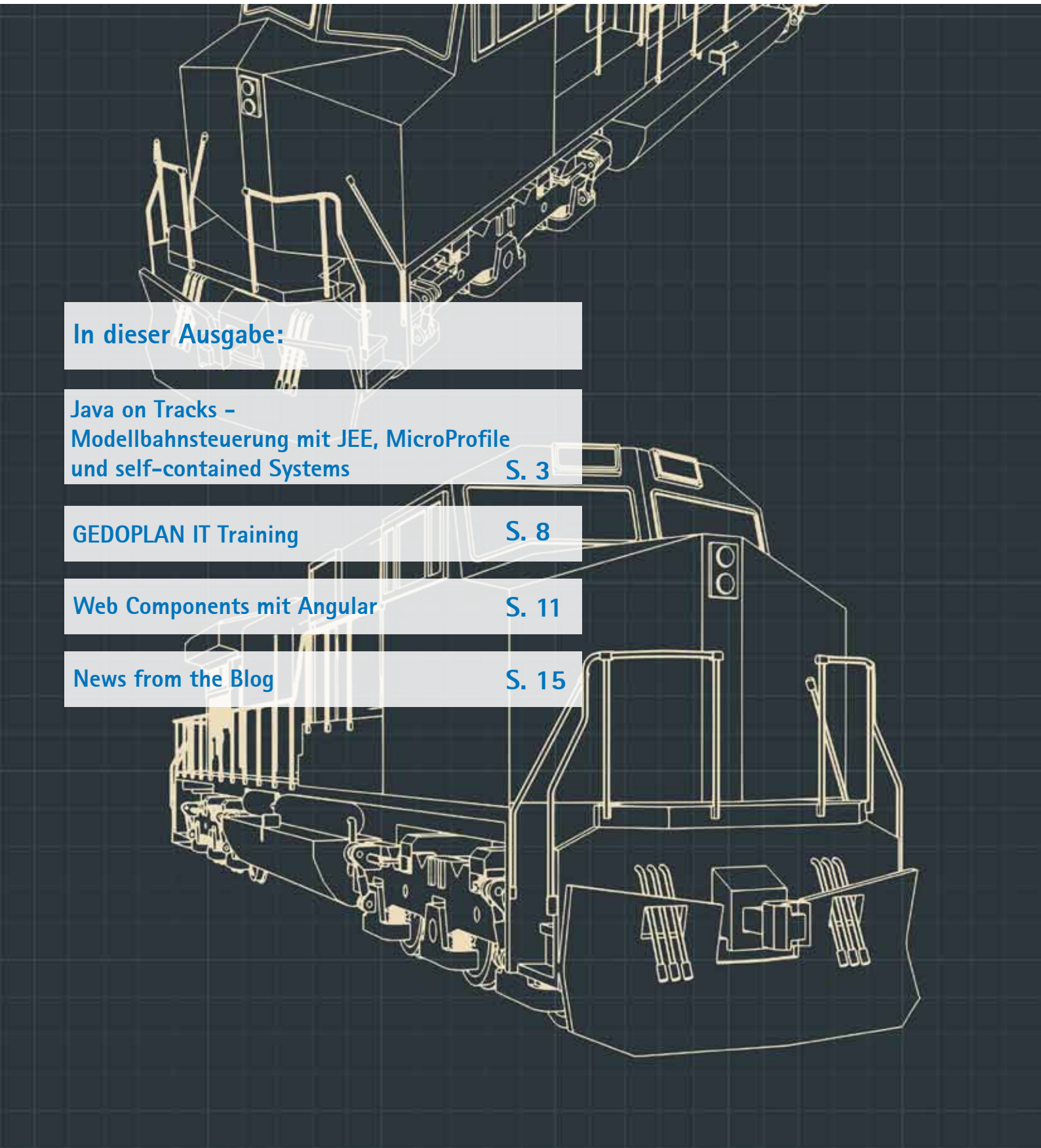
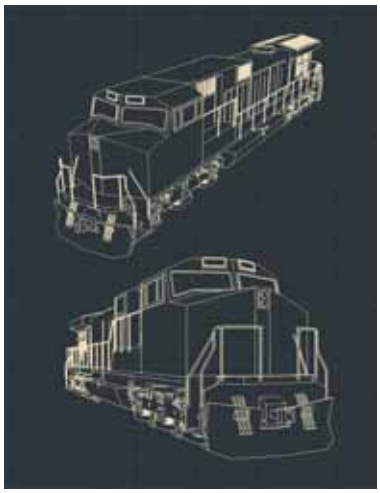
Ausgabe Frühjahr 2021

www.gedoplan.de

GEDOPLAN aktuell



In dieser Ausgabe:



Liebe Leserin, lieber Leser,

vor fast 10 Jahren haben wir schon einmal von unserem kleinen, internen Projekt "**v5t11**" berichtet. Wir haben die Modelleisenbahnsteuerung mittels JEE weiterentwickelt: weg von einer monolithischen Anwendung hin zu einer Version, die aus separaten deploy- und lauffähigen Services besteht. Dirk Weil, Geschäftsführer GEDOPLAN GmbH, zeigt, welche Vor- und Nachteile die Aufteilung der Anwendung in autonome Microservices mit sich bringt.

GEDOPLAN IT Training hat ein neues Angebot für Sie: "**Ask the Experts**" bietet Ihnen die Möglichkeit, mit unseren erfahrenen Trainern und Entwicklern zu einem vereinbarten Termin über Ihre Fragen zu einem bestimmten Fachgebiet zu sprechen.

Und GEDOPLAN IT Training hat einen neuen Dozenten. Wir stellen Ihnen Markus Pauer, einen erfahrenen JEE-Entwickler in einem Interview vor.

Wiederverwendbare Komponenten auch in unterschiedlichen Frameworks nutzen? Das stößt häufig an Grenzen. Dominik Mathmann, GEDOPLAN GmbH, zeigt Ihnen eine Möglichkeit, wie sich Angular-Komponenten in eine JSF-Anwendung integrieren lassen. Bühne frei für WebComponents...

Viel Spaß beim Lesen!

Ulrich Hake | ulrich.hake@gedoplan.de

Ulrich Hake

Termine

Expertenkreis Java

Thema: **Serverless Systems: The Future is Here**
Ort: remote
Termin: Donnerstag, 01.07.2021 | 18:00 - 19:30 Uhr
Referent: Sebastian Hesse

Thema: **Battle of the Languages: Java und Python im Wettstreit beim Lösen von Programmier-Challenges**
Ort: remote
Termin: Donnerstag, 27.05.2021 | 18:00 - 19:30 Uhr
Referent: Michael Inden

Thema: **ArchUnit: Testen von Architektur und Design**
Ort: remote
Termin: Donnerstag, 18.03.2021 | 18:00 - 19:30 Uhr
Referent: Thomas Much

Vorträge

Thema: **Mehr Dynamik bitte – Skriptsprachen in Java einbinden**
Ort: JAX
Termin: 04.05.2021
Referent: Dirk Weil, GEDOPLAN GmbH

Thema: **Dream-Team Jakarta EE + MicroProfile**
Ort: JAX
Termin: 04.05.2021
Referent: Dirk Weil, GEDOPLAN GmbH

Thema: **Java on Tracks**
Ort: IT-Tage 365
Termin: 24.03.2021
Referent: Dirk Weil, GEDOPLAN GmbH

Thema: **Dream-Team Jakarta EE + MicroProfile**
Ort: JavaLand
Termin: 17.03.2021
Referent: Dirk Weil, GEDOPLAN GmbH

Java on Tracks: Modellbahnsteuerung mit JEE, MicroProfile und self-contained Systems

Vor mittlerweile 9 Jahren ist in einem unserer Code Camps eine etwas ungewöhnliche Anwendung namens v5t11 entstanden: Statt mit - in unserem Business verbreiteten - relativ trockenen Finanz- oder Auftragsdaten zu hantieren, werden hier Züge auf die Reise geschickt, und zwar auf einer Modellbahn. Aber halt: Will ich hier über eine Anwendung aus dem Jahre 2012 berichten? Nein, natürlich nicht. Es geht vielmehr um die mittlerweile erfolgte Transformation der ursprünglich monolithischen Anwendung in eine Version, die aus separat deploy- und lauffähigen Services besteht.

Von Dirk Weil

Den Projektnamen muss man zuächst einmal erklären: **v5t11** steht für *Visual Traincontrol* - in dem Stil, wie wir IT-Nerds gerne Dinge abkürzen (*I18n, K8s, ...*). Die enthaltenen Zeichen haben aber auch noch einen bahnbezogenen Ursprung, nämlich in der Bezeichnung eines aus meiner persönlichen Sicht sehr ansprechenden Triebwagens, dem **VT 11.5**.

Abb. 1. Eine vierteilige TEE-Einheit im Verkehrsmuseum Nürnberg[1]

Diese Diesel-Triebzüge waren Paradezüge der DB im grenzüberschreitenden Verkehr (*Trans Europe Express - TEE*) in den 60ern und 70ern. Eine der letzten Einheiten konnte ich im vorletzten Jahr in bedauernswertem Zustand in Villingen-Schwennigen entdecken; sie ist mittlerweile abgewrackt (Abb. 2).

