Single-page Webanwendungen

W3L AG
info@W3L.de

2015

Agenda

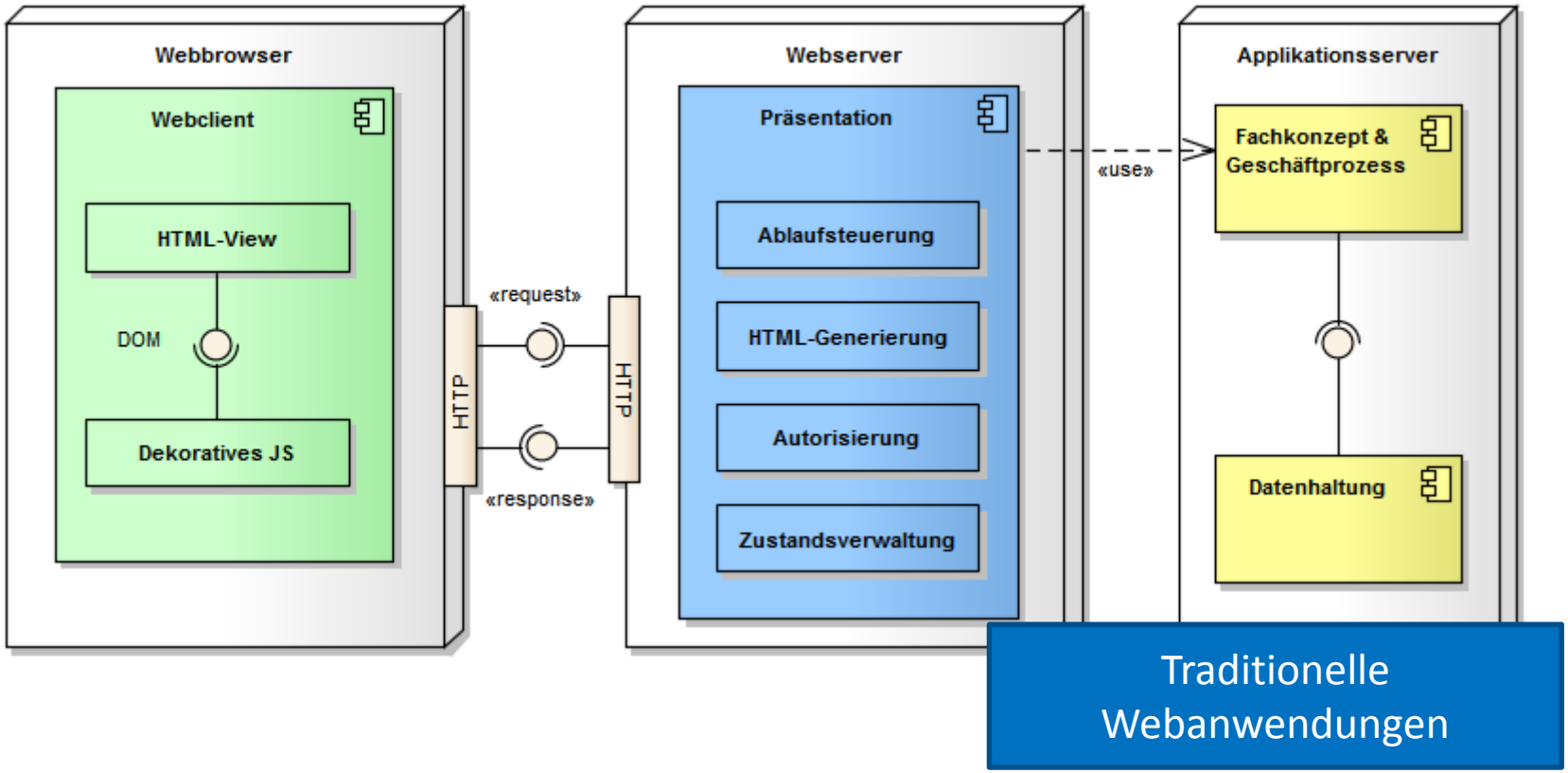
- **Motivation**
  - Einordnung des Themenbereichs
  - Problemstellung & Zielsetzung
- **Das SPA-Paradigma**
  - Vorgehensweise bei SPA-Webanwendungen
  - Relevante Architekturen
    - Interaktionsmuster
    - Kommunikation & Datenanbindung
- **Referenzarchitektur**
- **Schlussbetrachtung**
  - Probleme und Herausforderungen
  - Fazit & Einsatzempfehlungen

Motivation: Wandel der Webarchitekturen

- **HTML5 und Javascript: Webbrowser als Zielplattform**
  - Mobile Web-Apps / Offline-Unterstützung von Webseiten
  - Im Vordergrund: Javascript-basiert „Rich Internet Applications“
- **Webanwendungen werden zunehmend interaktiver**
  - Inhalte werden asynchron aktualisiert, ohne die komplette Webseite neu zu laden
  - Ähnliche Funktionsvielfalt und Bedienung wie bei Desktopanwendungen
    - Direkte Reaktion auf Benutzereingaben / Reduzierte Wartezeiten
- **Resultate**
  - Höhere Interaktivität und Selbstständigkeit der Webclients
  - Webanwendungen werden Client- und Javascript-lastiger
  - Wandel der Webarchitekturen: Thin-Client → Rich- bzw. Fat-Client

