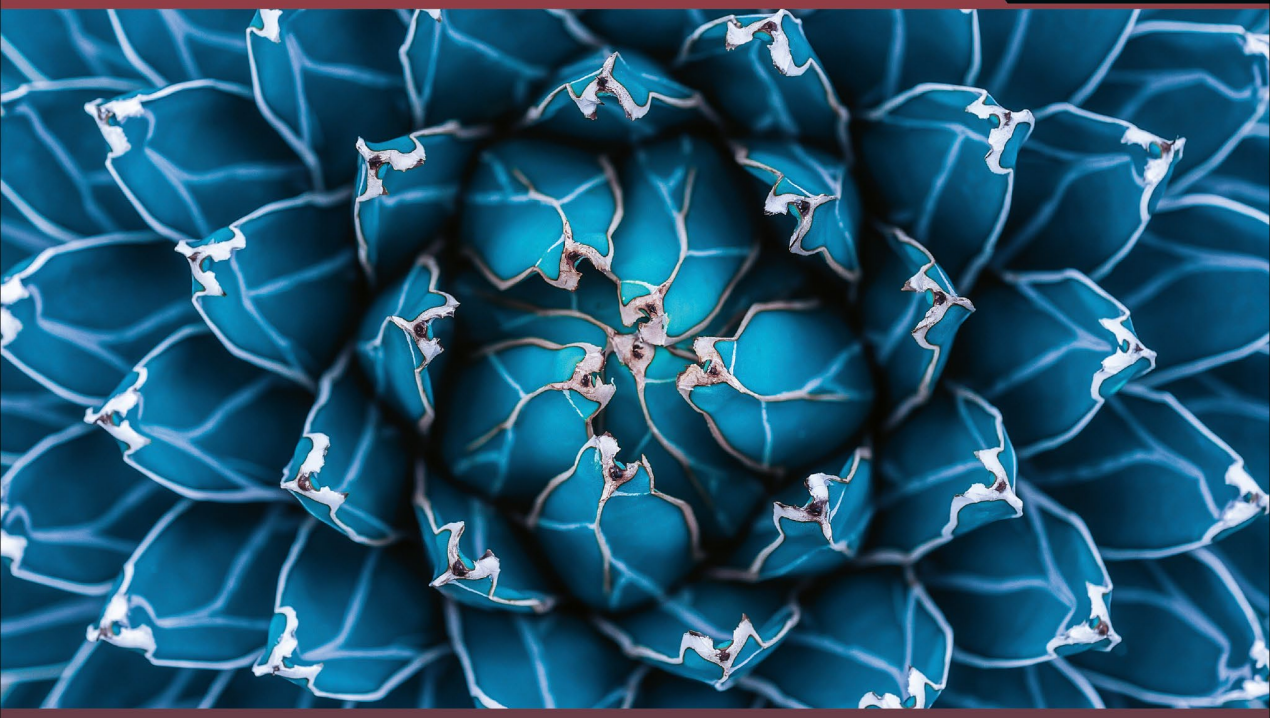
SAP PRESS



ABAP® RESTful Application Programming Model

Das umfassende Handbuch

2., erweiterte Auflage

- Anwendungsentwicklung für SAP S/4HANA und die Cloud
- Vom Datenmodell über die Behavior Definition bis zur Benutzeroberfläche
- Umgang mit CDS, Annotationen und SAP Fiori Elements

Lutz Baumbusch
Matthias Jäger
Michael Lensch

Rheinwerk Publishing

Kapitel 1

Einführung in das ABAP RESTful Application Programming Model

In diesem einleitenden Kapitel erhalten Sie einen Überblick über das ABAP RESTful Application Programming Model sowie einige Hintergrundinformationen. Sie erfahren etwas über die Entstehung des Programmiermodells, seine qualitativen Eigenschaften, die Architektur und grundlegende Elemente sowie über die technische Verfügbarkeit.

In diesem Kapitel stellen wir Ihnen das *ABAP RESTful Application Programming Model* vor. Sie gewinnen eine Vorstellung davon, um was es sich bei diesem Programmiermodell handelt und welche grundlegenden technischen Eigenschaften es mit sich bringt.

Aufbau des Kapitels

In Abschnitt 1.1 gehen wir auf den REST-Architekturstil und seine Prinzipien ein und erklären somit, was an dem Programmiermodell *RESTful* ist. Sie erfahren außerdem, welche bestehenden ABAP-Technologien sich im ABAP RESTful Application Programming Model wiederfinden und welche Einflüsse diese auf das Programmiermodell haben.

In Abschnitt 1.2 gehen wir auf die grundlegenden, architektonischen Konzepte des Programmiermodells ein, bevor wir in Abschnitt 1.3 zu einem Streifzug durch die relevanten Artefakte ansetzen und klären, welche Aufgabe diese jeweils haben. Außerdem werfen wir in Abschnitt 1.4 einen ersten Blick auf die Entwicklungsumgebung, in der Sie Anwendungen mit dem ABAP RESTful Application Programming Model entwickeln.

Aus der REST-Architektur und den Grundkonzepten des Programmiermodells ergeben sich die wesentlichen qualitativen Eigenschaften, die das Programmiermodell mit sich bringt. Diese behandeln wir in Abschnitt 1.5. Wir schließen dieses einleitende Kapitel, indem wir das ABAP RESTful Application Programming Model in Abschnitt 1.6 in den SAP-Produktkontext setzen, woraus sich seine technische Verfügbarkeit ergibt.

Abkürzung »RAP«

Häufig werden Sie die Abkürzung »RAP« für das ABAP RESTful Application Programming Model lesen. Auch wenn es sich nicht um eine offizielle Abkürzung handelt, hat sie sich in der Entwicklergemeinschaft doch eingebürgert.

1.1 Was ist das ABAP RESTful Application Programming Model?

In diesem Abschnitt klären wir zunächst, was ein Programmiermodell überhaupt ist. Vor diesem Hintergrund gehen wir auf die Grundlagen und wesentlichen Eigenschaften des ABAP RESTful Application Programming Models ein, betrachten den REST-Architekturstil, der als Basis für das Programmiermodell dient, und erläutern die Rolle des OData-Protokolls in diesem Zusammenhang. Anschließend zeigen wir die historische Entwicklung ABAP-basierter Programmiermodelle auf und erläutern das Zusammenspiel des ABAP RESTful Application Programming Models mit den wesentlichen technologischen Neuerungen und SAP S/4HANA.

1.1.1 Aufgaben des Programmiermodells

Entwicklung von Unternehmensanwendungen

Das ABAP RESTful Application Programming Model ist eine neue Art, Unternehmensanwendungen auf der ABAP-Plattform zu entwickeln. Es umfasst alle Schritte der Anwendungsentwicklung vom Datenmodell über Operationen wie das Anlegen, Lesen, Ändern oder Löschen von Geschäftsobjekten, die Geschäftslogik (wie Berechnungen oder Prüfungen) und das transaktionale Verhalten (wie Sperrverhalten, Nummernvergabe etc.) bis hin zur technischen Schnittstelle (typischerweise OData).

Aufgaben eines Programmiermodells

Unabhängig von der Technologie beschreibt ein *Programmiermodell*, wie man als Anwendungsentwicklerin oder -entwickler Anwendungen auf Basis einer Softwarearchitektur baut und testet und wie man eine Fehleranalyse betreibt. Dazu sieht ein Programmiermodell bestimmte Technologien, Konzepte, Artefakte (in ABAP in der Regel *Entwicklungsobjekte* genannt) und Werkzeuge vor. Ein definierter Entwicklungsfluss führt diese Bestandteile zusammen.

Bezogen auf die ABAP-Plattform muss ein Programmiermodell generell folgende Fragen beantworten:

- Welche Entwicklungsobjekte werden verwendet, um eine bestimmte Anwendung oder Funktionalität zu realisieren, und welche Aufgaben haben diese Entwicklungsobjekte?
- Wie hängen diese Entwicklungsobjekte zusammen, und wie bauen sie aufeinander auf (welche Abhängigkeiten gibt es also zwischen den Entwicklungsobjekten)?
- Welche Programmierschnittstellen (Application Programming Interfaces, kurz APIs) stehen Ihnen zur Verfügung, um typische Anforderungen einer Anwendung oder bestimmte Funktionalitäten zu realisieren?

Bei den APIs ist es sinnvoll, zwischen zwei verschiedenen Arten zu unterscheiden:

- Welche APIs werden der Anwendung zum *direkten Aufruf* zur Verfügung gestellt, um bestimmte Funktionen zu realisieren? Solche APIs werden über Bibliotheken bereitgestellt, daher spricht man bei diesen APIs auch von *bibliotheksbasierten APIs*.
- Welche APIs werden von der Anwendung *implementiert*, damit diese zu definierten Verarbeitungszeitpunkten von außen, das heißt von dem Framework, das den Kontrollfluss realisiert, aufgerufen werden? Bei diesen APIs spricht man von *Framework-basierten Schnittstellen*.

Ein Programmiermodell umfasst somit sowohl die Zutaten für die Anwendungsentwicklung als auch konkrete Rezeptvorschläge. Es beschreibt, wie Sie diese Zutaten für bestimmte Zwecke einsetzen können, und auch, wie Sie diese miteinander kombinieren.

Eigenschaften des Programmiermodells

Im Folgenden gehen wir auf einige Eigenschaften des ABAP RESTful Application Programming Models bzw. Anforderungen an dieses Programmiermodell ein, die eine strategische Bedeutung für die SAP-Anwendungsentwicklung haben.

Greenfield und Brownfield Development

Das ABAP RESTful Application Programming Model ist darauf ausgelegt, eine langfristig tragbare Lösung zur Realisierung von Geschäftsanwendungen auf der ABAP-Plattform zu sein. Daher können Sie mit diesem Programmiermodell sowohl vollständig neue Geschäftsanwendungen entwickeln (*Greenfield Development*) als auch bestehende Geschäftsanwendungen integrieren, das heißt mit den Mitteln des ABAP RESTful Application Programming Models verschalen (Integration von Legacy-Code oder *Brownfield Development*). So verschalte Bestandsanwendungen profitieren ebenso wie die neu entwickelten Anwendungen von den qualitativen Eigenschaften

des Programmiermodells. Auf den Aspekt der Evolutionsfähigkeit, das heißt der Anpassungsfähigkeit des Programmiermodells selbst an geänderte Rahmenbedingungen (z. B. technologische Änderungen) und folglich auch die Anpassungsfähigkeit von RAP-Anwendungen, gehen wir in Abschnitt 1.5.1, »Evolutionsfähigkeit«, genauer ein.

REST-basierte Entwicklung

Der *REST-Architekturstil* findet sich direkt im Namen des Programmiermodells wieder. Wenn Sie Unternehmensanwendungen mit dem ABAP RESTful Application Programming Model realisieren, entstehen dadurch automatisch APIs und Anwendungen, die REST-basiert sind und den Designprinzipien dieses Architekturstils folgen. Das ABAP RESTful Application Programming Model setzt damit auf einem zustandslosen Webserver als Bestandteil der ABAP-Plattform auf, der REST-basierte APIs bereitstellt. In diesem Zusammenhang kann der Anwendungszustand auf dem Client oder serverseitig persistent auf der Datenbank vorgehalten werden, aber nicht im Rahmen einer ABAP-Session, wie es bei klassischen Anwendungen auf dem *SAP NetWeaver Application Server (SAP NetWeaver AS) ABAP* der Fall ist.

Draft-Handling

Um den Anwendungszustand persistent auf der Datenbank vorzuhalten, können Sie im ABAP RESTful Application Programming Model die Funktion des *Draft-Handlings* nutzen. Anwendungsdaten können dabei als *Draft*, also in einem Entwurfsmodus, für einen Anwender oder eine Anwenderin persistent auf der Datenbank zwischengespeichert werden. Die Pflege der Daten kann dann zu einem späteren Zeitpunkt oder von einem anderen Endgerät aus fortgesetzt werden. Auf diese Weise können die Skalierbarkeit und Lastverteilung innerhalb einer Cloud-Umgebung genutzt werden, da API-Requests an eine RAP-Anwendung voneinander unabhängig und daher von unterschiedlichen Applikationsservern verarbeitet werden können. Diese zustandslose Kommunikation zwischen Client und Server und weitere Details zum REST-Architekturstil erläutern wir in Abschnitt 1.1.2.