Als Modell lebt sie aber noch weiter, u. a. auf meiner Modellbahn!

Abb. 2. Triebkopf VT 11.5014 in Villingen-Schwennigen vor der Verschrottung

Aufgaben von V5T11

Die Projektsoftware bildet zwei Aspekte des Bahnbetriebs ab. Zum einen ist es die Sicht eines Stellwerks (Abb. 3). Hier wird der Gleis-Parcours visualisiert und werden Gleisbelegungen angezeigt. Weiterhin können Weichen und Signale gestellt sowie Fahrstraßen reserviert werden.

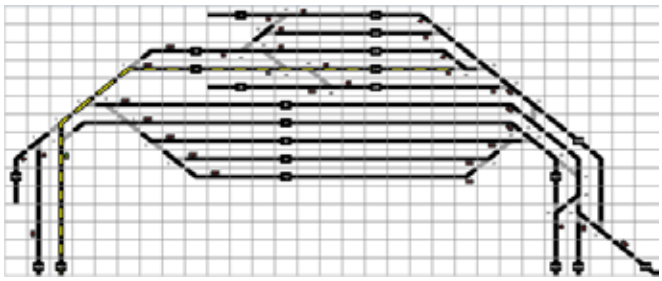


Abb. 3. Stellwerk

Die zweite Sicht ist die des Führerstands eines Fahrzeugs, mit der Geschwindigkeit, Fahrtrichtung, Beleuchtung und diverse andere Funktionen einer Lok gesteuert werden können (Abb. 4).

Abb. 4. Führerstand

Technische Basis: Digitalsteuerung Selectrix® und DCC

Um überhaupt mehrere Loks unabhängig voneinander steuern zu können, ist eine andere Elektrik notwendig, als wir sie ggf. von unserer ersten Spiel-Eisenbahn kennen. Damals dienten mehrere voneinander getrennte Stromkreise dazu, eine kleine Menge von Loks gleichzeitig fahren lassen zu können. Beim Einsatz einer Digitalsteuerung wird dagegen jedes Gleis der Anlage gleich mit Strom versorgt. Der Versorgungsspannung wird dabei ein Digitalsignal aufgeprägt, aus dem die Loks ihre Befehle entnehmen. Die Fahrzeuge werden somit in einem Bus-System betrieben, in dem sie jeweils ihre eigene eindeutige Adresse besitzen.
Dazu enthalten die Loks Decoder, die neben den Grundfunktionen wie Fahrstufe und Richtung mittlerweile eine große Anzahl von Zusatzfunktionen wie Fahrgeräusche, Innenbeleuchtung oder Bahnsteigansagen steuern (Abb. 5).

Abb. 5. Lok mit Decoder

In den letzten Dekaden wurden einige Digitalsysteme für Modellbahnen entwickelt, von denen sich aber i. W. zwei durchgesetzt haben: *Selectrix* von der Fa. Trix - mittlerweile Märklin - hatte bereits in den 80ern recht kleine Decoder, die sich auch für die von mir benutzte Spurweite *N* (Maßstab 1:160, 9 mm Gleisabstand) eigneten. Weitere Verbreitung hat mittlerweile der Standard *DCC* (_Digital Command Control), der von vielen Herstellern wie Fleischmann, Pico und auch Märklin eingesetzt wird.
Viele Digitalsysteme sind multiprotokollfähig, können also bspw. Selectrix- und DCC-Decoder gleichzeitig ansteuern.
Die Gleise werden üblicherweise mit Besetztmeldern überwacht, die über eine Strommessung feststellen, ob ein Gleis besetzt oder frei ist. Die Zuständen von 8 Gleisen werden als ein Byte zusammengefasst über ein Bussystem an die Zentrale gemeldet (Abb. 6).

Abb. 6. Besetztmelder

Für Weichen und Signale gehts umgekehrt: Hier werden die über den Bus angelieferten Bytes in Stellungen von Weichen und Signalen umgesetzt (Abb. 7).

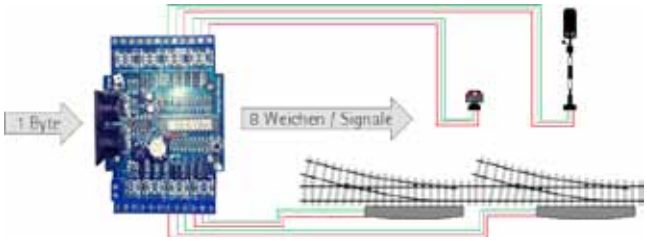


Abb. 7. Funktionsdecoder

Die fachliche Domäne

Kommen wir nach diesen Vorüberlegungen zurück zur eigentlichen Aufgabe. Die vor Jahren entstandene Version von **v5t11** ist als Java-EE-Anwendung zum Deployment auf einem WildFly aufgebaut. Dieser Deployment-Monolith litt an den gleichen Problemen wie viele geschäftliche Anwendungen:

- Die Codebasis wurde über die Zeit immer komplizierter. Obwohl eigentlich weitgehend getrennte Bereiche wie die Ansteuerung der Digitalsteuerung (technischer Aspekt) und die Reservierung von Fahrstraßen (vollkommen losgelöster fachlicher Aspekt) existieren, führt die Integration in eine große Anwendung nicht selten zu geringerer Übersichtlichkeit und stärkerer Verflechtung des Anwendungscodes.
- Ein Teilaspekt wie die erwähnte Fahrstraßensteuerung kann nicht modifiziert oder erneuert werden, ohne die Gesamtanwendung für einen Moment außer Betrieb nehmen zu müssen. Das machte die Weiterentwicklung von **v5t11** immer langsamer, zumal man ja bedenken muss, dass ein physikalisches System angesteuert wird, das seine Status-Informationen nicht in beliebig hoher Geschwindigkeit liefern kann.

Den ersten Punkt kann man natürlich durch eine entsprechende Organisation des Codes angehen. Der zweite legt eine Aufteilung der Anwendung in deparate Deployment-Einheiten nahe. Für beide ist ein passender Schnitt durch die Anwendungsdomäne notwendig.
Domain-driven Design ist eine Möglichkeit, zu einer Aufteilung der Fachlichkeit zu gelangen. Wir hatten in den vergangenen Ausgaben von GEDOPLAN aktuell diverse Artikel zu DDD, sodass ich Hinter-

gründe und Begrifflichkeiten hier nicht wiederholen, sondern stattdessen auf die früheren Artikel verweisen möchte.
Für die Gesamt-Domäne von v5t11 kommt eine Menge von Domänenobjekten aus der Bahnfachlichkeit wie auch aus der Fachlichkeit der Digitalsteuerung zusammmen. Den für die folgende Betrachtung relevanten Anteil zeigt Abb. 8.

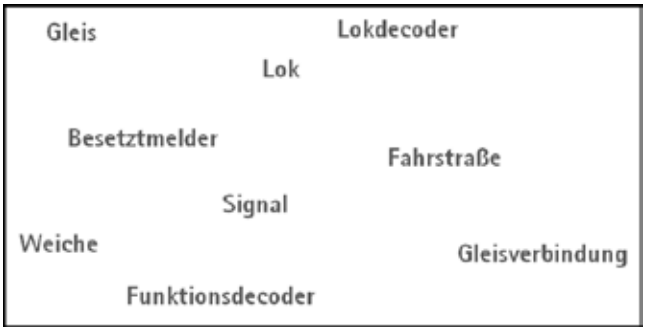


Abb. 8. Domänenobjekte von **v5t11**

Für eine Aufteilung in kleinere "Häppchen" ist jetzt also eine Gruppierung gesucht, die innerhalb jeder Gruppe eine möglichst hohe Bindung der Objekte miteinander erzeugt, wohingegen die Gruppen untereinander möglichst lose gekoppelt sind. Eine mögliche solche Aufteilung zeigt Abb. 9. Beachten Sie zunächst nur die normal gesetzten Begriffe. Die Aufteilung ist einigermaßen naheliegend: Die technischen Objekte wie Melder und Decoder liegen in der Statusverwaltung. Die damit eng zusammenhängenden Objekte **Gleis**, **Signal** und **Weiche** sind hier auch platziert worden [2]

