

ABAP RESTful Application Programming Model

Das umfassende Handbuch

» Hier geht's
direkt
zum Buch

DIE LESEPROBE

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.

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.

Aufbau des Kapitels



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.

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

Greenfield und
Brownfield
Development

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.

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

■ **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.

Domänen-spezifische 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-Produkt-merkmale

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.

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.

**Softwarebausteine
und Standards**

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.



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.

Links

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.

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.

Client-Server-Architektur

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

Zustandslose Kommunikation

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 Sitzungsstatus (*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 Sitzungsstatus 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.

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.

Einheitliche
Schnittstelle

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 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*).

HATEOS

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.

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:

HTTP-Methoden
einer RESTful API

■ **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 *Service-metadatendokument* bereitgestellt. In Abbildung 1.1 sehen Sie ein Beispiel für ein Service-metadatendokument aus dem neuen ABAP-Flugdatenmodell, dem *ABAP Flight Reference Scenario*. TravelType ist hier eine im Datenmodell definierte Entität.

```

</edmx:Reference>
-<edmx:DataServices m:DataServiceVersion="2.0">
  -<Schema Namespace="cbs_xdmoxmzi_travel_processor_m" xml:lang="de" sap:schema-version="1">
    <Annotation Term="Core.SchemaVersion" String="1.0.0"/>
    +<EntityType Name="BookingType" sap:label="Booking projection view" sap:content-version="1"></EntityType>
    +<EntityType Name="BookingSupplementType" sap:label="Booking supplement projection view" sap:content-version="1">
      </EntityType>
    -<EntityType Name="TravelType" sap:label="Travel projection view" sap:content-version="1">
      -<Key>
        <PropertyRef Name="TravelID"/>
      </Key>
      <Property Name="copyTravel_ac" Type="Edm.Boolean" sap:label="Dyn. Aktions-Control" sap:creatable="false"
        sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
      <Property Name="Delete_mc" Type="Edm.Boolean" sap:label="Dyn. Methodensteuerung" sap:creatable="false"
        sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
      <Property Name="Update_mc" Type="Edm.Boolean" sap:label="Dyn. Methodensteuerung" sap:creatable="false"
        sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
      <Property Name="to_Booking_oc" Type="Edm.Boolean" sap:label="Dyn. AdA-Steuerung" sap:creatable="false"
        sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
      <Property Name="TravelID" Type="Edm.String" Nullable="false" MaxLength="8" sap:display-format="NonNegative"
        sap:label="Travel ID" sap:quickinfo="Flight Reference Scenario: Travel ID" sap:creatable="false" sap:updatable="false"/>
      <Property Name="AgencyID" Type="Edm.String" MaxLength="6" sap:display-format="NonNegative" sap:text="AgencyName"
        sap:label="Agency ID" sap:quickinfo="Flight Reference Scenario: Agency ID" sap:value-list="standard"/>
      <Property Name="AgencyName" Type="Edm.String" MaxLength="80" sap:label="Agency Name" sap:quickinfo="Flight
        Reference Scenario: Agency Name" sap:creatable="false" sap:updatable="false"/>
    </EntityType>
  </Schema>
</edmx:DataServices>

```

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.

```

- <app:collection sap:searchable="true" sap:content-version="1" href="Travel">
  <atom:title type="text">Travel</atom:title>
  <sap:member-title>Travel projection view</sap:member-title>
  <atom:link href="Travel/OpenSearchDescription.xml" rel="search" type="application/opensearchdescription+xml"
  title="searchTravel"/>
</app:collection>
- <app:collection sap:creatable="false" sap:updatable="false" sap:deletable="false" sap:searchable="true" sap:content-version="1"
  href="TravelAgency">
  <atom:title type="text">TravelAgency</atom:title>
  <sap:member-title>Agency View - CDS Data Model</sap:member-title>
  <atom:link href="TravelAgency/OpenSearchDescription.xml" rel="search" type="application/opensearchdescription+xml"
  title="searchTravelAgency"/>
</app:collection>

```

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.

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.

OData-Vokabulare
und Annotationen

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.

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.

Abfragen und
Operationen

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

Detaillinformationen 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 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 Software-schichten:

SAP S/4HANA

- 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

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.