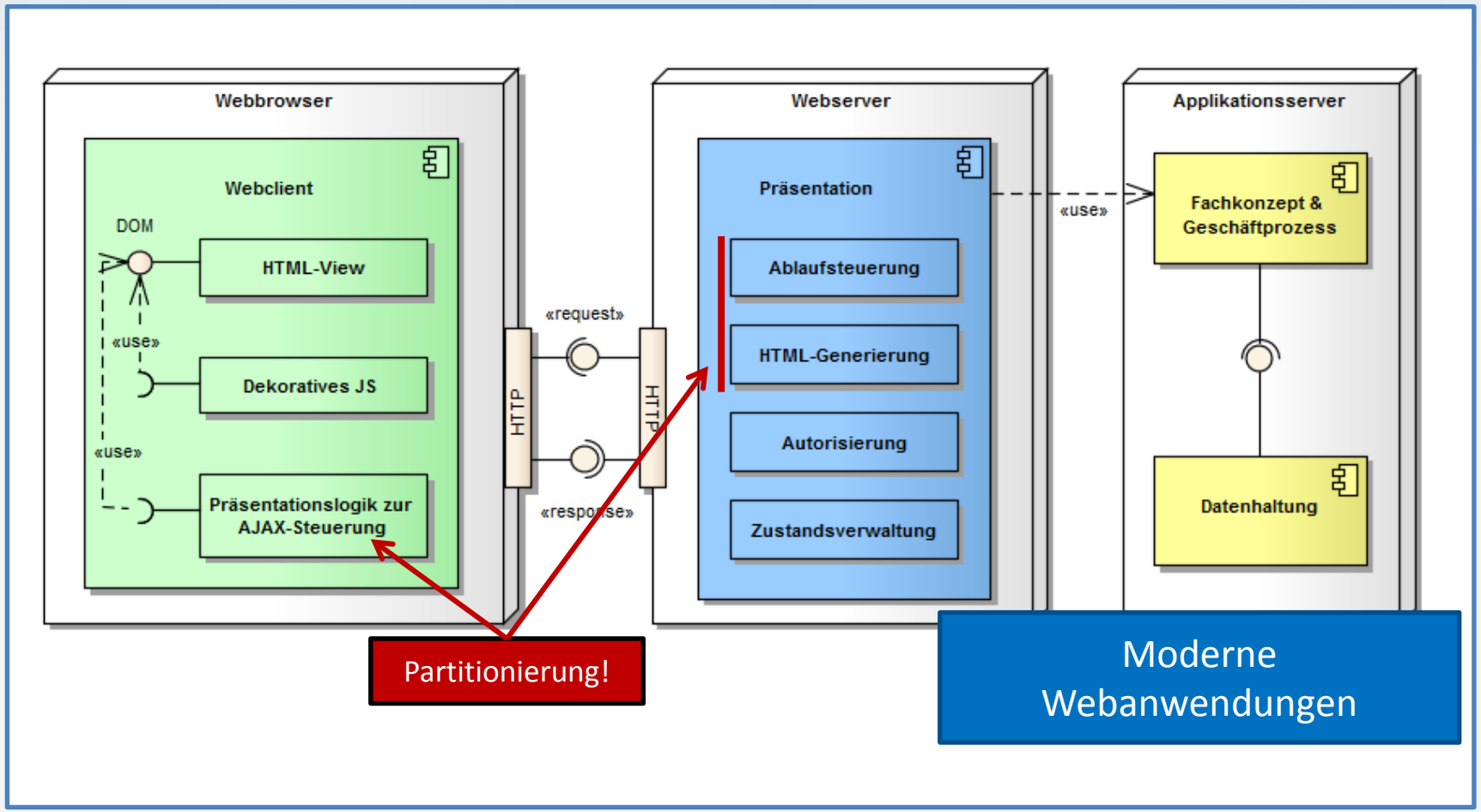
Motivation: Wandel der Webarchitekturen

- Der Ausgangspunkt sind traditionelle Webanwendungen nach der Thin-Client-Architektur



