QUICKMQ Provider for Java Message Service

Prof. Sushila Aghav
Professor, MITCOE
Kothrud, Pune, India

Swapnil S. Mahajan
MITCOE
Kothrud, Pune, India

Swapnil M. Gaikwad
MITCOE
Kothrud, Pune, Inda

Subodh S. Karwa
MITCOE
Kothrud, Pune, Inida

ABSTRACT

QuickMQ is asynchronous enterprise messaging system. It is one of the providers of Java Message Service API, which implements Sun Microsystems Java Message Service API 1.1 Specification. QuickMQ provides a powerful messaging platform, with powerful and unique features including dynamic routing, dynamic deployment, scalability, reduces system bottlenecks along with heterogeneous system integrations.

The primary advantage of using QuickMQ is it is a lightweight Messaging systems compared to other JMS providers which adds to its performance. Also, it also implements both producer-subscriber and point to point messaging models

Keywords— JMS, QuickMQ, Message Queue, Point to Point Messaging, MOM

1. INTRODUCTION

As in this kind of heterogeneous world there are so many programming languages, script languages and platforms each having their own distinct characteristics, for example 'Shell Script' mainly good enough in dumping up for backups and other side it is not good as it need to launch a new process for each new shell command executed also has portability issues which can be removed through using platform independent languages like Java, Dot net, thus every language has its own advantages and disadvantages. If an enterprise wants to develop application which requires efficient back up technique and also it need to be portable i.e. platform independent, then it to satisfy desired needs it requires to import specific characteristics from different programming languages. How this can be achieved? Earlier there were solutions like RMI(Remote Method Invocation) but this solution was rather synchronous in nature i.e. system would get blocked and was unable to process other tasks. Then came the concept of MOM (Message Oriented Middleware) which mainly uses messaging to communicate between two systems that mainly remove synchronous problems.

As now a day's messaging has become an important part of communication between different users and systems therefore large numbers of way of communication as been invented. As messaging supports loose coupling [4] between two systems it has got lot of importance mainly in heterogeneous integration and communication. Over the years system has grown in terms of complexity and sophistication [4], to have system more reliable, scalable and flexible, it has given rise to system which is having more complex and sophisticated architecture [1]. In order to increase in demand for better and faster systems, messaging is found to be one way of solving these complex problems.

The Application-to-Application system which is used in business system is generally referred to as Enterprise Messaging System or Message Oriented Middleware (MOM) [8].

QuickMQ is lightweight MOM which mainly supports both messaging models Point To Point and Publish-Subscribe. This QuickMQ MOM is mainly used to transfer messages from one application to another across network. It ensures that all messages are properly distributed among applications, although different Enterprises uses different formats and network protocols for exchanging messages, but their semantics are same. QuickMQ uses standards of SUN Microsystems JMS API to create message, load information in format, assign routing information and send message. The same API is used to receive message on other end. Thus QuickMQ increases reusability by reusing same API for different systems. Enterprise Messaging products are also available such as IBM Websphere MQ, SonicMQ, Microsoft Message Queuing and TIBCO Rendezvous are being in use for many years. Recently open source messaging products such as ActiveMQ have entered market and are being used in Enterprise environment, but all these need to be embed in whole console, that is they come in complete package of functionality while some of functionality might not be used, this in turn increases cost and memory requirement of product, also these products do not provide any feature to have customization of required functionalities.

In case middle scale Enterprises, some these functionalities are not being used and those functionalities are mandatory to be taken in whole console also these functionalities are running in background irrespective of their use, as middle scale enterprises mainly focuses on resource efficient products. For example 'Distributed Transactions' are mainly used in critical business applications like banking transactions were these transactions are need to Atomic(ACID property) [8] these can be achieved through 'Commit and Rollback', which results in more processing time and large memory space. But in middle scale enterprises do not always process critical business application for example simple news broadcasting or sports updates do not require to use ACID properties, which leads to unnecessary usage of resources.

2. BACKGROUND OF PROBLEM

Now days RMI (Remote Method Invocation), RPC (Remote Procedure Call) are mainly used communication in heterogeneous systems, but these methods are synchronous and are tightly coupled [5], we can't do any other kind processing till the current operations gets completed, so we have to wait till current operation completes, which may lead to starvation in some cases[4]. These synchronous methods may affect performance of system in case of large record processing.

