Skriptum Kommunikationstechnik

Teil 3 Webservices

DI(FH) Andreas Pötscher, HTL Litec

9.2.2026

Inhaltsverzeichnis

Datum	Beschreibung der durchgeführten Änderung	Autor
09.04.2025	V1.0 . . . 1.Version Basierend auf FI1 bis FI3 von Gebhard Klinkan	Andreas Pötscher, HTL Linz–Paul-Hahn-Straße (LiTec)

Client-Server Modell

Die **Client-Server-Architektur** ist das fundamentale Konzept der modernen Netzwerkkommunikation und basiert auf einer klaren Rollenverteilung zwischen zwei Partnern.

- **Der Client (Auftraggeber)**: Ein Endgerät oder eine Software (z. B. ein Webbrowser, eine Smartphone-App oder ein HMI-Panel an einer Maschine), die aktiv eine Anfrage (Request) stellt. Der Client „konsumiert“ Dienste oder Daten.
- **Der Server (Dienstleister)**: Ein leistungsstarker Rechner oder ein Dienst, der passiv im Netzwerk wartet. Sobald eine Anfrage eingeht, verarbeitet er diese und sendet eine Antwort (Response) zurück. Der Server kann dabei lokal im gleichen Netzwerk, oder über entfernt über das Internet erreichbar sein. In der Mechatronik kann z.B. auch eine moderne SPS (Speicherprogrammierbare Steuerung) als Webserver fungieren, um Maschinendaten bereitzustellen.

Die Kommunikation folgt einem strikten Protokoll (meist HTTP). Ein entscheidendes Merkmal ist, dass der Server niemals von sich aus ein Gespräch beginnt; er reagiert ausschließlich. Diese Architektur ermöglicht eine zentrale Datenhaltung und Sicherheit, da der Server die Kontrolle darüber behält, wer welche Informationen erhält.

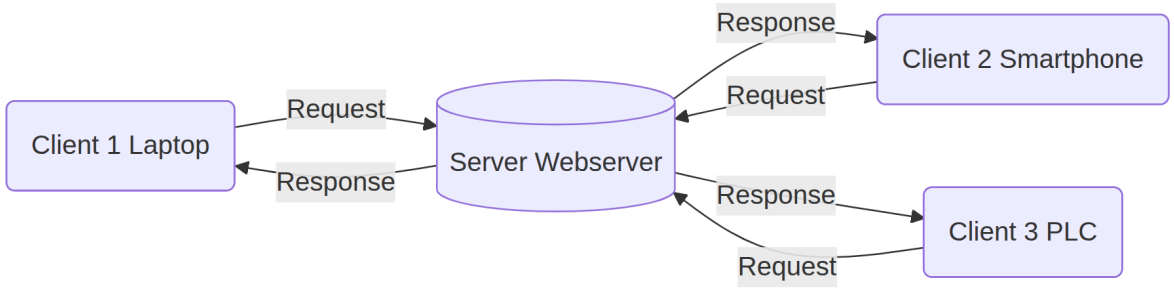

Abbildung 1: Ein Server mit mehreren verbundenen Clients

Erreichbarkeit im Client-Server-Modell

Damit das Client-Server-Modell funktioniert, muss der Server für den Client über das Netzwerk erreichbar sein. Hierbei unterscheidet man zwei Szenarien:

1. **Lokale Erreichbarkeit**: Der Server befindet sich im selben Netzwerk wie der Client (z. B. im selben Firmen- oder Heimnetz). Die Kommunikation erfolgt direkt über die lokale IP-Adresse des Servers.
2. **Erreichbarkeit über das Internet**: Der Server steht weltweit zur Verfügung. Hierfür müssen der Internetzugang und die Netzwerkinfrastruktur (Router/Firewall) entsprechend konfiguriert sein.

Dadurch, dass der Client die Verbindung zum Server aktiv von innen nach außen aufbaut, kann der Server Daten an den Client zurücksenden, ohne dass dieser selbst öffentlich aus dem Internet erreichbar sein muss. Die Firewall des Clients (oder des Routers) erkennt, dass die eingehenden Datenpakete die Antwort auf eine zuvor intern gestellte Anfrage sind, und lässt diese passieren.

