

MIDDLEWARESPECTRA

incorporating Enterprise Middleware

Volume 14, Report



Contents

MIDDLEWARESPECTRA — May 2000

- | | |
|--------------------------|---|
| 2. | The ASP model and middleware — at LivePerson
Kevin Delafield, Vice President, Application Architecture, LivePerson |
| 10. | SAP, IMS and deploying middleware at Lucent
Vince Marzella and Mary Kathleen Liberto,
Lead Developers — Order Management System, Lucent Technologies |
| 16. | Real time changes are need for e-business
Vivek Ranadivé, Chief Executive Officer, TIBCO Software |
| 24. | Middleware is for operations
[or, development complexity kills its use]
Charles C. C. Brett, President, International Advisory Board |
| MIDDLEWARESPECTRA | |
| 32. | Integration ... what can middleware deliver?
Keith Jones, Principal Consultant, IBM Corporation |
| 38. | EAI — arguments for active or passive integration?
Rosemary Rock-Evans, Consultant |
| 46. | Middleware 'everyware' — a new age?
Phil. Manchester, Consulting Editor, MIDDLEWARESPECTRA |
| | |
| | |
| | |

The ASP model and middleware — at LivePerson

Kevin Delafield
Vice President, Application Architecture
LivePerson

Management introduction

Kevin Delafield is the Vice President of Application Architecture at LivePerson, a start up based in New York. His responsibilities include key infrastructure and much of the development group at the Company.

LivePerson (www.liveperson.com) offers real time customer relations management over the Internet. In effect it is an Applications Service Provider (ASP) providing an Internet solution completely serviced on host NT systems (which are themselves outsourced). All that is needed to access the LivePerson function is a browser — to access the service.

In this interview, Mr. Delafield (who is ex-Transarc) discusses how middleware was put in place to support a large number of simultaneous customer conversations with a minimum of staff. This involved the use of Softwired's iBus publish and subscribe messaging as part of the mechanism by which LivePerson supports its customers' staff.

The rationale

We are focused on the provision of customer relations management. Our objective is to make our (LivePerson's) customers' staff more productive — specifically to enable each of their sales or support staff (typically) to handle several customers at a time. In particular we target e-commerce sites that need to make the most of their sales/support people, not least because most of their customer contact work is accomplished over the phone or via the Internet.

This plethora of 'customers' can, however, sound confusing — for there are:

- **LivePerson's customers; these are the organizations with sales/support staff who pay LivePerson to provide a customer relationship application which can maximize the utilization of their people**
- **our customers' customers, who are the people calling in to obtain the information or support from our customers.**

Although not absolutely correct, let me — from now on — refer to LivePerson's customers as 'clients'. 'Customers' will refer to those people who are calling our clients looking for that information or support.

What we provide are real time text chat capabilities. For example, if you are an e-commerce site that sells some product and you want to provide an opportunity for your customers to interact with you in real time (as either a sales tool or as a trouble shooting tool), you could put an 'I would like to contact someone' link on your Web site (having already become a client of LivePerson). This link would enable your customer, using his or her browser, to contact your sales or support staff direct.

Meanwhile your staff would have logged into the LivePerson site — as an operator — using a browser. As your customers clicked on the 'I would like to contact someone' link they would connect to the physical LivePerson ASP Web host where they would be connected to your staff. To the customer, however, it would look as if the connection had been made to your people on your site.

Once the link is made, a chat in real time is established with your operator. Your operator can now interact by:

- **text chatting**
- **pushing content (text, attachments, additional links, etc.).**

All this occurs in real time — without any requirement for software installation by you (our client) or your customers. Everything is HTML and, therefore, works on and with almost all browsers and almost all operating systems.

Furthermore, the electronic conversations are 'recorded' in our databases. This is significant because it enables us to provide additional services. For example, our clients can data mine the interactions after the chats have occurred. This gives feedback about what is happening and allows trends to be identified and acted upon.

Our rationale, therefore, is to increase the effectiveness of our clients' operators by giving those operators the ability to interact with multiple customers at any given time. The objective is to reduce the cost of support for each customer coming into your Web site. Using the LivePerson service, four concurrent users do not require four concurrent operators; they require (at most) one operator.