Sometimes when we need to upload certain big record like CSV(Comma Separated Files) files of say 1,00,000 elements then till all elements are not being acknowledged we cannot process any other request, which may lead to blocking of system and starvation.

In tightly coupled systems there are many-to-many problems of managing connections between these systems [4], when you want to add some more application to mix, we need to go back and let all other system know about it, which is rather more tedious and leads to problems in up- gradation of system. Also if some part of system goes down, everything halts.

In this case there is more load on server as all processing load is being handled by server because of which leads to system bottlenecks and system halts.

3. WHY "QUICKMQ"

The main purpose of creating new implementation of Java Message Service is its weight. Most of the implementations of JMS are very costly and bulky consisting of many features, which are not required for many small or middle level organizations. Such implementations also do not provide any provision to turn off those features. So we targeted those small and middle level organizations and came up with elegant solution QuickMQ. It provides absolutely necessary and sufficient features with very efficient implementation of those. So to be abstract its

"As Light As 'Light' and As Fast As 'Light"

In QuickMQ messaging system applications exchange messages through virtual channels called destinations (i.e. queue or a topic). When message is sent, it's addressed to a destination, not a specific application. Any application that subscribes or registers an interest in that destination may receive the message. In this way the applications that receive messages and those that send messages are decoupled. Senders and receivers are not bound to each other in any way and may send and receive messages as they see fit.

QuickMQ being provider it provides MOM facilities which reduces load on server and reduces bottlenecks in turn, .QuickMQ being asynchronous, we don't have to wait till completion of operations, as in case synchronous uploading of CSV files of 10,000 records we need to upload all records first and then perform processing on them, but till all records gets uploaded it needs to wait and cannot process another processing's, this problem is now being solved in QuickMQ by using asynchronous message communication in which some amount of records can be uploaded and can perform processing's on those uploaded records and can also upload remaining records simultaneously, which may lead to save much more time and does not lead to halt of system.

4. JMS API ARCHITECTURE

JMS API for enterprise messaging created by SUN Microsystems. It is not just messaging system itself; but it is abstraction of the interfaces and classes needed by messaging clients when communicating with messaging system [1].

Figure 1 illustrates the way these parts interact. Administrative tools allow you to bind destinations and connection factories into a Java Naming and Directory Interface (JNDI) API namespace [1].

A JMS client can then look up the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.

5. MESSAGE DOMAINS

Before the JMS API existed, most messaging products supported either the point-to-point or the publish/subscribe approach to messaging. The JMS Specification provides a separate domain for each approach and defines compliance for each domain. A standalone JMS provider may implement one or both domains. A J2EE provider must implement both domains. In fact, most current nt implementations of the JMS API provide support for both the point-to-point and the publish/subscribe domains, and some JMS clients combine the use of both domains in a single application. In this way, the JMS API has extended the power and flexibility of messaging products.

5.1 Point to point messaging domain

A point-to-point (PTP) product or application is built around the concept of message queues, senders, and receivers. Each message is addressed to a specific queue, and receiving clients extract messages from the queue(s) established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire.

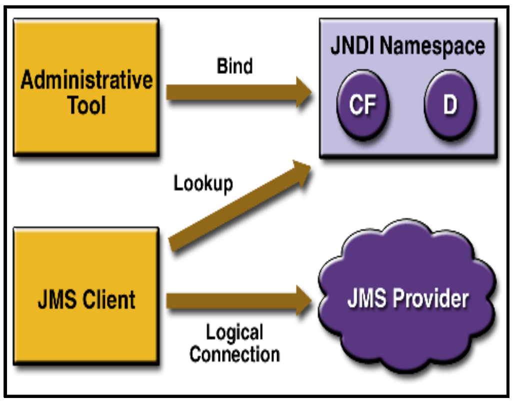

Fig. 1 JMS API Architecture

PTP messaging has the following characteristics and is illustrated in Figure 2. The common example of point to point messaging model is banking transaction in which transactions like crediting for debiting money from his own account. As one person performs operations on his accounts it is one to one communication.

