



Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

Data Structure Types

Scalar, Atom, Primitive Type

Plain integer or real number, single character or character string, not indexed and (for string only) accessed in O(1) by character position.

Tuple, Object, Structural Type, Record

Ordered, fixed-size sequence of scalar elements, each of individual type, indexed by name and accessed in O(1) by element name.

Sequence, Array, List

Ordered sequence of elements, each of same type, indexed by position and accessed in O(1) or O(n) by element position.

Set, Bag, Bucket

Unordered set of elements, each of same type, not indexed and accessed in O(1) or O(n) by element reference.

Map, Hash, Associative Array

Unordered sequence of elements, each of same type, indexed by (scalar) key and accessed in O(1) by key.

Graph, Nodes & Edges

Unordered set of linked elements (nodes), each of individual type, indexed by (scalar) key and accessed in O(1) by key or by following a directed link (edge).

Data Evolution Approaches

In-Place Editing

Modify data through direct in-place editing, overwriting the previous revision.

Stacking Revisions

Modify data through stacking revisions, preserving all previous revisions. Latest revision is always on top of stack.

Structural Difference

Modify data through merging, journaled domain-unspecific structural differences.

Operational Transformation (OT)

Modify data through applying journaled, domain-specific operational transformations.

Data Sharing Approaches

Event Sourcing & CRDT

Share data as a chronological sequence of data change events from which the data states can be (re)constructed. Optionally, use a Conflict-Free Replicated Data-Type (CRDT) protocol for the change events.

Ref-Counting & Copy-on-Write

Share data between resources by using reference-counted data chunks, duplicating a chunk (and resetting its reference count to one) on write operations only and destroying a chunk once the reference count drops to zero.

Data Store Types

Key-Value Store

Storage of values in an unordered manner, indexed and queried by key.

Redis, Riak, Memcached, RocksDB, LevelDB

Large-Object Store

Storage of unstructured binary-large object (BLOB) data and its associated meta-data, indexed and queried by unique id.

Minio, SeaweedFS, AWS S3

Triple Store

Storage of subject-predicate-object triples, indexed and queried by subject/predicate/object values and example triples.

Redshift, Virtuoso

File-Tree Store

Storage of unstructured data as named files in a directory tree, indexed and queried by name path from root directory to leave file.

ZFS, XFS, UFS2, APFS

Graph Store

Storage of values as vertices and edges in a graph, both optionally referencing associated key/value pairs. Indexed and queried by key/value pairs and traversed by following edges.

Neo4j, OrientDB, ArangoDB

Document Store

Storage of structured "documents", indexed by id and key/value fields and queried by id and example documents.

MongoDB, CouchDB, RethinkDB

Relational/Table Store

Storage rows of fixed-size, typed value columns, indexed and queried by column values.

PostgreSQL, MariaDB, SQLite, H2, ORACLE DB, IBM DB2

ElasticSearch, Solr, Groonga

Wide-Column Store

Distributed storage of rows of sparse (often untyped) value columns, indexed and queried by column values.

Cassandra, Memcached, HBase, ScyllaDB

Time-Series Store

Storage of integer or real values (y-axis) of a time-series (x-axis) into a fixed-size storage format in a round-robin manner where older values are increasingly aggregated (leading to lower resolutions at older times) and finally overwritten.

InfluxDB, MetricTank, RRDTool

DataVault Store

Long-term historical storage of foreign, arbitrary relational data in a fixed schema of hubs, links and satellites, indexed and queried for analysis and reporting purposes.

DataVault 2.0

BlockChain Store

Storage of values in an unordered manner within information blocks which are cryptographically chained through their hash values and distributed in a peer-to-peer way.

Ethereum, Quorum, Tendermint, Hyperledger

Data Guarantees

CAP (Trade-In)

A distributed data store cannot provide more than two out of three guarantees: Consistency (C), Availability (A), Partition-Tolerance (P). So, it has to choose between Consistency (CP) and Availability (AP) when a network partition or failure happens.



BASE (NoSQL)

The semantics (usually of NoSQL systems) of (B)asically (A)vailable, (S)oft state, and (E)ventual consistency. BASE systems favor Availability over Consistency in the CAP-context.