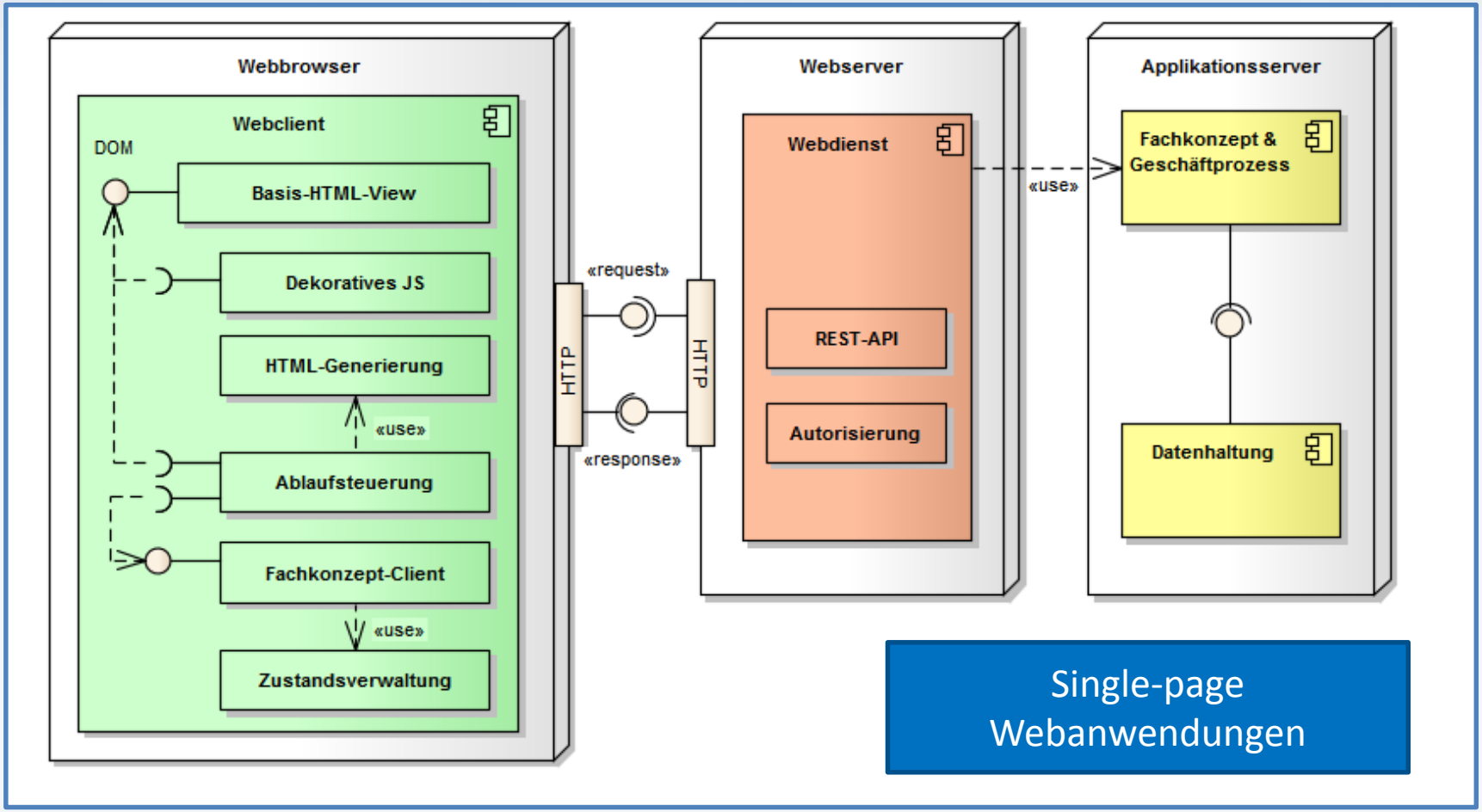
Motivation: Wandel der Webarchitekturen

- Aufwertung des Webs durch AJAX: Wandel in Richtung „Rich-Client“

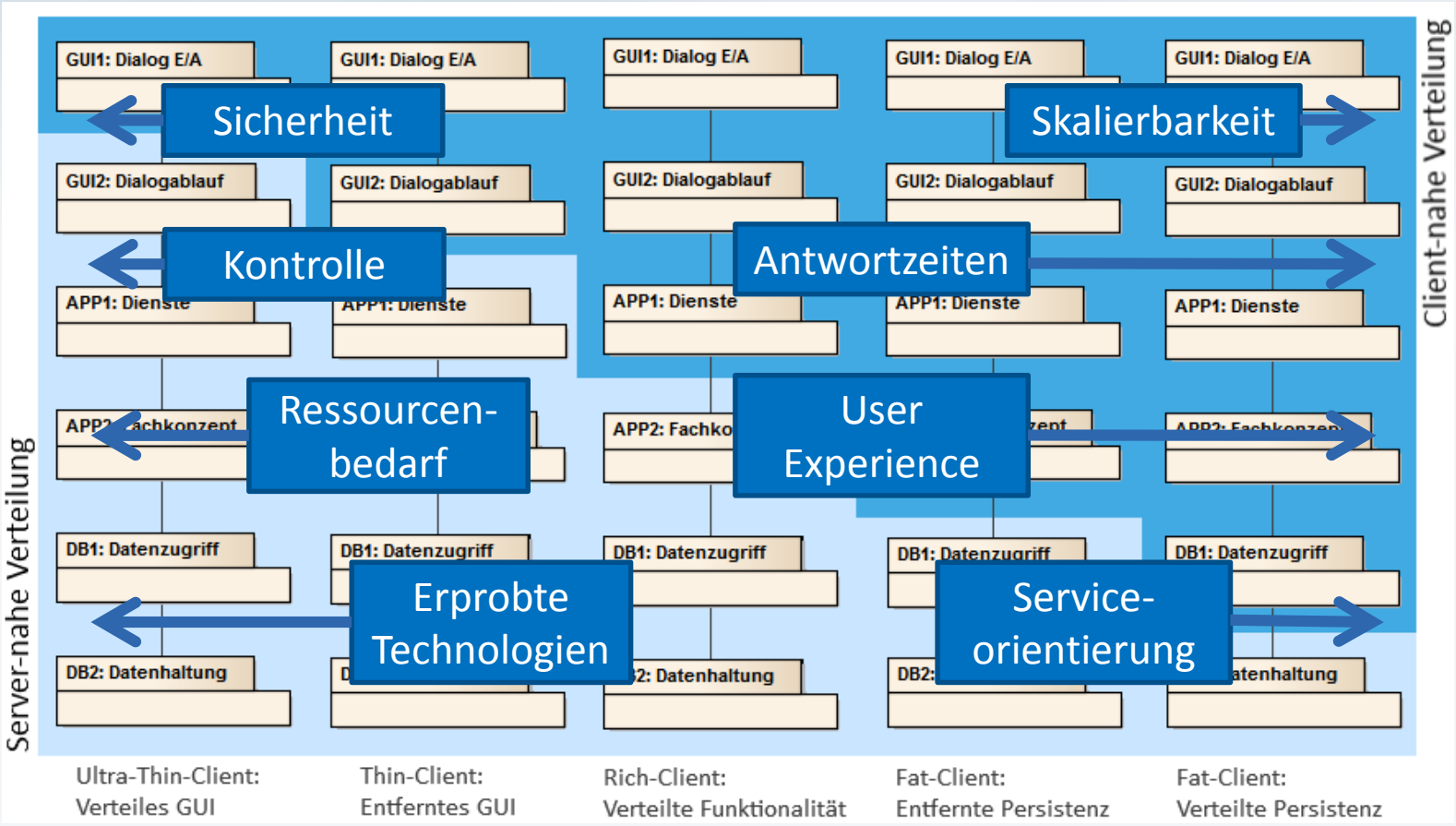


Motivation: Wandel der Webarchitekturen

- Das Paradigma von single-page Webanwendungen ist das Ergebnis dieses Wandels



Wandel zur client-nahen Verteilung



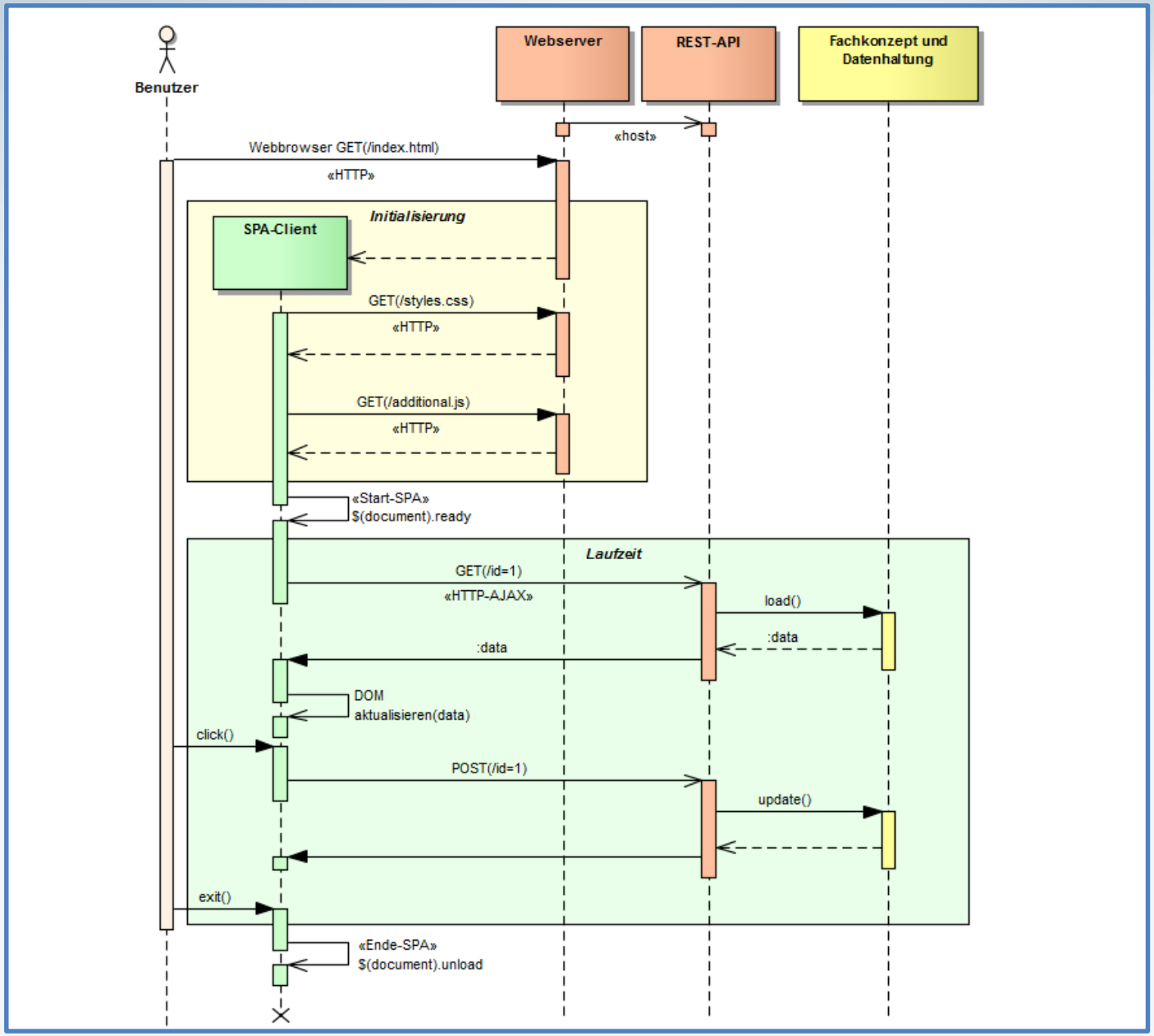
Problemstellung & Zielsetzung

- **Das „single-page“-Paradigma repräsentiert keine Architektur**
  - **P1:** Welche bestehenden Architekturen und Techniken können zur Realisierung verwendet werden? „Es gilt vermehrt mit clientseitiger Logik umzugehen“
    - Anwendungslogik auf dem Client: Beherrschung der Komplexität & Wartbarkeit
    - Zeitweiser Betrieb ohne einen Web-Server (Offlinefähigkeit)
  - **P2:** Welche Rahmenbedingungen der Plattform „Webbrowser“ müssen berücksichtigt werden?
    - Umgang mit Javascript: „Dynamische Programmiersprache“
    - Mögliche Restriktionen und Grenzen des Webbrowsers
- **Ziel → Es gilt das SPA-Paradigma zu konkretisieren**
  - Betrachtung des Wandels der Webarchitekturen
  - Bildung einer Referenzarchitektur als „Best-Practice“-Vorlage
    - Betrachtung relevanter Architekturen und Techniken
  - Betrachtung von Problemstellungen und Nachteilen
  - Definition von Einsatzempfehlungen

