Starbucks REST API

Drinks

Brewed Coffee

- Bold Pick of the Day
- Caffè Misto
- Clover® Brewed Coffee
- Coffee Traveler
- Decaf Pike Place® Roast
- Iced Coffee
- Iced Coffee with Milk
- Pike Place® Roast

Espresso Beverages

- Caffè Americano
- Caffè Latte
- Caffè Mocha
- Cappuccino
- Caramel Macchiato
- Cinnamon Dolce Latte
- Espresso
- Espresso Con Panna
- Espresso Macchiato
- Flavored Latte
- Iced Caffè Americano
- Iced Caffè Latte
- Iced Caffè Mocha
- Iced Caramel Macchiato
- Iced Cinnamon Dolce Latte
- Iced Flavored Latte
- Iced Peppermint Mocha
- Iced Peppermint White Chocolate Mocha
- Iced Salted Caramel Mocha
- Iced Skinny Flavored Latte
- Iced White Chocolate Mocha
- Peppermint Mocha
- Peppermint White Chocolate Mocha
- Pumpkin Spice Crème
- Pumpkin Spice Latte
- Salted Caramel Mocha
- Skinny Caramel Macchiato
- Skinny Cinnamon Dolce Latte
- Skinny Flavored Latte
- White Chocolate Mocha