The virtual Call Center

In effect what we deliver is a virtual Call Center where you, our client, does not have to invest in a Call Center and all the software and systems involved. In addition, you can locate your sales or support staff wherever you like. If you want to provide 24 hour service around the globe you can have the appropriate operators connect over the Web at whatever times you like and from wherever you like. You do not need to establish geographical locations — for the Web bypasses this.

As you might imagine our initial focus has been on organizations with limited sales/support resources who need to make the most of the productivity of their people. Typically our clients are e-commerce sites who use LivePerson as a sales/marketing tool and for regular customer service — where customers may have issues or problems that they need to contact an individual about. We also have plans for taking this concept further — with other or new types of real time interaction and/or delivery channels.

Technically, the way this works is, as you might imagine, complex. Essentially, once an organiza-

tion signs up with LivePerson, it puts all the materials that it might wish to push onto our Web site. This sits 'behind' our 'interaction engine'. It is the interaction engine that provides the transparent (for all intents and purposes) connections to your customer so that he or she perceives the connection to be direct with you. Your customer will have no idea that Live Person was involved.

When you sign up, besides uploading all the information that might need to be pushed to customers, you customize your presentation to your own look and feel as well as set up pre-formatted responses to those questions or issues where this is appropriate. In addition, if you need multi-country support, you can set up individual language areas ... and so on.

Other customizable features include the ability to create surveys, for example. These can then be mined to provide statistics, satisfaction rates or whatever might be useful. Remember that, as part of the LivePerson service, all the contacts carried through our interaction engine are stored in a database so that these can be mined as well.

Providing all this is, however, not simple. Maintaining state and connections between (at its simplest) one operator and several customers has to be accurate.

When you have multiple operators spread across multiple client organizations working with multiple databases and support files, the need for scaleable messaging and communication is paramount (Figure 1.1).

Technical challenges

The primary technical challenge we originally faced was that the current Internet is not designed for real time interaction. In addition, in order that we might gain maximum market and mind share (and win 'clients' quickly), it was necessary to make sure that we introduced as little overhead as possible to our clients when setting up their new accounts with LivePerson.

What we had to avoid was asking our new clients to install a database server or Web server or specific application server (or some combination) at their site. If we had tried this it would have substantially reduced our potential market. The ASP model was far preferable; it avoided the need for either us or our clients to install infrastructure.

But that brought us back to the need for real time interaction over the Internet. While the ASP model is excellent for concentrating excellence, we had to provide the infrastructure that made sure that people could chat back and forth. This led us to the need to select a least common denominator technology for communications over the Internet — namely HTTP 1.0.

But providing both scalability and real time interaction with HTTP becomes a very difficult challenge. Specifically, every firewall incorporates proxy servers which only allow HTTP traffic. This is a more complicated issue, with both technical and business dimensions, than it initially appears.

The reality is that many customers only make e-commerce purchases if these purchases are small ticket items — maybe gifts or flowers or similar. For major purchases — cars, homes, vacations, airline tickets, etc. — they usually prefer to interact with a person before they actually commit to buy.

Furthermore these larger purchases are often undertaken during the working day and from within a corporate environment — while people are at work, during lunch time or when the boss is away. This means that these purchases are made from within the very types of environments where the greatest restrictions exist. Often corporate firewalls prevent cookies or remove Java or ActiveX application code.

The implication for us was that, if our clients wanted their customers to buy after interacting with one of their operators, we had to make sure we could provide the greatest possible reach from their (our) Web site. This is why we have adopted the least common denominator — HTTP — even though it was not designed for real time.

So the base issue is that we have had to introduce a polling system where we periodically poll (say) every 5-7 seconds. In computing and communications terms, this is a very expensive process. For example, the average user hitting Yahoo may only have 12 to 14 HTTP hits per 'conversation'. In contrast, the average customer using LivePerson to chat with our clients may have 300-400 hits in total per conversation.

Load balancing and other complications

This meant that we had to identify a way to maintain the system which we had built around a cluster

of Web servers. We had to support a load which could rise to 1200 hits per second. In order to do so we needed to introduce some form of load balancing. Currently we use Alteon's Traffic Control software to provide round robin support.

