Smarte Planung

Webbasierte Schichtplan-Optimierung mit Smart GWT und OptaPlanner

Jan-Philipp Friedenstab, David Jorch, Christian Treptau

In einem großen Callcenter arbeiten mehrere tausend Agenten. Für diese müssen optimal aufeinander abgestimmte Schichtpläne erstellt werden. Workforce-Manager geben die Rahmenbedingungen der Planung vor und Callcenter-Agenten dürfen ihre Einsatzwünsche nennen. Es wird eine Architektur für eine Webanwendung vorgestellt, die die Arbeit der Workforce-Manager erleichtert und das Webframework Smart GWT unter anderem mit einer Optimization Engine kombiniert, um bessere Schichtpläne zu erstellen.

Gute Schichtpläne unter komplexen Bedingungen

In einem großen Callcenter gehen jeden Tag Tausende Anrufe ein. Es ist erfolgsentscheidend, diese Anrufe zeitgerecht bearbeiten zu können. Workforce-Manager stehen regelmäßig vor der Herausforderung, mehrere tausend Agenten auf verschiedene Schichtzeiten zu verteilen. Das Ziel ist eine effiziente Planung der Personalkapazität, die sich der vorhergesagten Menge an Anrufen angleicht.

Bei der Planung sind Rahmenbedingungen zu berücksichtigen wie zum Beispiel regionale und nationale Feiertage. Es bestehen typischerweise komplexe Regelwerke. Diese enthalten gesetzliche Vorgaben, spezielle Arbeitszeitenregelungen der Arbeitgeber und vertragliche Vereinbarungen mit den Arbeitnehmern.

Eine speziell auf die Problemstellung zugeschnittene Webanwendung führt zu erheblicher Arbeitserleichterung für das Workforce-Management. Besonders wichtig ist es, dass die Daten im Frontend einfach erfasst werden können.

Häufig legen Workforce-Manager zusätzlich Wert auf die Änderbarkeit des hinterlegten Regelwerks. Daher wird JBoss Drools Expert in die Architektur integriert, um Regeln zu pflegen und deren Einhaltung bei der Planung sicherzustellen.

Darüber hinaus sollen oft die individuellen Einsatzwünsche der Agenten berücksichtigt werden. Die Agenten erfassen ihre Wünsche idealerweise selbst, indem sie die Webanwendung als Selfservice nutzen. Der komplizierte Umweg an Workforce-Manager über eine Kette von E-Mails entfällt.

Neben der Effizienzsteigerung steht die Verbesserung der Lösungsqualität des Schichtplanungsproblems im Mittelpunkt. Eine moderne Ziellösung unterstützt die Personaleinsatzplanung durch eine automatische Optimierung, die über die Anwendung bedient werden kann. Die vorgestellte Softwarearchitektur bindet das Webfrontend an *JBoss OptaPlanner* an, um im Hintergrund komplexe Optimierungsverfahren auszuführen.

Schließlich werden regelmäßig Reports gewünscht, die es erlauben, komplizierte Planungsprobleme zu überschauen und Entscheidungen zu treffen. Dazu wird *Eclipse BIRT* in die Architektur eingebunden. Verschiedene Planungsszenarien werden grafisch aufbereitet und mit Hilfe üblicher Kennzahlen des Workforce-Managements vergleichbar gemacht.