Standardarchitektur von Geschäftsanwendungen

Das ABAP RESTful Application Programming Model definiert eine Standardarchitektur von Geschäftsanwendungen und setzt dabei auch Leitplanken für die Entwicklung, um diese Architektur konsequent umsetzen zu können. Daraus ergibt sich ein standardisierter Entwicklungsfluss für die Geschäftsanwendung, bis hin zur Bereitstellung webbasierter APIs zum Zugriff auf die jeweiligen Geschäftsdaten. Auf Basis dieser Standardarchitektur lassen sich beispielsweise moderne Benutzeroberflächen realisieren oder eine Web-API als technische Schnittstelle für integrative Zwecke anbieten.

Standardisierung

Die Standardisierung zeigt sich im ABAP RESTful Application Programming Model durch folgende Eigenschaften:

- **Definierter Entwicklungsfluss**
  Das Programmiermodell stellt für verschiedene Aspekte der Anwendungsentwicklung Entwicklungsobjekte im *ABAP Repository* bereit. Zur Datenmodellierung verwenden Sie das *ABAP Dictionary* (Transaktion DDIC) sowie darauf aufbauend *Core Data Services* (CDS). Die mit CDS definierten Entitäten ergänzen Sie um eine *Verhaltensdefinition*, mit der Sie transaktionale Eigenschaften und Geschäftslogik realisieren können. ABAP-Coding können Sie in der zur Verhaltensdefinition passenden *Verhaltensimplementierung* formulieren. *Business-Services* exponieren schließlich die Entitäten der Anwendung und deren Geschäftslogik als externe (OData-)Schnittstellen. Auf alle hier angesprochenen Entwicklungsobjekte dieses Flusses gehen wir in Abschnitt 1.3 ausführlicher ein.
- **Unterstützung technischer Aspekte**
  Das Programmiermodell berücksichtigt standardmäßig verschiedene technische Aspekte, die typisch für Unternehmensanwendungen sind. Dazu gehören z. B. die Realisierung von Standardoperationen, wie das Anlegen (`create`), Lesen (`read`), Ändern (`update`) oder Löschen (`delete`) von Geschäftsobjekten innerhalb einer logischen Transaktion sowie die Persistenz der transaktionalen Daten (CRUD-Operationen). Auch Nummernvergabe, Persistenz, Sperrverhalten, Draft-Handling oder der Einbau von Berechtigungsprüfungen werden durch das Programmiermodell unterstützt. Das Programmiermodell bietet dazu fertige Implementierungen an, z. B. für das Handling von CRUD-Operationen oder für die Funktion des Draft-Handlings. Sie können Ihr Hauptaugenmerk somit auf die Entwicklung von Geschäftslogik und Geschäftsregeln legen und müssen sich um die genannten typischen Querschnittsthemen der Anwendungsentwicklung nicht mehr kümmern. Auf diese *Entwicklungseffizienz* gehen wir in Abschnitt 1.5.2 genauer ein.
- **Definierter Programmfluss zur Verarbeitung von Geschäftsoperationen**
  Die Laufzeitumgebung des Programmiermodells definiert bereits einen passenden Programmfluss für die Operationen einer Anwendung (wie die Anlage einer Bestellung oder die Stornierung eines Kundenauftrags). Um diesen Programmfluss müssen Sie sich somit keine Gedanken machen. Es gibt klar definierte Zeitpunkte, zu denen Sie sich mit eigenem ABAP-Coding in den Programmfluss einklinken (über Framework-basierte Schnittstellen), um beispielsweise eine Nummernvergabe oder eine Berechtigungsprüfung zu implementieren.

- **Deklarativer, modellbasierter Ansatz**
  Das Programmiermodell verfolgt einen modellbasierten Ansatz. Das bedeutet, dass das Programmiermodell auf einen konkreten Anwendungszweck hin zugeschnittene spezifische Sprachen (*Domain Specific Languages*, DSL) zur Verfügung stellt. Diese grenzen sich von allgemeinen Programmiersprachen ab. Ein gutes Programmiermodell führt den Anwendungsarchitekten bzw. die -entwicklerin so, dass es ihm oder ihr leichtfällt, sich an das Modell zu halten und es »im Sinne des Erfinders« zu verwenden. Das wird durch einen deklarativen, modellbasierten Ansatz erreicht.

Domänenspezifische Sprachen

In Tabelle 1.1 finden Sie eine Liste aller im ABAP RESTful Application Programming Model relevanten domänenspezifischen Sprachen. Die bereits mit SAP NetWeaver 7.40 eingeführten CDS stellen mit der *Data Definition Language* (DDL) beispielsweise eine Sprache zur Verfügung, mit der Sie die semantischen Datenmodelle einer Anwendung definieren können. Ein so definiertes Datenmodell kann direkt als Datenmodell eines OData-Service exponiert werden oder als lesende Schnittstelle für die Entitäten einer Anwendung dienen. Wird das CDS-Datenmodell um RAP-Verhalten erweitert, kann auch dieses nach OData exponiert werden oder zeigt sich als intern aufrufbares Verhalten. Aus den Modellen der Anwendung kann das ABAP RESTful Application Programming Model somit technische Informationen (beispielsweise die OData-Schnittstelle, interne Schnittstellen, Datentypen etc.) ableiten.

Sprache	Bezeichnung
DDL	Data Definition Language
DDLA	Data Definition Language Annotations
DCL	Data Control Language
BDL	Behavior Definition Language
SDL	Service Definition Language

Tabelle 1.1 Domänenspezifische Sprachen im ABAP RESTful Application Programming Model

Unterstützung von SAP-Produktmerkmalen

SAP-Produktmerkmale

Das ABAP RESTful Application Programming Model ist darauf ausgelegt, bestehende SAP-Technologien und Produktmerkmale bestmöglich zu unterstützen. Dazu gehören die folgenden Merkmale:

- **SAP Fiori User Experience**
  Mit dem Programmiermodell können Sie Anwendungen mit einer OData-basierten Schnittstelle entwickeln, auf denen eine SAP-Fiori-Benutzeroberfläche nahtlos aufsetzen kann. Solche Oberflächen können z. B. mit *SAP Fiori Elements* realisiert werden. Für auf diesem Framework basierende Anwendungen steht eine Vorschaufunktion in der Entwicklungsumgebung zur Verfügung.
- **Möglichkeiten der SAP-HANA-Datenbank**
  Das Programmiermodell verwendet CDS als Basistechnologie für das Datenmodell einer Geschäftsanwendung. Dadurch ist ein *Code Pushdown* auf die SAP-HANA-Datenbank aus dem ABAP-Entwicklungskontext heraus möglich. Berechnungen können somit direkt auf der SAP-HANA-Datenbank ausgeführt werden.
- **Cloud Readiness**
  Mit dem Programmiermodell entwickelte Anwendungen sind RESTful, das heißt, sie stellen zustandslose APIs zur Verfügung. Ein serverseitiger Anwendungszustand wird nicht vorgehalten. Die dadurch bedingte Skalierbarkeit ermöglicht es, das ABAP RESTful Application Programming Model auch in Cloud-Umgebungen zu nutzen. Es steht daher auch mit SAP S/4HANA Cloud zur Verfügung.

  Das Cloud-Betriebsmodell erfordert es außerdem, dass Anwendungen bzw. ihre Schnittstellen über Software-Updates hinweg stabil bleiben. Mit dem im ABAP RESTful Application Programming Model eingebauten Erweiterungskonzept lassen sich RAP-Anwendungen modifikationsfrei erweitern, was der Update-Stabilität zugutekommt.

1.1.2 Der REST-Architekturstil

Der Architekturstil *Representational State Transfer* (REST) geht auf die im Jahr 2000 verfasste Dissertation von Roy Fielding zurück. REST beschreibt die Architektur des *World Wide Web* (WWW) und setzte die Leitplanken für dessen Design und Weiterentwicklung.

Softwarebausteine und Standards

In der Theorie ist der Architekturstil unabhängig von der Implementierung durch konkrete Technologien. Aus praktischen Gründen setzen wir die REST-Grundprinzipien und -Architekturelemente jedoch gleich in den Kontext der dem WWW zugrunde liegenden Softwarebausteine und Standards. Zuvor gehen wir jedoch auf die wesentlichen Elemente des WWW ein. Sind Sie damit bereits vertraut, können Sie direkt mit dem Abschnitt »REST-Architekturprinzipien« fortfahren.

World Wide Web

Ressourcen

Mit dem WWW werden Webinhalte oder *Ressourcen* wie HTML-Dateien, Bilddateien oder Programmskripte global und verteilt verfügbar gemacht. Diese Ressourcen werden von einem *HTTP-Server* bereitgestellt und können über einen *HTTP-Client*, wie beispielsweise einen Webbrowser, angefragt werden. Das Kommunikationsprotokoll für die Anfrage und Übertragung von Ressourcen zwischen Client und Server heißt *Hypertext Transfer Protokoll* (HTTP).

Ressourcen

Bei den Ressourcen handelt es sich nicht notwendigerweise um konkrete Dateien, wie HTML-Seiten, Skripte oder Bilddateien. Mit Ressourcen können auch Elemente gemeint sein, die z. B. in einer Datenbank vorgehalten werden, etwa ein Kundenauftrag mit der Nummer 123.

Uniform Resource Identifier und Locator

Eine Ressource ist eindeutig über ihre Webadresse identifizier- und abrufbar. Die Webadresse wird auch *Uniform Resource Locator* (URL) genannt. Eine URL beschreibt also die Lokation einer Ressource in einem Netzwerk. Eine URL ist ein konkreter Anwendungsfall eines *Unified Resource Identifiers* (URI). Der URI-Standard spezifiziert den Aufbau und die Syntax von Webadressen.

HTTP

Der HTTP-Client verwendet die URL, um eine Ressource auf dem HTTP-Server anzufragen. Die Anfrage des Clients wird *HTTP-Request* genannt, die Antwort des Servers ist die *HTTP-Response*, die mit einem HTTP-Statuscode (z. B. `200 OK` bei erfolgreicher Verarbeitung) übermittelt wird. Sowohl beim HTTP-Request als auch bei der HTTP-Response handelt es sich um HTTP-Nachrichten (auch *HTTP-Messages* genannt), die in ihrem HTTP-Body Anwendungsdaten aufnehmen können.

HTTP-Methoden

Die *HTTP-Methode* (auch *HTTP-Verb* genannt) ist ein wesentlicher Teil eines HTTP-Requests und wird von einem HTTP-Client verwendet, um die Art seiner Anfrage an den HTTP-Server zu spezifizieren. Die häufigste HTTP-Methode ist die `GET`-Methode, mit der eine Ressource angefragt wird. Der HTTP-Request `GET www.sap.com HTTP/1.1` liefert Ihnen beispielsweise die Homepage von SAP zurück.

Repräsentationen

Der Server antwortet mit dem HTTP-Request, in dessen HTTP-Body die *Repräsentation* einer Ressource übertragen wird. Diese Repräsentation besteht aus textbasierten oder binären Daten, die die jeweilige Ressource darstellen. Diese werden durch Metadaten genauer beschrieben. Beispielsweise wird der Medien- oder Content-Type angegeben. Damit kann der

Empfänger des Requests diese Anfrage auf passende Weise verarbeiten. Der Content-Type für HTML-Inhalte lautet beispielsweise `text/html`, der für XML-Inhalte `text/xml` und der für JSON-Inhalte `application/json`.

Repräsentationen für schreibende Zugriffe

Repräsentationen können auch von einem HTTP-Client an einen HTTP-Server übertragen werden, beispielsweise wenn die Ressource »Geschäftspartnerstamm mit der Kundennummer 123« von einer Benutzerin oder einem Benutzer über den Webbrowser geändert wurde und diese Änderungen im JSON-Format an den Webserver übertragen werden. Repräsentationen werden also für lesende und schreibende Zwecke verwendet.

