Category: original scientific paper

Celar Stipe [1]

Mudnic Eugen [2]

Seremet Zeljko [3]

State-of-the-Art of Messaging for Distributed Computing Systems

Abstract:

Modern software applications rarely live in isolation and nowadays it is common practice to rely on services or consume information provided by remote entities. In such a distributed architecture, integration is key. Messaging, for more than a decade, is the reference solution to tackle challenges of a distributed nature, such as network unreliability, strong-coupling of producers and consumers and the heterogeneity of applications. Thanks to a strong community and a common effort towards standards and consolidation, message brokers are today the transport layer building blocks in many projects and services, both within the physics community and outside.

Moreover, in recent years, a new generation of messaging services has appeared, with a focus on low-latency and high-performance use cases, pushing the boundaries of messaging applications. This paper will present messaging solutions for distributed applications going through an overview of the main concepts, technologies and services.

Keywords:

messaging; message-oriented middleware; MQ; message queuing; distributed systems

Author´s data:

[1] Celar, Stipe, PhD, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture in Split, Croatia, Stipo.Celar@fesb.hr

[2] Mudnic, Eugen, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture in Split, Croatia, Eugen.Mudnic@fesb.hr

[3] Seremet, Zeljko, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture in Split, Croatia

004.5; DOI 10.2507/IJVA.3.2.1.34

Introduction

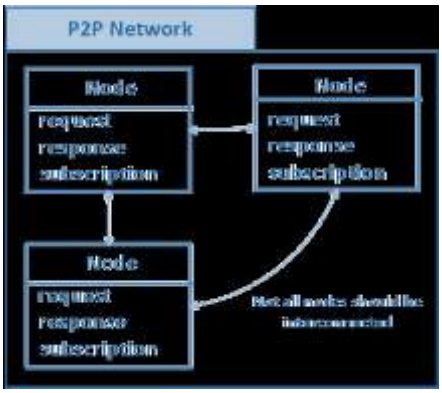
This paper presents an overview of messaging concepts, functionalities and modern technologies. It starts with an introduction of messaging for distributed communication and system integration. A review of the main messaging features is then provided, followed by an overview of the major technologies for messaging, from broker to broker-less systems. In conclusion, a list of successful CERN's stories concerning the use of messaging for solving the communication problem of distributed applications is presented.

Message-oriented middleware

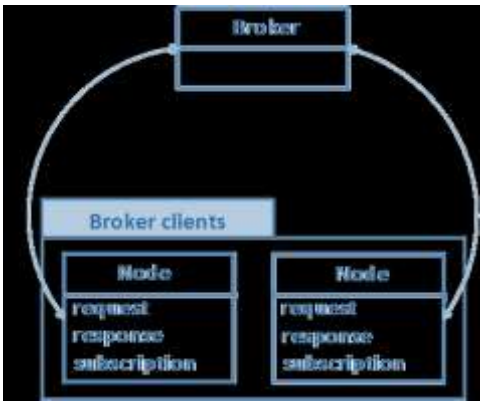
To cope with increasing demands on scalability, flexibility, and reliability, a message-oriented middleware (MOM) is an infrastructure for loosely coupled interprocess communication in an enterprise service bus or clouds [1] [2]. Particularly in clouds, loose coupling allows to rapidly scale message producers and consumers. A message with respect to MOM is an autonomous, self-contained entity that models an event and separates into a header and a body or payload. The middleware provides technical means of exchange, so a peer can exchange messages with other connected peers. A central concept in MOM is the notion of a message queue (or channel) for storing, transforming, and forwarding messages. Message queues enable asynchronous interaction, and a simple form is a First-In-First-Out (FIFO) queue. There are two different approaches to MOM using message queues as shown in Fig. 1.:

- Peer-to-peer messaging. A unified middleware component in every peer coordinates discovery and interaction between peers.
- Broker-based messaging. The middleware acts as a broke to provide a messaging infrastructure between the heterogeneous peers.

Peers can participate as client, service, or both [1]. A broker reduces the communication complexity between a numbers of peers but can incur delays in real-time applications because an additional store-and-forward procedure is necessary. [3]



a)



b)

Figure. 1. A message-oriented middleware abstracts communication between heterogeneous peers: (a) Peer-to-peer messaging; (b) Broker-based messaging

In terms of interaction patterns, a trivial message queue allows bilateral Send and Receive, for asynchronous messaging, and multilateral One-to-Many Send, e.g., publish-subscribe. Using message queues in a broker architecture allows to implement sophisticated routing patterns. In general, a MOM is characterized by Curry [1]:

- Messaging specification. A MOM needs to specify th format of messages and transport

004.5; DOI 10.2507/IJVA.3.2.1.34

mechanisms. Interconnectin proprietary MOM systems is achieved through adapters or bridges.