SAP-HANA-Datenbank

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).

Core Data Services

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-

SAP Gateway und OData

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.

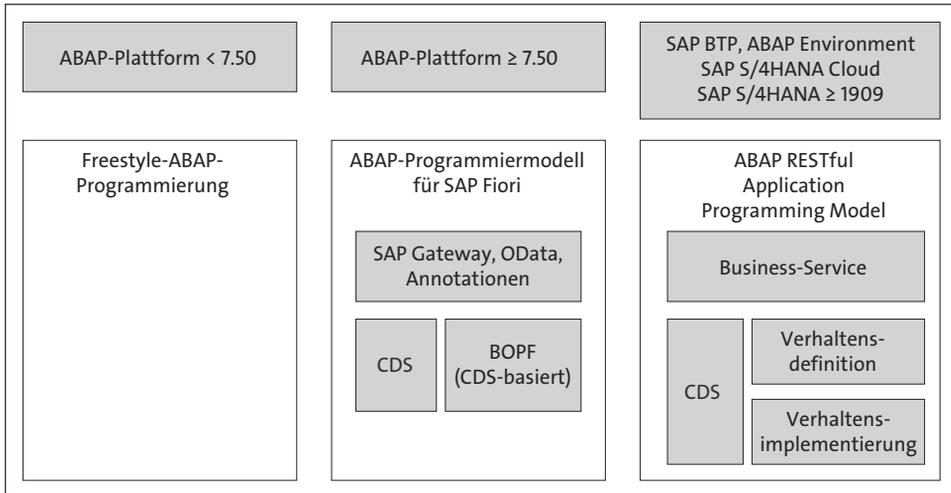


Abbildung 1.3 Evolution ABAP-basierter Programmiermodelle (Quelle: SAP)

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.

Das klassische Programmiermodell für ABAP basiert auf den folgenden Elementen:

Klassisches
Programmiermodell

■ 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 (SPO5) 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 SPO2, 7.40 SPO5 bzw. SPO8). 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
Programmier-
modells

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.

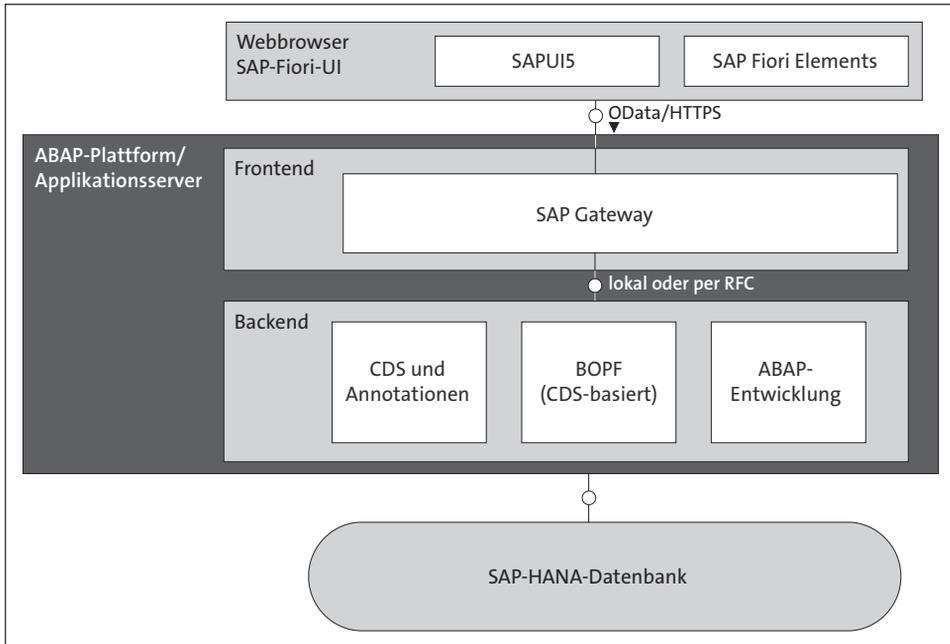


Abbildung 1.4 ABAP-Programmiermodell für SAP Fiori

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

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

Verarbeitungsphasen