



Dejan Milojicic
 Hewlett Packard Laboratories
 1501 Page Mill Rd.
 Palo Alto, CA 94304
 dejan@hpl.hp.com

Trend Wars

Middleware's role, today and tomorrow

This second installment of “Trend Wars” is dedicated to middleware. Middleware represents a programming infrastructure positioned typically between the operating system (or bare hardware in certain cases) and the user applications. It serves as a framework for building (typically distributed) applications. Examples of middleware frameworks include OSF DCE, Microsoft DCOM, Java RMI, and OMG CORBA. Middleware frameworks provide common services, such as network communication primitives (RPC and distributed objects, for example), naming and locating services, and security support.

Middleware is meant to abstract away the underlying environment from applications and present a homogeneous interface to application programmers and users. There are various levels for achieving homogeneity.

OMG CORBA supports heterogeneous environments—both hardware and software—with the middleware level itself offering homogeneity. Because Microsoft's DCOM approach is based on the Intel architecture and the MS Windows operating system, it is designed around a homogeneous hardware and operating system base. Sun designed its middleware solution on top of a programming language environment (Java), using it as a single homogeneous base. Each approach has its advantages and challenges.

The controversy around these different approaches makes this topic an excellent choice for our department. In an attempt to predict the outcome of today's “middleware trend wars,” we have invited five distinguished researchers and developers. Mary Kirtland is a program manager with

the COM+ team at Microsoft. Doug Lea is a professor at New York State University at Oswego. Joe Sventek is a lab director at HP Labs, who has been involved with middleware development for over 20 years. As Chief Technical Officer and cofounder of Iona Technologies, Annrai O'Toole has been pivotal in development of Iona's ORB. Ann Wollrath is a senior staff engineer with Sun Microsystems and the principal architect of the Java Remote Method Invocation (RMI) system and one of the original architects of the Jini technology.

We asked each interviewee a common set of questions that tries to unveil the middleware trends in the institutions that make middleware's today and future. The questions are listed in the box on the facing page.

—Dejan Milojicic



Mary Kirtland

Who is using middleware?

I think eventually everybody is going to use middleware. The people I talk to are primarily Web application developers and LOB [line of business] application developers, ranging from tens of clients to 10,000s of clients. Some of the higher-end customers or people doing automation, process control—that sort of thing—have rolled their own middleware, because they couldn't buy something that met their needs at the time they were making design decisions. But that means they have to maintain the custom solution, which might not make good business sense in the long run. So they're looking at using a third-party product, getting out of the middleware business themselves.

So, you can't earn money on middleware; you need it, but that's not where you get money.

I think for application developers that's true—most businesses are not in the business of developing software for software's sake. They're in some other business and they have to get the software written to support the business. And for that, the less extraneous stuff they have to write themselves and support themselves, the better.

Another thing about middleware is that it's rare to find people, at least that I talk to, who have in-depth knowledge of all the different technologies used in a distributed application. They want to use them, but they don't necessarily want to—or should need to—completely understand the theory or how to write the middleware themselves.

How can we make middleware more accessible to users?

I think that we need to make the programming model very, very simple. We need to make it so we have a simple declarative model or rules-based model for the people who just want to use stuff and maybe don't understand it or don't want to think about it too much, but we should provide the power for the people who do need it. And that's a really hard thing to do, because you don't want to force everybody to get down and dirty at the power level, but you don't want to handcuff the people who need that ultimate level of flexibility and control.

So composability is also very important?

Not just composability—building from simple to powerful—but making the power accessible to people who need it. In a lot of cases, if you constrain the range of options, you can do a pretty good job of figuring out what people need, without them having to make that decision. I think that's a real interesting thing—to see how much of that decision-making can be encapsulated either within the middleware or the tools so that this broad range of people that we have doing development doesn't need to understand everything just to write an application. Pretty much every application is going to be distributed.

What do middleware users require?

It's all over the map. In a lot of cases, the people who I talk to are just getting started. They know that they need something that scales, but they might not need the ultimate performance yet; they might not have very high-end transaction requirements.

Another type of customer I talk to is an ISV [independent software vendor] who has a mainframe or Unix-based system that they want to make available on Windows NT. So, they're trying to figure out how to take what they've got and move it into NT—how they can take advantage of MTS [Microsoft Transaction Server], MSMQ [Microsoft Message Queue Server], and other NT services.

What important problems do researchers and developers need to solve?

This is one of the things where I'm probably a bit atypical of many of the people at Microsoft or at other systems companies. I don't really find middleware that interesting in and of itself. I think it's a tool that people need to be able to use—so I just want the stuff to be there so that I can meet all those “ability” requirements: scalability, reliability, availability, etc. I don't want to understand how all that works just to build an application.

Think about your compiler. When's the last time you really thought about how a compiler worked? You don't, you just use it, and every once in a while you might tweak some options. Even fewer times, you might run into a bug or something that gets optimized in a way that doesn't work quite right, so then

Middleware questions

1. Using middleware: Who is using middleware? How can we make middleware more accessible to users?
2. Middleware solutions today and in future: What are the important problems that researchers and developers need to solve? Are there any red herrings (nonproblems)? How does the current situation with middleware systems compare with the early days of distributed systems? What is on the middleware horizon?
3. What is the role of freeware in middleware solutions (compared to programming languages, tools, and operating systems, such as GNU or Linux)?
4. What will be the dominant underlying middleware environment of the future (if any)—a single hardware architecture, a homogeneous environment, or an ultimately heterogeneous environment? What are the R&D goals for these environments?
5. Middleware systems versus system software: Is middleware moving toward integrated suites from a few big players? Are there business opportunities to contribute “best of breed” services?
6. Transfer of research done in academia and research labs to industry: How successful has it been? What is most needed?
7. What is the impact of technology trends to middleware: in hardware (fast networking, connectivity, large storages), in mobile computing (ubiquitous connectivity), in real time (quality of service), in fault tolerance, and in integration of the Web and middleware?

you have to think about it. But 80% or 90% of the time, you don't care. You just use it.