- Message filtering. A core functionality of a MOM is filtering for message delivery. Curry [1] distinguishes:
  - A channel-based system offers predefined groups of events as channels, where clients can subscribe to.
  - Messages in a subject-based system carry metadata in the message header, e.g., a subject. A client subscribes messages, where the metadata matches some given pattern.
  - In a content-based system, a client subscribes messages, where the message body satisfies a set of properties expressed in a query language.
  - Composite events functionality extends a content based filtering with property matching across sets or sequences of messages.
- Message transformation. Messages can originate from various heterogeneous sources and consequently carry all kinds of content types as payload. A MOM can offer APIs to modify messages, e.g., XML transformations.
- Integrity, reliability, and availability. A MOM can have properties to increase the overall Quality-of-Service:
  - Transactions and Atomic Multicast Notification;
  - Reliable message delivery: at-least-once, exactly-once, or at-most-once;
  - Guaranteed message delivery by acknowledgments;
  - Prioritization of messages;
  - Load balancing over several brokers or queues; and
  - Message broker clustering for fault tolerance.

A MOM is typically accessed through an API to abstract the technical details of message exchange. Due to the transport-agnostic design of SOAP/WS-* services, a MOM can also serve as a transport mechanism for SOAP messages. [3]

Java Message Service

The general purpose API named Java Message Service (JMS) [4] is maintained in a Java community process for MOM support. JMS defines a number of operations for creating, sending, receiving, and reading messages. It is transport-agnostic to abstract messaging from MOM implementations and therefore relaxes vendor lock-in. JMS is a universal interface for interacting with heterogeneous messaging systems [1]. A message body is dynamically typed according to the content type information stored in the header.

Some examples for JMS-enabled software implementations are the JMS reference implementation OpenMQ [5], IBM Websphere MQ [6], or TIBCO Enterprise Message Service [7]. [3]

RESTful Messaging Service

The motivation for RESTful Messaging Service (RestMS) [8] is Web-compatible messaging by using HTTP as transport mechanism and REST principles to describe locations, i.e., URLs, where messages can be posted to and received from. RestMS is an API specification, where XML-based messages are sent and received using HTTP methods. With respect to the REST service, resource locations are distinguished into feeds for incoming and pipes for outgoing messages. Feeds are joined with pipes on the service-side for message distribution. Message types in RestMS refer to XML, JSON, and a set of MIME content types for dynamically typing data. The specification also includes profiles to connect to other messaging infrastructures, e.g., AMQP. [3]

International Journal - VALLIS AUREA • Volume 3 • Number 2 • Croatia, December 2017
004.5; DOI 10.2507/IJVA.3.2.1.34

Open Middleware Agnostic Messaging API

Due to the diversity in middleware standards and wire formats, the Open Middleware Agnostic Messaging API (OpenMAMA) [9] initiative is an attempt to provide a single API for developing applications spanning across multiple MOMs. For correct translation messages and operations, a MOM has to provide a so-called OpenMAMA bridge implementation.

OpenMAMA is available as open-source library. It offers a built-in bridge for AMQP-enabled Apache Qpid and supports several bridges for proprietary messaging infrastructures in the finance sector. [3]

Proprietary messaging solutions

MSMQ [10] is a MOM for standalone integration or as a transport mechanism in Microsoft's WCF, next to Web services and COM+. It offers guaranteed message delivery, message routing, transactions, prioritization, and a simple type system for message body types. When used as a transport in WCF, a message body is either XML, binary, or ActiveX format. Beside its proprietary protocols, messages can also be transmitted over COM+. In terms of security, MSMQ allows authentication and encryption of messages. There is no broker in MSMQ; similar to Fig. 18a, a queue is hosted locally on a peer, and processes can store and retrieve messages. In terms of service interaction patterns, MSMQ is bilateral Send and Receive. MSMQ can exploit IP multicast to replicate a message for addressing multiple queues. A Microsoft alternative with brokerage support is SQL Server Service Broker [11].

Other proprietary MOM software products are the brokerless TIBCO Rendezvous [12], which uses direct connections between peers similar to MSMQ, Oracle Tuxedo Message Queue [13] as part of the Oracle Tuxedo application server for cloud middleware, and Terracotta Universal Messaging [14]. [3]

Advanced Message Queuing Protocol

Historically, MOM solutions have relied on proprietary protocols, and JMS is an attempt to agree on a compatible interface. Interoperability between varying MOM solutions is still difficult; costly JMS adapters or bridges are necessary to connect different transport mechanisms. AMQP [15] unifies messaging through an agreed-on wire format and has a similar role like HTTP in Web applications. While the OASIS AMQP 1.0 standard is restricted to the transport model for interoperability over the Internet, messaging architectures are specified by the AMQP working group [16].

The AMQP specification distinguishes a transport model and a queuing model [17]. The semantic queuing model defines terms like message, queue, exchange, and binding with respect to AMQP. Messages always end up in queues which are analogous to postal mailboxes. A queue stores messages and offers functionality for searching, reordering, or transaction participation. If a client wants to send a message, it chooses a broker-like exchange which is responsible for delivering messages to queues. An exchange can be offered as a service, and there exists an individual URI scheme (amqp: or amqps:) [18] to locate an exchange. A binding is a set of queue-specific arguments for an exchange. As shown in Fig. 2., there are different exchange types with respect to message filtering capabilities [19]:

- In a direct exchange, a message has a routing key and is sent to the queue, whose binding is equivalent to the routing key. In case of multiple queues with identical bindings, multiple message copies are delivered, i.e., a channel-based system.
- A topic exchange forwards copies of a message to all client queues, where the message routing key matches a queue's binding pattern, i.e., a subject-based system for publish-subscribe delivery.