ACID (RDBMS, NewSQL)

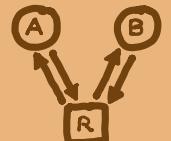
The four guarantees provided in parallel (usually by RDBMS and NewSQL systems): Atomicity, Consistency, Isolation and Durability. ACID systems usually favor Consistency over Availability in the CAP-context.



Data Access

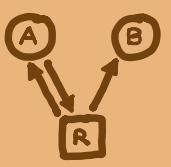
Shared Read/Write

Shared access to data for both read and write operations. Example: Multiple threads on heap or Master-Master database setup.



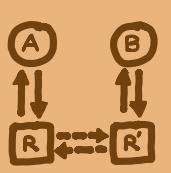
Shared Read / Exclusive Write

Shared access to data for read operations and exclusive access (via a single "owning" component) to data for write operations. Example: RDBMS Master-Slave cluster with shared storage.



Shared Nothing

No shared access to data at all for both read and write operations. Example: Leader-Follower setup with RAFT consensus where Leader writes data only.



Data Access Grouping

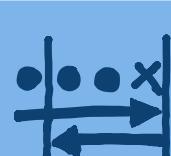
Transaction

Protect a sequence of operations from interim exceptions by bracketing the operations in a technical transaction (ensuring that either all or none of the operations succeed).



Compensation

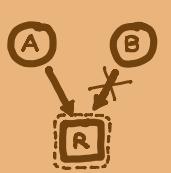
Protect a sequence of operations from interim exceptions by undoing the already succeeded operations through domain-specific compensating (reverse) operations.



Data Consistency

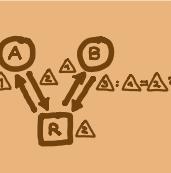
Exclusive Locking (Mutex)

Protect data from concurrent access and resulting inconsistencies with a mutual exclusion lock (mutex) which allows just a single peer to access the data at a time.



Optimistic Locking

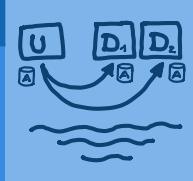
Protect data from concurrent access and resulting inconsistencies by taking note of a revision number or content hash during read operations and checking that this information has not changed before writing the data.



Data Spreading & Aggregation

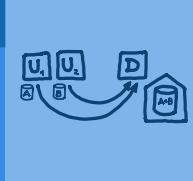
Data River (1-to-N)

A real-time fan-out replication of data from a single upstream/source data repository to multiple downstream/target data repositories.



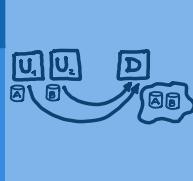
Data Mart (N-to-1), ODS

A massive sized, easily accessible data repository for storing "big data" from many upstream sources in a (real-time) and structured way and with knowing the actual subsequent analysis usage.



Data Lake (N-to-1), Cache

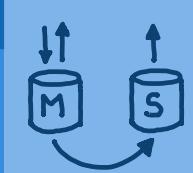
A massive sized, easily accessible data repository for storing "big data" from many upstream sources in a (real-time) semi-structured way and without knowing the actual subsequent usage.



Data Transfer

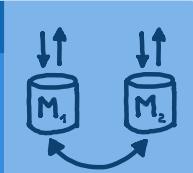
Replication

Continuously stream or regularly copy data from a master system to one or more slave systems in order to either read the data from slave systems faster or have slave systems available as a fallback/backup in case of a failure of the master system.

