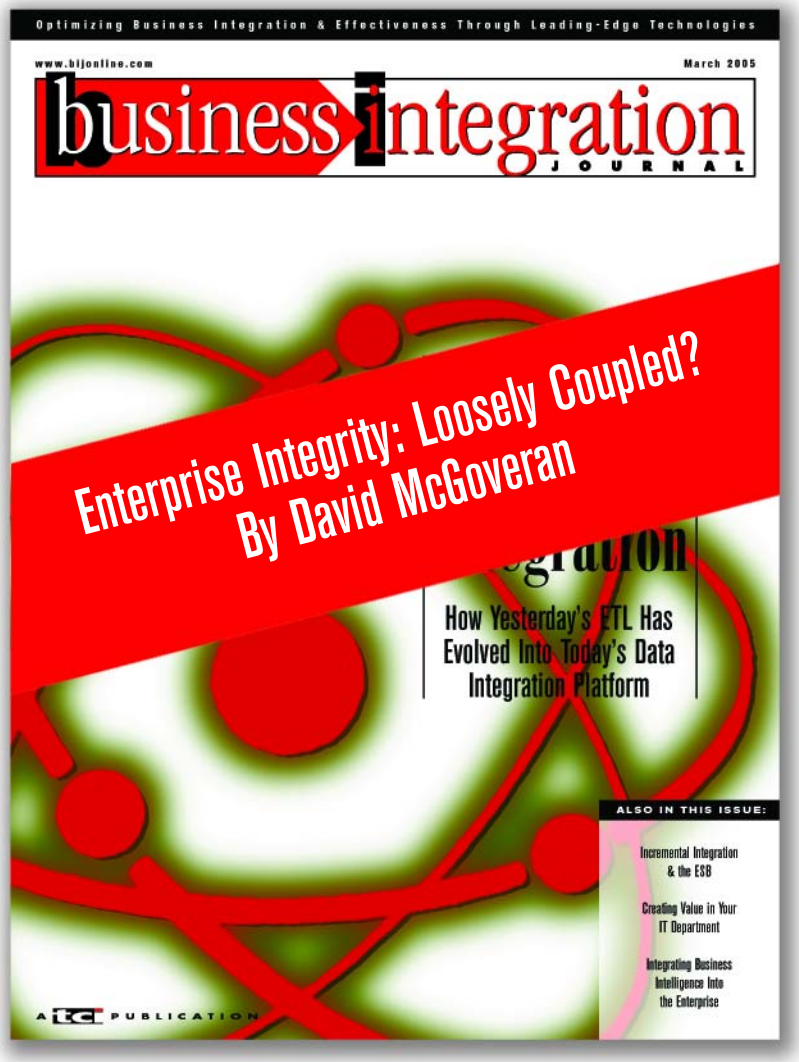
This article appeared in the
March 2005 issue of

business integration JOURNAL



Subscribe instantly at
www.bijonline.com

- Free in the U.S.
- $48 per year in Canada and Mexico
- $96 per year everywhere else

ENTERPRISE INTEGRITY

Loosely Coupled?

BY DAVID McGOVERAN

These days Service-Oriented Architecture (SOA) is presented as the solution to everything. This is especially true of the Web Services (i.e., an implementation of a Web-accessible service compliant with Web Services standards) SOA implementation. Far too many vendors purport that support of standard interfaces such as Web Services is sufficient for an SOA to be loosely coupled. This is mostly marketing hype from those with a vested interest. In fact, it's just one example of a malignant disease that has been invading IT surreptitiously over the last two decades—a kind of wolf in sheep's clothing.

I'm a big fan of producer-consumer software models, service encapsulation, and SOA incorporating these. In fact, this is largely what was intended by client/server architecture before technically incompetent press and analysts joined a marketing bandwagon of incapable vendors to distort and redefine it to mean a simple two-tier hardware architecture or, worse, a variant of PC network computing. We learned quickly that standard interfaces were necessary for maintenance, and that dynamic service interrogation was necessary for robustness in the face of changing requirements and implementations. We also learned, at great expense, that these interface standards were insufficient.

Early SOAs required tremendous discipline to build scalable systems. Few designers or developers, and even fewer vendors, had a clue as to how to do this for distributed applications. You certainly weren't forced to build scalable, highly available distributed applications by either tools or the standards. Invocation was by remote procedure calls, meaning that both consumer and producer maintain dedicated application-level threads, with the consumer blocking (i.e., being synchronized) from request until response. Of course, most early applications were single-threaded, so the idea of asynchronous activity by the client application was a non-starter.

Highly scalable and available distributed applications must avoid synchronization. To the degree business requirements permit, every service request is an atomic message rather than conversational; the request message is sent and consumed asynchronously, the service produces asynchronously with consumer activity and other service activities, client processes are stateless between request and response, server state preservation between requests is minimized and, if necessary, state is persisted. Inter-service coupling through shared resources is the most difficult synchronization to minimize, with client-service coupling being almost as bad. When a service can be composed of other services (think composite applications), avoiding coupling becomes even more critical. Notice that such an SOA is inherently event-driven and loosely coupled with respect to service interaction. To be highly available, the architecture also must be loosely coupled with respect to service instantiation (to enable failover and load balancing) and change (to avoid downtime when services are upgraded).

Now reconsider the claim that loosely coupled just means service interfaces are based on standards. Wouldn't it be nice if all you had to do was buy standards-based products to get something as complex as scalability or high availability? You know, as in "we support Web Services so our product is loosely coupled." The truth, however, is that Web Services' self-describing interfaces and discoverable services don't provide the operational benefits of loose coupling. These aspects of a loosely coupled architecture depend on the details of how Web Services are used, and on whether or not both asynchronous interaction and true messaging transports are supported. Unfortunately, implementing asynchronous interaction with Web Services requires considerable care. It currently requires SOAP over JMS using only one-way request messaging. Response handling on the client is less well-defined, easily raising what should be a middleware service to the level of application code.

This brings us back to the main point: You should ask why it's so advantageous for some to define loosely coupled as meaning standards adherence, conveniently glossing over the real implementation issues that induce strong coupling. Interface standards can't make architectures successful, let alone suffice to make an architecture loosely coupled. This is just one example of the danger of adopting standards for standards' sake. Standards are great time savers and the right standards choices can enable functionality, but too often we've fallen prey to the persuasion of standardization over business requirements and technical appropriateness. Standards that are misrepresented aren't worth the biased committees that develop them. Adherence to any inappropriate standard, or to a bad standard, will consume enterprise resources as quickly as the fabled wolf that ate all the sheep. Your *enterprise integrity* depends on rejecting these "sheep" that don't bear wool. **bij**

About the Author

David McGoveran is president of Alternative Technologies. He has more than 25 years of experience with mission-critical applications and has authored numerous technical articles on application integration.
e-Mail: mcgoveran@bijonline.com
Website: www.alternativetech.com