

An Expert Opinion on SoftWired iBus

Bert J. Dempsey

Assistant Professor

University of North Carolina at Chapel Hill, USA

<http://www.ils.unc.edu/~bert/>

phone: +1 919.962.8066

email: [dempsey@cs.unc.edu](mailto:dempsey@cs.unc.edu)

January 31, 2000

## **Background**

To set the context for iBus, I will briefly define messaging and overview its importance today. Trends in software development point towards development of functional components (objects) that can be configured to communicate across networks in order to implement distributed applications. Many frameworks exist today for object groups of some flavor, but Java-based objects will play an important (if not dominant) role going forward. Java offers widely accepted platform-independent language with strong object-oriented features and enormous industry momentum in refining and extending the Java environment. Accordingly, in order to facilitate a standardized method of communication between Java-based components in the most general case of components running on different machines, the Java Messaging Service (JMS) API was developed by Sun in cooperation with a number of major industry participants. SoftWired and several other companies have implemented (and enhanced) the API to create messaging middleware for Java developers. Given the list of industry sponsors endorsing JMS, I believe it is well positioned to be the de-facto standard for distributed Java-based applications. (Its incorporation into Sun's Java 2 Platform, Enterprise Edition (J2EE), for example, is a good sign.) Legacy component migration to JMS will be available through bridging products such as the set of adapters provided by SoftWired (e.g., iBus//Ansi-C).

The JMS API adopts a publish/subscribe model for its communication abstraction. (iBus//MessageBus v3.0, rightly I think, implements only this model and not the less-general

point-to-point model also supported in JMS.) Publish/subscribe means that software components using JMS can subscribe and publish to “logical channels” (topics), instead of performing some sort of rendez-vous with a communication partner. The point is that publishing/subscribing to topics is inherently a highly scalable and flexible form of asynchronous communication for multiple users, not just two-party (e.g., client-server) communication. This communication model then fits the needs of distributed components with group communication requirements. Furthermore, iBus’s support for reliable multicast delivery leverages network-level group communication support (IP multicast) to enhance the scalability of their implementation of this publish/subscribe model (see below on Reliable Multicast as a Message Delivery Option).

### **Key Features of the iBus Software**

My perspective on SoftWired’s iBus products derives from my 10-year career as an academic researcher in computer networking, during which I have had extensive experience with the development of distributed applications within a research environment (most recently, within the Internet2 Distributed Storage Infrastructure (I2-DSI) project, <http://dsi.internet2.edu/>) and regular contact with commercial systems developers. My hands-on experience with iBus involves iBus//MessageBus (version 3.0.1) running in both WindowsNT and Solaris environments. I found the documentation distributed with iBus//MessageBus to be informative and quite adequate for a Java-aware programmer to install, test, and begin development with the product. All functionality that I tested worked as documented. Since I do not have hands-on experience with all iBus products, the iBus-specific comments in this document should be taken as extrapolations from my experiences with iBus//MessageBus.

iBus layers on additional programming abstractions above the Java Messaging Service (JMS) API that it implements, providing significant value-added functionality to the vanilla JMS service. In my view the key additional functions are in the way that (1) iBus aids programmers in customizing the messaging service to the requirements of an application through a *Quality-of-Service Framework* and (2) its integrated *reliable multicasting* transport option for message delivery. I discuss the importance of these distinguishing features in detail below.

### *Quality-of-Service Framework*

The most compelling element of the iBus design is the development of a very flexible “quality-of-service” (QOS) framework over JMS. This feature allows developers to choose from a set of modular protocol objects to construct a message service with the desired characteristics. I like this modular design; it fits the well-known principle of developing “protocol stacks” with layered functionality such that different combinations of layers can be used to construct different services. Flexibility is important since, while certain reliable services are clearly needed, the application-specific nature of messaging solutions means that application developers will sometimes want the flexibility to create customized message delivery services, i.e., to mix and match iBus protocol objects for different types of QOS. These objects include high-layer functionality like encryption and compression along with lower-layer transport mechanisms like reliable multicast and TCP. For “power users”, there is the option of developing a custom protocol object and using it with existing protocol objects. This design then makes iBus readily extensible for application-specific communication needs, and the iBus architects have provided a rich mix of built-in protocol objects that covers most of the common ground.

Experience has shown that modular protocol design makes complex communications easier to implement and maintain since implementors and testers can isolate functionality to relatively simple components. Indeed the iBus distribution includes test modules for use in developing either new protocol objects or applications. Given the complexity of distributed programming, the value of a modular design should not be underestimated--- as a developer, I would be inclined to have more faith in iBus software because the granularity of the protocol objects is such that the functionality of these objects can be readily verified.