004.5; DOI 10.2507/IJVA.3.2.1.34

- In a fan-out exchange, messages are forwarded to a set of queues without a specified binding, i.e., channel-based system.
- A headers exchange matches the headers of a message against predicate arguments of client queues beyond the routing key, i.e., a content-based system.

Messages are finally fetched from queues by consumer processes. AMQP provides guaranteed delivery, authentication, wire-level encryption, and transaction-based messaging for reliability. In terms of patterns, an exchange applies pattern Send in case of direct delivery or One-to-Many Send in other cases. Due to the self-contained type system and self describing message content, messages are dynamically typed in AMQP.

Examples for JMS-compatible broker implementations are OpenAMQ [20], JORAM [21], WSO2 Message Broker [22], SwiftMQ [23], Apache Qpid [24], and Red Hat Enterprise MRG [25].

AMQP defines four types of exchanges. A producer creates a message and sends it to an exchange. Depending on the exchange type and bindings, the message is delivered to queues, where consumers can fetch it from (a) direct exchange; (b) topic exchange; (c) fan-out exchange; (d) headers exchange. [3]

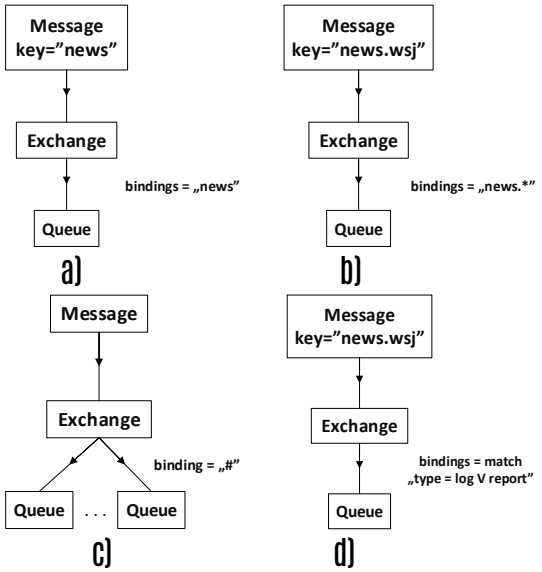


Figure. 2. AMQP types of message exchange

Extensible Messaging and Presence Protocol

While the XMPP [26][27][28] has been intended as an open standard for instant messaging, presence information, and contact list maintenance in chat applications, it also has middleware properties. In its base specification, XMPP exchanges messages as XML stanzas in client-to-service and service-to-service communication for federated services. An XMPP service therefore takes the role of a broker.

XMPP is particular attractive for MOM scenarios, where Web agents are involved because it supports HTTP as transport mechanism and most Web browsers and JavaScript runtime environments are capable of processing XML stanzas. Furthermore, XMPP is also considered as a suitable messaging protocol for Internet of Things applications [29]. The protocol is extensible, and extensions are specified in a community process. MOM-specific extensions are:

- Transfer of Base64-encoded binary content with an assigned MIME media type [30];
- RPC over XMPP [31];
- Service discovery [32];
- Publish-subscribe [33] for broker scenarios, extended addressing [34] for message routing, and event notification extensions [35][36];
- Reliable message transport [37]; and
- SSL/TLS protected transport mechanism and S/MIME [38] for end-to-end message encryption.

By default, messages in XMPP are XML stanzas and bodies are restricted to text only; there exists a notion of message type, but it is limited to instant messaging applications. Therefore, out-of-band signaling or a custom protocol, e.g., XMPP bits of binary [39], is required to discover message content types in a middleware scenario.

004.5; DOI 10.2507/IJVA.3.2.1.34

XMPP can also serve as a messaging infrastructure for SOAP/WS-* Web services [40] [41]. Beside instant messaging, XMPP has been successfully deployed in the VIRTUS middleware for Internet of Things applications [42] using the real-time collaboration server software OpenFire [43]. Another software that offers XMPP messaging over Web-Socket is the Kaazing WebSocket Gateway [44]. [3]

Streaming Text Oriented Messaging Protocol

The simple text-based wire protocol STOMP [45] is for asynchronous message exchange between a client and a service or broker with simplicity and interoperability in mind. In the open standard of STOMP, a client and a service establish a session and asynchronously exchange frames of type Message, Receipt, or Error; a frame is partitioned into a command, header fields for metadata, and content of a certain MIME type. Messages are therefore dynamically typed. The protocol supports transactions and acknowledgments for reliable message delivery.

STOMP supports either bilateral messaging, i.e., Send and Receive, or broker-based publish-subscribe for One-to-Many Send interaction. Two notable service implementations are CoilMQ [46] and, for the latest protocol version 1.2, Stampy [47]. [3]

Message Queue Telemetry Transport

MQTT [48] originates from IBM and is now an open OASIS standard [49] for lightweight machine-to-machine messaging and Internet of Things applications, where bandwidth is limited. MQTT is intended for broker-based publish-subscribe architectures, One-to-Many Send interaction. An MQTT message can encapsulate binary payload up to 256 megabytes, but there is no notion of content type. The participating parties therefore have to agree on allowed formats out-of-band. For reliability, the protocol offers acknowledgments and retransmissions, but there is no transaction functionality.

