This highlights one of the main distinctions between REST and SOAP that causes confusion when people conflate the intent of the two styles. SOAP is a fine technology for invoking behavior, but it falls down as a means of managing information. REST is about managing information, not necessarily invoking arbitrary behavior through URLs. When people start scratching their heads and wondering if four verbs are enough to do what they want to do, they are probably not thinking about information; they are thinking about invoking behavior. If you are doing RPC through URLs, you might as well use SOAP. If you are treating important business concepts as addressable information resources that can be manipulated and represented in different forms in different contexts, you are taking advantage of REST and are likely to see some of the same benefits we see on the Web. Even if your backend systems use SOAP to satisfy a request, you can imagine benefiting from a RESTful interface. Not only does providing that kind of an address allow users to "surf for data," you also potentially introduce the ability to cache results and eliminate some of the pain of a changed WSDL contract. The clients would go through a logical coupling that gets translated into a SOAP message and response being generated. The content of the response could be stripped out of what we get back. We simply do not need to advertise that fact, and can gain architectural migration strategies in the process.

As we see in Figure 5-5, the same named resource might be returned in different physical forms in different contexts while retaining the same identity. We can imagine some type of company report organized into an information space that can be traversed through a time facet (e.g., year and then month). As long as there is only one type of report, *http://server/report/2008/02* is a reasonably good, long-lived name. At no point in the future will we change the fact that we had a report for February 2008. We may wish to access the data as XML in one scenario, as an Excel spreadsheet in another, or as a rendered JPEG image for inclusion in a summary report. We do not want different names for each of the scenarios, so we leverage content negotiation to specify our preference. The resource-oriented engine needs to know how to respond to a request type, but that is easy enough to enable. Some future data format might emerge that no current clients support. The clients will not need to be modified simply because we add support on the server and some other client takes advantage of it. This resilience in the face of change was designed into the Web and is something that we will want to take advantage of in the Enterprise as well. The client and server can negotiate a particular form for a named resource during the resolution process. This allows the same named resource to be structured differently in different contexts (e.g., XML during the middle tier, JSON in the browser, etc.). The structured forms can be cached by the server if it chooses to in each form.