DS Lecture 3
Service Oriented Architectures

September 21st, 2022

Michele Albano
`mialb@cs.aau.dk`

DEIS
Aalborg University
Denmark

AALBORG UNIVERSITY
DENMARK

Agenda

Service-oriented architecture

What is a SOA?

- SOA is a set of principles for the design, deployment and management of both applications and software infrastructure using sets of *loosely coupled* services that can be *dynamically discovered* and then *communicate with each other* or are coordinated through *choreography* to provide enhanced services
- Services become building blocks that form business flows

- Gartner 18 July 2006: SOA is entering the "trough of disillusionment"

Service

- Many different definitions
  - Bad definition based on the technology used, since the principle is not just for "web-services"
- Possible definitions: A service is
  - a discrete business function that operates on data
  - a reusable component that can be used as a building block to form larger, more complex business-application functionality

Properties of Services

Four properties of a service:

- It is a black box for its consumers
- It is self-contained (loose-coupling between services)
- It may consist of other underlying services
- It should be stateless

Properties of Services

- It is a black box for its consumers
  - A service presents a simple interface articulated in endpoints to the requester that abstracts away the underlying complexity
  - The SOA infrastructure will provide standardized access mechanisms to discover services with service-level agreements
- It is self-contained (loose-coupling between services)
  - The consumer of the service is required to provide only the data stated on the interface definition, and to expect only the results specified on the interface definition
  - In the context of web services, loose coupling refers to minimizing the dependencies between services in order to have a flexible underlying architecture (reducing the risk that a change in one service will have a knock-on effect on other services)

Properties of Services

- It may consist of other underlying services
  - It allow users to combine and reuse them in the production of applications
  - It should be based on open standards. Open standards ensure the broadest integration compatibility opportunities
- It should be stateless
  - The service does not maintain state between invocations
  - If a transaction is involved, the transaction is committed and the data is saved somewhere
  - The id of the transaction can provide context (sessions)

What can we do with services?

- Multiple services can be composed (orchestrated) to provide the functionality of a large software application
  - Services must communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services
  - Thus, need for protocols that describe how they pass and parse messages using description metadata (for both functional and non-functional requirements such as QoS)
- Facilitated by technologies and standards that facilitate components' communication and cooperation over a network

Innovation

What is a SOA?

The major innovation in SOC is the move from the object oriented paradigm to a service oriented one

- **Object Oriented:**
  - Object: stateful
- **Service Oriented:**
  - Service: stateless

Object oriented vs Service oriented

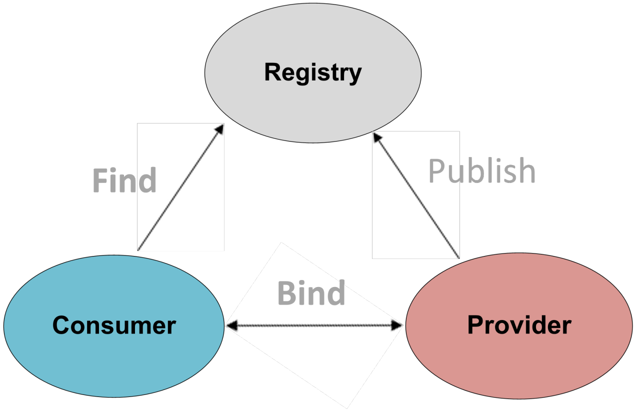
Features	Object-oriented computing	Service-oriented computing
Methodology	Application development by identifying tightly coupled classes. Application architecture is hierarchical based on the inheritance relationships.	Application development by identifying loosely coupled services and composing them into executable applications.
Level of abstraction and cooperation	Application development is often delegated to a single team responsible for the entire life cycle of the application. Developers must have knowledge of application domain and programming.	Development is delegated to three independent parties: application builder, service provider, and service broker. Application builders need to understand application logic and may not know how individual services are implemented. Service providers can program but do not have to understand the applications that use their services.
Code sharing and reuse	Code reuse through inheritance of class members and through library functions. Library functions have to be imported at compilation time and are platform dependent.	Code reuse at the service level. Services have standard interfaces and are published on Internet repository. They are platform-independent and can be searched and remotely accessed. Service brokerage enables systematic sharing of services.
Dynamic binding and recomposition	Associating a name to a method at runtime. The method must have been linked to the executable code before the application is deployed.	Binding a service request to a service at runtime. The services can be discovered after the application has been deployed. This feature allows an application to be recomposed at runtime.