Links

URLs werden nicht nur zur Anfrage von Ressourcen verwendet, sondern können auch selbst inhaltlicher Bestandteil der Repräsentation einer Ressource sein. Beispielsweise kann eine HTML-Seite mit *Hyperlinks* (kurz Links) auf weitere HTML-Seiten oder auf die auf der Seite eingebetteten Bilddateien verweisen. Ein solcher Link dient dann für einen HTTP-Client wie einen Webbrowser als Information, wie dieser die weitere HTML-Seite oder die Bilddatei anfragen kann.

REST-Architekturprinzipien

Der REST-Architekturstil beschreibt technische Anforderungen an die Bausteine eines hypertextbasierten Softwaresystems bzw. die Restriktionen. Er beschreibt außerdem die Interaktion zwischen den Bausteinen. Da REST und das WWW eng miteinander verbunden sind, stellen wir die technischen Anforderungen im Folgenden in den Kontext des WWW.

Client-Server-Architektur

Es gibt Systembausteine, die als *Client* oder als *Server* miteinander interagieren. Server stellen Dienste zur Verfügung, die Clients nutzen können. In betrieblichen Anwendungssystemen stellen Serverkomponenten typischerweise Software zur Verfügung, um betriebliche Prozesse abzubilden, also Fach- oder Geschäftslogik. Client-Bausteine können beispielsweise die Benutzeroberfläche bereitstellen, fragen Geschäftsdaten vom Server an und stellen die Antwort des Servers, z. B. die Abfrage von Kundenaufträgen, auf der Benutzeroberfläche dar. Hierfür wird im REST-Architekturstil die Infrastruktur des WWW genutzt. HTTP-Clients bzw. HTTP-Server stellen die grundlegenden Bausteine einer Client-Server-Architektur dar.

Zustandslose Kommunikation

Die Kommunikation zwischen Client- und Serverbausteinen erfolgt zustandslos, das heißt, jede Anfrage des Clients an den Server erfolgt unabhängig von zuvor durchgeführten Anfragen. Die Client-Anfrage muss demnach in sich abgeschlossen und vollständig sein, damit der Server die Anfrage

verstehen und verarbeiten kann. Der Client kann nicht davon ausgehen, dass Kontextinformation auf dem Server über verschiedene Anfragen hinweg vorgehalten werden.

HTTP ist ein zustandsloses Anfrage-Antwort-basiertes Protokoll. Verschiedene HTTP-Requests stehen für den HTTP-Server in keinem inhaltlichen Zusammenhang, er speichert somit keine Kontextinformationen (den sogenannten *Zustand* oder *State*) über die Requests. Die Kommunikation wird daher *zustandslos* oder *stateless* genannt.

Vollständige Repräsentation einer Ressource

Ein per HTTP formulierter REST-Request zur Anlage eines Kundenauftrags sendet die vollständige Repräsentation dieser Ressource beispielsweise über den JSON-Content mit. Das heißt, die gesamten Daten des Kundenauftrags müssen in dem Request enthalten sein, damit der REST-konforme Server diesen verarbeiten kann.

In klassischen Webanwendungen ist es möglich, den Sitzungszustand (*Session State*) auf dem HTTP-Server über eine Session-Kennung (*Session-ID*) vorzuhalten. Die Session-Kennung kann als Cookie oder URL-Parameter innerhalb einer HTTP-Anfrage abgebildet werden. Auf diese Weise können mehrere HTTP-Anfragen mit einer Session-Kennung sich auf einen Sitzungszustand beziehen. So lässt sich auf Anwendungsebene eine zustandsbehaftete Kommunikation zwischen Client und Server realisieren. Eine solche Implementierung ist jedoch *nicht* REST-konform.

Pufferung

Innerhalb der Kommunikation zwischen Client und Server kann eine Serverantwort, das heißt eine Repräsentation, *gepuffert* werden, um damit die allgemeine Netzwerklast zu reduzieren oder Antwortzeiten zu verringern. Dazu werden Repräsentationen mit Metadaten versehen, die sich auf die Pufferung beziehen und die Gültigkeitsdauer des Puffers festlegen. Ist eine Serverrepräsentation pufferbar, kann die gepufferte Repräsentation als Ergebnis für gleichartige Client-Anfragen verwendet werden, anstatt eine erneute Anfrage an den ursprünglichen Server zu stellen.

Schichten von Systemen

Die *Schichtung* von Systemen bedeutet, dass Client- und Serverkomponenten zwar in verschiedenen Schichten angeordnet sein können, aber nur mit der jeweils nächsten Schicht interagieren sollen. Dem Client muss also nicht bekannt sein, ob eine Antwort vom originären Server stammt oder von einem Server in einer mittleren Schicht.

Im WWW nimmt eine Anfrage von einem HTTP-Client verschiedene technische Wege zum originären Server. Dabei können Server zwischengeschaltet

sein, die als *Proxy* agieren, das heißt die Anfrage entgegennehmen und diese selbst an den angefragten HTTP-Server weiterleiten.

Einheitliche Schnittstelle

Client- und Serverbausteine interagieren auf Basis einer einheitlichen Schnittstelle. Einheitlich bedeutet hier, dass die Schnittstelle zwischen Client und Server anwendungsübergreifend standardisiert und somit unabhängig von der konkreten (betrieblichen) Anwendung ist. Der Server stellt durch die Implementierung der Schnittstelle einen Dienst zur Verfügung, der Client konsumiert diesen Dienst über die Schnittstelle. Durch die Trennung von Schnittstelle und Implementierung sind Client und Server voneinander entkoppelt (beide »kennen« nur die Schnittstelle) und können somit in verschiedenen Programmiersprachen und Ablaufumgebungen realisiert sein.

Eine einheitliche REST-basierte Schnittstelle spezifiziert Folgendes:

- Identifikation von Ressourcen
- Manipulation (also schreibende Zugriffe) von Ressourcen, erfolgend über eine Repräsentation der Ressource
- selbst beschreibende Nachrichten
- Hypermedia as the Engine of Application State (HATEOS)

HATEOS

HATEOS bezeichnet den Mechanismus, dass ein Client über Links mit entsprechender URL innerhalb der Repräsentationen von Ressourcen zu anderen Ressourcen navigieren bzw. diese Ressourcen manipulieren kann. Der Client muss dabei lediglich die Links und deren Typen kennen, um mit dem Server in Interaktion treten zu können. Die konkrete URL muss dem Client nicht bekannt sein, da sie Teil der vom Server zurückgelieferten Repräsentation ist. Diese Repräsentation ist der Anwendungszustand (*Application State*).

Abfrage von Kundenauftragspositionen

Beispielsweise kann die JSON-Repräsentation einer Entität »Kundenauftrag« einen Link anbieten, unter dem alle Einträge der Entität »Kundenauftragsposition« zu dem konkreten Kundenauftrag mit der Nummer 123 zu finden sind. Eine `GET`-Anfrage an diese URL wird dann alle Kundenauftragspositionen zum Kundenauftrag 123 zurückliefern. Außerdem beinhaltet sie weitere Links, um beispielsweise zu einer konkreten Kundenauftragsposition 10 navigieren zu können. Auf diese Weise bildet sich ein Netz an Verlinkungen, die ein Client aufgrund der einheitlich definierten Schnittstelle nutzen kann.

Code-on-Demand (optional)

Bei Bedarf kann eine Client-Komponente um Programmcode erweitert werden, der auf dem Server vorgehalten wird. Diese Eigenschaft ist im REST-Architekturstil optional.

Sind die im REST-Architekturstil definierten technischen Anforderungen und Restriktionen eingehalten, sprechen wir von einer *REST-konformen Softwarearchitektur*. Da die technische Realisierung von Unternehmensanwendungen auf verschiedenen Plattformen und mit verschiedenen Programmiersprachen erfolgen kann, sollte ein Hauptaugenmerk darauf liegen, dass diese Anwendung REST-konforme oder -basierte Schnittstellen (*RESTful APIs*) bereitstellt. Daher gehen wir im Folgenden näher auf die wesentlichen Eigenschaften REST-basierter Schnittstellen ein und legen deren typische Funktionsweise dar.

RESTful APIs

Die REST-basierte Architektur einer Softwareanwendung folgt den im vorangehenden Abschnitt beschriebenen Architekturprinzipien, verwendet die technische Infrastruktur des WWW und exponiert REST-basierte APIs, diese beispielsweise von einer Weboberfläche konsumiert werden können.

Einheitliche Schnittstelle

Eine der beschriebenen Anforderungen an REST ist die Verwendung einer einheitlichen Schnittstelle. Wir nehmen im Folgenden auf diese Anforderung Bezug und erläutern, mit welchen Mitteln eine REST-basierte API typischerweise umgesetzt wird. Dazu gehören die folgenden Mechanismen:

- Über HTTP-Methoden werden Standardoperationen für die Entitäten umgesetzt, die eine Unternehmensanwendung modellieren. Solche Operationen können sowohl lesende als auch schreibende Zugriffe sein.
- Der Body von HTTP-Nachrichten wird verwendet, um die Repräsentationen von Ressourcen zwischen Client und Server zu transferieren, das heißt, um lesende und schreibende Zugriffe auf Entitäten zu realisieren.
- Der Content-Type (im HTTP-Header oder -Body) wird verwendet, um das Format dieser Repräsentation anzugeben.
- Zur HTTP-Methode passende Statuscodes werden in der HTTP-Response zurückgeliefert.
- Weitere HTTP-Header werden verwendet, um z. B. mit konkurrierenden Zugriffen auf Geschäftsobjekte umzugehen. Als Beispiel sei hier das *ETag-Verfahren* genannt, eine Umsetzung des optimistischen Sperrverfahrens.
- Links werden in den ausgetauschten Repräsentationen von Ressourcen verwendet, um auf weitere Ressourcen zu verweisen und auf diese Weise

URLs anzubieten, mit denen der Client die Ressource lesen oder ändern kann.

HTTP-Methoden einer RESTful API

Im Folgenden listen wir alle relevanten HTTP-Methoden auf und beschreiben ihre Bedeutung, ihre Verwendung ist Teil der Umsetzung des REST-Architekturprinzips der Verwendung einer einheitlichen Schnittstelle:

- **GET**
  Die `GET`-Methode fordert einen oder mehrere Ressourcen an. Eine HTTP-`GET`-Abfrage setzt einen lesenden Zugriff um, der vollständig ohne Seiteneffekte implementiert sein muss. Bei erfolgreicher Verarbeitung liefert die Response den Statuscode `200 OK` zurück. Falls die Ressource nicht gefunden wurde, wird Statuscode `404 Not found` zurückgeliefert.
- **POST**
  Eine HTTP-`POST`-Abfrage legt eine neue Ressource an. Bei erfolgreicher Verarbeitung liefert die Response idealerweise den Statuscode `201 Created` zurück. Gängig sind allerdings auch `200 OK` oder `204 No content`.
- **PUT**
  Eine HTTP-`PUT`-Abfrage ändert eine bestehende Ressource. Bei erfolgreicher Verarbeitung liefert die Response den Statuscode `200 OK` oder `204 No content` zurück.
- **DELETE**
  Eine HTTP-`DELETE`-Abfrage löscht eine bestehende Ressource. Bei erfolgreicher Verarbeitung liefert die Response den Statuscode `200 OK` oder `204 No content` zurück.
- **PATCH**
  Eine HTTP-`PATCH`-Abfrage ändert eine Auswahl von Attributen einer bestehenden Ressource. Bei erfolgreicher Verarbeitung liefert die Response den Statuscode `200 OK` oder `204 No content` zurück. Von einer `PUT`-Abfrage lässt sich die `PATCH`-Abfrage wie folgt abgrenzen: `PUT` ersetzt im Gegensatz zu `PATCH` eine Ressource vollständig durch die übermittelte Repräsentation.

1.1.3 OData

Das *Open Data Protocol* (OData) ist eine Spezifikation einer REST-konformen API auf Basis von HTTP. Als offener Standard wird es vom OASIS-Konsortium verwaltet. Ein *OData-Service* exponiert Anwendungsdaten für lesende und/oder schreibende Zugriffe eines Konsumenten. Das OData-Protokoll legt die Kommunikation zwischen Konsument und OData-Service fest.

