Foreword

I remain convinced that service-oriented architecture (SOA) is at least a "next big thing." SOA is about delivering on the promise of best-of-breed IT by unleashing the *shared* business services that are currently bound up in monolithic and often isolated applications.

In my experience, there is nothing that does more to protect IT investment long-term than SOA: With SOA

- You can more closely align IT resources with their business functions.
- You can more easily deliver composite applications that provide unified, task-oriented views across the business (such as via automated business-process management or workflow).
- You can achieve greater application life-cycle flexibility by incrementally managing requirements and change.
- You can more easily deliver real-time business intelligence on the aggregate information flowing through your applications.

SOA can empower you to deliver more dynamic and cost-effective business systems by enabling an optimal "mix and match" across the spectrum of "buy" versus "build" and insourcing versus outsourcing.

One of BEA's competitors is fond of saying best of breed only works in dog shows. I find that view both depressing and somewhat offensive, but it is reflective of the reality that best of breed is much harder to achieve today than it ought to be. As an industry, we must make best-of-breed IT a reality. We do not have much choice:

- Applications are proliferating.
- Applications are becoming ever more specialized.
- Applications are being aggregated and customized to meet user demands.
- Applications are being outsourced and offshored.
- The pace of application change is accelerating.

SOA via the Web application platform is just in time!

SOA is founded on modular programming practice, and as such has been part of the vision for distributed computing with each iteration—from DCE to DCOM to CORBA to MQSeries to RDBMS stored procedures. But we as an industry have never quite gotten there in terms of practicing SOA at scale.

Why is the application integration that underlies SOA so hard? We developers have been quite successful at modular re-use within applications (procedures, objects, components, events …). However, a look at the history of distributed computing suggests that we have done much less well at *inter*-application re-use. Each of the aforementioned architectures was at one time envisioned as being the universal SOA fabric (or bus) for interconnecting all applications. And each came up short. Why?

Application integration is simply a hard problem—if an application is code that is developed and deployed as a unit, then we can easily recognize that

- Applications are developed by different teams,
- Who work in different places,
- On different schedules, and
- Who don't talk to each other enough.

Most critically, there generally *is* no shared vision of precisely how a *customer* or *order processing* should be represented. So each team builds their own and hardwires it into their code. Exporting such "local" object models is inherently fragile.

My hope for XML and web services is that we can enshrine SOA once and for all within our application architectures. By defining or surfacing web-standard "service access points" for individual business operations, XML and web services will become the universal SOA fabric, replacing, over time, the existing *ad hoc* mix of proprietary and partially standard technologies for interconnecting applications.

One reason XML and web services have gained greater momentum than their distributed computing predecessors is that they have gained nearly universal industry support. Web services provide communications infrastructure for both synchronous (request/response) operations and asynchronous (message-oriented) operations within a single, unified framework. Web services is the first to meet integration and scalability challenges across enterprise application integration (EAI) and business-to-business interactions (B2B). Preceding technologies have never come close to achieving the critical mass required for a transitive closure of participants (that is, working with all of an enterprise's own business systems, the business systems of their partners, of their partners' partners, and so on).

Understanding how the different aspects of web services come together and support SOA in real-word situations and thus fulfill business requirements are critical for developers and architects. One of the values of this book is that it showcases a live implementation of SOA architecture at Hewlett-Packard using web services on BEA's WebLogic Platform. It goes through the lifecycle of web services by tracing this case study from the business requirements through to architecture, design, development, testing and management.

This book details out how the WebLogic platform is leading the effort in providing one end-to-end integration tool for developers to do Java/J2EE/web services development and integration with business processes and backend systems. The book is reflective of start-of-the-art innovations such as using Javadoc annotations for capturing metadata, transformations, business processes through BPELJ, and web services through Java web services (JWS), which are collectively being standardized through the JCP, OASIS, and W3C.

Those wishing to systematically transform their organizations with SOA based on XML and web services—specifically architects, developers, and technical project managers—will find this book useful to guide them through the lifecycle of a web service and help solve issues and problems. In all of my conversations with enterprise IT personel, implementation of SOA architecture with web services (especially the security, performance and complexity of their EAI/B2B requirements) reigns among their top concerns. This book makes it quite a bit easier to comprehend all the facets of web services. It's a reference book that architects, developers, and technical managers may well want to keep handy.

Scott Dietzen, Ph.D.

Chief Technology Officer BEA Systems