Journal of Mobile, Embedded and Distributed Systems, vol. VI, no. 4, 2014
ISSN 2067 – 4074

A Survey on Middleware Approaches for Distributed Real-Time Systems

Malihe SAGHIAN, Reza RAVANMEHR
Department of Computer Engineering
Islamic Azad University, Central Tehran Branch, Tehran
IRAN
mal.saghian.eng@iauctb.ac.ir, r.ravanmehr@iauctb.ac.ir

***Abstract**:* Nowadays distributed real-time systems are very common in different areas and have many applications. Increasing need to exchange information among distributed and heterogeneous applications, the need for faster development, simplicity of design and implementation, software reuse and coordination problems are the major factors that lead to the design and implementation of middleware architectures in this field. For this purpose, different Quality of Service parameters such as reliability, extendibility, low latency, etc. must be considered. The current survey shows the state of the art of the various researches in this domain by providing and discussing the architecture and the features of major middleware for distributed real-time systems.

***Key-Words**:* Distributed Real-time Systems, Middleware, Quality of Service

1. Introduction

Nowadays, increasing needs for distributed applications which also supports real-time behaviors in different domains is obvious. Various definitions for real time systems have been proposed in different texts [1-3], but all these definitions are referring to the systems with some levels of execution time constraints that leads to hard or soft real-time scenarios. Indeed, in these systems, time plays a critical role in system functionalities and the suitable response should be generated within a definite time limit [3].

Selecting the most appropriate middleware is one of the most successful techniques used to simplify and speed up distributed real-time systems development. The middleware is a software layer located between applications and the operating system that enables interactions and communications among different modules of applications in a distributed environment. Increasing need for interaction among distributed and heterogeneous applications, achieving lower development costs, decreased effort, coordination problems and software reuse are major factors that led to the middleware widely usage. Using middleware simplifies and accelerates the development of distributed applications and systems and provides many capabilities available to designers and developers [2-5].

Although the existing common middleware solutions are useful, but for some reasons such as unpredictability and lack of support of QoS parameters, they could not be used in distributed real-time systems. As explained before, real-time systems are performance critical and have strict and precise QoS requirements for accurate timing, predictable latency, efficiency, scalability, security, etc. [6, 7]. Therefore, there is an urgent requirement to utilize the distributed real-time middleware in distributed real-time systems that supports the existing requirements of the real-time systems and variety of QoS properties.

In recent years, interest has grown in designing a middleware that meets the requirements of distributed real-time systems. But it is important to mention that limited work has been done on surveying middleware for distributed real-time systems in general. For this purpose, we have studied different issues and challenges of distributed real-time systems and the requirements that the middleware must support for a successful development of a real-time system. Then, we focused on various middleware

designs for these types of systems and present well-suited work in this field.
The structure of this paper is organized as follows: in the next section, some related works in the field of distributed real-time middleware are reviewed. After that, the most relevant challenges faced with in middleware design for distributed real-time systems are described. Some of the most widely used distributed real-time middleware are provided in section 4. Then, in section 5, these middleware are evaluated and analyzed considering different QoS parameters and requirements. Finally, we conclude and provide the future works.

2. Related Works

In this section, we discuss some existing work and aspects related to our research in middleware for distributed real-time systems.
In recent years, various types of middleware have been emerged, each of them have focused on different applications and domains for specific problems. These middleware are classified into the following four groups [8-9]: Transactional Middleware such as Tuxedo [10], Message Oriented Middleware such as JMS [11], Publish-Subscribe [12-14] and MQSeries [16], Procedural Middleware such as RPC, and Object Oriented Middleware such as CORBA [17] and Java RMI. The main advantages and capabilities of each middleware includes adaptability in different environments, heterogeneity, scalability, reliability, ease of use, etc. [9].
Procedural middleware or RPCs (Remote Procedure Call) are the oldest type of middleware. RPC provides the ability to invoke a function or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. Transactional middleware or TP monitors were designed in order to support distributed transactions and clustering of the requested service into transactions. In Message-Oriented Middleware (MOM) exchanging of messages are used to communicate between distributed system's components. In fact in this middleware, information and data are exchanged in message format and these messages inform system's entities about some events or occurrences in other subsystems. Object-oriented middleware evolved from RPCs and extends them by adding object-oriented concepts such as inheritance, object references and exceptions [8, 9].
Despite the benefits of each middleware architectures, most of them cannot meet tough requirements of distributed real-time systems. Among the four major types of middleware, only object-oriented and message-oriented middleware have enough capability to use in distributed real-time environments. Since there is no middleware that supports all requirements of distributed real-time systems, so far many research works have been done to improve the performance of middleware, each of them have focused on some requirements.
Over the last decade, various middleware have emerged to facilitate and reduce the complexity of developing distributed real-time systems, each of them has its own pros and cons. Some of the most important and widely used distributed real-time middleware are JMS [11], DDS [19], CORBA Notification Service [22] and WSN [33-36] that uses Publish-Subscribe systems.
Publish-subscribe systems are considered as a very useful and efficient communication middleware for disseminating information. In this mechanism, messages are published by publishers and delivered to requesters subscribed to receive those messages. The strength of this event-based interaction style lies in the full decoupling in time, space and synchronization between publishers and subscribers. Actually this pattern completely decouples publishers from subscribers and they do not need to synchronize and identify each other [12-15].
Real-time CORBA is another middleware that is an extension of CORBA standard and uses ORB pattern [25-26]. Also, RTSJ [28] and TMOSM [30-31] are other middleware that have been discussed and studied in many researches and many

