TECHNISCHE UNIVERSITÄT MÜNCHEN

Software Engineering in der industriellen Praxis (SEIP)

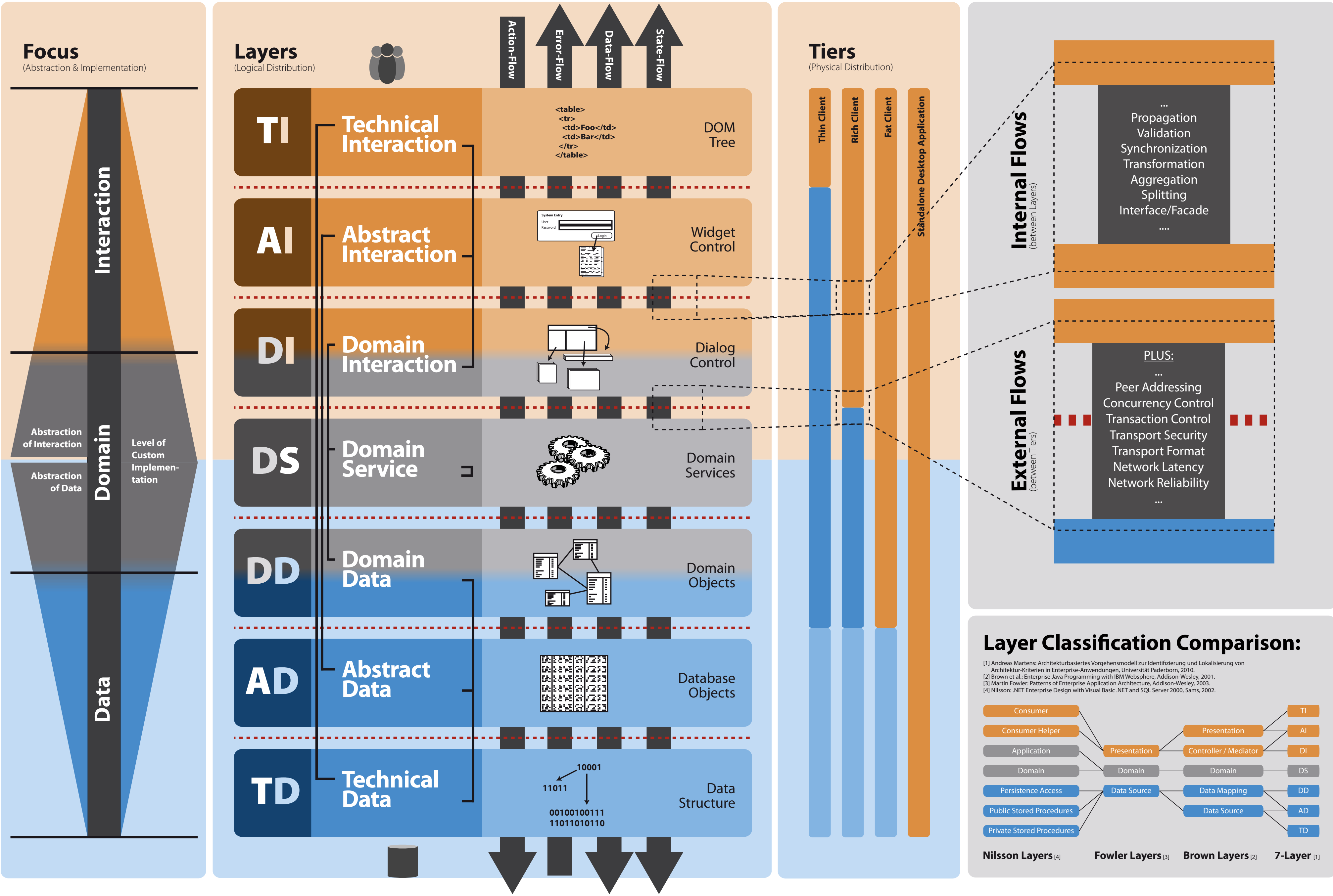
Dr. Ralf S. Engelschall

ARCHITECTURE FUNDAMENTALS