To understand why we chose round robin for load balancing I need to go back into our original architecture. Looking back, some very low level detail had a major impact on the way we had to architect our solution. The Alteon software offers several different methods for load balancing, including by:

- sticky sessions
- CPU load
- active HTTP sessions
- round robin.

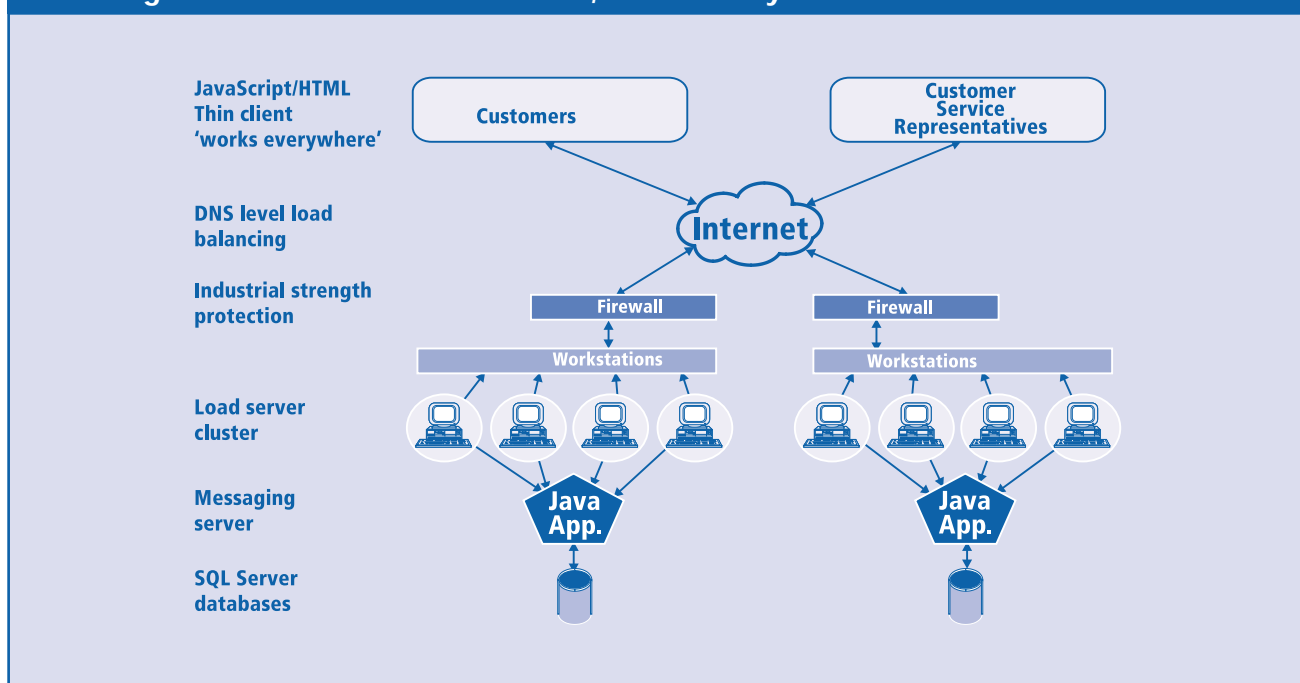
We went for round robin because it provided the most equitable distribution of hits across a cluster. The issue we encountered was that many of the high volume ISP sites — like AOL or a large company like American Express or an airline — originate their Web sessions from just two or three IP addresses. All traffic from AOL looks as if it is coming from the same IP system address. Therefore, using sticky sessions with any type of load balancing algorithm in a Web server cluster would have meant we had the potential to overload specific servers.

This is a real problem for people who need to do some type of sticky state management. The way our chat application works is that we use the browser simply as a presentation of the current state — which we have to maintain at our site. We provide an entire interactive GUI for our client's operators; this acts like a real time application. But the state of that application is not maintained in the browser: it is maintained for the most part in our NT cluster. By adopting round robin distribution, we lost the ability to use sticky sessions as the way to maintain state for a particular browser — which was a challenge that we addressed via iBus (see below).

Then we had to confront the need to have a single system image across an entire cluster of Web servers which would have extremely high volumes of traffic coming through. The issue here concerned the choice between a traditional three tier or a two tier architecture.

