anyone is allowed to fetch the definition of a PURL. There is the direct resolution process of hitting a PURL such as *http://purl.org/employee/briansletten* (which will result in the 303 redirect), as well as the indirect RESTful address of the PURL resource *http://purl.org/admin/purl/employee/briansletten*, which will return a definition of the PURL that currently looks something like the following:

```
<purl status="1">
    <id>/employee/briansletten</id>
    <type>303</type>
    <maintainers>
       <uid>brian</uid>
    </maintainers>
    <seealso>
       <url>http://bosatsu.net/foaf/brian.rdf</url>
    </seealso>
</purl>
```

Clients of the PURL server can "surf" to the data definition as a means of finding information about a PURL resource without actually resolving it. No code needs to be written to retrieve this information. We can view it in a browser or capture it on the command line with curl. As such, we can imagine writing shell scripts that use data from our information resources to check whether a PURL points to something valid and is returning reasonable results. If not, we could find the owner of the PURL and fire off a message to the email address associated with the account. Addressable, accessible data finds its way into all manner of unintended orchestrations, scripts, applications, and desktop widgets because it is so easy and useful to do so.

In the interest of full disclosure, we failed to support JSON as a request format in the initial release, which complicated the AJAX user interface. JavaScript XML handling leaves a lot to be desired. Even though we use the XML form internally, we should have gone to the trouble of exposing the JSON form for parsing in the browser. You can be sure we are fixing this oversight soon, but I thought it was important to highlight the benefits we could have taken advantage of if we had gotten it right in the first place. You do not need to support all data formats up front, but these days supporting both XML and JSON is a good start.

As an interesting side note, we could have chosen several containers and tools to expose this architecture as expressed so far. Anything that responds to HTTP requests could have acted as our PURL server. This represents a shallow but useful notion of RESTful interfaces and resource-oriented architecture, as demonstrated in Figure 5-7. Any web server or application server can act as a shallow resource-oriented engine. The logical HTTP requests are interpreted as requests into servlets, Restlets, and similar addressable functionality.