Metadaten eines OData-Service

OData definiert beispielsweise, dass ein OData-Service ein Datenmodell besitzt und wie dieses aufgebaut ist. In einem solchen Datenmodell sind die Entitäten (`EntityType`) mit ihren Attributen (`Property`) und Assoziationen (`Association`, `NavigationProperty`) beschrieben.

Das Datenmodell eines OData-Service wird über das *Servicemetadatendokument* bereitgestellt. In Abbildung 1.1 sehen Sie ein Beispiel für ein Servicemetadatendokument aus dem neuen ABAP-Flugdatenmodell, dem *ABAP Flight Reference Scenario*. `TravelType` ist hier eine im Datenmodell definierte Entität.

Abbildung 1.1 Beispiel eines OData-Service-Metadatendokuments

Servicedokument

Des Weiteren liefert jeder OData-Service ein *Servicedokument* zurück, das alle exponierten Entitätsmengen (`EntitySet`), mit den dafür gültigen Standardoperationen bereitstellt. Das Servicedokument dient als Einstieg in die bereitgestellten Ressourcen über die darin aufgeführten Links. In Abbildung 1.2 sehen Sie ein Beispiel für ein Servicedokument aus dem ABAP Flight Reference Scenario. `Travel` ist eine über den Dienst zur Verfügung gestellte Entitätsmenge.

Abbildung eines OData-Service auf HTTP

OData legt weiterhin fest, auf welche Weise die Elemente von HTTP (HTTP-Request, -Response, Statuscodes etc.) genutzt werden, um Daten, CRUD-Operationen und weitere Nicht-Standardoperationen einer Anwendung über einen OData-Service bereitzustellen. Beispielsweise gibt es URL-Konventionen zur Identifizierung, Abfrage und Manipulation von Ressourcen oder zur Darstellung von Repräsentationen innerhalb der HTTP-Nachricht.

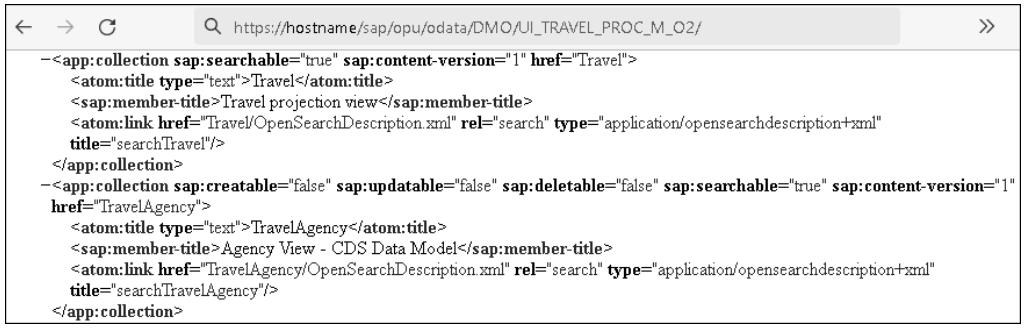


Abbildung 1.2 Beispiel eines OData-Servicedokuments

Zu den durch OData definierten Konventionen gehören auch ein reichhaltiger Satz an Abfrageoptionen für Geschäftsdaten (Filtern, Projizieren, Sortieren, Blättern etc.) oder das Handling von Binärdaten (Media-Link-Einträge) über die Elemente von HTTP.

OData-Vokabulare und Annotationen

Während OData an sich bereits eine Fülle an Funktionalität spezifiziert, mit der ein Konsument auf eine Anwendung zugreifen kann, ist es mit *OData-Vokabularen* möglich, das Datenmodell eines OData-Service um spezifische Konzepte anzureichern. Diese Konzepte bilden typischerweise anwendungsübergreifende Funktionalität ab, etwa Analytics-Verfahren oder die Ausgabe von Daten auf einem User Interface (UI). Über einen OData-Service exponierte *Annotationen* werden in der Anwendung verwendet, um Elemente des Datenmodells mit zusätzlicher Information anzureichern. Sie können beispielsweise durch die Benutzeroberfläche ausgewertet werden.

So kann es im Bereich des UI beispielsweise das Konzept des Titels, abgebildet als Annotation, geben. In einer Anwendung zur Personenverwaltung könnten Sie z. B. den `EntityType` »Person« um eine »Titel«-Annotation anreichern, die besagt, dass sich der UI-Titel aus den Properties »Nachname« und »Vorname« zusammensetzt. Eine UI-Komponente einer Weboberfläche kann diese Annotation zur Laufzeit auswerten und einen UI-Titel generieren, der die aktuellen Werte für Vorname und Nachname einer Personeninstanz ausgibt.

Abfragen und Operationen

Im Folgenden stellen wir Ihnen einige wichtige OData-Abfragen vor. Da OData auf HTTP basiert, ist ein OData-Service über einen HTTP-Endpunkt erreichbar. Dieser Endpunkt wird auch *Service-URL* genannt und dient als Einstieg in den OData-Service. Unter der Service-URL erreichen Sie das bereits erwähnte Servicedokument.

Um im Folgenden die wesentlichen OData-Anfragen zu skizzieren, verwenden wir einen OData-Service aus dem ABAP Flight Reference Scenario, der

über Business-Services in das ABAP RESTful Application Programming Model exponiert wurde (siehe auch Abschnitt 1.3.5, »Business-Services«). Der zugehörige HTTP-Endpunkt lautet */sap/opu/odata/DMO/UI_TRAVEL_PROC_M_O2/*. Informationen zu den verwendeten HTTP-Methoden finden Sie in Abschnitt 1.1.2, »Der REST-Architekturstil«.

Wenn Sie das Datenmodell eines OData-Service abfragen möchten, verwenden Sie eine HTTP-`GET`-Anfrage `GET /<service-url>/$metadata`, beispielsweise `GET /sap/opu/odata/DMO/UI_TRAVEL_PROC_M_O2/$metadata`. Wenn Sie einen konkreten Eintrag abfragen möchten, beispielsweise eine konkrete Reise, verwenden Sie ebenfalls HTTP-`GET`. Die Anfrage `GET /.../DMO/UI_TRAVEL_PROC_M_O2/Travel('1')` liest eine Travel-Ressource mit dem Schlüssel 1. Wenn Sie einen neuen Eintrag anlegen möchten, sieht OData einen HTTP-`POST` vor, für Änderungen HTTP-`PUT`, und um einen Eintrag zu löschen, verwenden Sie HTTP-`DELETE`.

Detailinformationen zu den URL-Konventionen, Abfrageoptionen und weiteren Aspekten des OData-Protokolls finden Sie unter der URL *https://www.odata.org*.

Navigation zur Service-URL

Wenn Sie die Beispiele dieses Abschnitts im Webbrowser nachvollziehen möchten, können Sie dazu das Service-Binding */DMO/UI_TRAVEL_PROC_M_O2* in den ABAP Development Tools öffnen und mit einem Klick auf **Service URL** zum HTTP-Endpunkt des OData-Service navigieren. Hostname und Port werden passend ergänzt, und das Servicedokument öffnet sich im Webbrowser. Im Browser können Sie das Servicedokument bzw. das Servicemetadatendokument lesen sowie weitere lesende Operationen durchführen. Was es mit dem Service-Binding auf sich hat, erfahren Sie in Abschnitt 6.3, »Service-Binding«.

RESTful APIs ausprobieren

Unter der Adresse *https://api.sap.com* erreichen Sie den *SAP Business Accelerator Hub* (vormals *SAP API Business Hub*), in dem SAP die exponierten Remote-APIs seiner Produkte bereitstellt. Dort können Sie die APIs ansehen und auch ausprobieren. Als Beispiel bietet sich die OData-basierte *Business Partner API* der SAP S/4HANA Cloud an (*http://s-prs.de/v1007802*).

1.1.4 Technologische Neuerungen mit SAP S/4HANA

SAP S/4HANA

SAP S/4HANA ist der Nachfolger von SAP ERP und die neueste und modernste betriebswirtschaftliche Standardsoftware aus dem Hause SAP. Mit SAP S/4HANA wurden grundlegende technologische Änderungen der ABAP-Plattform vorgenommen, die weitreichenden Einfluss darauf haben, wie Sie Anwendungen anpassen und erweitern oder vollständig neue Anwendungen erstellen. Diese Änderungen betreffen alle technischen Softwareschichten:

- SAP-HANA-Datenbank
- Core Data Services (CDS) und virtuelles Datenmodell
- SAP Gateway und OData
- SAP Fiori User Experience mit dem SAP Fiori Launchpad und dem SAPUI5-Framework

SAP-HANA-Datenbank

Mit SAP HANA ist eine In-Memory-Datenbanktechnologie in das Umfeld der betriebswirtschaftlichen Software von SAP eingezogen. Die SAP-HANA-Datenbank hält Datenbestände abhängig von ihrer Zugriffskategorie größtenteils im Hauptspeicher des Datenbankservers. Um die Zugriffsgeschwindigkeit lesender Zugriffe zu erhöhen und den Speicherbedarf im Hauptspeicher zu reduzieren, werden Datenbestände spaltenorientiert im sogenannten *Column Store* abgelegt und automatisch indiziert.

Core Data Services

Mit der Einführung von CDS ergibt sich für Sie die Möglichkeit eines Code Pushdowns in die SAP-HANA-Datenbank. Berechnungen können damit direkt dort, wo die Daten liegen, also auf dem Datenbankserver, ausgeführt werden. Die jeweiligen Daten müssen dazu nicht mehr erst in den Hauptspeicher des Applikationsservers geladen werden. Mit CDS können Sie außerdem ein semantisches Datenmodell Ihrer Anwendung auf der Basis von Datenbanktabellen definieren. Die CDS-Entitäten können Sie an verschiedenen Stellen in die Anwendung integrieren. Das mit CDS modellierte semantische Datenmodell von SAP S/4HANA heißt *virtuelles Datenmodell* (VDM).

SAP Gateway und OData

Mit OData kann eine Anwendung auf einheitliche Weise REST-konforme APIs zur Verfügung stellen. Wie in Abschnitt 1.1.3, »OData«, beschrieben, nutzt OData offene Standards wie HTTP, XML oder JSON. *SAP Gateway* ist ein Produkt, das die ABAP-Welt mit dem OData-Protokoll verknüpft. Während der Entwicklung haben Sie die Möglichkeit, lesende oder schreibende Zugriffe auf Anwendungsdaten mit den Funktionen von SAP Gateway zu re-

alisieren, ohne die Details des OData-Protokolls kennen zu müssen. SAP Gateway stellt auch Mittel zur Verfügung, um die Funktionalitäten der CDS zu integrieren und auf dieser Basis OData-Services zu exponieren.

SAP Fiori UX

Mit SAP S/4HANA wurden die Benutzeroberfläche und das gesamte Look-and-feel, also die *User Experience* (UX) des SAP-Systems weiterentwickelt. Rollenbasierte Anwendungen lösen transaktionsbasierte Anwendungen ab. Die neuen Anwendungen stellen die jeweilige Aufgabe der Anwenderin oder des Anwenders in den Mittelpunkt, indem sie die Auswahl an sichtbaren Feldern und Dialogschritten auf diese Aufgabe zuschneiden. SAP-Fiori-Apps sind moderne, browserbasierte Anwendungen, die technologisch durch das SAPUI5-Framework umgesetzt werden. Sie erfüllen außerdem die Anforderungen der *SAP Fiori Design Guidelines*.

Alle einer Anwenderin oder einem Anwender zur Verfügung stehenden SAP-Fiori-Apps werden im *SAP Fiori Launchpad* als Kacheln angezeigt. SAP-Fiori-Apps bzw. SAPUI5-Anwendungen sind darauf ausgelegt, Daten aus dem Backend über das OData-Protokoll zu lesen und über dieses Protokoll auch wieder zurück ins Backend zu schreiben. Sie kommunizieren daher in aller Regel mit SAP Gateway, um auf Anwendungsfunktionalität zuzugreifen.