ISSN 2067 – 4074

attempts have been done to develop them so far.

3. Middleware Challenges for Distributed Real-Time Systems

Despite all advances in the last decade, there are significant challenges for existing middleware to meet the requirements of distributed real-time systems. One characteristic of the distributed real-time systems is the time constraints that must be met accurately and efficiently. These systems are usually heterogeneous, complex and they must support various platforms. They are usually network-centric environments with a high degree of distribution.

Some of the most challenging requirements for these systems can be characterized as follows [3-4]:

- Multiple QoS properties must be met in real-time.
- Different levels of QoS are appropriate under different configurations, environmental conditions and costs.
- The levels of QoS in one dimension must be coordinated and traded off with levels of QoS in other dimensions.

Briefly, some of the most important requirements of real-time middleware includes: timeliness, integration, heterogeneity, scalability, fault tolerance, reliability, adaptability and flexibility, managing and sharing of resources, scheduling and assigning priorities to computing and communication resources. However, it is difficult to develop a reliable and predictable real-time middleware with the full support of all mentioned requirements [6, 7].

4. Middleware Approaches for Distributed Real-Time Systems

In this section, we introduce some of the most widely used distributed real-time middlewares. For this purpose, we present the major characteristics and the core architecture of each middleware. As explained in section 2, we focus on object-oriented and message-oriented middleware.

4.1 Data Distribution Service (DDS)

DDS is a standard middleware based on publish-subscribe mechanism that has been published by the OMG group [19-21]. This specification is based on a completely decentralized architecture, and provides an extremely rich set of QoS properties that are configurable at different levels and gradations. The goal of DDS specification is to facilitate efficient distribution of data in a distributed system. Several implementations of the DDS standard are available, including OpenSplice, RTI, etc.[37, 38]

DDS specification defines two levels of interfaces: a lower level namely Data Centric Publish-Subscribe (DCPS) whose goal is efficient delivery of the proper information to the proper recipients. On top of the DCPS, it defines the optional Data Local Reconstruction Layer (DLRL), which allows for integration of the service into the application layer, automates the reconstruction of data, locally from updates received, and allows the applications to access distributed local data.

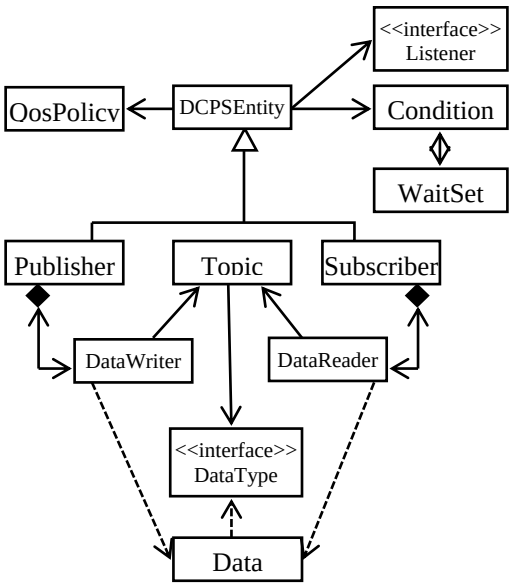

Figure 1. Overall DCPS model

Figure 1 illustrates the DCPS Schema. As shown, communication is performed using different components: DomainParticipant,

DataWriter, DataReader, Publisher, Subscriber, and Topic entities.

