Service-Oriented Architecture with Web Services

1 Introduction

How quickly can new services and applications be implemented, and at which costs? How expensive is it to change the applications later? How do you reduce maintenance costs?

Usual solutions involved group fixes, random integration approaches with accidental architectures. Architectures become unmanageable. **No Adapters**.

There was a big focus on system integration, and not business integration. System processes were inflexible and scattered. Rigid architectures and no process alignment resulted in costly maintenance.

Instead of one-to-one connection, services are used to interface the different components of the web application. Services can also be offered to other organizations.

Proper architectures use a layered system. Systems are interconnected by services → **Service Oriented Architecture**.

Services can be packaged, reused, and distributed.

Service oriented architecture (SOA) is a design discipline for applications whose parts interact by referring to each other's services. Web service is a software system designed to support interoperable machine-to-machine interaction over a network.

Web services and technologies can be used to implement service-oriented architectures.

> A service is a logical representation of a repeatable business activity that has a specified outcome, is self contained (complete in itself) and is a "black box" to consumers of the service. A service can be composed of other services. It can be considered the evolution of middleware.

A service offers an interface to consumers. Consumers can utilize multiple interfaces in their own service. They are designed to allow interoperability.

Naive view: applications are designed with top-down development. However, legacy applications cannot be ignored in the development of a bigger application.
Legacy applications impose additional technological requirements, addressed by bottom-up development.

1.1 Layers of distributed applications

Presentation (GUI)⟺Application Logic⟺Resource Management (database)

Logical layers do not impose any combination and distribution of functionality

A tier is a combination of layers in a physical system.

One Tier architecture:

Client (terminal)⟺Presentation⟺Application Logic (monolithic application)⟺Resource Management
This was very performant, but had high maintainability costs.

Two tier architecture: Client (terminal) + Presentation⟺Application Logic⟺Resource Management
- Free server from presentation issues
- Poor scalability
- Difficult to integrate with other two-tier applications (had to be done via a client)

Three tier architecture Client + Presentation⟺Application Logic (middle tier)⟺Resource Management N Tier architecture

Client (web browser) ⟺Web Server + HTML filter⟺Application Logic⟺Resource Management

Web server forms an additional tier, setting for web service architectures.

1.2 Interaction between tiers

Synchronous communication

1. A request is sent trough a service interface
2. When it arrives to the provider, a response is formed (blocking operation)
3. Only after the response is received, the thread is resumed.

Relatively simple architecture, but the calling and the called are tightly coupled. There is poor performance if processing takes too much time or multiple tiers.

Asynchronous communication

- Calling and called are loosely coupled
- Intermediaries can store and process messages
- Potentially more complex due to the need to synchronize

2 Service Composition

Service composition is born from architectural needs. Realistic services are made of smaller, elemental (atomic), reusable ones (composition). Composition can be seen as a workflow (process), in which each task is performed by a web service. The workflow describes how the services are composed and **how they work together**.

> Web services are invoked and not compiled and linked together as libraries.

Atomic services remain separate from the composite service to promote reusability. The composition specifies the services to be invoked, the order and the conditions.

SOA architectures have better support for business processes.

2.1 Choreography

With choreography (implicit support), a collection of services knows what they must do in order to work together and implement a certain service. The choreographer informs the dancers (components) on what to do beforehand. The process definition is explicit.

The public process is the one everyone sees. The private process is the one that is given to the implementer.

2.2 Orchestration

With orchestration (explicit support), a service is defined to implement each process and orchestrate other services in order to make sure the process is correctly performed. The maestro tells the musicians (components) how to play their parts.

Either orchestration and composition can be used when defining a service architecture BPMN is a process modelling language.

3 Docker

Services must be able to be hosted and ran on the cloud.

3.1 Cloud Computing

Cloud computing is a model for computing in which something is done “in the cloud”. The cloud is an iconic representation of the internet.
SOA was introduced as an architectural solution to enable cloud computing. It allows resources to be

leveraged trough the internet as services in order to control IT costs and make IT more agile. This way, companies can focus on their core business.

- Software as a service
- Platform as a service
- Infrastructure as a service: a virtual machine
- AI as a service

3.2 Containerization

Offers a location and operating system-independent runtime environment to applications. Applications are ran in these containers. It doesn't matter where the application is ran (web, cloud, local device).

Nowadays, an application can be ran on multiple devices. They should also be ran anytime and anywhere. Also, companies no longer want their own server room.

Instead of having one big application on the server, smaller services are hosted on the cloud.

Virtual machine VS container: the VM has more overhead, a container is just the environment where an application can be ran. Docker is a containerization engine.