1.1.5 Evolution der ABAP-basierten Programmiermodelle

Um die Neuerungen des ABAP RESTful Application Programming Models besser verstehen zu können, ist es hilfreich, die technologischen Einflüsse und historischen Entwicklungen im ABAP-Umfeld zu kennen. Wir gehen daher in diesem Abschnitt auf die Entstehungsgeschichte des Programmiermodells ein und stellen Vorgängertechnologien vor, die einen wesentlichen Einfluss auf dessen Bestandteile hatten.

Vorgänger-programmier-modelle

Wir besprechen folgende Programmiermodelle und Technologien:

- klassische Anwendungsentwicklung mit ABAP
- das Business Object Processing Framework (BOPF)
- das ABAP-Programmiermodell für SAP Fiori

In Abbildung 1.3 sehen Sie von links nach rechts die zeitliche Abfolge der ABAP-basierten Programmiermodelle bis zur Einführung des ABAP RESTful Application Programming Models. BOPF ist dabei nicht als eigenes Programmiermodell aufgeführt, sondern findet z. B. im Programmiermodell für SAP Fiori Anwendung. Da darin wichtige Konzepte verankert sind, führen wir im Folgenden trotzdem kurz in dieses Framework ein.

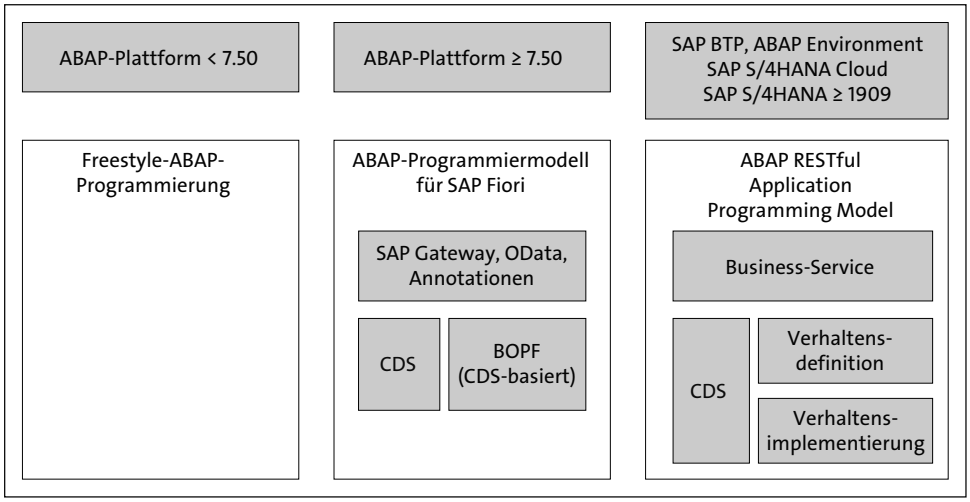


Abbildung 1.3 Evolution ABAP-basierter Programmiermodelle (Quelle: SAP)

Klassische Anwendungsentwicklung mit ABAP

Ursprünglich aus SAP R/3 hervorgegangen, ist der *SAP NetWeaver AS ABAP* (häufig als *ABAP-Stack* bezeichnet), die zentrale technologische Laufzeitumgebung für ABAP-basierte Anwendungen. Der Applikationsserver ist ein von der jeweiligen Anwendung unabhängiges Produkt und stellt zentral technische Dienste zur Anwendungsentwicklung bereit, wie UI-Technologien, Schnittstellentechnologien, Bibliotheken sowie Komponenten zur Druckausgabe, Hintergrundverarbeitung und Prozessierung von Geschäftsregeln.

Klassisches Programmiermodell

Das klassische Programmiermodell für ABAP basiert auf den folgenden Elementen:

- **Datenmodellierung mit dem ABAP Dictionary**

  Mit dem ABAP Dictionary legen Sie globale Datentypen wie Datenelemente, Strukturen und Datenbanktabellen an. Mit Version 7.40 Support Package 5 (SP05) des SAP NetWeaver AS ABAP wurde die CDS-Technologie eingeführt, mit der, aufbauend auf dem ABAP Dictionary, semantische Datenmodelle entwickelt werden können.

- **Entwicklung und Kapselung von Geschäftslogik**

  In ABAP stehen Ihnen verschiedene Modularisierungseinheiten wie Formroutinen, Funktionsbausteine oder Klassen und Interfaces zur Verfügung, um Programmlogik zu kapseln. Wir empfehlen Ihnen, Klassen bzw. Interfaces als Mittel der Objektorientierung zur Modularisierung ei-

ner Anwendung zu verwenden. Das ABAP RESTful Application Programming Model verwendet ausschließlich Klassen und Interfaces.

- **UI-Entwicklung (SAP-GUI- oder webbasiert)**
  Der ABAP-Stack stellt verschiedene technische Mittel zur Realisierung von Benutzeroberflächen bereit. Klassischerweise können etwa SAP-GUI-basierte Benutzeroberflächen unter Verwendung der *Dynpro-Technologie* erstellt werden. Webbasierte UIs können mithilfe der Technologien *Business Server Pages* (BSP), *Web Dynpro ABAP* oder des *Floorplan Managers für Web Dynpro ABAP* realisiert werden. In seltenen Fällen werden mit dem klassischen Programmiermodell schon SAPUI5-Oberflächen genutzt.
- **Schnittstellenentwicklung**
  Des Weiteren bietet der ABAP-Stack verschiedene Technologien zur Anwendungsintegration an: IDoc, SOAP, RFC, HTTP und SMTP. Diese Technologien werden sowohl aus Konsumenten- als auch aus Provider-Sicht unterstützt. SAP Gateway, als Implementierung des OData-Protokolls, ist seit Version 7.40 integraler Bestandteil des SAP NetWeaver AS ABAP.

ABAP-Sprachkonstrukte ab Version 7.40

Mit SAP NetWeaver AS ABAP 7.40 ist eine wichtige neue Funktionalität in den ABAP-Sprachumfang eingegangen, die für die Anwendungsentwicklung mit dem ABAP RESTful Application Programming Model besonders hilfreich ist. Diese Änderungen umfassen *Konstruktor-* und *Tabellenausdrücke* (ab Version 7.40 SP02, 7.40 SP05 bzw. SP08). Mit Konstruktor- und Tabellenausdrücken können Sie Datenobjekte auf komfortable Weise mit Werten versorgen. Im Kontext des ABAP RESTful Application Programming Models erleichtern Sie die Verhaltensimplementierung und den Umgang mit EML deutlich. Die neuen ABAP-Schlüsselwörter `VALUE`, `CORRESPONDING`, `FOR`, `COND` etc. sind dabei besonders hervorzuheben.

Das Business Object Processing Framework

Kontrollfluss von Unternehmensanwendungen

Während der ABAP-Stack Technologien und Infrastruktur bereitstellt, die die Peripherie der Anwendung betreffen (Benutzeroberfläche, Schnittstellen, Persistenz etc.), wurde mit dem *Business Object Processing Framework* (BOPF) erstmals ein Framework entwickelt, das die unmittelbare Umsetzung des fachlichen Kerns einer Anwendung unterstützte. BOPF basiert auf ABAP Objects. In seiner Eigenschaft als Framework definiert es den Kontrollfluss der Verarbeitung von Geschäftslogik. Eine ABAP-Anwendung implementiert bestimmte BOPF-Interfaces und klinkt sich so in diesen Kontrollfluss ein.

Einige zentrale Elemente von BOPF sind:

Konzepte von BOPF

- **Geschäftsobjekte**
  Sie werden auch *Business-Objekte* genannt, bilden betriebswirtschaftliche Objekte ab und sind hierarchisch in Knoten mit (Schlüssel-)Attributen aufgebaut.
- **Geschäftslogik**
  Die Anwendungslogik wird in Form von Validierungen, Ermittlungen und Aktionen implementiert.
- **Geschäftsobjekt-API**
  Diese API dient der multiinstanzfähigen Implementierung von Geschäftsobjekten und deren interner Geschäftslogik. Konsumenten kennen die Geschäftsobjekt-API nicht, sondern greifen über die Consumer-API darauf zu.
- **Consumer-API**
  Diese API ermöglicht Konsumenten (z. B. ABAP-Quellcode im Kontext einer Benutzeroberfläche oder eines Hintergrundjobs) die Nutzung von Geschäftsobjektfunktionalität.

Viele dieser Konzepte werden Sie im ABAP RESTful Application Programming Model wiederfinden, auch wenn sie dort technisch anders realisiert wurden. Im ABAP RESTful Application Programming Model ist die Verarbeitung von Operationen und Geschäftslogik im ABAP-Core aufgegangen, das heißt, sie wird über spezielle ABAP-Schlüsselwörter umgesetzt, die von der ABAP-Laufzeitumgebung ausgeführt werden. Diese Art der Integration in den ABAP-Core kommt Ihnen vielleicht von der Erweiterungstechnik der *Business Add-ins* (BAdIs) bekannt vor. Zunächst war das BAdI-Framework in ABAP Objects realisiert worden (ebenso wie BOPF). Im später eingeführten *Enhancement Framework* gingen die BAdIs dann als kernelbasierte BAdIs in den ABAP-Core ein.

Verfügbarkeit von BOPF

BOPF steht mit dem Release 7.40 des SAP NetWeaver AS ABAP zur Nutzung zur Verfügung.

Das ABAP-Programmiermodell für SAP Fiori

Auf die technologischen Neuerungen im Kontext von SAP S/4HANA aus Sicht der Entwicklung reagierte das *ABAP-Programmiermodell für SAP Fiori*. Es nutzt Technologien wie CDS, BOPF und SAP Gateway und steht mit Ver-

sion 7.50 des ABAP-Stacks zur Verfügung – auch wenn Sie noch kein SAP-S/4HANA-System einsetzen. Mit der Version 7.50 SP01 wurde die Funktion des Draft-Handlings ergänzt.

Konzepte des Programmiermodells

Wesentliche Elemente dieses Programmiermodells sind:

- **Datenmodellierung mit CDS**
  Mit CDS wird das Datenmodell der Anwendung definiert. CDS-Entitäten können dann als OData-Services exponiert werden, um den lesenden Zugriff auf die jeweiligen Anwendungsdaten zu ermöglichen. Um in ABAP-Programmen auf native SAP-HANA-Funktionalität zugreifen zu können (Code Pushdown), stehen CDS-Tabellenfunktionen, ABAP Managed Database Procedures (AMDP) und SQLScript zur Verfügung.
- **Transaktionales Verhalten mit BOPF**
  Mittels einer leichtgewichtigen CDS-basierten Variante von BOPF fügen Sie den CDS-Entitäten transaktionales Verhalten hinzu.
- **SAP Gateway**
  OData-basierte Schnittstellen werden über SAP Gateway exponiert. Zentral ist dabei die Transaktion SEGW, über die Sie die ABAP-Welt mit OData verbinden und beispielsweise CDS-Entitäten als referenzierte Datenquellen exponieren können. SAP Gateway kann dabei auf einem eigenen ABAP-Stack installiert sein (in diesem Fall spricht man vom *Hub Deployment*) oder auf demselben Stack wie die jeweilige Anwendung der SAP Business Suite oder SAP S/4HANA (*Embedded Deployment*).
- **SAPUI5 und SAP Fiori**
  Die Entwicklung der Benutzeroberflächen erfolgt mit SAPUI5 oder SAP Fiori Elements. Mit SAPUI5 können die Oberflächen frei gestaltet werden, während SAP Fiori Elements Vorlagen für bestimmte Anwendungsfälle bereitstellt. Die Anwendungen werden den Anwenderinnen und Anwendern typischerweise über das SAP Fiori Launchpad zugänglich gemacht. *UI-Annotationen*, die über CDS oder direkt in der SAPUI5-Anwendung hinterlegt werden, definieren als Metadaten das Layout und Verhalten der Benutzeroberfläche.

