Softwired

White Paper

Every Dad needs a Mom

Message-oriented Middleware

by Martin Erzberger and Marcel Altherr

© Softwired AG 1999

Content:

1. Introduction 3

2. MOM – beyond the slash/ 3

2.1.Just like the postal system 4

2.2. MOM advantages 4

2.3. Stock exchange: the right information at the right time 6

2.4. Fault-tolerant server 7

3. Messaging is not messaging 8

3.1. Publish/Subscribe 8

3.2. Message Queuing 9

4. JMS – A Messaging Standard at last! 10

5. On Board of J2EE 10

6. MOM – every Dad's future 11

1. Introduction

Every Dad (Distributed Application Developer) needs a Mom (Message-oriented Middleware). Indeed without messaging middleware, no company can get the best out of enterprise computing. It makes natural communications architectures possible — because messaging integrates heterogeneous systems without sacrificing flexibility. “MOM“ eliminates frustration in system development, conversion and maintenance. Why? What does message-oriented middleware actually mean? And what is the future of messaging? These questions are worth thinking about. In this article we examine the characteristics of the "Java Message Service" (JMS) programming interface based on our well-proven MOM-Middleware: the unique SoftWired iBus™.

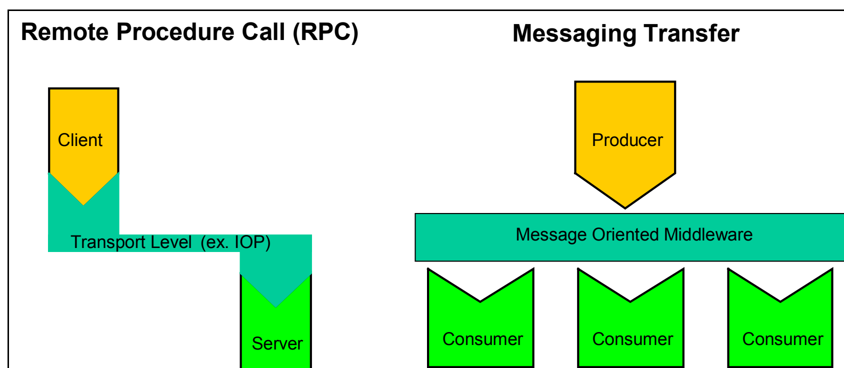
2. MOM – beyond the slash/

The term middleware is vague and causes confusion as it is used for many different technologies. This term originated in the times of client/server systems, and means software which enables communications between the distributed components of an application. Roughly speaking, middleware stands for the slash in ‘client/server‘. However, this would immediately limit it to only those distributed applications: whose structure could be defined as having a client part and a server part. And this would impose an artificial limit on the countless possibilities of using messaging middleware — MOM simply cannot be put in a straitjacket.

For the sake of discussion, two models of middleware can be defined in terms of communications architecture:

- **Synchronous, tightly coupled communication** between distributed components. This is the model of CORBA, RMI, and DCOM. The respective programming model is called Remote Procedure Call (RPC).
- **Asynchronous, loosely coupled communication** between components. This is the MOM model. The respective programming model is called messaging. (see Fig 1)

Fig 1: Tight vs. loose coupling



In other words: message-oriented middleware is a software for transporting messages from a source component (originator) to target components (recipients), usually running on different computers.

2.1.Just like the postal system

Message-oriented middleware works like postal mail: write a letter, stick an address on the envelope, and put the letter into a letterbox. For me as the sender, that's all there is to it. Now I can get on with the next letter. This kind of communication is asynchronous:, there is no need to wait for an answer before continuing with the next task[1]. The sender and recipient of the letter are decoupled, and the Post Office acts as a transporter. This sort of communication still works when the recipient has changed address, because the Post Office knows where to forward the letter.

