Communication Styles and Risk Analysis

Mojtaba Shahin

Week 9: Lectorial – Part 1

Content

- Part 1
  - Communication Styles
- Part 2
  - Architecture Risk Analysis

Acknowledgements

- Most of the **texts** and **images** in the slides come from the following sources:
  - Pautasso, C., Software Architecture- Visual Lecture Notes, LeanPub, 2023 (https://leanpub.com/software-architecture/)
  - Newman, Sam. Building Microservices, O'Reilly Media, Second Edition, 2021
  - Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.
  - Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020 (First Edition).
  - Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2025 (Second Edition).
  - Gandhi, R., Richards, M., Ford, N., Head First Software Architecture, O'Reilly Media, Inc. 2024
  - Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.
  - https://bytebytego.com/
  - Introduction to gRPC (https://grpc.io/docs/what-is-grpc/introduction/)
  - Building a GraphQL service (https://spring.io/guides/gs/graphql-server)
  - What Is a REST API? Examples, Uses, and Challenges (https:// blog.postman.com/rest-api-examples/)
  - Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy, Software Architecture: Foundations, Theory and Practice, John-Wiley, January 2009, ISBN 978047016774
    - https://www.softwarearchitecturebook.com/resources/

Styles of (Micro)Service Communication (Interaction)

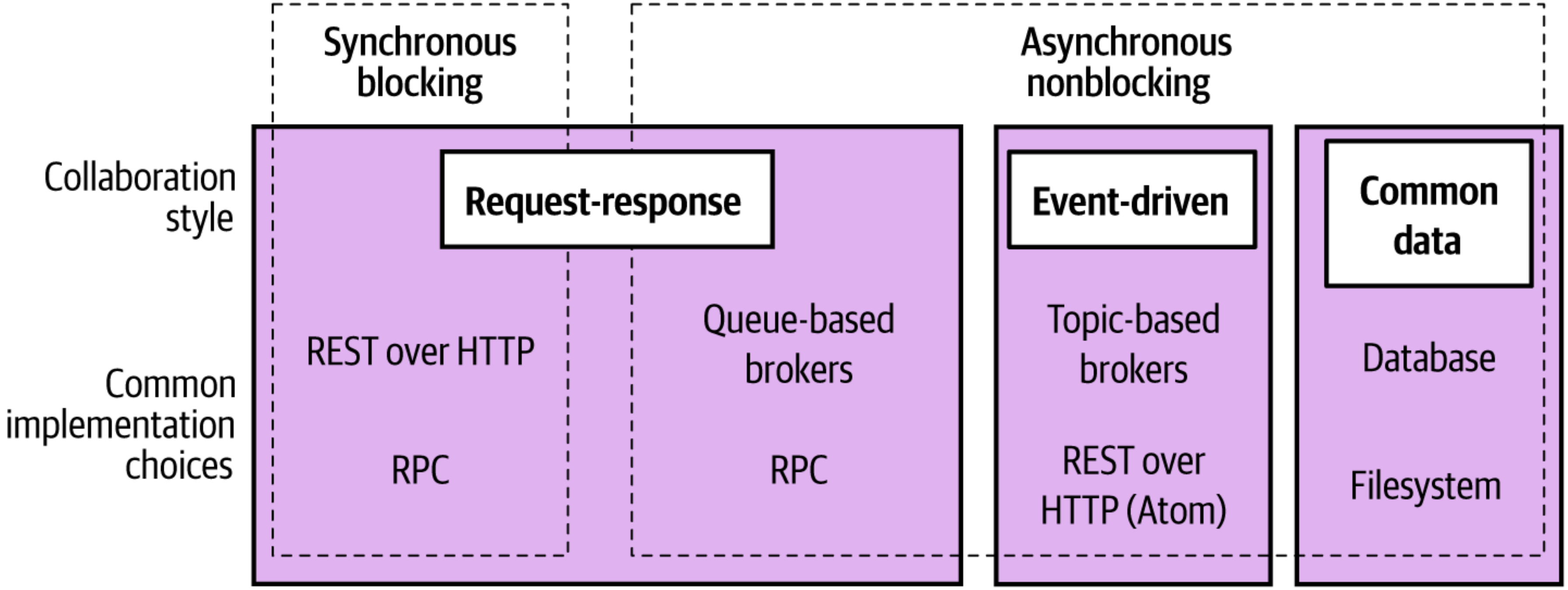


Image source: Newman, Sam. Building Microservices, O'Reilly Media, Second Edition, 2021

Communication/Interaction Styles

- **Synchronous blocking:**
  - A microservice makes a call to another microservice and blocks operation waiting for the response.
- **Asynchronous nonblocking**
  - The microservice emitting a call is able to carry on processing whether or not the call is received.
- **Request-response**
  - A microservice sends a request to another microservice asking for something to be done. It expects to receive a response informing it of the result.
- **Event-driven**
  - Microservices emit events, which other microservices consume and react to accordingly. The microservice emitting the event is unaware of which microservices, if any, consume the events it emits.
- **Common data**
  - Not often seen as a communication style, microservices collaborate via some shared data source.