Publishers and subscribers are objects responsible for sending and receiving data respectively. Participants use DataWriter(s) to communicate with value publishers to change data of a given type and use a DataReader attached to subscriber to access the received data. Once new data values have been communicated to the publisher, it is the Publisher's responsibility to determine when it is appropriate to publish the corresponding message. The Publisher will do this according to its QoS, or the QoS attached to the corresponding DataWriter, and/or its internal state.
Published or subscribed data is identified by Topics. Each topic associates a unique name, a data type, and QoS related to the data and communication of DataWriter with DataReader's objects is done by means of it.
All above classes extend *DCPSEntity*, and so all of them have a certain type of *listener*, *Condition* objects and a set of *QoSPolicy* values that are suitable for it. In this specification, there are two methods to read and access data: In the Listener-based approach, the participant is notified of the events occurrence and appearance or change in value of data by means of a listener object. But in the wait-based approach, threads inside the participant block while waiting for occurrence of specific changes that is specified in *Condition*.

4.2 JMS

JMS [11, 18] is an API for accessing messaging systems from Java programs that has been developed by sun microsystems. Actually, this middleware is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of a messaging system.
This middleware supports two types of messaging models: point-to-point model is used for one-to-one delivery of messages and message exchange is performed by using the queues. But in publish-and-subscribe model, messages exchange is performed via an intermediate node namely topic, and Publishers and subscribers are anonymous to each other. Therefore, this model is quite decoupled and is used for one-to-many delivery of messages.
API and interfaces that JMS defines to communicate with messaging systems, contains three main types: the general API, the point-to-point API, and the publish-and-subscribe API. In the general API, there are seven main interfaces related to messages transmission that can be used to send and receive messages from either a queue or a topic. The relationship between these seven interfaces is illustrated in Figure 2. Point-to-point and publish-and-subscribe interfaces are quite similar with general API in terms of functionality, but the point-to point API is used solely for messaging with queues, and the publish-and-subscribe API is used solely for messaging using topics.

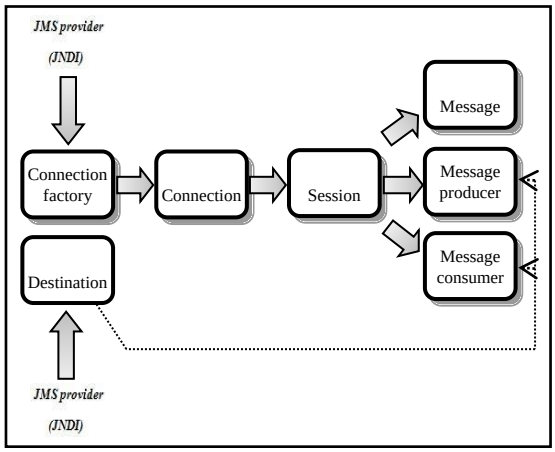

Figure 2. JMS general API

Connection is the first object created within this middleware that is created by Connection Factory interface. This object represents an active connection of client to a provider and it is used to create sessions with the desired characteristics. Session is a single-threaded context for sending and receiving messages that is responsible for creating multiple messages, MessageProducer and MessageConsumer. Finally created MessageProducer/ MessageConsumer objects are used for sending/ receiving messages to/from a destination. Because sessions control transactions, concurrent access by multiple threads is restricted

ISSN 2067 – 4074

and multiple sessions can be used for multi-threaded applications.

4.3 CORBA Notification Service

This specification that has been standardized by the OMG is an extension of CORBA Event Service [24] that in addition to using publish-subscribe mechanism, provides capabilities such as advanced events filtering mechanism and QoS properties support in different levels [22,23].

Figure 3 shows the components in the CORBA Notification Service. The architecture of this middleware is similar to the architecture of the CORBA Event Service, though some components' capabilities have increased in the Notification Service.

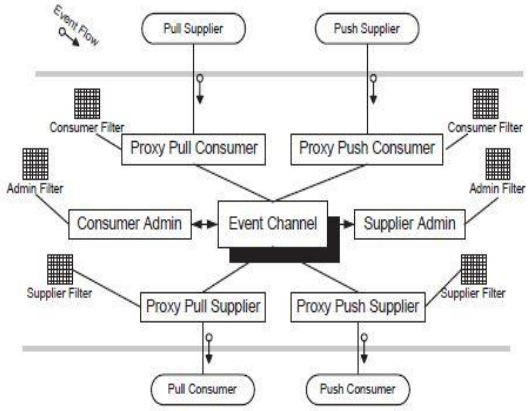

Figure 3. Components in the CORBA Notification Service

An event channel is the first object created within the Notification Service process that is created by EventChannelFactory interface. Various QoS and administrative properties can be set on an Event Channel during its creation. The suppliers for connecting to the channel, obtain a ProxyConsumer and connect to it and finally send the events to their ProxyConsumer, whereas each consumer obtains a ProxySupplier and connects to it and receives events from its ProxySupplier objects. The proxy objects decouple the communication between suppliers and consumers. Proxies are created by admin objects that, in turn, are created by a channel event.