Design Patterns

Three main roles

- **Service provider / publisher** : Offers the service, registers it in the service broker for consumers to find
- **Service consumer** : Retrieves service providers from service broker, then uses the services
- **Service broker / registry / repository**

Design Patterns



Coreography

- Service composition where the interaction is specified from a global perspective, to describe a set of interactions, showing the behavior of each member of a set of participants
- For example: it might specify constraints on the order and the conditions in which messages are exchanged by participants
- Difference with orchestration: it is performed by the services itself Orchestration is centralized

Solid principles

Recap from OOP

- **Single-responsibility Principle** : each component should do just one job
- **Open/Closed Principle** : your component should be open for extension but closed for modification
- **Liskov Substitution Principle** : if q(x) is a property provable about x of type T, q(y) should be provable for objects y of type S where S is a subtype of T
- **Interface Segregation Principle** : clients should not be forced to depend upon interfaces that they do not use
- **Dependency Inversion Principle** : an interface is an abstraction between a higher and a lower level component
  - Abstractions should not depend on details. Details should depend upon abstractions

Solid principles

SOA vs SOLID

- **Single-responsibility Principle** : each service should be specialized
- **Open/Closed Principle** : service orchestration
- **The Liskov Substitution Principle** : clients consume services, which consume lower granularity services
- **The Interface Segregation Principle** : in place of using one fat service interface, we create multiple small services
- **The Dependency Inversion Principle** : you can replace interface implementations without changing the interface

Microservice Architecture

Modern interpretation of SOA. Principles:

- fine-grained interfaces (to independently deployable services),
- business-driven development (e.g. domain-driven design),
- services are autonomous,
- polyglot programming and persistence,
- lightweight container deployment,
- decentralized continuous delivery.

Microservice Architecture

MSA service types

- **Functional Services** :
  - Support specific business operations.
  - Accessing of services is done externally and these services are not shared with other services.
- **Infrastructure services** : Auditing, security, and logging.
- Traditional SOA had 4 service types.

Agenda

Service Oriented Architecture

REpresentational State Transfer

Tools

- OpenAPI code generation
- Postman
- Wireshark

Uniform Resource Identifiers

- URIs are a way to identify resources (and endpoints) on the Web, and other Internet resources such as electronic mailboxes
- URIs are 'uniform': their syntax is defined through URI schemes, and there are procedures for managing the global namespace of schemes
- Specialization: Uniform Resource Locator (URL)
  - Example of URL: http://www.aau.dk/ a web page at the given path ('/') on the host www.aau.dk, and it is accessed using the HTTP protocol
  - Another example: mailto:mialb@cs.aau.dk mailbox of user mialb at cs.aau.dk

REpresentational State Transfer

- REST is an approach to services with a very constrained style of operation, applying “verbs” to “nouns”
- Nouns are the URLs that identify web resources
- Verbs are the HTTP operations GET, PUT, DELETE and POST (and lately PATCH) to manipulate resources
  - GET to retrieve the representation of a resource
  - POST to add a new resource
  - PUT to update the representation of a resource using a new one
  - PATCH to change part of the representation of a resource
  - DELETE to discard a resource

The tenets of REST

- Resources are identified by a URIs (can be a URL)
- Resources are manipulated through their representations
- Multiple representations are accepted or sent
- Messages are self-descriptive and stateless
- Hypermedia is the engine of application state