Technology Choices

- RPC (Remote Procedure Call)
- REST (Representational State Transfer)
- GraphQL
- Message Brokers

RPC

- ***Remote Procedure Call (RPC).***
  - RPC refers to the technique of making a local call and having it execute on a remote service somewhere.
    - The programmer codes the call as if a local method were being called (with some syntactic variation); the call is translated into a message sent to a remote element where the actual method is invoked.
    - The results are sent back as a message to the calling element.
  - **Options**:
    - SOAP (Simple Object Access Protocol)
    - gRPC
    - Java RMI (Remote Method Invocation)

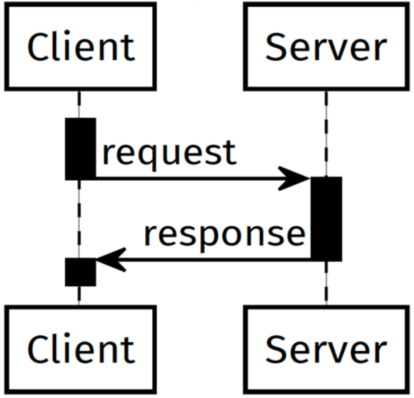


Image Source: Pautasso, C., Software Architecture- Visual Lecture Notes, LeanPub, 2023 (https://leanpub.com/software-architecture/)

REST

- ***REpresentational State Transfer (REST).***
- REST is a protocol for web services. It imposes six constraints on the interactions between elements:
  - ***Uniform interface***. All interactions use the same form (typically HTTP). Resources are specified via URIs (Uniform Resource Identifier).
  - ***Client-server***. The actors are clients and the resource providers are servers using the client-server pattern.
  - ***Stateless***. All client-server interactions are stateless.
  - ***Cacheable***. Caching is applied to resources when applicable.
  - ***Layered architecture***. The “server” can be broken into multiple elements, which may be deployed independently.
  - ***Code on demand (optional)***. It is possible for the server to provide code to the client to be executed. JavaScript is an example

REST

- **REST is resource-oriented**
  - To understand how REST APIs work, it is critical to understand **resources**.
  - A resource can be any information that could be named, such as a document or image, a collection of other resources, and more.
  - REST uses a resource identifier to recognise the specific resource involved in an interaction between components.

HTTP Command	CRUD Operation
POST	**CREATE a new resource**
GET	**READ data about an existing resource**
PUT	**UPDATE an existing resource**
DELETE	**DELETE an existing resource**

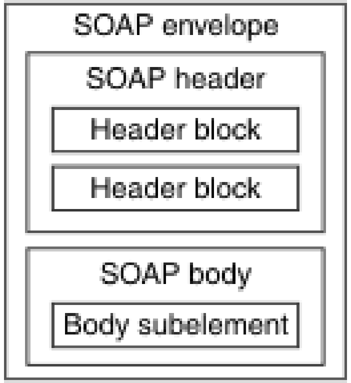
Representation and Structure of Exchanged Data

- XML
- JSON
- Protocol Buffer
- Plain Text
- …

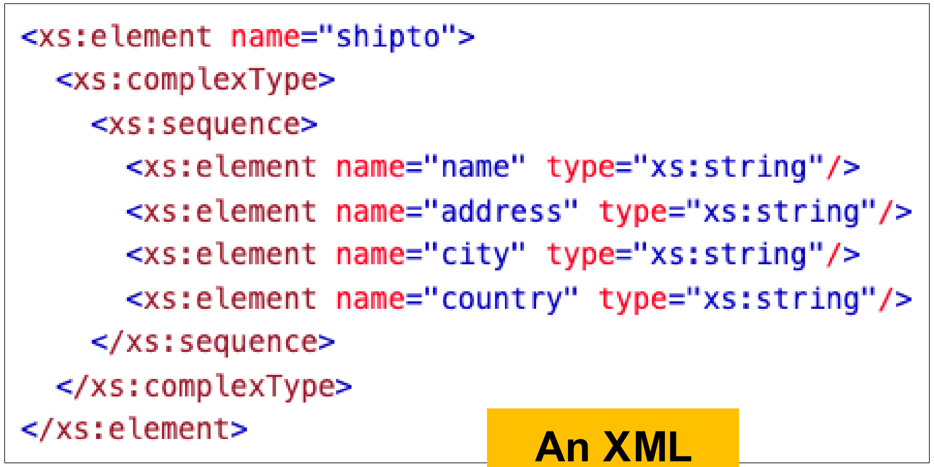
EXtensible Markup Language (XML)

- XML annotations to a textual document, called *tags*, are used to specify how to interpret the information in the document by breaking the information into fields and identifying the data type of each field.
  - Tags can be annotated with attributes.
- XML is a meta-language: Out of the box, it does nothing except allow you to define a customized language to describe your data.
- Your customized language is defined by an *XML schema*, which specifies the tags you will use, the data type used to interpret fields, and the constraints on the document.