Filter objects contain a set of constraints that affect the event forwarding decisions. These objects can be associated with admin and proxy objects, so clients can be easily subscribed to their interested events and receive them.

Each object in this middleware can have its own filter objects, QoS and administrative properties. Also the set of filter objects associated with a given Admin object applies to all Proxy objects which that Admin creates. With this work, the filtering of a given event on behalf of a set of clients can be optimized since the same subscription information applies to multiple clients. This feature is particularly useful for ConsumerAdmin objects, since it enables the channel to optimize the servicing of a group of consumers that are interested in receiving the same set of events.

4.4 Real-time CORBA

The Real-time CORBA specification is standardized by the OMG to support the QoS requirements of embedded and distributed real-time systems [25-27]. This specification that extends the existing CORBA standard [17], provides features that allow applications to allocate, schedule, configure and control CPU, communications and memory resources.

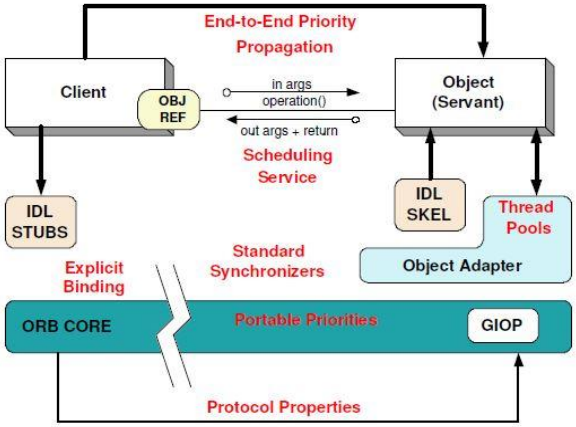

Figure 4. Real-time CORBA

Architecture of this middleware has been shown in figure 4. In this middleware, ORB is the central component that is responsible for making communications between clients and servers transparent and for allowing interoperability between applications in hetero- and homogeneous environments. An application sends

requests to an ORB, which directs the request to an appropriate object that provides the desired service. Actually ORB acts as an intermediary which allows the object requestor to access multiple remote or local objects.
Real-Time CORBA provides standard features that allow applications to manage system resources:

- Managing Processor Resources via thread pools, priority mechanisms, intra-process mutexes, and a global scheduling service.
- Managing communication resources via selection and configuration of protocol properties and explicit bindings.
- Managing memory resources via buffering requests in queues and bounding the size of thread pools.

4.5 RTSJ (Real-Time Specification for Java)

RTSJ specification [28, 29] includes seven main features: scheduling, memory management, synchronization, asynchronous event handling, asynchronous control transfer, asynchronous thread termination and access to physical memory. These features are seven major areas in RTSJ that provide a real-time software development platform suitable for a wide range of applications.
RTSJ uses thread priority for scheduling, and therefore when there is competition to use processing resources, threads with higher priority are generally executed in preference to threads with lower priority. This specification extends the Java memory model to prevent the unpredictable delays generated by the garbage collector. Thus, RTSJ introduces the notion of memory areas such as Scoped Memory and Immortal Memory that are outside the heap memory and can be used to allocate objects out of heap, and so they do not suffer from garbage collectors delays. In Immortal Memory, objects will exist until the end of the application lifetime and will not be removed by garbage collectors. But objects in scoped memory have a limited lifetime. This specification provides classes that allow direct access to physical memory and ability to create objects in physical memory.
Sometimes occurred changes and conditions are so that the current point of execution should be transferred to another location. The RTSJ supports this feature using Asynchronous Transfer of Control (ATC). Also this specification performs asynchronous real-time thread termination through a combination of the asynchronous event handling and the asynchronous transfer of control mechanisms without the risk of the `stop()` or `destroy()` methods. In addition, RTSJ uses the Priority Inheritance Protocol and Priority Ceiling Protocol for synchronization to avoid the priority inversions.