We're really moving toward a world with tons of connected devices, people who use multiple machines—a ubiquitously connected world or semiconnected world, and the day of an application sitting on one little machine, just running there and not sharing data with anybody, is just about over. So we've got to make writing these distributed applications a lot easier. That's the thing that's most important; it doesn't matter how powerful the middleware is or how many great things it can do if no one can figure out how to use it.

What's your take on security?

Clearly, everybody has their own perspective on how accessible they want to make data. If you're in a company, it's very important to you to protect your business assets, and so you need to secure your data. You probably also want to track some information and do some auditing on who's connecting to your system and who's modifying your data, for whatever reasons—just in terms of internal tracking or for legal reasons or who knows what. On the other side, people who don't work for you may want to protect their private, personal information for the same reason. There's got to be some balance there.

What is on the middleware horizon?

I think one of the next big application spaces is integrating smart devices into your typical consumer lifestyle: making television smart, or making it possible for me to control something in my kitchen from work so that when I get home, my food has been safely defrosted. Things like that, that people will really see the benefit of, but are difficult to do today, because we just don't have the smart devices quite yet. Interactive things like e-commerce, home automation, and Internet communities are going to pick up over time.

So writing applications that connect these things together is

going to be pretty tough, especially with the available communications bandwidths. So there are things that we need middleware to do for us so they don't all have to be done by every single device. And hopefully our tools are going to get a lot smarter. The people who are doing development, a lot of them are not really that experienced in software engineering or distributed applications development, and yet that's what they're being asked to do. There are, of course, people who are very experienced at it, but that range is getting broader and broader. We've got to find a way to meet their needs as applications get more and more complex.

What about the dominant underlying middleware environment?

I think that no matter what, no matter how much anyone would wish otherwise, it's always going to be heterogeneous. You're never going to be able to get everybody updated to the same thing, and the rate of change is so rapid that there's always going to be something that's legacy. So even if there is a predominant standard, you're always going to have to solve the problems of interoperability among heterogeneous systems.

That said, I don't think that developers are really going to put up with middleware wars. They just want to get their work done. I think we should figure out where things should interoperate, where it makes sense to have standards, and where it makes sense to have alternatives. If you look at what COM+ and EJB and CORBA are doing, they're all basically looking at the same problem space, but each one has made different assumptions or trade-offs in the specifications.

As an application developer, you need to look at those alternatives and say, "this is how these trade-offs are going to affect my requirements," and you pick the one that you think is going to work best for you. I think it's good that there's the ability to make that choice. If there's only one, someone's going to come back and say, "Well that set of assumptions doesn't apply to me," and then they'll go off and they'll invent something new.

What should be the goal of research and development?

Clearly, we're going to have to deal with interoperability and standards issues, figuring out what standards are needed and how to make those work. We need to be able to design middleware that is resilient to hardware changes and changes in the underlying communications infrastructure, so we can adapt quickly and get products out there for applications targeting new devices. And to reemphasize what I've said before, we need to find a way to provide power for the power developer but also provide a simple model for the person who just wants to build a typical application.

Is middleware moving toward integrated suites from a few big players? Are there business opportunities to contribute "best of breed" services?

It looks like right now we're in this period where things are

being consolidated. I think that's good for a lot of enterprises, because they can go to just one company and get the basic middleware they need. The services have been tested together. They can get support and consulting from people who understand how the different services work together. Those things are really good for the consumer. And the nice thing about a component-based architecture is that the best-of-breed model can still succeed. You get the basics in place, and if you've got good standard interfaces defined about how to hook in your middleware services, then people can plug in a higher-level service or a slightly different service without impacting the standard set that's already there. I think we will continue to have the sort of usual suspects with their suites of middleware, but there will be room for other people to come in and provide best-of-breed services.

What would you prefer to get from research if offered the choice?

One of the things that's really interesting to me is how people go about the process of building applications. What is their thought process, and how do they make design decisions? I

think we really need to understand more about how people approach the problem and are solving problems in order to make the products more accessible to people.

If you want to make something more usable, you have to understand where people are having trouble with it, and how they think about problems. But it's difficult to watch people building a distributed application. One, it takes a long time. Two, companies write applications

to give them a competitive advantage. They might be a little leery of letting someone within the industry see what they're up to. Also, if you just look at your current customers, you don't necessarily get the broad perspective of where people from different backgrounds have trouble building these complicated applications and what things are easy for them. So those are situations where having a completely independent observer makes a lot of sense.

What's the impact of technology trends?

For me, one of the interesting things is embedded systems in real time, because that's kind of what my background is. How do you scale some of these services down to work on an embedded device? And which services do you even need? Do you need to participate in distributed transactions? I don't know. Maybe you do and maybe you don't. If you're an application that's dealing with an embedded device, how do you deal with a device that's missing a service that you think you need? So there are some real interesting challenges there that I think are going to need to be addressed. Also, can you meet the needs of the real-time developer with the same programming model that you're providing to a line-of-business-application developer? There are a lot of different problems to solve

in that domain. Can you come up with one model that will work for both sets of developers?