EXtensible Markup Language (XML)

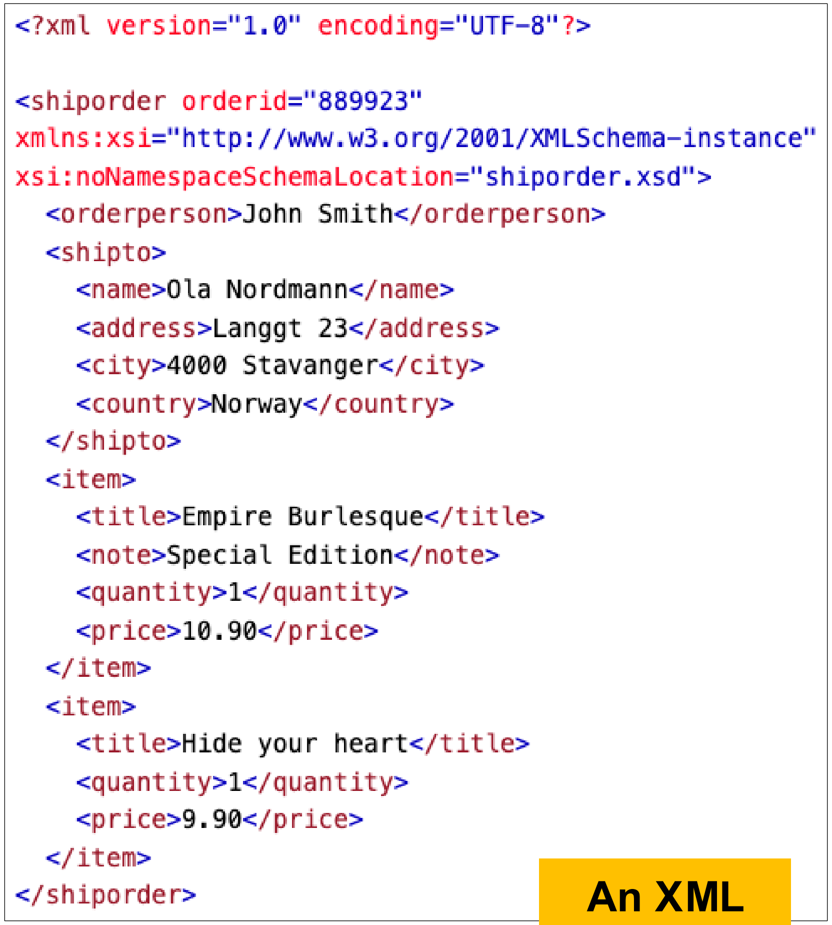


XML Structure



```
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

An XML schema



```
<?xml version="1.0" encoding="UTF-8"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

An XML file

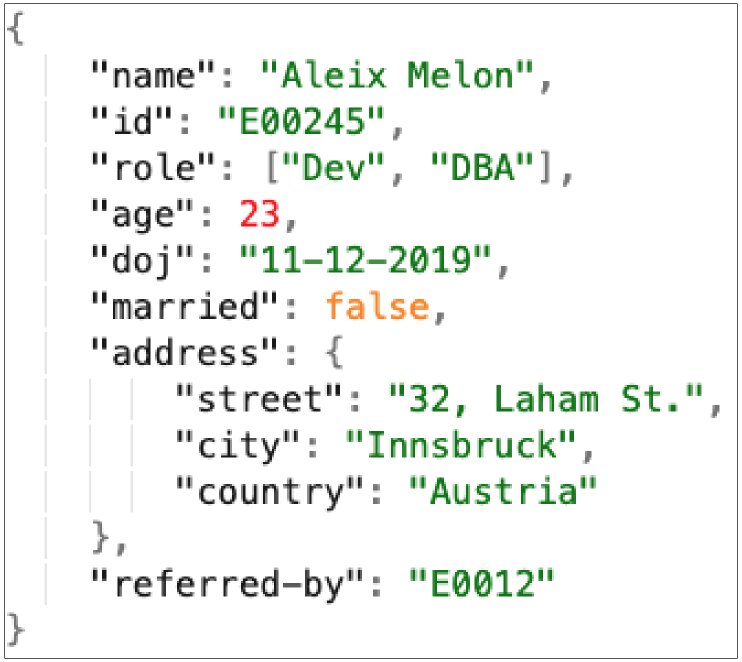
Image and Code Source: https://www.ibm.com/docs/en/integration-bus/10.0?topic=soap-structure-message

Image Soucre: https://www.w3schools.com/xml/schema_example.asp

JavaScript Object Notation (JSON)

- JSON structures data as nested name/value pairs and array data types.
- Like XML, JSON is a textual representation featuring its own schema.
- JSON data types are derived from JavaScript and resemble those of any modern programming language.
  - This makes JSON serialization and deserialization much more efficient than XML.