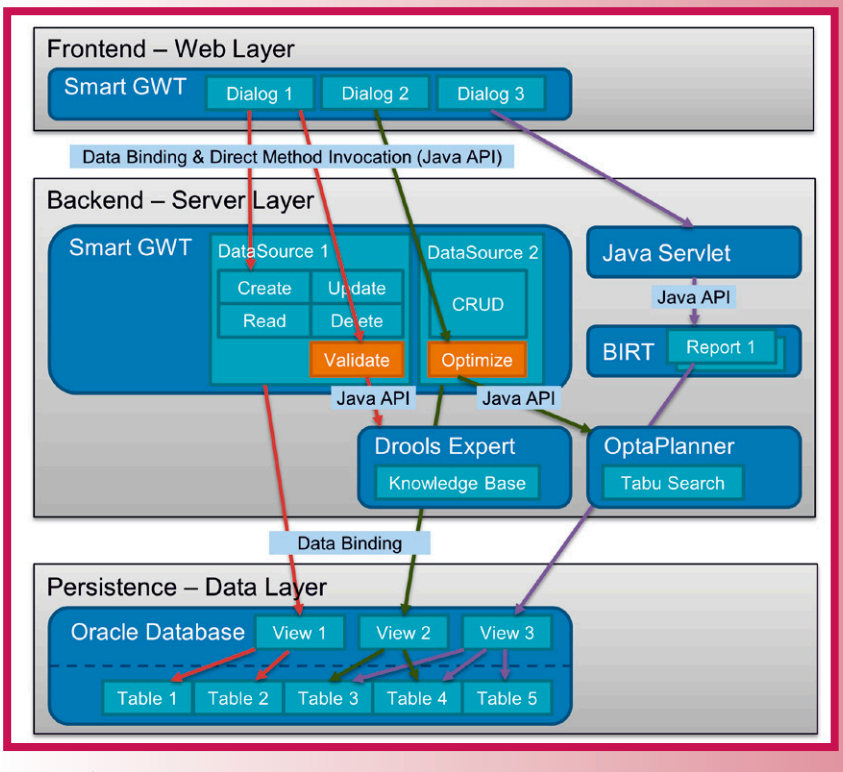

Abb. 1: Softwarearchitektur einer Webanwendung mit Planungskomponenten im Backend

Softwarearchitektur

Die Architektur einer modernen Webanwendung für das Workforce-Management kombiniert Komponenten für das Frontend mit weiteren Komponenten im Backend. Alle Komponenten stehen als Open Source frei zur Verfügung. Ein beispielhafter Architekturaufbau ist in Abbildung 1 dargestellt. Die einzelnen Komponenten werden in den folgenden Abschnitten beschrieben.

Isomorphic Smart GWT

Isomorphic Smart GWT (in der Version 4.0) baut auf dem *Google Web Toolkit (GWT)* auf. Die Programmierung selbst erfolgt in Java. Der Code wird automatisch in Websprachen übersetzt.

Smart GWT stellt diverse Funktionen zur Verfügung, um *Rich Internet Applications* zu entwickeln (vgl. [Sma]). Die Grundedition kann um kostenpflichtige Komponenten erweitert werden. Im Folgenden werden die wichtigsten Funktionen beschrieben, die in der Webanwendung im Einsatz sind.

User-Interface(UI)-Komponenten

Es werden fertige Bausteine genutzt, um Anwendern mit Hilfe von JavaScript, HTML5 und Ajax interaktive Bedienelemente zur Verfügung zu stellen. Dazu gehören zum Beispiel *ListGrids*, die sortier- und editierbare Tabellen und Listen mit individuell einstellbaren Filtern darstellen. Bei Bedarf kann ein GUI-Editor für den Oberflächenentwurf eingesetzt werden.

Data Binding

Die relationale Datenbank wird über sogenannte *DataSources* an das Backend angebunden. Eine DataSource beschreibt ein Datenbankobjekt im XML-Format (*.ds.xml-Datei). Die Beschreibung enthält die einzelnen Spalten und deren Wertetypen. Sie kann auch Validierungsregeln vorgeben. Es ist möglich, die DataSource-XML-Datei automatisch für Datenbankobjekte generieren zu lassen.

Eine UI-Komponente kann mit einer DataSource verknüpft werden (s. Listing 1). So lassen sich ohne weiteren Programmieraufwand Standard-Operationen wie Create, Read, Update und Delete (CRUD) auf den Daten durchführen. Dazu gehören auch das Sortieren, Filtern, Paging und Caching von Daten. Eine serverseitige Standard-Validierung der Eingaben im Frontend steht ebenfalls zur Verfügung. Diese passt sich automatisch den Wertetypen an, die in der DataSource hinterlegt sind.