Two notable MQTT broker software implementations are HiveMQ [50] and Mosquitto [51]. Both support Web clients using WebSocket. Another application that relies on MQTT messaging is Facebook Messenger [52].

Data Distribution Service for real-time systems

The open standard DDS [53] specifies a machine-to-machine MOM for publish-subscribe message distribution, real-time message delivery, scalability, and high throughput. Fields of application include the finance and defense sector, industry, aerospace, Internet of Things, and mobile devices [54].

Contrary to MQTT, DDS facilitates a data-centric, peer-to-peer interaction in the spirit of Fig. 1. (a). A domain partitions entities such as publisher, subscriber, and topic. A topic in a domain has a unique name and a strong datatype for publishing; these types are specified in an IDL, and messages are therefore statically typed. Subscribers in the domain request data via the topic, and publishers in the domain are responsible for message distribution [55].

DDS supports rich Quality-of-Service policies for data transmission. Interoperability between software implementations is achieved by the RTPS [56] wire protocol. To locate endpoints of peers, DDS provides dynamic discovery of publishers, subscribers, topics, and datatypes with respect to topics [54]. Reliable message delivery is achieved by negative acknowledgment when data is missing [57]. Security extensions for DDS, e.g., encrypted transport, are still in a beta state at time of writing [58].

004.5; DOI 10.2507/IJVA.3.2.1.34

Notable software implementations are OpenDDS [59], RTI Connext DDS [60], PrismTech OpenSlice DDS [61], and Twin Oaks CoreDX DDS [62]. [3]

Apache Kafka

Developed by LinkedIn, Apache Kafka [63] is a message broker specification and implementation for high-throughput publish-subscribe messaging, i.e., One-to-Many Send interaction. Kafka has an individual binary wire format protocol on top of TCP, and for fault tolerance, it supports clustering of brokers, persistent storage, and replication of messages.

On a conceptual level, Kafka distinguishes between topics for messages, producers that publish messages, and consumers that subscribe to topics. For every topic, a Kafka cluster maintains a partitioned log, where every partition stores an ordered sequence of published messages. The messages are kept for a configurable timespan, and partitions are replicated and distributed over servers in the Kafka cluster for fault tolerance and performance. The distributed log in Kafka guarantees the ordering of published and consumed messages in a certain topic. For a subscribed topic, a consumer maintains an offset in the message sequence to keep track of already processed ones. Through this offset, a consumer can also access older messages if they are still available on the cluster.

A message body is a byte sequence of a certain length and has no notion of type. Content type information therefore needs to be agreed out-of-band or by using a custom protocol. An interface for Web clients to subscribe to Kafka over WebSockets is already in an experimental state [64].

Polyglot message brokers

A natural approach for interconnecting several MOM standards is polyglot message brokerage. Three notable JMS-compliant software implementations in this area are Apache ActiveMQ [65], RabbitMQ [66], and JBoss HornetQ [67].

Beside features for scaling and clustering, the messaging core of Apache ActiveMQ, referred to as Apollo [68], uses the OpenWire [69] wire format, but also supports standards like AMQP, MQTT, and STOMP over WebSockets. ActiveMQ furthermore provides a proprietary HTTP-based RESTful API for Web clients.

RabbitMQ supports AMQP, STOMP, MQTT, and also HTTP as transport. Messages over HTTP can be sent in three ways: a native Web management API, STOMP over WebSockets, and JSON-RPC for Web browser integration.

HornetQ [67] is a MOM that originates from the JBoss application server. It supports AMQP, has an HTTP-based RESTful Web interface, and provides STOMP over Web-Sockets for Web clients. [3]

Message queuing as a service

Message brokerage has become an attractive cloud service. A broker is a critical component in a MOM architecture and needs fault tolerance, regular maintenance, and scalability; a message queue cloud service can eventually reduce cost. Amazon Web Services offers Simple Queue Services (SQS) [70] for transporting untyped text-based messages up to 256 kilobytes. SQS operates on a SOAP/WS-* Web service stack accessible through HTTP and HTTPS bindings.

Google's App Engine offers Pull Queues [71] and Push Queues [72] for messaging and App Engine task distribution. Both queue types are accessible through a RESTful API and use JSON format for messages. While Pull Queues need to be polled, Push Queues rely on webhooks for HTTP-based message delivery. Google has also announced Cloud Pub/Sub [73], a broker-based publish-subscribe messaging service for the App Engine, cloud apps, and Web clients. Using a RESTful API, Cloud Pub/Sub

004.5; DOI 10.2507/IJVA.3.2.1.34

distributes JSON based messages according to topics. Subscribers can either poll for new messages or register a webhook for notification. The service supports guaranteed message delivery by maintaining a queue for every subscriber, and messages are removed from the queue, when the client acknowledges the message.

Microsoft also offers two cloud-based messaging solutions: Azure Queues and Service Bus Queues [74]. Azure Queues provide direct messaging between cloud services, and they are accessible through a RESTful interface. Messages are sequences of bytes and therefore not typed similar to Microsoft SQS. Service Bus Queues offer advanced architectures such as publish-subscribe and routing patterns. Windows applications and peers can access a service bus through WCF or directly by HTTP. A Brokered Message in a Service Bus Queue explicitly refers to a user-specified message body content. Service Bus Queues also offer an AMQP interface [75].