A sample JSON



```
{
    "name": "Aleix Melon",
    "id": "E00245",
    "role": ["Dev", "DBA"],
    "age": 23,
    "doj": "11-12-2019",
    "married": false,
    "address": {
        "street": "32, Laham St.",
        "city": "Innsbruck",
        "country": "Austria"
    },
    "referred-by": "E0012"
}
```

Code Source: https://www.freecodecamp.org/news/what-is-json-a-json-file-example/

gRPC

- The most recent version of RPC, called gRPC, transfers parameters in binary, is asynchronous, and supports authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts.
- By default, gRPC uses ***protocol buffers***.

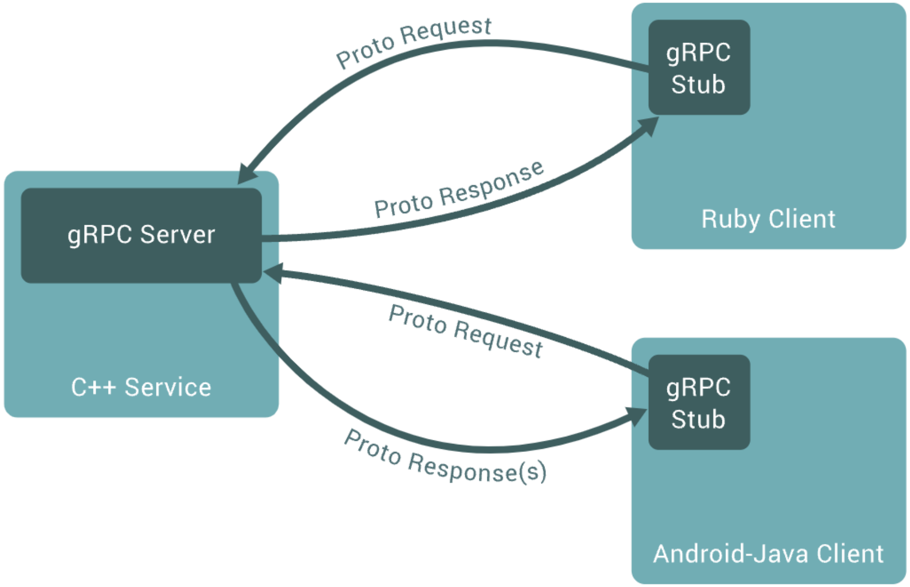
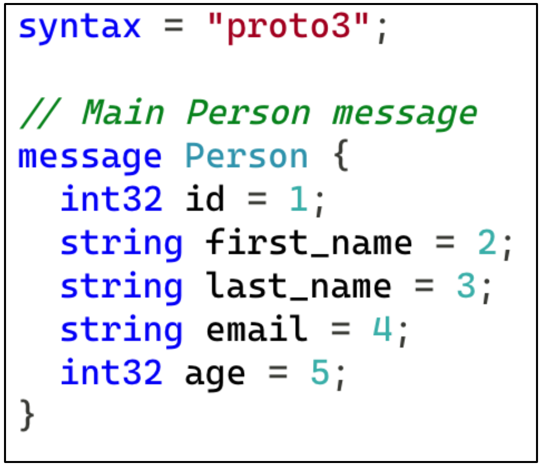


Image Source: https://grpc.io/docs/what-is-grpc/introduction/

Protocol Buffers

- Kept in a **.proto file**
- Like JSON, Protocol Buffers use data types that are close to programming-language data types, making serialization and deserialization efficient.
- As with XML, Protocol Buffer messages have a schema that defines a valid structure, and that schema can specify both **required** and **optional elements** and **nested elements**.
- However, unlike both XML and JSON, Protocol Buffers are a **binary format**, so they are extremely **compact** and **efficient**.

A sample buffer protocol schema



```
syntax = "proto3";

// Main Person message
message Person {
  int32 id = 1;
  string first_name = 2;
  string last_name = 3;
  string email = 4;
  int32 age = 5;
}
```

