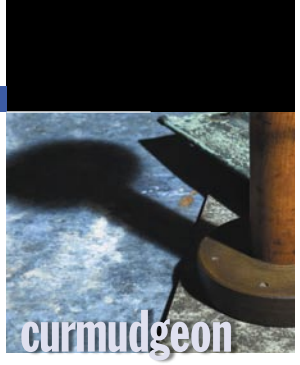

DOA with SOA

Alex Bell, The Boeing Company

Adopting THIS ARCHITECTURAL STYLE IS NO CURE-ALL

It looks like today is finally the day that we all knew was coming—it was only a matter of time. An ambulance has just pulled up to haul away Marty the Software Manager after his boss pummeled him for failing to deliver on promises of money savings, improved software reuse, and reduced time to market that had been virtually guaranteed merely by adopting SOA (service-oriented architecture). Everything could have been so different for Marty. If only there had been a red-hot market for a software application that fetched the price of London gold, converted the price from pounds to dollars, calculated the shipping costs for the desired quantity, and then returned a random verse from the King James Bible. As opposed to the currently unfolding scenario involving an ambulance, Marty's mental vision was one of a Brinks truck speeding to the scene to empty coffers buckling under the strain of overflowing cash.

Should anyone really be surprised? After all, Marty is probably still sporting a hook in his mouth from having been reeled in by Victor the Vendor's SOA fishing pole. The hype and propaganda sprinkled onto the bait that Marty swallowed must have caused a mind-numbing sense of euphoria that resulted in business and technical justification for his decisions being sloughed off as mere annoyances. Despite his headlong charge into the SOA arena, Marty would have had a difficult time describing SOA the same way to three different people.

In Marty's defense, however, many people have different ideas about what SOA is and is not. Thankfully, I have the benefit of a 13-year-old daughter in the household, so there is no shortage of expert opinion on any topic. I asked her what she thought was meant by service-oriented architecture. She told me that this was an approach used for constructing the buildings where she buys, among other things, her Hollister and American Eagle clothing. There are certainly different opinions about what SOA might be, but this one might be a bit extreme.

Some projects might say they are dancing the SOA tango merely by using XML, WSDL, SOAP, and UDDI technologies. Others may believe they are saluting the SOA flagpole if they are using OOD and their classes are stateless. In actuality, SOA describes an architectural style that is independent of using a particular technology. This architectural style involves advertisement of services in some form of a registry that clients can use to introspect, discover, hook up to, and invoke services of their choosing. The properties associated with these services are described by SLAs (service-level agreements), which might be measured in terms of processing time, number of messages per minute, and number of rejected transactions. SOA is enabled by technologies such as those mentioned earlier, as well as others such as CORBA and DCOM, which have been around much longer.

For many software organizations, the primary dilemma is not deciding whether or not SOA is appropriate to suit their development objectives, but instead determining which technology they should use to enable SOA and which implementation tactics best suit their needs. Not all SOA users or prospective users even realize that they have some options when it comes to *how* they should use SOA to develop their products. The SOA lemmings who fail to consider the usage tactics at their disposal have the potential for actually causing negative impacts to their software architectures as opposed to capitalizing on some of the benefits that SOA truly does offer.

I wonder which straw finally broke Marty's back. In an effort to be good SOA practitioners, did Marty's software staff generalize some of their services to such an extent that they were not able to meet even their own product's needs? Specifically, the idea of developing distributed infix operations where users input a numerical radix upon which service requests should be based sounded good, but the performance impacts of remote process invocations simply to add and subtract numbers might have been a bit of an SOA stretch. Even though the infix operations were written with the hallmark SOA qualities of being discoverable, stateless, composable, and not dependent on any other services, an SOA style must be selectively applied and used only where appropriate to do so.

Continued on page 54

Continued from page 56

Perhaps Marty's misfortune was an after-effect of firing all of his systems engineers because of a belief that adoption of SOA transformed the traditional software life cycle into one where the only relevant activities were development and integration. There is a lot of money to be saved by expecting jack-in-the-box system architectures just to pop up amidst a collection of services as opposed to investing time and effort on traditional engineering activities. After all, in the event that performance or usability issues arise as a result of the absence of systems engineering activities, the mitigation tactics are simply to discover and hook up to new services with better SLAs!

There might be yet another possibility at the root of Marty's looming ambulance ride: Did his staff choose the wrong level of abstraction with which to implement SOA? As opposed to service users hooking into services directly at the "stub" level, there are often circumstances where a service layer encapsulating such stubs has the potential of improving important properties for those service users, including performance, availability, and survivability. Specifically, performance can be improved by short-circuiting remote method invocations in the event that requested information has been previously fetched. Availability can be improved by the service layer hooking up to alternate service providers in the case of failures or SLA violations. Survivability can be improved by providing service users with some fidelity of reply even if connectivity to the actual service provider is temporarily unavailable.

As previously mentioned, the benefits of encapsulation should not be ignored when selecting the tactics with which to best implement SOA. In fact, even in the context of my daughter's rather, ahem, interesting definition of SOA, she recognizes the value of encapsulation: "Dad, I can wear and enjoy my clothes without having to know any of the details of how they were made." Perhaps your SOA tactics should heed these wise words and similarly hide applicable implementation details from service users.

Why should a user of an extremely simple service be bothered with having to know about UDDI, for example, when all that is needed is the monthly payment for a given principal, periodic interest rate, and duration of loan? A preferred implementation may be to hide UDDI from users of such services beneath a service layer.

Unless a project is merely gluing together simple services such as those that involve fetching stock prices, querying the weather of a specific lat/long, or requesting the zip code(s) for a particular city, adopting SOA should not generally change a traditional software life cycle. Sure, some development activities might be shortened as the result of reusing certain components which can be purchased, but how often is one able to actually buy preexisting components that provide a product's core "business logic"? On the flip side of the reuse coin, how many software managers are actually willing to adjust their development schedules, investing time to find out how a particular component they imminently require can be generalized or specialized to meet the needs of unknown or future users? Meeting software reuse goals to any significant extent, such as those supporting product-line aspirations, generally requires engineering and planning as opposed to simply happening as the result of adopting SOA.

Sadly, Marty did not make it to the hospital. Whichever combination of misguided business decisions and implementation tactics may have finally led to his demise, Marty was DOA, just like his SOA project. It did not have to be this way. Adoption of SOA did not constitute authorization for Marty to ignore best practices or to show contempt for common sense. How about you? How is your SOA health? Do you presume that SOA can be enabled only by Web services? Do you believe that the benefit of properties such as encapsulation and abstraction are important only in "old-fashioned" architectural approaches? Has implementing your definition of SOA resulted in elimination of any major engineering activities? Pay close attention to how you answer. You will not want to miss the warning signs of potentially being DOA with SOA. Q

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

ALEX BELL is a software architect with The Boeing Company. He has written several pieces for *ACM Queue* including "Death by UML Fever" (March 2004) and "Software Development Amidst the Whiz of Silver Bullets" (June 2006).