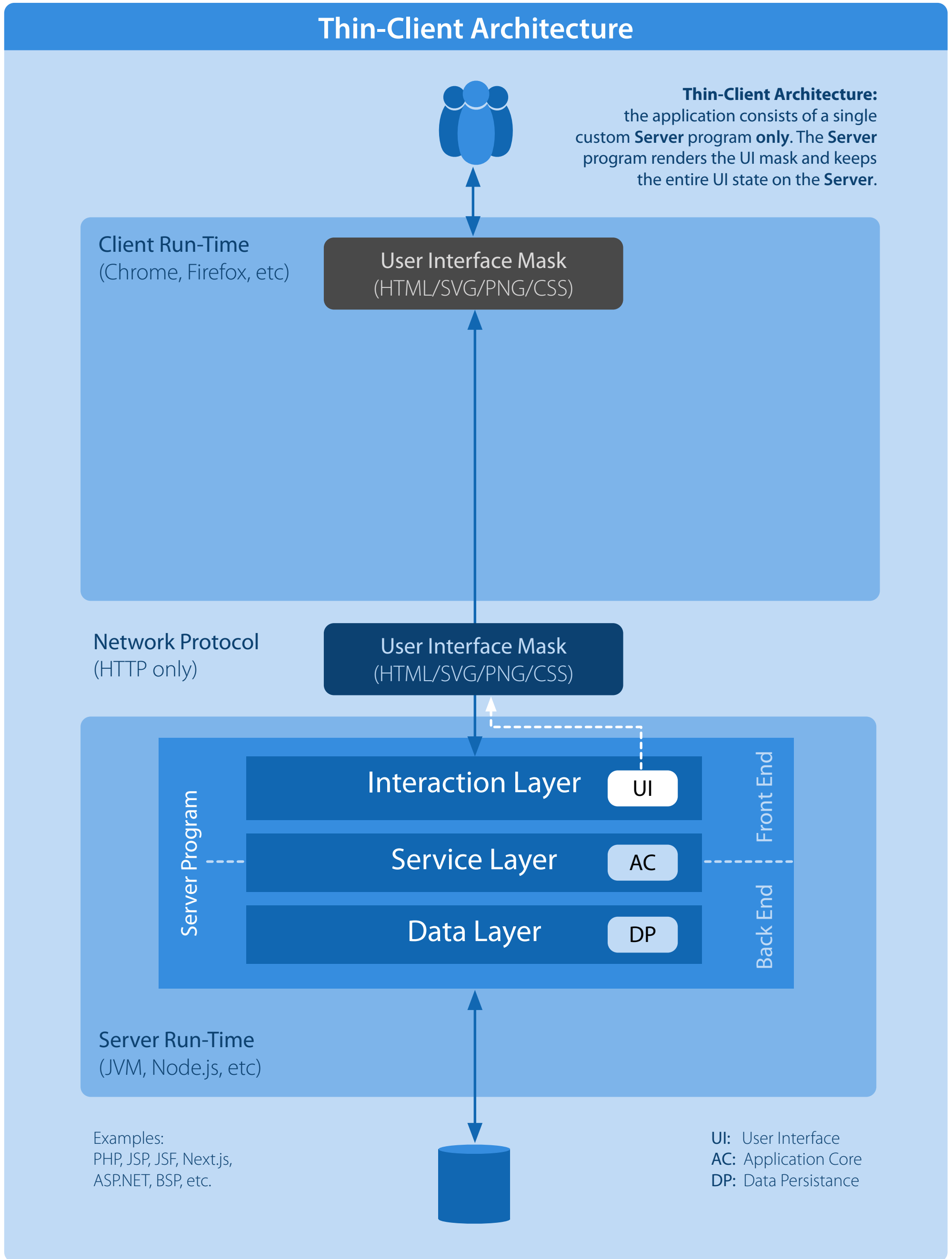
Application Reference Architecture

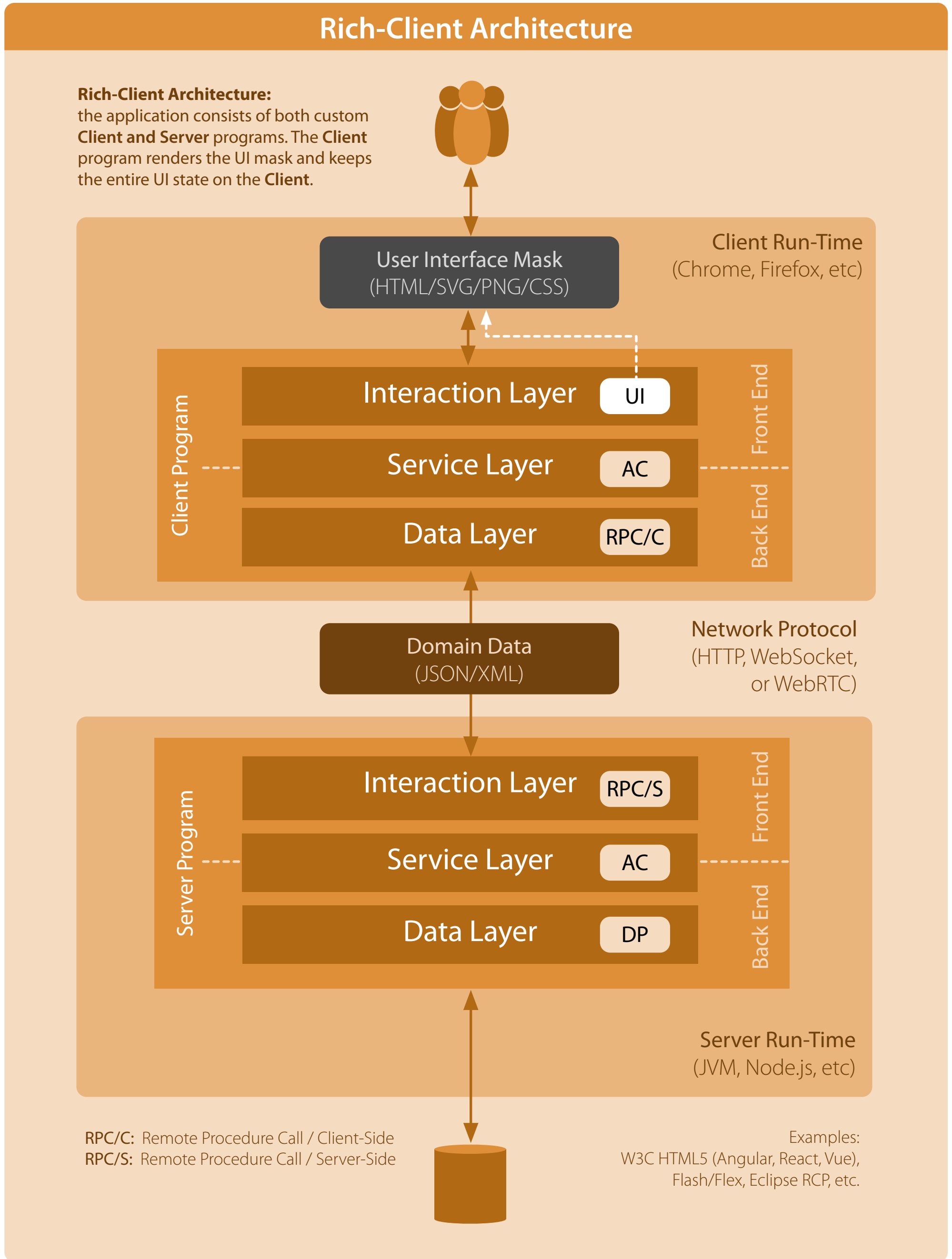
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 10.1