Das SPA-Paradigma

- **Ziele des Paradigmas**
  - Umgang mit großen Benutzerzahlen
    - Bekannte Pioniere des Paradigmas: Google, Facebook & Twitter
    - Entlastung der serverseitigen Infrastruktur → Performance & Kostenreduktion
  - Verbesserung der User Experience bei Webanwendungen → Bessere Produktivität
    - Unmittelbares Feedback auf Benutzeraktionen: „Flüssige Übergänge“
    - Unabhängigkeit von Netzverbindungen → Offlinebetrieb
  - Ausnutzung der technischen Gegebenheiten → Der Webbrowser als Plattform
- **[~] Technische Umsetzung des Paradigmas**
  - Keine Seitenwechsel: „Einseitennavigation“
    - → Eine Webseite als Fundament für die komplette Webanwendung
    - → Kein Abbruch der clientseitigen Programmlogiken durch Navigationen
  - Nicht blockierende Kommunikation: „Interaktionsfähigkeit“
    - → AJAX & WebSockets
  - Clientseitige Zustandsverwaltung

SPA-Paradigma: Arbeitsweise



RELEVANTE ARCHITEKTUREN

für single-page Webanwendungen

Relevante Architekturen

- **Der SPA-Webclient übernimmt „klassische“ serverseitige Aufgaben im Rahmen von Webanwendungen**
  - Präsentationsschicht
    - *„Erzeugung von Bedienungsoberflächen“:* Generierung von HTML-Strukturen und Manipulationen von DOM-Elementen (verstärkt)
    - *„Die Steuerung des Präsentationsflusses“*: Die Verarbeitung von Benutzeraktionen und Koordination von Dialogschritten
    - *„Interaktion mit dem Fachkonzept“*: Aufruf von Fachkonzeptdiensten
  - Anwendungsschicht
    - *„Verwaltung des Zustandes“*: Verwaltung von Sitzungsinformationen
    - *„Ausführung von Fachkonzeptlogik“*: Clientseitige Ausführung der Logiken
  - Kommunikationsschicht / Datenhaltungsschicht
    - *„Anbindung von serverseitigen Anwendungsdiensten“*
    - *„Verteilte Datenhaltung“*: Umsetzung von Offlineszenarien (für Fat-Client-Varianten)

Relevante Architekturen: MVC & MVVM

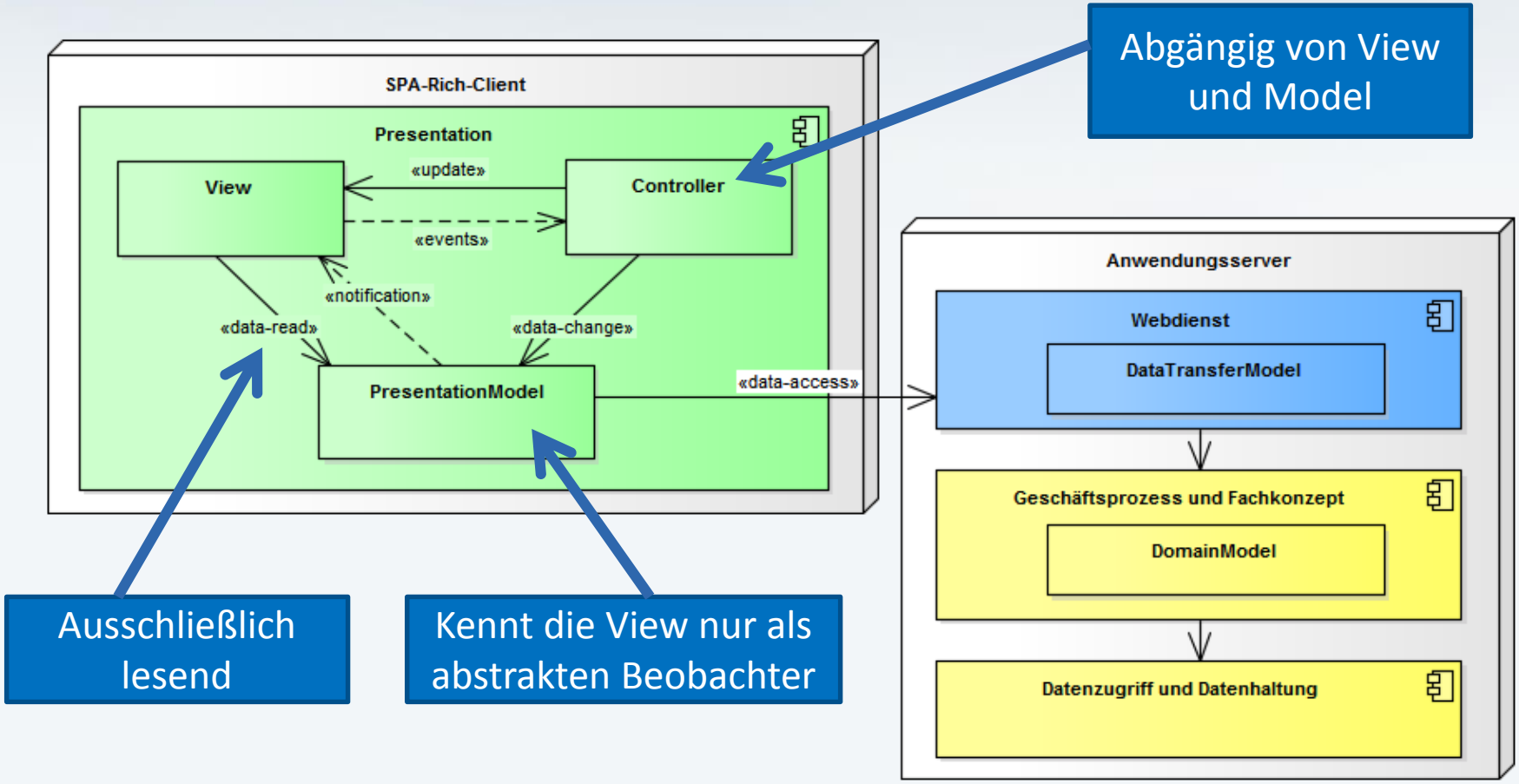
- **Strukturierung der Präsentationsschicht („Bekanntes Problem“)**
  - → Interaktionsmuster befassen sich mit der Strukturierung von interaktionsorientierten Systemen
    - Es wird das Ziel verfolgt, von einer Unordnung zur Struktur zu gelangen
  - Das MVC oder das MVVM-Muster können bei SPA-Anwendungen clientseitig eingesetzt werden
- **Hierdurch werden gleichzeitig wichtige Basisziele erreicht**
  - Die Benutzerschnittstelle wird häufiger angepasst als die Geschäftslogik
    - Es gilt, beiden Komponenten voneinander zu entkoppeln
  - Fachliche Daten müssen in verschiedenen Darstellungsformen präsentiert werden können
  - Änderungen an den Daten sollen mit mehreren Ansichten synchronisiert werden können

Relevante Architekturen: MVC

Das Model-View-Controller-Muster

- „Model“: Das Modell enthält die darzustellenden Daten und repräsentiert stellvertretend das Fachkonzept innerhalb der Präsentationsschicht
  - Es ist ein nicht visuales Objekt
  - Anbindung der Präsentationsschicht an das Fachkonzept
- „View“: Die Präsentation sorgt für die Anzeige der Daten
  - Sie umfasst alle Komponenten zur Darstellung der Bedienungsoberflächen
  - Sie observiert das Model, um Änderungsinformationen zu erhalten
- „Controller“: Bindeglied zwischen Model und View
  - Steuerung des Präsentationsflusses
  - Nimmt die Benutzeraktionen entgegen und führt entsprechende Aktionen aus