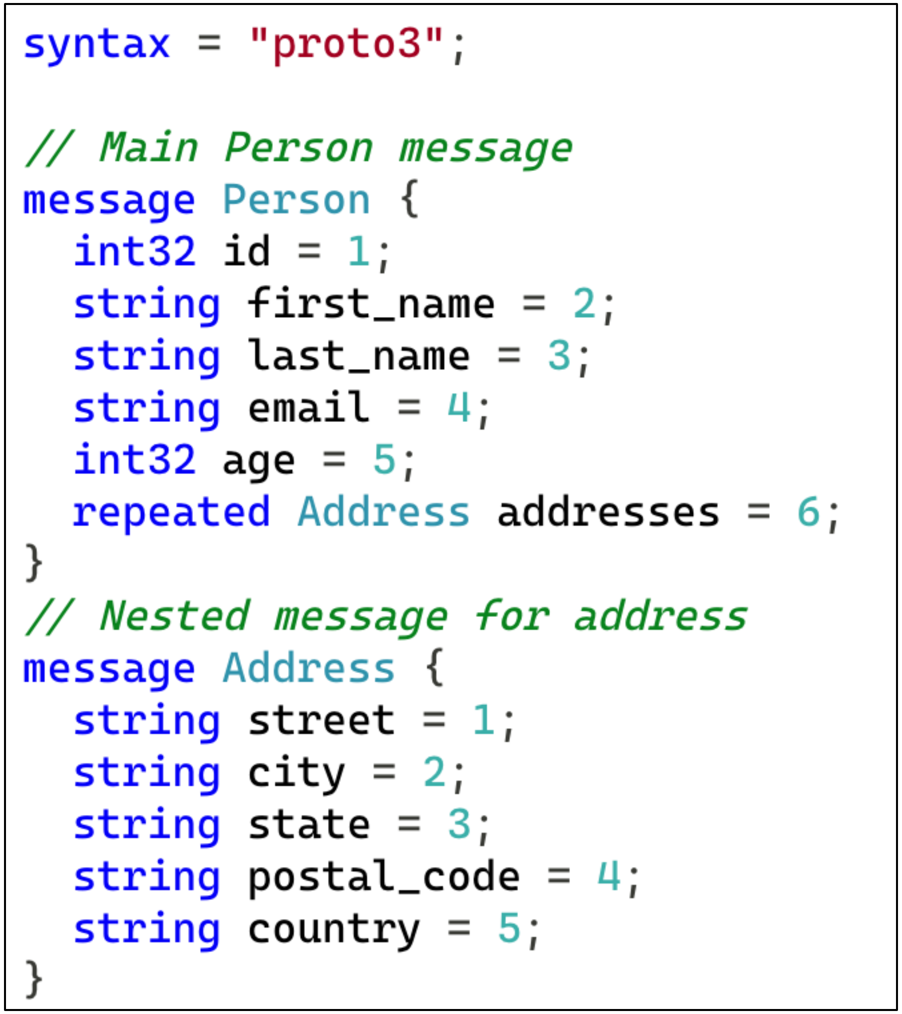
It defines a structured data type called Person

Field numbers (= 1, = 2, = 3, etc.) unique IDs used in the binary encoding.

Image Source: https://grpc.io/docs/what-is-grpc/introduction/

Protocol Buffers

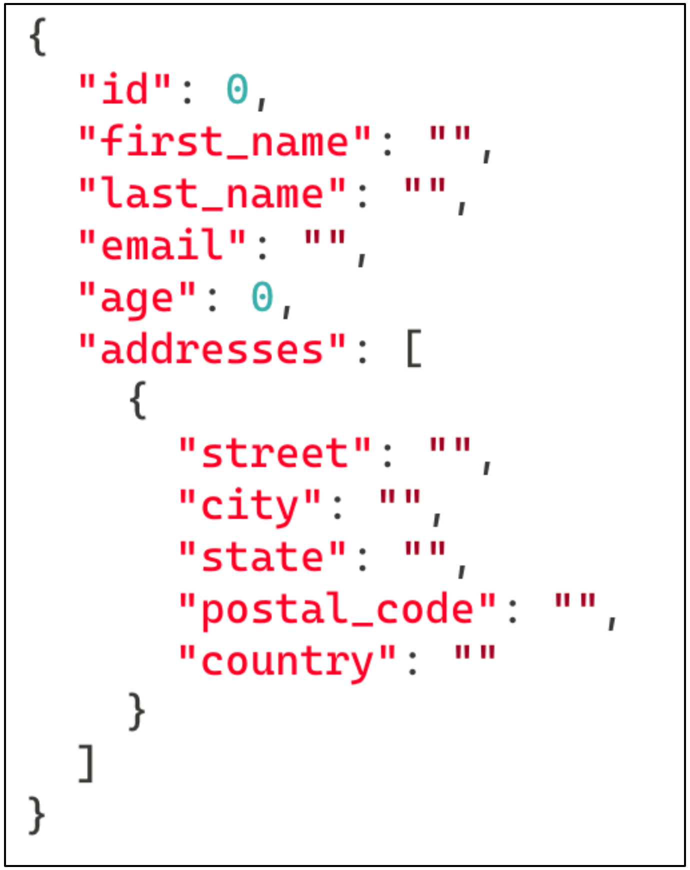
Message –Protocol Buffer



```
syntax = "proto3";

// Main Person message
message Person {
  int32 id = 1;
  string first_name = 2;
  string last_name = 3;
  string email = 4;
  int32 age = 5;
  repeated Address addresses = 6;
}
// Nested message for address
message Address {
  string street = 1;
  string city = 2;
  string state = 3;
  string postal_code = 4;
  string country = 5;
}
```

Message -JSON



```
{
  "id": 0,
  "first_name": "",
  "last_name": "",
  "email": "",
  "age": 0,
  "addresses": [
    {
      "street": "",
      "city": "",
      "state": "",
      "postal_code": "",
      "country": ""
    }
  ]
}
```

Query-Oriented APIs: GraphQL

- Query-oriented APIs seek to solve some problems that occur with other API approaches such as REST-oriented APIs.

- In REST-oriented APIs, obtaining the desired information may require combining results from **multiple calls to different endpoints**, which often results in the retrieval of **redundant information** in each of the calls.

