COMS W4156 Advanced Software Engineering (ASE)

September 29, 2022

Agenda

1. RPC APIs
2. gRPC demo

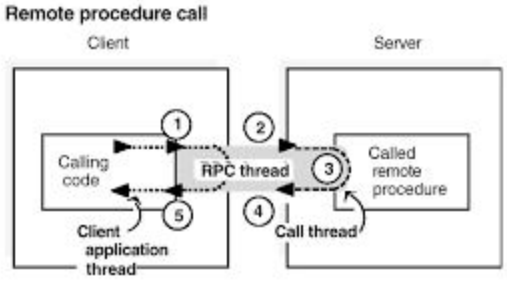


Figure 6-1 Execution Phases of an RPC Thread

Single Computer

One computer (or one OS process)

Caller makes a regular function call to the callee, providing parameters if required by the function signature, e.g., int foo (int x, int y); requires passing two integers

The callee (foo) receives the parameters, performs some computation, and returns result if required by the function signature, e.g., int foo (int x, int y); requires returning an int

Distributed Computing

Two or more computers (or two or more OS processes on same computer):

Caller makes what looks like a regular function call to the callee, providing parameters if required by the function signature, e.g., int foo (int x, int y); requires passing two integers

The callee (foo) receives the parameters, performs some computation, and returns what looks like a regular result if required by the function signature, e.g., int foo (int x, int y); requires returning an int

Magic happens under the covers

What Magic?

Client has to find (*binding*=naming+locating) and connect to server

Client parameters need to be *serialized* to send between processes or machines

And then indeed sent via *middleware* interprocess plumbing

Connection has to be made at server and specific function at server determined

Parameters need to be *deserialized* and that function called with the parameters

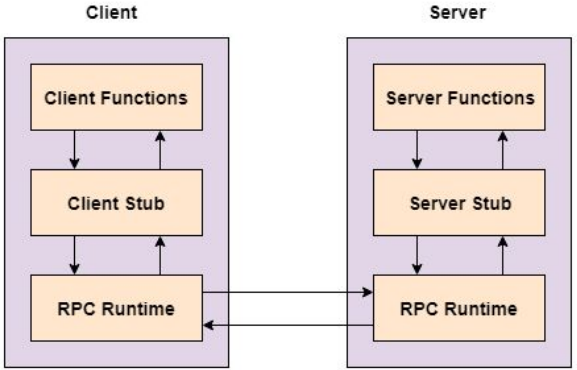
Server result has to be serialized and sent back to client via middleware, to be deserialized as return value

What Magic? 2

(de)serializing aka (de)marshalling

(De)Serializing parameters and return values is much more complicated for data structures than for primitives, so typically restricted (e.g., no pointers, no call-by-reference)

The client and server *do not* include any network communication code (e.g., no sockets)



What Could Possibly Go Wrong?

Server is down or unreachable

- Hang forever waiting for reply
- Time out and raise exception to caller
- Time out and retry

Possible failures worse if request and/or response require multiple network packets not just one

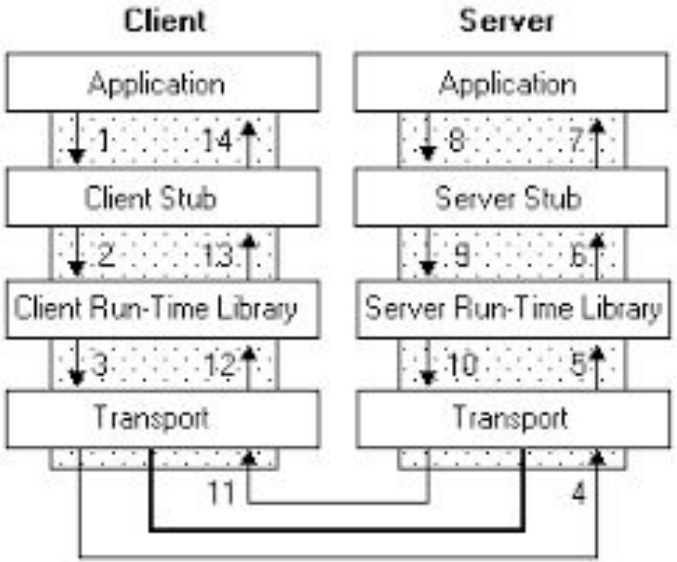
All these problems apply to RESTful APIs as well as RPC, hopefully handled by whatever framework you use since client/server code do not handle network communication

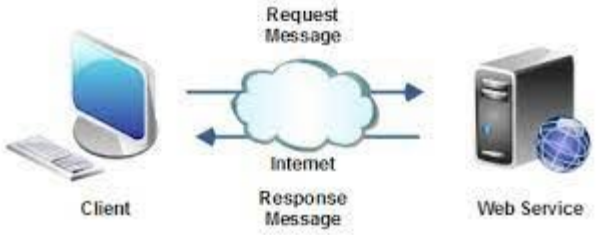
Time out and retry sounds best, but lots more things could go wrong:

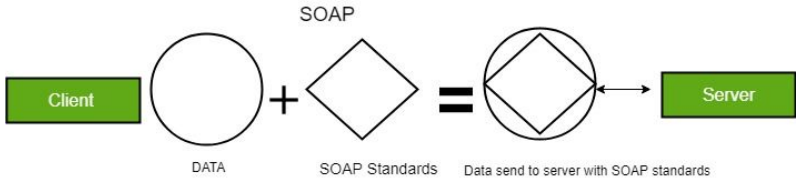
- Server receives request, performs computation with side-effects, crashes before sending response
- Server restarts and gets request again, but the computation is not idempotent (exactly once vs. at most once vs. at least once)
- Server restarts and gets request again, but it's rejected because of anti-replay semantics

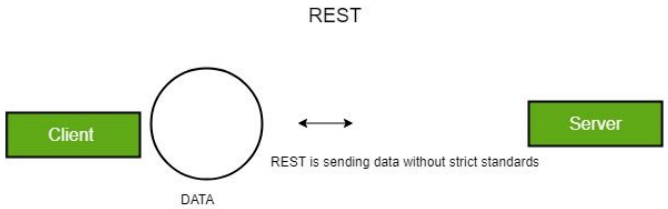
Client-Server Request-Response

Web service messages: send a message over HTTP/HTTPS, wait for response message









SOAP and friends

SOAP = Simple Object Access Protocol

There are several RPC API styles similar to SOAP that generally do ‘the same thing’, e.g., gRPC, Thrift, XML-RPC, JSON-RPC, once upon a time CORBA and COM

Most RPC APIs can use multiple different transport protocols, usually at least HTTP/S (web), SMTP (email), TCP (vanilla internet), and others

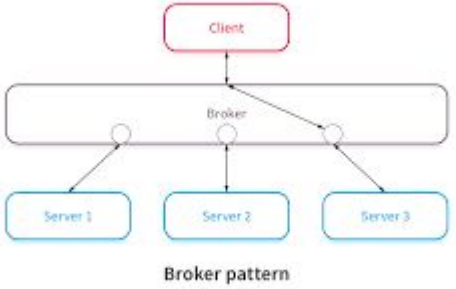
Data types of the parameters and results expected to be sent/received by a given service endpoint defined using a schema defined in some IDL (Interface Description Language)

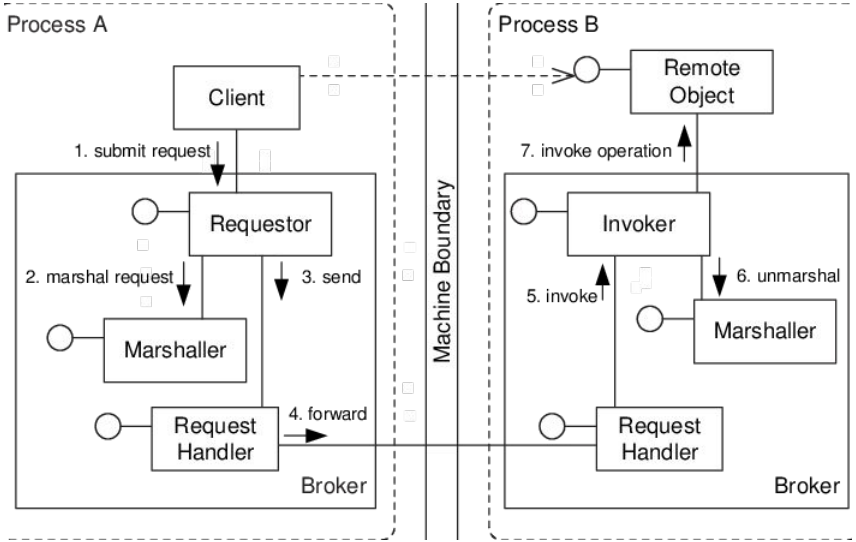
Actual data sent back and forth serialized in XML, JSON or another predefined language-agnostic format

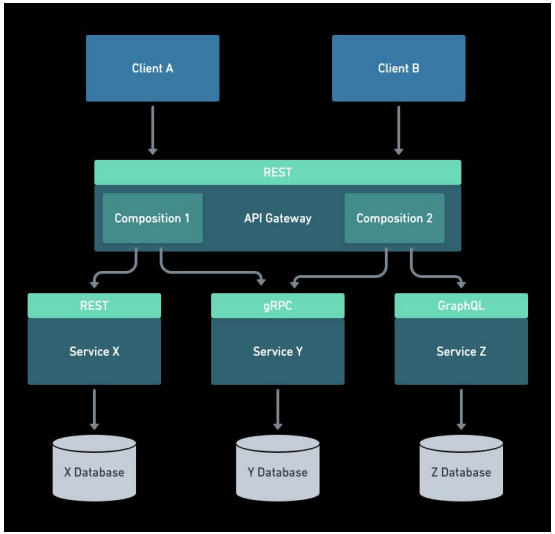
Standard structures for communicating errors and exceptions

Broker Mediated Request-Response

From client perspective, essentially same as client-server except the client does not know where the server is and sends instead to a broker or gateway