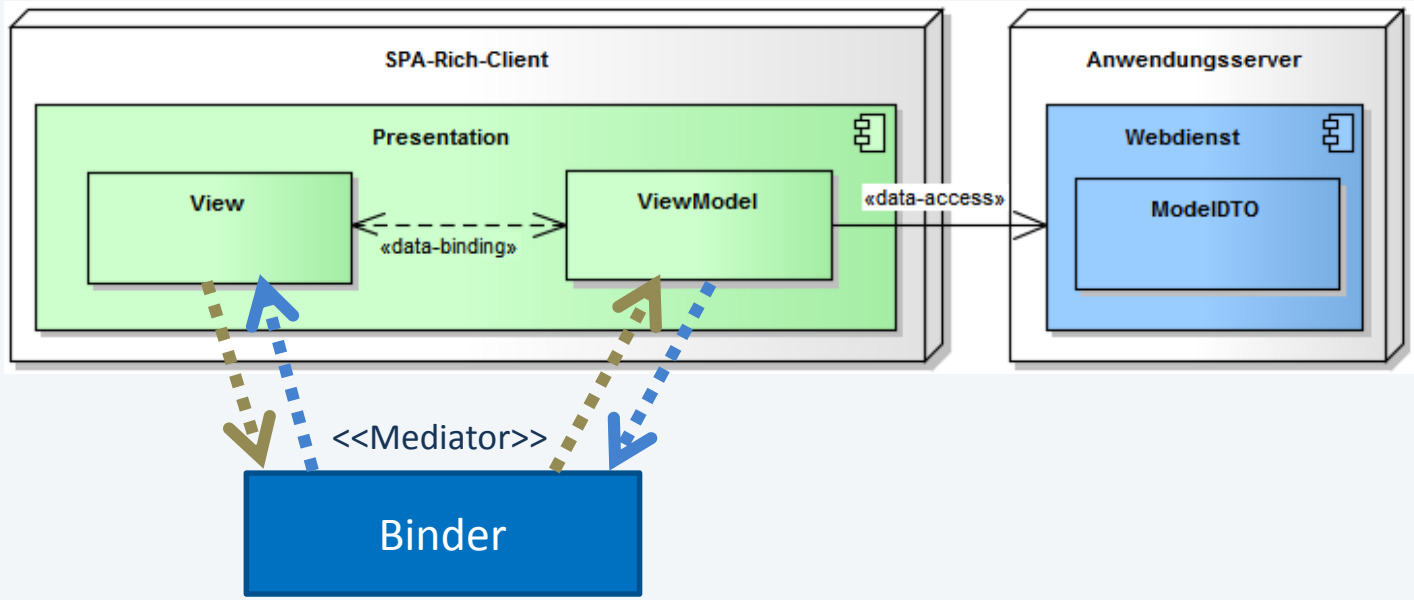
Relevante Architekturen: MVC in einer SPA-Landschaft



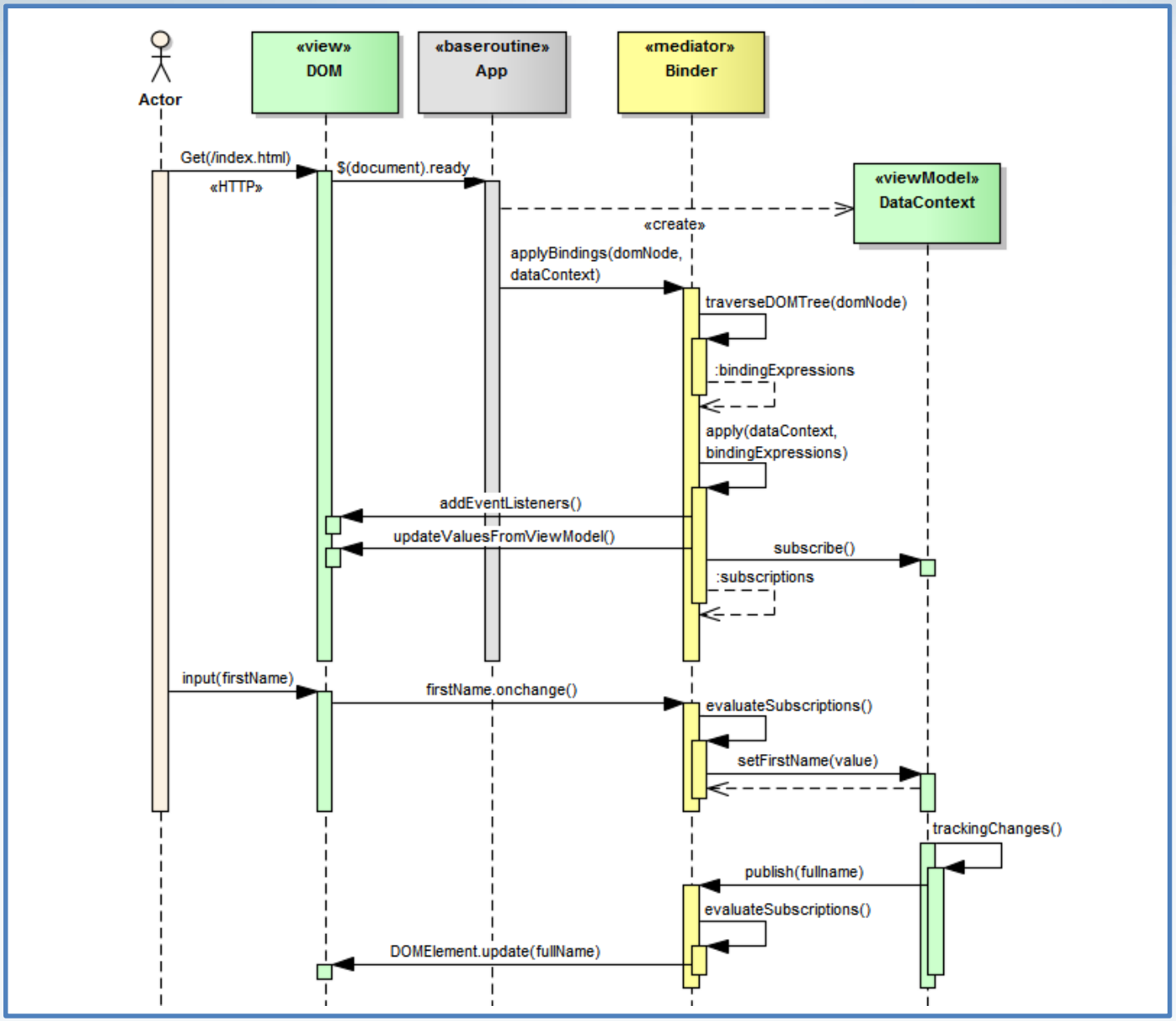
Relevante Architekturen: MVVM

Das Model-View-ViewModel-Muster

- Es stellt eine Variation bzw. Weiterentwicklung des MVC-Musters dar
- Auf *Controller*-Instanzen wird verzichtet. Stattdessen wird Datenbindungsmechanismus zwischen View und ViewModel verwendet
- Für die Bindung zwischen View- und ViewModel-Instanzen ist ein Binder verantwortlich. Dieser Binder fungiert als Mediator