Client-Server Architecture





AF 10.2

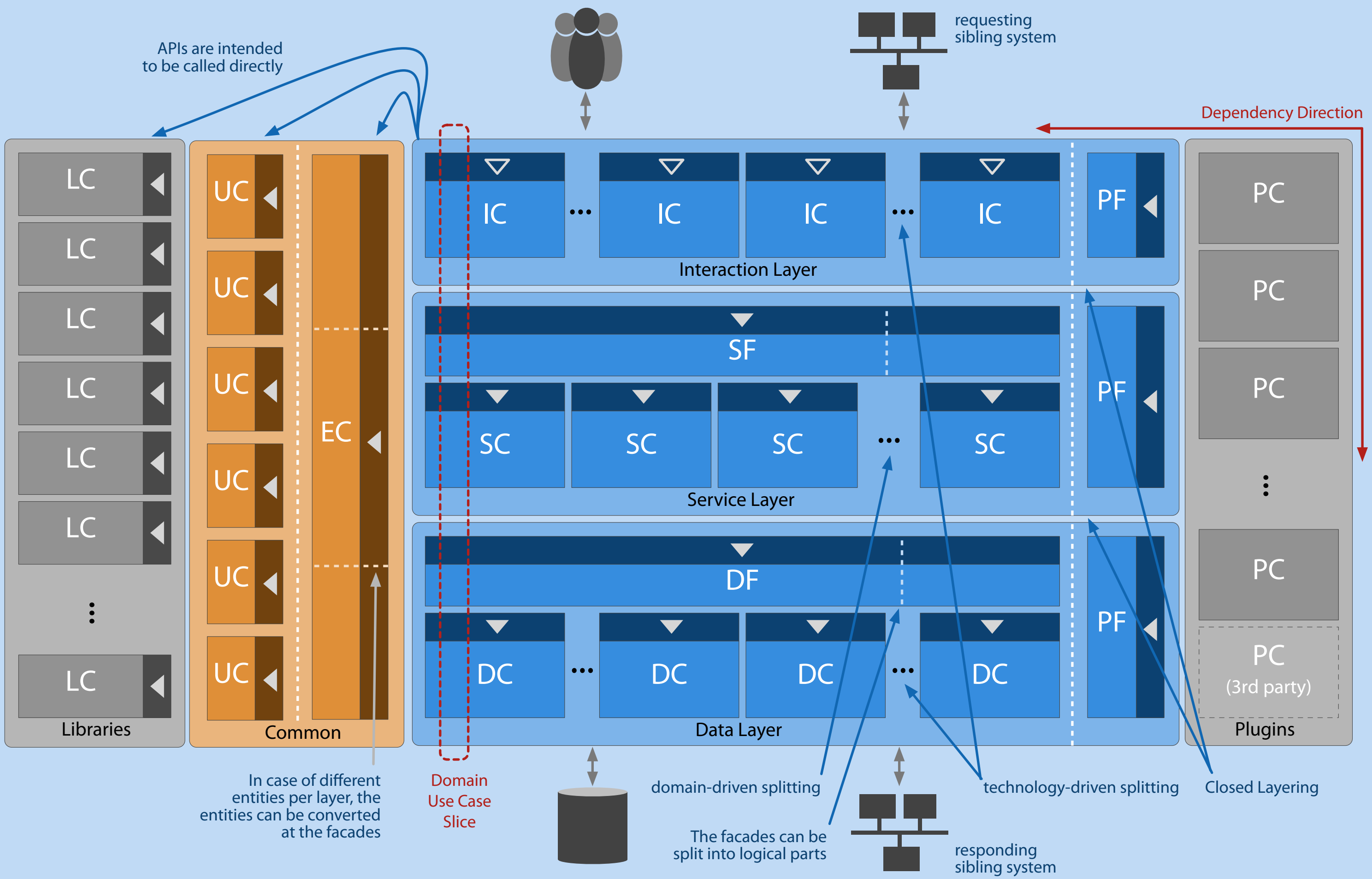
ARCHITECTURE FUNDAMENTALS

Information System Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 10.3

Reference Architecture Blueprint



Component Types:

- IC: Interaction Component
- SF: Service Facade
- SC: Service Component
- DF: Data Facade
- DC: Data Component
- PF: Plugin Facade
- PC: Plugin Component
- UC: Utility Component
- EC: Entity Component
- LC: Library Component

Service Facade (SF) Potential Functionality: Request Validation, Use-Case Authorization, Process Orchestration (Dispatching), Data Splitting/Aggregation/Conversion, Transaction Handling, Run-Time Use-Case Tracing, Result Caching, etc.

Data Facade (DF) Potential Functionality: Data Access Authorization, Data Orchestration (Dispatching), Data Splitting/Aggregation/ Conversion, Transaction Handling, Run-Time Data Access Tracing, Data Caching, etc.

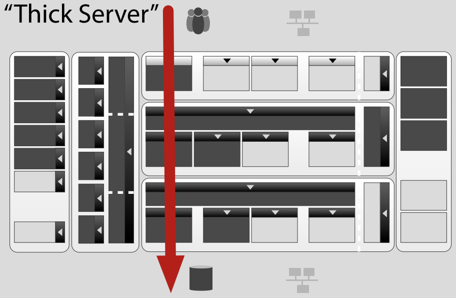
Input/Output Interface

Application Programming Interface (API)

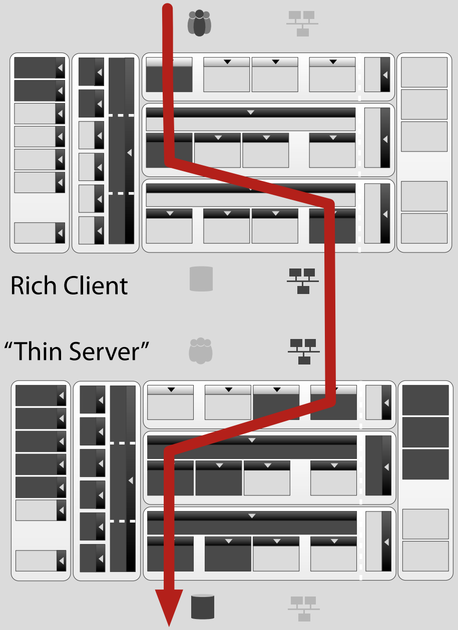
Blueprint Usage