Direct Method Invocation (DMI)

Jede Ajax-Anfrage aus dem Browser führt über ein Java-API zu einem direkten Methodenaufruf einer Java-Klasse. Dazu werden die Daten in der Anfrage, die ein Browser an das zentrale Smart GWT-Servlet im Backend sendet, in ein Java-Objekt übersetzt (DSRequest) und an die entsprechende Methode weitergeleitet. Die Rückgabewerte der Methode werden wiederum in ein Java-Objekt verpackt (DSResponse) und an den Browser zurückgegeben. Die Client-Server-Kommunikation wird durch Smart GWT zur Verfügung gestellt. Im Hintergrund werden bekannte Formate (JSON/XML) genutzt. Es entfällt die manuelle Programmierung eines Request-Response-Handlings und die Übersetzung in Nachrichtenaustauschformate.

```
public class AgentenTab extends Tab {
  private ListGrid agentenListe;

  private void init() {
    agentenListe = new ListGrid();

    // Data Binding (Verknuepfung von UI-Komponente und DataSource)
    agentenListe.setDataSource(DataSource.get(Consts.DS_AGENTEN));
    agentenListe.setSort(new SortSpecifier[] {
      new SortSpecifier(Consts.F_AGENT_NAME,
        SortDirection.ASCENDING)});
    // Filterfelder am Kopf der Liste anzeigen
    agentenListe.setShowFilterEditor(true);

    // Agentenliste kann vom Benutzer editiert werden
    agentenListe.setCanEdit(true);
  }
  private void onSave() {
    // Standard-CRUD-Operation von Smart GWT
    agentenListe.saveAllEdits();
  }
}
```

Listing 1: Data Binding zur Nutzung von CRUD-Operationen in einer Frontend-Java-Klasse (Ausschnitt)

Es besteht die Möglichkeit, eigene DMI-Java-Klassen in der DataSource-Beschreibung über das XML-Tag `<serverObject>` zu hinterlegen. So können die Standard-CRUD-Operationen nach eigenen Wünschen angepasst werden. Die DSRequests und DS-Reponses werden dazu letztendlich über zusätzliche Java-Logik modifiziert. Darüber hinaus können dem Frontend weitere Methoden für Datenoperationen zur Verfügung gestellt werden.

JBoss Drools Expert

JBoss Drools Expert (in der Version 5.5) ist an das Webfrontend angebunden (vgl. [Bal13] und [Dro-a]). Die serverseitige Komponente von Smart GWT wird dazu mit dem *Knowledge Java API* von Drools Expert verknüpft. Die *Rules Engine* stellt während der Erstellung von Schichtplänen sicher, dass gesetzliche Vorgaben und spezielle Bedingungen im Arbeitgeber-/Arbeitnehmer-Verhältnis eingehalten werden.

Matching von Regeln und Daten

Das Herzstück von Drools ist eine *Inference Engine*, die einen *Pattern Matcher* enthält (s. Abb. 2). Der Pattern Matcher prüft, ob sich die vorliegenden Daten (*Facts*) mit den Bedingungen bestimmter Regeln decken, sodass diese ausgeführt werden können. Schließlich erstellt der Pattern Matcher eine *Agenda*, die die Reihenfolge der Ausführung mehrerer Regeln vorgibt und Konflikte auflöst.

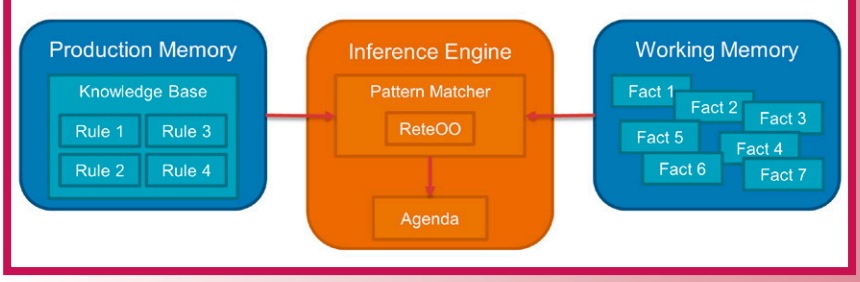

Abb. 2: Aufbau und Funktionsweise der Drools Expert Rules Engine (Quelle: vgl. [Dro-b], S. 111)

Definition von Business Rules

Die Definition des fachlichen Regelwerks erfolgt in der *Drools Rule Language (DRL)*. Mehrere Regeln werden in *.drl-Dateien zusammengefasst. Die Regeln innerhalb einer *.drl-Datei werden der Rules Engine als *Knowledge Base* vor dem Pattern Matching bereitgestellt. So lassen sich Regeln fachlich in mehrere Knowledge Bases trennen, auf die bei Bedarf zurückgegriffen wird.

