functionality that will interpret the request. This logical step hides a whole world of possibilities and technology choices, and simply does not leak unnecessary details to the client. It does not support all interaction styles, but you may be surprised at how much can fit comfortably behind a URL.
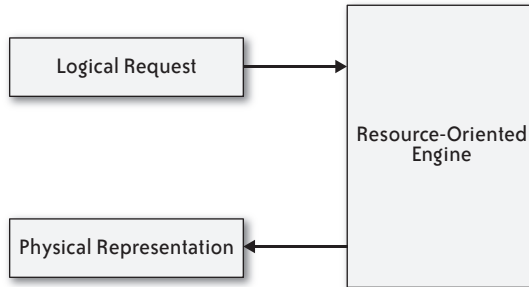

FIGURE 5-4. Resource-oriented architectures

Consider the address *http://server/getemployees&type=salaried*. Many people who think they are doing REST create URLs that look like this. Unfortunately, it is not a good REST service name (most Restafarians would argue it isn't REST at all!), because it conflates the nouns and verbs. That is what I like to call "addressing behavior through URLs" or "RPC through URLs." There is nothing magic about the REST way of separating nouns from verbs; it just allows us to identify the thing we care about. The aforementioned URL cannot be reused to update the employee list, because POSTing an employee record to "/getemployees" does not make any sense. If the URL were, instead, *http://server/employee/salaried*, then issuing a GET request to it will result in the same information, but this becomes a longer-lived address for the business concept of "salaried employees," just as *http://server/employee/hourly* can refer to employees who are paid by the hour. We may not choose to update these information resources, as they represent queries into whatever our backing store is. However, it is consistent within the */employee* information space, which we may choose to navigate in other ways. *http://server/employee/12345678* represents an employee with a particular ID, and *http://server/employee* might represent all employees. POSTing a record to this latter URL could represent hiring someone. PUTing a record to the specific employee ID URL could represent updating an employee record after a move, a raise, a promotion, etc. DELETEing the same address could indicate that the named resource is no longer of interest in the organization (i.e., either they quit or were fired).