Example Scenario 1

Thin Client / “Thick Server”



Example Scenario 2



ARCHITECTURE FUNDAMENTALS

Reactive System Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

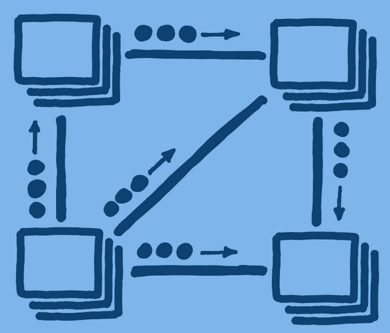
AF 10.4

Architecture & Systems

DEF Definition

Reactive System Architecture enables the realization of *Reactive Systems.*

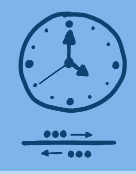
Reactive Systems are in *subordinated interaction* with their *dominating environment*. They *continuously* process *endless data streams* as *small messages*, react at *any time* and respond within *tight time limits*. For this, they continuously *observe* their *environment* and adapt their *behaviour*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



Demand & Deliverables

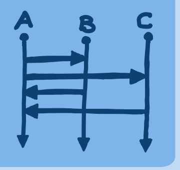
CTX Context

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



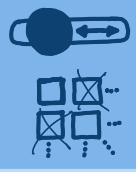
VAL Values

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



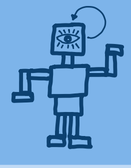
REQ Requirements

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.

