



Whitepaper

Future, Past & Present of a Message

Whitepaper

Future, Past and Present of a Message

by Patrick De Wildeⁱ

What is wrong with the above title?

No, I do not mean to write about “**The Message**” in any religious or ideological or even email sense. You probably came across this paper in an IT-context, and presume this is about asynchronous computer application communication techniques. You are right. “**Future, Past and Present of Messaging**” could also be the title of a paper. In it you’d read something like “Asynchronous messaging in computer systems was first used in ... it became a popular means of integrating existing applications with the introduction of ...” and so on.

Interesting as this would be, I suffice here with this summary: Messaging based middleware has come of age and is now used as the basis for both integration of legacy applications and for development of new applications using several design models. It has a great future.

This is neither about “**The Message**” nor about “**Messaging**”. The focus is on a simple message somewhere in your computer network, perhaps one of millions. It could be insignificant, or it could mean a lot to your business. How do you know? It might not exist anymore, but what happened to it. Perhaps it doesn’t exist yet. If it does exist, where will it go?

Significantly, “future” is the first word. After all, it’s the future of your messages you are concerned about, isn’t it? Service Level Agreements, the Mission Statement of your group, perhaps even your job description. Read them again, and you’ll see much of it can be translated into assuring those messages get their payload delivered correctly, on time and as cost-effective as possible. But in order to be able to make intelligent statements about the messages’ future, you should know the history of past messages. Remarkably, most administrators will be more confident in predicting the future, than in giving exact information about the past. Hereunder we will explain why.

Nothing is wrong with the above title.

Managing WebSphere® MQⁱⁱ

The scope of what follows is limited to the most popular of messaging middleware: WebSphere MQ ((formerly MQSeries and commonly known as MQ), although the principles are applicable to other products as well. Past, Present and Future (in

whatever order) of your message depends on both functionality and performance. How do you get your arms around this?

In my experience with the management of WebSphere MQ, I haven't seen a site yet where administrators are not struggling to obtain information on that particular message. And by extrapolation, also struggling to get statistical information about the application's behaviour and performance. Managers often expect the techies to produce such data quickly upon request. Why is this so difficult? I believe the strength of MQ is its asynchronous aspect together with multi-operating system support, but that also results in the biggest administration challenges. Different interfaces and OS differences are two such challenges. A more important one is the problem of correlating events that occur on different machines.

Yes, MQ is robust, hardly any bugs at all. The application is well designed and tested. On paper everything should work perfectly. We all know the occasional gremlin forces us to dig into the byte-level of our state-of-the-art certified publish/subscribe based services-orientated enterprise application integration framework developed by an n-th generation system with 3D GUI and twenty-seven abstraction layers. But even in the gremlinless world, you are faced with questions on the past, present, and future of a message. IBM's often quoted principle of assured delivery is almost legendary, but still dependent on a number of implementation specifics. Can you guarantee that these are met?

In its most general form, the problem is the following:

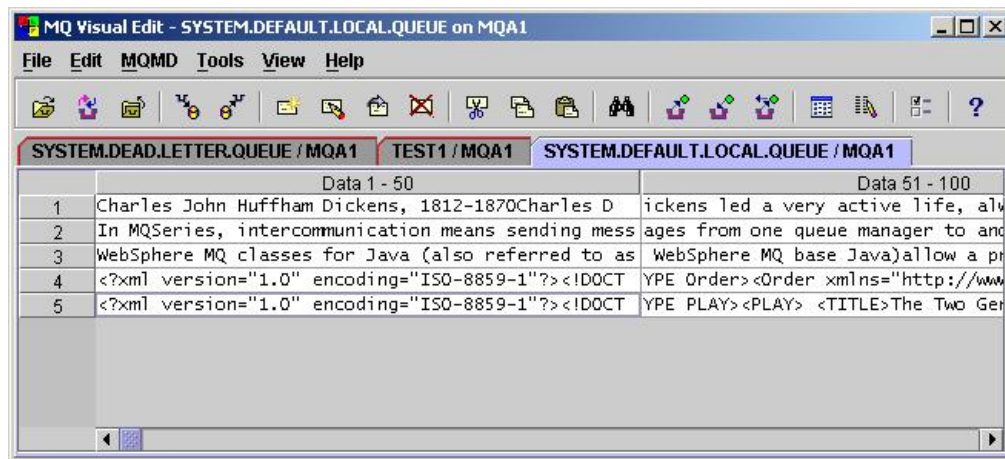
Given a number of predicates that identify zero, one or more messages that exist, existed, or will exist in your network; can you find their current whereabouts, their contents, their origin, their destination and the route and timings between the messages at key intermediate points? Can you identify related messages, like the responses for a request?

