Integration of Applications, Web Services

Petr Aubrecht (Martin Ledvinka)

aubrecht@asoftware.cz

Winter Term 2022

Contents

Cooperating Programs – Why?

What's Wrong – Reliability

- Business loses millions of dollars every minute the server is down.
- Have you ever tried to run server? How much downtime did you have?
- Critical systems need 99.999 % reliability = 5 minutes/year.
- Examples of failure: "České spořitelně v sobotu několik hodin nefungovalo internetové bankovnictví."
- Amazon cloud 2017: `https://en.wikipedia.org/wiki/Timeline_of_Amazon_Web_Services#Amazon_Web_Services_outages`
- Solution: Backup systems
- Problem: double/triple price, same performance

What's Wrong – Scaling

- Hardware doesn't scale well
- RAM scaling:
  - 16 GB CZK 1.839
  - 32 GB CZK 4.819
  - 64 GB CZK 12.090
  - 128 GB is the highest capacity of RAM module available for enterprise (DDR 4) cost $4,054.93, 6 RAM slots, e.g. 768 GB/ per machine
  - 1 TB ??? How? Mainframe? Great for very rich customers.3-
- The same problem is with disks (RAID helps a bit), CPU...

Solution – Horizontal Scaling

- Let's use backup system to cooperate on processing data!
- Let's have multiple **cheap** computers, where price of 1 TB RAM $= 16 \times 64$ GB, CZK 193.440 (compare to 128 GB, \$4.000)
- Similar approach as RAID (Redundant Array of Inexpensive Disks)
- How to distribute the tasks?

Distributed Systems

- Distributed (fault tolerant) systems
  - Able to process requests concurrently
  - Scalable
  - Can handle faults, only decrease performance
- Caveats
  - Less predictable
  - More complex
  - More difficult to secure
  - Effort to manage the system

Approaches

File

- Applications exchange data by writing into a shared file
- Pipeline processing
- $\Rightarrow$ Local system
- Problems: format, schema, scalability, concurrency, notifications

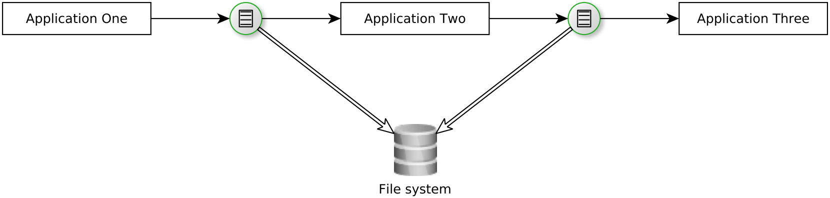


Figure: Application pipeline diagram.

Database

- Applications share database, possibly use different views of the same database
- No integration layer needed, application data always up to date
- Problems: schema (general or complex), schema evolution, notifications

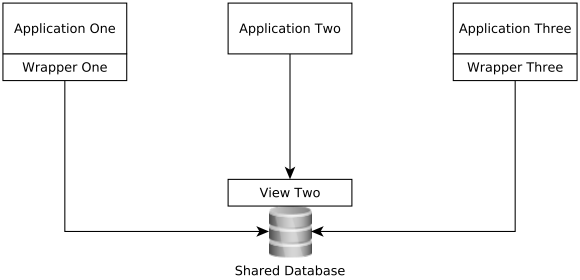


Figure: Applications using shared database.

Java RMI

- *Remote Method Invocation*
- Object-oriented equivalent of *remote procedure call* (see later)
- Java-specific technology for distributed systems
- Java Remote Method Protocol
  - Wire-level protocol (application layer) on top of TCP
  - Binary
- RMI supports primitive types and `Serializable`

Java RMI

- Client invokes methods of a *remote interface* on a local *stub*
  - Stub is a RMI-generated *proxy* object representing the remote implementation
- Server implements *remote interface* to export methods which can be called remotely
- RMI registry
  - Server registers at RMI registry as a provider of remote objects
  - Client uses RMI registry to look up remote objects

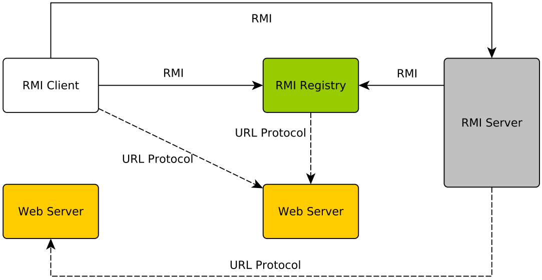


Figure: Schema of Java RMI components.