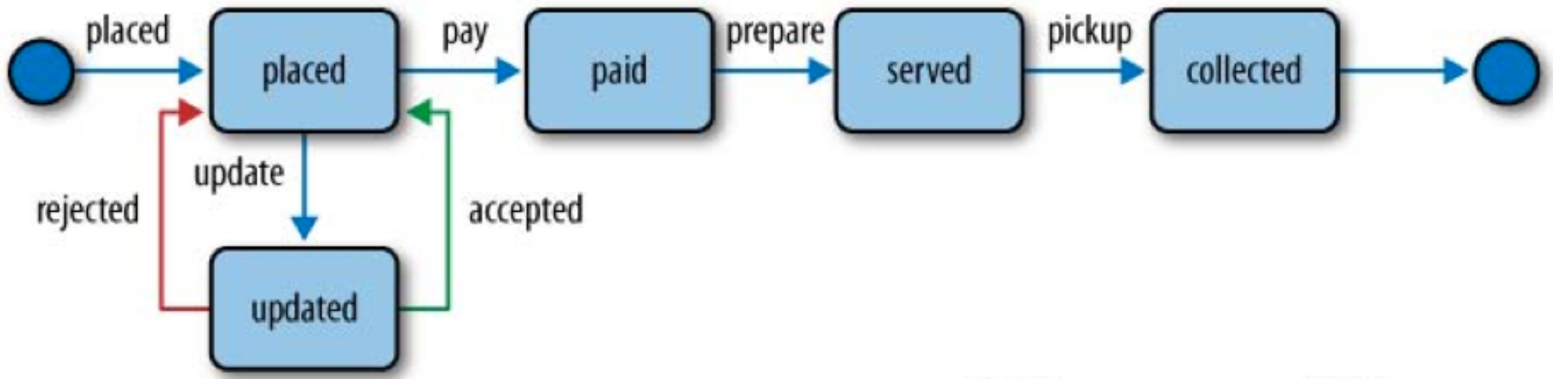


Figure 4-1. *Possible states for an order*

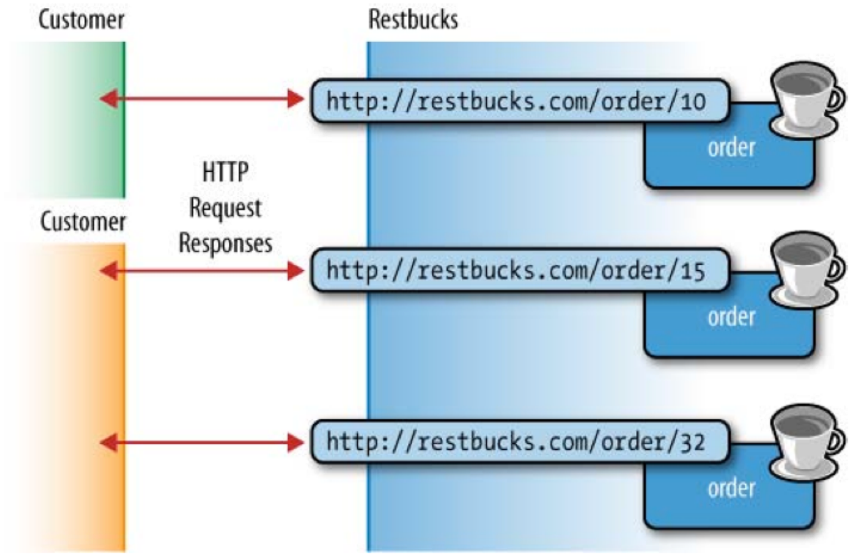

Customer
Restbucks
http://restbucks.com/order/10
order
HTTP
Request
Responses
Customer
http://restbucks.com/order/15
order
http://restbucks.com/order/32
order

Figure 4-2. *CRUD ordering service high-level architecture*

Table 4-1. *The ordering service contract overview*

Verb	**URI or template**	**Use**
`POST`	`/order`	Create a new order, and upon success, receive a `Location` header specifying the new order's URI.
`GET`	`/order/{orderId}`	Request the current state of the order specified by the URI.
`PUT`	`/order/{orderId}`	Update an order at the given URI with new information, providing the full representation.
`DELETE`	`/order/{orderId}`	Logically remove the order identified by the given URI.

Building CRUD Services

When you're building a service, it helps to think in terms of the behaviors that the service will implement. In turn, this leads us to think in terms of the contract that the service will expose to its consumers. Unlike other distributed system approaches, the contract that CRUD services such as Restbucks exposes to customers is straightforward, as it involves only a single concrete URI, a single URI template, and four HTTP verbs. In fact, it's so compact that we can provide an overview in just a few lines, as shown in Table 4-1.

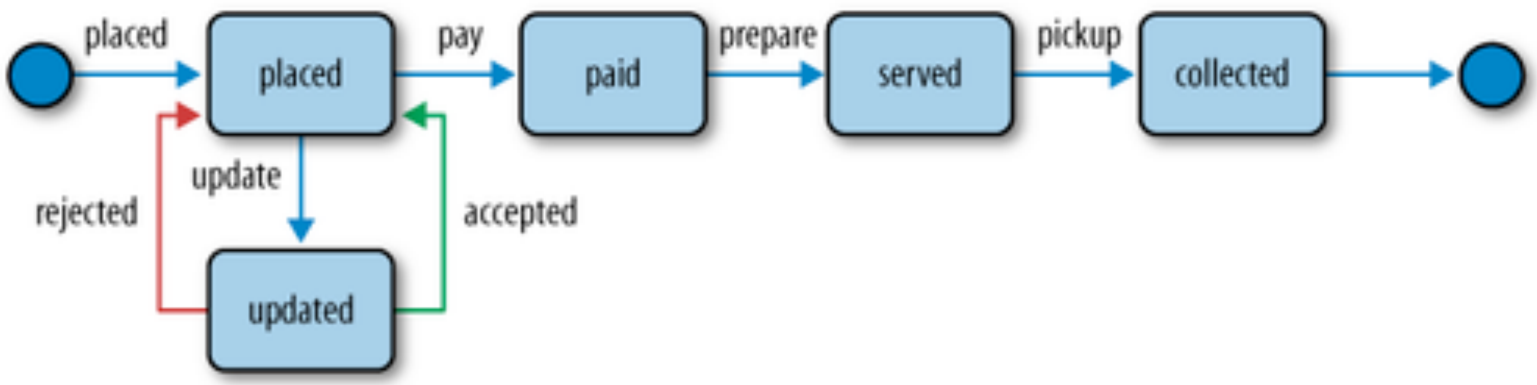
Table 4-1. *The ordering service contract overview*

Verb	URI or template	Use
`POST`	`/order`	Create a new order, and upon success, receive a `Location` header specifying the new order's URI.
`GET`	`/order/{orderId}`	Request the current state of the order specified by the URI.
`PUT`	`/order/{orderId}`	Update an order at the given URI with new information, providing the full representation.
`DELETE`	`/order/{orderId}`	Logically remove the order identified by the given URI.