Abb. 9. Bounded Contexts von **v5t11**

Der zweite Kontext umfasst die Fahrstraßen-Objekte **Gleisverbindung** - "Welches Gleis wird von einem anderen Gleis aus erreicht - ggf. mit den dazu nötigen Weichen- und Signalstellungen?" - und **Fahrstraße** - Kombination von Gleisverbindungen. Der dritte Kontext adressiert die Loks.
Nun sind die Kontexte aber nicht komplett voneinander getrennt: Die Fahrstraßenverwaltung darf Fahrstraßen nur dann reservieren, wenn die zu befahrenden Gleise frei sind. Sind die Voraussetzungen erfüllt, müssen die zugehörigen Weichen und Signale in die richtigen Stellungen gebracht werden. Der Kontext Fahrstraßenverwaltung benötigt die Domänenobjekte **Gleis, Signal** und **Weiche** somit auch! Ähnlich verhält es sich für **Lokdecoder**: Jede **Lok** in der Fahrzeugverwaltung hat einen **Lokdecoder**, der als technisches Objekt in der *Statusverwaltung* seine Heimat hat.
Ein gemeinsames Datenmodell ist mit Blick auf die gewünschte lose Kopplung der Kontexte und das Ziel "unabhängige Deployments" eher ungünstig. Sollen die Kontexte als eigenständige (Micro-) Services implementiert werden, so wird jeder Service seine eigene Datenhaltung betreiben. Die Daten müssen also repliziert werden. In den meisten Fällen - so auch hier - läßt sich für jedes Domänenobjekt ein "Heimatservice" ausmachen. Dies ist dann der "Single Point of Truth", der gesichert den Zustand des jeweiligen Objektes enthält. Andere Services erhalten "nur" Kopien, i. d. R. asynchron, arbeiten also mit "Eventual Consistency".
Für die konkrete Implementierung der kontextübergreifenden Zugriffe gibt es natürlich mehrere Möglichkeiten. Der asynchrone Austausch von Daten kann bspw. mittels Messaging geschehen. Die *Statusverwaltung* wird dazu jede Änderung einer Gleisbelegung oder einer Weichenstellung über einen Message Broker an alle anderen Services melden, die sich dafür interessieren.
Wenn umgekehrt Services auf Domänenobjekte Einfluss nehmen wollen, deren Heimat sie nicht sind, können sie sich entsprechender synchroner Schnittstellen bedienen, zumindest wenn die jeweilige Änderung sofort geschehen muss. Ansonsten wäre auch hier Messaging eine Option. In **v5t11** werden RESTful Services verwendet, da die auszulösenden Aktionen (z. B. Weiche stellen) stets unmittelbar ausgeführt werden müssen.
Die mehrfache Datenhaltung hat neben dem zweifellos vorhandenen Zusatzaufwand zur Synchronisation der Daten auch Vorteile: Die Objekte können in den verschiedenen Services unterschiedlich ausgeprägt sein. So hat bspw. ein **Gleis** in der *Statusverwaltung* diverse technische Attribute, die die Anbindung an die Digitalsteuerung beschreiben ("An welchem Ausgang welchen Decoders hängt das Gleis?"), während es in der *Fahrstraßenverwaltung* nur einfach "belegt" oder "frei" ist.

v5t11-Implementierung als Self-contained Systems

Der wie beschrieben aufgeteilte ehemalige Monolith liegt nun in Form mehrerer separat lauffähiger Microservices vor. Für die persistenten Daten wird aus Gründen des Ressourcenverbrauchs (und auch der Bequemlichkeit ...) nur eine PostgreSQL-Instanz eingesetzt, in der aber jeder Service ein eigenes, abgetrenntes Schema benutzt.
Da der Monolith als Java-EE-Anwendung implementiert war, lag es nahe, bei diesem Programm-Modell zu bleiben. Statt dem klassischen Serverdeployment kommt nun aber eine Micro Runtime zum Einsatz, die den Serveranteil mit der Anwendung integriert. Mit Quarkus[3] steht dafür ein mittlerweile sehr populäres Framework zur Verfügung, das durch sein stark gewachsenes Ökosystem und die vollständige Unterstützung von MicroProfile[4] eine hervorragende Basis für **v5t11** darstellt. Quarkus war zwar schon mehrfach Thema in der *GEDOPLAN aktuell* und auch in unserem *Expertenkreis Java,* aber wir werden sicher weiter über dieses interessante Framework berichten - stay tuned!

Im Einzelnen besteht **v5t11** nun aus den folgenden Microservices:

- *Statusverwaltung* **v5t11-status**
  - Anbindung der Digitalzentrale (Abb. 10) mit serieller Schnittstelle / USB
    Hier kommt die Open-Source-Bibliothek **jSerialComm [5]** zum Einsatz
  - Abstraktion Selectrix-"Bytes" < = > Gleise, Signale, Weichen, ... (Abb. 11)

Abb. 10. Zentrale mit USB-Schnittstelle

Abb. 11. Abstraktion Digitalinformation zu Gleisen, Signalen etc.

Die *Statusverwaltung* bietet den restlichen Services ein REST-API zur synchronen Steuerung an:

```
@Path("signal")
@Dependent
public class SignalEndpoint {
    @PUT
    @Path("{bereich}/{name}")
    @Consumes(MediaType.MEDIA_TYPE_WILDCARD)
public void putSignalStellung(@PathParam("bereich")
                              String bereich,
```

Für die asynchrone Replikation von Zustandsdaten nutzt **v5t11** MicroProfile Reactive Messaging[6]. Bei jeder Statusänderung wird eine Methode wie diese aufgerufen:

```
@Inject @Channel("gleis-out")
Emitter<String> gleisEmitter;

public void publish(Gleis gleis) {
String json = ...
    this.gleisEmitter.send(json);
```

Der **Channel** wird in der Anwendungskonfiguration **application.properties** mit einem Topic des MQTT-Brokers Eclipse Mosquitto [7] verknüpft:

```
mp.messaging.outgoing.gleis-out.connector
        =smallrye-mqtt
mp.messaging.outgoing.gleis-out.host=...
mp.messaging.outgoing.gleis-out.port=...
mp.messaging.outgoing.gleis-out.topic=gleis
```

- Fahrstraßenverwaltung v5t11-fahrstrassen
  - Reservieren und Freigeben von Fahrstraßen
  - Fahrstraßenauflösung bei/nach Zugdurchfahrt
  - Vorsignalautomatik: Vorsignale folgen dem nächsten Hauptsignal im Fahrweg (Abb. 12)

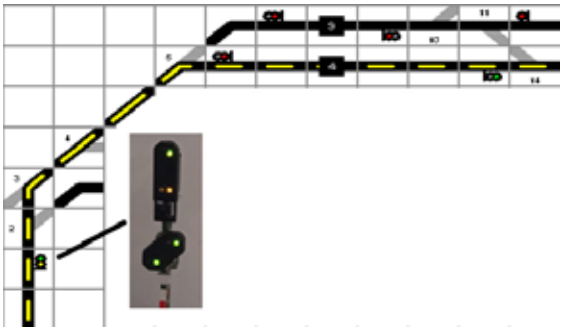


Abb. 12. Fahrstraße mit Vorsignal

Hier finden sich die "Gegenstücke" der oben angedeuteten Kommunikationswege. REST Calls lassen sich sehr elegant und kompakt mit MicroProfile Rest Client formulieren:

```
@RegisterRestClient(configKey = "v5t11.status")
public interface StatusGateway {
    @PUT
    @Path("signal/{bereich}/{name}")
    @Consumes(MediaType.TEXT_PLAIN)
    void signalStellen(@PathParam("bereich")
                       String bereich,
```

Der asynchrone Empfang von Meldungen ist ebenfalls mit wenig Code zu realisieren:

```
@Incoming("gleis-in")
void gleisChanged(byte[] msg) {
    String json = new String(msg);
...
```

- Fahrzeugverwaltung **v5t11-fahrzeuge**
  - Konfigurieren und Steuern von Fahrzeugen (Loks)

Dieser Service ist der erste mit einem UI (Abb. 13). Dabei kommt interessanterweise JSF zum Einsatz, ein Standard zum serverseitigen Seitenaufbau, der in der letzten Zeit in Misskredit geraten ist, weil er vermeintlich zu schwergewichtig sei. Im Mainstream befinden sich eher JavaScript Frameworks wie Angular oder React, mit denen si-

cher hochgradig ansprechende clientseitig gerenderte Benutzeroberflächen bereitgestellt werden können. Wird das letzte Quäntchen UX jedoch nicht benötigt, kommt man mit JSF sehr schnell zu sehr guten Ergebnissen, ohne den Zusatzaufwand einer weiteren Programmiersprache inkl. eigenem Buildsystem, Tooling etc. in Kauf nehmen zu müssen. Für Quarkus bietet das MyFaces-Projekt eine JSF-Extension an [8].

Abb. 13. JSF-UI der Fahrzeugverwaltung

- Visualisierung **v5t11-leitstand**
  - Anzeige von Parcours und Statusinfos
  - Stellen von Signalen und Weichen
  - Reservieren von Fahrstraßen

Der bislang letzte Service dient als UI für die Fahrwegesteuerung und ist damit ein gemeinsames UI für die Statusverwaltung und die Fahrstraßensteuerung. Die stärkere Kopplung musste hier akzeptiert werden, da die Informationen sich nunmal in der Sicht eines Stellwerks integrieren.
Aus historischen Gründen nutzt die Anwendung derzeit noch ein in der Zeit des Monolithen entstandes Swing-UI (Abb. 3). In einem der nächsten Entwicklungsschritte wird es sicher in ein Web-UI umgebaut

- weitere Services in Planung
  - Fahrzeug-Verfolgung: "Welches Fahrzeug steht/fährt wo?"
  - Teilautomatisierung: Automatisches Abstellen im Schattenbahnhof, Ausblenden der Geräusche in unsichtbaren Abschnitten

Erfahrungen, Rückblick

- Die erste EE-Version von v5t11 war ein Monolith auf Basis von Java EE 6 und JBoss 7.
- Um leichter und schneller Teile der Anwendung weiter entwickeln zu können, ohne stets die Gesamtanwendung redeployen zu müssen, wurde eine Aufsplittung in Self-contained Systems vorgenommen.
  - Pro: Überschaubare Einheiten, Entkopplung der Daten modelle, zügiges (Re-) Deployment.
  - Contra: Teilredundante Daten, Kommunikationsaufwand.
- Als "Trägersystem" wurde Quarkus gewählt.
  - Pro: JEE-Programmmodell, schnelle Entwicklungszyklen, schnell wachsendes Ökosystem.
- MicroProfile spielt neben JEE eine tragende Rolle bei der Implementierung.
  - Pro: Simple und leistungsfähige Bausteine (Config, Rest Client, ...), leichte Einbindung von Message Brokern (MQTT, AMQP, ...).