Die Trennung der Regeldefinitionen vom Java-Code erleichtert die Anpassung der Regeln an veränderte Rahmenbedingungen. Zusätzlich schafft die Kapselung das Potenzial, bestimmten Anwendern selbst die Möglichkeit zu geben, das Regelwerk über das Webfrontend anzupassen.

Einsatz der Rules Engine

Die Rules Engine wird durch das Webfrontend zur Validierung von Eingaben der Anwender aufgerufen. Die entsprechenden Regeldefinitionen folgen einem einheitlichen Schema (s. Listing 2).

```
rule "<short_description>" // Bezeichnung der Regel
  when <conditions> // Bedingungen für die Regel-Ausführung
  then <actions> // Konsequenzen der Regel-Ausführung
end
```

Listing 2: Regeldefinition mit der Drools Rule Language

Dabei stellen die `<conditions>` eingegebene Datenkonstellationen dar, die fachlich invalide sind. Die `<actions>` sind Einträge in eine globale Liste von Fehlermeldungen. Wird eine Regel ausgeführt, weil die `<conditions>` erfüllt sind, ist diese Liste nicht mehr leer. Nachdem die Rules Engine eingesetzt worden ist, reagiert dann das Webfrontend mit der Ausgabe entsprechender Hinweise für den Anwender.

Um die Rules Engine als Validierer anzubinden, wird von der oben beschriebenen Möglichkeit Gebrauch gemacht, dem Webfrontend neben den CRUD-Operationen eine spezielle Methode zur Datenvalidierung zur Verfügung zu stellen. Dazu wird eine eigene DMI-Java-Klasse in der Beschreibung der DataSource hinterlegt, die mit einem Eingabedialog im Webfrontend verknüpft ist (s. Abb. 1). Die DMI-Klasse enthält eine `validate()`-Methode, die den Aufruf der Rules Engine über das Java-API implementiert.

Auch JBoss OptaPlanner nutzt die Rules Engine, um die Anwendbarkeit von automatisch erstellten Schichtplänen sicherzustellen.

JBoss OptaPlanner

JBoss OptaPlanner (oder *Drools Planner*; in der Version 5.5) ist die eingesetzte, serverseitige Komponente zur Lösung des mathematischen Optimierungsproblems der Schichtplanung (vgl. [Opta]).

Die Zielkriterien und Nebenbedingungen der Optimierung werden für den konkreten Anwendungsfall in der Drools Rule Language formuliert. Zudem muss die Optimization Engine mit den notwendigen Daten versorgt werden. Fachliche Objekte werden in POJOs beschrieben. Schließlich werden aus den out-of-the-box angebotenen Verfahren zur heuristischen Lösungssuche die geeigneten ausgewählt. Die Lösungssuche selbst übernimmt OptaPlanner.

Optimierung – Zielkriterien und Freiheitsgrade

Unter den Nebenbedingungen der im Regelwerk abgebildeten Arbeitszeitenregelungen erstellt OptaPlanner gut aufeinander abgestimmte Schichtpläne für die Agenten. Im Ergebnis werden diese zusammen so verplant, dass sich die eingesetzte Personalkapazität der erwarteten Menge an Anrufen angleicht (s. Abb. 3). Dazu werden Forecast-Daten berücksichtigt.

Die Lösungsqualität des Optimierungsproblems bemisst sich aus definierten Zielkriterien. Zu den wichtigsten Kriterien zählen üblicherweise:

- *Geringe Überdeckung:* Die geplante Personalkapazität in den verschiedenen Schichten soll nicht viel größer sein als der voraussichtliche Bedarf.
- *Geringe Unterdeckung:* Die geplante Personalkapazität in den verschiedenen Schichten soll nicht viel kleiner sein als der voraussichtliche Bedarf.

Innerhalb der Zielfunktion wird die Summe der absoluten Abweichungen *[geplante Personalkapazität minus voraussichtlicher Bedarf]* über die einzelnen Schichten gebildet und minimiert. Um große Über- und Unterdeckungen stärker zu bestrafen, werden die Abweichungen häufig quadriert.

Die Optimierung nutzt vorgegebene Freiheitsgrade, um eine möglichst gute Lösung zu finden. Bei der Schichtplanung ergeben sich als Freiheitsgrade typischerweise:

- *Verteilung auf Wochentage:* Die Agenten können abhängig von den vertraglich vereinbarten Wochenarbeitstagen und festgelegten Rahmendaten der Schichtplanung auf verschiedene Wochentage verteilt werden. Dabei werden ihre Einsatzwünsche berücksichtigt.
- *Verteilung auf Schichtzeiten:* Innerhalb der Arbeitstage können die Agenten auf verschiedene Schichtzeiten verteilt werden.

