Middleware and Web Services

Lecture 2: Service Architecture

doc. Ing. Tomáš Vitvar, Ph.D.
tomas@vitvar.com • @TomasVitvar • http://vitvar.com

Czech Technical University in Prague
Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/mdw

Modified: Tue Oct 08 2019, 07:33:08
Humla v0.3

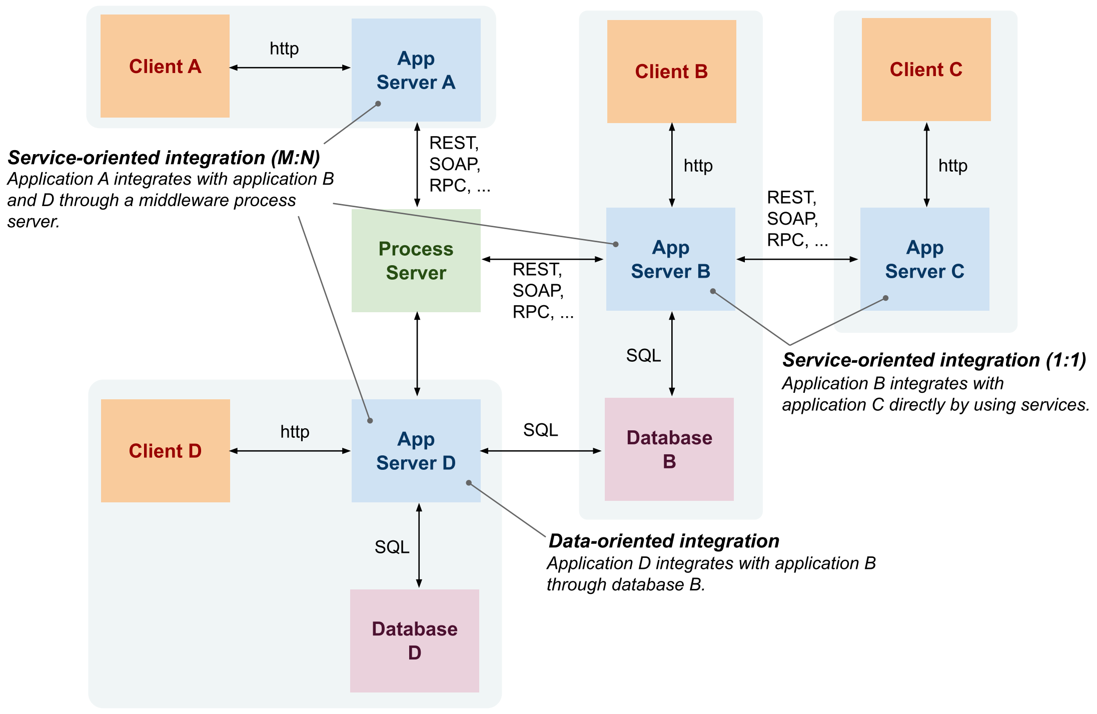
Overview

- Integrating Applications
- Service Definition
- Service Communication

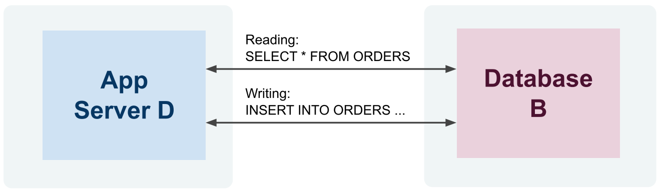
Integration and Interoperability

- 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.*

Integration Approaches Overview

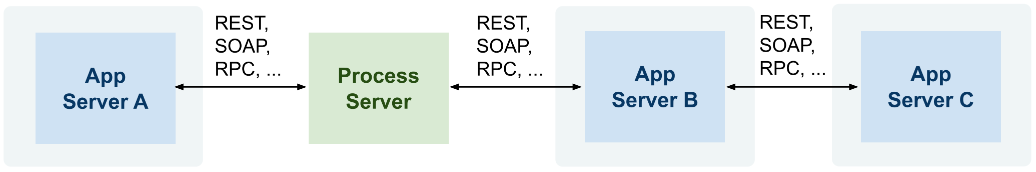

Client A
http
App Server A
REST, SOAP, RPC, ...
Client B
http
Client C
http
REST, SOAP, RPC, ...
App Server B
App Server C
Process Server
REST, SOAP, RPC, ...
SQL
Client D
http
App Server D
SQL
Database B
SQL
Database D
Service-oriented integration (M:N)
Application A integrates with application B and D through a middleware process server.
Service-oriented integration (1:1)
Application B integrates with application C directly by using services.
Data-oriented integration
Application D integrates with application B through database B.