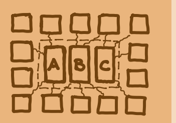
Decouple Space
Create flexibility by embracing the network.

Handle Dynamics
Continuously adapt to varying demand and resources.

Patterns & Paradigms

ARC Architecture

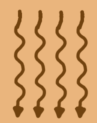
Microservices, Cloud-Native Architecture (CNA), Event-Driven Architecture (EDA).



+

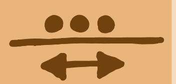
EXE Execution

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



COM Communication

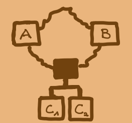
Asynchronous Communication, Non-Blocking I/O, Sequence, Push, Backpressure, Quality of Service (QoS).



+

INF Infrastructure

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



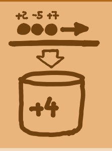
DAT Data

Semantical Event, Small Message, Endless Stream.

+

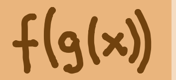
PRC Processing

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



STY Style

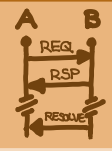
Functional Programming, Asynchronous Programming.



+

ASY Asynchronism

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

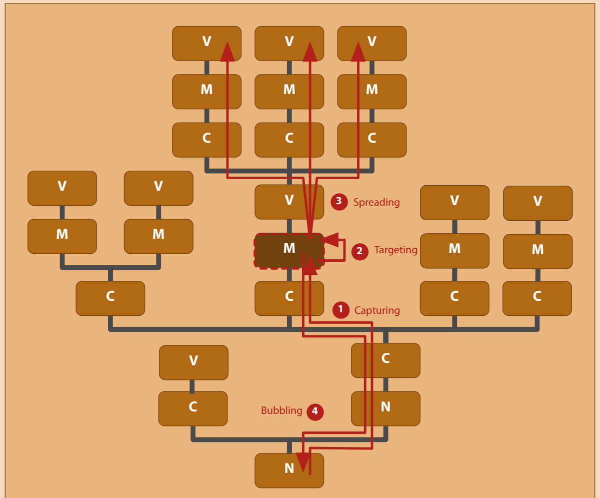


ARCHITECTURE FUNDAMENTALS

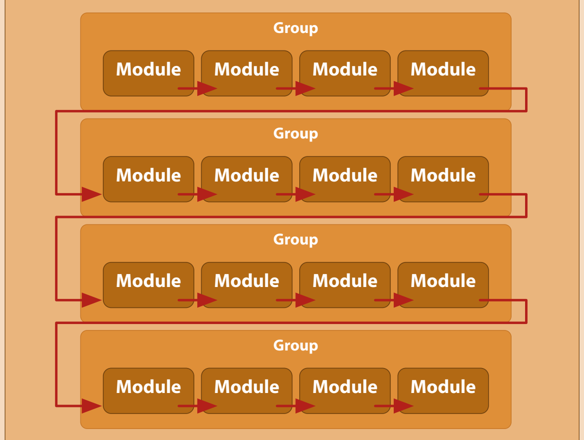
Component-System Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

Component Structure & Dependencies (Dialogs / Services)

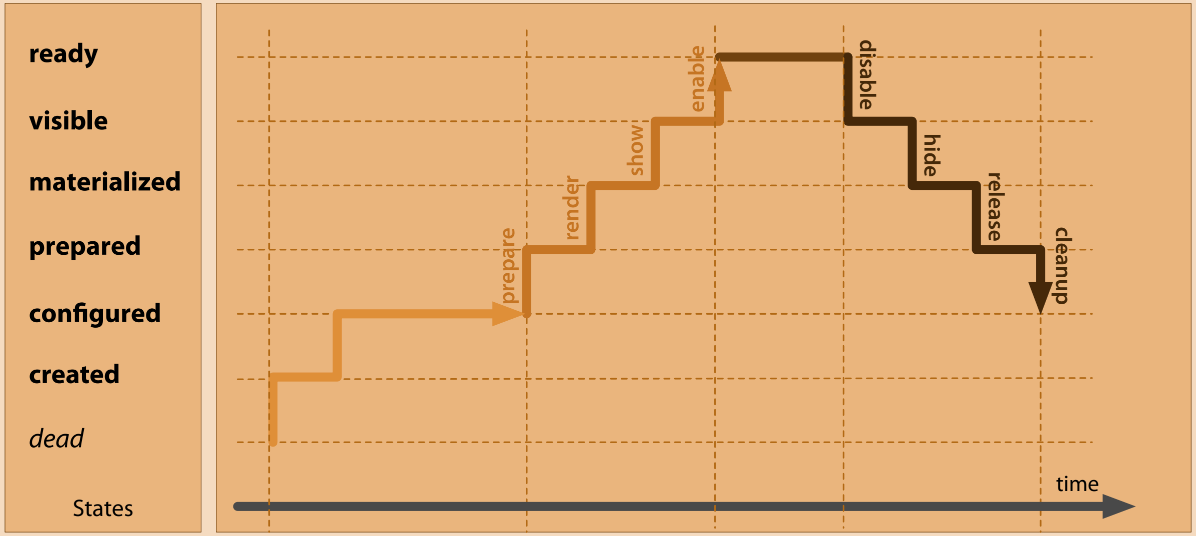


Hierarchical Structure (Dialogs)