RMI Alternatives

Similar technologies exist for

- Python – *RPyC*
- Ruby – *Distributed Ruby*
- Erlang – built into the language itself

RPC

- *Remote Procedure Call*
- Invocation of subroutine in a different address space (usually a different computer)
- Client-server architecture
- Typically synchronous

XML-RPC

- Standard for remote procedure call using XML as message format
- Platform independent
- Over HTTP

XML-RPC Example

Request

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><int>41</int></value>
    </param>
  </params>
</methodCall>
```

Response

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

XML-RPC – Try it Yourself

1. Download/clone a simplistic XML-RPC server implementation from `https://gitlab.fel.cvut.cz/ear/xmlrpcserver`
2. Start the server using `mvn package exec:java`
3. Open Postman or other HTTP client
4. Send a POST request to `http://localhost:8080` with body

```xml
<?xml version="1.0"?>
<methodCall>
  <methodName>EarServer.hello</methodName>
  <params>
    <param>
      <value><string>Master Chief</string></value>
    </param>
  </params>
</methodCall>
```

CORBA

- *Common Object Request Broker Architecture*
- OMG standard for language and platform-independent distributed computing architecture
- Similar to RPC but object-oriented
- Transparent location – client is unaware whether invocation is local or remote
  - Also a caveat – local invocation cannot be optimized and has to go through the whole ORB machinery
- Standards for interface definition, communication protocols, location

CORBA – Concepts

Interface Definition Language (IDL)

- Standardized language for specification of interface provided by an object
- Mappings for IDL exist in all major programming languages
- Used to generate Stub/Skeleton code

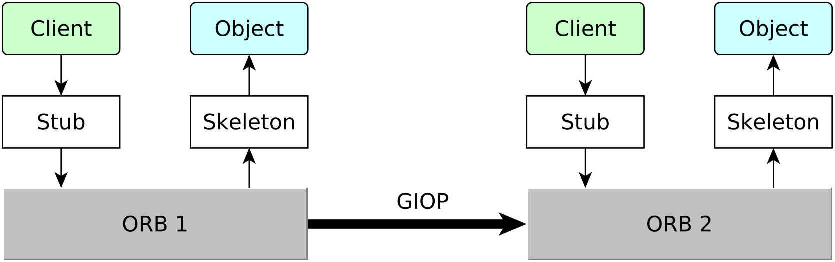
Object Request Broker (ORB)

- Middleware allowing transparent local and remote invocation
- Handles data serialization/deserialization based on IDL
- Knows location of the actual service implementation
- Is able to handle, e.g., transactions

CORBA – Concepts

General InterORB Protocol – GIOP

- Protocol for communications between ORBs
- Best known (and most often used) is IIOP (Internet InterORB Protocol) which uses TCP/IP
- Other versions exist, e.g., HTIOP, SSLIOP



CORBA – IDL Interface Example

```
module HelloApp {
  interface Hello {
  string sayHello();
  oneway void shutdown();
  };
};
```

CORBA – Java Implementation Example

```
class HelloImpl extends HelloPOA {
 private ORB orb;

