QoS-enabled Middleware for Smart Grids

Abdel Rahman Alkhawaja, Luis Lino Ferreira, Michele Albano, Ricardo Garibay
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto
R. Dr. António Bernardino de Almeida, 431
4200-072 Porto / Portugal
{abdel,llf,mialb,rgmz}@isep.ipp.pt

I. Introduction

Emerging smart grid systems must be able to react quickly and predictably, adapting their operation to changing energy supply and demand, by controlling energy consuming and energy storage devices. An intrinsic problem with smart grids is that energy produced from in-house renewable sources is affected by fluctuating weather factors. The applications driving smart grids operation must rely on a solid communication 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).

The European project ENCOURAGE [9], 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. Such a system can be structured upon a Message Oriented Middleware. 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) [8]. Examples of such technologies are RabbitMQ [2], Data Distribution Service (DDS) [4] and the Extensible Messaging and Presence Protocol (XMPP) [3].

II. Message Oriented Middleware Solutions

Message Oriented Middleware (MOM) allows a simplified connection between distributed applications, since it allows applications not to know each other's address and/or identity. Furthermore, and also very important, the middleware concurs to the adoption of a same communication protocol for the application, further easing the communication activities.

A middleware can also provide some degree 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.

A Publish Subscribe Message Oriented Middleware (PSMOM) provides an asynchronous and highly scalable many-to-many communication model [6]. In this scheme, 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 into.

The capabilities of a MOM in relation to QoS play a critical role in the overall system performance. In the remainder of this section we analyze the support by selected MOM protocols, in relation to 4 QoS metrics: latency, bandwidth, delivery guarantees and, message priority and ordering [7].

A. *Data Distribution Service (DDS).*

The Data Distribution Service for Real-Time Systems (DDS) standard has been designed with an emphasis on high-performance and predictability, but also to be very efficient on the use of resources, which are ensured by a lightweight architecture and by its capabilities to reserve resources by enforcing QoS on communications and local execution. DDS is based on the Data Centric Publish-Subscribe (DCPS) model, which uses specific structures, 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.

DDS ensures low latency by providing a set 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.

DDS controls network bandwidth by using the time-based-filter policy, which defines the minimum inter-arrival time between messages. Also, it 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 [1].

DDS provides a reliability QoS policy that specifies two different data delivery guarantee modes: 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 sent once, and should the transmission fail, the message will be lost.

DDS provides QoS policies for both the definition of message transport priority, and to control the order of received messages. The destination order QoS policies allow the subscriber to maintain a logical order for the same data in-stance among changes made by multiple publishers; this is achieved by using timestamps when a message is produced.

B. *Extensible Messaging and Presence Protocol (XMPP).*

The Extensible Messaging and Presence Protocol (XMPP) is an open eXtensible Markup Language (XML) protocol for near-real-time messaging, presence, and request-response services [5]. XMPP is based on the client/server paradigm where clients are interconnected through servers, although it also supports publish/subscribe model.

XMPP protocol basically is a best effort protocol, enriched with some Extensions Protocols (XEP) that supports some QoS functionalities. XEP-0203 protocol provides timestamp information regarding stored messages, which can be useful in case of late delivery, so that if a message is delayed, the original send time can be determined. XEP-0079 defines the Advanced Message Processing extensions that enable an application to define rules to handle time sensitive messages.

XEP-0138 allows negotiating the compression of XML streams. 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 policy has implications on QoS guarantees on both latency and bandwidth.

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. Finally, XEP-0168 allows specifying priorities for connected resources associates with applications.

C. *AMQP (RabbitMQ)*

RabbitMQ is an open source message broker based on the Advance Messaging Queue Protocol (AMQP) standard [5]. It defines both a wire protocol, and a protocol model that specifies the semantics for AMQP implementations making AMQP implementations interoperable with other 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.

In RabbitMQ, which is an implementation of AMQP, it uses a method called "pre-fetch" to determine how many messages will be sent before the consumer acknowledges a message. The objective is to send messages data in advance, to reduce latency [7]. It also supports channels, which provide a way to multiplex one robust TCP/IP connection into several lightweight connections, making efficient use of the network, and allowing the available bandwidth to be shared among concurrent activities.

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.

RabbitMQ ensures content ordering by exploiting the TCP/IP transport layer, which the AMQP is built upon. 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 ensure transfer between queues according to message's priority.

III. CONCLUSIONS

Based on the analysis presented in this work it is possible to conclude that DDS is the most suitable technology for smart grid application with QoS requirements, since it provides more adequate solutions with respect to its competitors.

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] A. Corsaro, et al., "Quality of Service in Publish/Subscribe Middleware", Global Data Management, IOS Press,2006, pp.20- 20.

[2] A. Videla, J.Williams, "RabbitMQ in Action Distributed Messaging for Everyone" MEAP Edition, Manning Early Access Program, RabbitMQ in Action version 8, 2011.

[3] P. Saint-Andre, K. Smith, and R. Tronçon, "XMPP: The Definitive Guide", O'Reilly, 2009..

[4] Object Management Group, Inc. (OMG), "Data Distribution Service for Real-Time Systems Specification", Version 1.1, formal/05-12-04, December 2005.

[5] AMQP Advanced Message Queuing Protocol, Protocol Specification, Version 0-9-1, 13 November 2008, http://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf.

[6] P. Eugster, P. Felber, R. Guerraoui, A-M. Kermarrec, "The Many Faces of Publish/Subscribe", ACM Computing Surveys 35(2), 2003, pp. 114-131.

[7] S. Behnel, L. Fiege, G. Muhl, "On Quality-of-Service and Publish-Subscribe,", 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06), 2006, pp.20.

[8] Somaya Arianfar, "Optimizing Publish/Subscribe Systems with Congestion Handling", Helsinki University of Technology, 2008.

[9] E. Curry, "Message-Oriented Middleware", in Middleware for Communica-tions, Q. H. Mahmoud, Ed. Chichester, England: John Wiley and Sons, 2004, pp. 1-28.

[10] Embedded iNtelligent COntrols for bUildings with Renewable generAtion and storage (ENCOURAGE), http://www.encourage-project.eu