The contract in Table 4-1 provides an understanding of the overall life cycle of an order. Using that contract, we can design a protocol to allow consumers to create, read, update, and delete orders. Better still, we can implement it in code and host it as a service.

NOTE

What constitutes a good format for your resource representations may vary depending on your problem domain. For Restbucks, we've chosen XML, though the Web is able to work with any reasonable format, such as JSON or YAML.



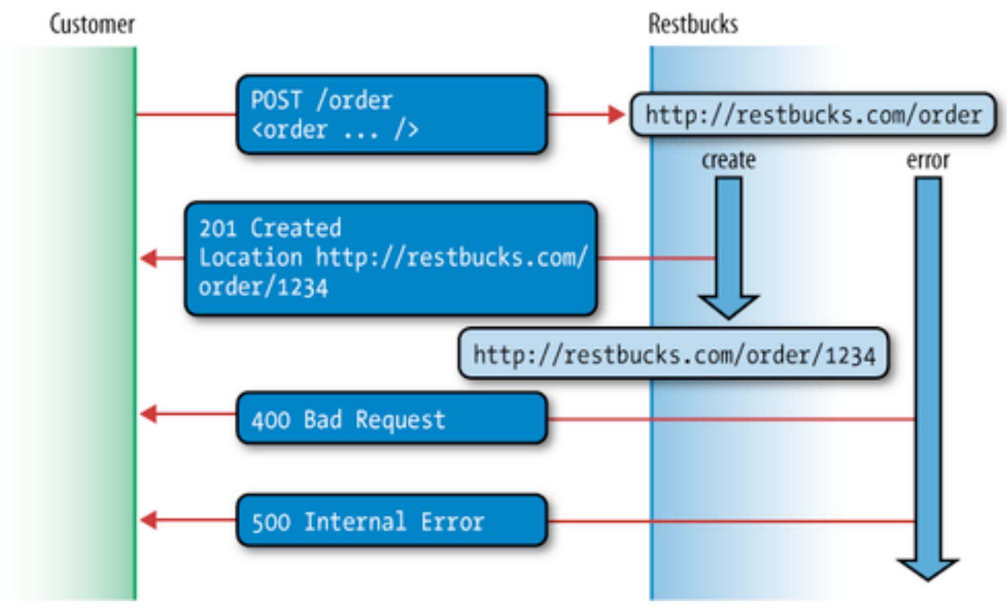
Each operation on an order can be mapped onto one of the HTTP verbs. For example, we use `POST` for creating a new order, `GET` for retrieving its details, `PUT` for updating it, and `DELETE` for, well, deleting it. When mixed with appropriate status codes and some commonsense patterns, HTTP can provide a good platform for CRUD domains, resulting in really simple architectures

Verb	URI or template	Use
`POST`	`/order`	Create a new order, and upon success, receive a `Location` header specifying the new order's URI.
`GET`	`/order/{orderId}`	Request the current state of the order specified by the URI.
`PUT`	`/order/{orderId}`	Update an order at the given URI with new information, providing the full representation.
`DELETE`	`/order/{orderId}`	Logically remove the order identified by the given URI.

POST

```
POST /order HTTP/1.1
Host: restbucks.com
Content-Type: application/xml
Content-Length: 239

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

```
HTTP/1.1 201 Created
Content-Length: 267
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:45:03 GMT
Location: http://restbucks.com/order/1234