Relevante Architekturen: MVVM



Relevante Architekturen: MVC & MVVM

- **Das MVVM-Muster und das MVC-Muster erlauben beide das Erreichen der aufgeführten Basisziele**
  - Die Darstellung kann ohne direkte Auswirkungen auf das Fachkonzept ausgetauscht werden
  - Klare Trennung von Verantwortlichkeiten → Verbesserung der Wartbarkeit
- **Das MVVM-Muster bietet weitere Vorteile**
  - Reduzierung des Implementierungsaufwands → Keine Controller
    - Ein Binder muss für die jeweilige UI-Technologie vorhanden sein
  - Zusätzliche Unabhängigkeit von der eingesetzten UI-Technologie
    - UI-spezifische Vorgänge werden vom *Binder* durchgeführt
    - Controller-Instanz beim MVC sind abhängig von der UI-Technologie

Relevante Architekturen: Kommunikation

- **Clientserver-Kommunikation bei single-page Webanwendungen**
  - Die Infrastruktur erlaubt die Kommunikation via HTTP und WebSockets
  - Um die Interaktionsfähigkeit des SPA-Webclients aufrecht zu erhalten, sollte stets asynchrone kommuniziert werden
    - Bei der HTTP-Kommunikation wird hierzu die AJAX-Schnittstelle genutzt
  - Das SPA-Paradigma stellt eine Grundlage für die Nutzung von WebSockets dar
    - → Keine Verbindungsabbrüche durch Navigationen
- **Nachrichtenformate**
  - XML-basiert mit dem SOAP/WDSL-Standard
  - REST-basiert mit dem JOSN-Format
  - WebSockets: Es muss ein eigenes Format definiert werden
    - Binärdaten (ArrayBuffer bzw. Blobs) bzw. UTF-8 codierte Strings (per API)

Relevante Architekturen: Kommunikation

- **Bewertung der Kommunikationsvarianten in Rahmen von SPAs**
  - SOAP/WDSL – eher ungeeignet für SPA-Webanwendungen
    - Typsicherheit bei der Datenkommunikation
    - Eigene Nachrichtenheader und Metadaten → Overhead bei der Kommunikation
    - Komplexes Nachrichtenformat → Aufwendiger Parser erforderlich
      - Zusätzliche Belastung für die Clientressourcen (CPU-Leistung, Batterieladung, ..)
  - REST/JSON - Leichtgewichtig
    - REST-basierte Kommunikation = URL + HTTP
    - Kein Overhead durch Nachrichtenheader oder Metadaten
    - Das JOSN-Format kann clientseitig direkt zur Erzeugung von JS-Objekten verwendet werden
  - WebSockets – Verbindungsorientiert für Echtzeitanforderungen
    - Ein eigenes Nachrichtenformat muss festgelegt werden (Aufwand)
    - Ermöglicht eine bidirektionale Kommunikation (Server-Push)
    - Blockiert Server- und Clientressourcen

Relevante Architekturen: Datenhaltung und Caching

Datenhaltung für Offlineszenarien

- Das Thema der clientseitigen Datenvorhaltung nimmt an Bedeutung zu
  - Verstärkter mobilen Einsatz von Webanwendungen
    - Keine durchgängige Kommunikationsverbindung
  - Serverseitiger Wandel bei der Datenhaltung von ACID nach BASE
    - Verzicht auf eine Transaktionskonsistenz → eventuelle Konsistenz
    - Ermöglicht eine performante Replikation der Daten (Skalierung/Verteilung)
    - Problem: Der Client sieht möglicherweise Daten nicht, welche er angelegt hat
- Lösung: Einführung einer clientseitigen Datenhaltung
  - HTML5 erlaubt die Umsetzung einer clientseitigen Datenhaltung
    - Verteilte Datenhaltung → SPA-Fat-Client
  - Zusätzlich verbessern clientseitige Zwischenspeicher die Antwortzeiten

- Offline-Speicher
- Webbrowser-Cache

- Output-Cache
- Application-Cache

Relevante Architekturen: Datenhaltung und Caching

- **Der Aufbau des clientseitigen Zwischenspeichers kann nach verschiedenen Replikationsstrategien erfolgen**
  - *1.) Caching*: Daten, welche aufgerufen wurden, werden im Zwischenspeicher auf dem Client abgelegt
    - → Im Offlinebetrieb sind nur bereits gelesen Daten verfügbar
  - *2.) Replikation*: Alle Informationen des Fachkonzepts werden auf dem Client repliziert
    - → Maximale Datenverfügbarkeit
    - → Ggfs. muss ein großes Datenvolumen übertragen werden
  - *3.) Hoarding:* Es werden gezielt bestimmte Informationen auf dem Client repliziert
    - → 1.) Manuelle Auswahl von bestimmten Bereichen
    - → 2.) Heuristisches Laden von bestimmten Inhalten (Analyse des Nutzerverhaltens)
- **Weitere Problematiken: Bearbeitungssperren & Synchronisation**
  - Konfliktlösungen: Die zuletzt bearbeitete Version eines Datensatzes gewinnt?
  - Bearbeitung im Offlinebetrieb nur, wenn vorher eine explizite Sperre gesetzt wurde?

