Enter Microsoft

Interview With Alan Dickman, author of *Designing Applications with MSMQ*.

Topics Covered

Trends In Enterprise Application Integration

The Evolution of Messaging Middleware

The Differences Between MSMQ and MQSeries

Alan Dickman is a Chief Technologist at PSW Technologies with 20 years of computing experience. Generally, companies use his expertise to help design, implement, test, and then deploy mission-critical, distributed systems. Currently, he is leading an effort to design and develop a Business-to-Business Extranet sales application.

Alan, why is Enterprise Application Integration (EAI) on the business agenda for many corporations today?

For one thing, EAI is key to most companies because they can?t afford to start over every 2-5 years. It means disrupting users, re-training developers and operations personnel, and incurring huge risks and costs. Furthermore, it?s really hard to integrate several systems together. At a recent conference on E-commerce, company after company testified that these seemingly-simple integration efforts would cost at least twice as much and take two times longer than expected.

Given that integration projects take twice as long as expected and cost two or more times what most companies anticipate, choosing the right software solutions is fairly important. It?s still a shock to me that lots of companies continue to roll their own middleware. Many experts agree that roll-your-own software adds 30-35% in development and maintenance costs to the average integration project. To me, this seems penny-wise and pound-foolish.

What are the important EAI technologies

The way in which you tie applications together is critical. I don?t mean to take anything away from CORBA, Java Remote Method Invocation (RMI), Remote Procedure Call (RPC) and similar solutions, but none of these offer the flexibility or robust communication features of messaging. Among other things, each application usually is enhanced and upgraded on its own schedule - a messaging architecture that uses agents to interface with existing applications offers better isolation between disparate systems that need to share data or processing.

Web technologies for the Extranet and Intranet are clearly of critical importance to EAI. According to one estimate, the E-commerce market is going to grow to more than $30 Billion per year by 2002. Most of that business is going to use Web technologies.

Distributed object technologies are becoming critically important to the initial and continued success of integration projects. Increasingly, it is the glue that ties new applications together within applications and across applications. I believe Microsoft?s Component Object Model is the best object technology for integration. Microsoft has a much better interoperability and scalability story than most people realize.

What other trends do you see in the marketplace?

Total Cost of Ownership (TCO) is a big trend. Over the next several years, I think it will change the face of corporate computing.

Microsoft?s approach to TCO is to offer tools that take advantage of transaction processing, messaging, and Web services in the operating system. Integration is really hard and costly. Microsoft tools and services are very well integrated, and this takes some of the cost and complexity out of integration efforts. IBM has a great integrated services story too.

Over time, it will become more difficult for companies to charge so much, up to $35,000 per CPU, for basic middleware. The growth of Linux among Unix users is another TCO trend to watch closely. I think it appeals to companies as a lower cost Unix alternative when integrated services aren?t necessary.

We will see a big growth in Directory-enablement in the next couple of years. This is a great thing for customers since it will extend the best elements of distributed and centralized management to networks, applications, and systems management. Already, LDAP is having an important impact on Extranet computing. Directory-enablement should have a very positive impact on lowering administration costs.

In the middleware space, I?m hoping that our industry will come to embrace publish/subscribe (pub/sub) services as a preferred method of integration over the next 5 years. Tight coupling of application components is a truly costly way to develop enterprise applications that need to evolve over a decade or longer. Messaging probably addresses most of the needs of loosely coupled applications, but it has limitations. Publish/subscribe is more elegant from a design perspective.

How do you see the messaging middleware market evolving?

In the last year, a lot of Windows developers began hearing about the virtues of messaging and learned something about MQSeries. And, many Unix-messaging developers are taking a hard look at MSMQ. MSMQ and MQSeries probably lived in a state of "coopetition". I don?t expect this to last.