4.6 TMOSM (TMO Support Middleware)

TMOSM is an efficient middleware that supports TMO (time-triggered message-triggered object) [32] and can be adapted easily for many platforms [30, 31]. TMO specification is a unified approach for designing and implementation of real-time and non-real time distributed applications. It is simple and syntactically small, but semantically it is a powerful extension of the object-oriented design. The basic TMO structure consists of four primary parts:

- SpM and SvM: spontaneous methods are triggered automatically when the clock reaches specific values determined at design time, whereas the SvMs are conventional service methods that are called by service request messages from clients. Activation of a SvM is allowed only when conflicting SpM executions are not active.
- ODS: is object-data-store section that includes the shared data between methods of a TMO.
- EAC (Environment access capability section): These "gate objects" provide efficient call-paths to remote object methods.

ISSN 2067 – 4074

- In TMOSM middleware, the inter-node communication is based on the UDP broadcasts, neither ORB services nor point-to-point protocols. Architecture of this middleware is shown in Figure 5.

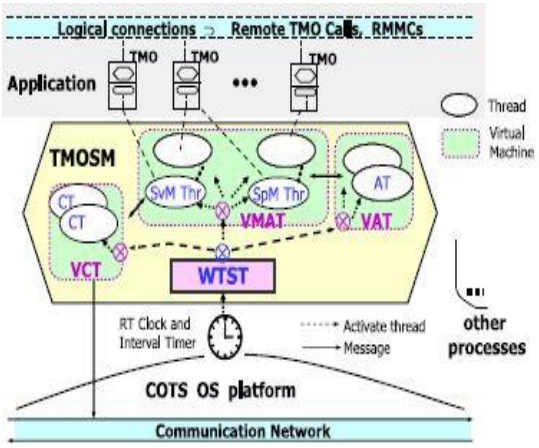

Figure 5. TMOSM Architecture

TMOSM consists of a number of virtual machines that each VM is responsible for part of the TMOSM functionality and manages a set of threads. Exchanging and distribution of messages, local I/O activities, executing methods of application (SpM or SvM), scheduling and activation of all threads are important parts of TMOSM functionality that are managed by ICT, LIIT, VMST and WTST threads respectively.

4.7 Web Service Notification (WSN)

WSN family of specifications defines a set of related, interoperable and modular specifications including WS-Base Notification [33], WS-Topics [34], and WS-Brokered Notification [35]. These specifications that are developed by OASIS technical committee, allow the notification pattern to be modeled in a standardized fashion [36].

This specification uses publish-subscribe mechanism and it includes entities such as publisher, notification producer, consumer and subscriber. Publishers create notification messages that are sent to the appropriate consumers by notification producer. Notification producer acts as a broker that is responsible for maintaining a list of interested consumers, arranging messages to be sent and matching notifications against subscriptions. The subscriber creates a subscription for notification consumer by sending a subscribe request message to a notification producer, and finally notification producer sends notifications to the relevant consumers.

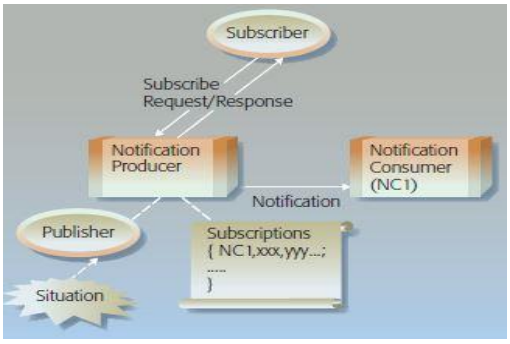

Figure 6. Web Services Notification entities

WS-Base Notification provides the foundation for the WSN family of specifications and other WSN specifications are dependent on it. It defines the basic roles needed to express the notification pattern and it can be used in combination with the WS-Topics and WS-Brokered Notification.

In WSN, topics are described in the WS-Topics specification. A topic is a concept used to describe and categorize types of notifications. In WS-Brokered Notification, responsibility of matching notifications against the list of subscriptions and sending notification messages to each appropriately subscribed consumer, delegated to another entity, notification broker. This entity performs some tasks of a notification producer.

5. Evaluation and Comparison

Quality of service properties have an important role in the performance of middleware, so supporting them in distributed real-time middleware has been widely studied. Some of the most important QoS requirements in this field are reliability, meeting deadlines, scalability, overhead, etc. In addition, these middleware can be evaluated and compared in terms of processors and

memory allocation, security, bandwidth, latency, rate of missed deadline, load balancing, and priority consideration [39]. Since in each existing middleware, part of these requirements are addressed and there is no middleware that supports all of these QoS properties or resolves all problems, therefore in the rest of the paper we review the studied middleware based on these QoS requirements. We also identify the advantages and disadvantages of each approach.