There are even more similarities between the Post Office and MOM. Both of them are message (or "packet") oriented, and each message is transported as a self-contained unit. As an advantage, the recipient does not have to be present while messages are being transported. This is particularly important in networks where long-lasting connections are expensive and often difficult to maintain, such as mobile telephone networks. Sessions can still be held between the originator and the recipient: a fixed component of each message can be a session identifier. This enables the recipient to identify various messages as belonging to the same session.

At first glance, communications via MOM seem to be more complicated than a direct client/server link. But closer analysis shows clear advantages, particularly when the same message has to be sent to more than one recipient. E.g. a vehicle guidance system wants to send a traffic jam message to 5000 cars. As transporter, a MOM can exploit the fact that the message is identical for every of the 5000 recipients and use a more efficient transport protocol than TCP/IP, e.g. IP Multicast or the Radio Data System (RDS). This means that the message has to be sent only once, irrespective of the number of recipients. The scalability of MOM is tremendous.

2.2. MOM advantages

But even with only two communication partners, message-oriented middleware has many significant advantages:

- **Time independence of components**: The message sender and recipient do not have to be online at the same time, since MOM queues messages when their recipients are not available.
- **Location independence of components**: The message sender and recipient can be migrated from computer to computer at run-time, since senders and receivers are decoupled using message queues or communication "topics". This enables 7x24-hour operation, because software and hardware can be serviced without bringing the system down.

---

[1] A very good example of synchronous vs. asynchronous communication is given in Isaac Asimov's short story "My Son, the Physicist", to be found e.g. in Asimov, Isaac; The Complete Stories, Volume 2; HarperCollins Publishers, London 1995

- **Latency hiding**: In client/server systems, a client is typically suspended while its request is transmitted to the server, processed by the server, and a reply sent back to the client. This makes for hard to use graphical user interfaces (GUIs), as the GUI will "freeze" during a long-running transaction, such as a credit card check. Making a GUI more responsive despite this problem places heavy demands on the developer: He or she has to work with threads, and has to be able to interrupt pending RPCs at any time. With MOM, however, the request and reply phases are fully decoupled. Once a request message is transmitted, the GUI can immediately proceed with the next task, such as updating a scroll list or processing further user interactions. This makes for more user friendly client applications.

- **Scalability**: publish/subscribe MOM systems (see below) scale very well in respect to the number of receivers of a particular message.

- **Event-driven systems**: Since events are simply a special form of message, MOMs explicitly support the development of event-oriented distributed applications:, ranging from user interfaces to server components.

- **Simplicity:** MOMs are based on a simple idea, developers can quickly grasp the basic principles and become productive almost immediately.

- **Optimal quality of service**: With RPC there is only one quality of service: 100% reliability. So what more could one wish for? The answer is that less is often better: for example, with the traffic jam information system mentioned above, it is perfectly adequate if 98% of the vehicles get the traffic jam message. This allows a low-cost protocol such as RDS to be used for reaching all vehicles in a certain geographical area— except 2% of them might be in tunnels. To be sure of reaching 100% of the vehicles, a MOM would have to send 5000 SMS messages. This would not only be time-consuming, but far too expensive for the extra benefit. In other words: a **choice of quality of service** (QoS) is highly desirable. This is precisely what SoftWired's iBus™ product provides. To summarize: MOMs are practical and easy to handle, and their characteristics make communications more realistic. Next we present two typical examples of MOM applications that have been realized with iBus:

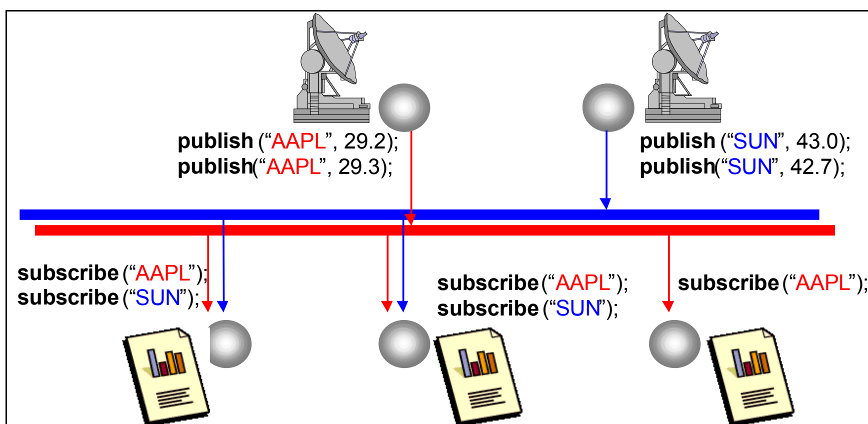
2.3. Stock exchange: the right information at the right time

A typical stock exchange information system has to handle vast quantities of financial data and needs tens of thousands of instruments. Every second, several thousand updates are generated on the stock exchanges worldwide (e.g. buy/sell transactions). It is very difficult for the stock exchange information system to pass on this immense amount of real-time data efficiently to many subscribers, notably traders. And the task is not made any simpler by the fact that the several hundred traders, all working at the same location, demand equal treatment: the last one is not ready to wait two seconds longer for information than the first one.

It is precisely for this reason that SoftWired developed the iBus™ publish/subscribe middleware. iBus makes use of the fact that certain financial instruments (such as bluechips stocks) are of interest for most traders. Other financial instruments are hardly ever looked at. That is why information can be grouped according to *topics*.

This is how the stock exchange information system works: incoming data is first analyzed and categorized. Then it is *published* into the MOM and labeled with the appropriate *topic* (e.g. "/currencies/Europe/DEM"). The clients or recipients *subscribe* to the topics they are interested on. As soon as something is published on a topic in question, the middleware automatically presents this information to the client: no polling is required, no server gets overloaded due to the large number of traders interested in the same information. This enables optimal network utilization: with IP Multicast as transport protocol, the message only has to be transmitted only even though there are several hundred subscribers.

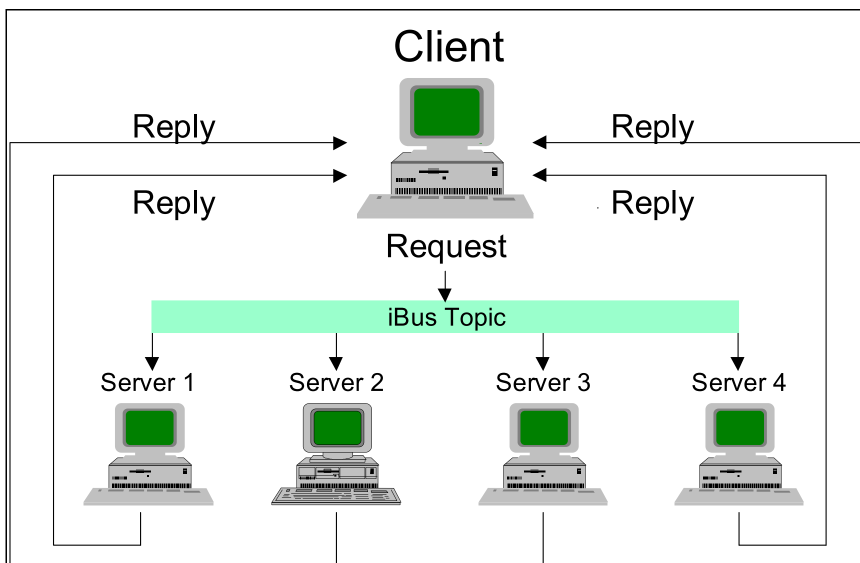
Fig. 2: Stock exchange information system