Two cloud services that offer AMQP brokerage as a service are StormMQ [76] and IronMQ [77]. CloudAMQP specifically offers the polyglot broker RabbitMQ as a Service [78]. CloudMQTT [79] is another pay-per-use broker for MQTT messaging, e.g., for complex event processing in Internet of Things environments. Rackspace Cloud Queues [80] supports publish-subscribe architectures by a HTTP based RESTful API in the spirit of RestMS. [3]

ZeroMQ and Nanomsg

Although there are many comprehensive messaging systems available, matching with the application requirements could be very difficult and without required performance characteristics. Consequently, in recent years, a new generation of low level messaging services has appeared such as ZeroMQ [81] and Nanomsg [82].

The intelligent socket library ZeroMQ aims for more flexible connectivity between peers. ZeroMQ offers several network transports, including TCP, UDP, and IP multicast, and a number of sockets types for architectural patterns. Messages are delivered to a thread- or process-local queue and made available through a socket. The specification defines the following socket types:

- REQ and REP for bilateral Send-Receive;
- DEALER and ROUTER for routing patterns;
- PUB and SUB for publish-subscribe One-to-Many Send;
- PUSH and PULL for workload distribution through One-to-Many Send and One-from-Many Receive;
- PAIR for asynchronous Send or Receive between two sockets.

ZeroMQ has no notion of broker because it is a socket abstraction. However, a MOM broker could be implemented using ZeroMQ. Messages are sequences of bytes and do not have a specified content type. The content type needs to be agreed on out-of-band or requires a custom protocol.

An attempt to provide ZeroMQ access in Web environments is NullMQ [83]. The JavaScript library uses Web-Sockets and a modified version of STOMP to bridge ZeroMQ messages into Web browsers. ZeroRPC [84] integrates RPC on top of ZeroMQ. Information is serialized as JSON-based MessagePack format and forwarded over ZeroMQ connections. A service interface is dynamically typed, and an ZeroRPC has been used in the dotCloud PaaS.

Nanomsg, however, is a reimagining of ZeroMQ– a complete rewrite in C. It builds upon ZeroMQ's rock-solid performance characteristics while providing several vital improvements, both internal

004.5; DOI 10.2507/IJVA.3.2.1.34

and external. It also attempts to address many of the strange behaviors that ZeroMQ can often exhibit.

USE CASE - CERN

This section presents several implementations where messaging-based communication has been successfully adopted to solve the problem of exchange information in distributed system. [85]

CERN Beam Control middleware

The Beam Control department at the CERN laboratory is using messaging for highly reliable control/monitoring/alarm applications for the Large Hadron Collider (LHC). Since 2005, a cluster of ActiveMQ brokers, in a store and forward configuration, is used to collect the critical data generated by the safety systems (e.g. 30 producers, 2MB/s, 4.5K msg/s) and to forward it to many consumers (e.g. monitoring tool, dashboards). Being safety data mission critical, the store and forward configuration allow to completely decouple data production from consumption, preventing misbehaving clients to affect data collection and archiving [86]. Moreover, the LHC Control framework has been recently migrated from CORBA to ZeroMQ as communication layer [87]. [85]

DAQ Online Monitoring

Messaging has been also extensively used in several monitoring tools for Data Acquisition (DAQ) systems, which are responsible to filter and collect data from detectors (e.g. high energy physic experiments) to storage facilities. [85]

The ATLAS TDAQ shifter assistant project

It relies on messaging to distribute operational alarms from private TDAQ network to GPN to a number of heterogeneous consumers. An ActiveMQ cluster is used in a master/slave configuration in order to minimize the impact on the required firewall configuration to a single outbound connection. [88]

The STAR Online framework

It relies on an AMQP-based system for flexible, loosely coupled distribution of detector metadata, using messaging as unified transport layer for processing, storage and monitoring. Moreover, investigation has been done to re-write the control framework over MQTT, profiting from the protocol flexibility and interoperability [89]. [85]

WLCG Messaging Service

Messaging has been also successfully used on large-scale geographically distributed infrastructure. The WLCG (Worldwide LHC Computing Grid) messaging service is the backbone transport layer used for monitoring WLCG sites and services around the world, with more than 50000 clients and an average message rate of 100 KHz. The monitoring infrastructure is based on STOMP with JSON payload. Thanks to the interoperability of the STOMP protocol across several broker flavours, heterogeneous message-broker clusters (ActiveMQ, Apollo or RabbitMQ) are used in a scenario where client applications produce to any and consume to all [90]. [85]

Conclusion

Messaging is pragmatic reaction to the problem of communication in distributed systems. It allows loosely coupled communication acting as intermediate layer between producer and consumer. It brings many benefits in distributed applications flexibility and scalability, with implications in application and infrastructure complexity. Messaging systems are still evolving technology with the AMQP standardization effort pointing in the good direction, but still with partial adoption.

004.5; DOI 10.2507/IJVA.3.2.1.34

Message brokers are solid and reliable technology used as transport layer building blocks in many projects and services, both within the physics community and outside. In the recent years, a new generation of systems is promoting messaging for low-latency / high-throughput / data-intensive communication, like ZeroMQ, narrowing use cases and relaxing assumptions, but pushing the boundaries of messaging applications towards new domains and successfull implementations for demanding CERN applications.