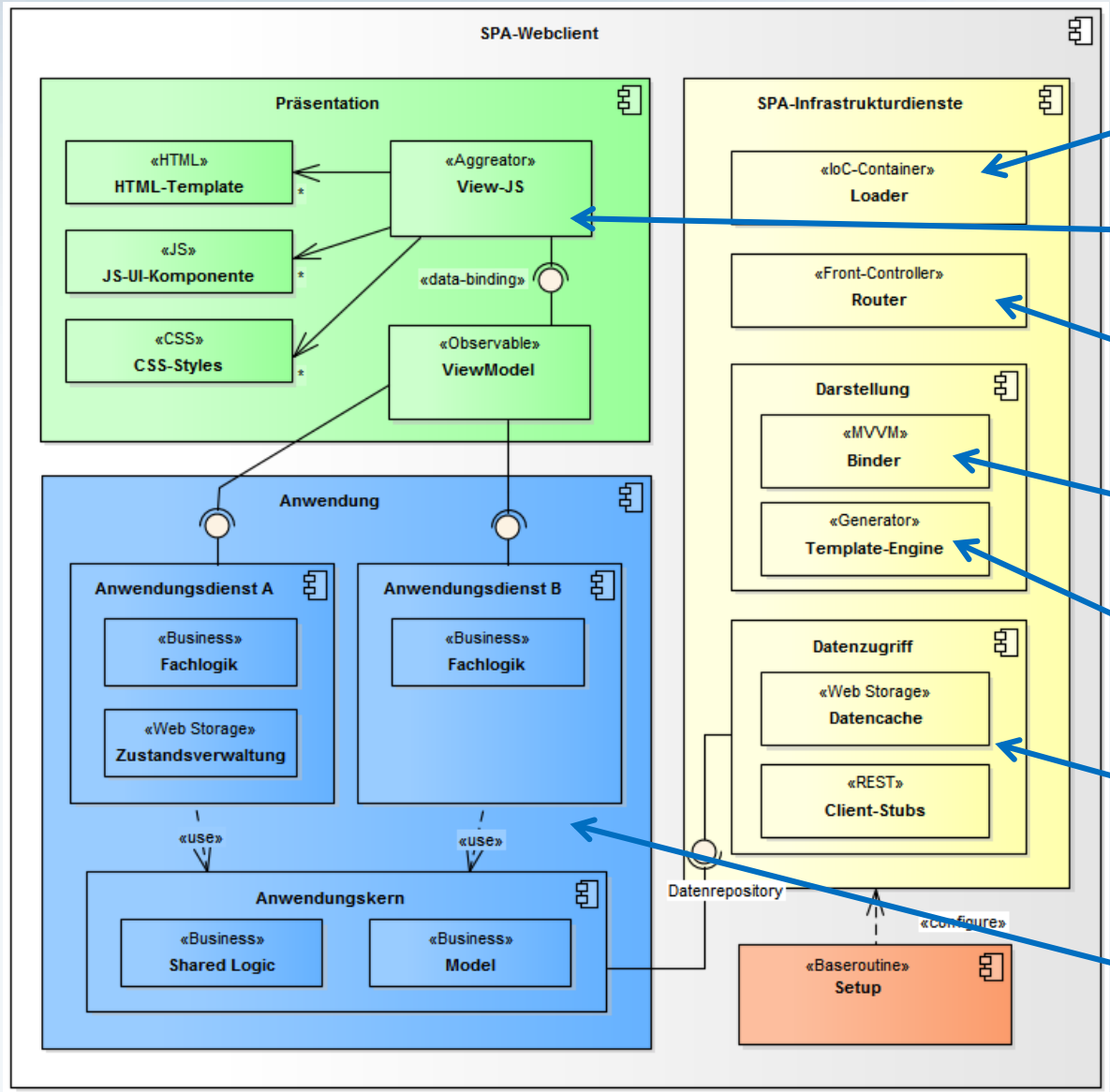
REFERENZARCHITEKTUR

für single-page Webclients

Referenzarchitektur für SPA-Webclients

- **Es existiert keine SPA-Referenzarchitektur**
  - Verschiedene Ansätze für die Umsetzung von clientseitigen Interaktionsmuster mit Javascript sind vorhanden (MV*-Frameworks: AngularJS, Backbone.js [→MP])
  - Gemeinsam mit den vorherigen Betrachtungen lässt sich eine Referenzarchitektur als „Best-Practice“-Vorschlag bilden
- **Folgende Aufgabenstellungen gilt es u.a zu berücksichtigen**
  - Aufbau der Benutzerschnittstelle: Erzeugung von DOM-Strukturen
    - Javascript-Logik für die UI
  - Synchronisation zwischen View und Model: Behandlung von Events
    - Einsatz eines Interaktionsmusters
  - Navigation innerhalb des SPA-Clients
  - Implementierung der Anwendungslogik mit Javascript
    - Wartbarkeit → Bildung von Komponenten; Minderung von Abgängigkeiten (lose Kopplung)
  - Unterstützung von Offlineszenarien
  - Abstraktion von wiederverwendbaren Infrastrukturkomponenten

Referenzarchitektur für SPA-Webclients



DI-Container / „AMD“

Erzeugung der View

Navigation

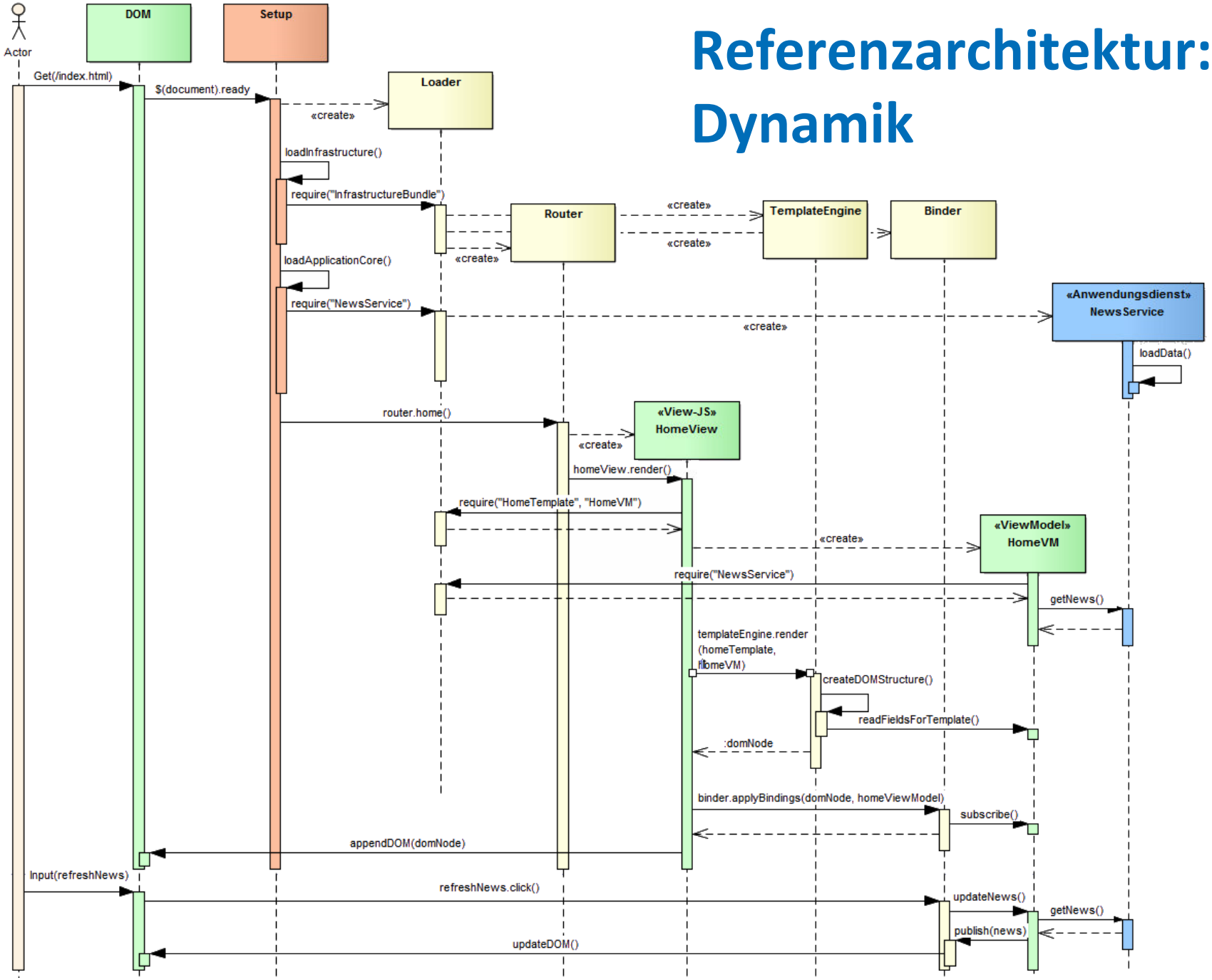
MVVM-Binder

HTML-Generator

REST-Konsumierung
+ Offline-Unterstützung

Dienstorientierte
Strukturierung des
Fachkonzepts

Referenzarchitektur: Dynamik



SCHLUSSBETRACHTUNG

Schlussbetrachtung: Probleme und Herausforderungen

- **Probleme und Herausforderungen: Das Web wird in einer anderen Form verwendet als es konzipiert wurde**
  - Speichermanagement von Javascript → Der Seitenwechsel fehlt
    - Javascript-Implementierung sind bekannt für Speicherprobleme bekannt
    - In aktuellen Version der Javascript-Engines wurden diese Probleme deutlich verbessert
      - Garbage Collection: „Mark-and-Sweep“ statt Referenzzählen
    - Globale Objekte, DOM-Referenzen und Eventlistener können dennoch leicht zu Speicherleaks führen → Abhilfe durch Memory-Profiler von modernen Webbrowsern
  - Zugänglichkeit und Barrierefreiheit → Aufwand
    - → Javascript muss zwingend aktiviert sein
    - Zusätzliche HTML-Attribute zum Abbau von Barrieren: WCAG 2.0 und WAI-ARIA
    - Webbrowser-Navigation und Bookmarks: HTML 5 History API + URL Rewriting
  - Performance: Nicht alle Clientarchitekturen sind für rechenintensiven Aufgaben geeignet
    - Für solche Vorgänge ist weiterhin der Applikationsserver zu verwenden
    - Ebenfalls für Hintergrundprozesse (ETL-Prozesse, Berichtserzeugungen usw.)