Mary Kirtland is a program manager with the COM+ team at Microsoft. Her research interests include software architecture and design, approaches to building distributed applications, and the accessibility of component-based distributed application design. She received her ScB in chemistry from Brown, and she is the author of *Designing Component-Based Applications* (MS Press, 1998). Contact her at marykir@microsoft.com.



Doug Lea

Who's using middleware today?

I mainly deal with people who are involved with Java—mostly server-side programmers and developers. All of them use middleware in the sense that they use a language that has the ability to provide various distributed-object services. However, I think that the majority of projects I see are still using custom, socket-based solutions, JDBC connections to traditional databases, and similar solutions which are just barely middleware—really just light abstractions over fairly primitive capabilities.

Do you see trends that depart from this less structured approach in the future?

Yes and no. As people build more integrated applications, they naturally move toward more structured solutions. But, increasing numbers of people are doing things that haven't been done before, and the underlying services aren't available. The most fun examples are people who make things such as game servers or Internet relay chat servers. These are so different from what much of this middleware was developed for that they're striking out on their own and working from raw sockets and similar primitives on up, although not without some problems.

What kinds of problems does middleware present?

With most middleware, there are typically two levels of usage: a novice cookbook level and a sort of guru level, where if you know how it's built, you can tweak and rebuild parts of it. The large gap between these two extremes is difficult to cope with. Examples include needing to construct special proxies that perform caching or streaming. Solutions might include custom-loadable protocols and transports and a more open architecture. We still don't know how to do these very well. Very few prototype applications or research systems even exist as guides. This feeds into the notion of aspect-oriented programming and related research on how to create systems that are sufficiently open to solve real problems.

Do you think that there is an emerging need for customization?

Typically, the industry only does what is really necessary. For example, some of Doug Schmidt's work at Washington University has shown that there are many ways you can enhance and customize middleware to solve real problems. But

such work does not always make it into practice, and it seems that people reinvent the wheel constantly. I think that's a problem, because it's not easy to get these things right. There are very good reasons for some of the strange design decisions that go into middleware systems. When people cannot use this middleware, they often sacrifice quality, scalability, and performance to meet some special need.

What are some other important problems that researchers and developers have to solve?

One problem area surrounds scalability. People try to use systems in contexts that were just unimaginable five years ago. It's not uncommon to hear talk about a server that will support 10,000 simultaneous open connections. That's a very different design context than those that were anticipated when people were developing this middleware. So we need to consider scalability, fault tolerance, quality of service, and, as a typical consequence, nonprocedural protocols. Most middleware has been designed with procedural protocols in mind, but many systems evolve to use various nonprocedural ones: Multicast-based event frameworks, remote-execution frameworks, and mobile agent frameworks all exist in one form or another, but the most common versions are uncomfortably grafted on the same old stuff.

How do these problems compare with those of the past? Can you give an example?

The issue of transparency is one example. Middleware builders believed for a long time that they wanted transparency. But in the Java community, people were fairly sure they didn't want it. They now have to mature a level from that and say okay—even though there are vast differences as you go from a single, simple program to a concurrent, parallel, LAN-based, or wide area-based program—it would be nice to have some principled way of designing systems that incorporates any or all of those aspects without having to continually redesign or reimplement them. Many people have, unfortunately, just completely divorced their RMI-based code from their within-process code, because that's what the language encourages. It's not clear that this is a bad idea, but it's not clear that it's a good idea, either.

What's the next step for middleware?

The scattered approaches to some of the nonprocedural protocol issues need to converge quickly. Academics, in particular, have put out some pretty good demonstration systems, but they haven't congealed enough; they haven't matured to the point where they are attractive in practice. For example, mobile agents frameworks are still one of the open areas—it's difficult to integrate a framework like Aglets with a large-scale distributed-object system.

How successful has the transfer been of research done in labs and academia to industry?

I see the mark of academics, researchers, and advanced developers in almost every aspect of systems and software. There are three things you can do if you're a systems researcher. You can

try to demonstrate something that's never been done before and show that it's possible. You can focus on quality and try to create designs and engineering strategies that make existing things better, which is the niche that I fall in. And then there's finding how to apply these things to improve the productivity of building software. Most academics like to focus on the first one. Unfortunately, that creates problems when people take prototypes and find that they can't really be used in practice.

Is there a role for freeware that is similar to the role of tools?

I've written a lot of freeware. Like many academics, much of what I do tries to raise both the state of the art and the state of the practice. From that point of view, getting results in many cases is not writing a paper, but building something and letting people use it. I also think freeware has a role in promoting flexibility, quality, and openness. One of the ways that you can deal with a middleware product that doesn't do exactly what you want is to get the source for it and help improve it, or at least change it for your purposes.

Rather than quality, what about the support issue? That requires more than just putting something on the Web. Who will maintain the code and evolve it?

I haven't found this to be a problem. For example, C++ library code I wrote 15 years ago was picked up and supported by Cygnus. Similar things have always happened when I haven't been able to support other software myself.

Do you see a dominance, either of architecture or homogeneous versus heterogeneous?

No, I don't. In the part of the world that I know best, there is nothing like uniformity across vendors of middleware, operating systems, languages, or anything else. Even though I've been focusing my own efforts on Java for the past few years, this heterogeneity doesn't upset me in the least. I hope it continues.