Statelessness: Two kinds of state

- **Application state** is the information necessary to understand the context of an interaction (e.g.: authentication, session)
- All REST messages must include all application state as part of the content transferred from client to server back to client
- Changes in **Resource state** are unavoidable (someone has to POST new resources before others can GET them)
- REST avoids implicit or unnamed state; resource state is named by URIs
- No application state implies:
  - It prevents partial failures
  - It allows for substrate independence
  - Load-balancing
  - Service interruptions

Hypermedia as the Engine of Application State

- Web application as a state machine:
  - the states are expressed as resources with links indicating transitions
- HATEOAS: hypermedia links in the response contents so that the client can dynamically navigate to the appropriate resource by traversing the hypermedia links
- A REST client hits an initial API URI and uses the server-provided links to dynamically discover available actions and access the resources it needs

Idempotent Operations

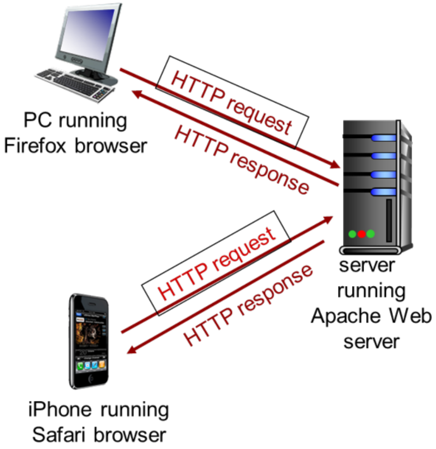
- A method, if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request.
- GET, PUT and DELETE methods are idempotent. Same example:
  - GET with Authentication: returns 401 (invalid authentication) vs with valid authentication (e.g.: 200).
  - PUT: either a 201 (resource is created) or 200/204 if the resource is updated
  - DELETE: 204 if the resource is deleted or a 404 if the resource was already deleted
- Not necessarily same response, but consecutive calls to the same method and resource MUST have the same intended effect on the server.

What if REST is not enough?

Current theories suggest that there are no applications you can think of which cannot be made to fit into the GET / PUT / POST / DELETE resources / representations model of the world.

Hypertext Transfer Protocol

- HTTP is a well-establish protocol that exhibit many features of a RESTful system
- Client/server model:
  - Client: It initiates the interaction (creates socket) with the server
  - server: It hosts resources identified by URIs, and sends objects back to the client in response to requests
- An HTTP request often causes the server to execute code that computes the representation of a resource
  - e.g.: Dynamic web pages, DB access, REST APIs



Agenda

Service Oriented Architecture

REpresentational State Transfer

Tools

- OpenAPI code generation
- Postman
- Wireshark

OpenAPI

OpenAPI 3.0 is a standard, accessible at:
`https://github.com/OAI/OpenAPI-Specification`
It describes:

- Available endpoints and operations on each endpoint
- Operation parameters Input and Output for each operation
- Authentication methods
- Contact information, license, terms of use and other information.

Mapped on the OpenAPI 3.0 format:

- “headers” parts, such as openapi version, specification info and servers
- Paths
- Components

OpenAPI

Paths:

- It is a series of `path` objects
- “path” part of the resources’ URL, followed by the operations it accepts
- Also, a list of HTTP status code, and optionally the specific of return data for each case

OpenAPI

Components:

It contains a set of reusable objects:

- schemas
- responses
- parameters, etc

Objects defined within the components object will have no effect on the API unless they are explicitly referenced

OpenAPI

Example:

```
openapi: 3.0.3
info:
  title: ping test
  version: ’1.0‘
servers:
  - url: ’http://localhost:8000/‘
paths:
  /ping:
    get:
      operationId: pingGet
        responses:
          ’201’:
            description: OK
```