How does one go about answering these questions?

No administrator can be happy without the skill and tools to answer questions related to the past, present and future of messages. Be it individual qualified messages, or a large group of messages for which statistical information is required.

Present

Perhaps easiest to find is a message that is still 'alive'. It exists on a queue, and is available for an application to be read. Given you know the queue and the queue manager, there are numerous tools to read the message contents and its header.



A tool like MQ Visual Edit (see www.cressida.info/products_capitalware_VE.htm) will make an effort to format these to your liking. But finding the queue that the messages currently reside on is not always a trivial task. All too often in practice, the administrator is forced to look in all possible queues he can think of: several application queues, transmission queues and dead letter queues. In the worst case he or she will be forced to use different interfaces for the different operating systems involved. And what if the message is not found? Has it already been processed, or did it never exist?

What is it you want to know about the live messages? What is it you want to do with them? Here are some thoughts.

- Display data in the format I want
 - fixed columns
 - XML
 - JMS-type
 - hexadecimal.
- Display system information.
 - the “normal” message header, the MQMD
 - dead letter queue header
 - transmission queue header
 - event message
- Edit a message
- Delete a message
- Reroute a message, possibly to another queue manager
- Export and import messages.
- Search for a message with certain data or header content.
 - within a queue
 - across multiple queues
 - across multiple queue managers
- Have batch facilities to do some of this from within a script.
- Know where it will go (see below: future of a message).
- Know where it came from (see below: past of a message).

Future

Given an existing message, what will happen to it? It's a classic question at a certification exam: With these definitions of queue managers, queues (local, alias, remote, transmission), and etcetera... where will a message put on queue ABC end up? Such a question is good to test a student's knowledge of middleware, but hardly realistic. Of course you need a system like Candle®/IBM®'s "PathWAI™ XE for WebSphere MQ" to look at all current object definitions and to manipulate them. But that is not enough. In the real world the question "What will happen when I put this message on queue ABC?" can often not be answered by looking at some queue properties. Applications are involved whose behaviour is dependent upon the contents of the message. A request/reply model with an applicative dead letter queue is a simple example of this. We can consider more complex workflow models, the usage of message brokers, or publish/subscribe mechanisms. Perhaps some sort of workload balancing is involved, in which case the route might be stochastic.

Why do we care about what happens to a message in the future?

Evidently this mental exercise has already been done during the design of the application. In this stage the looking-at-definitions approach combined with some unit testing is normally sufficient. But what will happen when that application is brought into production where it has to process a million messages a day and it interferes with dozens of other applications? Have we considered all possible effects to functionality? And what about performance? The truth is that except for the simplest of applications we cannot be sure.

But by doing realistic simulation runs, we can get very close to certainty. A simulation in which you pump 10 million times the same test message in the network is not realistic. You need a real-life sample of your production workload, coming in at the original rate. You are probably unable to do this on your production systems. So it does require a hardware and software simulation environment mimicking your production systems. This requires a serious effort to set up and maintain, but the investment is worth it. You'll avoid application bugs and outages in production. You'll be able to tune the different levels of your system (application, middleware, operating system, network, hardware), thus avoiding costly upgrades.

Past (only a dead I/O is a good I/O)

So now you can find your message, and you know what's going to happen to it. Everyone involved in the management of a messaging based application will tell you that more is needed. Sooner or later you'll want detailed information about the history of a particular message. This could be for a variety of reasons:

- problem diagnosis and rectification

© Cressida Technology Ltd., all rights reserved.