Since the publish-subscribe communication paradigm provides a high degree of decoupling between entities, middleware such as DDC, JMS, CORBA Notification Service and WSN that utilizes this mechanism, have a high degree of flexibility and scalability. In addition, due to decoupling of entities and asynchronous mode of communication, transmission requirements and consequently cost and bandwidth consumption have been reduced. Therefore, these types of middleware are more suitable to use in dynamic environments. Multicast nature of publish-subscribe mechanism leads to reduced communication facilities and response time. Meanwhile, this decoupling between entities in publish-subscribe paradigm, may generate issues such as lack of reliability and security.

In middleware approaches such as Real-time CORBA that uses point-to-point communication, we face to severe problems such as traffic, bottleneck, reduction of scalability and flexibility. Table 1 summarizes the result of analysis of different features, pros and cons of the middleware approaches we presented in previous sections.

Table 1. Evaluation and Comparison of introduced middleware approaches

Middleware / Features	**DDS**	**JMS**	**CORBA Notification Service**	**Real-time CORBA**	**TMOSM**	**RTSJ**	**WSN**
Real-time Mode	Extreme Real-time	Soft Real-time	Soft Real-time	Hard Real-time	Hard Real-time	Hard Real-time	Soft Real-time
Architecture	Publish-Subscribe (multicast)	Publish-Subscribe	Publish-Subscribe	Point to point	Broadcast or multicast	Point to point	Publish-Subscribe
Portability and platform independence	Yes	Yes	Yes	Yes	Yes	Yes	Yes
support of Configurable QoS	Excellent	Limited	Appropriate	Limited	Limited	Limited	Limited
Main Features and Capabilities	Ability to control and management of resources Definition of QoS Policies	Supporting two messaging models	Advanced filtering mechanism Definition of QoS Policies	Ability to control and management of system resources	Simplicity and affordability of implementation and maintenance	New memory management and scheduling models	Definition a set of interoperable and modular specifications
Disadvantages	Difficulty in guarantee of data delivery and ordering events	The neglect of handling security, fault tolerance, error notification and load balancing Concurrenc	Nonconformity to QoS consistency across a path The neglect of handling security and load balancing Usage restriction to	Unsuitable for Highly dynamic environments Complexity and difficult learning Weak support of security,	Unsuitable for Highly dynamic environments Weak load balancing, fault tolerance, and security Lack of attention to	Weak support of scalability, fault tolerance, load balancing and security High consumptio	Weak support of reliability, fault tolerance, load balancing and security

ISSN 2067 – 4074

		y problems	applications with stringent QoS requirements	scalability and fault tolerance	priorities	n of memory and memory leakage	

- **DDS:** One of the key distinguishing features of the DDS when compared to other middlewares is its extremely rich QoS support. In this middleware, ability to control and improve most of QoS properties has been provided easily. DDS makes it possible to control and restrict the use of resources such as network bandwidth and memory, and also provides non-functional properties such as durability, response time, reliability, flexibility, scalability, etc. This specification is very appropriate and flexible for dynamic environments with a large number of nodes. However, there are some issues such as: ordering of events that comes from different publishers, guarantee of data delivery and security.
- **JMS:** JMS specification, despite utilization of benefits of the publish and subscribe mechanism, does not address issues such as security, fault tolerance, error notification and load balancing. This specification does not specify an API for controlling the privacy and integrity of messages. These features are considered JMS provider-specific. In addition, this middleware restricts using and concurrent accessing to sessions and using multi-threading benefits. The required concurrency can be achieved using multiple sessions that will lead to more overhead.
- **CORBA Notification Service:** The advanced mechanism that this middleware provides for event filtering will lead to more control for published events, simpler management of clients, reducing network traffic and response time. In addition, this specification allows on-demand notification; this feature will lead to reduced traffic, bandwidth and memory consumption. Specific QoS properties and policies for reliability, ordering/discarding of events, etc. allow clients to adjust and improve these properties based on their needs and provide the ability of management and restriction of resource consumption to applications. However, since setting QoS properties is done in different levels, it causes the problem of inconsistent QoS across a path from supplier to consumer. Furthermore, due to the lack of bounds on the number of filters or the complexity of evaluating each filter, filter processing itself could consume an excessive and unpredictable amount of time, leading to deadline failures for delivery and processing of notifications. As a result, applicability of this specification is limited to applications with precise QoS requirements, such as real-time deadline guarantee.
- **Real-time CORBA:** This specification improves the distributed systems predictability by bounding the priority inversions and managing end-to-end system resources and allow configuring and controlling the system resources including processor, communication and memory resources. Other advantages of this middleware are flexibility in implementation, low latency and reduced priority inversions. In addition, specific thread pool in this middleware is very useful to benefit from multi-threading and to control and restrict memory consumption. Real-time CORBA suffers from issues such as the complexity and inadequate

support for extensibility. Most implementations of this middleware are done in C+ +, that is complex and error-prone. This middleware requires IDLs that are not defined in some languages yet and much time is required to learn them.