2.4. Fault-tolerant server

Client/server computing is traditionally implemented using RPC (CORBA, RMI, or DCOM). But why? It is only necessary to send a request and receive a reply. It is irrelevant which server this reply comes from. So the alternate solution is to send a message through a MOM, instead of issuing an RPC on a particular server. The message topic defines the service required (e.g. "/services/time/timeofday"). Registered with the MOM are several servers, all subscribed to the same topic. The request for time of day is therefore transmitted automatically to each of the servers. The servers then reply to the MOM, which delivers the reply to the client. This is a simple way of ensuring a fault-tolerant server system: the system works as long as at least one server is active (See Fig 3)

Fig. 3: Multicast request / reply



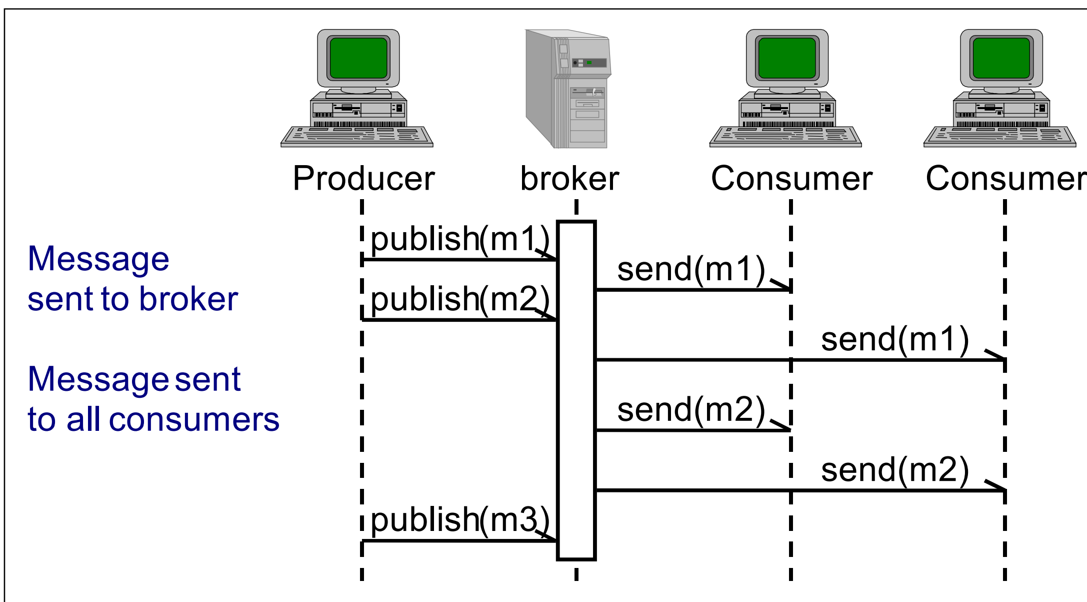
3. Messaging is not messaging

There are differences in messaging middleware products. Some of these differences are specific to implementation (for example, whether a product is written in Java or not), while others are specific to architectural (for example, whether a product is fully distributed or relies on a network-centric hub). Messaging middleware can therefore be subdivided. For practical purposes, there are three types of messaging middleware: message passing, publish/subscribe, and message queuing. In this article we shall focus on the latter two models, as those are the two models addressed by JMS.

3.1. Publish/Subscribe

The basic mechanism of publish/subscribe was illustrated above in the stock exchange information example: The central concept of this middleware type is a topic. The originators produce information and publish it under a certain topic, such as "/currencies/Europe/DEM", and the recipients subscribe to this topic. The middleware ensures that all subscribers of "/currencies/Europe/DEM“ receive all information from all publishers on this topic. This type of communication can be illustrated by a round table discussion. Everyone sitting at the table can be viewed as a "subscriber" for the ongoing conversation. As soon someone talks, he acts as a "publisher" of messages. The "middleware", i.e. the air surrounding the table, ensures that all subscribers receive all message. Publish/subscribe (See Fig 4) is much more efficient than RPC whenever "m:m" or "1:m" communication takes place.

