

Middleware Architectures 1

Lecture 2: Service Architecture

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • https://vitvar.com



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • https://vitvar.com/lectures



Modified: Sun Oct 05 2025, 20:07:12
Humla v1.0

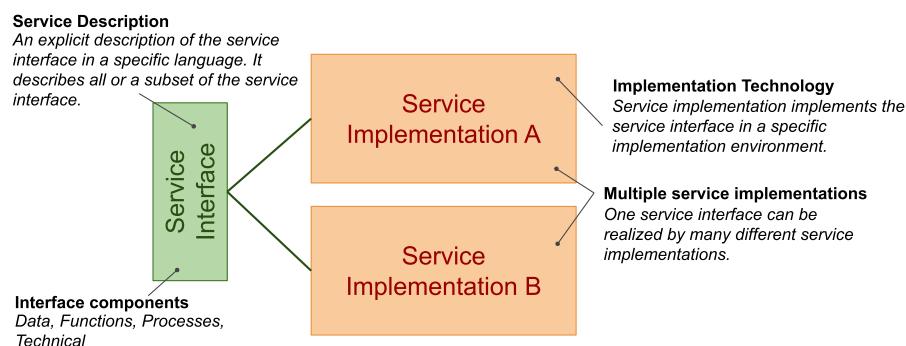
Overview

- Service Definition
- Integrating Applications
- Integration Patterns

Service Views

- Business view
 - A service realizes an effect that brings a business value to a service consumer
→ for example, to pay for and deliver a book
- Conceptual view
 - reusability, loose coupling, contracting, discoverability
- Logical view
 - service interface, description and implementation
 - RPC-style and resource-oriented
- Software architecture view
 - business service (also application service)
→ external, exposed functionality of an application
 - middleware service
→ internal/technical, supports processing of requests
- Technology architecture view
 - REST/RESTful, GraphQL
 - XML-RPC/SOAP, RMI, gRPC
 - WebSocket, WebRTC, SSE

Interface, Description and Implementation



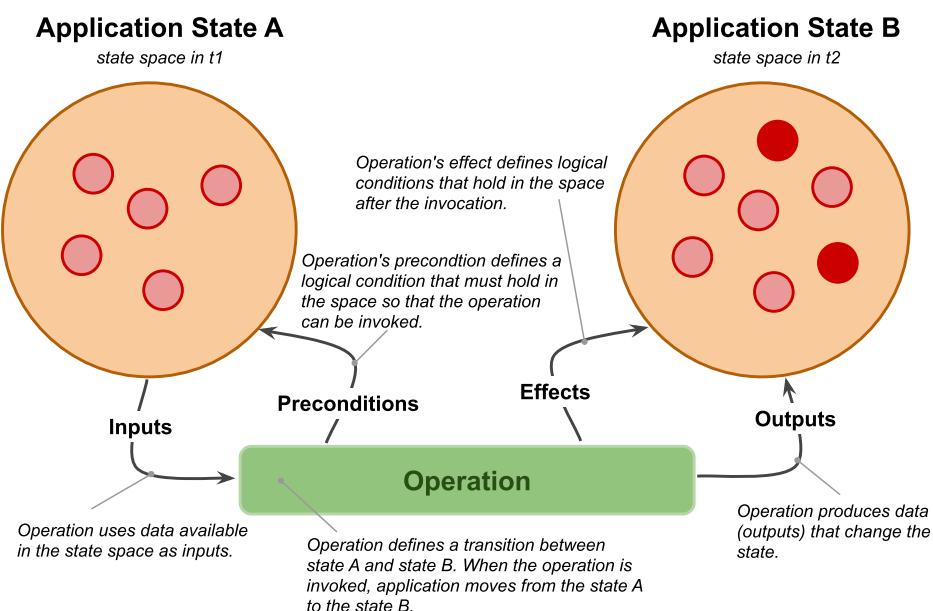
- Terminology clarification
 - service ~ service interface + service implementation
 - REST/RESTful service ~ service interface that conforms to REST architectural style and HTTP protocol
 - Microservice ~ a set of services that realize an app's capability
 - Kubernetes service ~ a service that routes traffic to a set of pods