> Wichtig: In einem Standard-Heimnetz kann nicht ohne Weiteres ein Webserver betrieben werden, der von außen erreichbar ist. Aufgrund von Sicherheitsbarrieren (Firewalls) und der Network Address Translation (NAT) des Routers werden Anfragen von außen standardmäßig blockiert. Um dies zu ermöglichen, müssten Maßnahmen wie eine Port-Weiterleitung (Port Forwarding) oder ein VPN-Tunnel eingerichtet werden.

Beispiele

Öffnen einer Website

1. Der Benutzer tippt in seinem Browser die Adresse *www.litec.ac.at* in die Adressleiste.
2. Der Browser baut eine Verbindung zum Webserver auf (meistens mit TCP/IP) und sendet einen Request.
3. Der Server sendet einen Response, der die Website enthält, zurück da den Browser.
4. Der Browser zeigt die Website an.

Abfragen eines Webservice

1. Eine Dashboard-Software aktualisiert Wetterdaten einmal in der Stunde.
2. Die Software baut einer Verbindng zum Server auf (meistens mit TCP/IP) und sendet einen Request.
3. Der Server sendet einen Response, der die Wetterdaten als json String enthält, zurück an die Software.
4. Die Wetterdaten werden am Dashboard angezeigt.

HTTP (Hypertext Transfer Protokoll)

Für die Kommunikation vom Geräten in einem Netzwerk wird sehr häufig das HTTP Protokoll verwendet. Eine HTTP-Nachricht ist das formale Paket, das zwischen Client und Server geschnürt wird. Da HTTP ein textbasiertes Protokoll ist, folgt der Aufbau einer strengen, für Menschen lesbaren Struktur, die in Request (Anfrage) und Response (Antwort) unterteilt wird.

Request

Ein HTTP Reqest wird vom Client an den Server in einem lesbaren Format (als Text) gesendet. Beispielsweise:

```
> GET /news HTTP/1.1
> Host: www.litec.ac.at
> User-Agent: curl/8.5.0
> Accept: */*
```

Der Request besteht dabei aus drei Teilen:

- **Request start line:** `GET /news HTTP/1.1`
  - Die Methode die Verwendet wird z.B. `GET`
  - Den Pfad zur gewünschten Unter-Seite z.B. `/news`. Wenn die Hauptseite aufgerufen werden soll dann muss `/` verwendet werden.
  - Die Protokollversion z.B. `HTTP/1.1`

- **Request Headers:**
  - Enthält Parameter in der Form `Name: Wert`
  - Die Adresse die aufgerufen werden soll `Host: www.litec.ac.at`
  - Software den Reqeust absetzt. `User-Agent: curl/8.5.0`
  - Dateien die im Response akzetiert werden. `Accept: */*`
  - Danneben gibt es noch mehr optionale Parameter
- **Request Body:**
  - Optional für bestimmte Methoden (z.B. POST).
  - Enthält die Daten die übertragen werden.

Methoden

Für den HTTP Request gibt es folgende Methoden:

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website. *Message Body*: Nein
- **HEAD:** Holt den Header eines Dokuments vom Server. *Beispiel*: Vorladen einer Website *Message Body*: Nein
- **POST:** Sendet Daten zum Server. *Beispiel*: Daten werden auf einer Website eingegeben und zum Server gesendet. *Message Body*: Ja
- **PUT:** Sendet eine Datei zum Server. *Beispiel*: Hochladen von neuem Content einer Website. *Message Body*: Ja
- **TRACE** Verfolgt eine Message zum Server. *Beispiel*: Testen einer Verbindung. *Message Body:* Nein
- **OPTIONS** Anfrage nach den Options eines Servers. *Beispiel:* Anfragen welche Methoden der Server anbietet. *Message Body:* Nein
- **DELETE** Löscht ein Dokument vom Server. *Beispiel*: Löschen einer Artikels aus einem Online Shop. *Message Body* Nein

> **PUT vs. POST** Der Hauptunterschied liegt in der Idempotenz: Während ein PUT-Request den Zielzustand exakt festschreibt und daher ohne Nebenwirkungen wiederholt werden kann (z. B. „Setze Temperatur auf 20°C“), löst ein POST-Request jedes Mal eine neue Aktion aus. In der Mechatronik nutzt man PUT für das Setzen absoluter Werte, während POST für das Starten von Prozessen oder das Absenden von Befehlen verwendet wird, die nicht doppelt ausgeführt werden dürfen. Kurz gesagt: PUT definiert das Endergebnis, während POST eine Verarbeitung anstößt.