Das Zusammenspiel dieser Komponenten verdeutlicht Abbildung 1.4. SAP Gateway kann als integraler Bestandteil des ABAP-Stacks lokal mit dem ABAP-Backend kommunizieren, da beide Komponenten auf einem Applikationsserver laufen. Grundsätzlich ist SAP Gateway auch auf einem eigenen ABAP-Stack lauffähig, der dann als Frontend-Server agiert und per Remote Function Call (RFC) mit dem ABAP-Backend kommuniziert.

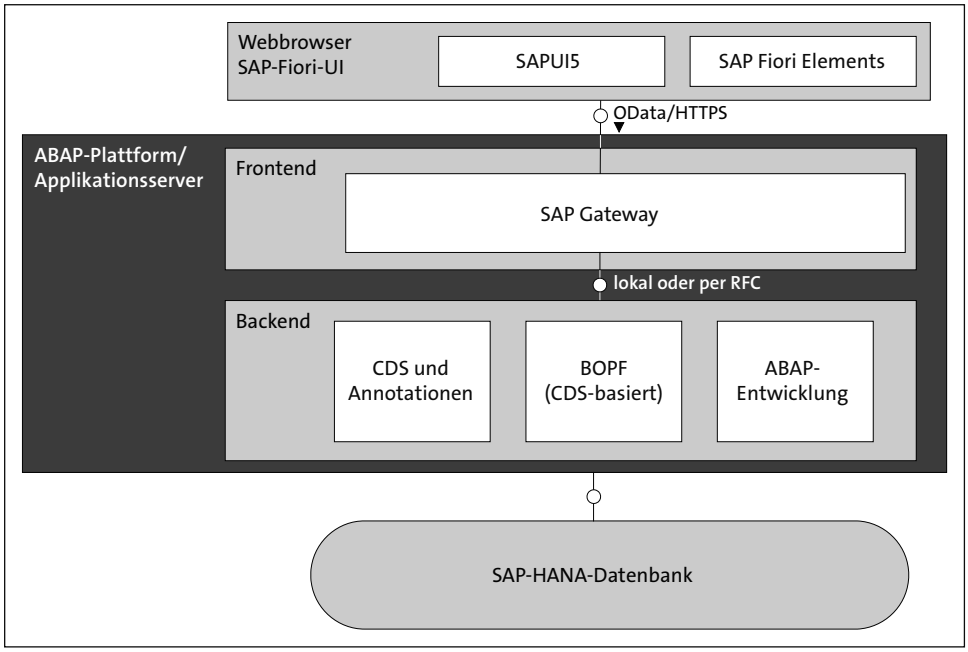


Abbildung 1.4 ABAP-Programmiermodell für SAP Fiori

Migration der CDS-basierten BOPF-Objekte in das ABAP RESTful Application Programming Model

Aktuell entwickelt SAP ein Werkzeug zur Migration der CDS-basierten BOPF-Objekte, die im Kontext des ABAP-Programmiermodells für SAP Fiori entstanden sind. Künftig können Sie hierfür eine BOPF-verwaltete Implementierung verwenden (Implementierungstyp »BOPF-Managed«).

1.2 Architektur und Konzepte des ABAP RESTful Application Programming Models

In diesem Abschnitt gehen wir ausführlicher auf die wesentlichen architekturrelevanten Konzepte des ABAP RESTful Application Programming Models ein. Dazu gehören auch die technischen Komponenten, in die eine RAP-Anwendung zur Laufzeit eingebettet ist.

1.2.1 RAP-Transaktionsmodell

Verarbeitungsphasen

Das *RAP-Transaktionsmodell* unterscheidet zwei Phasen in der Verarbeitung von Operationen auf Geschäftsobjekten:

Vorwort

Alle, die sich mit dem Thema Softwareentwicklung beschäftigen, sei es in der Rolle eines Entwicklers, einer Architektin oder eines Projektleiters, kennen die Herausforderungen und Unwägbarkeiten, die stetig neue technologische Trends, Frameworks, Tools und Standards mit sich bringen. Die treibende Kraft hinter den Bemühungen ist häufig, dass die Entwicklung betrieblicher Anwendungssoftware weiter vereinfacht und damit effizienter gestaltet werden soll. Dabei ist allerdings auch zu berücksichtigen, dass sich bestehende Software und die damit getätigten Investitionen gut den neuen Trends und Standards anpassen bzw. darin integrieren lassen sollen. Das ist in der Realität häufig nicht der Fall, denn oftmals werden technische Aspekte und fachliche Logik im Quellcode einer Anwendung vermischt. Das erschwert die Anpassbarkeit an veränderte technologische Rahmenbedingungen, wie etwa eine neue UI- oder Schnittstellentechnologie, und macht zum Teil sogar komplette Neuentwicklungen notwendig.

Während die Technik die eine Seite der Medaille der Softwareentwicklung ausmacht, steht auf der anderen Seite die vielfach zitierte Geschäftslogik. Über die Jahre haben wir immer wieder das folgende Credo gehört, das Sie sicherlich auch kennen:

> *»Die Anwendungsentwicklung soll sich mehr auf ihre eigentliche Arbeit fokussieren, nämlich die Implementierung der Geschäftslogik auf Basis der Anforderungen der jeweiligen Fachbereiche.«*

Wird eine bessere Fokussierung gefordert, muss es wohl offensichtlich irgendeine Form der »Ablenkung« geben. Diese Ablenkung besteht z. B. in sich verändernden technologischen Rahmenbedingungen, die neben den fachlichen Anforderungen bei der Realisierung berücksichtigt werden müssen. Sie treiben typischerweise die Implementierungskosten (Total Cost of Development, kurz TCD) in die Höhe. Solche Ablenkungen gilt es also zu eliminieren.

Vor diesem Hintergrund ist in der Vergangenheit bereits der ABAP-Stack entstanden, ein reichhaltiges und stabiles technisches Fundament, auf dem die Anwendungsentwicklung schon seit vielen Jahren mit großem Erfolg aufsetzt. Das Business Object Processing Framework (BOPF) hat sich erstmalig der Standardisierung von Geschäftslogik angenommen, also der fachlichen Seite von Geschäftsanwendungen. Das ABAP-Programmiermodell für SAP Fiori greift BOPF in einer CDS-basierten Variante auf und entkoppelt

sich von technischen Komponenten, wie beispielsweise SAP Fiori Elements, durch die Verwendung von Annotationen.

Wenn wir also die Entwicklung von ABAP als Sprache zur Erstellung betrieblicher Anwendungssoftware mit den Grundideen von BOPF, den Programmiermodellen der ABAP-Plattform sowie dem Wunsch, evolutionsfähige Software zu bauen oder bestehende Software mit einem »Evolutionsschutz« auszustatten, vermengen, dann ist die logische Konsequenz ein im ABAP-Kern verankertes Programmiermodell. Es bringt eigene Repository-Objekte sowie umfangreiche Erweiterungen der ABAP-Sprache mit sich und sorgt von Haus aus dafür, dass Fachlichkeit und Technik zwei getrennte Seiten der gleichen Medaille bleiben. Da die Reise der Softwareentwicklung in die Cloud führt, bietet sich ein REST-basierter Ansatz an, um Anwendungsinteraktion von der genutzten Cloud-Infrastruktur zu entkoppeln. Als Ergebnis dieser Überlegungen ist das ABAP RESTful Application Programming Model zur Entwicklung von SAP-Fiori-Apps und Web-APIs entstanden.

Wir danken den Autoren und dem Verlag für die Erstellung dieses Buches und wünschen allen Lesenden viel Freude damit.

Volker Drees
RAP Product Expert

Marcel Hermanns
RAP Product Lead/Chief Development Expert

Einleitung

Seit jeher entwickeln sich die Programmiersprache ABAP und das zugehörige Programmiermodell stetig weiter. Mit dem aktuellen *ABAP RESTful Application Programming Model* (RAP) haben sich einschneidende Veränderungen in der Art und Weise ergeben, Anwendungen zu entwickeln. Dieses Programmiermodell sieht einige komplett neue Programmierkonzepte vor, wie die Verhaltensdefinitionen und Business-Services.

Ziel dieses Buches

Das Ziel dieses Buches ist es, Ihnen einen umfassenden Überblick über das neue Programmiermodell zu geben. Sie erwerben nicht nur das erforderliche theoretische Hintergrundwissen, sondern wenden das Erlernte auch in drei umfangreichen Praxiskapiteln an. Die dort beschriebenen praktischen Anwendungsfälle können Ihnen als Grundlage dienen, um die Vorgehensweise für Ihre eigenen Entwicklungsprojekte dem neuen Programmiermodell anzupassen.

Zielgruppe

In Teil I, »Konzepte des ABAP RESTful Application Programming Models«, wird das Fundament für den anschließenden Praxisteil II, »Praktische Anwendungsentwicklung mit dem ABAP RESTful Application Model«, gelegt, und die grundlegenden Konzepte des Programmiermodells werden im Detail vorgestellt. Damit ist der erste Teil des Buches für Leserinnen und Leser mit unterschiedlichen Vorkenntnissen geeignet. Er richtet sich sowohl an ABAP-Entwicklerinnen und -Entwickler, die bereits Erfahrungen mit dem Programmiermodell gesammelt haben und ihre Kenntnisse vertiefen wollen, als auch an solche, die bisher noch keine Erfahrungen mit den neuen Konzepten haben. Grundlegende Kenntnisse der ABAP-Entwicklung sollten jedoch bereits vorhanden sein. Für das Verständnis des Buches ist es unerheblich, ob Sie in einer Cloud-Umgebung oder auf einem SAP-S/4HANA-on-Premise-System arbeiten.

Wenn Sie neu in das Thema einsteigen, bietet es sich an, das Buch vom ersten bis zum letzten Kapitel chronologisch durchzuarbeiten. Wenn Sie bereits erste Erfahrungen mit dem ABAP RESTful Application Programming Model gesammelt haben, kann es durchaus sinnvoll sein, wenn Sie einzelne Kapitel überspringen und gezielt tiefer in die für Sie interessanten Themen eintauchen. Die Beispiele aus Teil II sind voneinander unabhängig und bauen nicht aufeinander auf.

Programmierbeispiele zum Download

Alle Programmierbeispiele aus dem Buch stehen auf der Seite *www.sap-press.de/5869* auf der Registerkarte **Materialien** zur Verfügung. Von einigen Entwicklungsobjekten der Beispieldatenmodelle und -anwendungen werden in den Listings dieses Buchs nur Ausschnitte abgedruckt. In diesem Fall finden Sie den vollständigen Quelltext ebenfalls im Download-Bereich.

Aufbau dieses Buches

Wie bereits erwähnt, besteht das Buch aus zwei Teilen. In **Teil I** lernen Sie die grundlegenden Konzepte des ABAP RESTful Application Programming Models kennen.

Teil I

In **Kapitel 1**, »Einführung in das ABAP RESTful Application Programming Model«, stellen wir das Programmiermodell vor und ordnen seine Softwarearchitektur in andere Architekturmodelle ein. Die verschiedenen Anwendungsfälle und Varianten der RAP-Entwicklung werden hier ebenfalls erläutert.

In **Kapitel 2**, »Core Data Services: Datenmodellierung«, werden die Möglichkeiten und Funktionen von CDS beschrieben und anhand von Beispielen erläutert. Da es sich bei CDS um einen zentralen Baustein des Programmiermodells handelt, bildet dieses Kapitel die Grundlage für die darauf aufbauenden Sprachelemente und Entwicklungsobjekte.

Kapitel 3, »Verhaltensdefinition«, erläutert aufbauend auf der Einführung in CDS die verschiedenen Implementierungstypen des Programmiermodells. Wir beschreiben den Aufbau, die wesentlichen Sprachelemente und die Möglichkeiten von Verhaltensdefinitionen anhand zahlreicher Beispiele.

In **Kapitel 4**, »Entity Manipulation Language: Zugriff auf Geschäftsobjekte«, werden das Konzept, die Syntax und die Möglichkeiten der in ABAP integrierten EML beschrieben. An dieser Stelle schlagen wir auch bereits die Brücke zur Verhaltensimplementierung.

Kapitel 5, »Verhaltensimplementierung«, erläutert im Detail, wie das Interface zur Implementierung der Geschäftslogik einer Anwendung aufgebaut ist und an welchen Stellen und zu welchen Zeitpunkten eine Implementierung von Verhalten umgesetzt werden kann.