Linear Structure (Services)

Component States & Life-Cycle (e.g. Dialogs)



Component Management & Communication

Component Creation & Lookup

Create Components and either establish a (life-cycle-based) dependency hierarchy out of them, or topologically sort them into a (processing-based) dependency linearization. Additionally, allow a flexible path- or identifier-based lookup.

State Definition & Transition

Define one or more distinct Component states and trigger state transitions. The state transitions are performed either directly or hierarchically by ensuring the life-cycle dependencies are always met.

Property Observation & Modification

Provide key/value-based properties which can be read and written and whose value changes can be observed via callbacks. The property value resolution is either hierarchically transitive or direct.

Socket Definition & Plugging

Provide named and unnamed sockets to plug/unplug partial results into/from a result store/queue. The plugging/unplugging on the provider side is realized through custom callbacks. The socket resolution is either hierarchically transitive or direct.

Event Subscription & Publishing

Provide subscription and publishing of event objects. The subscription is realized through custom callbacks and an event is dispatched onto all subscribers either directly or hierarchically in the phases Capturing, Targeting, Spreading and Bubbling.

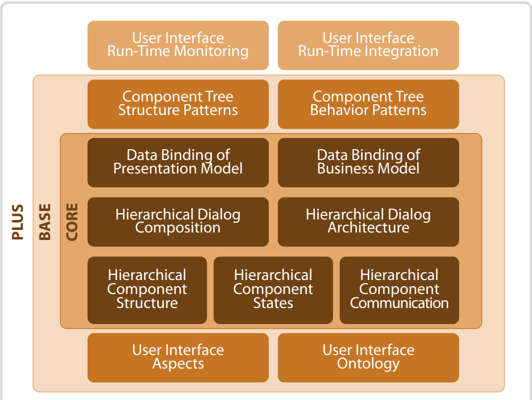
Hook Latching & Execution

Provide named execution hooks and the latching into those hooks via callbacks. The resolution of the hooks is always direct. The hooks receive an input, optionally allow the catchers to process it and finally send the output back.

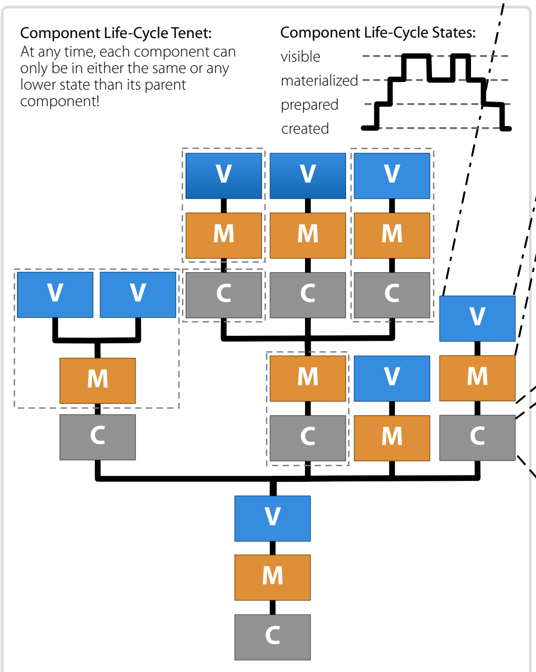
Service Registration & Calling

Provide named execution services and the calling of them. The service execution is provided by callbacks. The resolution of the services is either direct or hierarchical.