Fig. 4: Publish / subscribe messaging middleware

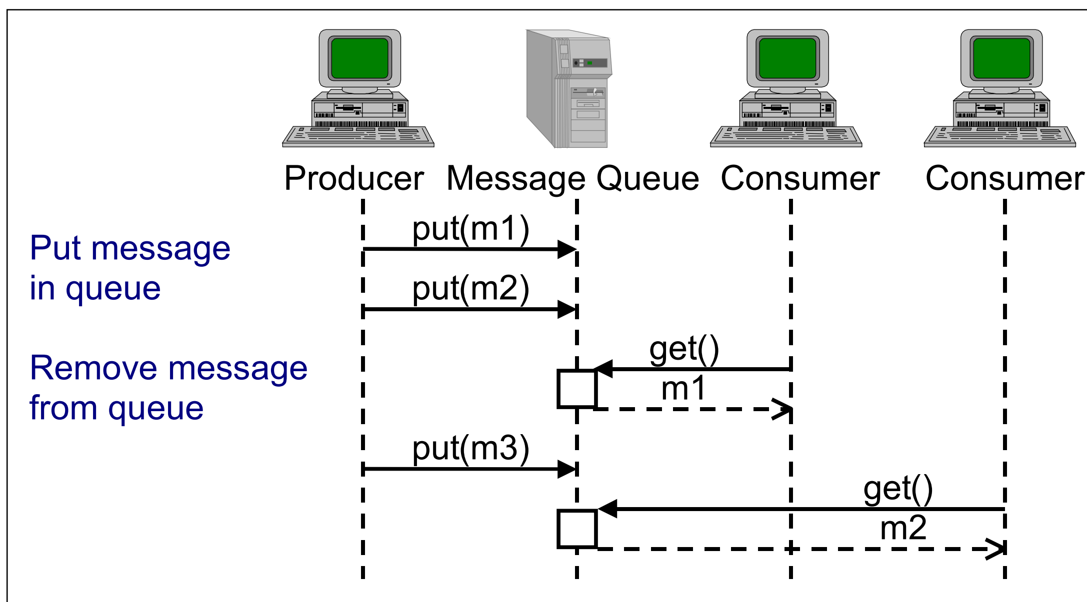


3.2. Message Queuing

A message queuing system works basically like a virtual mail box system. The originators put their messages into one (or several) mail boxes, the recipients remove messages from mail boxes whenever they feel that time is right for processing their messages.

Message queuing (See Fig 5) is ideal for messages, which are to be "consumed". While financial information is sent to all dealers (i.e. not consumed by the first recipient alone), a dealer's share purchase order only has to be processed once. It is therefore placed in a queue, which is processed by several servers specialized on taking such orders. As soon as the first available server has taken the purchase order from the queue, it has been "consumed" and is no longer available to any other server.

Fig. 5: Message queuing middleware



4. JMS – A Messaging Standard at last!

Why do we hear so little about messaging, but so much about CORBA and RMI? Primarily, because CORBA is backed by by the OMG, a coalition of manufacturers who have agreed on a common standard. The same applies to RMI, which is backed by Sun and by the Java community. Until recently, messaging middleware products only had one thing in common: MOM manufacturers pushed their proprietary programming interfaces.

Things have changed! At the end of 1998, the Java Message Service API (JMS) was launched by Sun Microsystems. From the beginning, JMS was hailed by all leading messaging middleware manufacturers. Apart from these established firms, all new manufacturers backed JMS, including SoftWired with its iBus™ product family. Why? For the simple reason that the JMS standard supports publish/subscribe as well as message queuing, the two most important MOM types. This standard has been kept very slender, with an easily to learn API.

5. On Board of J2EE

At the 1999 JavaOne Conference, Sun launched their flagship platform, the "Java 2 Enterprise Edition" (J2EE). J2EE comprises mainly of a series of coordinated APIs primarily for use in enterprise applications. These include specifications such as "Enterprise Java Beans" (EJB), Java Transaction API (JTA), and JMS of course. Messaging is thus put on the same level as RMI. This requires further modifications to the EJB specification, since the current version (1.1) only admits RMI as communication mechanism. Full JMS integration will be included in EJB version 2.0.