Response

Die Antwort (Response) wird vom Server an den Client zurückgesendet. Sie bestätigt den Erhalt der Anfrage und liefert die gewünschten Daten oder eine Fehlermeldung. Beispielsweise:

```
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=UTF-8
< Content-Length: 1542
< Server: Apache/2.4.58
<
< <!DOCTYPE html><html>... (Daten)
```

Der Response besteht ebenfalls aus drei Teilen:

- **Status line:** `HTTP/1.1 200 OK`
  - Die verwendete Protokollversion, z. B. `HTTP/1.1`.
  - Der Status-Code, eine dreistellige Zahl, die das Ergebnis angibt, z. B. `200`.
  - Die Status-Message, eine kurze Textbeschreibung zum Code, z. B. `OK`.
- **Response Headers:**
  - Enthält Metadaten zur Antwort in der Form `Name: Wert`.
  - Der Datentyp der Antwort: Content-Type: `text/html` (zeigt an, dass eine Webseite folgt).
  - Die Größe der Daten im Body: Content-Length: `1542` (in Bytes).
  - Informationen über die Server-Software: `Server: Apache/2.4.58`.
  - Weitere optionale Parameter (z. B. Datum, Cache-Einstellungen).
- **Response Body**
  - Enthält die eigentliche Nutzlast (Payload).
  - Das kann der HTML-Code einer Webseite, ein Bild oder ein Datensatz im JSON-Format (z. B. Sensorwerte der SPS) sein.
  - Der Body ist durch eine Leerzeile von den Headern getrennt.

Response Codes

Der Server teilt dem Client über einen dreistelligen Code mit, wie die Anfrage verarbeitet wurde. Man unterscheidet fünf Kategorien:

- **1xx Information:** Die Anfrage wurde empfangen, die Bearbeitung dauert an. *Beispiel:* 100.
- **2xx: Erfolg:** Die Anfrage wurde erfolgreich bearbeitet. *Beispiel:* `200 OK` Die Standardantwort für erfolgreiche GET-Anfragen, `201 Created` Die Anfrage war erfolgreich und eine neue Ressource wurde erstellt (oft nach POST/PUT).
- **3xx Umleitung:** Weitere Schritte sind erforderlich, um die Anfrage abzuschließen.*Beispiel:* `301 Moved Permanently` Die angeforderte Seite/Datei befindet sich nun dauerhaft unter einer anderen Adresse.
- **4xx Client Fehler:** Die Anfrage war fehlerhaft oder kann nicht ausgeführt werden. *Beispiel:* `400 Bad Request` Die Anfrage war syntaktisch falsch (z. B. Fehler im JSON-Body). `401 Unauthorized` Der Zugriff erfordert eine Authentifizierung (z. B. Login fehlt).
- **5xx** Der Server hat die Anfrage eigentlich verstanden, konnte sie aber nicht verarbeiten. *Beispiel:* `500 Internal Server Error` Ein allgemeiner Fehler im Server (z. B. Absturz des SPS-Webservice-Skripts). `503 Service Unavailable` Der Server ist überlastet oder wird gerade gewartet.

URL (Uniform Resource Locator)

Eine URL (Uniform Resource Locator) ist die eindeutige Webadresse, die als Adresse einer Ressource (Webseite, Bild, Datei) im Internet dient. Z.B.

```
https://www.litec.ac.at/fileadmin/user_upload/00_TS_Anmeldeformular.dotx
```

Die Adresse besteht aus 3 Teilen.