Microsoft is really pushing NT as a server platform for enterprise applications as well as File and Print services. Their decision to bundle of transaction processing, messaging, and Web services with NT is no accident. Microsoft?s purchase of the FalconMQ Bridge is an example of how important messaging and interoperability is to their strategy. With the release by Level 8 Systems of MSMQ Independent Clients for Unix and MVS in 1Q99, MSMQ becomes a viable enterprise messaging solution.

IBM is reacting to this threat to MQSeries, their flagship messaging and transaction processing technology. It would be hard to make the MQSeries programming interfaces much simpler to use. Rather, IBM is focused on making MQSeries easier to configure and administer. This will benefit user of NT, Unix, and MVS - a very good thing. IBM is also making it easier to use MQSeries in a COM programming environment.

In the end, all of this is great for messaging customers!

What are the differences between MSMQ and MQSeries?

I?ll talk about two technologies at a very high level. If people are interested in the details, they should read the monthly column that Jay Lang and I write for MessageQ.Com.

MQSeries is a proven commodity. No one questions whether it works or whether it can support enterprise application requirements. A lot of users are having great success with MSMQ, but it hasn?t been subjected to the same amount of testing and stress as MQSeries. That said, MSMQ performance is startling - on a standard desktop system, I am able to send 20,000 Express or non-persistent messages per second and 1500 Recoverable or persistent messages per second. And, MSMQ reliability is quite good. Over time, its reputation will grow.

Through Microsoft?s partnership with Level 8 Systems, MSMQ can interoperate with the most popular computing platforms, and it?s a viable message queuing option for many companies. But, MQSeries is available on many more platforms than MSMQ. MQSeries offers connectivity beyond MVS, Windows, and the more common flavors of Unix. For some companies, this is an important consideration.

MSMQ and MQSeries share a lot of features in common. For this reason, MSMQ satisfies most of the requirements of message queuing users. In fact, MSMQ offers transaction processing features, such as exactly-once semantics, that are not provided by MQSeries. Some of these features will be extremely valuable to users.

A facet of MQSeries that impresses me is that it is extremely configurable. As a result, it is more likely to satisfy or at least support the unique requirements of some customers.

Large and changing MQSeries installations will be harder to administer than large and changing MSMQ installations.

If you were to examine requirements carefully, I think you?d find that both MSMQ and MQSeries are viable options for most companies interested in message queuing. When companies demand proven reliability and interoperability outside the MVS, NT, and Unix mainstream, MQSeries has an advantage. The advantage shifts to MSMQ when ease of administration and software costs are a key considerations.

How should developers approach writing MOM applications?

For people who use message queuing, they should think long and hard about whether each queue is shared by two or several processes, the variety of messages that are sent and received from each queue, and the format or content of each message. These are highly interrelated considerations.

Whether an application will use message queuing or another form of messaging, developers should also consider these issues in the context of their reliability and recovery requirements. Messaging provides far more robust communications than technologies like CORBA - developers should design their application to take advantage of these features.

Why did you write your book, *Designing Applications With MSMQ*?

When I first began using MSMQ, the completeness of the product as a transactional messaging solution came as a shock. It showed me how serious Microsoft is about offering the best, most scalable support for enterprise-class applications. Then, I began testing it and observed its performance and stability. At that point, I decided to write the most useful book people could read to learn how to design messaging applications using MSMQ.

Does the title imply that the book is for MSMQ Developers only?

About one-quarter to one-third or the book is probably useful to any developer of message-based applications. For example, one chapter talks about message content options and queue architecture considerations. Another chapter compares online transaction processing (OLTP) and transactional messaging - what are the strengths and weaknesses of each technology, when to use one versus the other, and how to combine them to satisfy application requirements. On a lesser scale, there are discussions on how to use acknowledgements and dead letter queues, how to think about security in the context of messaging. These elements of the book should be useful to MQSeries developers and to users of other messaging products.

*Alan's book, **Designing Applications With MSMQ**, can be purchased at messageQ Books.*