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*

    $\rightarrow$ *when real-time integration fails and there is poor error handling*
- **SOA provides both Web service and ETL capabilities**

Overview

- Integrating Applications
- Service Definition
- Service Communication

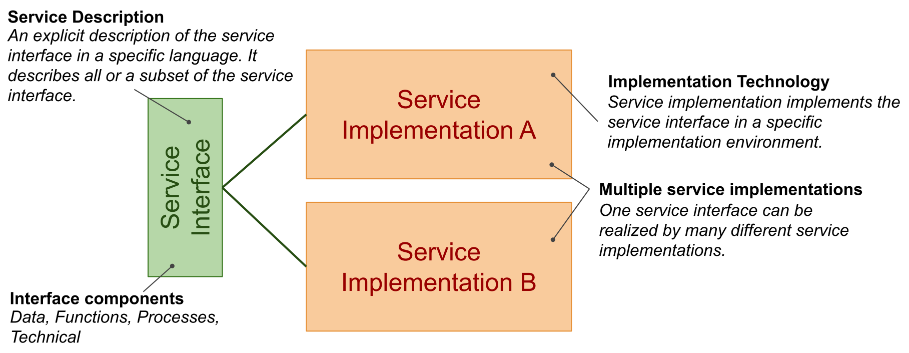
Web Service Architecture

- Web Service Architecture
  - *Defined by W3C in Web Service Architecture Working Group Note*
  - *Defines **views***
    - *→ message-oriented view (WSDL and SOAP)*
    - *→ resource-oriented view (REST and HTTP)*
  - *Defines **architecture entities** and their **interactions***
    - *→ Abstraction over underlying technology*
    - *→ Basis for service usage processes and description languages*
- Service Oriented Architecture
  - *Collection of tools, methods and technologies*
  - *There is some implicit understanding of SOA in the community such as*
    - *→ SOA provides advances over Enterprise Application Integration*
    - *→ SOA is realized by using SOAP, WSDL, (and UDDI) technologies*
    - *→ SOA utilizes Enterprise Service Bus (ESB)*
  - *⇒ ~ a realization of Web Service Architecture message-oriented view*

Service

- Difficult to agree on one definition
- Business definition
  - *A service realizes an effect that brings a business value to a service consumer*
    - *→ for example, to pay for and deliver a book*
- Conceptual definition
  - *service characteristics*
    - *→ encapsulation, reusability, loose coupling, contracting, abstraction, discoverability, composability*
- Logical definition
  - *service interface, description and implementation*
  - *service usage process*
    - *→ service use tasks, service types*
- Architectural definition
  - *business service (also application service)*
    - *→ external, exposed functionality of an application*
  - *infrastructure service*
    - *→ internal/technical, supports processing of requests*

Interface, Description and Implementation


Service Description
An explicit description of the service interface in a specific language. It describes all or a subset of the service interface.
Service Interface
Service Implementation A
Service Implementation B
Implementation Technology
Service implementation implements the service interface in a specific implementation environment.
Multiple service implementations
One service interface can be realized by many different service implementations.
Interface components
Data, Functions, Processes, Technical