- **RTSJ:** In addition to low complexity, this specification, is easy to use and design patterns that have high levels of predictability and reliability. RTSJ gives implementers, high flexibility and ability to use arbitrary scheduling algorithms. Therefore, it is executable on many computing platforms. Thus, most important features of this specification can be referred to as portability, timeliness and low latency. The problems that exist in this middleware are that it does not support private connections and pre-established connections, and still there are problems related to excessive use of memory and memory leak in it.
- **Web Service Notification:** some of the most important goals of WSN is to define a set of related, interoperable and modular specifications that allow the notification pattern to be modeled in an explicit and standardized fashion. This specification has a good level of interoperability and portability. Organizing topics in hierarchical form will lead to facilitating the identification of topics, reduce the response time and the time required for filtering. But despite the advantages of using topic trees, the need of its maintenance and updating will cause the increase of costs and memory consumption. Furthermore, in this middleware, there are other problems in terms of security, reliability and fault-tolerance.
- **TMOSM:** TMOSM middleware support TMO and thus has the advantages of implementing the TMO. TMO is simple and syntactically small, but semantically it is a powerful extension of the object-oriented design. The main goal of the TMO is significant reduction in costs of development and maintenance of large-scale real-time systems and substantial increases in the overall robustness of systems. Timeliness, affordability of implementation and maintenance and proper portability are other advantages of TMOSM. This middleware can be easily implemented with small effort on top of any OS which has the essential features. Since in this middleware the inter-node communication is based on UDP protocol and data channels are implemented by using the connectionless UDP/IP broadcast protocol, there is no overhead due to creating and destroying a connection. However, it leads to a significant reduction in reliability. Moreover, this middleware suffers from problems such as load balancing, fault tolerance and security.

6. Conclusion

In this paper, we presented an overview of the concepts of distributed real-time middleware and their requirements and challenges. After studying some of the most widely used distributed real-time middleware, we proceeded to evaluate and compare them. Each of presented middleware approaches have different applications and solve specific problems. However, according to previous studies, it is obvious that none of these middleware approaches could not meet all requirements and QoS attributes of distributed real-time systems. In fact, each one focuses on some of these QoS parameters.

Considering our studies and evaluations in previous sections of paper, we found that the middleware based on publish-subscribe mechanism have significant benefits and features for real-time distributed systems. Among the existing publish-subscribe middleware, DDS

ISSN 2067 – 4074

specification is able to handle many of the problems and requirements of distributed real-time systems. Efficient distribution of data with minimal overhead and the ability to control Quality of Service are among the most important issues that have been addressed in this middleware. In addition, extreme rich set of QoS policies in this middleware solves most of the existing problems in distributed real-time environments.

References

[1] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*, Springer, 2011.

[2] P. A. Laplante and S. J. Ovaska, *Real-time systems design and analysis: tools for the practitioner*, John Wiley and Sons, 2011.

[3] D. C. Schmidt, R&D Advances in Middleware for Distributed Real-time and Embedded System, *Communication of ACM special issue on Middleware*, 2002.

[4] R. E. Schantz and D. C. Schmidt, Middleware for distributed systems: Evolving the common structure for network-centric applications, *Encyclopedia of Software Engineering*, Vol.1, 2002.

[5] L. Jingyong, Z. Yong, C. Yong, and Z. Lichen, Middleware-based distributed systems software process, *International Conference on Hybrid Information Technology*, 2009, pp. 345-348.

[6] D. C. Schmidt, A. S. Gokhale, R. E. Schantz, and J. P. Loyall, Middleware R&D challenges for distributed real-time and embedded systems, *SIGBED Review*, vol.1, no.1, 2004, pp. 6-12.

[7] D. C. Schmidt, Adaptive and reflective middleware for distributed real-time and embedded systems, *in Embedded Software*, 2002, pp. 282-293.

[8] L. Qilin and Z. Mintian, The state of the art in middleware, *International Forum on Information Technology and Applications (IFITA),* Vol.1, 2010, pp.83-85.

[9] W. Emmerich, Software engineering and middleware: a roadmap, *in Proceedings of the Conference on The future of Software engineering*, 2000, pp. 117-129.