 public void setORB(ORB orb_val) {
   orb = orb_val;
  }

 public String sayHello() {
   return "\nHello world !!\n";
  }

 public void shutdown() {
   orb.shutdown(false);
  }
}
```

What is a web service?

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.

— W3C, Web Services Glossary

We can identify two major classes of Web services:

- *REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and*
- *arbitrary Web services, in which the service may expose an arbitrary set of operations.*

— W3C, Web Services Architecture (2004)

SOAP

- *Simple Object Access Protocol*
- Standard protocol for *web service* communication
- Combo SOAP + WSDL + UDDI
- XML-based
- In contrast to CORBA:
    - Universal, no language binding (IDL) required
    - XML-based (CORBA protocols binary)
    - Stateless
    - Possibly asynchronous

SOAP

WSDL

- *Web Service Description Language*
- XML-based description of web service interface
- Clients know how to communicate with web service based on WSDL description
  - No generated skeleton or stub needed

UDDI

- Universal Description, Discovery and Integration
- Universal register of WSDL descriptions of SOAP web services
- Simplifies web service discovery

SOAP

SOAP

- XML-based protocol
- Messages consist of:
  - *Envelope* – single per request/response
  - (Optional) *header* – additional information, e.g., timeout, security
  - *Body* – data
  - (Optional) *Fault* – error handling
- Over HTTP POST
- Caveats:
  - Verbosity and slow parsing of XML
  - Client-server interaction model (one is always client, the other is always client)
  - Complex structure

SOAP

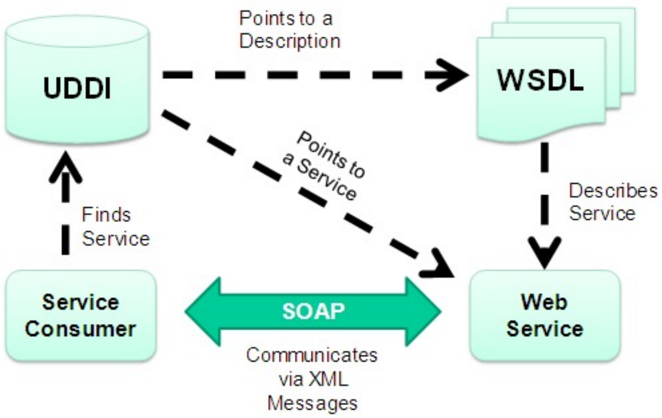


Figure: SOAP+WSDL+UDDI. Source: `http://www.wst.univie.ac.at/workgroups/sem-nessi/index.php?t=semanticweb`

Architectures

General Remarks

Different characteristics of architectures

- Vertical distribution
  - Distribution of logical levels of the system
- Horizontal distribution
  - Distribution of clients and servers
- Temporal distribution
  - Communication is synchronous or asynchronous?

Client-Server vs. Distributed Objects

Client-Server

- Clients and servers are treated differently
- Servers process requests, provide functionality
- Clients make requests, consume functionality
- Example: SOAP, REST, HTTP

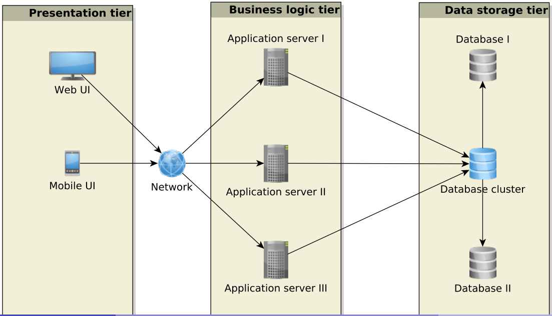
Distributed Objects

- Objects are equivalent, can call each other
- Example: Java RMI, CORBA

Vertical Distribution

N-tier Architecture

- Layers are distributed between processes, can be distributed between machines as well
- Examples
  - *Single-tier* – terminal/mainframe configuration
  - *Two-tier* – client + server
  - **Three-tier** – typical, separate client, server application and database



Services

Service Oriented Architecture (SOA)

- System is split into self-contained separate units – *services*
- Services use each other to provide functionality
- Services can be developed separately, use different technologies, be removed or replaced without affecting the system as a whole
- NOT to confuse with Web Services
- Example: SSO, text analysis service

Microservices

- No precise definition exists, for some it is a more advanced (purer) implementation of SOA
- Software units communicating over lightweight mechanisms (HTTP), deployed using automated machinery and DevOps

Communication in SOA

Enterprise Service Bus (ESB)

- ESB is a *middleware*
- Indirection in service communication – decoupling, routing, synchronous or asynchronous communication
- May support multiple protocols – SOAP, REST
- Simple or Advanced
  - Simple – RabbitMQ, Apache Kafka, Apache ActiveMQ
  - Advanced – Oracle, IBM, Microsoft

Smart Services and Dumb Pipes

- Microservices – decentralized orchestration, often peer to peer
  - Each service may have configuration of other possible services it can use
- Or single service registry

Peer to Peer (P2P)

- Decentralized architecture where nodes function as servers and clients
- Content distribution, sharing, grid computing
- Types
  - *Unstructured* – no central node, peers discover each other (each peer starts with a few possible connections and builds a list of other peers)
  - *Structured* – network has a topology, more efficient peer discovery
  - *Hybrid* – combination of P2P and client/server – usually server helps clients discover other peers, search etc.

P2P

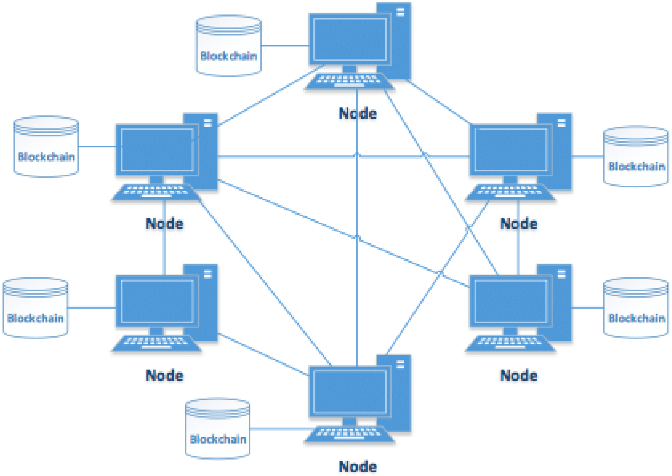


Figure: Source: `https://www.researchgate.net/figure/Blockchain-P2P-Network_fig1_320127088`

Conclusions

Conclusions

- Most of today's applications are distributed
  - At least tiered – backend and frontend separate
- Most applications are integrated using web services
- Services allow to build systems from independent modules

Coming Next Week

- HTTP
- Currently most popular Web service architecture – REST

The End

Thank You

Resources

- https://martinfowler.com/bliki/IntegrationDatabase.html
- M. Fowler: Patterns of Enterprise Application Architecture
- http://xmlrpc.scripting.com/spec.html
- http://www.corba.org/
- K. Richta: Standardy pro webové služby WSDL, UDDI
  - https://www.ksi.mff.cuni.cz/~richta/publications/Richta-MD-2003.pdf
- https://www.slideshare.net/PeterREgli/soap-wsdl-uddi
- http://www.aqualab.cs.northwestern.edu/component/attachments/download/228
- https://ifs.host.cs.st-andrews.ac.uk/Books/SE7/Presentations/PDF/ch12.pdf
- https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html
- https://martinfowler.com/articles/microservices.html