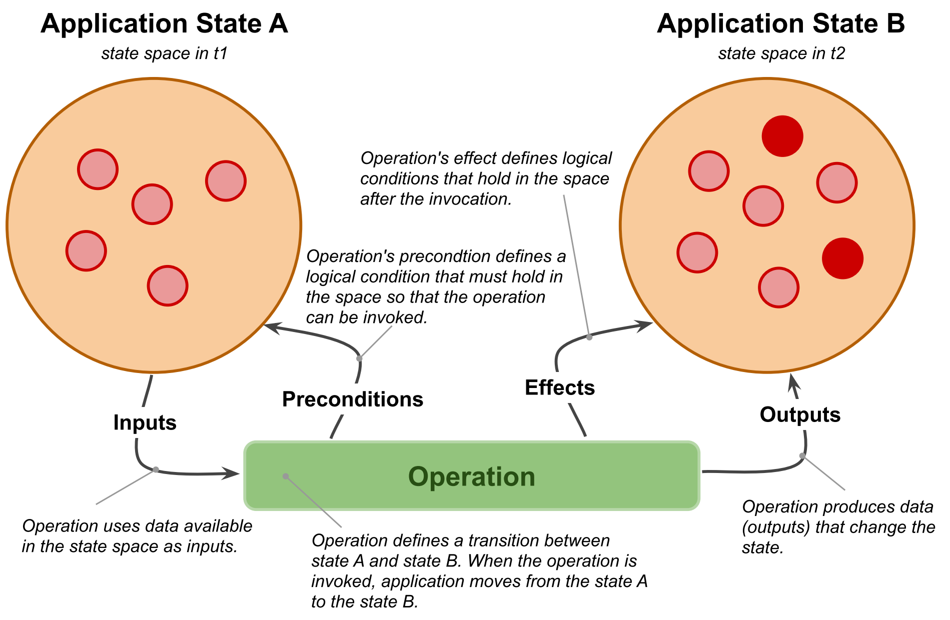
- Terminology clarification
  - *service ~ service interface + service implementation*
  - *WSDL service ~ service description in WSDL language*
  - *SOAP service ~ a service interface is possible to access through SOAP protocol; there is a WSDL description usually available too.*
  - *REST/RESTful service ~ service interface that conforms to REST architectural style and HTTP protocol*

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*
    - → *orchestration: realization of the service's functionality by its implementation*
  - *Technical*
    - → *security, usage aspects (SLA-Service Level Agreement)*
    - → *other technical details such as IP addresses, ports, protocols, etc.*

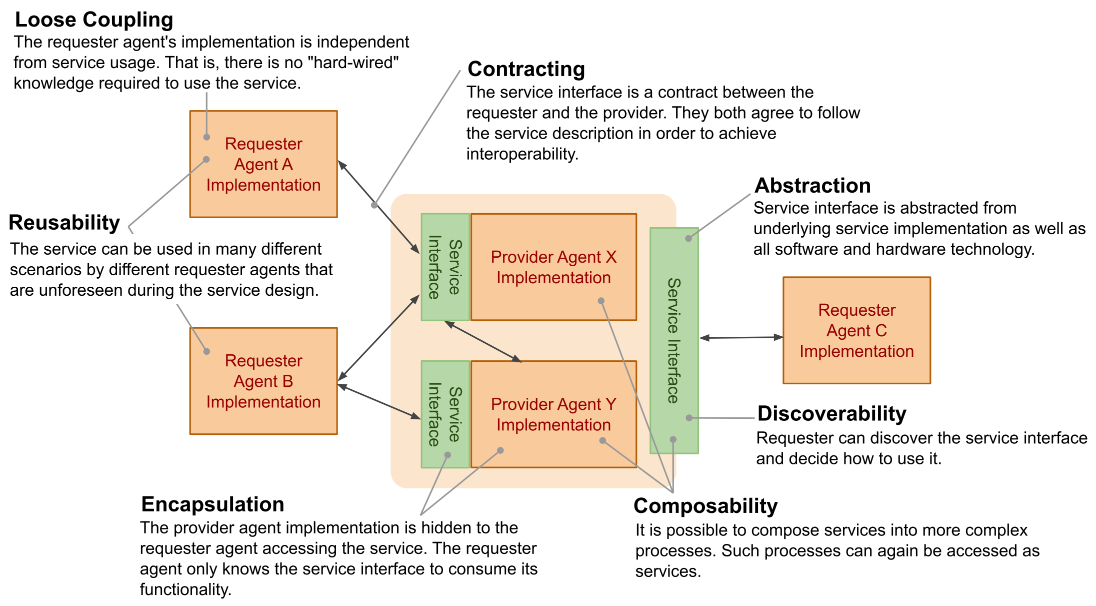
Public Process

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



Application State A
state space in t1
Application State B
state space in t2
Operation's effect defines logical conditions that hold in the space after the invocation.
Operation's precondtion defines a logical condition that must hold in the space so that the operation can be invoked.
Preconditions
Effects
Inputs
Outputs
Operation
Operation uses data available in the state space as inputs.
Operation defines a transition between state A and state B. When the operation is invoked, application moves from the state A to the state B.
Operation produces data (outputs) that change the state.

