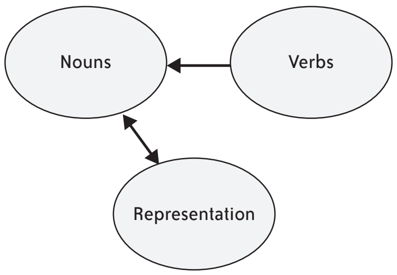

FIGURE 5-3. REST separation of concerns

This separation contrasts sharply with the contractual nature of a SOAP service invocation, where the structure of the request, the behavior being invoked, and the form of the return type are often bound to a contract through the Web Services Definition Language (WSDL). Contracts are not bad things; they are useful until we want to get out of them. One of the primary goals of the Web Services technology stack was to reduce coupling and introduce an asynchronous processing model where the handler of a message could be updated to reflect new business logic without affecting the client. The WSDL-binding approach took note of this goal and did precisely the opposite. We usually cannot change the backend binding on the same port without affecting the client (which is what we were explicitly trying to avoid!).

The resource-oriented approach allows us to enforce contracts if and when we want to, but it does not require us to do so. By separating out the name of the thing from the structure of the form we accept, we can reuse the same logical name to support multiple types of interaction. We can upgrade the back-end without necessarily breaking existing clients. If we move from a model where all existing clients POST messages to a URL with the version 1 of the message schema, we can add support on the backend for version 2 of the schema while allowing business to proceed as usual if it makes sense to do so. If we ever want to reject an older schema, we can, but again we can choose when to do so. This flexibility is one of the reasons resource-oriented architectures help put the business back in control: backend system changes do not necessarily force frontend updates. If we wrap a legacy system with a RESTful interface, we can continue to use it until there is a compelling business reason to change it. Certainly other technologies allow us to wrap legacy systems in this way. It is the general approach to using the logical names that gives us a greater opportunity for avoiding middleware flux that makes the difference here.

In an effort to promote horizontal scalability, the RESTful style requires that requests be stateless. This means that any information that is needed to respond to a request comes in as part of the request. This allows us to use load balancers to bounce the request handling among any number of backend servers. In the face of increased load, more hardware can be thrown at the problem, and any of the servers can pick up and handle the request. Although scalability was the goal of this architectural constraint, another important consequence emerges from the