Schlussbetrachtung: Probleme und Herausforderungen

Sicherheit bei SPAs

- Klassischen Webanwendung: „geschlossenes System“
  - Der Server hat die volle Kontrolle über den Informationsfluss
  - Clientseitig werden lediglich auswählte Daten bereitgestellt
  - Die Anwendungslogik befindet sich auf dem Server
- Single-page Webanwendungen: „offenes System“
  - Sämtliche clientseitige Logik ist manipulierbar
  - Jede Information, welche zum Client übertragen wird, ist einsehbar

Regeln für SPAs

- „Geschützte Webseiten“ müssen stets HTTPs verwenden (Formular oder Login)
  - Google-Gmail: 1 % CPU-Last, 10 KB Arbeitsspeicher pro Verbindung und 2 % Traffic
- Es muss stets von einer Kompromittierung eines Webclients ausgegangen werden → Die Serverseite muss dem entsprechend abgesichert werden
- Nicht öffentliche Berechnungen oder Entscheidungslogiken dürfen nicht clientseitig ausgeführt werden

Schlussbetrachtung: Fazit

- **Vom SPA-Paradigma zu einer Referenzarchitektur**
  - Mögliche Definition des „SPA-Paradigmas“: „Lösungsansatz um die Abhängigkeit zwischen Client und Server zu reduzieren, und die Skalierbarkeit zu verbessern“
  - Klarere Verteilungsstruktur gegenüber „modernen Webanwendungen“
    - Die Präsentation wird nicht in serverseitige und clientseitige Teile partitioniert
- **Wandel vom Thin-Client zum Fat-Client**
  - Serverseitige Infrastrukturen werden weniger belastet
  - Höhere Bedarf an clientseitigen Ressourcen (diese sind vorhanden)
  - Weiterhin eine Webanwendung → Verteilungsvorteil & Plattformunabgängigkeit
- **SPA-Webanwendungen besitzen folgendende Merkmale**
  - Hoch skalierbar → Durch den Verzicht auf serverseitige Sitzungszustände
  - Bandbreiten schonende → Minimierte Kommunikation (in Anzahl und Volumen)
  - Mobil einsetzbar → Es werden nur Webstandards verwendet
  - Offline-freundlich → Clientseitige Ausführung der Anwendung ermöglicht den Verzicht auf eine bestehende Kommunikationsverbindung

Schlussbetrachtung: Fazit

- **Nachteile von SPA-Anwendungen**
  - Kein „Progressive Enhancement“: Die Webanwendung ist nicht unabhängig von den Fähigkeiten des Clients nutzbar
    - Javascript-Pflicht + HTML5-Pflicht + Ressourcenbedarf
  - Sicherheitsmechanismen könnten fast ausschließlich zum Schutz der Webdienste eingesetzt werden
  - Defizit im generellen Einsatz der Programmiersprache „Javascript“
  - Höhere Entwicklungsaufwand gegenüber klassischen Webanwendungen
- **Ausblick**
  - HTTP/2.0 als nächster Meilenstein in der Kommunikation bei Webanwendungen
  - HTML5 Device API: Einbettung von SPA-Clients in native mobile Anwendungen wird hinfällig
  - Javascript-Standards (ES6/ES7) : Typisierung + Komponentenformat
  - WebAssembly: Browser übergreifende virtuelle Maschine für Webclients

Schlussbetrachtung: Einsatzempfehlungen

- **Bei großen Benutzerzahlen („B2C-Markt“)**
  - Skalierbarkeit + Reduktion der serverseitige Infrastruktur = Dezentralisierung
  - Erfolgreiche Projekte wie Facebook, Google Gmail, Google Maps und Twitter haben hierbei Pionierarbeit geleistet
- **Für Offlineszenarien**
  - Die Nutzung von Geschäftslogiken im Offlinebetrieb ist nur bei clientseitiger Ausführung möglich
- **Kleines Fachkonzept: „Product-Langing-Pages“**
  - Wenig Fachkonzeptlogik; hohe Anforderung an die User Experience
- **Hohe Interaktivität: Webbrowser basierte Computerspiele**
  - Ablösung von Adobe Flash oder Microsoft Silverlight durch SPAs mit HTML5, Canvas, Web-Sockets und WebGL
- **Web-Apps: Mobile hybriden Anwendungen**
  - Die verwendeten Architekturen gleichen dem SPA-Paradigma [Einbettungsfähig]

→ Kosten-Nutzen-Analyse: Mehraufwand gegen den Nutzen abwägen

Literatur (Ausschnitt)

- [Bal11] Balzert, Helmut (2011). *Lehrbuch der Softwaretechnik - Entwurf, Implementierung, Installation und Betrieb.* Heidelberg, DE: Spektrum Akademischer Verlag.
- [Kuu11] Kuuskeri, Janne (2011). Experiences on a design approach for interactive web applications. *2nd USENIX conference on Web application development.* Portland, OR, USA.
- [Ham07] Hammerschall, Ulrike (2007). *Verteilte Systeme und Anwendungen.* München, DE: Pearson Studium.
- [Fow02] Fowler, Martin (2002). *Patterns of Enterprise Application Architecture.* Boston, MA, USA: Addison-Wesley.
- [Sta14] Starke, Gernot (2014). *Effektive Softwarearchitekturen: Ein praktischer Leitfaden.* München, DE: Carl Hanser Verlag.
- [Mos12] Moser, Christian (2012). *User Experience Design: Mit erlebniszentrierter Softwareentwicklung zu Produkten, die begeistern.* Heidelberg, DE: Springer Vieweg.
- [Hal12] Hales, Wesley (2012). *HTML5 and JavaScript Web Apps.* Sebastopol, CA, USA: O'Reilly Media.

Vielen Dank!

Inhouse-Schulungen

Wir bieten Inhouse-Schulungen und Beratung durch unsere IT-Experten und –Berater.

Schulungsthemen

- Softwarearchitektur (OOD)
- Requirements Engineering (OOA)
- Nebenläufige & verteilte Programmierung

Gerne konzipieren wir auch eine individuelle Schulung zu Ihren Fragestellungen.

Sprechen Sie uns an!
Tel. 0231/61 804-0, info@W3L.de

W3L-Akademie

In Zusammenarbeit mit der Fachhochschule Dortmund bieten wir

zwei Online-Studiengänge

- B.Sc. Web- und Medieninformatik
- B.Sc. Wirtschaftsinformatik

und 7 Weiterbildungen im IT-Bereich an.

Besuchen Sie unsere Akademie!
http://Akademie.W3L.de