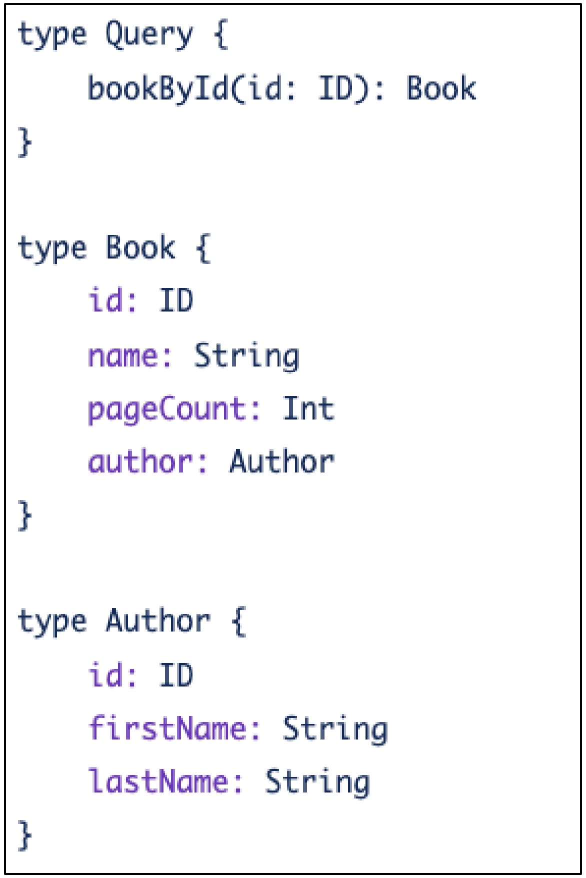
- This interaction model can be inefficient and lead to **unsatisfactory performance**.

GraphQL

- In query-oriented APIs, such as **GraphQL**, each client specifies exactly which information it is interested in.
- GraphQL uses a schema to define what data clients can request and how.
- Queries are executed on the server side by a specialized component that can retrieve data from different sources of information, and the results are returned to the client.
- Only the data specified in the query is returned—unlike the case with standard REST-oriented APIs, which tend to return unnecessary data.

Define a GraphQL Schema*

An example of a GraphQL schema



```
type Query {
    bookById(id: ID): Book
}

type Book {
    id: ID
    name: String
    pageCount: Int
    author: Author
}

type Author {
    id: ID
    firstName: String
    lastName: String
}
```

Every GraphQL schema has a top-level **Query type**, and the fields under it are the query operations exposed by the application.

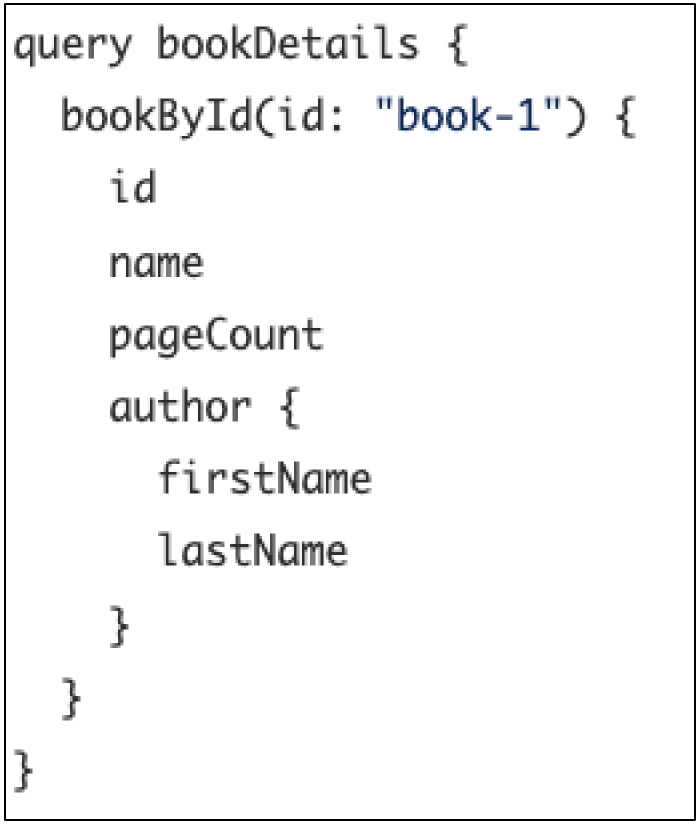
This schema defines one query called **bookById** that returns the details of a specific book.

It also defines the type **Book** with fields id, name, pageCount and author, and the type **Author** with fields firstName and lastName.