References

[1] Curry, E. (2005). Message-oriented middleware. In: Mahmoud QH (ed) Middleware for communications. Wiley, Chichester

[2] Celar, S.; Seremet, Z. & Turic, M. (2011). Cloud computing: definition, characteristics, services and models, Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726 - 9679 ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011

[3] Lampesberger, H. (2016). Technologies for Web and cloud service interaction: a survey, Service Oriented Computing and Applications, June 2016, Volume 10, Issue 2, pp. 71-110

[4] Oracle: Java Message Service (2014). Available from: http://www.oracle.com/technetwork/java/jms/index.html Accessed: 2016-02-19

[5] GlassFish Project: Open Message Queue (2014). Available from: https://mq.java.net/ Accessed: 2016-02-19

[6] IBM: WebSphereMQ (2014). Available from: http://www-03.ibm.com/software/products/en/websphere-mq. Accessed: 2016-08-22

[7] TIBCO: Enterprise Message Service (2014). Available from: http://www.tibco.com/products/automation/enterprise-messaging/enterprisemessage-service/default.jsp. Accessed: 2016-07-21

[8] Wikidot.com: RestMS (2008). Available from: http://www.restms.org/. Accessed: 2016-02-21

[9] OpenMAMA: Introduction to OpenMAMA (2014). Available from: http://www.openmama.org/what-is-openmama/introduction-to-openmama Accessed: 2016-07-09

[10] Microsoft Developer Network: Message Queuing (MSMQ) (2014). Available from: http://msdn.microsoft.com/en-us/library/ms711472.aspx Accessed: 2016-02-20

[11] Microsoft Developer Network: SQL Server Service Broker (2014). Available from: http://msdn.microsoft.com/en-us/library/bb522893.aspx Accessed: 2016-02-23

[12] TIBCO: Rendezvous Messaging Middleware (2014). Available from: http://www.tibco.com/products/automation/enterprise-messaging/rendezvous/default.jsp Accessed: 2016-07-22

[13] Oracle: Oracle Tuxedo Message Queue Product Overview (2013). Available from: http://docs.oracle.com/cd/E35855_01/otmq/docs12c/overview/overview.html. Accessed: 2016-04-28

[14] Software AG: Terracotta Universal Messaging (2014). Available from: https://www.softwareag.com/corporate/images/SAG_Terracotta_Universal_Messaging_FS_Jun14_Web_tcm16-114090.pdf Accessed: 2016-06-04

[15] OASIS: Advanced Message Queuing Protocol v1.0 (2011). Available from: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview1.0-os.html Accessed: 2016-02-19

004.5; DOI 10.2507/IJVA.3.2.1.34

[16] OASIS: AMQP Working Group 0-10 (2014). Available from: http://www.amqp.org/specification/0-10/amqp-org-download Accessed 2016-08-06
[17] O'Hara, J. (2007). Toward a commodity enterprise middleware. Queue 5(4), 48-55. Available from: http://queue.acm.org/detail.cfm?id=1255424 Accessed: 2016-07-24
[18] Pivotal Software Inc: AMQP URI Specification (2014). Available from: http://www.rabbitmq.com/uri-spec.html Accessed: 2016-07-27
[19] O'Hara, J. (2007). Toward a commodity enterprise middleware. Queue 5(4), 48-55. Available from: http://queue.acm.org/detail.cfm?id=1255424 Accessed: 2016-07-21
[20] iMatix Corporation: OpenAMQ (2009). Available from: http://www.openamq.org/ Accessed: 2016-02-21
[21] JBoss Community: JBoss Web Services (2014). Available from: https://www.jboss.org/jbossws Accessed: 2016-03-28
[22] WSO2: WSO2 Message Broker (2014). Available from: http://wso2.com/products/message-broker/ Accessed 2016-07-23
[23] SwiftMQ: Enterprise messaging platform (2014). Available from: http://www.swiftmq.com/ Accessed: 2016-07-23
[24] Apache Software Foundation: Apache Qpid (2013). Available from: https://qpid.apache.org/ Accessed: 2016-07-21
[25] Red Hat Enterprise: MRG - Messaging, Realtime, Grid (2009). Available from: http://www.redhat.com/f/pdf/MRG_brochure_web.pdf Accessed: 2016-07-21
[26] Saint-Andre, P. (2011). Extensible messaging and presence protocol (XMPP): Address Format. RFC 6122 (Proposed Standard), Available from: http://www.ietf.org/rfc/rfc6122.txt Accessed: 2016-07-21
[27] Saint-Andre, P. (2011). Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard), Available from: http://www.ietf.org/rfc/rfc6120.txt Accessed: 2016-07-21
[28] Saint-Andre, P. (2011). Extensible messaging and presence protocol (XMPP): Instant messaging and presence. RFC 6121 (Proposed Standard), Available from: http://www.ietf.org/rfc/rfc6121.txt Accessed: 2016-07-21
[29] XMPP: IoT Systems (2013). Available from: http://wiki.xmpp.org/web/Tech_pages/IoT_systems Accessed: 2016-07-24
[30] Saint-Andre, P. & Simerda, P. (2008). XEP-0231: Bits of Binary, Available from: http://xmpp.org/extensions/xep-0231.html Accessed: 2016-07-25
[31] Adams, D. (2011). XEP-0030: Service Discovery, Available from: http://xmpp.org/extensions/xep-0009.html. Accessed: 2016-07-23
[32] Hildebrand, J.; Millard, P.; Eatmon, R. & Saint-Andre, P. (2008). XEP-0009: Jabber-RPC, Available from: http://xmpp.org/extensions/xep-0030.html Accessed: 2016-07-23
[33] Millard, P., Saint-Andre, P. & Meijer, R. (2010). XEP-0060: Publish-Subscribe, Available from: http://xmpp.org/extensions/xep-0060.html Accessed: 2016-07-23
[34] Hildebrand, J. & Saint-Andre, P. (2004). XEP-0033: Extended stanza addressing, Available from: http://xmpp.org/extensions/xep-0033.html Accessed: 2016-07-23

