asynchronously. We wanted to be able to upgrade a service without affecting the clients that use it. Unfortunately, the unending and ever-changing technology stack associated with this goal has confused people and not solved the problems we face in real architectures for real organizations. Our goal in this new vision is not simply to be different, but to add value and improve the status of the Service-Oriented Aggravation we have seen.

We have a collection of technologies that comprises our basic understanding of Web Services: SOAP for service invocation, WSDL for contract description, and UDDI for service metadata publishing and discovery. SOAP grew out of different traditions including the remote procedure call (RPC) model and the asynchronous XML messaging model (doc/lit). The first approach is brittle, does not scale, and did not really work out all that well under its previous names of DCOM, RMI, and CORBA. The problems are neither caused nor solved by angle brackets; we simply tend to build systems in this manner at the wrong level of granularity and prematurely bind ourselves to a contract that clearly will not remain static. The second advances the art and is a fine implementation strategy, but does not quite live up to the interoperability hype that has hounded it from the beginning. It complicates even simple interactions because its processes are influenced by the goal of solving larger interaction problems.

The doc/lit style allows us to define a request in a structured package that can be forwarded on, amended, processed, and reprocessed by the loosely coupled participants in a workflow. Like an itinerant pearl, this message accretes elements and attributes as it is handled by intermediaries and endpoints in a potentially asynchronous style. We achieve horizontal scalability by throwing ever more message handlers at a tier. We can standardize interaction styles across partner and industry boundaries and business processes that cannot be contained by a single context. It represents a decontextualized request capable of solving very difficult interaction patterns.

When strict service decomposition and description alone (i.e., SOAP and WSDL) proved insufficient to solving our interaction needs, we moved up the stack and introduced new business processing and orchestration layers. A proliferation of standards and tools has thus complicated an already untenable situation. When we cross domain and organizational boundaries we run into conflicting terms, business rules, access policies, and a very real Tower of WS-Babel. Even if we commit to this vision, we have no real migration strategies and have fundamentally been lied to about the potential for interoperability. Clay Shirky has famously categorized Web Services interoperability as "turtles all the way up."[†]

The problem is, when most people want to invoke reusable functionality in a language- and platform-independent way, these technologies are overkill, they are too complicated, and they leak implementation details. In order to invoke this functionality, you have to speak SOAP. That is a fine implementation choice, but in this world of loosely coupled systems, we do not always like to advertise or require our clients to know these details for simple interaction styles.

† *http://en.wikipedia.org/wiki/Turtles_all_the_way_down*