Service Interface

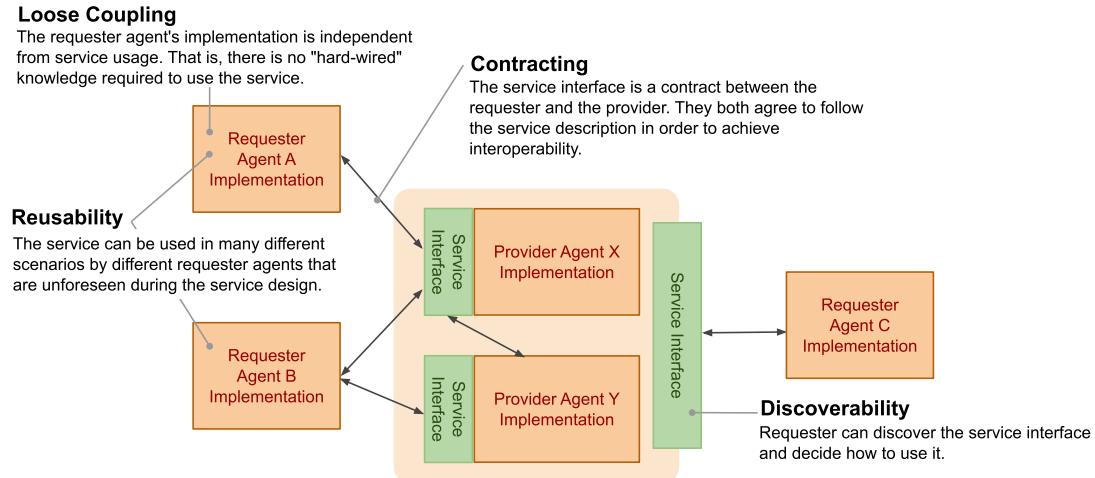
- Service interface components
 - Data
 - Data model definition used by the service
 - for example, input and output messages, representation of resources
 - Functions
 - operations and input and output data used by operations
 - Process
 - public process: how to consume the service's functionality
 - Technical
 - security, usage aspects (SLA-Service Level Agreement)
 - other technical details such as IP addresses, ports, protocols, etc.

Public Process

- A state diagram
 - operation of a service defines a **state transition** between two states.



Service Characteristics



Overview

- Service Definition
- Integrating Applications
- Integration Patterns

Integration and Interoperability

- Integration
 - *A process of connecting applications so that they can exchange and share capabilities, that is — information and functionalities.*
 - *Includes methodological approaches as well as technologies*
- Interoperability
 - *Ability of two or more applications to understand each other*
 - *Interoperability levels*
 - *Data – syntax/structure and semantics*
 - *Functions/Processes – syntax and semantics*
 - *Technical aspects – protocols, network addresses, etc.*

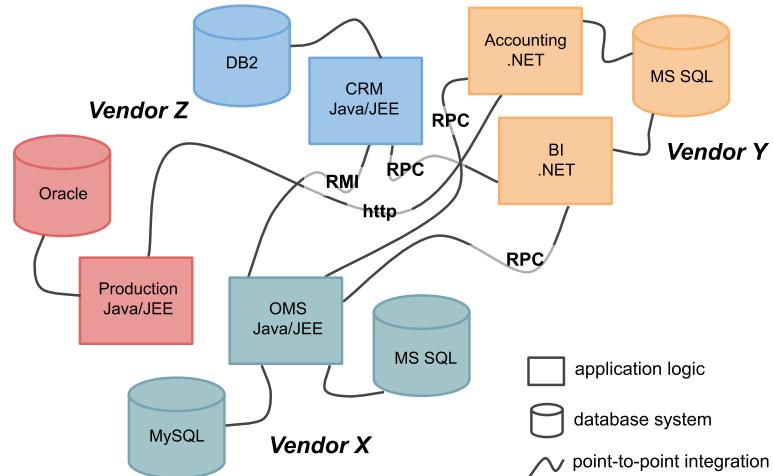
Service Oriented Architecture



- SOA supports two core business strategies
 - *Growing top-line revenue*
 - *Enterprise reacts quickly to requirements from the market*
 - *Business processes can be reconfigured rather than reimplemented*
 - *Improving bottom-line profit*
 - *Saving development costs by resuing existing services*
- Pre-integrated solutions
 - *Out-of-the-box applications and integration solutions among them*