Having Web servers running against one database was not acceptable — because the database would become a single point of contention, and failure. Replication was not an answer. Another reason why a single database was insufficient was that our application is a real time one: most OLTP databases were not designed to deal with the high volatility we need (they were designed mostly for 'read often/write occasionally' rather than the high

Figure 1.1: LivePerson's redundant, multi-facility distributed architecture on JMS



volume of read/write throughs of a small sub-set of data which occurs in our operator-customer sessions).

Complicating this still further was that we needed to be able to read an entire block of accounts, not just a specific account. We needed to be able to separate the traffic within the cluster while also providing a single system image without having a central point of contention.

So the challenge was how to do all this within the context of the round robin load balancing while avoiding single points of failure and contention. The question we posed to ourselves was — how do we remove every single point of contention while also providing the single system image?

We tried several possibilities. We did try a database solution first. We tried a single application server — but we found that also became a single point of contention (as well as being extraordinarily difficult to replicate or repeat). Indeed we concluded that off-the-shelf application servers suffer from a whole host of operational and practical difficulties once you try to deploy them.

The solution

The approach we took finally was to do something which was not a traditional three tier system. A

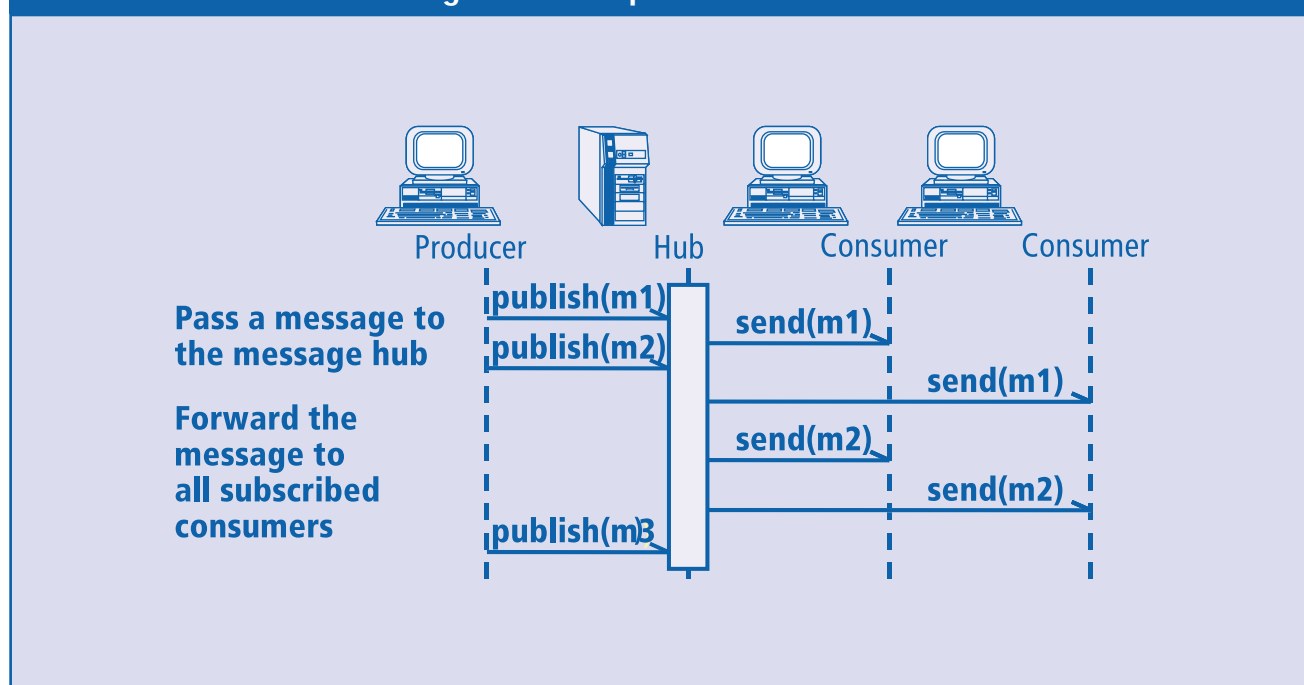
better way of describing it would be to call it a kind of ‘switched three tier system’. Rather than just synchronous requests to the back ends (and returning responses) we did something different.