Users can fully commit to containerization without having to worry about the engine used (currently docker).

3.3 Docker

Docker is the most popular container technology available today.

An image describes a container. Images can be pulled from a docker hub. The daemon exposes an API, and the CLI communicates with the API. the CLI manages networking, data volumes, containers and images.

An image is a readonly template with instructions for creating a container. A container is a runnable instance of an image. A container can be started, stopped, moved or deleted using the API or CLI. A container can be connected to one or more networks, and storage can be attached to it. A container is defined by its image and configuration options. A registry is used to store and distribute Images. The daemon listens for docker API requests and manages objects. It can also communicate with other daemons in clusters.

The bridge network is giving port exposure at `8080:80`.

3.4 Docker Storage

Data stored in a container is lost when the container stops, since the image is immutable.

- Temporary mount: used for caching
- Bind mount: connects to the host's filesystem
- Volume: docker manages the storage of the container

3.5 Docker in Practice

- Dockerfile: Describes the image/blueprint
- Docker image: snapshot of the container
- Docker repository:

the container runs the compiled java file defined in the image

In production, make the container as small as possible, with no root and non-shell. Always specify the version, never use `:latest`. Always use a repository proxy that allows caching. This way, when dockerHub is down, your infrastructure won't fail.

Solutions have been developed to support clusters of containers to facilitate their configuration and management.

https://docs.docker.com/get-started/docker_cheatsheet.pdf

4 Communication Protocols

Technical interoperability technical connection to communicate and share information. They are project independent

Syntactic interoperability coding and structure of messages (data formats: JSON, XML)

Semantic interoperability meaning, common understanding of the messages

Before SOA there were proprietary protocols (RPC, CORBA). With SOA 1.0 internet standards are used: XML, SOAP. With SOA 2.0 there are lightweight solutions: JSON, JSON schema, REST, JSON-API and GraphQL

4.1 Data formats

They are the foundation of syntactic interoperability.

XML Documents that conform with XML grammar rules

XSD XSD files define the structure of XML files. They can also be used as a blueprint for java classes. JAXB facilitates the marshalling (java object to XML document) and un-marshalling (XML to java object). Used to validate information from other services. An XML schema defines the structure of an XML document, and the rules for data content (what fields an element can contain, which sub elements it can contain, and how many items can be present).

XPath can be used to query XML files.

JSON Lightweight, text-based language independent data. It's a human readable interchange format.

YAML Has less overhead than XML, and consists of key-value pairs. YAML files are more flexible than json files.

4.2 REST

Representation State Transfer is an **architectural style** for invoking services over the internet (list of styles and agreements).

The rest principles are:

- Stateless client-server architecture: request messages are self contained. All the necessary information is in the request message.
- Web services are viewed as resources identifiable by their URIs: URIs offer a global addressing space for services.
- Web service clients and providers choose a representation to send application content to each other (XML, JSON,...). Client and provider have a mutual understanding of the meaning of data since there is no formal way to describe web service interfaces.

Within rest, there are a set fo remote methods that describe the actions to be performed on the resource: CRUD (Create, Read, Update, Delete) actions.

> With REST, the state of an object is changed, and then the whole object is sent (update operation).

4.3 Rest URI design guidelines

1. URIs must point to resources
2. Relationships must be exposed in the URI
3. Different URIs are used for instances and collections
4. Allow filtering

Crud actions are related to HTTP methods. With PATCH you can only send the field that you want to change.

No additional protocol information is needed.

4.4 GraphQL

REST is static in nature, with GraphQL you don't request all the data to the server, just the information you need (avoid over and under-fetching). A server-side runtime executes queries by using a type system defined for the data that is being requested.

Possible operations are:

- Query: retrieve data;
- Mutation: adapt data;
- Subscription: be notified of changes in data

GraphIQL is used to play with GraphQL APIs.

4.5 OpenAPI

JSON:API is a specification on how a client should request resources and how a server should respond to those requests. It standardizes implementation of RESTful APIs and their requests/responses.

OpenAPI specification (OAS) is a language agnostic interface to discover RESTful APIs that allows humans and computers to discover and understand the capabilities of the service without access to source code or documentation. The consumers can understand and interact with the remote service with a minimal amount of implementation logic.

SWAGGER is an UI for displaying APIs according to OAS (also has an editor and codegen).

HATEOAS is an alternative to OpenAPI. Hypermedia is used as the engine of application state (the state is transferred trough hypermedia links).

POJO: plain old java object Model or Data transfer object (DTO) Database access object (DAO), use mock data