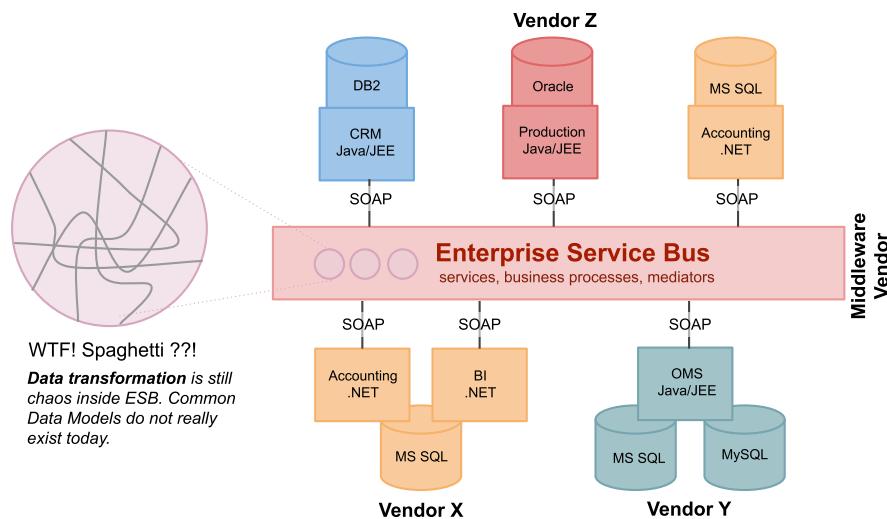
One-to-One Service Integration

- Direct integration of applications
 - *Multiple protocols problem, multiple vendor problem*
 - *Replication of integration functionalities such as interoperability solutions*

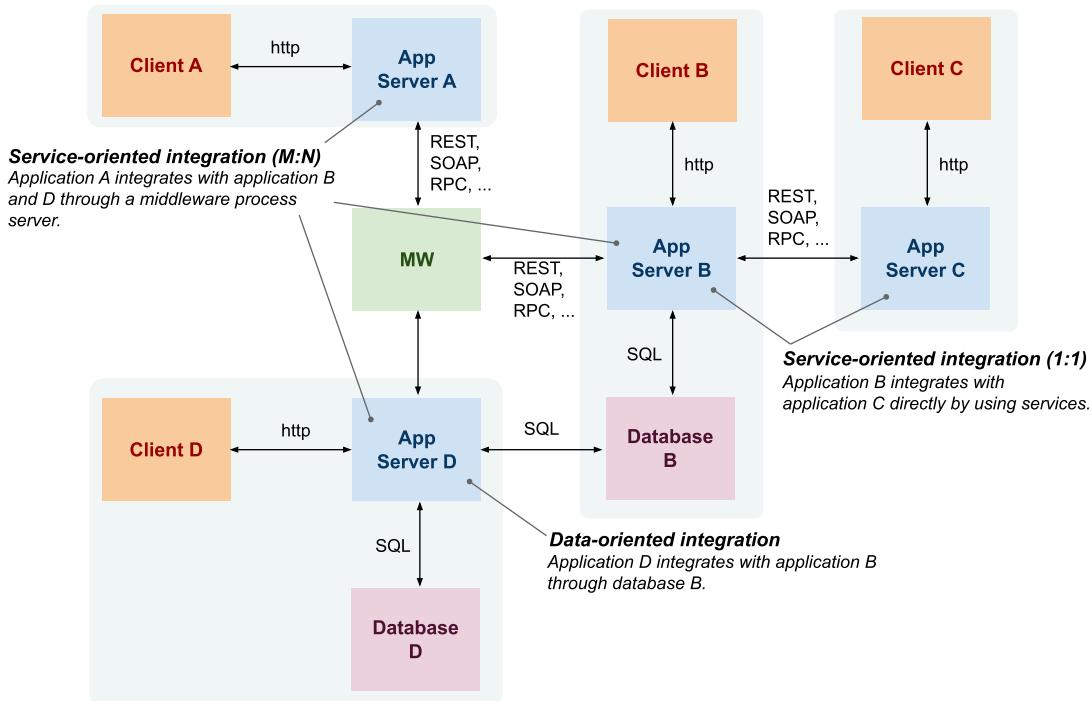


Many-to-Many Service Integration

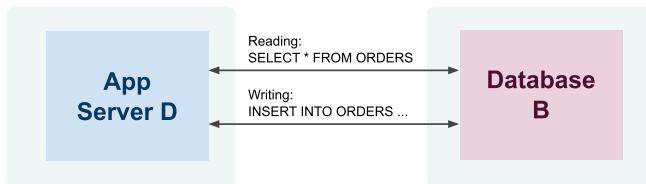
- Enterprise Service Bus – central integration technology
 - *Realizes so called Service Oriented Architecture (SOA)*
 - *Contains various integration components such as process server, mediators, messaging middleware, identity management, etc.*