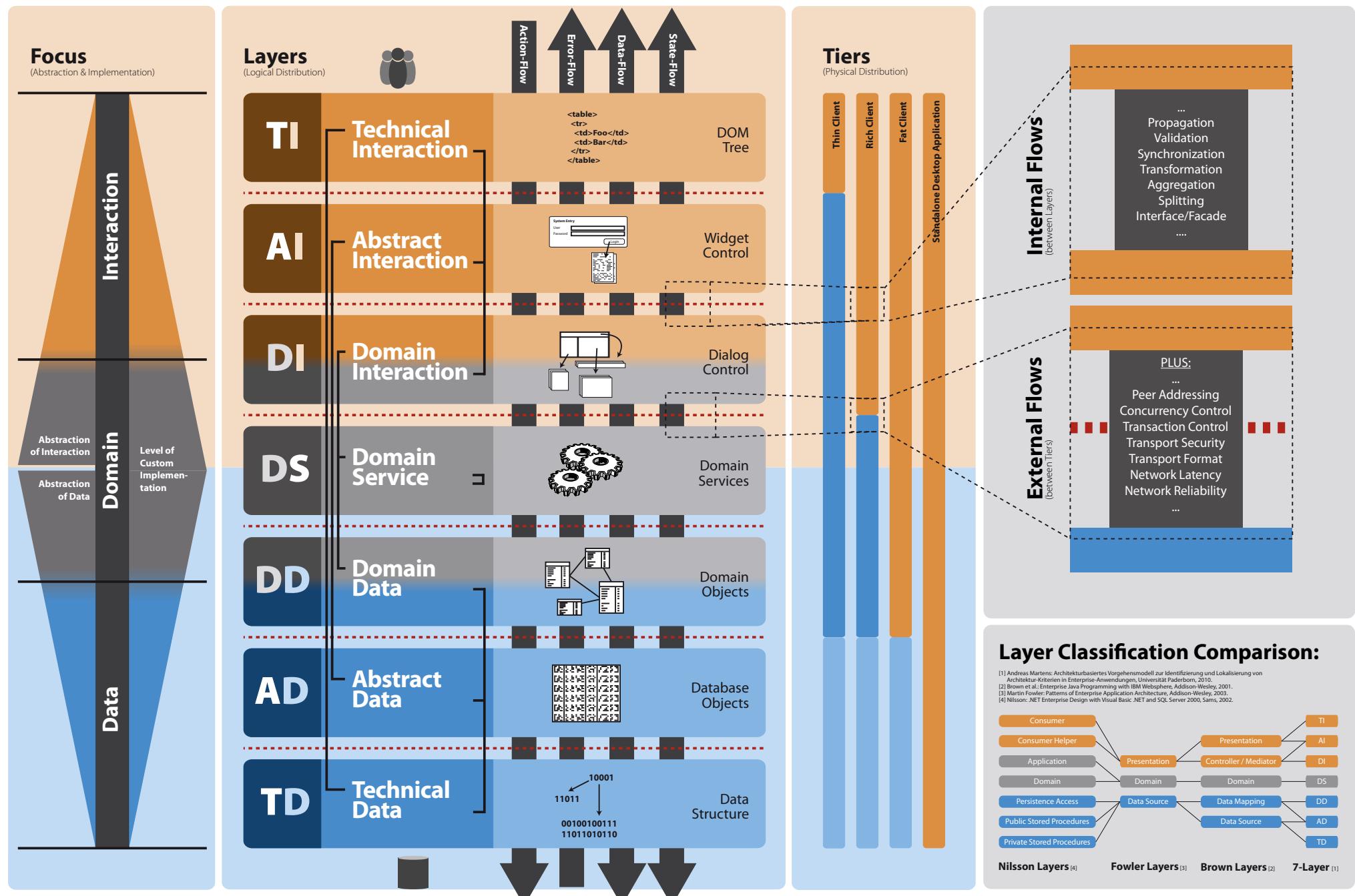


Synchronization

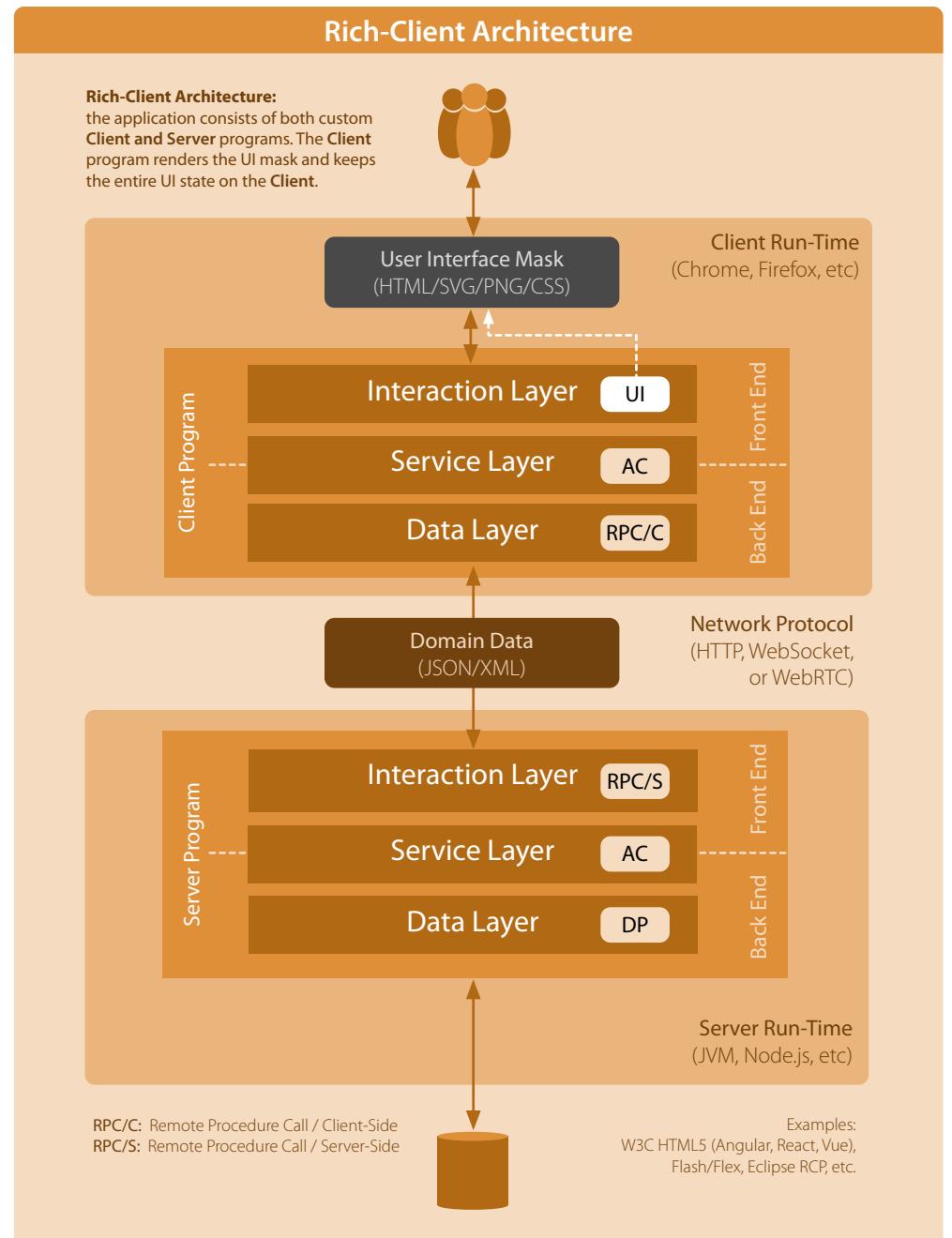
