Conclusion

The resource-oriented architecture approach walks a series of fine lines. On the one hand, to initiates of convention, the approaches might seem a little strange and untried. People concerned about their resumes want to stick with tried and true approaches. On the other hand, to those who have studied the Web and its fundamental building blocks, it makes perfect sense and represents the largest, most successful network software architecture ever imagined and implemented. In one light, it requires a radically different way of thinking. In another light, it allows a powerful mechanism for wrapping and reusing existing code, services, and infrastructure with logically named interfaces that do not leak implementation details for many forms of interaction. We have the freedom to be resilient in what we accept on the server without breaking existing clients. We can support new structural forms for the same data over time. We are able to migrate backend implementations without necessarily affecting our clients. Additionally, important properties such as scalability, caching, information-driven access control, and low-ceremony regulatory compliance fall out of these design choices.

Software developers do not usually care about data; they care about algorithms, objects, services, and other constructs such as this. We have some fairly specific recommended blueprints and technologies for our J2EE, .NET, and SOAP-based architectures. Unfortunately, most of these blueprints ignore information as a first-class citizen. They tie us into specific bindings that make it hard to make changes without breaking existing clients. This is the flux treadmill we have been on for years, and the business units are tired of paying for it. Web Services were supposed to be an exit strategy, but inappropriate levels of abstraction and overly complicated edge-case use cases have made the whole process an entirely WS-Unsatisfying experience. It is time to take a step away from software-centric architectures and start to focus on information and how it flows. We will still write our software using the tools we know and purport to love; it will just not be the focus of our architectural bindings.

The resource-oriented approach offers compelling bridges between business units and the technology departments that support them. There are real efficiencies and business value propositions offered by an information-centric view on how our systems are connected. Rather than starting from scratch with each new Big Idea from our vendors, we can learn valuable lessons from the Web on how its architectural style elicits important properties. Architecture is inhabited sculpture; we are forced to endure the choices that we make for quite some time. We should take the opportunity to imbue our architecture with functionality, beauty, and a resilience to change to make our time in it more useful and pleasant.