[10] Oracle Corporation, Oracle Tuxedo Product Overview, http:// docs.oracle.com, 2012.

[11] M. Richards, R. Monson-Haefel, and D. A. Chappell, Java message service, O'Reilly Media, Inc, 2009.

[12] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, The many faces of publish/subscribe, *ACM Computing Surveys (CSUR),* vol. 35, no.2, 2003, pp.114-131.

[13] C. Esposito, D. Cotroneo, and S. Russo, On reliability in publish/subscribe services, *Computer Networks*, vol.57, no.5, 2013, pp. 1318-1343.

[14] A. Corsaro, L. Querzoni, S. Scipioni, S. T. Piergiovanni, and A. Virgillito, Quality of service in publish/subscribe middleware, *Global Data Management*, vol.19, 2006, pp. 1-20.

[15] S. Oh, J.-H. Kim, and G. Fox, Real-time performance analysis for publish/subscribe systems, *Future Generation Computer Systems*, vol.26, no.3, 2010, pp. 318-323.

[16] C. Aranha, C. Both, B. Dearfield, C. L. Elkins, A. Ross, J. Squibb, et al., *IBM WebSphere MQ V7.1 and V7.5 Features and Enhancements*, IBM, 2013.

[17] Object Management Group, The Common Object Request Broker Architecture Specification, Version 3.3, http://www.omg.org, 2012.

[18] W. Farrell, Introducing the Java Message Service, IBM, 2004.

[19] Object Management Group, Data Distribution Service for Real-time Systems, Version1.2, http://www.omg.org, 2007.

[20] G. Pardo-Castellote, OMG data-distribution service: Architectural overview, *International Conference on Distributed Computing Systems Workshop*, 2003, pp. 200-206.

[21] M. Xiong, J. Parsons, J. Edmondson, H. Nguyen, and D. C. Schmidt, Evaluating the performance of publish/subscribe platforms for information management in distributed real-time and embedded systems, http:// www.omgwiki.org, 2010.

[22] Object Management Group, Notification Service Specification, Version1.1, http://www.omg.org, 2004.

[23] P. Gore, I. Pyarali, C. D. Gill, and D. C. Schmidt, The design and performance of a real-time notification service, *Real-Time and Embedded Technology and Applications Symposium*, 2004, pp. 112-120.

[24] Object Management Group, Event Service Specification, Version1.2, http://www.omg.org, 2004.

[25] Object Management Group, Real-time CORBA Specification, Version1.2, http://www.omg.org, 2005.

[26] D. C. Schmidt and F. Kuhns, An overview of the real-time CORBA specification, *Computer*, vol.33, no.6, 2000, pp. 56-63.

[27] A. S. Krishna, D. C. Schmidt, R. Klefstad, and A. Corsaro, *Real-time CORBA middleware, Middleware for Communications*, John Wiley & Sons, 2004, pp. 413-435.
[28] Java Expert Group, Real-Time Specification for Java, Version 1.0.2, http://www.rtsj.org, 2008.
[29] G. Bollella and J. Gosling, The real-time specification for Java, *Computer*, vol.33, no.6, 2000, pp. 47-54.
[30] K. Kim, M. Ishida, and J. Liu, An efficient middleware architecture supporting time-triggered message-triggered objects and an NT-based implementation, *2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 1999, pp. 54-63.
[31] S. F. Jenks, K. Kim, Y. Li, S. Liu, L. Zheng, M. H. Kim, et al., A middleware model supporting time-triggered message-triggered objects for standard Linux systems, *Real-Time Systems*, vol.36, no.1-2, 2007, pp. 75-99.
[32] K. Kim, Object structures for real-time systems and simulators, *Computer*, vol.30, no.8, 1997, pp. 62-70.
[33] OASIS Technical Committee, Web services base notification 1.3, http://docs.oasis-open.org/wsn, 2006.
[34] OASIS Technical Committee, Web services topics 1.3, http://docs.oasis-open.org/wsn, 2006.
[35] OASIS Technical Committee, Web services brokered notification 1.3, http://docs.oasis-open.org/wsn, 2006.
[36] P. Niblett and S. Graham, Events and service-oriented architecture: The oasis web services notification specification, *IBM Systems Journal*, vol.44, no.4, 2005, pp. 869-886.
[37] PrismTech, OpenSplice DDS, Available on: http://www.prismtech.com/opensplice
[38] Real Time Innovations, Connext DDS, Available on: http://www.rti. com
[39] Wang, N., D. Schmidt, et al. *QoS-enabled middleware, Middleware for communications,* John Wiley & Sons, 2004, pp. 131-162.