Integration Approaches Overview

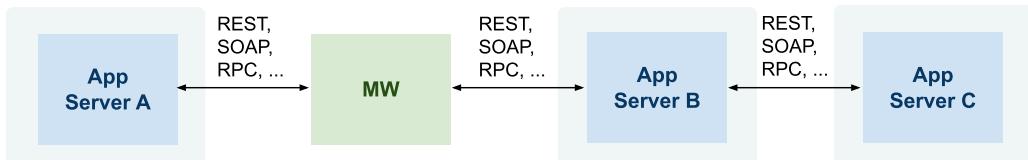


Data-oriented Integration



- Third-party database access
 - Application D accesses a database of application B directly by using SQL and a knowledge of database B structure and constraints
 - In the past: monolithic and two-tier client/server architectures
 - Today: ETL (Extract, Transform, Load) technologies
- Problems
 - App D must understand complex structures and constraints
 - Data – very complex, includes structure and integrity constraints
 - Functions/processes – hidden in integrity constraints
 - Technical – access mechanisms can vary

Service-oriented Integration



- Integration at the application layer
 - Application exposes services that other applications consume
 - Services hide implementation details but only define interfaces for integration
- Problems
 - Can become unmanageable if not properly designed
 - Interoperability
 - Data – limited to input and output messages only
 - Functions/processes – limited to semantics of services
 - Technical – access mechanisms can vary

Integration and Types of Data

- Real-time data – Web services
 - Service-oriented integration
 - online, realtime communication between a client and a service
 - Usually small data and small amount of service invocation in a process
- Bulk data – ETL
 - Data-oriented integration
 - processing of large amount of data in batches
 - Sometimes required for reconciliation across apps
 - when real-time integration fails and there is poor error handling
- **SOA provides both Web service and ETL capabilities**

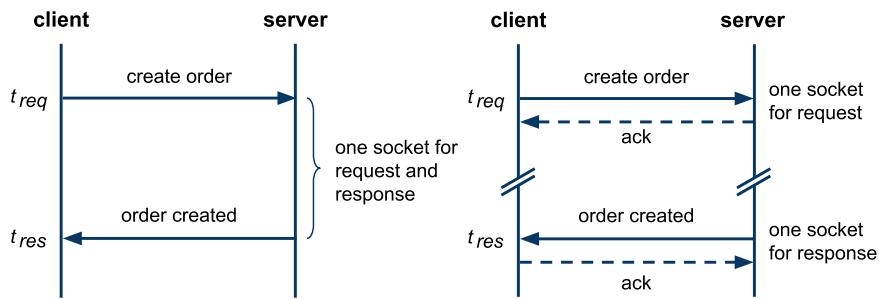
Enterprise Service Bus

- ESB is a central intermediary in SOA
 - *Types of services: shared and infrastructure*
 - *Types of processes: Technical and Business*
- ESB Application
 - *Application running on an application server*
 - *Exposes functionality via Web service interface*
 - *Allows to communicate with various messaging protocols*
- Middleware Integration Patterns
 - *Technical-level interoperability – message broker*
 - *Location transparency*
 - *Dynamic routing*
 - *Session pooling*
 - *Message enrichment*
 - *Data transformation*
 - *Key mapping*

Overview

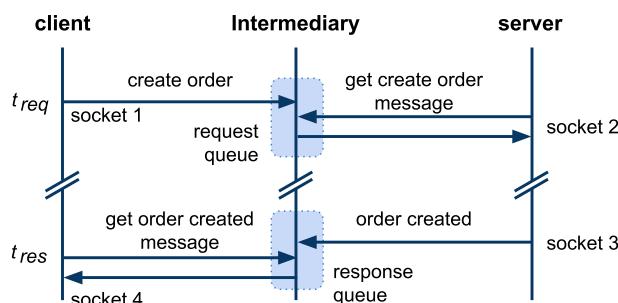
- Service Definition
- Integrating Applications
- Integration Patterns
 - *Synchronous and Asynchronous Integration*
 - *Microservices Architecture*