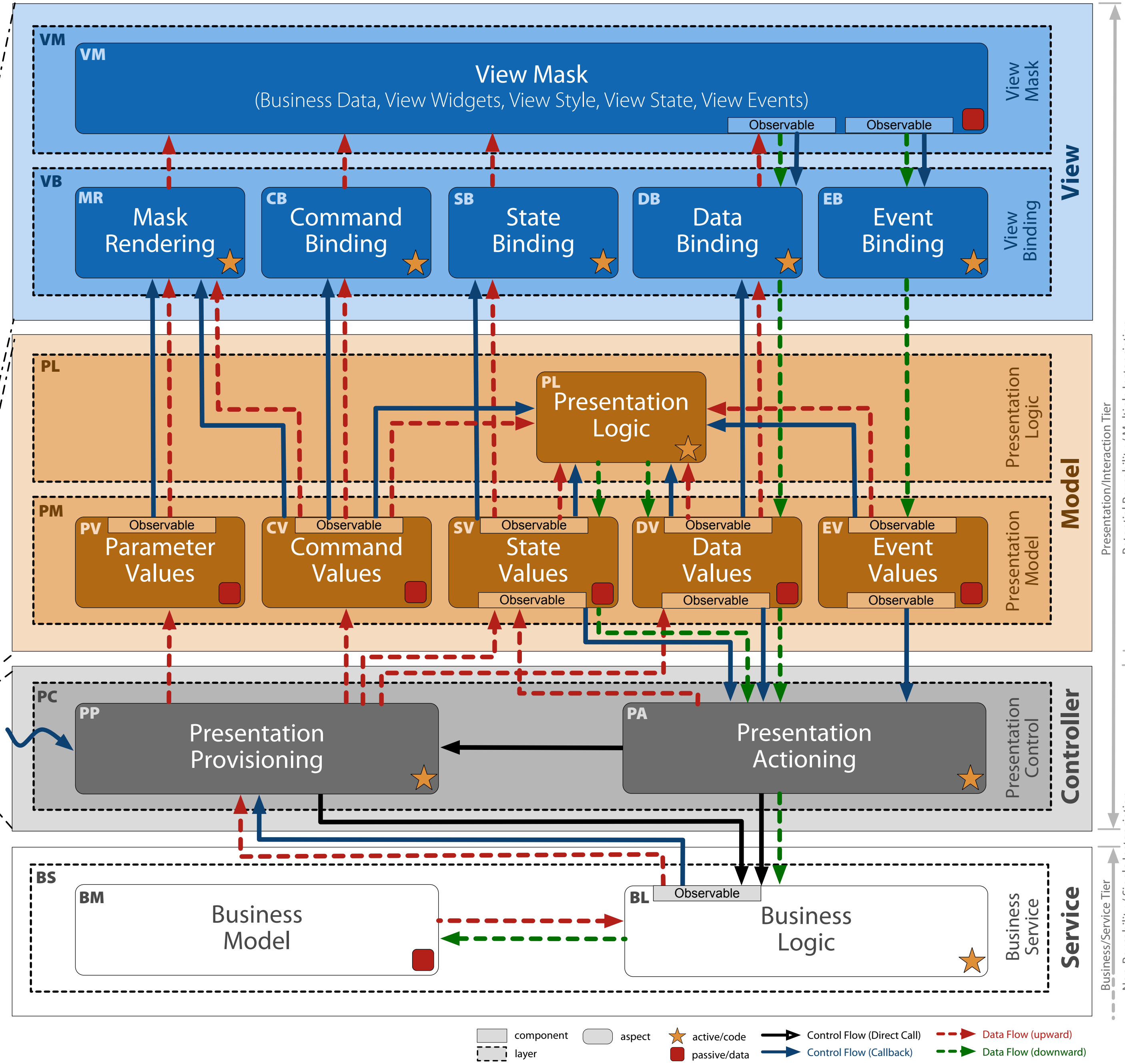
Client Component Architecture



Hierarchical User Interface Component Architecture (HUICA)



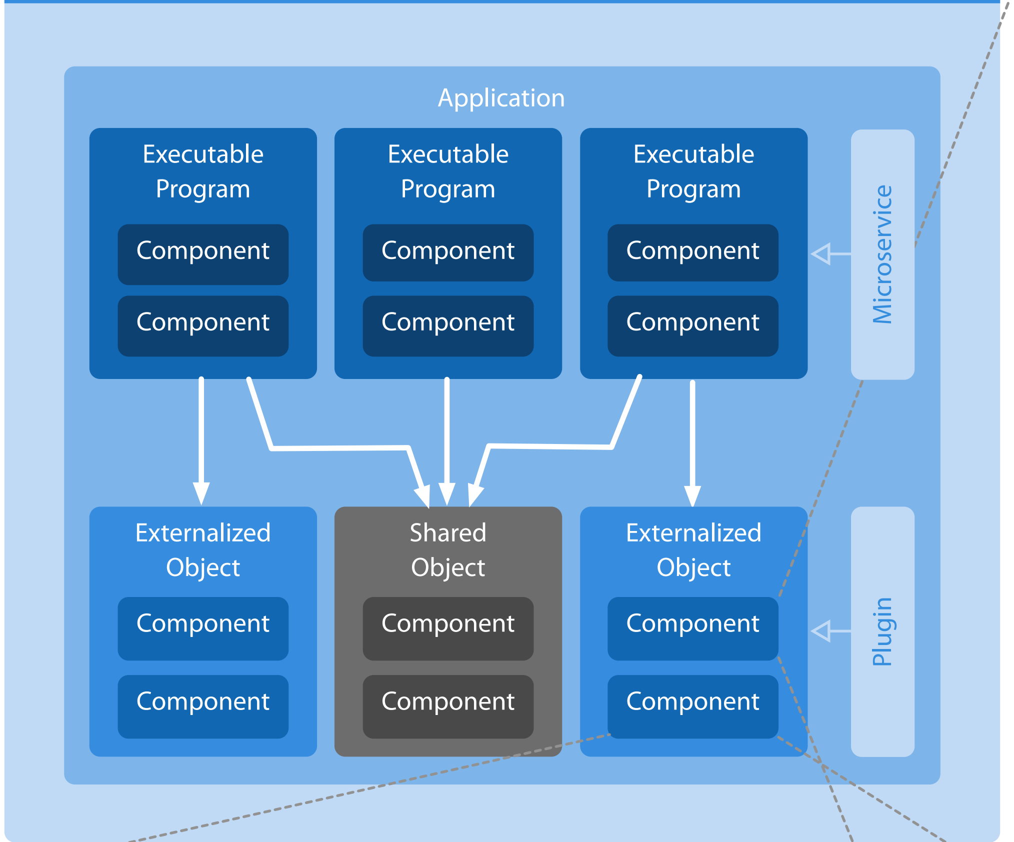
Component Tree (Run-Time)