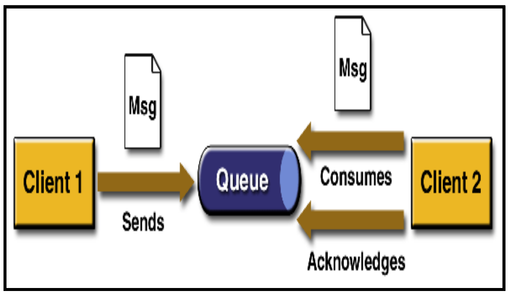

Fig. 2 Point-to-Point Messaging

5.2 Publish/Subscribe Messaging Service

In publish/subscribe (pub/sub) application; clients address messages to a topic. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy [1]. The system takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers. Topics retain messages only as long as it takes to distribute them to current subscribers.
Pub/sub messaging has the following characteristics.

- Each message may have multiple consumers.
- Publishers and subscribers have a timing dependency.

A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages [1]. The JMS API relaxes this timing dependency to some extent by allowing clients to create durable subscriptions.

Durable subscriptions can receive messages sent while the subscribers are not active. Durable subscriptions provide the flexibility and reliability of queues but still allow clients to send messages to many recipients [4].

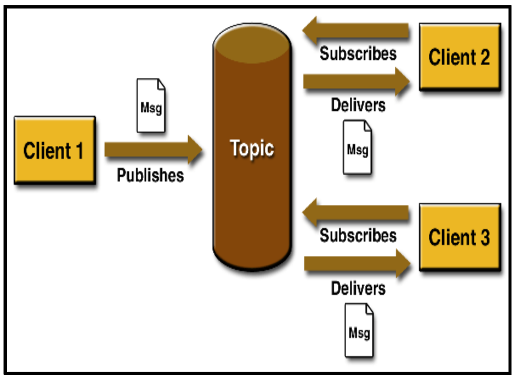

Fig. 3 Publish/Subscribe Messaging

Use pub/sub messaging when each message can be processed by zero, one, or many consumers. Figure 3 illustrates pub/sub messaging.

The common example for pub/sub model is stock update in which main server at Mumbai stock exchange will publish stock updates to virtual channel called as Topic which is being subscribed by more than one subscriber. Thus it seems to be one-to-many communication.

6. JAVA NAMING AND DIRECTORY (JNDI)

JMS clients look up configured JMS objects using the JNDI API [7]. JMS administrators use provider-specific facilities for creating and configuring these objects. This division of work maximizes the portability of clients by delegating provider-specific work to the administrator. It also leads to more administrable applications because clients do not need to embed administrative values in their code [12]. JNDI is used mainly in naming services to locate administered objects, and administered objects are objects that are created by system administrator [4]. These administered objects are bound to name in naming service. A naming service associates name with distributed objects, files, and devices so that they can be located on network using names instead of cryptic network address [6].

Advantages of Using JNDI

1. It hides provider-specific details from JMS clients.
2. It abstracts JMS administrative information into Java objects that are easily organized and administrated from a common management console [12].
3. Since there will be JNDI providers for all popular naming services, this means JMS providers can deliver one implementation of administered objects that will run everywhere. [6] Thereby eliminating deployment and configuration issues.

7. PROPOSED WORK

We have developed a “QuickMQ” messaging platform that facilitates asynchronous messaging between heterogeneous systems. We analysed many implementations of the JMS specification and gathered the information about the features that are not required by small and middle level organisations. We eliminated those features in the implementation of QuickMQ but still maintaining the high degree of performance [2].

7.1 Implementation

As QuickMQ is mainly provider for java message service the name itself suggest that it is mainly implemented in java as specifications are available in java language. J2EE Version 1.5 is used in implementation of QuickMQ. As shown in fig. 4

1) Connection Factory - an administered object used by a client to create a Connection.
2) Connection - an active connection to a JMS provider.
3) Destination - an administered object that encapsulates the identity of a message destination or ca be called as virtual channel for communication.
4) Session - a single-threaded context for sending and receiving messages.
5) Message Producer - an object created by a Session that is used for sending messages to a destination.
6) Message Consumer - an object created by a Session that is used or receiving messages sent to a destination.