Do you think that middleware impacts the underlying system software or vice versa?

That's a really great question, because it's at some level impossible to distinguish operating systems from middleware. Most do exactly the same thing, at just slightly different levels of abstraction. Minimally, there's increasing pressure put by middleware on operating systems to do some things better. Examples include at the transport level, zero-copy approaches to getting data across wires. This is something that can only be done with the cooperation of an operating system, but was entirely motivated by middleware concerns. Will they collapse into the same thing? I think eventually, although eventually could be an extremely long time.

Has the transfer research we've discussed earlier been successful?

I guess the answer has to be yes in that, without academics,

many of these things would never have happened. As one of the best examples, the whole issue of fault tolerance and multicast started in academics with people like Ken Birman. There was no industrial interest whatsoever. But now that whole school of research has become very important. As people become more concerned with fault tolerance, they look at what academics did. Sometimes, unfortunately, there isn't much transfer and people reinvent things.

What do you think we need most to come from academia?

I think it is important for at least some academics to turn from a focus on innovation to a focus on software quality. Research focused on how to design systems with high reliability, reusability, and strong performance is desperately needed. These are the things that academics have the time and opportunity to do that most people in the industry never do. Usually, we work to build exactly the kinds of components and systems that practitioners are building, but we have the opportunity to think harder about them because we don't have product deadline pressures.

We can try to generalize solutions and to come up with more clever algorithms and designs. I hope that more academics take these difficult issues head-on. We have almost more than enough prototype systems that push distributed objects in one direction or another, and not enough of them that demonstrate what the engineering principles behind them should be.

It is important for at least some academics to turn from a focus on innovation to a focus on software quality.

How do various technology trends impact middleware?

I've mentioned fast networking, connectivity, and scalability and how hardware can support greater demands. Also, there are issues surrounding mobile computing, higher-order protocols, and security. They are, of course, vastly important and vastly difficult. Some of these problems are so difficult that many people shut them out of their minds. Denial-of-service attacks are especially difficult to deal with, and there are very few people with good, generic answers.

Quality of service also becomes more important and more interesting every day as people find that you can't just optimistically believe that you'll always have enough CPUs, wires, and storage. The impact of the real-time community is only increasing, because the people from those systems have had to deal with saturation problems for a very long time. Practically none of the solutions are directly transferable. But the approaches are transferable, and I think we're learning a lot from that community.

Also, while academic work on fault tolerance has been around for a long time—and there are still many research issues—I think they're slowly becoming applied to real systems. And from embedded systems, there is the notion of discovery and naming of detachable services—Jini is by far the best example, which I suspect Ann [Wollrath] discussed. This has a big impact on the larger issues of distributed computing. And the Web has been a constant force on all of these things for the past five years.

What about composability?

There are several ways to answer this question. If you have subsystems that are using different kinds of middleware, the glue and bridging issues (across transports, marshalling strategies, naming services) are truly ugly. Any large system has a lot of such code, and it seems like many programmers do nothing but write glue code their entire lives. So, there are those kinds of very practical composability issues that have a solution, but the solution is so distasteful that you almost wish it didn't exist.

The more fundamental composability issues are at the level of building subsystems, engineering things from a modular perspective. I think CORBA was instrumental here. CORBA was intended to be rather small, with a set of interface-based services you can compose together—it's a story that's mostly worked. For example, the idea that naming could be provided as an interface-based service was still a novel idea when CORBA was first being introduced. There are lots of very pragmatic problems, but it's a small success story that was initiated with OMG.

Doug Lea is a professor the Computer Science Department at State University of New York, Oswego. His interests include algorithm design and object-oriented languages. Lea received his BA, MA, and PhD from the University of New Hampshire. Contact him at dl@cs.oswego.edu.



Annrai O'Toole

What is the role of middleware solutions today and in the future?

Middleware is at the convergence of two technologies. One is the underlying set of technologies used to support basic distributed computing—the basic program-to-program communication over networks. We've developed RPC systems and that kind of technology over time to solve those problems. We do that pretty well today.

The other set of technologies is used for large-scale system development. Here I'm thinking particularly of transaction monitors and things like that, developed in the 70s by IBM and others to support tens of thousands, if not hundreds of thousands, of users who want online access to information systems.

Today, with middleware, we're trying to blend those two worlds—the distributed-computing world with the large-scale systems development world—to give us an application platform that lets us host information services that can be accessed by millions of users in a secure, reliable way, with those services themselves hosted on multiple machines communicating together. The broad work of bringing together these two technologies is pretty much done. In the future, I see us focusing on ease of use, which I would break into two separate categories: ease of development, for making it simpler to develop applications on top of some middleware platform, and more importantly, ease of deployment.

Our customers say that it's very easy to get a system up and running to do development work. The hard part for customers is the deployment issues. Rolling out an application to potentially millions of users where that application itself is coordinated over hundreds of server machines is a nontrivial task.

Deployment also assumes the next step—management afterwards.

I use the word deployment quite carefully, as opposed to management, because when you say management, people instantly think of tools like Tivoli, SNMP, and technologies like that, which we normally think of as network management. Deployment issues around software servers built on a middleware platform are slightly different, because what you want to control many different properties of the server such as its transaction behavior. For instance, a simple two-phase commit transaction with no recovery logging is perfectly adequate for one scenario, but if you scale it up and involve multiple databases, you need that recovery. You don't want to recode all of those services just to add a property like recovery.

