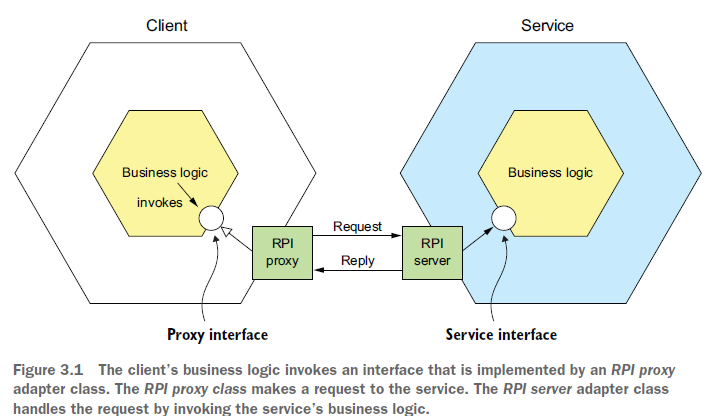
Synchronous Remote Procedure

Invocation (RPI) in Microservices

In RPI, A **client invokes a service** using a **synchronous,** **remote procedure invocation-based protocol, such as REST.**

When using a remote procedure invocation-based IPC mechanism, a client sends a request to a service, and the service processes the request and sends back a response**. Some clients may block waiting** for a response, and **others might have a reactive, nonblocking architecture**. But **unlike when using messaging, the client assumes that the response will arrive in a timely fashion.**



**The *proxy interface* usually encapsulates the underlying communication protocol**.

There are numerous protocols to choose from. We’ll talk about REST and gRPC as examples and how to improve the availability of your services by properly handling partial failure and **explain why a microservices-based application that uses RPI must use a service discovery mechanism.**

# Using REST

*REST* is an IPC mechanism that **(almost always) uses HTTP.** Roy Fielding, the creator of REST, defines REST as follows:

**(You should later on look for how exactly each one of these benefits are fulfilled:**

<https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>

<https://www.ibm.com/topics/rest-apis>

<https://en.wikipedia.org/wiki/REST>

).

“*REST provides a set of architectural constraints that, when applied as a whole, emphasizes* ***scalability of component interactions****,* ***generality of interfaces, independent deployment of components****,* ***and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems*.”**

* A key concept in REST is a ***resource***, **which typically represents a single business object, such as a Customer or Product, or a collection of business objects.**
* **REST uses the HTTP verbs for manipulating resources**, **which are referenced using a URL:**

For example, a GET request returns the **representation of a resource**, which is **often** in the form of **an XML document or JSON object**, although **other formats such as binary can be used.** A POST request creates a new resource, and a PUT request updates a resource. The Order Service, for example, has a POST /orders endpoint for creating an Order and a GET /orders/{orderId} endpoint for retrieving an Order.

Many developers claim their HTTP-based APIs are RESTful. But as Roy Fielding describes in a blog post**, not all of them actually are** (<http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>). To understand why, let’s take a look at the **REST maturity model:**

## The REST Maturity Model

Leonard Richardson (no relation to your author) defines a very useful maturity model for REST (http://martinfowler.com/articles/richardsonMaturityModel.html) that consists of the following levels:

 *Level 0*—Clients of a level 0 service invoke the service by making HTTP POST requests to its **sole URL endpoint.** Each request specifies the action to perform, the target of the action (for example, the business object), and any parameters.(single verb/single URL)

I add: on this level you’re basically using the HTTP as a transport tool to just invoke a remote procedure and you’re not using it as a platform to its fullest potential (verbs, status codes, etc.). and at this level they use **a single verb** that’s usually POST and the request details come in the body of the request or GET with parameters and actions in the URL, and even the errors like 400 will be sent by 200ok announcing the error in the response body.

 *Level 1*—**A level 1 service supports the idea of resources**. Level one employs **many URIs but only a single HTTP verb To perform an action on a resource**.(single verb/ multiple URLs)

a client for example makes a POST request that specifies the action to perform and any parameters in the body, or Get request with actions and parameters in the URL.

 *Level 2*—**A level 2 service uses HTTP verbs to perform actions**: GET to retrieve, POST to create, and PUT to update. **The request query parameters and body**, if any, specify the **actions' parameters**.(multiple URLs/multiple verbs/multiple status codes/even location header in a 201ok response pointing to where the new resource can be retrieved)

**This enables services to use web infrastructure such as caching for GET requests.**

 *Level 3*—The design of a level 3 service is based on the terribly named HATEOAS (Hypertext As The Engine Of Application State) principle. **The basic idea is that the representation of a resource returned by a GET request contains links for performing actions on that resource**. For example, a client can cancel an order using a link in the representation returned by the GET request that retrieved the order(They can be sent as a response to a POST request too. Check out the below example). The benefits of HATEOAS **include no longer having to hard-wire URLs into client code** (<www.infoq.com/news/2009/04/hateoas-restful-api-advantages>). As an example of a response to a GET request for a list of open slots:

<openSlotList>

<slot id = "1234" doctor = "mjones" start = "1400" end = "1450">

<link rel = "/linkrels/slot/book"

uri = "/slots/1234"/>

</slot>

<slot id = "5678" doctor = "mjones" start = "1600" end = "1650">

<link rel = "/linkrels/slot/book"

uri = "/slots/5678"/>

</slot>

</openSlotList>

Each slot now has a link element which contains a URI to tell us how to book an appointment. The point of hypermedia controls is that **they tell us what we can do next**, **and the URI of the resource we need to manipulate to do it**. Rather than us having to know where to post our appointment request, the hypermedia controls in the response tell us how to do it:



* One obvious benefit of hypermedia controls is that it allows the server to change its URI scheme without breaking clients. As long as clients look up the “addTest” link URI then the server team can juggle all URIs other than the initial entry points.

### The meaning of the Levels

I should stress that the RMM, while a good way to think about what the elements of REST are, is not a definition of levels of REST itself.

Roy Fielding has made it clear that Level 3 RMM(Richardson Maturity Model) is a pre-condition for REST APIs meaning that REST APIs must be hypertext-driven. You can see the levels as **tool to help us learn about the concepts** and **not something that should be used in some kind of** **assessment mechanism:**

* **Level 1** tackles the question of **handling complexity** by using **divide and conquer,** breaking a large service endpoint down into multiple resources.
* **Level 2** introduces a standard set of verbs so that **we handle similar situations in the same way, removing unnecessary variation.**
* **Level 3 introduces discoverability**, providing a way of making a protocol **more self-documenting**.

## Specifying REST APIs

Did u figure out what exactly synchronous mean/reactive/asynchronous?