Service Characteristics



Requester Agent A Implementation
Requester Agent B Implementation
Requester Agent C Implementation
Service Interface
Provider Agent X Implementation
Service Interface
Provider Agent Y Implementation
Service Interface

Loose Coupling
The requester agent's implementation is independent from service usage. That is, there is no "hard-wired" knowledge required to use the service.

Contracting
The service interface is a contract between the requester and the provider. They both agree to follow the service description in order to achieve interoperability.

Reusability
The service can be used in many different scenarios by different requester agents that are unforeseen during the service design.

Abstraction
Service interface is abstracted from underlying service implementation as well as all software and hardware technology.

Discoverability
Requester can discover the service interface and decide how to use it.

Encapsulation
The provider agent implementation is hidden to the requester agent accessing the service. The requester agent only knows the service interface to consume its functionality.

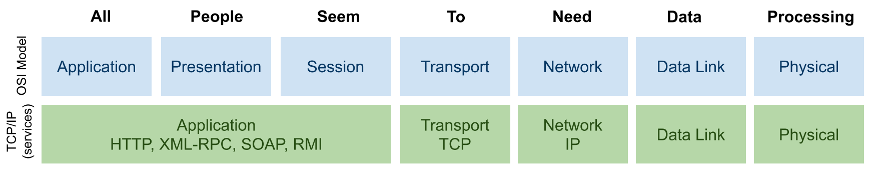
Composability
It is possible to compose services into more complex processes. Such processes can again be accessed as services.

Overview

- Integrating Applications
- Service Definition
- Service Communication

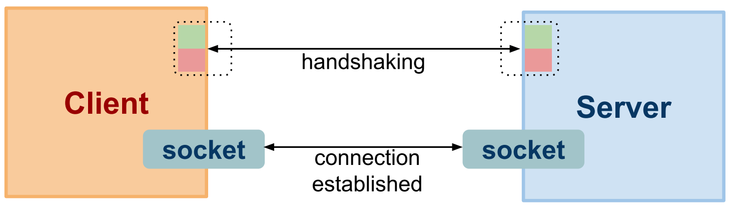
Application Protocols

- Remember this



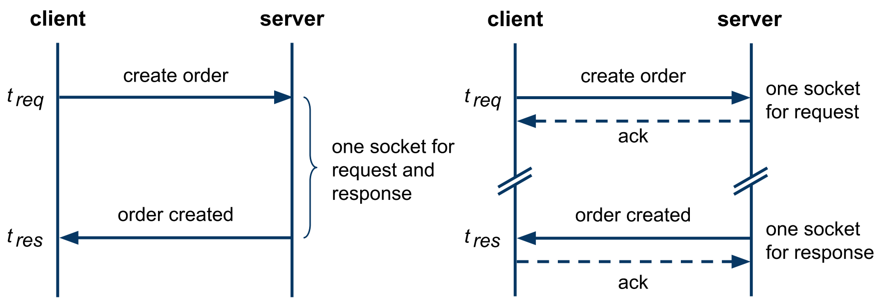
- App protocols mostly on top of the TCP Layer
  – *use TCP socket for communication*
- Major protocols
  – *HTTP – most of the app protocols layered on HTTP*
    → wide spread, but: implementors often break HTTP semantics
  – *RMI – Remote Method Invocation*
    → Java-specific, rather interface
    → may use HTTP underneath (among other things)
  – *XML-RPC – Remote Procedure Call and SOAP*
    → Again, HTTP underneath
  – *WebSocket – new protocol part of HTML5*