As an even simpler example, let's say you build a piece of business logic, credit-card authorization or something like that, and then you want to deploy it—that's a very typical customer scenario. You build a simple piece of logic and deploy it to 25 users and it works fine. Then it gets very popular and you want to deploy it to a thousand users. Typically, what you would have to do today is to recode the whole thing. That isn't very useful for you as a customer. You shouldn't need to recode a piece of business logic just to change its deployment characteristics.

So, under deployment, you see more of these subsequent services, not just management in the terms you define, but maintenance, evolution of the code...

Yes, load balancing, recovery strategies—all those things have to be configurable, rather than programmable.

Is there something that you think is an important problem that we should address that hasn't already been addressed by the community?

Not that I can see. What I've learned in being involved in this particular industry for the past 10 years or so is that in all the time we've been doing this stuff—whatever technologies you want to take, CORBA, EJB, or anything like that—there's absolutely nothing new in them. They're just slightly different applications of something that we've already known. It's these different combinations of technologies that are well understood that is interesting. I don't believe we've done anything new.

Do you think that we'll see something new in the future?

I don't think so. My current view on middleware is very similar, I remember, to my view on TCP/IP back in 1989. When I first had a Telnet conversation with someone in Japan, I was really excited, and I ran around to everybody and told them, "Hey, this Internet thing is going to change the world. The TCP/IP thing is way cool." And everybody said, "No, you know this TCP/IP thing can only be programmed with sockets."

Then, all of a sudden the Web came along. Again, Web tech-

nology wasn't new. Apple had the Hypercard product for some time before the Internet. So all the concepts that the Web popularized were already there, but it was the combination of those hyperlink technologies with the underlying TCP/IP—that particular unique combination—has absolutely propelled the Web onto the world. I feel the same about middleware. All the various pieces are already floating around.

How do you feel about freeware? Isn't there already a lot of implied, if not true freeware, given-away software, for example, both by Sun and Microsoft and even by your company, that already exists?

The model that the world has moved toward over time is to charge people for the deployment, not for the technologies to develop applications. There's much more money to be made in deploying applications. In the freeware world, what you're seeing is a lot of tools that are available for developers to experiment with and understand. But going back to my earlier theme about deployment being the hard thing, most people can take any of the middleware technologies that exist today in any form and build something fairly quickly with them. It's when that thing gets big—when the patterns that people use to, let's say, deal with load-balancing or with highly complex applications—that's where the knowledge is. That's also where companies like Iona make their money—in helping people deploy.

So the focus will be more on the deployment, both in development and in earning money?

Exactly. That's the implicit model behind everybody's efforts. Microsoft will give away the development tools, or price them very low, to make NT server sales. Sun is promoting EJB and Java in order to drive hardware sales.

I can understand Microsoft and Sun, which have other parts of the house that can earn a lot of money and make the business viable, giving away this basic plumbing. But your company, Iona, is a basic plumbing house. How will that strategy work for you?

The interesting difference between our approach to plumbing and everybody else's is that if you talk to Sun, the plumbing is pretty much a single-platform plumbing, just for Java. And if Microsoft talks to you about plumbing, it's a single platform for them, for NT. We view the world differently. We view it as being a very heterogeneous one. We see that lasting forever.

The investment that the world has just made in the Y2K problem demonstrates that when the industry was faced with the option to either throw out old software or rejuvenate it, by and large people rejuvenated it. There will be mainframes in 100 years' time and Cobol applications, too.

When we go talk to customers, we say that what we're selling you is plumbing that you can think of as an insurance policy. It works on all the systems that run today, and we've demonstrated in the past that we can keep up with technology

developments. When new things come online, we can integrate them into our plumbing.

We can make money very easily in that environment, selling that kind of rich, middleware plumbing.

I see how you motivate people to buy your plumbing, but given the general perception that middleware is cheap, you still have to find other ways to survive, won't you?

Our medium-term strategy is still to focus on the plumbing. We go back to the plumbing and point out the complexity and richness of what we're offering. We say to people that if they're looking for plumbing to run their business on, we have it. One of our customers is Credit Suisse, who are doing their online retail banking with us, combining mainframes and Unix boxes, using Orbix. The plumbing they need is not just a vanilla EJB thing.

So you adapt and specialize for them.

Yes, basically. It's not so much customizing, as it is offering people choices. They can have "cheap-and-cheerful" plumbing, if they want to do something simple—if vanilla will work. They can have a slightly midrange flavor that's got a lot more features built in, or they can have the really high-end stuff if they want to run hundreds of millions of transactions per day and have all the features of recovery and load-balancing—the whole nine yards. It's a tiered approach

What will be the dominant underlying middleware environment in the future?

The only thing that's going to survive in the middleware world is a standards-based approach that will run on everything. I have a famous line, "There are no ugly babies." When we're selling a customer some infrastructure to bring lots of things together, it has to run on everything.

What about various technology trends in middleware?

From our point of view, it's all pretty much good news. The overall trends we predict is that every single computing device of any size will have an awareness of its network environment at a very deep level. Once, when you would buy a computer, you had to add some sort of network card. Nowadays, the idea that any new device is going to be somehow network-connected is a much more integral part of how things are built. That's really good news for us, because with all these online devices, you need some kind of infrastructure connecting them all together and providing an environment for the applications that are going to run on those devices so they can communicate with one another. That's exactly what middleware is all about.

How do you differ from your competitors?