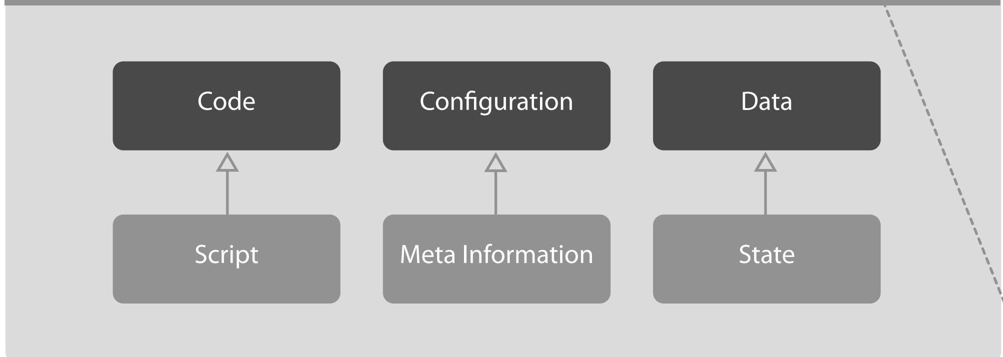
AF 11.2

Application Composition

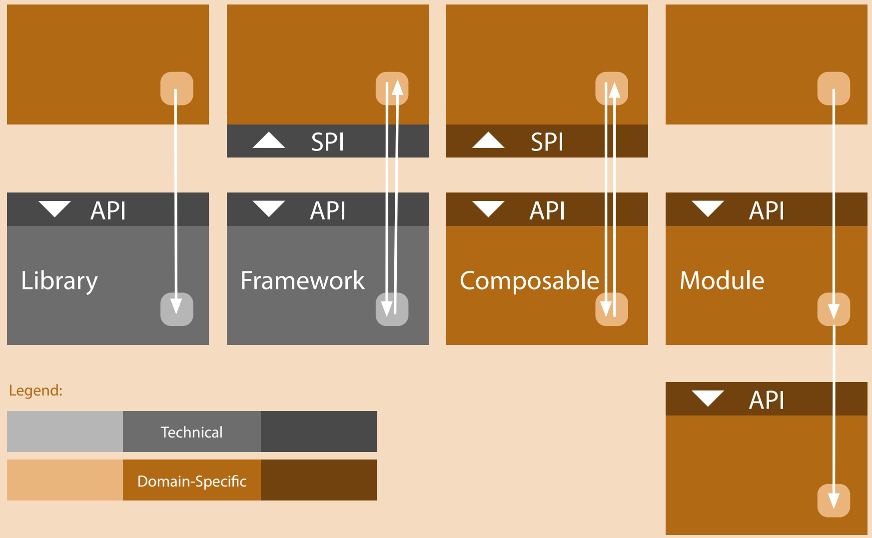
Application Structure



Component Assets



Component Types



Applications are composed out of one or more *Executable Programs*.

An *Executable Program* in turn can leverage zero or more (dedicated) *Externalised Objects* and/or (reusable) *Shared Objects*. Both *Executable Programs, Externalised Objects* and *Shared Objects* are composed through one or more *Components*.

Components are comprised of the assets *Code*, *Configuration* and *Data* and are of four distinct types: *Library*, *Framework*, *Composable* and *Module*, which are distinguished by their particular combination of characterizing aspects.

	data persistence	own life-cycle	requires integration	self-contained	invasive	passive	active	technical	domain-specific	reusable
Library				X		X		X		X
Framework		X	X	X	X	X	X	X		X
Composable	X	X	X	X			X		X	X
Module	X						X		X	