For Cressida worldwide sales and support contact information, please visit

www.cressida.info or email us at info@cressida.info or support@cressida.info

- auditing
- accounting
- regulatory compliance

Some realistic examples of these can be found on www.cressida.info/requestcase.pdf. Most, if not all of the information needed are logged by the middleware for recovery purposes using the high-performance ARIES set of algorithms¹. Unfortunately the tools that come with WebSphere MQ to convert this recovery log in human readable information are not practical enough to be used for this purpose. This is why most sites log the data from within the application or from an API wrapper. This often results in a tremendous overhead. As a technology aficionado I find this very unesthetic, because the data was already logged by the queue manager.

Only a dead I/O is a good I/O

After all, messaging middleware is not different from the operating system or other subsystems like database managers. In modern computing environments with a tendency for straight-through processing its I/O operations is the bottleneck. Every systems administrator knows this. In the assembler development days, programmers knew it as well. But somewhere during the introduction of the n-th abstraction layer in some next-generation programming language they lost sight of this. Don't get me wrong: I am not advocating against modern development techniques.

The point is that the concern for performance and resource usage has shifted from programmers to administrators. Instead of having your applications or API-wrapper do these superfluous I/O's, you could use Cressida's ReQuest™ software to analyse the WebSphere MQ recovery log. Make good use of the work that the queue manager has already done. Since the analysis can be done on another machine and at any time, there is zero overhead for the applications. Here is an example where someone would search for the occurrence of some suspect messages in the past using ReQuest. First a message filter is created:

¹ "ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging" ACM Transactions on Database Systems, Volume 17, Issue 1 (March 1992).

Filter Definition ✖

Filter Name
suspect_orders

Define New Constraint

MessageType == Hex Oct Char Unicode

Resulting Constraint to Add to Expression
MessageType == <1>

Current Expression

Comparison Strategy for Message Data

Use CCSID No translation Use Codeset

Comparison Strategy for MQMD Fields

Use CCSID No translation Use Codeset

Filter on Embedded MQMD

Then this filter is used to produce a report giving all occurrences of such messages, and possibly related messages:

Run Report ✖

QueueSet Name
 ORDERS ▼

Report Type
 MessagePropagation ▼

Filter Name
 suspect_orders ▼ No filtering

Report Criteria
 Time based
 LSN/RBA based

Platform
 Distributed
 Z/OS

Time From
 15 : 59 : 07 . 607 17 / 01 / 2007 Any

Time To
 15 : 59 : 07 . 607 17 / 01 / 2007 Any

Records to get
 All
 Only Puts

Output
 Database
 File

Report Name:

C:\Program Files\Cressida\ReQuestMQ\reports\suspectOrder.xml

Run Report ✖

Max User Data (bytes)
 Any

Max RFH Data (bytes)
 Any

Max MQ Operations
 All

Report Timeout (seconds)
 None

Report Stylesheet

Display Embedded MQMD

Filter for response
 No Response Filter

Character set of Message Data
 Use CCSID Use Codeset

Response Correlation
 No correlation Message Id Correlation Id

Report Correlation
 No correlation Message Id Correlation Id

A Message Summary

Successful management of your messaging based middleware will depend on your choice and usage of a combination of tools and skills that will allow you to answer any question on the past present and future of any message in your network. This information should be available on individual message basis, as well as in the form of statistics of various granularities (queue, application, enterprise...).

The best way to achieve this:

- * Past of a Message: by reading the MQ recovery log.
- * Present of a Message: by an off-the-shelf MQ customisable application.
- * Future of a Message: by a combination of an MQ object definitions management and workload simulation software.



ⁱ Patrick De Wilde is a 20 year IT veteran with extensive knowledge and experience with Messaging Technologies, WebSphere MQ, IMS and DB2 databases and numerous other related systems and subsystems.

ⁱⁱ WebSphere® MQ and IBM® are registered trademarks of IBM Corporation; Candle® is a registered trademark of Candle Corporation, an IBM company. Cressida ReQuest™ for WebSphere MQ is a trademark of Cressida Technology Ltd. MQ Visual Edit™ is a trademark of Capitalware, Inc.