When you think of our approach to middleware, we're quite different from some of the other guys, guys like HP, Microsoft,

When we're selling a customer some infrastructure to bring lots of things together, it has to run on everything.

and Sun. They're all trying to create a layer above their own operating systems. They're trying to create a new platform for programmers to program against that makes it easier for people to develop network-based applications on their platform. With Microsoft, it's COM+, with Sun, it's Jini, and so on. Our view of middleware is slightly different. We view middleware's role in life as a way to connect lots of things together.

Interoperability is more the key for you.

Yes, the "no ugly babies" idea. As far as we're concerned, we love COM+ just as much as we love EJB. Our job is to make them able to talk together seamlessly and we use middleware to solve that problem. We're not in the middleware business to create another island of computing.

Which approach do you bet will win?

Obviously, my own leaning is clear. The first person who ever taught me about computing told me that incompatibility means business. He's been right so far, and I don't see anything slowing incompatibility down, so that will continue to be a very good source of business.

Others have a different view.

Yes, but their approach is to say, "Make us the center of your universe and design everything around us and we'll work things out." That's more of a hub-and-spokes approach that requires the customer to move everything toward some center or gravity, be that Windows or Java or something else. Well, we don't see that as being a realistic option for the vast majority of people who can't assume some sort of center of gravity.

Anraí O'Toole is cofounder and chief technology officer of Iona Technologies. He is the visionary behind the evolution of Orbix, Iona's flagship product. He speaks often at trade shows and technology conferences about the business and technology implications of middleware issues. Before cofounding Iona, he was a researcher at Trinity College, Dublin. Contact him at aotoole@iona.com; www.iona.com/; IONA Technologies, The IONA Bldg., Shelbourne Rd., Dublin 4, Ireland.



Joe Sventek

Who's using middleware?

A lot of people are using it, but they don't realize it. Middleware is finding its way into all sorts of systems. Very few people actually program directly into sockets anymore.

How can we make middleware more accessible to users?

We have to continue to embed it into helper classes. For example, we define specifications for CORBA in IDL, and then we expect people to actually architect their systems as well as define any additional specifications. We need to provide a helper class that abstracts away from the idiosyncrasies that technology such as CORBA needs to allow multilanguage

tasks—this ensures that everything fits in more naturally with the user's language.

So you think we'll see more user programming?

Yes, it's still an open question about how we're going to make all this stuff available to everybody, but it's gotten to the point where every application is practically a distributed application. Now, applets talk to back-end systems, which talk to a database system, and middleware is critical in gluing these things together.

What important problems do researchers and developers need to solve?

I'd say the biggest and most important problem is that most middleware packages today are one size fits all. We're finding that as computation appears in more and more everyday devices, we need to partition functionality so that users don't expect a Nokia handset to have the same amount of code to support communication as a gigabyte server requires.

I think we also have too many middleware platforms. As an industry, we need to come to some agreement on a common model to help us define which things interact with each other. Right now it's really difficult if somebody asks me, "Which one should I use?" Well, what are you trying to do? What's the heterogeneity of the platforms you're looking at? What's the heterogeneity of the languages you think your components are going to be written in? One of the things we tried to do in the Object Management Group was to define things so that we had all those things available, but I'm the first one to admit, having written the IDL specification language, that it's not complete. It's not perfect.

Are there any red herrings?

I don't know of any. One of the things that confuses people is Internet protocols versus the protocols that underlie CORBA, DCOM, and RMI. I think these could be considered as red herrings.

How does this compare to the past?

It's certainly a lot easier to perfect an application using CORBA or RMI than it was using sockets, because we've eliminated the need for the programmer to have to keep state machines synchronized on either end of the connection. That said, some middleware packages tend to focus very heavily on request/response only. We're finding that for certain kinds of systems we still want asynchronous messaging, so I think we need to have a more unified model that actually encompasses different ways for the components to interact and different notions of binding.

Some universities are looking into another important research problem: reflective middleware. If we try to provide a certain quality-of-service level over a binding between a pair of components, it's usually done through an explicit binding. It would be nice to have some kind of introspective or reflective mechanism to help us see how the system is built. Then we would know how to specify what we want, because what we ask for might differ depending on how it's built.

What's the role of freeware in middleware solutions?

I'd really like to see freeware take off. I did a lot of work early in my career on what would now be called open source. I think things like Linux and publicly accessible middleware solutions are a way to get us all to a common model and technology base. There are no panaceas, but it certainly is an exciting area.

What will be a dominant underlying middleware environment of the future?

Well, there won't be a single hardware architecture. There certainly won't be a homogeneous environment. I think that the goals that drove us to CORBA specifications are still going to be with us until the end of time. If we get to a common hardware architecture and common software environment, then there's no innovation and I don't see how we can address new problems. So I think it's going to continue to be a problem, especially as more and more computing moves into smart devices. It's going to be a different kettle of fish to try and get those things to work together.

What about middleware systems versus system software?

You mean middleware moving toward integrated suites with a few big players? I think one of the problems with middleware is that if it's successful, we can't see it. It's always been really hard to demonstrate it as a great innovation, and my feeling is that most people will find it's difficult to make a lot of money off it. We use it as glue. I see middleware, and the whole industry, moving toward a common model, because it'll mean we've eliminated one useless degree of freedom. But I don't think we're going to see it consolidate into a few big players, especially if the open source or freeware environment really takes off.

But if people can't make money off middleware, won't that speak exactly in favor of a few big players?