OpenAPI

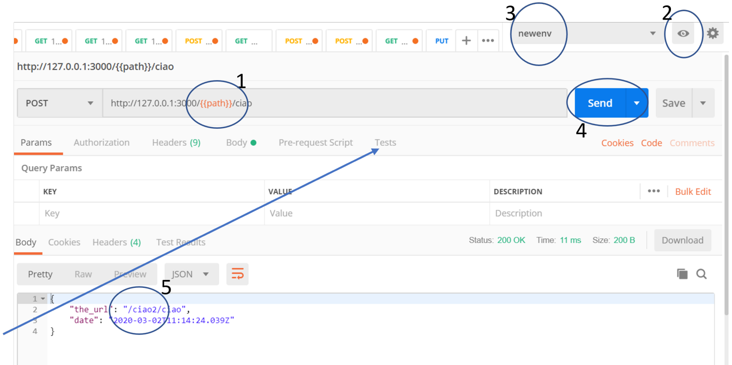
Try OpenAPI out:

- Write an OpenAPI specification
- Use openapi-generator or Swagger
  - `http://api.openapi-generator.tech/index.html`
  - `https://editor.swagger.io/`
- Download the generated code
  - for a server
  - for a client
- Implement the business logic
- Try it out: Compile, execute, etc

Postman

- Tool useful to test APIs
  - Set up GET/PUT/POST/DELETE/etc requests and see responses
  - Save requests to repeat later (History)
  - Organize requests into folders called *Collections*

- Parametrization of requests
  1. Parametrize using {..}
  2. Set up an environment
  3. Apply it
  4. Execute the request
  5. See the result
- Possible to create "tests"
- Automate with CLI



http://127.0.0.1:3000/{{path}}/ciao

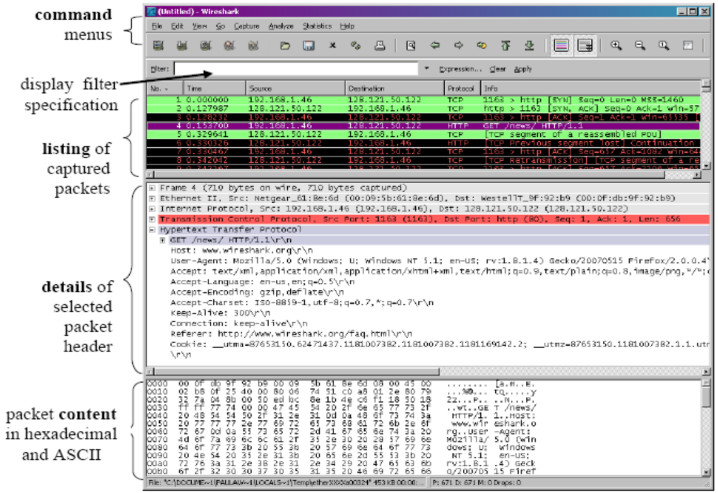
POST http://127.0.0.1:3000/{{path}}/ciao

Status: 200 OK Time: 11 ms Size: 200 B

```
{
    "the_url": "/ciao2/ciao",
    "date": "2020-03-02T11:14:24.039Z"
}
```

Wireshark

- To intercept network communication and look into it
- Useful e.g.: to verify if we are receiving data from a given server
- Do not forget to execute it as root / super-user