Synchronous and Asynchronous Integration



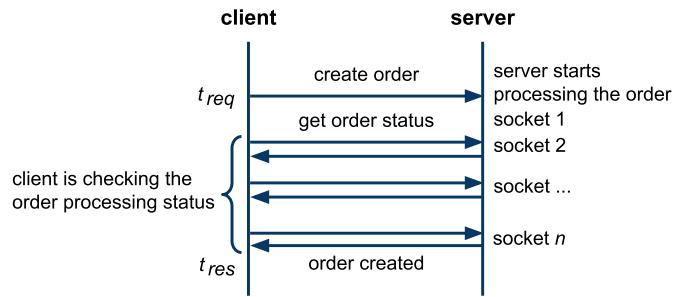
- **Synchronous**
 - one socket, $|t_{req} - t_{res}|$ is small
 - easy to implement and deploy, only standard firewall config
 - only the server defines endpoint
- **Asynchronous**
 - request, response each has socket, client and server define endpoints
 - $|t_{req} - t_{res}|$ can be large (hours, even days)
 - harder to do across network elements (private/public networks issue)

Asynchronous via Intermediary



- **Intermediary**
 - A component that decouples a client-server communication
 - It increases reliability and performance
 - The server may not be available when a client sends a request
 - There can be multiple servers that can handle the request
- **Further Concepts**
 - Message Queues (MQ) – queue-based communication
 - Publish/Subscribe (P/S) – event-driven communication

Asynchronous via Polling



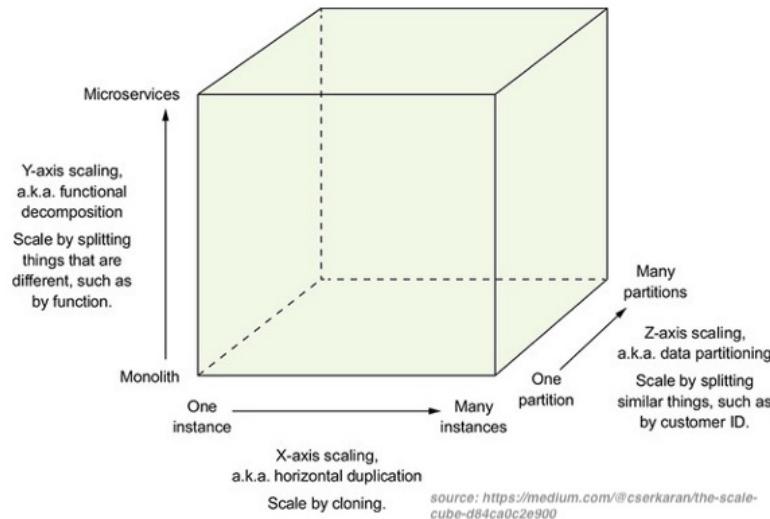
- Polling – only clients open sockets
 - *A client performs multiple request-response interactions*
 - *The first interaction initiates a process on the server*
 - *Subsequent interactions check for the processing status*
 - *The last interaction retrieves the processing result*
- Properties of environments
 - *A server cannot open a socket with the client (network restrictions)*
 - *Typically on the Web (a client runs in a browser)*

Overview

- Service Definition
- Integrating Applications
- Integration Patterns
 - *Synchronous and Asynchronous Integration*
 - *Microservices Architecture*

The Scale Cube

- Three-dimensional scalability model
 - *X-Axis scaling requests across multiple instances*
 - ***Y-Axis scaling decomposes an application into micro-services***
 - *Z-Axis scaling requests across "data partitioned" instances*



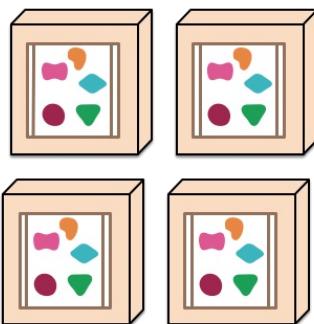
Overview

- Emerging software architecture
 - *monolithic vs. decoupled applications*
 - *applications as independently deployable services*

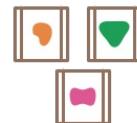
A monolithic application puts all its functionality into a single process...



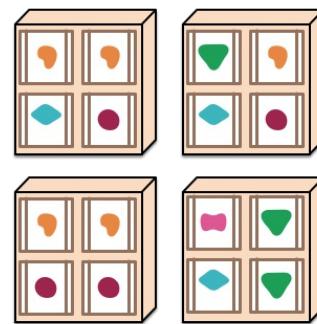
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Major Characteristics

- Loosely coupled
 - *Integrated using well-defined interfaces*
- Technology-agnostic protocols
 - *HTTP, they use REST architecture*
- Independently deployable and easy to replace
 - *A change in small part requires to redeploy only that part*
- Organized around capabilities
 - *such as accounting, billing, recommendation, etc.*
- Implemented using different technologies
 - *polyglot – programming languages, databases*
- Owned by a small team