7.2 Performance

Though, we are providing customization of functionality but still our performance has been maintained in comparison to other messaging products.

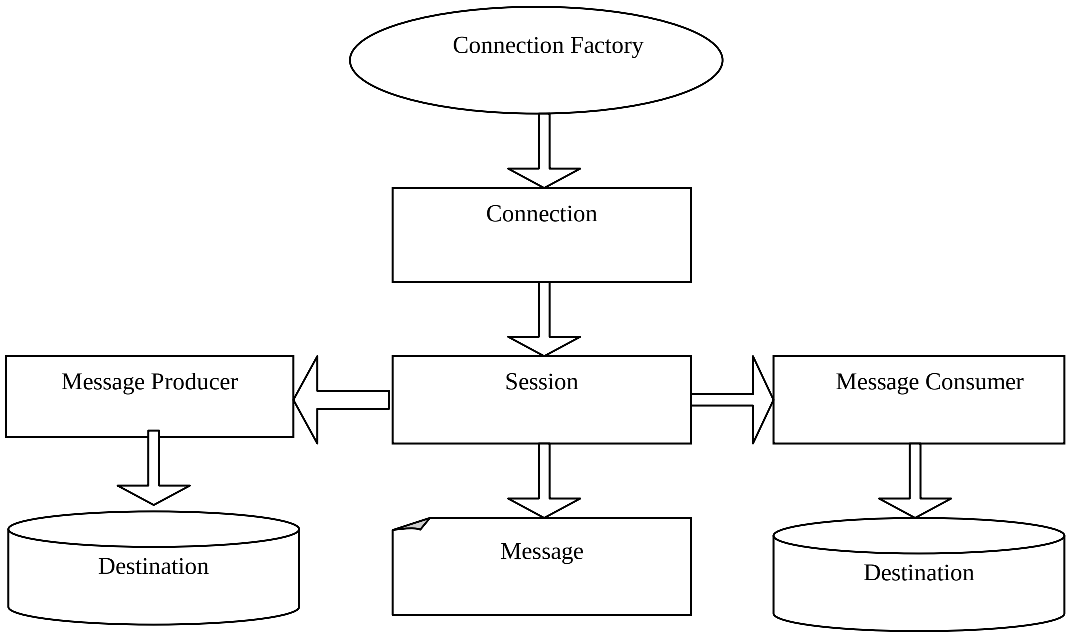

Fig. 4 LightMQ Specifications

7.2.1 Effect of the Number of Publishers

We study the effect of the number of publishers on the message throughput. Two different machines carry a varying number of publishers and one machine hosts 1 subscriber. The throughput of received and dispatched messages as well as their sum which we define it the overall throughput. The overall message throughput reaches its maximum rate at 1200 msgs/s for 3 or more publishers. Hence, the number of publishers has very little influences on the JMS server throughput. As a consequence, we use in the following experiments at least 3 or more publishers.

In this particular experiment series, the JMS server could only utilized to 90% due to the limitation of a single subscriber. To assess the impact of the persistent mode, we conduct the same experiment in the non-persistent mode and the results are the overall throughput is about 2000 msgs/s for the non-persistent mode in contrast to about 1200 msgs/s for the persistent mode. Due to the non-persistent mode, the dispatched message rate is lower than the received message rate for a large number of publishers which leads to about 10% message loss in the end.