Ausblick

Die Aufteilung der Anwendung in autonome Microservices birgt ein Laufzeit-Risiko: Fällt einer der Services aus, beeinflusst dies natürlich auch andere Services bis hin zur kompletten Fehlfunktion. Bei einem Monolithen ist der Nachteil system-immanent, aber bei einem Bündel von Microservices erwarten wir eine höhere Stabilität. In einem späteren Artikel werde ich auf die Möglichkeiten von Fault Tolerance eingehen und ihren Einsatz in **v5t11** beleuchten.
Weitere Infos
Den Programmcode von **v5t11**, der hier nur in wenigen Ausschnitten gezeigt werden konnte, finden Sie auf GitHub [9].
Und wenn ich Sie hier ein wenig für DDD, Java EE / Jakarta EE, Quarkus und MicroProfile begeistern konnte, schauen Sie doch mal in unser umfassendes Seminarangebot [10].

---

1. http://commons.wikimedia.org/wiki/File:601_Verkehrsmuseum_Nuernberg_11092010_complete_train.JPG
2. Dies ist eine willkürliche Designentscheidung. Es hätte auch ein separater Kontext für Gleis etc. aufgestellt werden können.
3. quarkus.io
4. microprofile.io
5. fazecast.github.io/jSerialComm/
6. github.com/eclipse/microprofile-reactive-messaging
7. mosquitto.org
8. github.com/apache/myfaces/tree/master/extensions/quarkus/
9. github.com/dirkweil/v5t11
10. gedoplan.de

Dirk Weil [dirk.weil@gedoplan.de]
- Geschäftsführer der GEDOPLAN GmbH -
Fachbuchautor, schreibt Artikel für Fachmagazine, hält Vorträge und leitet Seminare und Workshops zu diversen Java-SE-/EE-Themen

GEDOPLAN
IT Training

Markus Pauer arbeitet seit 2004 als Java Enterprise-Entwickler, Softwarearchitekt und Trainer. Für Gedoplan ist er seit 2020 als Entwickler und Trainer tätig. Als Fullstack-Entwickler kennt er sowohl die Probleme der Backend- als auch die Tücken der Frontend-Entwicklung.

Herr Pauer, Sie sind seit 17 Jahren als Entwickler im JEE-Bereich tätig. Im Laufe der Jahre sind immer neue Technologien und Programmiersprachen populär geworden und haben sich auch im Bereich der Enterprise-Entwicklung etabliert. Worin besteht Ihrer Meinung nach bei einem Dinosaurier wie JEE die Daseinsberechtigung in der heutigen Zeit?

Wenn ich eines im Laufe meiner Zeit als Entwickler gelernt habe, dann ist es eine Software nachhaltig zu bauen. Dazu zählt neben einer sauber strukturierten Codebasis auch eine Architektur, die sich gut und richtig anfühlt. Und genau das steht mir mit Jakarta EE zur Verfügung. Ich brauche nur eine einzige Abhängigkeit zur Plattform API und ich kann starten.

Jetzt wird der ein oder andere sagen: Aber das hat doch nichts mit der verwendeten Technologie oder der Programmiersprache zu tun?

Da kann ich nur mit einem klaren Jein antworten. Die Jakarta EE-Plattform gibt mir eine gewisse "Sicherheit". Die Konzepte der einzelnen Standards sind immer ähnlich. Als Basis dient immer ein Mehrschichtenmodell, das ich für den Aufbau meiner Anwendung nutzen kann. Ich habe nie das Gefühl, wenn ich eine Anwendung längere Zeit nicht mehr betreut habe, dass ich mich erstmal lange wieder einfinden muss.

Welche Trends sehen Sie in der Entwicklung mit Jakarta EE, die die Zukunft dieses Standards beeinflussen werden?

Da sehe ich ganz klar zwei Trends, die es geschafft haben, innerhalb kürzester Zeit uns Jakarta EE-Entwicklern die Arbeit zu erleichtern. Zum einen ist es die Schaffung eines de facto-Standards für Microservices innerhalb der Microprofile Working Group. Der Zusammenschluss von Herstellern zur Definition eines herstellerunabhängigen Standards ist aktuell zur Version 4.0 gereift. Das Microprofile 3.2 lässt sich mittlerweile in fast allen Enterprise-Servern verwenden.
Microprofile baut dabei auf die Jakarta EE-Standards CDI, JAX-RS, JSON-P und JSON-B auf und stellt ihnen weitere Technologien wie beispielsweise Metrics oder Health-Checks zur Seite. Gerade diese Vorgehensweise, die Basis aus dem Jakarta EE-Standard zu verwenden, bietet eine enorme Flexibilität in der Enterprise-Entwicklung.
Zum anderen geht Quarkus als das Jakarta EE Runtime Framework einen ähnlichen Weg. Ich setze nicht mehr einen vollständigen EE-Server als Laufzeitumgebung voraus, sondern aktiviere nur die Technologien, die ich für meine Enterprise-Anwendung benötige. Auch damit fühle ich mich als JEE-Entwickler wohl. Ich kann meine Anwendungen wie bisher aufbauen und kann entscheiden, welche Laufzeitumgebung für meine Anwendung anschließend die richtige ist. Ich denke, diese beiden Technologien werden in Zukunft Jakarta EE am meisten beeinflussen.

Was ist mit der Jakarta EE-Spezifikation? Deren Entwicklung zeichnete sich in letzter Zeit nur durch "organisatorische" Änderungen aus. Was sind da Ihre Hoffnungen für die Zukunft?

Die Trennung von Oracle und die Übernahme und Überarbeitung des Standardisierungsprozesses in der Eclipse Foundation hat insgesamt viel Zeit in Anspruch genommen. Auch aktuell hat man noch nicht so recht das Gefühl, dass die Weiterentwicklung Fahrt aufnimmt. Das Ziel der aktuell in Entwicklung befindlichen Version 9.1 ist lediglich die Unterstützung für Java SE 11.
Erst das nächste Hauptrelease wird wohl einige lang ersehnte Neuerungen mitbringen. Da ist unter anderem die Definition einer einheitlichen API für den Zugriff auf NoSQL-Datenbanken zu nennen. Das würde Jakarta EE endlich um eine Möglichkeit erweitern, auch nicht-relationale Datenbanken in einer einheitlichen Art und Weise verwenden zu können. Aber auch schon bestehende Standards sollen mit dem neuen Release entschlackt werden. Die in Jakarta Server Faces als deprecated markierten Teile, wie beispielsweise die Managed Beans, sollen entfernt werden. An dieser Stelle soll verstärkt CDI zum Einsatz kommen.
Aber dabei wird es nicht bleiben. Die zuvor genannten Technologien werden die Weiterentwicklung von Jakarta EE beeinflussen. Dazu gab es erst kürzlich ein Statement der führenden Köpfe im Standardisierungsprozess. Diese erklärten, wie die Zusammenarbeit dieser beiden Prozesse zukünftig aussehen wird. Jakarta EE bleibt der stabile Kern, wobei Microprofile den Fokus auf Microservices setzt und etwas schneller und innovativer arbeiten wird. Das war eine wichtige Aussage für mich als Enterprise-Entwickler.

Für die Entwicklung von Backend-Anwendungen bietet Jakarta EE sehr umfangreiche Techniken. Beim Frontend sieht es allerdings nicht so rosig aus. Daher die Frage: Sollte man heutzutage noch Webanwendungen mit JSF entwickeln? Oder stellt sich die Frage aufgrund der modernen alternativen Javascript-Frameworks erst gar nicht?

Im Bereich der Unternehmensanwendungen, in denen ich überwiegend tätig bin, wird diese Frage in der Tat häufig gestellt. Die Entscheidung ist in der heutigen Zeit nicht leicht zu treffen. Oftmals sollen bestehende Backend-Systeme durch Frontends erweitert werden. Oder es sollen "ältere" Frontends nochmal überarbeitet und modernisiert werden.
Ich frage dann zunächst immer nach dem vorhandenen Know-how für das geplante Projekt. Meist kommt dann zur Antwort, dass es sich überwiegend um Java-Entwickler handelt, die ein bisschen etwas mit Javascript gemacht haben. An dieser Stelle habe ich dann schon einen wichtigen Punkt, der gegen die Verwendung eines Javascript-Frameworks für die Frontend-Entwicklung in solchen Projekten spricht. Nicht, dass ich den Entwicklern das nicht zutraue, aber neben dem Erlernen der Javascript-Sprache ist es notwendig,

sich darüber hinaus in das Framework selbst und die damit einhergehenden Werkzeuge einzuarbeiten. Gerade dieser Aspekt wird in den meisten Projekten unterschätzt.