As a final note on the QOS framework, I also like the way in which topics-with-QOS (iBus channels) can be expressed in a URL-like syntax for convenient naming. This is a small, but elegant, feature that leads to self-documenting code and documentation.

### ***Reliable Multicast as a Message Delivery Option***

A second major strength of iBus is its implementation of a reliable multicast transport solution. I have a long personal history of working with multicast: my first technical paper on the subject was in 1990<sup>1</sup>, and I have been working in this area and watching multicast technologies evolve and mature ever since<sup>2</sup>. Network-level (IP) multicasting supports efficient “packet-splitting” in the network such that a copy of a multicast packet is delivered to all endsystems participating in a multicast group address. Reliable multicast (RM) delivery over IP multicast can be added by endsystem protocols. Many proposals for reliable multicast are available, but no standards exist in this area. (iBus implements its version of one proposed RM protocol.) If a communication solution is to be fully de-centralized (e.g., no message server), RM protocols are essential for scalable reliable group communication since the alternative is to manage an N-by-N mesh of TCP point-to-point connections between N participants. Also, the use of network-level multicasting implies that a sender need only transmit one copy of a message since the network then delivers copies to all parties. Under a TCP-based solution, one copy of the message must be transmitted at the sender for each receiver, which is a performance penalty that increases with the number of receivers.

While not generally available in wide-area Internet paths, IP multicast can be readily deployed in enterprise-wide networks today. That is, all major operating systems support IP multicast in endsystems, and multicast routing support in network devices is mature and commercially available. Thus, leveraging a RM solution is a powerful feature for iBus today as the the scalability and flexibility of RM delivery can not be matched by point-to-point solutions. Moreover, the iBus software provides a complete RM solution with explicit group membership management and other features (e.g., deterministic addressing maps) that make their flavor of RM flexible and extensible for the developer. The RM delivery option is a powerful tool for building scalable applications today within the many enterprises that have enabled IP multicasting in their networks, and it positions the iBus products to leverage the ever-growing base of intranetworks and wide-area paths with IP multicasting support.

---

<sup>1</sup> Dempsey, Fenton, and Weaver. The Multidriver: A Reliable Multicast Service using the Xpress Transfer Protocol, 15<sup>th</sup> IEEE Local Computer Networks Conference, September 1990, pp. 351--359.

<sup>2</sup> MESH-R: Large-Scale, Reliable Multicast Transport, M. Lucas, B. Dempsey, A. Weaver, IEEE International Conference on Communication (ICC '99), Vancouver, BC, June 1999, pp. 657--665.

### **Note on Competing JMS Products**

In my examination of the claims of competing vendors, I find no ready evidence that iBus is not a best-of-breed JMS implementation. The feature lists touted by other vendors are similar to those of iBus, and, as discussed above, iBus maintains a key advantage with its QOS framework and its powerful RM capability. A number of competitors have more monolithic software designs where the pure JMS functionality is closely aligned (entangled?) with larger software solutions. A minor deficiency with Softwired's iBus is that it does not have a GUI-driven administration tool (only command-line) for configuring the distributed components of a middleware application. Due to the complexity of distributed environments, a GUI aid is warranted, and some competitors have this feature.

Another important issue for which I offer no evaluation in this document is the issue of performance. Let me state clearly here that I have no data on this issue--I have not undertaken any performance evaluation experiments with iBus or competing products. I merely raise here the point that performance (e.g., as measured in message latency or message throughput normalized to some hardware/software platform) will be a crucial issue for some near real-time or high-throughput messaging applications. Performance comparisons are notoriously difficult to achieve in an "oranges-to-oranges" manner, but iBus developers quite likely have internal data that will shed light on how much overhead the iBus middleware induces. Again, iBus is well-positioned with its rich reliable multicasting option since reliable multicasting generally outperforms multiple TCP connections and the advantage increases with the size of the receiver set.

### **Driving Applications for iBus**

The general trend driving the need for iBus functionality is the increasing number of applications that require distributed implementation for reasons of performance, fault tolerance, or just near real-time interplay between logic and resources located on physically separate machines. It is clear that Java-based solutions are going to play a very important role in such distributed application development. For example, Sun's recent announcement of Java 2 Platform, Enterprise Edition is another step in solidifying different pieces of Java technology into a coherent framework that is readily usable for developers of enterprise-wide solutions. J2EE is a very positive step, I think, for

Java in this enterprise-class solutions market and, as a central component of J2EE, JMS implementations will benefit.