Well that may be. CORBA and RMI, for example, have few suppliers. So I guess in that sense, you won't see a lot of players in each space, but I still think we'll see lots of different players, because all of them are approaching middleware as a one-size-fits-all solution. Eventually, something that could use middleware where that one size doesn't fit will force us to build something new or try something different.

I'd say that the really interesting bit about middleware is that it's not something that people want to pay a lot of money for; they think they ought to be able to get it just the same as a network system service. The same way they don't want to pay extra for system calls in an operating system that should just come for free.

Are there business opportunities to contribute best-of-breed services?

Yes. We discovered with middleware systems that if we just

have a specification for an event service or a naming service, we get a kitchen-sink specification. It solves all possible problems and does none of them particularly well. So, yes, I'd like to see best-of-breed services built so that they're partitionable, so that it's easy to select the piece that might keep us from having to do it on our own.

So partitionability for the sake of composability.

Exactly.

What do you think about the transfer of research done in academia and research labs to industry?

I'd say that it has been reasonably successful. CORBA's specifications as well as RMI and Java-system advancements came from work done in research labs. However, I also think that people doing work in academia are having a tremendous impact. Academia, research labs, and industry are all tied together. I know specifically of a reflective middleware project at the University of Lancaster in England; if that work were done on an

open-source platform, then it would be very easy for many of us to benefit. Completing that thought, I believe the ability to break things apart easily so that we only use what we need to use along with support for quality, dependability, availability, and security are absolutely crucial for the mission-critical systems that eventually affect the industry.

What technology trends will impact middleware?

Well, a number of trends will impact it in various ways. I think that faster net-

working will let us support more communication bindings. We can use middleware more often, have more connections going, and still obtain maximal speeds over the wire, whereas if we use a smaller network like a 10-base T, we start to see things get sluggish after a few of these connections set up.

Mobile computing is an interesting beast. I think with the ubiquitous connectivity and the fact that everything has processing power in it, these technologies are going to be that much more important because we have to plug them together. We won't have to resort to programming at the socket level or worse.

Real time and the quality-of-service issue are definitely important as more networks focus on providing us with bindings or channels with different qualities. Middleware will then be able to take advantage of that.

The integration of the Web with middleware should prove interesting. The Web enables us to operate in client-server mode without having a problem with delivering the client code to the client system. With Java-enabled browsers, the client is faulted in as byte codes, while the Java model only permits the applet client to communicate back to the server from which the byte codes were obtained. We still would like the client to be able to talk to a variety of services, in addition to the server

The problems with middleware is that if it's successful, we can't see it. Most people will find it difficult to make a lot of money off it.

it came from. Heterogeneous systems are here to stay, whether in the language space, the kernel space, or the networking space. These things ought to be tailored to make them easier to use in a heterogeneous environment.

What about manageability?

Well, I don't have any good answers for that. I do know that we really like our systems to depend as little as possible on static configuration files and things like that for successful behavior. I think that we'll move toward broadcast or multicast space rendezvous, such as what's been going on with the server-location protocol in the IETF. We're in the process of building systems that are extremely large scale, and it's more than just during execution time. We've actually had to resort to some fairly sophisticated management techniques to deploy, instantiate, initialize, and change configurations while we're running. I'm hoping that this work will prove to be useful in a more general environment than the one we've been working on. I think it is one of the great challenges. As this area became more and more populated, we had to go to a standard domain service interface, to a standard set of domain name servers, and then to a standard set of protocols for how we actually linked in and got information. In the Internet's early days, everybody was building their own ways to set up a host, networks, or some other static way of doing their bindings. As the system grew to a certain size, we had to go to more of a remote-procedure-call way of expanding and searching. At least with the Internet, because the Department of Defense imposed it, we have fixed points we can go to for this initial rendezvous, those being the Domain Name Servers for the top-level domains. As systems become extremely large scale, I think we have to go more to a multicast initial acquisition of the initial configuration.

Joe Sventek is the director of the Software Technology Lab at Hewlett-Packard Labs. He is also HP's laboratory scientist for distributed computing. His current research interests revolve around middleware mechanisms in support of extremely large-scale distributed systems. He received a PhD in nuclear chemistry from the University of California. He is a member of the IEEE and the ACM. Contact him at Hewlett-Packard Laboratories, 1501 Page Mill Rd., MS 1U-3, Palo Alto, CA 94304; sventek@hpl.hp.com.



Ann Wollrath

Who is using middleware?

Almost every application developer is using middleware in some form. Enterprise application developers use business object frameworks, database access software, and distributed programming infrastructures, for example. Today, where connectivity is key, applications ranging from large enterprise software systems to small embedded systems need to leverage middleware to get the job done.

How can we make middleware more accessible to users?

Because connectivity is so important, most applications today are by nature distributed, requiring the developer to consider things such as partial failure and concurrency. Distribution has its challenges, and making it more accessible is a challenge as well. We think that it is important not to make distribution completely transparent. If it is—if distribution is completely transparent—an unexpected network failure can cause application programs to fail in unexpected ways. Some applications can make distribution transparent, such as those that place all distributed communication in a transaction, but not every system is transactional, so a lot of applications need to take concurrency and partial failure into consideration. Making systems more transparent can actually make them less understandable, and therefore less accessible to the user.

You don't think that most of this reliability work and fault tolerance work that was mostly done on clusters and local-area networks would apply to wide-area networks?