Als weiteres Argument für den Einsatz von JSF spricht dann meist das Einsatzgebiet der geplanten Webanwendung. Oftmals handelt es sich um Anwendungen, die vollständig nur innerhalb eines Unternehmensnetzwerkes verwendet werden sollen. Oder es sind Anwendungen für Endkunden, die aber keine zusätzlichen Anforderungen wie Offline-Fähigkeit oder natives Design erfordern. Diese Aspekte sprechen dann einfach für einen Einsatz von JSF als Frontend-Technologie.
Jetzt entsteht wahrscheinlich gerade der Eindruck, dass ich diese Javascript-Welt überhaupt nicht mag. Das ist allerdings gar nicht der Fall. Es gibt genügend Fälle, in denen eine JSF-basierte Webanwendung einfach nicht passt.

Neben der Entwicklung von Enterprise-Anwendungen beschäftigen Sie sich auch mit anderen Themen. Dabei geht es um die Integration von Geschäftsprozessen in Anwendungssysteme. Worum geht es da genau?

In meiner Zeit als Entwickler musste ich immer wieder feststellen, dass Prosesse innerhalb einer Anwendung häufig im Verborgenen ablaufen. Diese Abläufe sind für den Anwender intransparent. Dabei kann es sich um eine einfache Prüfung einer Rechnung oder um einen vollständigen Antragsprozess handeln. Das vollständige Nachvollziehen dieser Abläufe ist nur vom Entwickler zu leisten und das hat mich immer gestört.

Vor 10 Jahren lernte ich Activiti kennen. Dabei handelt es sich um eine Workflow-Engine, die Prozesse auf Basis eines BPMN-Diagramms ausführen kann. BPMN steht für Business Process Model and Notation und ist mittlerweile zu einem ISO-Standard geworden. Mit ihr können nicht nur Prozesse grafisch modelliert werden, sondern auch mit Hilfe einer XML-basierten Beschreibung ausgeführt werden. Die fachliche Modellierung des Prozesses erfolgt damit auf der gleichen Basis wie die spätere Ausführung.
Das hat mich damals wie heute fasziniert. Mittlerweile ist die aus einem Activiti-Fork entstandene Camunda BPM-Plattform sehr populär geworden. Diese Workflow-Engine kann auch auf einem Jakarta EE Server wie beispielsweise Wildfly betrieben werden. Sie wird dort als Modul eingebunden. Die Integration erfolgt über einen EJB-Client und CDI-Beans in die Prozessanwendung. Der Rest der Anwendung lässt sich wie gewohnt auf Basis von Jakarta EE entwickeln. Eine Verbindung zum BPMN-Modell erfolgt beispielsweise durch die Nutzung einer Expression-Language, wie sie auch bei JSF zum Einsatz kommt.
In meinen Projekten versuche ich immer, wenn möglich, den Einsatz einer Workflow-Engine zu prüfen. Gerade wenn es um die Abbildung von Prozessen geht, die der Endbenutzer auch verstehen muss, ist der Einsatz meist sinnvoll. Denn neben der Integration als Modul im Anwendungsserver besteht auch die Möglichkeit, diese Engine eingebettet in der Anwendung zu betreiben. Damit können dann auch nur Teile von Anwendungen mit einer Workflow-Engine ausgestattet werden.

Herr Pauer, vielen Dank für den interessanten Einblick in Ihre Sicht auf die aktuellen Jakarta EE-Themen.

Kurse zum Thema:

Jakarta EE Intensivkurs
Schulung Grundlagen von Jakarta EE / Java EE kompakt aufbereitet für WildFly, Open Liberty oder Payara

Jakarta Persistence komplett
Datenbankzugriffe mit Eclipselink und Hibernate

Microservices mit Quarkus - kompakt
Jakarta EE mit MicroProfile kombinieren, um Microservices und verteilte Systeme für Quarkus zu entwickeln

Vorträge zum Thema:

Dream-Team Jakarta EE und MicroProfile
JEE und Micro – kein Widerspruch!
Microprofile-Anwendungen mit Quarkus
Vom Monolithen in die verteilte JEE-Welt

Infos: https://gedoplan.de/java-events/

Houston, haben wir ein Problem? -> Ask the Experts!

Sie stehen vor einem Problem, vor einer wichtigen Entscheidung oder vor neuen Herausforderungen? Oder Sie haben nur eine Frage zu einem bestimmten IT-Thema? Wir möchten Sie unterstützen und beraten Sie gerne in einem kostenfreien, unverbindlichen Experten-Online-Gespräch.

Schicken Sie uns bitte eine Mail mit einem Terminvorschlag und dem Thema, Frage oder Problem - wir bestätigen Ihnen diesen Termin und schicken Ihnen eine Einladung für Zoom oder MS-Teams:
ask-the-experts@gedoplan.de

Frontend-Entwicklung

Angular und React? Oder doch VueJS oder Svelte? Oder lieber den klassischen JSF-Weg gehen und auf Jakarta EE setzen? JavaScript-Experte Peter Hecker und/oder Jakarta-EE-Spezialist Markus Pauer stehen Ihnen gerne bei Fragen zum Thema Frontend-Entwicklung zur Verfügung.

Peter Hecker ist seit über 30 Jahren im IT-Bereich tätig und hält (fast) alle JavaScript-Framework-Schulungen wie Angular, Svelte, React, TypeScript und VueJS bei GEDOPLAN.

Markus Pauer arbeitet seit 20 Jahren im Jakarta EE-Backend- sowie -Frontend-Bereich als Consultant, Trainer und Softwareentwickler.

Backend-Entwicklung

Die Entwicklung von Enterprise-Anwendungen hat sich im Laufe der Jahre stark verändert. Webservices bedeuten heutzutage nicht nur mehr SOAP-Aufrufe zwischen den Systemen zu realisieren. Die Themen Restful-Services, Microservices, Container, Health-Checks, Metriken etc. sind nur einige, mit denen Backend-Entwickler konfrontiert werden. Jakarta EE bietet in Kombination mit Microprofile Lösungsmöglichkeiten an. Aber wie passen diese Technologien in meine IT-Landschaft und wie können diese integriert werden? Oder soll ich doch lieber Spring Boot in meinen Projekten nutzen?

Für Spring-Boot-Anwendungen steht Ihnen Dominik Mathmann Rede und Antwort. Er beherrscht Jakarta EE und das Spring Boot Framework.

Markus Pauer arbeitet seit 20 Jahren im Jakarta EE-Backend- sowie -Frontend-Bereich als Consultant, Trainer und Softwareentwickler.

DevOps

Haben Sie Fragen rund um das komplexe Thema Java DevOps? Beispielsweise wie Sie die Zusammenarbeit zwischen "development and operations" verbessern können, welche Tools eingesetzt werden und welche Fallstricke dabei bestehen? Ihnen stehen 2 Experten zur Verfügung.

Hendrik Jungnitsch ist Softwareentwickler und langjähriger Dozent, u. a. für Java DevOps: Er hat unsere DevOps-Kurse entwickelt und erstellt.

Dirk Weil ist Java-EE-Fachbuchautor und einer der bekannten Jakarta-EE-Enthusiasten in Deutschland. Er spricht regelmäßig auf JAX, Java-Land und anderen IT-Konferenzen.

Sprechen Sie uns gerne bei Bedarf an: ask-the-experts@gedoplan.de

Web Components mit Angular

Wiederverwendbare Komponenten zu schreiben, das ist das Ziel bei der Entwicklung von WebComponents. Nun ist die Idee natürlich nicht neu und in nahezu allen Frameworks, sei es JavaServer Faces, Angular oder React versuchen wir, die wiederverwendbaren Teile der Anwendung zu identifizieren und so zu implementieren, dass sie an möglichst vielen Stellen einsetzbar sind. Das geschieht jedoch immer in Bezug auf das verwendete Framework, so lässt sich eine Angular-Komponente nur schwer in eine JSF-Anwendung integrieren. Also doch nicht so wiederverwendbar? Bühne frei für WebComponents...

Von Dominik Mathmann

```
<h1>Willkommen</h1>
<gedoplan-login></gedoplan-login>
```

Wäre es nicht fantastisch, wenn der Browser nun an dieser Stelle ein Login-Screen anzeigen würde, der voll funktionsfähig für die Validierung einer Anmeldung sorgt? Ganz gleich, ob meine HTML-Seite, die dort verwendet wird, Teil einer JSF-, Angular-, React- oder statischen HTML-Anwendung ist? Seit einiger Zeit ist das kein Wunschdenken mehr, denn im Jahr 2012 veröffentlichte die W3C eine entsprechende HTML-Spezifikation, die es Entwicklern erlaubt, solche autarken Komponenten selber zu definieren.

Web Components

Die Spezifikation der Web Components stützt sich auf eine ganze Reihe von weiteren Spezifikationen, die in den letzten Jahren definiert und inzwischen in nahezu allen Browsern auch zur Verfügung stehen. An dieser Stelle werfen wir lediglich einen Blick auf zwei der grundlegenden Spezifikationen, die wir auch bei der Entwicklung unserer Komponente mit Angular direkt und indirekt verwenden werden.