Socket



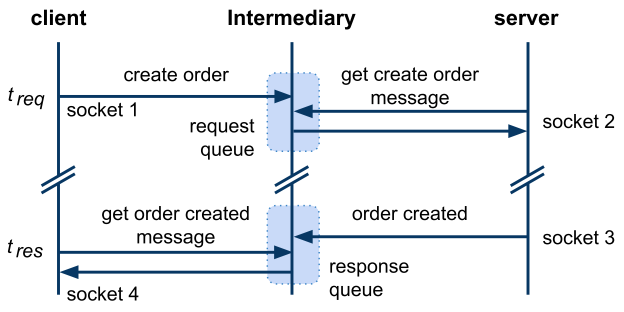
- Handshaking (connection establishment)
  - *The server listens at* `[dst_ip,dsp_port]`
  - *Three-way handshake:*
    - → *the client at* `[src_ip,src_port]` *sends a connection request*
    - → *the server responds*
    - → *the client acknowledges the response, can send data along*
  - *Result is a socket (virtual communication channel) with unique identification:* `socket=[src_ip,src_port;dst_ip,dst_port]`
- Data transfer (resource usage)
  - *Client/server writes/reads data to/from the socket*
  - *TCP features: reliable delivery, correct order of packets, flow control*
- Connection close

Synchronous and Asynchronous Communication



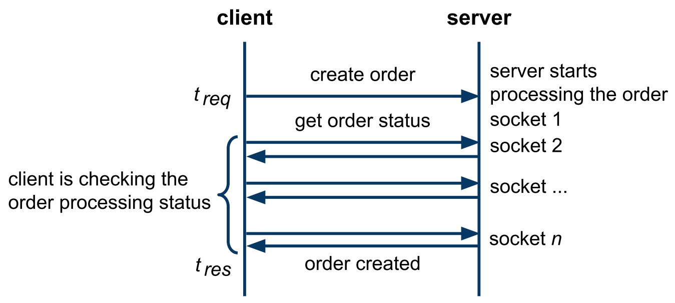
- Synchronous
  - *one socket,* $|t_{req} - t_{res}|$ *is small*
  - *easy to implement and deploy, only standard firewall config*
  - *only the server defines endpoint*
- Asynchronous
  - *request, response each has socket, client and server define endpoints*
  - $|t_{req} - t_{res}|$ *can be large (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*

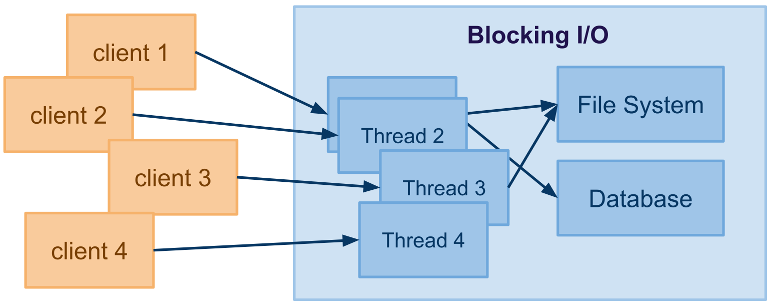
Asynchronous via Polling



- 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)*

Blocking I/O Model

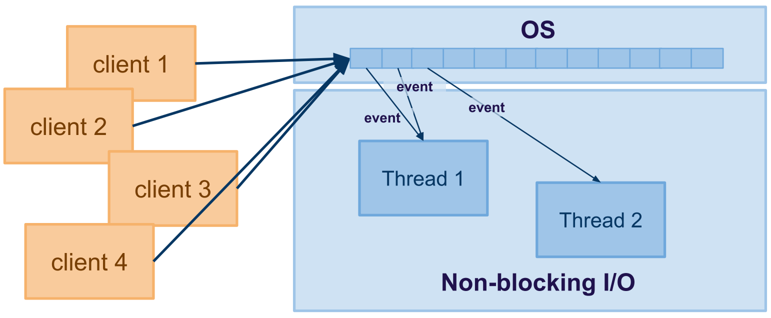
- The server creates a thread for every connection
  - *For example, 1K connections = 1K threads, big overhead*



- Characteristics
  - *the thread is reserved for the connection*
  - *When processing of the request requires other interactions with DB/FS or network communication is slow*
    → scales very bad as the thread's execution is "blocked"

Non-Blocking I/O Model

- Connections maintained by the OS, not the Web app
  - *The Web app registers events, OS triggers events when occur*



- Characteristics
  - *Event examples: new connection, read, write, closed*
  - *The app may create working threads and controls their number*
    → less number of working threads as opposed to blocking I/O
  - *On the outbound calls, there can still be blocking I/O*
    → this depends on the implementation framework