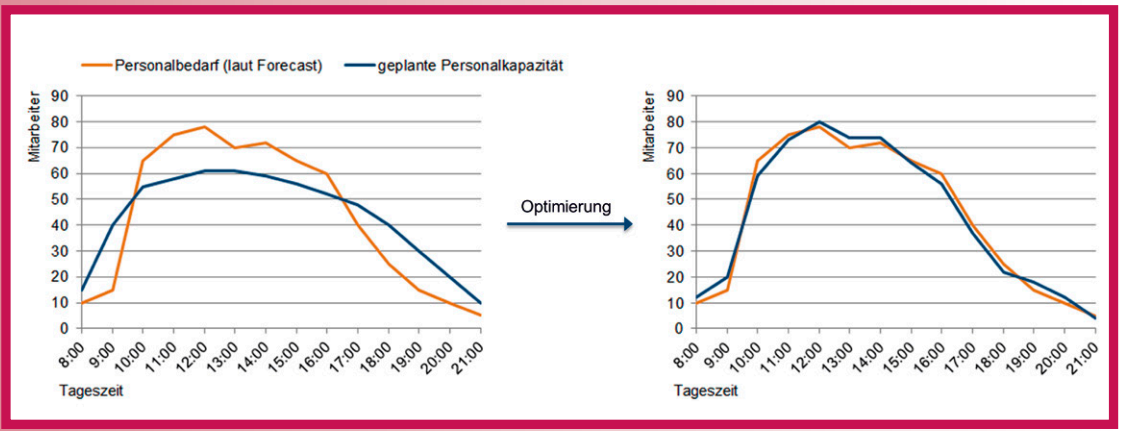

Abb. 3: Optimierung der geplanten Personalkapazität anhand des Bedarfs

Optimierung – Heuristische Lösungssuche

Häufig ist es effizienter, eine gute Lösung für ein Optimierungsproblem in angemessener Zeit zu finden als sehr lange nach der optimalen Lösung zu suchen. In solchen Fällen werden Algorithmen zur heuristischen Lösungssuche angewandt.

Der Ablauf der Optimierung lässt sich in zwei Phasen einteilen:

- initiale Lösungssuche und
- Suche nach einer besseren Lösung.

Für die initiale Lösungssuche bietet OptaPlanner verschiedene Heuristiken an, deren Eignung jeweils von dem vorliegenden Optimierungsproblem abhängt. In der XML-Konfiguration der Optimization Engine wird der passende Algorithmus im Tag `<constructionHeuristic>` festgelegt (s. Listing 3).

```
<solver>
  <termination>
    <maximumMinutesSpend>5</maximumMinutesSpend>
  </termination>
  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT</constructionHeuristicType>
  </constructionHeuristic>
  <localSearch>
    <termination>
      <maximumUnimprovedStepCount>500</maximumUnimprovedStepCount>
    </termination>
    <acceptor>
      <solutionTabuSize>1000</solutionTabuSize>
    </acceptor>
  </localSearch>
</solver>
```

Listing 3: XML-Konfiguration der Optimization Engine OptaPlanner (Ausschnitt)

Für die Suche nach einer besseren Lösung ausgehend von einer aktuell gefundenen Lösung stehen ebenfalls mehrere Heuristiken zur Verfügung. Typischerweise evaluiert ein Suchalgorithmus eine Anzahl an möglichen Veränderungen der aktuellen Lösung, die wieder zu gültigen Lösungen führen. Man spricht hier von einer Nachbarschaft an Lösungen. Schließlich wird diejenige Veränderung ausgeführt, die zur besten Lösung in der Nachbarschaft führt. Das iterative Verfahren endet, sobald eine Terminierungsbedingung eintritt.

Im XML-Tag `<localSearch>` wird der Algorithmus festgelegt. Für das vorliegende Optimierungsproblem wird eine Tabu-Suche angewandt. Dieser Algorithmus vermeidet, dass die Lösungssuche in einem lokalen Optimum verbleibt. Dazu wird ein Tabu-Kriterium für die Wahl der jeweils nächsten Lösung definiert. Im XML-Tag `<solutionTabuSize>` ist die Anzahl an Lösungen festgelegt, die in vorherigen Suchschritten gewählt wurden und festgehalten werden. Diese Lösungen sind für den jeweils nächsten Schritt tabu, das heißt, es wird keine Veränderung akzeptiert, die zurück zu einer solchen Lösung führt.