Shadow DOM

Das "Document Object Model" stellt im Speicher des Browsers die Objekt-Repräsentation unserer ausgelesenen HTML-Seite dar. So wird aus jeder Komponente, die wir im HTML deklarieren (`<button>`, `<div>`, `<table>` ...) eine Objektinstanz erzeugt, die in einer Baumstruktur abgelegt wird. Auf diese hierarchisch aufgebaute Struktur lässt sich per JavaScript zugreifen, um Änderungen durchzuführen, sich auf Benutzerinteraktionen zu registrieren oder sogar neue Elemente zu erzeugen. Zur Anzeige gebracht wird also diese Baumstruktur, nicht etwa die im HTML deklarierten Elemente. Dabei stützen wir uns auf die DOM-API, die entsprechende Methoden anbietet.

```
<html lang="de">
<head>
    <title>DOM Demo</title>
</head>
<body>
<script>
    const title = document.createElement("h1")
    title.append(document.createTextNode("Hello World."))
    title.addEventListener("mouseover", () => alert("Hello World"))

    document.body.append(title)
</script>
</body>
</html>
```

Ein Shadow DOM funktioniert nun auf exakt dieselbe Art und Weise und bietet all die Möglichkeiten, die auch der "normale" DOM bereitstellt. Der entscheidende Unterschied besteht darin, dass ein Shadow DOM einen eigenen Gültigkeitsbereich erzeugt. Dadurch erhalten wir die Möglichkeit, das verwendete Markup und Styling vom Rest der Seite zu separieren, um ungewollte Seiteneffekte zu verhindern. Dabei ist das Vorgehen ganz einfach. Wir deklarieren an einem (nahezu) beliebigen HTML-Element ein solches Shadow DOM und machen es damit zum "host". Anschließend fügen wir die gewünschten Elemente und Styling diesem Shadow DOM hinzu, anstatt dem Element selbst.

```
<script>
    const footer = document.createElement("footer");
    const footerShadow = footer.attachShadow({mode: "open"});
    const footerStyle = document.createElement("style");
    footerStyle.innerHTML="h1{ color: red }";
    const footerTitle = document.createElement("h1");
    footerTitle.append(document.createTextNode("by GEDOPLAN"));
    footerShadow.append(footerTitle);
    footerShadow.append(footerStyle);
    document.body.append(footer);
</script>
```

Jegliche Styling-Angaben innerhalb des Shadow DOMs beziehen sich nun ausschließlich auf Elemente, die innerhalb des DOMs deklariert sind und auch Selektionen (`document.querySelectorAll("h1")`) liefern nur Elemente aus dem Dokument selbst. Diese Kapselung verhindert also ungewollte Änderungen innerhalb solcher Komponenten. Dabei ist es aber durchaus möglich, auf die Komponenten zuzugreifen, die innerhalb des Shadow DOMs liegen, dies müssen wir nun jedoch ganz explizit über den Zugriff auf den Shadow Host implementieren:

```
document.querySelector('footer').shadowRoot.
querySelectorAll('h1')
```

Custom Elements

Der eigentlich spannende Teil ist natürlich nun die Definition der eigenen Komponenten. Dahinter steckt erst einmal nichts anderes als die Definition einer Klasse in JavaScript:

```
class GedoplanLogin extends HTMLElement{...}
```

In diesem Beispiel ist die Basis unserer Klasse das ganz allgemeine `HTMLElement`, da unsere Komponente, die wir entwickeln, eine Komposition aus vielen verschiedenen Elementen sein wird und damit keine der Standard-Elemente direkt erweitert. Solche allgemeinen Komponenten werden auch `autonomous custom elements` genannt, im Gegensatz zu Komponenten, die lediglich Standard Komponenten erweitern und z. B. einen Button um weitere Funktionalität ergänzen (customized component). Nun müssen wir uns natürlich darum kümmern, dass unsere Komponente auch mit Leben gefüllt wird. Dazu werden bei der Verwendung von nativem JavaScript die Techniken

eingesetzt, die wir schon gesehen haben. Wir erzeugen also einen Shadow DOM und fügen diesem die gewünschten Elemente hinzu. In diesem Beispiel verwenden wir dazu ein weiteres Feature, welches im Zuge der WebComponent-Spezifikation Einzug in unsere Browser gehalten hat: Templates

```
class GedoplanLoginComponent extends HTMLElement {
    templateContent =
      <label for="username">username</label>
      <input id="username">
      <label for="password">password</label>
      <input id="password" type="password">
      <button type="submit">login</button>

    connectedCallback() {
      this.attachShadow( init: {mode: "open"})

      const template = document.createElement( tagName: "template");
      template.innerHTML = this.templateContent;
      this.shadowRoot.append(template.content.cloneNode( deep: true))
    }
}
```

Damit sind wir fast fertig. Lediglich die Registrierung unserer Komponente fehlt noch:

```
customElements.define("gedoplan-login", GedoplanLoginComponent);
```

Das reicht aus, um dem Beispiel im Eingangstext Leben einzuhauchen. Nun haben wir bisher lediglich reines JavaScript verwendet, ohne auf irgendwelche Frameworks zurückzugreifen. Für die spätere Verwendung spielt das ja auch keine Rolle. Trotzdem werden viele Entwickler zusätzlich Bibliotheken und/oder ganze Frameworks einsetzen wollen, um deren Features auch bei der Entwicklung von solchen WebComponents nutzen zu können.

Angular > WebComponent, Schritt für Schritt

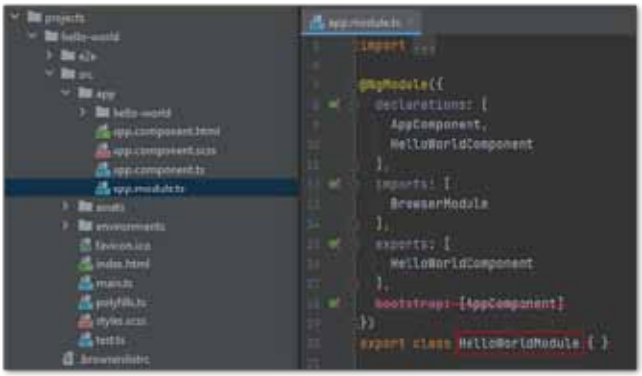
Wir beginnen ganz klassisch mit der Verwendung der Angular-CLI, um ein neues Projekt zu generieren. Im Grunde müssen wir dabei noch nichts Besonderes berücksichtigen, werden aber in unserem Beispiel keine Anwendung generieren lassen:

```
ng new ged-webcomponents --create-application false
```

Damit erhalten wir einen sogenannten Workspace mit den grundlegenden Konfigurationen, die Angular CLI benötigt, ohne aber bereits eine konkrete Anwendung generieren zu lassen. Anwendungen (application), genauso wie Bibliotheken (library), lassen sich nun aber natürlich wie gewohnt über die Kommandozeile erstellen. Der Vorteil bei diesem Vorgehen: Wir haben einen zentralen Workspace mit unseren Angular-Anwendungen und Bibliotheken, die sich gegenseitig referenzieren können, ohne eine Build Pipeline inklusive eigenem NPM Repository einrichten zu müssen.

```
ng generate application HelloWorld
ng generate application Rating
ng generate application ShowCase
```

Auf diese Weise erhalten wir nun unsere "gewohnte" Dateistruktur für Angular-Anwendungen für jede Anwendung unterhalb des projects-Ordners. Bei Verwendung der Angular CLI-Befehle müssen wir nun lediglich darauf achten uns innerhalb des Projektes zu befinden, für das wir den Befehl ausführen wollen oder die Kommandozeile um den Namen des entsprechenden Projektes ergänzen (z. B. ng serve showCase). Zum weiteren Vorgehen hat es sich nun bewährt, (mindestens) eine weitere Komponente zu generieren, die unsere Implementierung enthält. Die automatisch erstellte AppComponent kann dann zu Demo- oder Test-Zwecken während der Entwicklung verwendet werden. Aber Achtung! Für den späteren Build darf die Klasse AppComponent nicht im bootstrap-Array unseres Modules registriert sein, da es ansonsten zu Laufzeitfehlern kommt, da die app-root-Komponente natürlich nicht Teil unserer WebComponent sein wird. Zu empfehlen ist es also eher die bootstrap-Deklaration im Modul zu entfernen und eine separate Demo-Anwendung zum "Spielen" während der Entwicklung zu nutzen. Speziell dafür bietet es sich demnach an, neben dem Export unserer Komponente auch unser Modul fachlich passend umzubenennen:



Bei der Implementierung unserer Komponenten sind nun erst einmal kaum Grenzen gesetzt. Wir können wie gewohnt mit @Input- und @Output-Decoratorn arbeiten und auch der Einsatz von Routing, Services oder weiteren Komponenten ist möglich.
Bis hierhin haben wir noch keinen Finger in Sachen WebComponent krumm gemacht, sondern lediglich einige Angular-Komponenten generiert. Diese lassen sich natürlich nun auf gewohnte Art und Weise auch verwenden. So könnten andere Anwendungen innerhalb unseres Workspaces die erstellten Komponenten ganz einfach durch den Import des entsprechenden Modules importieren und dann verwenden:

```
demo.component.html
<ged-hello-world name="Herr Mathmann"
                 (message-click)="sayHello($event)">
</ged-hello-world>
```

Seine Komponenten nicht nur in ein eigenes Modul, sondern in eine eigene Anwendung oder Bibliothek auszulagern, empfiehlt sich eigentlich immer. Ob nun für jede Komponente ein eigenes Projekt ausgeprägt wird oder ein Projekt für mehrere unserer wiederverwendbaren Komponenten definiert wird, ist aber Geschmackssache und hängt natürlich auch von der Projektgröße ab. Bei der Verwendung innerhalb eines Workspaces wie oben zu sehen, haben wir keinerlei Nachteile bei der Entwicklung. Neue Anwendungen können nun aber problemlos unsere Utils- oder Komponenten-Projekte importieren und ebenfalls verwenden oder wir können diese doch als eigenständige Abhängigkeit in ein NPM-Repository bringen. Übrigens: Das Vorgehen, alles in einem Workspace oder besser gesagt in einem Git-Repository abzulegen, wird gemeinhin als Monorepo Pattern bezeichnet und auch z. B. von Google bei Angular selbst praktiziert.