*Source: https://spring.io/guides/gs/graphql-server

GraphQL*

An example of a **request** that can be sent to a GraphQL server to retrieve book details:



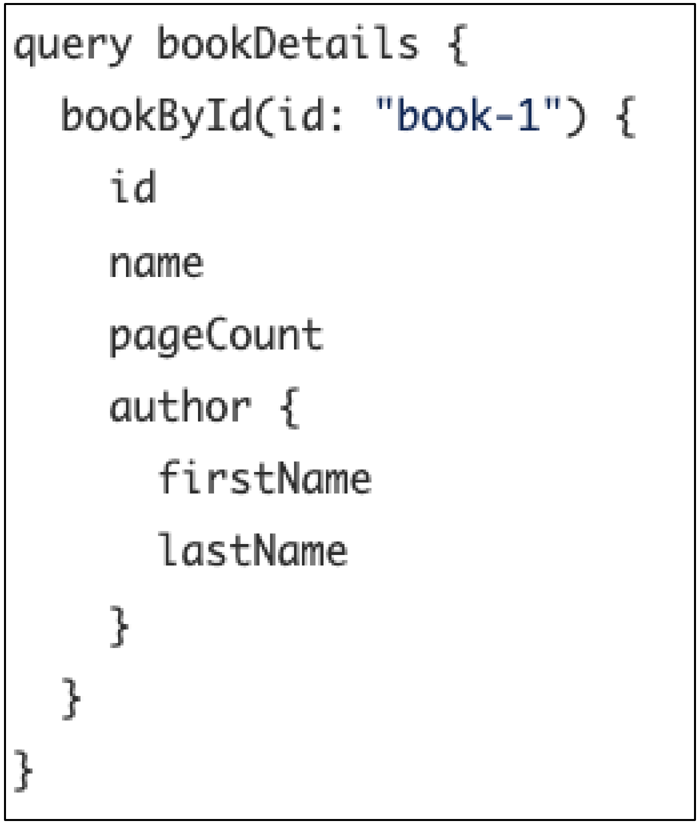
```
query bookDetails {
  bookById(id: "book-1") {
    id
    name
    pageCount
    author {
      firstName
      lastName
    }
  }
}
```

- Perform a query for a book with id "book-1”
- For the book, return id, name, pageCount and author
- For the author, return firstName and lastName

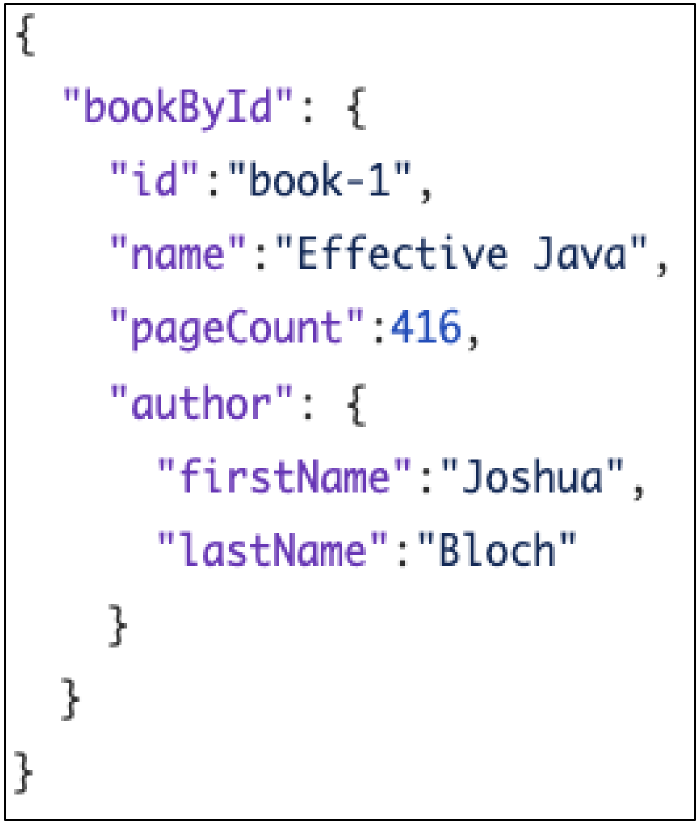
*Source: https://spring.io/guides/gs/graphql-server

GraphQL*

An example of a **response** returned by the server in JSON.



```
query bookDetails {
  bookById(id: "book-1") {
    id
    name
    pageCount
    author {
      firstName
      lastName
    }
  }
}
```