TABLE I
DATA OF FIG.5

Throughput(msgs/s)	Number of Publishers
1200	0
1100	5
1050	10
1000	15
1000	20

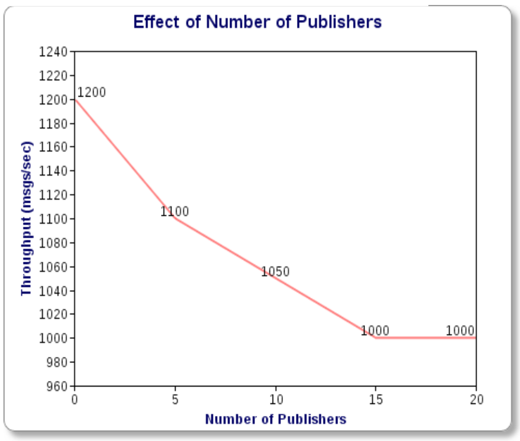

Fig. 5 Effect of the Number of Subscribers

7.2.2 Effect of the Number of Subscriber

Similarly to the above, we investigate the impact of the number of subscribers on the JMS server throughput. To that end, we have 3 publishers' threads running on one machine and vary the number of subscribers on two other machines. The overall throughput of the JMS server starts at 1200 msgs/s for 1 subscriber, it increases with an increasing number of subscribers to a value of about 3200 msgs/s for 20 subscribers, and it decreases then to 1300 msgs/s for many subscribers. Thus, the previous experiments led to a untypically low capacity due to the single subscriber. The received message rate decreases significantly with an increasing number of subscriber's n. This can be explained as follows. No filters are applied and all messages are delivered to any subscriber. Thus, each message is replicated n times. This requires more CPU cycles for dispatching messages and increases the overall processing time of a single message. As a consequence, the received message rate is reduced because the overall throughput capacity of the server stays constant. The throughput of a JMS server can be measured in messages per second (message throughput). The message body size has certainly an impact on these values. We test the maximum throughput depending on the message size.

We set up 5 publishers on one publisher machine and 5 subscribers on a single subscriber machine; a single subscriber is not able to fully utilize the server CPU. The throughput in msgs/s is measured but the throughput in Mbit/s is derived from these data.

TABLE II
DATA OF FIG.6

Throughput(msgs/s)	Number of Subscribers
1200	0
1400	5
2000	10
2500	15
3200	20
2800	25
2300	30

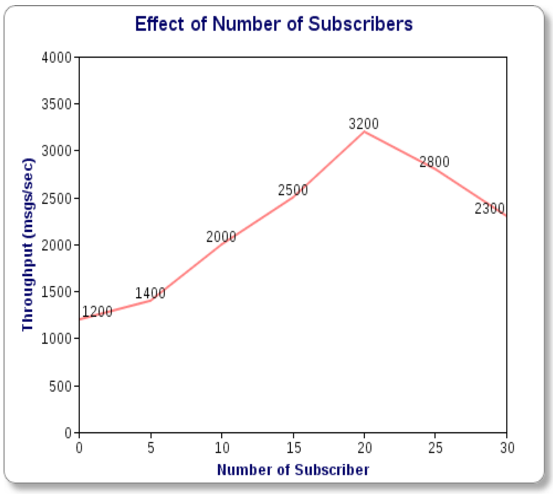

Fig 6. Effect of the Message Size

The calculation of the corresponding overall message size takes into account various message headers, i.e., 40 bytes JMS header, 32 bytes TCP header, 20 bytes IP header, and 38 bytes Ethernet header, as well as TCP fragmentation. an increasing message body size decreases the message throughput and increases the data throughput significantly. For small message bodies, the message throughput is limited by 3000 msgs/s while for very large message sizes, the data throughput increases significantly up to 300 Mbit/s. Obviously, the network interface of the JMS server becomes the system bottleneck.

8. FEATURES OF QUICKMQ

QuickMQ provider for JMS provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages. In comparison with traditional MOM it provides addition with the following features

1) Asynchronous Messaging: Sender doesn't require waiting till acknowledgement of previously sent message is received. It results in increased speed [1].

2) Reliability: Message is delivered at once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages[1].

3) Loosely coupled: A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate.

4) Message selectors: To get only desired messages out of all, consumer can take help of message selector, where SQL92 conditional expression syntax is used with message properties and headers as criteria of selection. Expression syntax contains Identifiers, Literals, Comparison operator and Arithmetic operator.

9. QUALITY OF SERVICE

1) Transacted Sessions

a) It maintains atomicity, i.e. the messages sent to destination (queue) during lifespan of Queue Session will not be delivered to receiver until commit is invoked on Queue Session.
b) This feature is achieved by making 1st parameter 'True' in createQueueSession function.
c) createQueueSession (true,session.AUTO_ACK)

2) Persistent/Non-Persistent Delivery

a) Persistent messages are survived on failure of JMS provider and later delivered till that they are stored on database. While non-persistent messages do not.
b) setDeliveryMode (DeliveryMode.PERSISTENT)