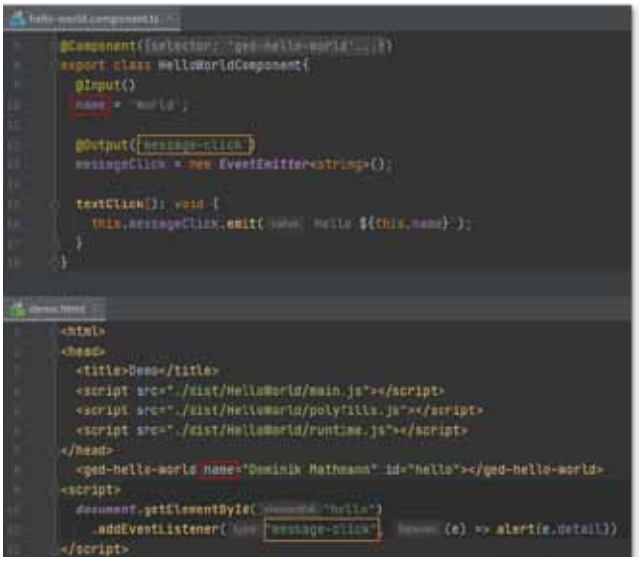
Hin zur WebComponent

Jetzt aber: Angular bietet uns seit der Version 6 ein optionales Paket, das uns in die Lage, versetzt unsere Anwendung als WebComponent auszuprägen. @angular/elements. Dieses fügen wir über einen entsprechenden Angular-CLI Befehl (jedem unserer Komponenten) hinzu:

```
ng add @angular/elements
```

Damit erhalten wir Zugriff auf eine statische Methode, die wir nun verwenden werden, um unsere Komponente zu registrieren. Dazu verwenden wir eine Lifecycle-Methode von Angular, die bei der Initialisierung unserer Anwendung ausgeführt wird ngDoBoostrap:

Im Grunde sollte es das bereits gewesen sein. Ein ng build HelloWorld --prod --output-hashing none liefert uns die entsprechenden JavaScript-Dateien (als Produktions-Build, ohne Hashcode im Dateinamen), die wir ganz klassisch in jegliche HTML-Datei einbinden können. Uns steht dabei natürlich auch die Möglichkeit zur Verfügung, diese Dateien mittels npm Tools zusammenzuführen, um nur eine einzelne Datei bereitstellen zu müssen, z. B. mittels jscat. Unsere Input-Attribute können wir nun mit den Bordmitteln unseres Ziel-Frameworks oder ganz klassisch mit HTML/JavaScript zuweisen. Auch die EventEmitter, die in unserer Komponente verwendet wurden, gliedern sich in den HTML-Standard ein. Dabei liefern sie ein Event vom Typ CustomEvent, welches unter anderem mit dem Attribut detail versehen ist, über das wir Zugriff auf das Objekt haben, welches durch den EventEmitter zurückgeliefert wird.



Die Ecken und Kanten

Soweit eigentlich alles gut. Im Detail stolpern wir allerdings über einige Punkte, die hier nicht unter den Teppich gekehrt werden sollen. Bei der Implementierung ist z. B. zu beachten, dass unsere Input-/Output-Decorator für die Verwendung innerhalb der WebComponent anstatt in CamelCase (messageClick) mit Bindestrichen verwendet werden müssen (message-click). Außerdem können wir in unseren Komponenten keine Content Projection verwenden, um mittels <ng-content> beliebige Inhalte an bestimmten Stellen innerhalb unseres Komponenten-Templates anzuzeigen. Stattdessen müssen wir dazu die <slot>-Mechanik verwenden, die Teil der Web Component-Spezifikation ist.
Ein ganz anderes Thema betrifft noch einmal den Build-Prozess: Webpack (welches für einen Teil des Standard Build Prozesses verantwortlich ist) nutzt eine globale Variable beim Bau der Projekte, was dazu führt, dass in der Form wie oben zu sehen nur eine einzige unserer Web Component auf der Seite per Script-Tag importiert werden kann. Diesem Problem und weiterer Optimierungen für die Erstellung der Komponenten hat sich ein OpenSource-Projekt angenommen, welches das NG-Tool erweitert: ngx-build-plus. Dieses fügen wir unseren Projekten hinzu und können dann über einen zusätzlichen Kommandozeilen-Parameter auch das Zusammenführen unserer Dateien erreichen. Damit reicht es auch, neben polyfills.js (einmalig) eine Datei pro Komponente zu importieren:

```
ng add ngx-build-plus
ng build HelloWorld --prod --output-hashing none --single-bundle true
```

Dieser Weg mit Hilfe von ngx-build-plus ist durchaus praktikabel, fühlt sich jedoch, insbesondere mit dem Blick auf die Issue-Liste, wie ein wackliger Workaround an. Darüber hinaus sollte uns ein entscheidender Punkt auch klar sein: Unsere Web Components müssen die Laufzeitbibliothek von Angular mit einbinden. Auch wenn dank neuem Ivy-Renderer die Bundle-Größen beeindruckend klein werden, (unsere HelloWorld-Komponente: ~100kb) schlägt eine pure JavaScript-Entwicklung uns natürlich in Sachen Dateigröße um Längen, insbesondere wenn wir weitere Module verwenden wie HttpClient oder Angular-Material. Das macht sich dann besonders bemerkbar, wenn wir viele kleine einzelne Komponenten verwenden, da wir die Angular-Runtime natürlich für jede integrierte Komponente duplizieren. Dem kann man entgegenwirken: Eine Möglichkeit besteht darin, die erweiterten Funktionen von ngx-build-plus zu verwenden und durch eine zusätzliche Webpack-Konfiguration die

Angular-relevanten Sourcen in eine separate JavaScript-Datei zu kompilieren, sodass wir diese nur einmal importieren müssen. Eine andere Variante besteht darin, die eigenen Komponenten oder Teile in einem Bundle zusammenzuführen, indem wir ein separates Projekt anlegen, das keine eigenen Komponenten definiert, sondern lediglich die bestehenden Module importiert und dessen Registrierungen durchführt (die zuvor in eine statische Funktion ausgelagert wurden). Die ngBoostrap-Methode dieses Bundle-Modules könnte dann so aussehen:

Fazit

Was bleibt am Ende? WebComponents sind eine tolle Technik, um Komponenten unabhängig von Frameworks wiederverwendbar zu machen. Auch mit Angular lassen sich solche Komponenten bauen. Wünschenswert ist hier sicherlich noch die Erweiterung des Standard Buildprozesses der Angular-CLI, um auf zusätzliche Tools/Erweiterungen verzichten zu können, aber die ersten Schritte sind gemacht. Sicherlich wird kaum ein Team seine Angular-Komponenten für Angular-Verwendung nun vollständig auf WebComponents umbauen, aber durch eine intelligente Strukturierung unserer Anwendung können wir uns die Möglichkeit, einzelne unserer Komponenten oder sogar Anwendungsteile als WebComponent auszuprägen, zumindest offen halten und bei Bedarf ganz einfach implementieren.

Links

- Demo-Sourcen
  https://github.com/GEDOPLAN/angular-webcomponents
- Doku Web Components
  https://developer.mozilla.org/de/docs/Web/Web_Components
- ngx-build-plus
  https://github.com/manfredsteyer/ngx-build-plus

Dominik Mathmann[dominik.mathmann@gedoplan.de]
Berater, Entwickler und Trainer bei der GEDOPLAN GmbH. Auf Basis seiner langjährigen Erfahrung in der Implementierung von Java-EE-Anwendungen leitet er Seminare, hält Vorträge und unterstützt Kunden bei der Konzeption und Realisierung von Webanwendungen vom Backend bis zum Frontend.