command menus
display filter specification
listing of captured packets
details of selected packet header
packet content in hexadecimal and ASCII
(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. -  Time        Source           Destination      Protocol  Info
   1 0.000000   192.168.1.46     128.121.50.122   TCP       1163 > http [SYN] Seq=0 Len=0 MSS=1460
   2 0.127987   128.121.50.122   192.168.1.46     TCP       http > 1163 [SYN, ACK] Seq=0 Ack=1 Win=57
   3 0.128232   192.168.1.46     128.121.50.122   TCP       1163 > http [ACK] Seq=1 Ack=1 Win=65535 [
   4 0.153700   192.168.1.46     128.121.50.122   HTTP      GET /news/ HTTP/1.1
   5 0.329641   128.121.50.122   192.168.1.46     TCP       [TCP segment of a reassembled PDU]
   6 0.330326   128.121.50.122   192.168.1.46     HTTP      [TCP Previous segment lost] Continuation
   7 0.330467   192.168.1.46     128.121.50.122   TCP       1163 > http [ACK] Seq=657 Ack=1082 Win=64
   8 0.342042   128.121.50.122   192.168.1.46     TCP       [TCP Retransmission] [TCP segment of a re
```

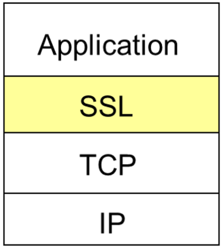
```
⊞ Frame 4 (710 bytes on wire, 710 bytes captured)
⊞ Ethernet II, Src: Netgear_61:8e:6d (00:09:5b:61:8e:6d), Dst: WestellT_9f:92:b9 (00:0f:db:9f:92:b9)
⊞ Internet Protocol, Src: 192.168.1.46 (192.168.1.46), Dst: 128.121.50.122 (128.121.50.122)
⊞ Transmission Control Protocol, Src Port: 1163 (1163), Dst Port: http (80), Seq: 1, Ack: 1, Len: 656
⊟ Hypertext Transfer Protocol
  ⊞ GET /news/ HTTP/1.1\r\n
    Host: www.wireshark.org\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Referer: http://www.wireshark.org/faq.html\r\n
    Cookie: __utma=87653150.62471437.1181007382.1181007382.1181169142.2; __utmz=87653150.1181007382.1.1.utr
    \r\n
```

```
0000  00 0f db 9f 92 b9 00 09  5b 61 8e 6d 08 00 45 00   ........ [a.m..E.
0010  02 b8 0f 25 40 00 80 06  74 51 c0 a8 01 2e 80 79   ...%@... tQ.....y
0020  32 7a 04 8b 00 50 ed bc  8e 1b 4e c6 f1 18 50 18   2z...P.. ..N...P.
0030  ff ff 77 74 00 00 47 45  54 20 2f 6e 65 77 73 2f   ..wt..GE T /news/
0040  20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1..Host:
0050  20 77 77 77 2e 77 69 72  65 73 68 61 72 6b 2e 6f    www.wir eshark.o
0060  72 67 0d 0a 55 73 65 72  2d 41 67 65 6e 74 3a 20   rg..User -Agent: 
0070  4d 6f 7a 69 6c 6c 61 2f  35 2e 30 20 28 57 69 6e   Mozilla/ 5.0 (Win
0080  64 6f 77 73 3b 20 55 3b  20 57 69 6e 64 6f 77 73   dows; U;  Windows
0090  20 4e 54 20 35 2e 31 3b  20 65 6e 2d 55 53 3b 20    NT 5.1;  en-US; 
00a0  72 76 3a 31 2e 38 2e 31  2e 34 29 20 47 65 63 6b   rv:1.8.1 .4) Geck
00b0  6f 2f 32 30 30 37 30 35  31 35 20 46 69 72 65 66   o/200705 15 Firef
```

```
File: "C:\DOCUME~1\PAULAU~1\LOCALS~1\Temp\etherXXXXa00324" 453 KB 00:00:...   P: 671 D: 671 M: 0 Drops: 0
```

Securing TCP

- Secure Socket Layer / Transport Layer Security
- First widely deployed internet security protocol
- supported since 1993 (Netscape)
- provides
  - confidentiality
  - integrity
  - end-point authentication
- SSL/TLS is at app layer: apps use TLS libraries, that “talk” to TCP



Wireshark vs TLS

General idea: ask chrome to dump the Pre_Master_Key (which is central to all TLS protocols)
Thus:

- Set up the environment variable SSLKEYLOGFILE (it defines on which file to dump all master keys)
- Start Wireshark to collect packets
- Tell Wireshark to use the key file (Preferences -> Protocols -> TLS)
- Restart chrome, and load a https web page
- Tell wireshark to “follow” a TLS session
- After you had fun, REMOVE variable SSLKEYLOGFILE!!!