In **Kapitel 6**, »Business-Services«, zeigen wir Ihnen, wie Sie mithilfe einer Servicedefinition den Umfang eines OData-Service für ein Geschäftsobjekt festlegen können und wie Sie mit dem Service-Binding das Protokoll des Service festlegen.

Kapitel 7, »Erweiterbarkeit von Geschäftsobjekten«, beschreibt, wie Sie RAP-Geschäftsobjekte erweiterbar machen und die Erweiterbarkeit solcher Geschäftsobjekte selbst nutzen können. In diesem Zusammenhang verdeutlichen wir einige Möglichkeiten zur Erweiterung anhand von praktischen Beispielen.

Kapitel 8, »Anwendungsoberflächen und SAP Fiori Elements«, richtet den Blick auf die Benutzeroberflächen Ihrer RAP-Anwendungen. Hier gehen wir nicht in der Tiefe auf alle Funktionen von SAP Fiori ein, sondern zeigen Ihnen, wie Sie mit SAP Fiori Elements einfache Oberflächen generieren und mit Annotationen näher konfigurieren können. Damit möchten wir vor allem die Grundlage für die folgenden Praxiskapitel schaffen.

Teil II

In **Teil II** werden die verschiedenen Entwicklungsszenarien des ABAP RESTful Programming Models jeweils anhand eines konkreten Beispiels veranschaulicht.

Zunächst betrachten wir dazu in **Kapitel 9**, »Anwendungsfälle«, die unterschiedlichen Anwendungsfälle des Programmiermodells in der ABAP-Entwicklung und diskutieren die Eignung der verschiedenen RAP-Szenarien im jeweiligen Kontext. Das Kapitel beantwortet damit die Frage, wann welcher Implementierungstyp eingesetzt werden soll.

Ihre erste Anwendung auf Basis des ABAP RESTful Application Programming Models erstellen Sie in **Kapitel 10**, »Managed Scenario: Entwicklung einer Anwendung mit SAP Fiori Elements«. Dabei handelt es sich um eine komplett eigenständige Anwendung ohne Bezug zum SAP-Standard. Im Fokus steht hier die Umsetzung des Managed Scenarios. Die Anwendung wird im Rahmen der Verhaltensdefinition und -implementierung mit Aktionen, Validierungen und Ermittlungen angereichert.

Auch in **Kapitel 11**, »Managed Scenario mit Unmanaged Save: Bestehende Anwendung integrieren«, befassen wir uns mit dem Managed Scenario, implementieren allerdings eine eigene Verbuchungslogik. Themen wie die Nummernvergabe des integrierten RAP-Geschäftsobjekts oder die Implementierung der Speichersequenz sind hier zentral. Ziel dieser zweiten Anwendung ist es, SAP-Standardfunktionen in die Eigenentwicklung einzubinden.

In **Kapitel 12**, »Unmanaged Scenario: Vorhandenen Quellcode wiederverwenden«, gehen wir von einer bereits bestehenden kundeneigenen Anwendung aus, die im Rahmen des Unmanaged Scenarios mit dem ABAP RESTful Application Programming Model abgebildet wird. Ein wesentlicher Punkt in diesem Kapitel ist die Implementierung der Interaktionsphase, da diese

beim Unmanaged Scenario im Gegensatz zum Managed Scenario nicht vom RAP-Framework zur Verfügung gestellt wird.

In **Kapitel 13**, »Besonderheiten in der Cloud-Umgebung«, stellen wir die Unterschiede zwischen der Cloud-Entwicklung und der On-Premise-Entwicklung heraus. Sie erfahren auch, wie Sie eine RAP-Anwendung den technischen Gegebenheiten des ABAP Environments anpassen können.

Informationskästen

In hervorgehobenen Informationskästen finden Sie in diesem Buch Inhalte, die wissenswert und hilfreich sind, aber etwas außerhalb der eigentlichen Erläuterung stehen. Damit Sie diese Informationen sofort einordnen können, haben wir die Kästen mit entsprechenden Symbolen gekennzeichnet:

In Kästen, die mit diesem Symbol gekennzeichnet sind, finden Sie Informationen zu *weiterführenden Themen*.

Dieses Symbol weist Sie auf *Besonderheiten* hin, die Sie beachten sollten. Es *warnt Sie* außerdem vor häufig gemachten Fehlern oder Problemen, die auftreten können.

Die mit diesem Symbol gekennzeichneten *Tipps* geben Ihnen spezielle Empfehlungen aus unserer Projektpraxis, die Ihnen die Arbeit erleichtern können.

Beispiele, durch dieses Symbol kenntlich gemacht, weisen auf Szenarien aus der Praxis hin und veranschaulichen die dargestellten Funktionen.

Danksagungen

Wir möchten uns für die freundliche Unterstützung und die Bereitstellung der Infrastruktur durch die All for One Group SE bedanken, ohne die dieses Buch nicht möglich gewesen wäre. Insbesondere bedanken wir uns bei Lars Woll und Andreas Klose.

Ein besonderer Dank geht außerdem an die Kollegen bei SAP für die tatkräftige Unterstützung und das konstruktive Feedback. Hier möchten wir speziell Marcel Hermanns und Volker Drees hervorheben.

Vielen Dank auch an Janina Schweitzer von SAP PRESS für die sehr gute Zusammenarbeit und kompetente Unterstützung im Rahmen des Lektorats.

Ein individueller Dank geht ...

- von Michael Lensch an seine Frau Stefanie und den gemeinsamen Sohn Paul für die Unterstützung und das Verständnis dafür, dass er unzählige Stunden und viele Wochenenden in das Buch investiert hat.

- von Matthias Jäger an seine beiden Kinder Emilia und Clemens, die ihm geduldig und unterstützend Raum und Zeit für dieses Buch gegeben haben.

Wir hoffen, dass Ihnen dieses Buch als hilfreiches Nachschlagewerk und Leitfaden für die Entwicklung von Anwendungen auf Basis des ABAP RESTful Application Programming Models dienen wird und Ihnen den Einstieg in das neue Programmiermodell erleichtert. Viel Vergnügen beim Lesen!

Lutz Baumbusch, **Matthias Jäger** und
Michael Lensch

Inhalt

Vorwort 17
Einleitung 19

TEIL I Konzepte des ABAP RESTful Application Programming Models

1 Einführung in das ABAP RESTful Application Programming Model 27

1.1 Was ist das ABAP RESTful Application Programming Model? ... 28
1.1.1 Aufgaben des Programmiermodells 28
1.1.2 Der REST-Architekturstil 33
1.1.3 OData 39
1.1.4 Technologische Neuerungen mit SAP S/4HANA 43
1.1.5 Evolution der ABAP-basierten Programmiermodelle 44

1.2 Architektur und Konzepte des ABAP RESTful Application Programming Models 49
1.2.1 RAP-Transaktionsmodell 49
1.2.2 Implementierungstypen 50
1.2.3 Entity Manipulation Language 52
1.2.4 Technischer Kontext einer RAP-Anwendung und RAP-Laufzeitumgebung 53

1.3 Entwicklungsobjekte des ABAP RESTful Application Programming Models 56
1.3.1 Datenmodellierung mit Core Data Services 56
1.3.2 Verhaltensdefinition 57
1.3.3 Verhaltensimplementierung 58
1.3.4 Projektionsschicht 59
1.3.5 Business-Services 60
1.3.6 Zusammenspiel der Artefakte 61

1.4 ABAP Development Tools als Entwicklungswerkzeug 62

1.5 Qualitative Eigenschaften des ABAP RESTful Application Programming Models ... 63
1.5.1 Evolutionsfähigkeit ... 64
1.5.2 Entwicklungseffizienz ... 65
1.5.3 Testbarkeit ... 66
1.5.4 Trennung zwischen Fachlichkeit und Technik ... 67

1.6 Verfügbarkeit des ABAP RESTful Application Programming Models ... 67
1.6.1 SAP BTP, ABAP Environment ... 68
1.6.2 SAP S/4HANA Cloud, ABAP Environment ... 69
1.6.3 ABAP-Plattform für SAP S/4HANA on premise ... 70

1.7 Die Rolle des ABAP RESTful Application Programming Models im ABAP-Cloud-Entwicklungsmodell ... 71

2 Core Data Services: Datenmodellierung 79

2.1 Was sind Core Data Services? ... 80

2.2 Aufbau und Syntax von Core Data Services ... 84
2.2.1 Erstellung eines Basic Interface Views ... 85
2.2.2 Analyse des Datenmodells ... 90
2.2.3 CDS Views verwenden ... 93
2.2.4 Erweiterung des Datenmodells ... 94

2.3 Assoziationen ... 99

2.4 Annotationen ... 104

2.5 Zugriffskontrollen ... 110

2.6 Erweiterbarkeit von CDS-Entitäten ... 116
2.6.1 CDS-View-Erweiterungen ... 116
2.6.2 CDS-Metadatenerweiterungen ... 121

2.7 Weitere CDS-Funktionalität ... 124
2.7.1 Virtuelle Elemente ... 124
2.7.2 CDS Custom Entities ... 128

2.8 Virtuelles Datenmodell ... 132

2.9 CDS-Sprachelemente zur Modellierung von Geschäftsobjekten ... 137

3 Verhaltensdefinition 141

3.1 Was ist eine Verhaltensdefinition? ... 142
3.1.1 Kontext und Aufbau einer Verhaltensdefinition ... 142
3.1.2 Syntax einer Verhaltensdefinition ... 146
3.1.3 Mögliches Verhalten ... 147

3.2 Verhaltensdefinition in den ABAP Development Tools bearbeiten ... 155
3.2.1 Eine Verhaltensdefinition anlegen ... 156
3.2.2 Eine Verhaltensdefinition ändern und aktivieren ... 159
3.2.3 Eine Verhaltensdefinition suchen und öffnen ... 161
3.2.4 Verhaltensdefinitionen und Beziehungen dokumentieren ... 162

3.3 Implementierungstypen ... 165
3.3.1 Managed Scenario ... 166
3.3.2 Unmanaged Scenario ... 169

3.4 Strict-Modus ... 170

3.5 Entitätsverhaltensdefinition ... 171

3.6 Behavior Pool definieren ... 172
3.6.1 Behavior Pool zur Verhaltensdefinition ... 173
3.6.2 Behavior Pool zur CDS-Entität ... 173
3.6.3 Behavior Pool zur Implementierungsgruppe ... 174

3.7 Nummernvergabe ... 175
3.7.1 Frühe, externe Nummernvergabe ... 177
3.7.2 Frühe, interne Nummernvergabe ... 177
3.7.3 Späte Nummernvergabe ... 179

3.8 Feldeigenschaften ... 180
3.8.1 Pflichtfelder ... 181
3.8.2 Schutz vor schreibenden Zugriffen ... 181
3.8.3 Kombination: Pflichtfeld im Anlagefall, Schreibschutz bei Änderungen ... 183

3.9 Feld-Mappings ... 183

3.10 Standardoperationen für eine CDS-Entität ... 186
3.10.1 Create, Read, Update und Delete ... 186
3.10.2 Create- und Read-Operation per Assoziation ... 188

3.11 Spezifische Operationen für eine CDS-Entität ... 190
3.11.1 Aktionen ... 190
3.11.2 Funktionen ... 199
3.11.3 Funktionen zur Feldvorbelegung ... 202

3.12 Konkurrierende Zugriffe und Sperrverhalten ... 205
3.12.1 Pessimistisches Sperren ... 205
3.12.2 Optimistisches Sperren ... 208

3.13 Interne Geschäftslogik ... 210
3.13.1 Ermittlungen ... 211
3.13.2 Validierungen ... 217
3.13.3 Ermittlungen oder Validierungen über Aktion aufrufen 220

3.14 Berechtigungsprüfungen ... 222
3.14.1 Authorization-Master ... 224
3.14.2 Authorization-Dependent ... 226
3.14.3 Berechtigungsprüfungen delegieren ... 227