Instead of having just Web servers attached to a database, we did introduce an application server tier in the middle (Figure 1.1). That application server tier is based on Java and runs on NT (we use the Microsoft JVM) but it does not rely on standard middleware such as EJB, CORBA, DCOM or anything like that.

In a sense we built our own application server: one reason was that I have a middleware background (I was at Transarc) and was familiar with the inadequacies of typical application server environments and the maintenance overheads that they impose. In some specific cases you can be better off to build your own (I would not recommend it for most cases). But, if you have a specialized environment with specific needs for high performance — and if the standard models and products do not work for you — then you may be obliged to build your own. This is what happened to us.

What we now have is purely asynchronous communication. We did not adopt the pseudo asynchronous approach (performing an asynchronous request and dropping it off at a server). What we use is IP multicast to give us asynchronous messag-

Figure 1.2: iBus publish and subscribe



ing between, for example, the request from our Web tier to our application tier.

To deliver this we deployed Softwired's iBus publish and subscribe messaging product to provide a reliable multicast solution for point (our clients' customers) to multi-point communication (the operators on our interaction engines). This was exactly what we were looking for.

Why did we choose iBus? As I mentioned above, I have a middleware background. As such I was familiar with the Java Messaging Service (JMS) and had used that API on previous projects. I had also worked on the IBM MQSeries Integrator initiative, as well as on its JMS API. From my knowledge I figured the JMS publish and subscribe did actually make sense for us.

It would, if we could find an implementation, give us the asynchronous communications to the back end systems and would act as the authoritative source of distributed state. Instead of having application servers pick up information out of the back end servers they could:

- receive asynchronous requests
- validate states
- raise connections back to the web servers.

These would be dropped off in in-memory databases on the Web server tier. So, with the Web

servers needing to pick up their state for the specific clients for specific sessions, each operator could pull whatever is needed directly out of local databases.

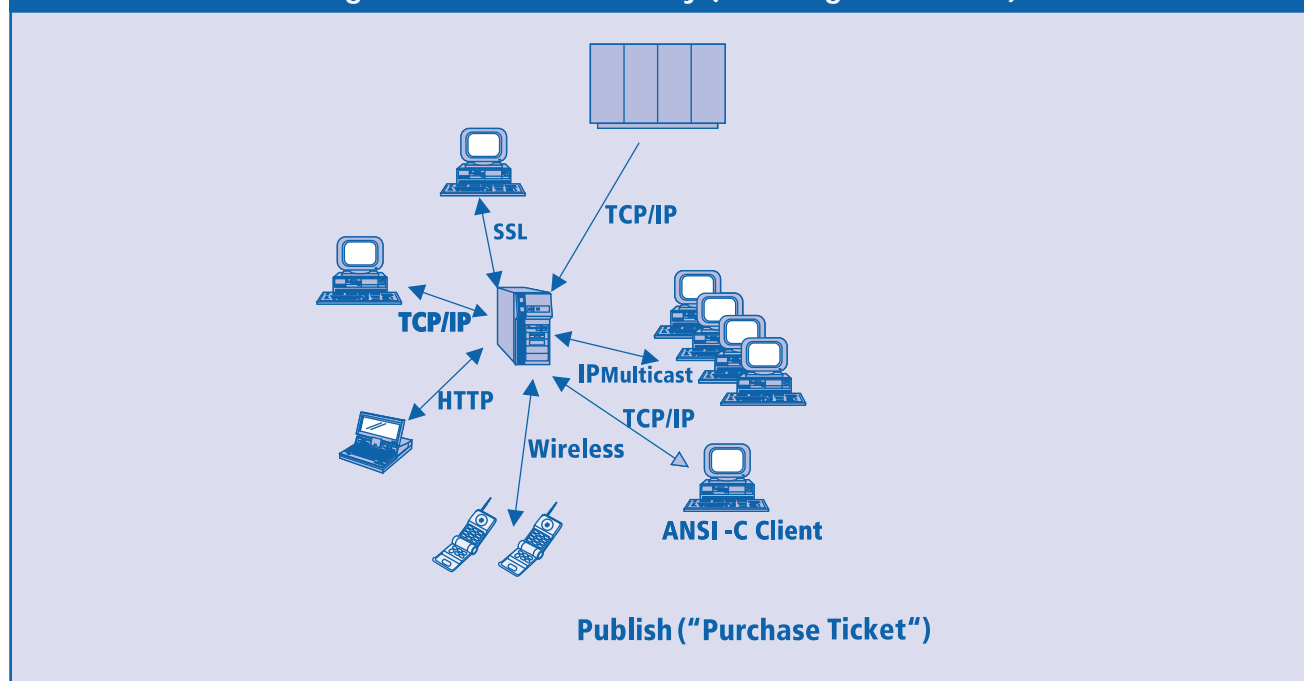
The advantage was that, rather than being exposed by having a single database, we could introduce a single database on each Web server. These Web servers — with their databases — are kept in sync., thereby providing the consistent image. Indeed, this was a second area where point to multi-point Java communications provided us with an answer. We could not use the traditional RPC synchronous request/response approach: it simply was not appropriate.