Workforce-Manager können die automatische Erstellung der Schichtplanung über das Webfrontend starten und deren Fortschritt verfolgen. Dazu kommuniziert Smart GWT im Backend über das *Planner Core Java API* mit OptaPlanner. Es lassen sich unterschiedliche Bedingungen festlegen, unter denen die heuristische Lösungssuche terminiert. In den XML-Tags `<maximumUnimprovedStepCount>` und `<maximumMinutesSpend>` werden zwei Bedingungen kombiniert: Die Suche endet frühzeitig, falls nach einer festgelegten Anzahl an Suchschritten keine bessere als die aktuell gewählte Lösung gefunden wird, spätestens jedoch nach einer festgelegten Maximaldauer.

Eclipse BIRT

Eclipse BIRT (in der Version 4.3.1) wird genutzt, um verschiedene Szenarien der Schichtplanung grafisch aufzubereiten. BIRT ist über ein Java-Servlet an das Webfrontend angebunden (vgl. [Ecl]). Das Servlet kommuniziert mit BIRT über das *Report Engine Java API*.

Fazit

Die vorgestellte Kombination der beschriebenen Frameworks hat sich bewährt. Es werden Komponenten integriert, die in einer sinnvollen Aufgabentrennung Funktionen zur Gesamtarchitektur beitragen. Die ausgereiften Frameworks überzeugen in ihrer jeweiligen Domäne.

Mit Smart GWT werden grafisch aufwendige Business-Anwendungen effizient entwickelt. Insbesondere die Vielseitigkeit der UI-Komponenten in Verbindung mit dem Data-Binding-Konzept hat überzeugt. Eine einfache Anbindung von weiteren Frameworks ist möglich.

Optimierungsprobleme lassen sich mit OptaPlanner und Drools Expert gut softwaretechnisch abbilden. In der Drools Rule Language werden die fachliche Zielfunktion und Nebenbedingungen kompakt formuliert. Durch die Verwendung von POJOs zur Beschreibung von fachlichen Objekten wird der Einstieg in OptaPlanner leicht gemacht.

Literatur und Links

[Bal13] M. Bali, Drools JBoss Rules 5.X Developer's Guide, Packt Publishing, 2013

[Dro-a] Drools, Business Rules Management System, `www.drools.org`

[Dro-b] The JBoss Drools team, Drools Expert User Guide Version 5.5.0 Final, `https://docs.jboss.org/drools/release/5.5.0.Final/drools-expert-docs/pdf/drools-expert-docs.pdf`

[Ecl] Eclipse Business Intelligence and Reporting Tools, `www.eclipse.org/birt/`

[Opta] Red Hat JBoss Middleware, OptaPlanner, `www.optaplanner.org`

[Sma] Isomorphic Software, Smart GWT Showcase, `www.smartclient.com/smartgwt/showcase/`

Jan-Philipp Friedenstab ist als Softwareentwickler bei der viadee Unternehmensberatung GmbH tätig. Sein Arbeitsschwerpunkt liegt im Java-Umfeld – hier war er in verschiedenen Kundenprojekten an der Konzeption und Realisierung von Webanwendungen und Job-basierten Backend-Systemen beteiligt.
E-Mail: jan-philipp.friedenstab@viadee.de

David Jorch ist als Unternehmensberater in Kundenprojekten tätig. Seine Projekterfahrung umfasst u. a. den Einsatz von Rules Engines und Optimization Engines in verschiedenen Branchen, z. B. in der Automobilindustrie und im Modehandel. Ein weiterer Schwerpunkt liegt im Einsatz von Data Mining.
E-Mail: david.jorch@viadee.de

Christian Treptau ist als Projektleiter für die viadee Unternehmensberatung GmbH tätig. In vielen Projekten hat er als Projektleiter oder Softwarearchitekt mitgewirkt. Sein Fokus liegt auf Prozessautomatisierung und Prozessoptimierung in komplexen Umfeldern.
E-Mail: christian.treptau@viadee.de