Even now, JMS can be used on EJB 1.1 compatible applications servers. A typical enterprise application will consist of EJBs, which can be invoked by clients through RMI, and which can deliver messages to recipients via JMS. Developers thus get the best of both worlds: the supporting services of an EJB container, and the publish/subscribe or message queuing functions of the JMS specification. Therefore, sending a JMS message from a EJB is not a problem today. However, EJB 2.0 will standardize the mechanisms needed by a container to dispatch JMS messages to EJBs, such that EJBs can also act as receivers of messages.

6. MOM – every Dad's future

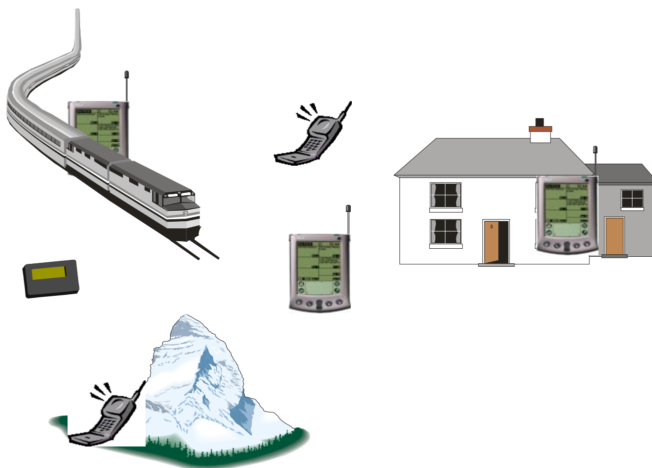
Messaging has always been important in the enterprise environment, even more important than RPC. In other segments, however, messaging was rarely adopted: this technology was just not available to developers, since expensive products with proprietary interfaces had to be bought. RPC, on the other hand, is always present: RMI is part of JDK and therefore free of charge, and since Java 2, CORBA is also available free of charge in the form of JavaIDL.

As a result, many developers today are not familiar with messaging. They design applications around RPC, although MOM technology enables much more elegant, slender and scalable designs, in many cases. In our opinion there are two main reasons why MOM will be the trend in the future:

1. The JMS standard allows developers to choose among an increasing number of JMS middleware products, according to price, scalability, and features. By sticking to this open standard, risk is minimized.

2. Mobile devices are conquering the world. Those devices are not continuously connected to a network. MOM are well suited for coping with communication outages and long transmission delays.

Fig. 6: Mobile intelligent units



SoftWired's iBus™ is designed with mobile intelligent devices in mind (See Fig 6). Thanks to its 100% Java implementation, small footprint and protocol independence, iBus™ is ideal for use on the PDAs of tomorrow. However, there is no need to wait until tomorrow. Today, iBus applications are developed for in-house use and for Web customers, but tomorrow they will run on SoftWired's iBus™ powered PDAs.

In our next issue we shall explain how YOU can put your programming ideas into practice. Based on SoftWired's unique iBus architecture, you will be introduced to the JMS programming interface. In less than one hour you will be running your first messaging application.

Marcel Altherr and Martin Erzberger are executive managers of SoftWired AG.
You can reach them at: info@JavaMessaging.com. **iBus** can be downloaded for evaluation free of charge from the SoftWired homepage: www.JavaMessaging.com

Contact Information

Softwired Inc.
www.softwired-inc.com, info@softwired-inc.com

Phone +41 1 445 23 70, fax +41 1 445 23 72

Technoparkstrasse 1, 8005 Zurich, SWITZERLAND

iBus™ and Softwired are trademarks of Soft-Wired.
iBus™ technology is Patent Pending.

Softwired
Next stop: ubiquity