3.15 Draft-Handling ... 228
3.15.1 Draft-Handling aktivieren ... 229
3.15.2 Draft-Handling im Geschäftsobjekt-Kompositionsbaum 230
3.15.3 Draft-Lebenszyklus und Draft-Aktionen ... 232
3.15.4 Seiteneffekte ... 235

3.16 Ereignisse ... 243
3.16.1 Manuell ausgelöste Ereignisse ... 244
3.16.2 Abgeleitete Ereignisse ... 245

3.17 Übergreifende Konzepte ... 247
3.17.1 Dynamisches Feature Control ... 247
3.17.2 Vorprüfungen von Operationen ... 252
3.17.3 Interne Sichtbarkeit von Operationen ... 253

4 Entity Manipulation Language: Zugriff auf Geschäftsobjekte 257

4.1 Datentypen ... 258
4.1.1 Abgeleitete Datentypen ... 259
4.1.2 Implizite Rückgabeparameter ... 261

4.2 EML-Operationen ... 262
4.2.1 READ ENTITIES ... 263
4.2.2 MODIFY ENTITIES ... 265

4.2.3 GET PERMISSIONS ... 271
4.2.4 SET LOCKS ... 272
4.2.5 COMMIT ENTITIES ... 273
4.2.6 ROLLBACK ENTITIES ... 274

4.3 Verwendung von EML außerhalb von Verhaltensimplementierungen ... 274
4.3.1 Verwendung im Rahmen eines ABAP-Reports ... 274
4.3.2 Implementierung im Rahmen einer Testklasse ... 276

4.4 Konkrete Anwendungsfälle ... 277

5 Verhaltensimplementierung 283

5.1 Business Object Provider API ... 283

5.2 Laufzeitverhalten des ABAP RESTful Application Programming Models ... 284
5.2.1 Interaktionsphase und Transaktionspuffer ... 285
5.2.2 Speichersequenz ... 287

5.3 Schnittstellen für den Interaktionshandler und den Speicherhandler ... 288

5.4 Interaktionshandler ... 289
5.4.1 FOR MODIFY ... 290
5.4.2 FOR INSTANCE AUTHORIZATION ... 294
5.4.3 FOR GLOBAL AUTHORIZATION ... 296
5.4.4 FOR FEATURES ... 299
5.4.5 FOR GLOBAL FEATURES ... 302
5.4.6 FOR LOCK ... 303
5.4.7 FOR READ ... 305
5.4.8 FOR READ per Assoziation ... 306
5.4.9 FOR DETERMINE ... 309
5.4.10 FOR VALIDATE ... 310
5.4.11 FOR NUMBERING ... 311
5.4.12 FOR PRECHECK ... 312

5.5 Speicherhandler ... 314
5.5.1 FINALIZE ... 315
5.5.2 CHECK_BEFORE_SAVE ... 316
5.5.3 ADJUST_NUMBERS ... 318
5.5.4 SAVE ... 319

5.5.5 CLEANUP ... 322
5.5.6 CLEANUP_FINALIZE ... 323

5.6 Ereignisse ... 324
5.6.1 Auslösung von Ereignissen ... 324
5.6.2 Konsumierung von Ereignissen ... 325

6 Business-Services 331

6.1 Projektionsschicht ... 332
6.1.1 CDS Projection View ... 333
6.1.2 Projektionsverhaltensdefinition ... 334

6.2 Servicedefinition ... 336

6.3 Service-Binding ... 337

6.4 Business-Services im SAP Gateway Client testen ... 341

6.5 UI-Services mit der SAP-Fiori-Elements-Vorschau testen ... 344

6.6 BO-Interfaces ... 345
6.6.1 Aufbau eines BO-Interface ... 345
6.6.2 Verwendung von BO-Interfaces ... 348
6.6.3 Einsatz von BO-Interfaces als neue BAPIs ... 348

7 Erweiterbarkeit von Geschäftsobjekten 355

7.1 Einführung in das Erweiterungskonzept ... 355

7.2 Erweiterungsoptionen ... 360
7.2.1 Erweiterungen des Datenmodells ... 360
7.2.2 Erweiterungen des Verhaltens ... 367
7.2.3 Erweiterbarkeit um zusätzliche CDS-Entitäten ... 372

7.3 Ein Standardgeschäftsobjekt erweitern ... 376
7.3.1 Beschreibung des Anwendungsfalls ... 377
7.3.2 Datenmodell erweitern ... 380
7.3.3 Verhalten erweitern ... 390
7.3.4 Abgeleitete Ereignisse ... 401

8 Anwendungsoberflächen und SAP Fiori Elements 407

8.1 Entwicklungswerkzeuge ... 408
8.1.1 SAP Business Application Studio ... 408
8.1.2 Visual Studio Code ... 411

8.2 SAP-Fiori-Elements-Oberflächen für RAP-Anwendungen ... 412
8.2.1 Floorplans in SAP Fiori Elements ... 412
8.2.2 Ausgewählte UI-Annotationen ... 415
8.2.3 UI-Annotationen in einem CDS View definieren ... 416
8.2.4 Annotationen mit dem Service Modeler generieren ... 434

TEIL II Praktische Anwendungsentwicklung mit dem ABAP RESTful Application Programming Model

9 Anwendungsfälle 449

9.1 Einsatzgebiete des ABAP RESTful Application Programming Models ... 449

9.2 Abgrenzung der verschiedenen Implementierungstypen ... 450

9.3 Entscheidungskriterien zur Auswahl des Implementierungstyps ... 452

10 Managed Scenario: Entwicklung einer Anwendung mit SAP Fiori Elements 457

10.1 Beschreibung des Anwendungsfalls ... 458

10.2 Datenmodell aufbauen ... 458
10.2.1 Datenbanktabellen ... 459
10.2.2 CDS-Modellierung ... 463

10.3 Verhaltensdefinitionen erstellen ... 473
10.3.1 Verhaltensdefinitionen für die Zertifikatsverwaltung anlegen ... 473
10.3.2 Draft-Handling aktivieren ... 478
10.4 Business-Service definieren ... 480
10.4.1 Servicedefinition anlegen ... 481
10.4.2 Service-Binding erstellen ... 483
10.5 SAP-Fiori-Elements-Oberfläche erstellen ... 485
10.6 Anreicherung um eine Ermittlung ... 493
10.7 Anreicherung um eine Validierung ... 498
10.8 Anreicherung um eine Aktion ... 502
10.9 Generierung und Deployment der Anwendung ... 505
10.10 Datei-Upload ... 512

11 Managed Scenario mit Unmanaged Save: Bestehende Anwendung integrieren 517

11.1 Beschreibung des Anwendungsfalls ... 518
11.2 Datenmodell aufbauen ... 522
11.2.1 Überblick über das logische Datenmodell ... 523
11.2.2 Datenbanktabellen ... 526
11.2.3 CDS-Modellierung ... 529
11.3 Verhaltensdefinition erstellen ... 535
11.4 Funktion »Bestellung anlegen« realisieren ... 537
11.4.1 Verwaltete Nummernvergabe deklarieren ... 538
11.4.2 Feldeigenschaften setzen ... 538
11.4.3 Behavior Pool anlegen ... 540
11.4.4 Ermittlungen realisieren ... 542
11.4.5 Speichersequenz: Anlage per BO-Interface realisieren ... 549
11.4.6 Validierungen realisieren ... 559
11.5 Funktion »Bestellung löschen« realisieren ... 565
11.5.1 Speichersequenz: Löschen per BO-Interface realisieren ... 566
11.5.2 Validierung realisieren ... 571

11.6 Business-Services definieren ... 571
11.6.1 Projektionsschicht für die Anwendung »Eigene Bestellungen« aufsetzen ... 572
11.6.2 Servicedefinition anlegen ... 574
11.6.3 Service-Binding anlegen ... 575

11.7 Berechtigungsprüfungen realisieren ... 575
11.7.1 Zugriffskontrollen für Lesezugriffe ... 576
11.7.2 Zugriffskontrollen für Schreibzugriffe ... 578

11.8 SAP-Fiori-Elements-Oberfläche erstellen ... 581
11.8.1 Metadatenerweiterung anlegen ... 581
11.8.2 Generierung und Deployment der Anwendung ... 584

12 Unmanaged Scenario: Vorhandenen Quellcode wiederverwenden 587

12.1 Beschreibung des Anwendungsfalls ... 588

12.2 Beschreibung der bestehenden Anwendung ... 589
12.2.1 Datenbanktabellen ... 590
12.2.2 Quellcode der bestehenden Anwendung ... 593

12.3 Datenmodell erweitern ... 596

12.4 Verhaltensdefinition erstellen ... 603

12.5 Verhaltensimplementierung erstellen ... 607
12.5.1 Implementierung der Interaktionsphase ... 610
12.5.2 Implementierung der Speichersequenz ... 619

12.6 Business-Service definieren ... 622

13 Besonderheiten in der Cloud-Umgebung 627

13.1 Technische Grundlagen ... 628
13.1.1 ABAP for Cloud Development ... 632
13.1.2 Technische Infrastrukturkomponenten ... 633
13.1.3 Migration von Bestandscoding ... 635

13.2 Identity and Access Management ... 636

13.3 SAP-Fiori-Apps bereitstellen und Berechtigungen vergeben 639
13.3.1 IAM-App und -Anwendungskatalog anlegen 641
13.3.2 IAM-Anwendungsrolle anlegen 643
13.3.3 Integration ins SAP Fiori Launchpad 644

13.4 Business-Services konsumieren 648

Anhang 653

A Literaturverzeichnis 653

Das Autorenteam 655
Index 657

Die neue Art, ABAP zu programmieren

Technologie und Konzepte

Was macht den REST-Architekturstil aus? Welche Rolle spielen BO-Interfaces? Mit diesem Buch erwerben Sie das Grundlagenwissen für die Entwicklung eigener Anwendungen mit dem neuen Programmiermodell.

Vorgehen bei der Entwicklung

Von der Erstellung des Datenmodells über die Anwendungslogik bis hin zur Benutzeroberfläche werden Sie durch alle Schritte der Entwicklung geführt. Sie erhalten praktische Beispiele für die Neuimplementierung, Optimierung und Erweiterung von ABAP-Anwendungen.

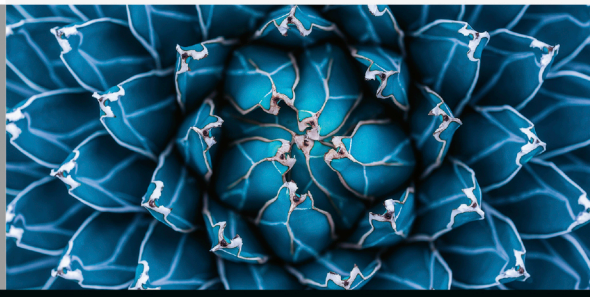
Neu in dieser Auflage

Die Autoren greifen die neuesten Weiterentwicklungen des Programmiermodells auf: Erweiterbarkeit von RAP-Geschäftsobjekten, BO-Interfaces, neue Schlüsselwörter in der Verhaltensdefinition – alles im praktischen Einsatz.

Auf einen Blick

- REST-Architektur
- Core Data Services (CDS)
- Verhaltensdefinition und -implementierung
- Definition und Binding von Business-Services
- Felddefinitionen, Standardoperationen und Aktionen
- Entity Manipulation Language (EML)
- SAP Fiori Elements
- Implementierungstypen Managed und Unmanaged
- BO-Interfaces
- Erweiterbarkeit von ABAP-Anwendungen
- ABAP Cloud

»Unentbehrlich, wenn man als ABAP-Entwickler für die Zukunft gerüstet sein will«
Leser-Feedback zur Vorauflage



Das Autorenteam
Lutz Baumbusch ist SAP-Entwickler und Softwarearchitekt bei der All for One Group SE. Michael Lensch ist dort für ein internationales Team von SAP-Entwicklerinnen und -Entwicklern verantwortlich. Matthias Jäger arbeitet als selbstständiger Fullstack-ABAP-Entwickler, Architekt und Trainer.

Rheinwerk Publishing

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

€ 89,90 [D] € 92,50 [A]
ISBN 978-3-367-10078-1