3) Priorities

a) Message producer gives priority to message; they are used by server for ordering of messages. Messages with higher priority are ahead of messages with lower priority.
b) Function Used: publish (msg, ….., PRIORITY, ……….)
c) Default Priority: 4
d) Normal Priority: 0-4
e) Speed Up Priority: 5-9

4) Time to Live:-

a) Messages have expiration time, its mainly for those messages who are relevant for fixed amount of time. We can use any of following two methods.
b) setTimeLive (VALUE in miliSeconds);
c) Or we can set these milliseconds as 5th argument of following function.
d) Function Used: publish(…., ……, ……, ….., Values in milisecond)

10. CONCLUSION AND FUTURE SCOPE

We build application of having multiple functionalities in different languages and we try to make these functionalities run asynchronously on different language processors without waiting for other, except for those functionalities having dependencies between them.

Now a day's all applications are complex and support different functionalities which can be implemented effectively in their respective languages depending on functional requirements [5]. These different functionalities are now a days can be executed by using java but these facilities are rather synchronous ,only after performing single functionality the other can be executed which may take longer for complete execution, these mainly avoided in QuickMQ and different functionalities are performed asynchronously which rather reduces clock time of complete execution

We have implemented light version of JMS provider but still we able to maintain the original performance nearly as that was for heavy weight JMS provider. In this work, we have tested QuickMQ for the performance and we tested it in various conditions.

Here are some of our findings that we found during our analysis.

1) At least 3 subscribers and 3 publishers sending in a saturated mode are required to fully utilize server CPU and to make server maximum message throughput.

2)In non-persistent mode, QuickMQ server achieves a significant larger throughput may lose messages.

3) The message size has impact on throughput of server.

4) The number of Topics and Queues has little impact on server capacity.

11. REFERENCES

[1] Java Message Service API Rev. 1.1, Sun Microsystems, Inc., April 2002, http://java.sun.com/products/jms/.

[2] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," in ACM Computing Surveys, 2003.

[3] Krissoft Solutions, "JMS Performance Comparison," Tech. Rep., 2004

[4] Java Message Service O'Reilly Publications – Mark Richards, Richard Monson- Haefel & David A. Chappell

[5] Professional JMS Programming – Wrox Programming Scott Grant, Michael P. Kovacs, Meeraj Kunnumpurath, Silvano Maffeis , K. Scott Morrison, Gopalan Suresh Raj, Paul Giotta ,James McGovern

[6] Integrating heterogeneous information services using JNDI

[7] Dirk Corissen, Piotr WVendykier, Dawid Kurzyniec, and Vaidy Sunderam Emory University Dept. of Math and Computer Science Atlanta, GA 30322 USA

[8] Hsin-Ta Chiao, Chun-Han Lin, Kai-Chih Liang, and Shyan-Ming Yuan, Department of Computer and Information Science National Chiao Tung University 1001 Ta Hsueh Rd., Hsinchu 300, Taiwan {gis84532, gis89505, kcliang, smyuan}@cis.nctu.edu.tw

[9] A. Heydon and M. Najork, "Performance Limitation of the Java Core Libraries," Proc. of the ACM 1999 Conf. on Java Grande, pp. 35 – 41, 1999

[10] Sun Microsystems, Java Remote Method Invocation-Distributed Computing for JAVA, http://java.sun.com/marketing/collateral/javarmi.html,Sun Microsyetems

[11] The OpenJMS Project .http://openjms.exolab.org/

[12] Sun Microsystems. The JNDI tutorial. http://java.sun. com/products/jndi/tutorial/.

[13] TIBCO Enterprise Message Service, Tibco Software, Inc., 2004,http://www.tibco.com/resources/software/enterprise _backbone/messageservice.pdf.

[14] IBM WebSphere MQ 6.0, IBM Corporation, 2005

[15] Enterprise-Grade Messaging, Sonic Software, Inc., 2004, http://www.sonicsoftware.com/products/docs/sonicmq.pdf

[16] S. Bittner and A. Hinze, "A Detailed Investigation of Memory Requirements for Publish/Subscribe Filtering Algorithms," in International Conference on Cooperative Information Systems (CoopIS), Agia Napa, Cyprus, Oct. 2005, pp. 148–165.