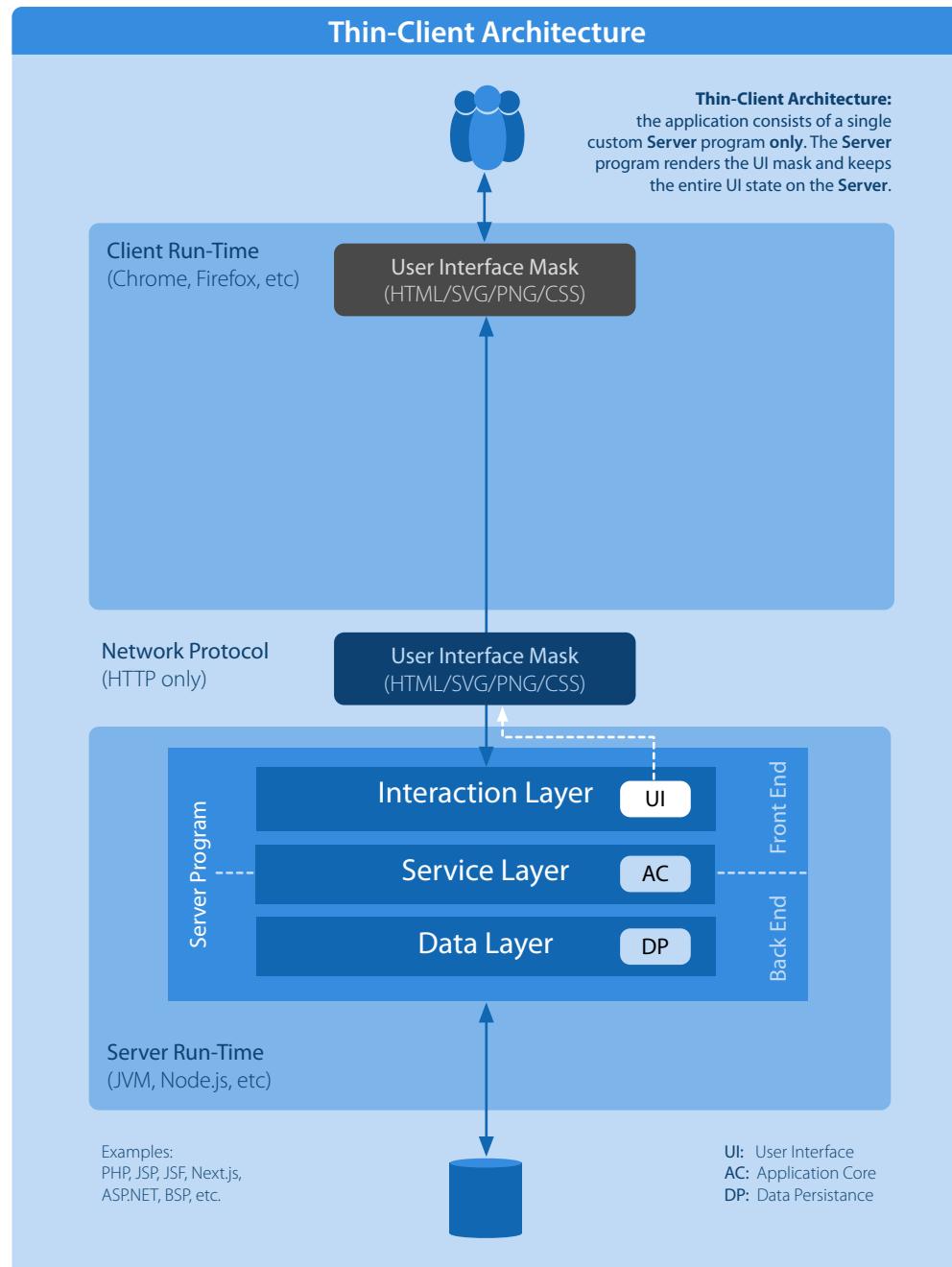
Continuously stream or regularly copy data between multiple master systems and resolve potential concurrent data modification conflicts. This way allow distributed and even disconnected computing.