News from the Blog

GEDOPLAN betreibt seit Jahren einen Blog, in dem wir über Neuigkeiten und Tipps rund um Java EE berichten. Regelmäßig informieren unsere Mitarbeiter von ihren Erfahrungen in aktuellen Projekten, geben Tipps, wie kleine Probleme mit Java EE gelöst werden können oder regen Diskussionen um unterschiedliche Lösungsmöglichkeiten an.
Einige der wichtigsten Beiträge möchten wir Ihnen in unserer Rubrik „News from the Blog" in unregelmäßigen Abständen kurz vorstellen. Vielleicht wecken wir Ihr Interesse und Sie besuchen unseren Blog unter javaeeblog.wordpress.com, um sich zu informieren und den gesamten Beitrag zu lesen. Oder, was uns noch mehr freuen würde, um sich an den Diskussionen zu beteiligen.

Angular E2E mit json-server

Dominik Mathmann, GEDOPLAN GmbH, stellt eine praktikable Möglichkeit vor, eine Anwendung "von vorne bis hinten" durchzutesten: einen json-server.

Angular mit Zeitstempel und Version

Um sicherzugehen, welche Version der Anwendung man gerade testet, können Zeitstempel und aktuelle Version hilfreiche Informationen sein. Die sind aber in der Anwendung nicht so einfach verfügbar. Dieser Eintrag zeigt, wie man sie sichtbar macht.

Richfaces und JSF 2.3.X

Richfaces spielen zwar in der aktuellen Softwareentwicklung keine große Rolle mehr, finden sich aber noch in vielen Anwendungen. Und sie bereiten Probleme, wenn andere Komponenten aktualisiert werden. Dieser Artikel zeigt, wie diese Probleme zum Teil behoben werden können.

Keycloak + Protractor E2E Test

Keycloak ist eine charmante Authentifizierungs-Lösung, die sich dank keycloak-angular relativ problemlos in der eigenen Anwendung verankern lässt. In Kombination mit E2E-Tests gibt es dann aber das eine oder andere Problem. Eine Lösung zeigt dieser Beitrag.

Spring Boot Swagger Keycloak

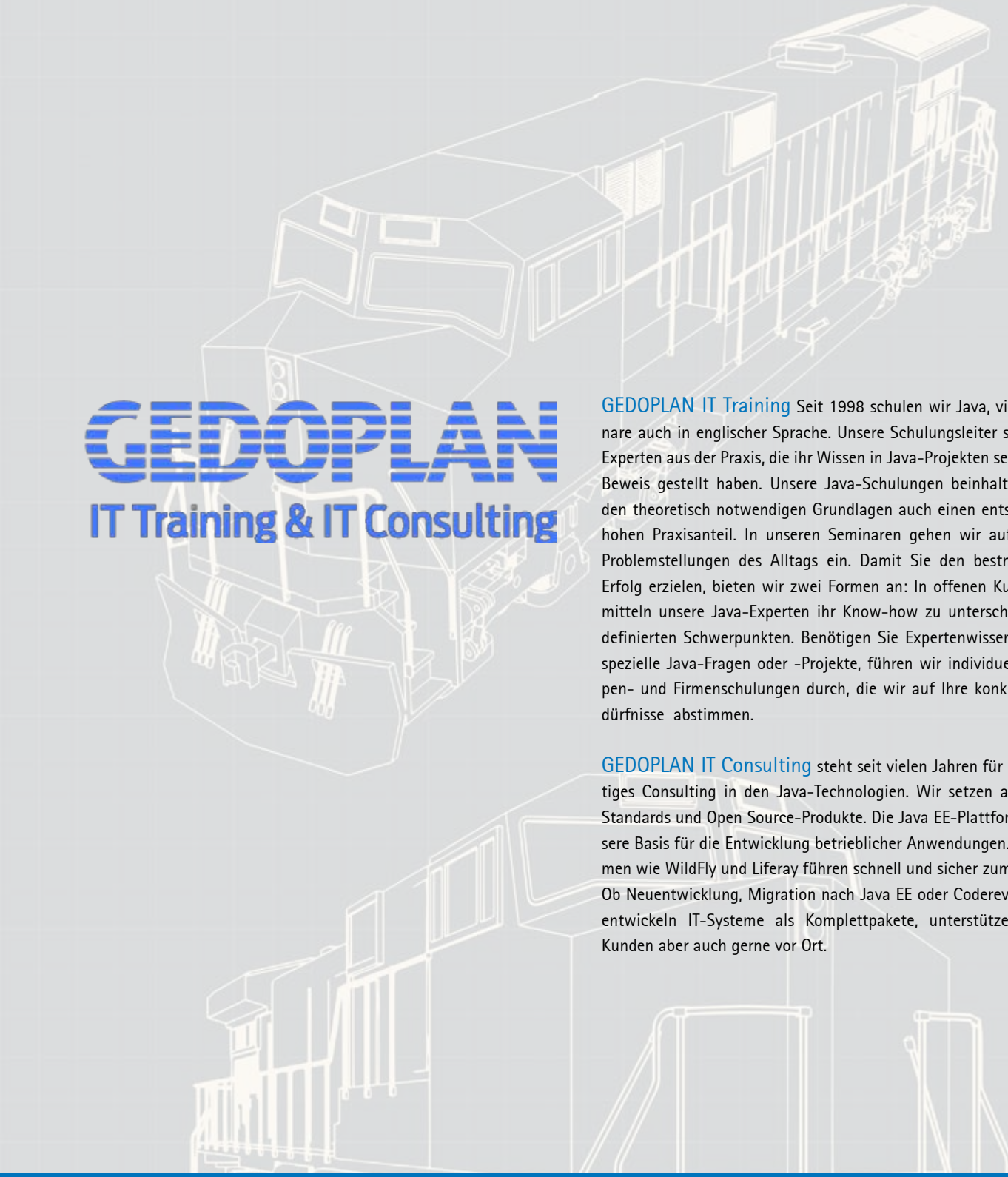
Um zusätzliche Information für unsere Schnittstellen bereitzustellen und wenn Rest-Aufrufe eine Authentifizierung benötigen, bedarf es zusätzlicher Konfiguration, um z. B. die swagger-ui nutzen zu können. Für die OAuth2 Variante, z. B. mit Keycloak, wird dafür eine ganz einfache der Konfigurationen gezeigt.

Angular-CLI Proxy + JEE

Sollen Anwendungen auf unterschiedlichen Serveren laufen, müssen die CORS.Richtlinien beachtet werden. Mit einem Angular-Dev-Server lässt sich das ganz einfach lösen.

GEDOPLAN Java EE Blog
Hier finden sich Neuigkeiten, Ideen und Kommentare rund um Java EE

javaeeblog.wordpress.com



GEDOPLAN
IT Training & IT Consulting

GEDOPLAN IT Training Seit 1998 schulen wir Java, viele Seminare auch in englischer Sprache. Unsere Schulungsleiter sind Java-Experten aus der Praxis, die ihr Wissen in Java-Projekten selbst unter Beweis gestellt haben. Unsere Java-Schulungen beinhalten neben den theoretisch notwendigen Grundlagen auch einen entsprechend hohen Praxisanteil. In unseren Seminaren gehen wir auf aktuelle Problemstellungen des Alltags ein. Damit Sie den bestmöglichen Erfolg erzielen, bieten wir zwei Formen an: In offenen Kursen vermitteln unsere Java-Experten ihr Know-how zu unterschiedlichen, definierten Schwerpunkten. Benötigen Sie Expertenwissen für sehr spezielle Java-Fragen oder -Projekte, führen wir individuelle Gruppen- und Firmenschulungen durch, die wir auf Ihre konkreten Bedürfnisse abstimmen.

GEDOPLAN IT Consulting steht seit vielen Jahren für hochwertiges Consulting in den Java-Technologien. Wir setzen auf offene Standards und Open Source-Produkte. Die Java EE-Plattform ist unsere Basis für die Entwicklung betrieblicher Anwendungen. Plattformen wie WildFly und Liferay führen schnell und sicher zum Ziel.
Ob Neuentwicklung, Migration nach Java EE oder Codereviews: Wir entwickeln IT-Systeme als Komplettpakete, unterstützen unsere Kunden aber auch gerne vor Ort.

GEDOPLAN
Unternehmensberatung und
EDV-Organisation GmbH
Stieghorster Straße 60
33605 Bielefeld
Fon: + 49 521 / 2 08 89 10
Fax: +49 521 / 2 08 89 45
info@gedoplan.de

Geschäftsstelle Berlin:
GEDOPLAN GmbH
UPPER WEST
Kantstraße 164
10623 Berlin

Postadresse Berlin:
GEDOPLAN GmbH
Kurfürstendamm 11
10719 Berlin

+49 30 / 2089 82 630
it-training@gedoplan.de.

Herausgeber: GEDOPLAN Unternehmensberatung und EDV-Organisation GmbH | Stieghorster Straße 60 | 33605 Bielefeld
Verantwortlich: Dipl.-Inform. Dirk Weil | Redaktionsleitung: Ulrich Hake | Fon: 0521 2088910 | E-Mail: info@gedoplan.de
Produktion: networker Medienfabrik GmbH | Nachdruck, auch auszugsweise, nur nach Rücksprache mit der Redaktion.

www.gedoplan.de