The only way we found to do this efficiently and scaleably was to re-use the multicast paradigm — but without actually using the multicast layer (because we would have had to build in the additional reliability which is really not our business). To avoid having to build that capability we exploited the iBus product.

iBus

In this context iBus (Figures 1.2-1.4) is a true, server-less, reliable, multicast product with a

Figure 1.3: iBus connectivity (including IP multicast)



standard API (JMS). In terms of communications it was the only offering that we could find. But it did bring four additional benefits to us:

- **it is, from our own evidence, genuinely scalable in that we can do this point to multi-point without having to endure multiple RPCs or anything like that**
- **it is extremely lightweight**
- **it can filter by content (and, by using a rules engine which Softwired purchased elsewhere, offers still more capabilities)**
- **it requires a minimum of configuration and administration (or, at most, just an entry in one config. file).**

The last has no immediate use for us. But, if we should ever build a packaged application that we could sell, we would not want to require customers to have to support software that itself required additional skills and/or a middleware administrator. Today, such people are too expensive — and tomorrow they will be even more expensive.

For routing we use the iBus channels. But we do not allocate channels to sessions. Trying to use JMS for thousands or millions of channels does not scale.

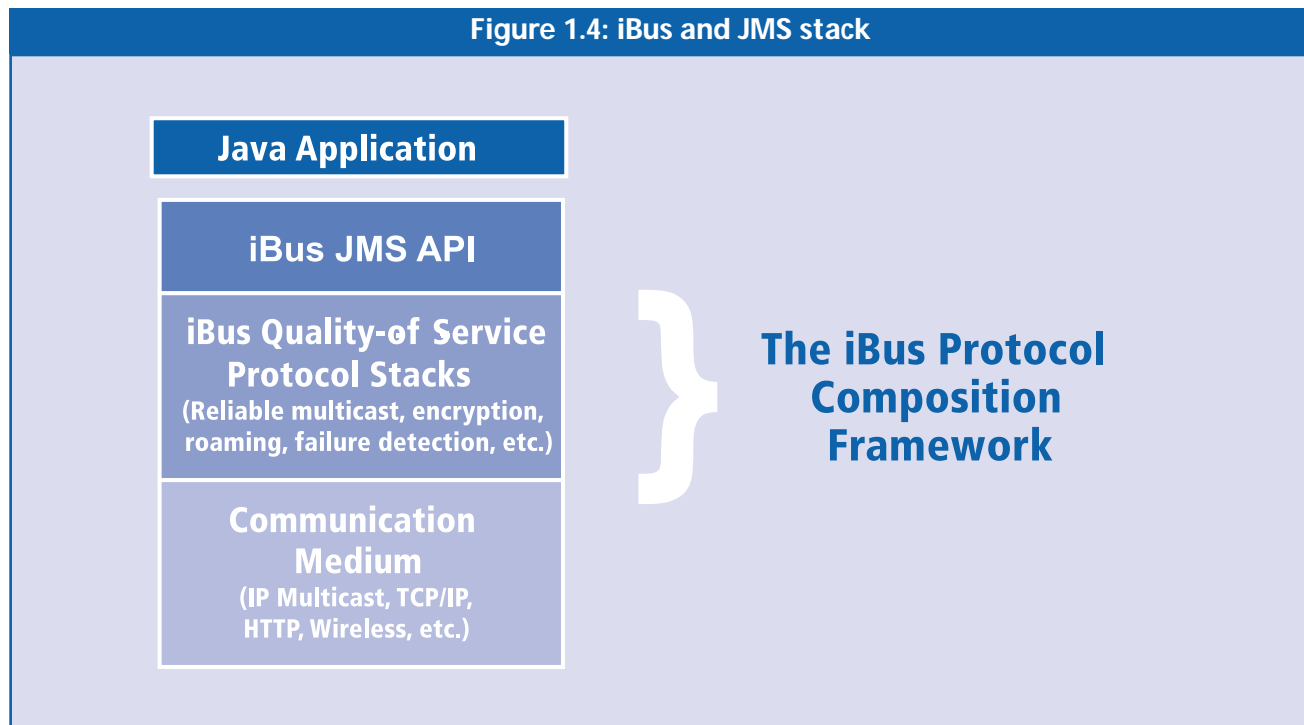
Instead we use the channels to identify different paths within the communications layers between the Web servers and the back end interaction servers, and we multiplex our communications over those channels.