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
  <status>pending</status>
</order>
```

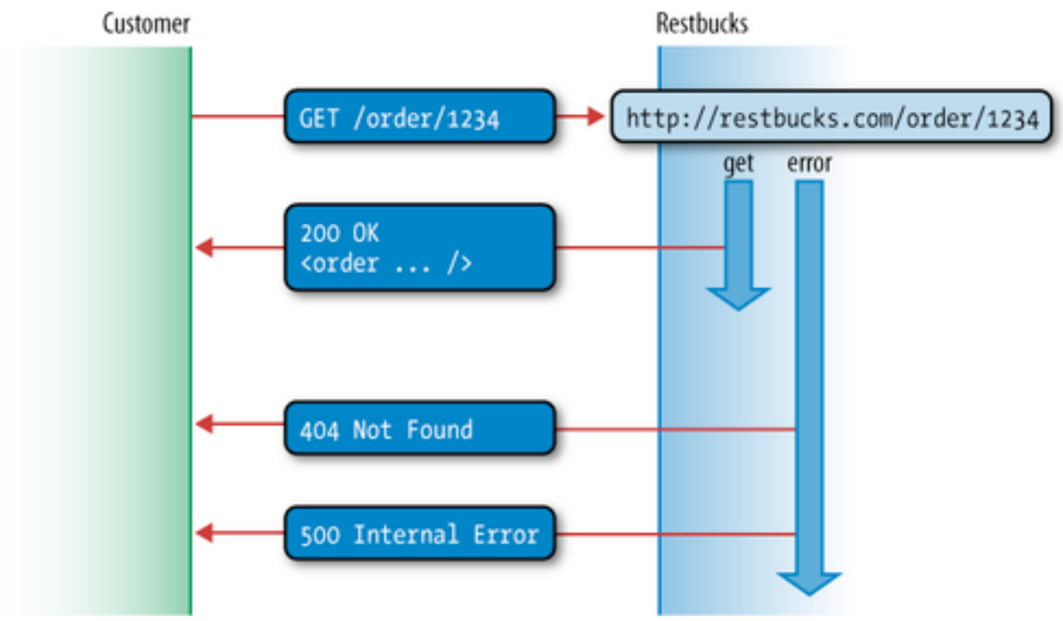

Customer
Restbucks
POST /order
<order ... />
http://restbucks.com/order
create
error
201 Created
Location http://restbucks.com/order/1234
http://restbucks.com/order/1234
400 Bad Request
500 Internal Error

```
HTTP/1.1 400 Bad Request
Content-Length: 250
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:48:11 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <!-- Missing drink type -->
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

GET

```
GET /order/1234 HTTP/1.1
Host: restbucks.com
```

```
HTTP/1.1 200 OK
Content-Length: 241
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:48:10 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

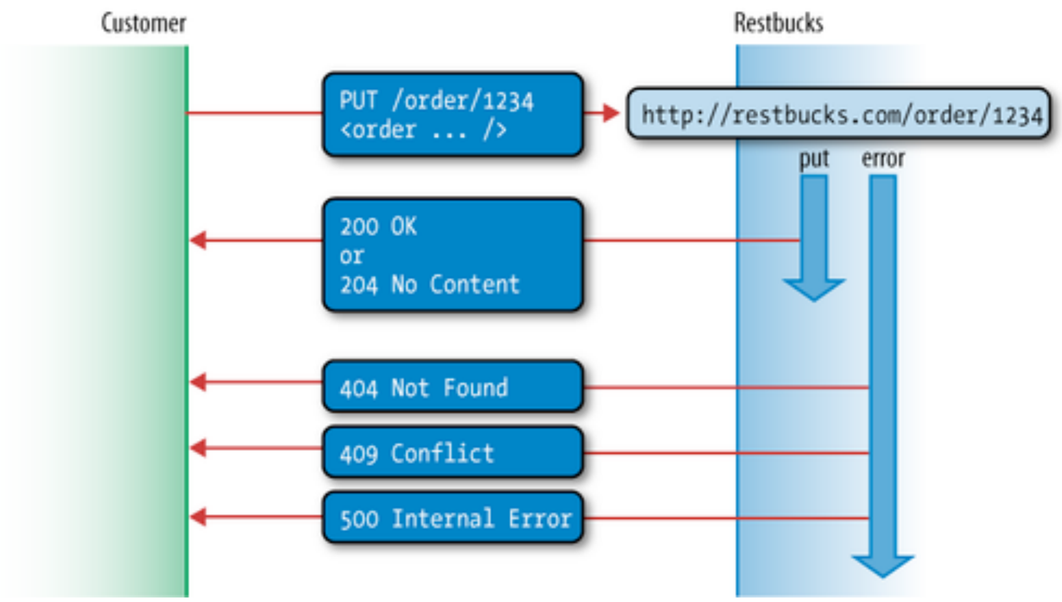