Not necessarily. Clusters are very different systems, with different failure assumptions. They're more tightly coupled than wide-area networks. In wide-area networks, you have more frequent failures, so you have to make different assumptions when you're programming in those types of environments. You could probably have more transparency in a cluster situation, but not in a wide-area network.

Given that today's application software can be complex, how do we make middleware more accessible?

On one hand, many users seem to be asking for simple-to-use APIs that are quite general and handle everything for you. On the other hand, other users want to reach down into the system's guts and tweak some parameter that you never thought of. So there's always a trade-off between users who want simplicity and generality, and those that want complete control over the system.

What are important problems that need to be solved?

A really hard problem that needs to be solved is dealing with software evolution. We need to be able to design extensible systems that make a small set of assumptions about their environment and allow room for growth. There are some partial solutions to these problems, but general solutions are very hard to define.

How do distributed systems of the past compare with today's middleware systems?

Distributed systems seem to have kept pace with language systems, moving from procedural to object-based to object-oriented distributed systems. Basically, the abstractions have gotten better over time.

What will be the dominant underlying middleware environment of the future?

In a connected world, there must be some homogeneity, if you will, somewhere in the system, or we couldn't be con-

nected. Network entities have to agree on some protocol to communicate. But I don't think that there will ever be a single hardware architecture. There might be hardware architectures that are useful for a large set of applications, but it seems unreasonable to have the same hardware architecture or operating system for a PC, a cell phone, and a set-top box.

Consequently, probably also, no single standard middleware?

Exactly. There will likely be heterogeneous hardware and operating system environments, but there must be some set of assumptions that communicating environments need to make so they can talk to each other. And since different environments make different assumptions, it just depends on where you want to draw the line.

For example, in the Jini system, there's a protocol for devices and services to announce themselves to a look-up service. The look-up service stores a smart proxy for the device or service. A client can query the look-up service for what it is looking for, and get the smart proxy for the device or service. Interestingly, the Java byte codes for the smart proxy is downloaded to the client and the smart proxy code can speak any protocol that it wants.

In the Jini environment, devices or services need to announce themselves in a particular way and make a Java smart proxy available, but devices and services can be non-Java entities. In Jini, there's an announcement protocol and a look-up service. But the protocol between the proxy and the service is up to the proxy provided by the service. This lets the protocol between the proxy and the service evolve over time.

It seems that you have a nice, clean architecture with a small number of assumptions, but that happens for most architectures at the beginning. Later on, when both industry and academia start making additional assumptions—for example, to fill more quality-of-service requirements or fault resilience—how can you prevent pollution of these assumptions with all kinds of additional assumptions? How will you evolve this architecture cleanly and simply?

In the Jini core architecture, there really is a small set of assumptions. If you want to build something on top, you can do that. Jini makes no assumptions between the client and the proxy; the client uses a smart proxy, defined by the service, to communicate with that service. If the service wants different quality-of-service requirements, if it wants to evolve its protocol over time, the system allows for that. The client would have to get a new proxy for the service, but that proxy can evolve over time. There is room for growth there. The Jini architecture is small and simple—services can be extended as necessary without changing the core architecture.

You're saying you can evolve it only between the clients? Isn't there

a certain amount of initial functionality that has to be fulfilled, such as booking up to clients, for example.

Right. There is the announcement part. Evolving protocols that are fixed is really tough. You can't just shut down the world's networks and, let's say, evolve TCP/IP overnight. So you have to try to define the protocol right initially, but still make the smallest set of assumptions that you can make, and allow the system to evolve. Once you pin down a protocol, you're stuck with it. The protocols should be pinned down in the fewest places possible.

It seems that manageability might be quite a big problem, given the size of the problem space that we're addressing and the amount of objects and everything.

Certainly, it's a problem. There are lots of systems and network management solutions available, but they are very particular to the systems they're managing. There's no real general framework out there that I know about. In Jini, there's some automatic configuration built in so the system is somewhat self-healing. Because the system is based on the notion of leases, when a service makes itself available through the look-up service, the

service will keep renewing its lease on its entry in the look-up service. If that service goes away, the entry in the look-up service goes away, so stale entries don't hang around. When you plug in a device, it just announces itself—it makes itself available. When the device crashes or is unplugged, or the service crashes or becomes unavailable because of a network partition, the service won't be able to renew its lease and it will disappear from the look-up service. Clients will no longer look it up and access it. They can look up something else instead, another printer that is still accessible, for example.

Evolving protocols that are fixed is really tough. You can't just shut down the world's networks and, let's say, evolve TCP/IP overnight.

Transfer of research done in academia to research labs in industry—how successful has it been?

In my own experience at Sun's research labs, the philosophy has been to fund an internal research program and, when a specific technology is mature, transfer the technology to a product group. In the technology transfer, Sun has been most successful when the people who develop the technology get transferred along with the technology to the product group.

I led the effort to develop the Java RMI technology. We were doing research in distributed systems using Modula 3, and ended up designing a distributed object model for Java. The Java RMI technology was transferred, along with its developers—that's us—to the Java Software organization. We rolled Java RMI into the JDK 1.1 release. That was a successful technology transfer.

Ann Wollrath is a senior staff engineer with Sun Microsystems where she is the architect of the Java Remote Method Invocation (RMI) system and one of the original architects of the Jini technology. Previously, during her tenure at Sun Microsystems Labs and at the MITRE Corporation, she researched reliable, large-scale distributed systems and parallel computation. Contact her at ann.wollrath@sun.com.