That multiplexing even persists once it gets to the back end; it is just that it is switched off of keys. For example, we have ‘session ids’ and ‘call ids’ and request identifying markers for each one of our asynchronous requests. We can switch off these within the application; the application can extract its state based on those ‘ids’.

There was one final attraction of iBus to us, and this was unexpected and mind-blowing. In terms of reliability and replication iBus provided several facilities that do not exist (to our knowledge) in any other middleware product. iBus gives us the ability to do achieve modular space computing.

Let me illustrate. With iBus we can determine which servers are in a specific group of replicates. The group can split load and can do so dynamically. It does not require a lot of coding or periodic checks, pings to other servers, etc. This is a clean way of load splitting. It does not require a lot of intervention. It is not a manual fail-over process but is automatic. What has impressed us, to take one instance, is that we have been able to do on-line upgrades without taking the system off line.

Figure 1.4: iBus and JMS stack



Status

Looking back, what I have found interesting is that it is the only JMS solution I've seen so far that looks credible to me. We did look at other JMS offerings. We were not impressed — often these were merely loose interpretations of the JMS spec. (which is not that special). Instead, iBus is a straightforward implementation of JMS, and more.

What impressed me was why Softwired has done what it delivers in iBus. The CTO at Softwired — Silvano Maffei — spent a lot of time in the CORBA community, building ORBs and CORBA applications. He was pretty much sold on it as middleware for distributed architectures. But with hard-won experience, he realized that CORBA, however good theoretically, was:

- **just too heavy**
- **too expensive for what you get back**
- **insufficiently Internet friendly**
- **too dependant on its IDL**
- **not modern enough.**

Instead he built what he thought was really needed. Softwired took this over and fitted it underneath JMS.

Now JMS, as an API and as I commented earlier, is not that impressive. But it is a standard. What appealed to me was that we at LivePerson could have an innovative publish and subscribe mechanism underneath a standard API. But it is the messaging over IP multicast which delivers the real value. The JMS API offers me freedom to change, should I need something else.

Today we have this all up and working with live LivePerson customers. We have around 500-1000 active users, which is increasing all the time (you probably saw that Dell and NBC invested \$18M in us in February).

What is more, the application performance is barely registering. The iBus/multicast combination is proving excellent because it appears resilient to failures. We can start and re-start without any reconnections — because there are no connections. The system kind of heals itself at the communications layer.

I will go further still. iBus has proved to be the most stable component in our systems environment. It is the best middleware I have ever used

(and I have worked with middleware my entire career).

Lessons learned

My first lesson learned is a composite. You should expect low administration software with light-weight spontaneous communications (as your right). It is the way to go. At least within a LAN, spontaneous communications is the way to make progress (over a WAN, the issues are different because you need to worry more about security as well as communications reliability).

My second is that, if you require a full time administrator at the middleware layer, then you are probably doing something wrong. A variant on this, from a business perspective, is — make your middleware layer as light and invisible as practical; you should not need to know it exists. Any middleware that requires high administration overheads is difficult to justify.

