “A Perspective on the Future of Middleware-based Software Engineering”, Future of Software Engineering 2007, L. Briand and A. Wolf edition, IEEE-CS Press. 2007.
11. E., Tokunaga, A. Zee ,“Object-Oriented Middleware Infrastructure for Dis-tributed Augmented Reality”, ISORC 2003.
12. E. Curry, D. Chambers, and G. Lyons, “Extending Message-Oriented Middle-ware using Interception”, presented at Third International Workshop on Distributed Event-Based Systems (DEBS '04), ICSE '04, Edinburgh, Scotland, UK, 2004.
13. P. Th. Eugster, P. Felber, R. Guerraoui, A-M. Kermarrec, “The Many Faces of Publish/Subscribe”, ACM Computing Surveys 35(2), 2003, pp. 114-131.
14. S. Behnel, L. Fiege, G. Muhl, "On Quality-of-Service and Publish-Subscribe," ICDCSW, 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06), 2006, pp.20.
15. L. Capra, W. Emmerich, C. Mascolo, “Middleware for Mobile Computing”, UCL Research Note RN/30/01, Submitted for publication, July 2001.
16. R. Nunn, “Distributed Software Architectures Using Middleware”, 3C05 Coursework 2.
17. S. Arianfar, “Optimizing Publish/Subscribe Systems with Congestion Handling”, Technical Report Helsinki University of Technology, 2008.
18. S. Rhea, D. Geels, T. Roscoe and J. Kubiatowicz, “Handling churn in a dht”, ATEC’04: Proceedings of the annual conference on USENIX Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 2004, pp. 10–20.
19. E. Curry, "Message-Oriented Middleware", in Middleware for Communications, Q. H. Mahmoud, Ed. Chichester, England: John Wiley and Sons, 2004, pp. 1-28.
20. T. Guesmi, R. Rekik, S. Hasnaoui and H. Rezig, “Design and Performance of DDS-based Middleware for Real-Time Control Systems”, 2007.
21. Embedded iNtelligent COntrols for bUildings with Renewable generAtion and storage (ENCOURAGE), http://www.encourage-project.eu