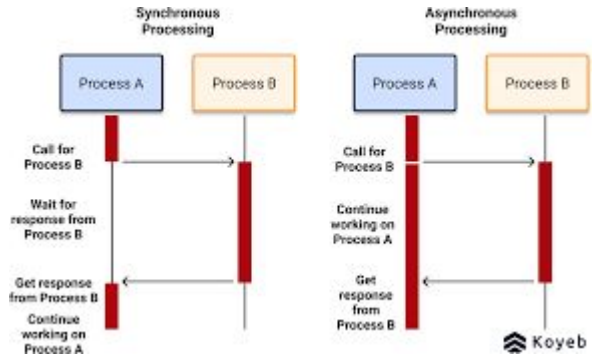


RPC vs. Publish-Subscribe

RPC is (usually) synchronous

Client waits for response from server just like a regular in-process function call

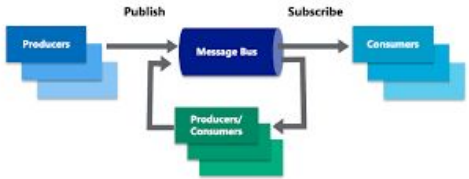
But there needs to be code somewhere to handle networking issues



Publish-subscribe is asynchronous

Client subscribes to everything from a specific server or everything that matches a given pattern from a set of servers (possibly from any server)

Client may or may not eventually get response(s) from server(s), may be deluged in many responses

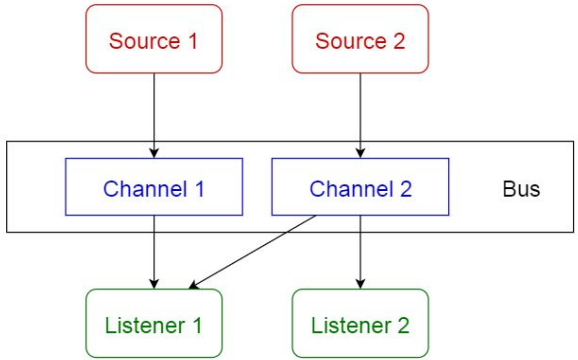


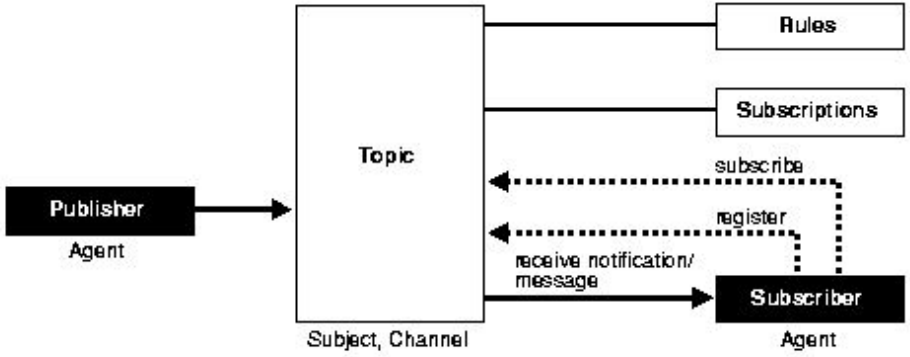
Publish-Subscribe

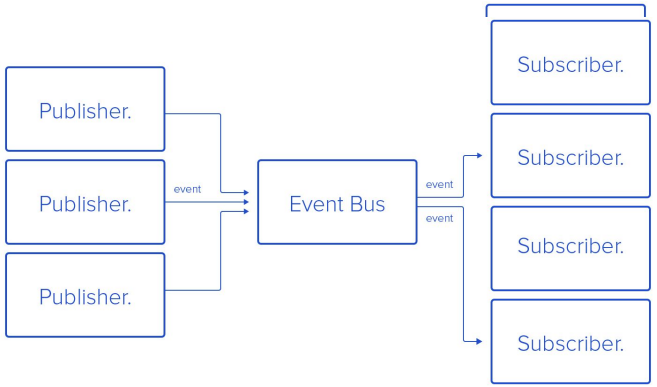
Twitter for software, not humans

Push - publisher or intermediary sends message or invokes callback for “interested” subscriber

Pull (or poll) - Subscribers keep checking







Agenda

1. RPC APIs
2. gRPC demo

gRPC demo: Kidus Mulu

[demo presentation]

Read The Docs

More information about RPC and other kinds of APIs here

Most RPC APIs work for multiple languages, not specific to Java, C++, etc.

Your Team Projects Need to Implement an API

Your service has to implement some API, but it does not need to be RESTful or RPC (publish-subscribe not usually used for conventional services, but it could be)

No end-user UI, neither GUI nor CLI

An administrative console is ok, just make sure your service cannot be mistaken for an “app”. If you find yourself describing or even thinking about your server as an “app”, you’re doing it wrong

Upcoming Assignments

Revised Project Proposal: due Monday October 3 (will accept until October 10) - meet with your Mentor to discuss your preliminary proposal *before* submitting revision

Mentors have been assigned! If you have not heard from your Mentor, contact Nihar (Head IA)

Next Class

Test Doubles (mocking)

API testing demo

Ask Me Anything