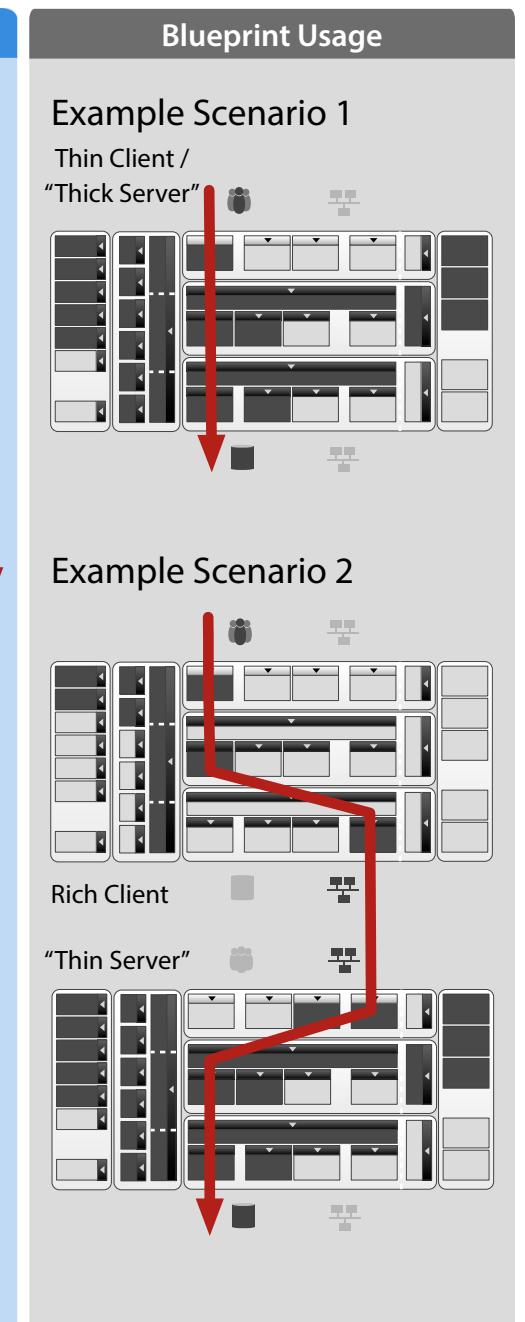
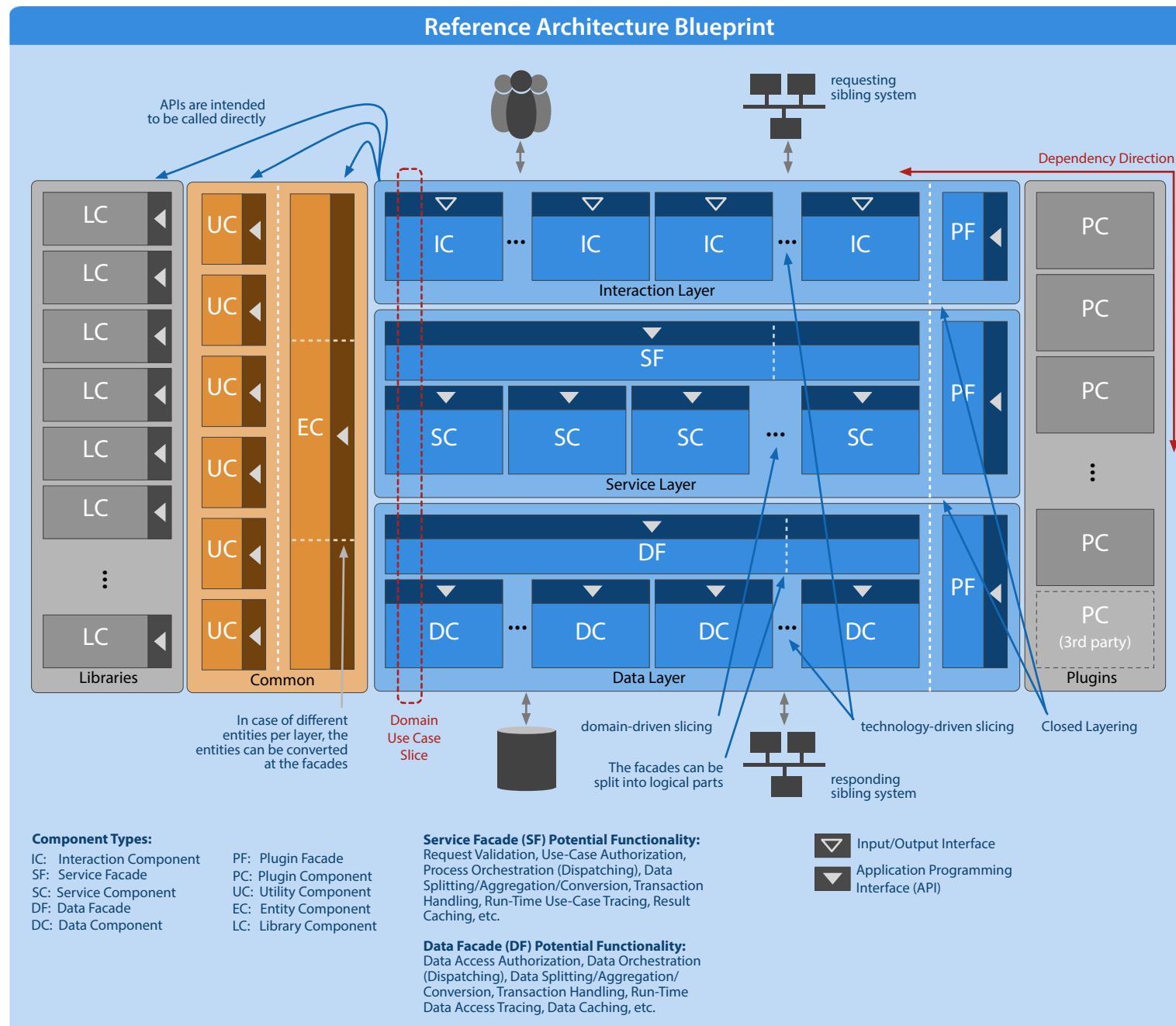


