that do not actually live on the Web but are still interesting to us) can be indicated by the use of a 303 response code rather than the usual 200. This is a hint to a client that, "Yes, the thing you asked for is legitimate and of interest, but does not actually live on the Web. You can find more information here...."

Web addresses begin with a reference to the HTTP protocol followed by the name of the server that will respond to the request. After that, there is a hierarchical scheme that should reflect a path through an information space. This is a logical name describing something about the structure of the data. Multiple paths might resolve to the same resource but will have value in different scenarios. *http://server/order/open* might return a list of open orders at a particular point in time, and *http://server/order/customer/112345* might reflect all open orders for a particular customer. Clearly, there would be overlap between the results returned from either of these logical requests. When we do not know what specifically to ask for, we might go the more general route. When we want to inquire as to the status of a specific customer, we would go the more direct route. We retrieve these logical URL references either from some other part of the system or generate them based on input from the client entering data through a user interface.

The separation of concerns here is among the key abstractions of the interaction style. We isolate the things we are interested in discussing, the actions by which we manipulate those things, and the forms we choose to send and receive them in. This is demonstrated in Figure 5-3 drawn from the discussion at RESTWiki.[§] In the REpresentational State Transfer (REST)[‖] architectural style, we refer to the resources (nouns), the verbs, and the representation of the response. The resources can be anything we can address (including concepts!). The verbs are GET (retrieve), POST/PUT (create/update), and DELETE (remove). GETs are constrained to have no consequences. This is called an idempotent request. The semantics of this interaction will contribute to the potential for caching. POSTs are generally used when there is no central authority to respond to the request (e.g., submitting news articles to a Usenet community) or we do not yet have the means of addressing our resource. We cannot identify an order before we create it, because the server application is responsible for creating order IDs. Therefore, we tend to POST these requests to a bit of functionality (e.g., a servlet) that accepts the request on our behalf and generates an ID in the process. PUTs are used to update and overwrite the existing state of a named resource. DELETE has no great use on the public Web (thankfully!), but in the context of an internally controlled, resource-oriented environment, indicating that we no longer need or care about particular resources is an important part of managing their life cycles. The REST style fundamentally works by separating the concerns of logically naming the resources we care about, the means by which we manipulate them, and the formats in which we choose to represent them, as shown in Figure 5-3.

§ *http://rest.blueoxen.net/cgi-bin/wiki.pl?RestTriangle*

‖ *http://en.wikipedia.org/wiki/REST*