- **Scheme:** Das Protokoll das verwendet wird. `http://`

- **Host:** Ort an dem der Server liegt. `www.litec.ac.at` Kann auch nur eine IP-Adresse sein. Z.B. `192.168.88.1`
- **Path:** Pfad zur gewünschten Resource. `/fileadmin/user_upload/00_TS_Anmeldeformular.dotx`

Query-Parameter

Parameter können direkt an die URL (Uniform resource locator) angehängt werden. Das wird oft für Filter oder IDs genutzt. Diese werden mit einem Fragezeichen getrennt an die URL angehängt und bestehen aus Key-Value Pairs `URL?key1=value1&key2=value2`. Der Server kann diese dann als Parameter weiter verwenden und dementsprechend für die Antwort nutzen.

Beispiel, es soll über ein Webservice die Temperatur von Sensor 42 in der Einheit celsius abgefragt werden:

```
/api/sensor?id=42&unit=celsius
```

HTTP Versionen

Die Entwicklung von HTTP zielte primär darauf ab, die Latenz zu verringern und die Effizienz bei vielen gleichzeitigen Anfragen zu steigern:

- **HTTP/1.1 (Standard)**: Der bewährte Text-Standard. Er erlaubt dauerhafte Verbindungen (Persistent Connections), sendet Anfragen aber strikt nacheinander, was bei vielen Dateien zu Warteschlangen führt.
- **HTTP/2 (Performance)**: Nutzt ein binäres Format statt Text. Durch Multiplexing können viele Anfragen gleichzeitig über eine einzige TCP-Verbindung gesendet werden, was das Blockieren (Head-of-Line Blocking) verhindert.
- **HTTP/3 (Modern/Speed)**: Ersetzt TCP durch das auf UDP basierende QUIC-Protokoll. Es eliminiert Verzögerungen beim Verbindungsaufbau und reagiert deutlich stabiler auf Paketverluste in Funknetzwerken oder mobilen Anwendungen.

Verschlüsselung

Um die Sicherheit der Datenübertragung zu gewährleisten, wird HTTP heute fast ausschließlich mit TLS (Transport Layer Security) kombiniert, was als HTTPS bezeichnet wird.

- Das Problem bei HTTP: Daten werden im Klartext übertragen. In einem Netzwerk könnte jeder Teilnehmer (z. B. im Firmen-WLAN) Passwörter oder sensible Sensordaten mitlesen (Sniffing).
- Die Lösung HTTPS: Bevor die eigentlichen HTTP-Daten fließen, bauen Client und Server einen verschlüsselten Tunnel auf. Dabei authentifiziert sich der Server mit einem Zertifikat, um sicherzustellen, dass der Client wirklich mit der echten SPS oder dem echten Server spricht (Schutz vor Man-in-the-Middle-Angriffen).
- Auswirkung auf die Nachricht: Die Anatomie der Nachricht (Request/Response) bleibt identisch, aber der gesamte Inhalt wird für Außenstehende unlesbar chiffriert.

REST - Der Architekturstil für moderne APIs

REST (Representational State Transfer) ist kein eigenes Protokoll, sondern eine Art „Regelwerk“, wie man HTTP effizient nutzt, um mit Ressourcen (z. B. Sensoren, Motoren oder Datenbanken) zu kommunizieren. Eine API (Schnittstelle), die diesen Regeln folgt, nennt man RESTful API.

1. **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
2. **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):
   - **C**reate = POST
   - **R**ead = GET
   - **U**pdate = PUT
   - **D**elete = DELETE
3. **Zustandslosigkeit**: Wie HTTP selbst ist auch REST zustandslos. Jede Anfrage enthält alle Informationen, die der Server braucht.
4. **Repräsentationen**: Eine Ressource kann in verschiedenen Formaten gesendet werden. In der modernen Technik ist JSON (JavaScript Object Notation) der absolute Standard, da es leichtgewichtig und für Mensch und Maschine gut lesbar ist.

Beispiel einer REST-Kommunikation

- **Anfrage:** GET /maschinen/foerderband/speed
- **Antwort (JSON)**: {“value”: 15.5, “unit”: “m/s”}

Quellen

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview
- https://www.rfc-editor.org/rfc/rfc9110.html
- HTTP The Definitve Guide. David Gourley and Brian Totty. 2002 O´Reilly & Assiciates.ISBN 1-56592-509-2