Many such application scenarios are evident today and doubtlessly new ones will emerge in time. I will discuss some key areas below.

### **Messaging and the WWW**

Given the ubiquity of Java-in-the-browser, JMS solutions like iBus have an opportunity to enable new data dissemination and collaboration applications through the universal WWW interface. Experience has certainly shown that communication (e.g., email, news, and chat services) is a tremendously powerful driver behind Internet use. JMS offers the flexibility of peer-to-peer communication using publish/subscribe messaging in order to enable large numbers of ad-hoc user groups communicating with minimal server support, which should improve the scalability of collaboration support. In addition, where multicasting has natural support in the network, data-oriented push services can use publish/subscribe dissemination of customized information channels to WWW clients. Satellite downlinks and other broadcast media (e.g., my local TV station is now experimenting with using spare capacity in its digital broadcasting spectrum for data-casting) offer the possibility of real-time dissemination of, for example, a traffic report service. Users could tune an applet in a WWW browser to subscribe to a few of the (very fine-grained) channels of traffic information emanating from the central source. A key point here is that constructing this type of service using only the client-server WWW of today would be difficult at best and most likely infeasible, given the constraints of point-to-point client-server models.

### **Enterprise Portals**

Back-end e-commerce applications, by which I mean business logic related to the processing of either consumer-to-business or business-to-business transactions, are poised for spectacular gains in the coming years. A recent estimate predicted a 10-fold increase in e-commerce from 1999 to 2004 with 85% of transactions in a business-to-business setting. Current systems must continue to expand in order to keep pace with new functionality and transaction volume. It is this latter emphasis on scaling up to higher throughput (and fault-tolerant) systems that will drive distributed implementations, which in turn point to the need for JMS products like iBus.

These trends should also be viewed in light of the explosive power of the emerging eXtensible Markup Language (XML) technologies to create entirely new e-business applications. XML promises (and will) create a revolution in the sharing of structured data. On-line transactions, legacy company documents, and---most importantly---the vast amount of business data stored in back-end databases can now be easily encapsulated for remote processing using controlled vocabularies under XML. This emerging area of new forms of electronic interaction between data-rich systems within and across departmental and organizational boundaries has been called the enterprise portal by some (see <http://www.iona-iportal.com/>). The benefits of enterprise portals will be enormous and, with the acceleration offered by XML encapsulation, lead to an explosion of new business services. Many of these applications will be tied to near real-time distribution of transactional information. This scenario favors distributed Java implementations that will turn to best-of-breed JMS solutions.

### **Network Intelligence and its Management**

Another driver for multi-platform, robust, scalable middleware will be in the implementation of network-based services. The best mainstream example of the importance and growing sophistication of network services is caching and replication. Efficient use of network bandwidth and effective delivery of time-sensitive media to clients at the edge of the network has led to the wide use of cooperating network caches in the current Internet, primarily for support of replaying WWW server responses. However, WWW caching is now being generalized to other forms of data replication, including data staging in dedicated replication servers, and the explosive growth of companies like Inktomi, Akamai, and Digital Island (as well as corporate interest in research projects like my Internet2 Distributed Storage project) point to the power of this market. Future drivers for these services include support for mobile and intermittently connected wireless Internet clients, support for near real-time collaborative tools between clients, and ever-greater emphasis on rich digital media delivery as backbone bandwidth and storage capacity continue to grow exponentially. The value of scalable, multi-platform development solutions for building the middleware to implement and manage high-layer network-based services across cooperating clusters of network devices will ramp up in the near-term future of the Internet.

## **Wireless Clients**

As a strong platform-neutral programming standard, Java will be a key software platform for wireless Internet devices. Messaging communications will be used for some local peer-to-peer applications between co-located wireless devices, but, I believe, the larger application area will be automated sensing and communication between mobile clients and the local network servers or smart network caches on the fixed-wire Internet. Intelligent network support for mobile clients will be a major driver of the network intelligence area (discussed above) because mobile clients have limited storage and computing resources and thus benefit greatly from off-device support. Key reasons that JMS will be an attractive solution in these applications include (1) the likelihood that adoption of Java standards for mobile clients will be widespread, (2) the flexibility of message-based communication over pure client-server paradigms for automatic connection and notification services, and (3) the natural multicast medium of wireless broadcast that will facilitate adoption and use of multicast-based message delivery.