Application Reference Architecture



Client-Server Architecture



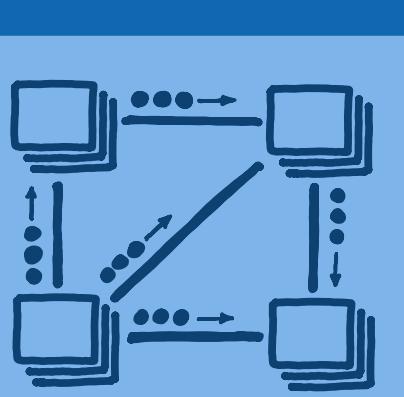


Architecture & Systems

DEF Definition

Reactive System Architecture enables the realization of Reactive Systems.

Reactive Systems are in *subordinated interaction* with their *dominating environment*. They *continuously process endless data streams as small messages*, react at *any time* and respond within *tight time limits*. For this, they continuously *observe their environment* and adapt their behaviour to the current situation.



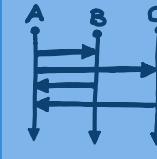
CTX Context

Real-time communication in the context of Digitization, Internet, Internet of Things (IoT), Systems of Engagement, Media and Analytics.



VAL Values

Non-blocking input/output data processing, fast responses within tight time limits, and continuous availability of the provided services.



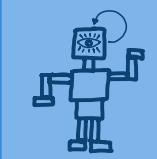
REQ Requirements

Services are elastic and provide high scalability, and are resilient and provide high fault tolerance.



PRP Properties

Services run fully autonomously, monitor themselves, and automatically adapt to changes in the environment.

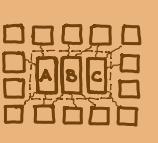


Principles							
Stay Responsive Always respond in a timely manner.	Accept Uncertainty Build reliability despite unreliable foundations.	Embrace Failure Expect things to go wrong and design for resilience.	Assert Autonomy Design components that act independently and interact collaboratively.	Tailor Consistency Individualize consistency per component to balance availability and performance.	Decouple Time Process asynchronously to avoid coordination and waiting.	Decouple Space Create flexibility by embracing the network.	Handle Dynamics Continuously adapt to varying demand and resources.

Patterns & Paradigms

ARC Architecture

Microservices, Cloud-Native Architecture (CNA), Event-Driven Architecture (EDA).



COM Communication

Asynchronous Communication, Non-Blocking I/O, Sequence, Push, Backpressure, Quality of Service (QoS).



DAT Data

Semantical Event, Small Message, Endless Stream.



STY Style

Functional Programming, Asynchronous Programming.



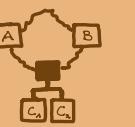
EXE Execution

Parallelization, Concurrency, Actors, Threads, Thread-Pool, Event-Loop.



INF Infrastructure

Message Queue (MQ), Load Balancer, Reverse Proxy, Service Mesh, Virtual Private Network (VPN).



PRC Processing

Complex Event Processing (CEP), EAI Patterns, Stream Processing (map, flatMap, filter, reduce), Event Sourcing.



ASY Asynchronism

Callback, Promise/Future, Observable, Publish & Subscribe.



AF 10.4

Intellectual Content: Version 1.1 (2020-11-09). Authored 2019-2020 by Michael Schäfer and Dr. Ralf S. Engelschall at msg Applied Technology Research. Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.