My third is that, as real time becomes more prevalent, point to multi-point communications requires multicast or something similar. Traditional RPC ways — like that proffered by DCOM or CORBA or EJB — are not an effective solution for point to multi-point.

My fourth is low level. Synchronous RPC, especially over TCP/IP, does not scale well. My fifth is: use standard APIs wherever you go. Give yourself the flexibility to 'change out' your middleware products if you need to do this.

Management conclusion

As an ASP working over the Internet, LivePerson depends on being able to enable multiple 'client' organizations to support their many customers. By choosing to provide real time chat, and more, LivePerson made its technical challenges, particularly its middleware ones, even harder.

In this case study, Mr. Delafield shows what can be achieved with lateral thinking and the acquisition of the innovative iBus publish and subscribe messaging to deliver a 'switched three tier system'. He also shows how that alone does not address other issues like load balancing and single image availability. However, what is perhaps most striking are his comments about traditional middleware approaches and why they did not satisfy LivePerson's needs.

**Members of the
International Advisory Board**

Charles C.C. Brett
President, C3B Consulting Limited &
President, Spectrum Reports

William Donner
Chief Architect, Reuters

Kathryn Dzubeck
Executive Vice President,
Communications Network Architects, Inc.

Ellen M. Hancock
President
Exodus Communications, Inc.

Paul Hessinger
Vision UnlimiTed

Pierre Hessler
Deputy General Manager,
Cap Gemini

H. William Howard
Vice President, Inland Steel Industries, Inc.

Michael Killen
President, Killen & Associates, Inc.

Dale Kutnick
President, Meta Group, Inc.

Norris van den Berg
General Partner, JMI Equity Fund, LP

Fiona A. Winn
Managing Editor & Publisher
Spectrum Reports

Philip Manchester
Consulting Editor

**Additional contributors
include:**

Francis X. Dzubeck
Communications Network Architects, Inc.

Jay H. Lang
Distributed Computing Professionals

Keith Jones
IBM

David McGoveran
Alternative Technologies

Will. Capelli
Giga Group

Amy Wohl
Wohl Associates

Martin Healey
Technology Concepts Limited

Mark Allcock
J.P. Morgan Asset management

Aurel Kleinerman
MITEM

Chris Cotton
Consultant

Ian Hugo
Year 2000 Taskforce

Yefim Natis
Gartner Group

Rosemary Rock-Evans
Consultant

Beth Gold-Bernstein
Hurwitz Group

Tom Heywood
University of Southampton

Eric Leach
ELM

Glen Macko & John Parodi
Digital Equipment Corporation

Randy Rhodes & Troy Terrell
Black & Veatch

John Carter
IBM UK Laboratories

Roy Schulte
Gartner Group

Jim Johnson
Standish Group

Tom Curran
TC Management

Alfred Spector
IBM Corporation

Max Dolgicer
International Systems Group, Inc.

David Baer
Consultant

Ely Eshel
MINT Communication Systems

Ken Orr
The Ken Orr Institute

Peter Houston
Microsoft Corporation

Jeff Tash
Database Decisions

Ed Cobb
BEA Systems

Bernard Abramson
Merck & Co.

Mirion Bearman and Kerry Raymond
CRC for Distributed Systems Technology

Geoff. Norman
Xephon

Jim Gray
Microsoft Research

Jason Longo
PRL Scotland

Wayne Duquaine
Grandview DB/DC Systems

Steve Craggs
Candle Corporation

Colin White
DataBase Associates International

Gustavo Alonso
Swiss Federal Inst. of Technology

Peter Mark
Lotus Corporation

**MIDDLEWARESPECTRA is published
and distributed worldwide by:**

USA and Canada:
Spectrum Reports, Inc.

Subscription Center
PO Box 32510,
Fridley, MN 55432, USA
Telephone: 763 502 8819
Fax: 763 571 8292

UK and Rest of the World:
Spectrum Reports Limited

Research and Editorial Office
St Swithun's Gate, Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334

Subscription Centre
St Swithun's Gate
Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334

Email and Internet

Email:
spectrum@middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570