```
HTTP/1.1 404 Not Found
Date: Sat, 20 Dec 2008 19:01:33 GMT
```

PUT

```
PUT /order/1234 HTTP/1.1
Host: restbucks.com
Content-Type: application/xml
Content-Length: 246

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>skim</milk>
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
</order>
```

```
HTTP/1.1 200 OK
Content-Length: 275
Content-Type: application/xml
Date: Sun, 30 Nov 2008 21:47:34 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>skim</milk>
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
  <status>preparing</status>
</order>
```

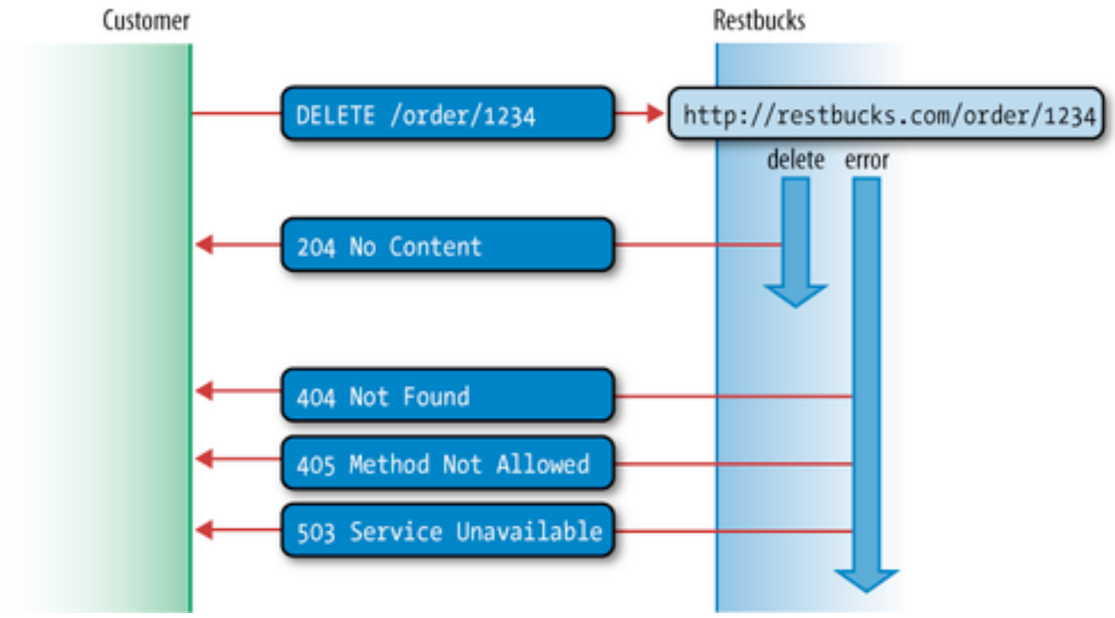