[35] Saint-Andre, P. & Smith, K. (2010). XEP-0163: PersonalEventing Protocol, Available from: http://xmpp.org/extensions/xep-0163.html Accessed: 2016-07-23
[36] Waher, P. (2014). XEP-0337: Event Logging over XMPP, Available from: http://xmpp.org/extensions/xep-0337.html Accessed: 2016-07-23
[37] Miller, M & Saint-Andre, P. (2005). XEP-0079: Advanced message processing, Available from: http://xmpp.org/extensions/xep-0079.html Accessed: 2016-07-23
[38] Ramsdell, B. & Turner, S. (2010). Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751 (Proposed Standard), Available from: http://www.ietf.org/rfc/rfc5751.txt Accessed: 2016-07-22
[39] Saint-Andre, P. & Simerda, P. (2008). XEP-0231: Bits of Binary. Available from: http://xmpp.org/extensions/xep-0231.html Accessed: 2016-07-25
[40] Forno, F. & Saint-Andre, P. (2005). XEP-0072: SOAP Over XMPP. Available from: http://xmpp.org/extensions/xep-0072.html Accessed: 2016-07-23
[41] Mansour Fallah, S. (2016). Multi Agent based Control Architectures, Proceedings of the 26th DAAAM International Symposium, pp.1166-1170, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-07-5, ISSN 1726-9679, Vienna, Austria
[42] Conzon, D.; Bolognesi, T; Brizzi, P.; Lotito, A; Tomasi, R. & Spirito, M. (2012). The virtus middleware: an xmpp based architecture for secure iot communications. In: 21st international conference on computer communications and networks, ICCCN'12, pp. 1-6
[43] ignite realtime: Openfire (2014). Available from: http://www.igniterealtime.org/projects/openfire/ Accessed: 2016-07-23
[44] Kaazing: WebSocket Gateway - XMPP (2014). Available from: http://kaazing.com/products/editions/kaazing-websocket-gateway-xmpp/ Accessed: 2016-07-23
[45] STOMP: The Simple Text Oriented Messaging Protocol v1.2 (2012). Available from: http://stomp.github.io/stomp-specification-1.2.html Accessed: 2016-02-19
[46] CoilMQ: Lightweight Python STOMP message broker (2012). Available from: https://github.com/hozn/coilmq/ Accessed: 2016-07-24
[47] Stampy: Java implementation of the STOMP 1.2 specification (2013). Available from: http://mrstampy.github.io/Stampy/ Accessed: 2016-07-24
[48] IBM Developer Networks: MQ Telemetry Transport (MQTT) V3.1 Protocol Specification (2010). Available from: http://www.ibm.com/developerworks/webservices/library/ws-mqtt/ Accessed: 2016-02-20
[49] OASIS: MQTT Version 3.1.1 (2014). Available from: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html Accessed: 2016-07-21
[50] HiveMQ: Enterprise Grade MQTT Broker (2014). Available from: http://www.hivemq.com/ Accessed: 2016-07-22
[51] Mosquitto: an open source MQTT v3.1/v3.1.1 Broker (2014). Available from: http://mosquitto.org/ Accessed: 2016-07-24
[52] Zhang L (2011). Building facebook messenger. Available from: https://www.facebook.com/notes/facebook-