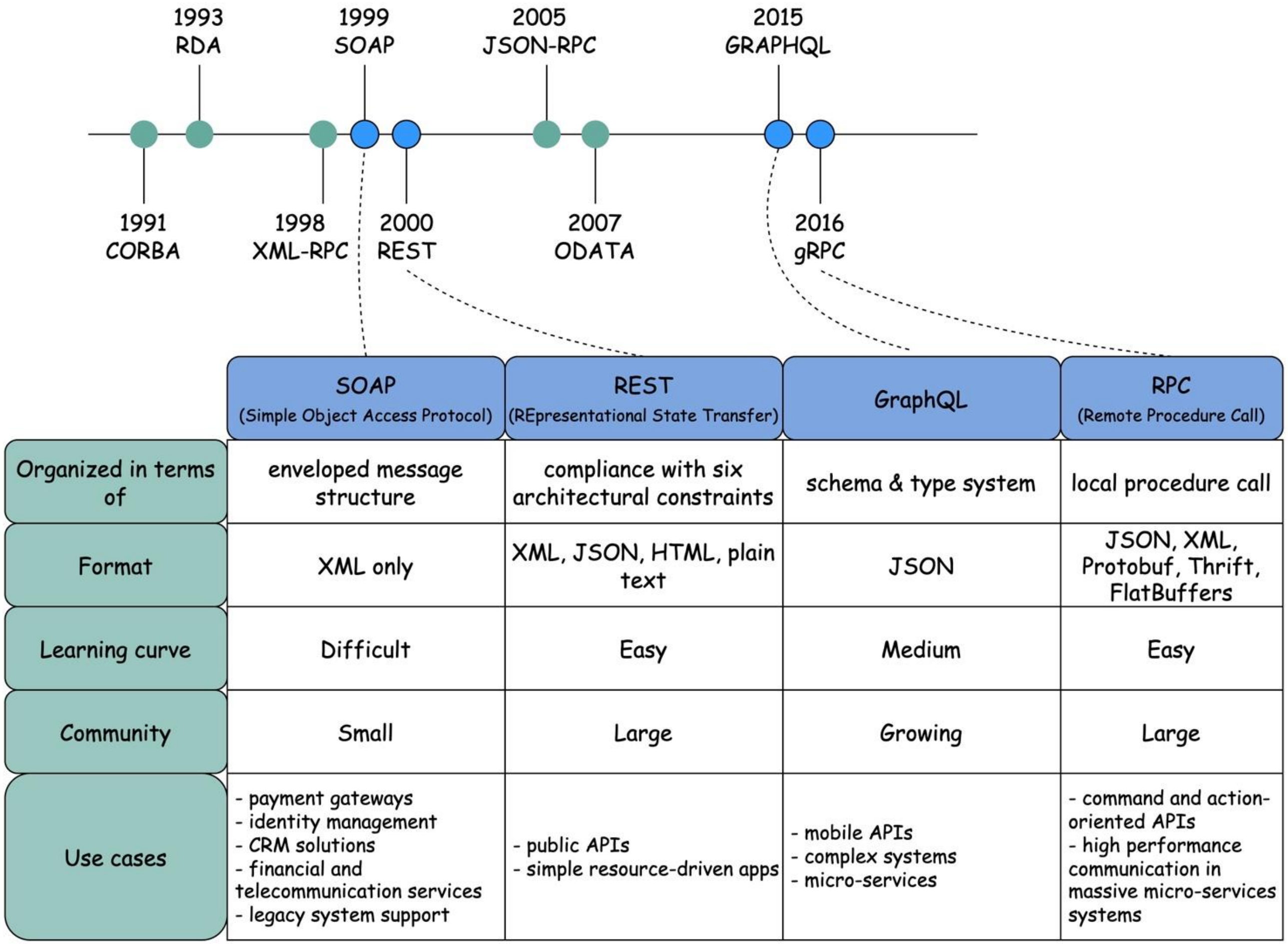


```
{
  "bookById": {
    "id":"book-1",
    "name":"Effective Java",
    "pageCount":416,
    "author": {
      "firstName":"Joshua",
      "lastName":"Bloch"
    }
  }
}
```

*Source: https://spring.io/guides/gs/graphql-server

API Architectural Styles Comparison

Source: altexsoft



	SOAP (Simple Object Access Protocol)	REST (REpresentational State Transfer)	GraphQL	RPC (Remote Procedure Call)
Organized in terms of	enveloped message structure	compliance with six architectural constraints	schema & type system	local procedure call
Format	XML only	XML, JSON, HTML, plain text	JSON	JSON, XML, Protobuf, Thrift, FlatBuffers
Learning curve	Difficult	Easy	Medium	Easy
Community	Small	Large	Growing	Large
Use cases	- payment gateways - identity management - CRM solutions - financial and telecommunication services - legacy system support	- public APIs - simple resource-driven apps	- mobile APIs - complex systems - micro-services	- command and action-oriented APIs - high performance communication in massive micro-services systems

Source: https://bytebytego.com/guides/soap-vs-rest-vs-graphql-vs-rpc/

source: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/

References

- Pautasso, C., Software Architecture- Visual Lecture Notes, LeanPub, 2023 (https://leanpub.com/software-architecture/)
- Newman, Sam. Building Microservices, O'Reilly Media, Second Edition, 2021
- Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.
- Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020 (First Edition).
- Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2025 (Second Edition).
- Gandhi, R., Richards, M., Ford, N., Head First Software Architecture, O'Reilly Media, Inc. 2024
- Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.
- Building a GraphQL service (https://spring.io/guides/gs/graphql-server)
- Introduction to gRPC (https://grpc.io/docs/what-is-grpc/introduction/)
- https://bytebytego.com/
- What Is a REST API? Examples, Uses, and Challenges (https://blog.postman.com/rest-api-examples/)
- Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy, Software Architecture: Foundations, Theory and Practice, John-Wiley, January 2009, ISBN 978047016774
  - https://www.softwarearchitecturebook.com/resources/