```
HTTP/1.1 204 No Content
Date: Sun, 30 Nov 2008 21:47:34 GMT
```

DELETE

```
DELETE /order/1234 HTTP/1.1
Host: restbucks.com
```

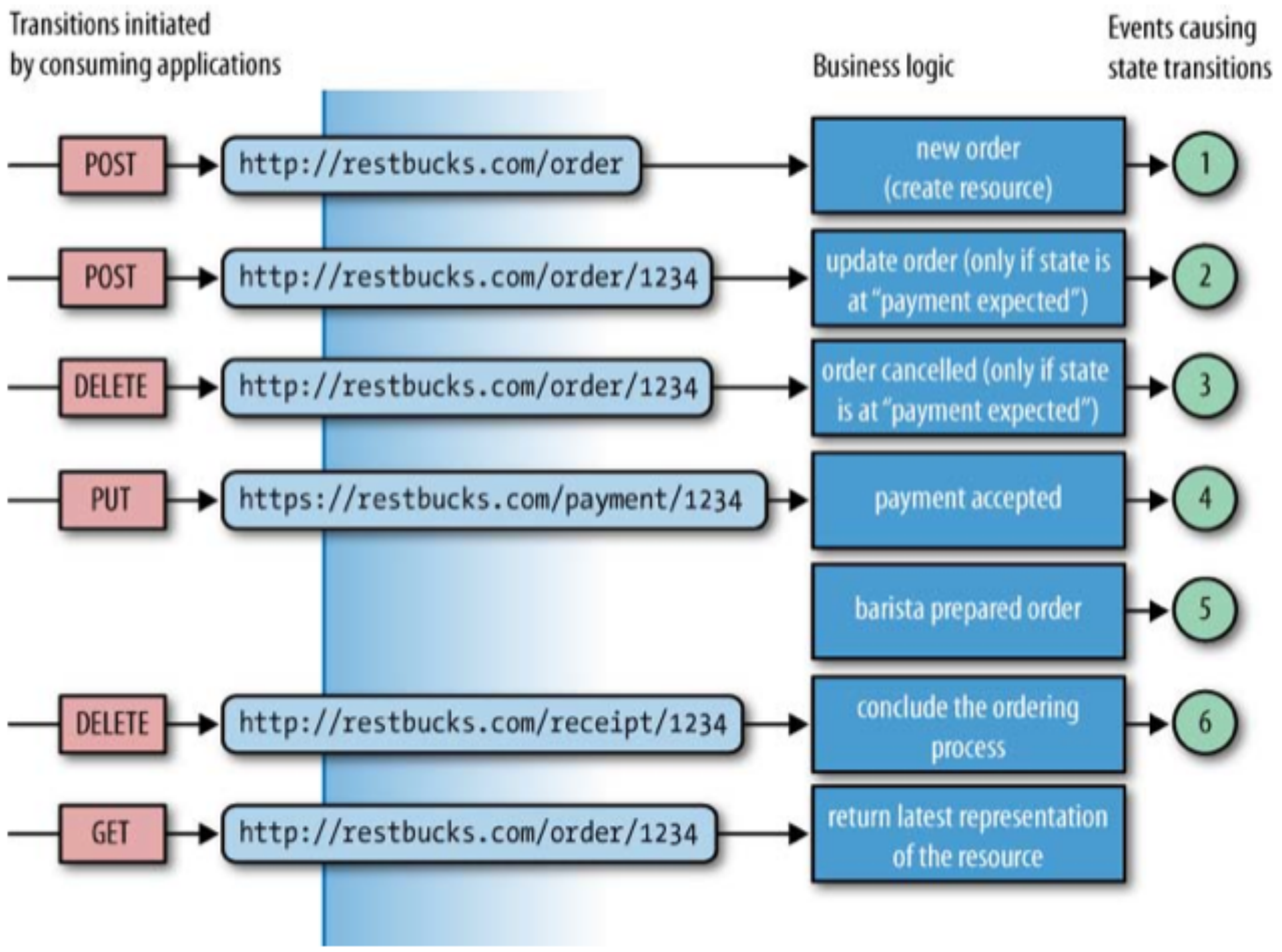
```
HTTP/1.1 204 No Content
Date: Tue, 16 Dec 2008 17:40:11 GMT
```



```
HTTP/1.1 404 Not Found
Content-Length: 0
Date: Tue, 16 Dec 2008 17:42:12 GMT
```

The Restbucks Domain Application Protocol

As a web-based system, Restbucks supports a DAP for ordering and payment. Figure 5-4 summarizes the HTTP requests that the ordering service supports and the associated workflow logic each request will trigger.



Transitions initiated by consuming applications
Business logic
Events causing state transitions
POST
http://restbucks.com/order
new order (create resource)
1
POST
http://restbucks.com/order/1234
update order (only if state is at "payment expected")
2
DELETE
http://restbucks.com/order/1234
order cancelled (only if state is at "payment expected")
3
PUT
https://restbucks.com/payment/1234
payment accepted
4
barista prepared order
5
DELETE
http://restbucks.com/receipt/1234
conclude the ordering process
6
GET
http://restbucks.com/order/1234
return latest representation of the resource

Figure 5-4. *Possible HTTP requests for the Restbucks ordering service*