engineering/building-facebookmessenger/10150259350998920. Accessed: 2016-06-13
[53] Object Management Group: Documents Associated With Data Distribution Services, V1.2 (2007). Available from: http://www.omg.org/spec/DDS/1.2/ Accessed: 2016-02-20
[54] Twin Oaks Computing Inc: What can DDS do for You? (2013). Available from: http://www.omg.org/hot-topics/documents/dds/CoreDX_DDS_Why_Use_DDS.pdf Accessed: 2016-07-24
[55] Object Computing Inc.: OpenDDS Developer's Guide (2014). Available from: http://download.ociweb.com/OpenDDS/OpenDDS-latest.pdf Accessed: 2016-07-27
[56] Object Management Group: Documents Associated With The Real-Time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification (DDSI), V2.1 (2009). Available from: http://www.omg.org/spec/DDSI/2.1/ Accessed: 2016-02-20
[57] Object Computing Inc.: OpenDDS Developer's Guide (2014). Available from: http://download.ociweb.com/OpenDDS/OpenDDS-latest.pdf Accessed: 2016-07-27
[58] Object Management Group: Documents Associated With DDS Security (DDS-Security) 1.0 - Beta 1 (2014). Available from: http://www.omg.org/spec/DDS-SECURITY/1.0/Beta1/ Accessed: 2016-07-25
[59] Object Computing Inc.: Welcome to OpenDDS! (2013). Available from: http://www.opendds.org Accessed: 2016-07-24
[60] RTI: Connext DDS Software (2014). Available from: http://www.rti.com/products/index.html Accessed: 2016-07-27
[61] PrismTech: OpenSplice DDS Community (2014). Available from: http://www.prismtech.com/opensplice/opensplice-dds-community Accessed: 2016-07-24
[62] Twin Oaks Computing Inc: CoreDXDDS data distribution service middleware (2014). Available from: http://www.twinoakscomputing.com/coredx Accessed: 2016-07-24
[63] Apache Software Foundation: Kafka (2014). Available from: http://kafka.apache.org/ Accessed: 2016-07-15
[64] Black, B. (2014). kafka-websocket. Available from: https://github.com/b/kafkawebsocket Accessed: 2016-08-03
[65] Apache Software Foundation: Apache ActiveMQ (2011). Available from: http://activemq.apache.org/ Accessed: 2016-02-21
[66] Pivotal Software Inc: RabbitMQ (2014). Available from: https://www.rabbitmq.com/ Accessed: 2016-02-19
[67] HornetQ: what is HornetQ? (2013). Available from: http://hornetq.jboss.org/ Accessed: 2016-07-24
[68] Apache Software Foundation: Apollo - ActiveMQ's next generation of messaging (2014). Available from: http://activemq.apache.org/apollo/ Accessed: 2016-07-24
[69] Apache Software Foundation: OpenWire version 2 specification (2011). Available from: http://activemq.apache.org/openwire-version-2-specification.html Accessed: 2016-02-20
[70] AmazonWeb Services: Amazon Simple Queue Service (Amazon SQS) (2013). Available from: http://aws.amazon.com/sqs/ Accessed: 2016-02-21
[71] Google Developers: Using Pull Queues in Java (2014). Available from: https://developers.google.com/appengine/docs/java/taskqueue/overview-pull Accessed: 2016-08-07

[72] Google Developers: Using Push Queues in Java (2014). Available from: https://developers.google.com/appengine/docs/java/taskqueue/overview-push Accessed: 2016-08-07
[73] Google Developers: Google Cloud Pub/Sub (2014). Available from: https://developers.google.com/pubsub/overview Accessed: 2016-08-07
[74] Microsoft Azure: Azure queues and service bus Queues–Compared and contrasted (2014). Available from: http://msdn.microsoft.com/library/azure/Hh767287.aspx Accessed: 2016-08-21
[75] Microsoft Azure: Windows Azure AMQP 1.0 Support in Service Bus (2014). Available from: http://www.windowsazure.com/en-us/documentation/articles/service-bus-amqp-overview/ Accessed: 2016-02-21
[76] stormmq Limited: Message queues as a service in the cloud (2013). Available from: http://stormmq.com/ Accessed: 2016-02-21
[77] Iron.io: IronMQ (2014). Available from: http://www.iron.io/mq Accessed: 2016-02-20
[78] CloudAMQP: RabbitMQ as a Service (2014). Available from: http://www.cloudamqp.com Accessed: 2016-07-21
[79] CloudMQTT: Hosted broker for the Internet of Things (2014). Available from: http://www.cloudmqtt.com Accessed: 2016-07-25
[80] Rackspace: Cloud Queues (2014). Available from: http://docs.rackspace.com/queues/api/v1.0/cq-gettingstarted/content/DB_Overview.html Accessed: 2016-07-21
[81] iMatix Corporation: ØMQ (2013). Available from: http://www.zeromq.org/ Accessed: 2016-02-19
[82] nanomsg (2016). Available from: http://nanomsg.org/ Accessed: 2016-02-19
[83] Lindsay, J.; Shakirzyanov, B. (2014). NullMQ. Available from: https://github.com/progrium/nullmq Accessed: 2016-03-10
[84] dotCloud: ZeroRPC (2013). Available from: http://zerorpc.dotcloud.com/ Accessed: 2016-02-19
[85] Magnoni, L. (2015). Modern Messaging for Distributed Sytems, Journal of Physics: Conference Series 608 (2015) 012038
[86] Ehm, F. (2011). Running a Reliable Messaging Infrastructure for CERN's Control System. Proceedings of ICALEPCS2011 (Grenoble, FRANCE)
[87] Dworak, A.; Ehm, F.; Sliwinski, W. & Sobczak, M. (2011). Middleware Trends and Market Leaders 2011. Proceedings of ICALEPCS2011 (Grenoble, FRANCE)
[88] Kazarov, A; Miotto, G. L. & Magnoni, L. (2012). The AAL project: automated monitoring and intelligent analysis for the ATLAS data taking infrastructure. Journal of Physics: Conference Series, Volume 368
[89] Arkhipkin, D.; Lauret, J. & Betts, W. (2011). A message-queuing framework for STARs online monitoring and metadata collection. Journal of Physics: Conference Series, Volume 331
[90] Cons, L. & Paladin, M. (2011). The WLCG Messaging Service and its Future. Journal of Physics: Conference Series, Volume 396