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 seine Entstehung, seine qualitativen Eigenschaften, die Architektur und grundlegende Elemente des Programmiermodells 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 ab, 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 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 Model 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 Model 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 Model 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 Model 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 zustandlosen 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 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, »Der REST-Architekturstil«.

Standard-Architektur von Geschäftsanwendungen

Das ABAP RESTful Application Programming Model definiert eine Standard-Architektur von Geschäftsanwendungen und setzt dabei auch Leitplanken für die Entwicklung, um diese Standard-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 Standard-Architektur 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, »Entwicklungsobjekte des ABAP RESTful Application Programming Model«, ausführlicher ein.
- **Unterstützung technischer Aspekte**
  Das Programmiermodell berücksichtigt standardmäßig verschiedene technische Aspekte, die typisch für Unternehmensanwendungen sind. Dazu gehören z. B. die Realisierung von Standard-Operationen, 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, »Entwicklungseffizienz«, 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-Push-down* auf die SAP-HANA-Datenbank aus dem ABAP-Entwicklungskontext heraus möglich. Berechnungen können somit direkt auf der SAP-HANA-Datenbank ausgeführt werden.
- **Cloud Readyness**
  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 die 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 Identifier* (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 bei 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-Request 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. 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

Request 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 Server-Bausteinen erfolgt zustandslos, das heißt, jede Anfrage des Clients an den Server erfolgt unabhängig von zuvor durchgeführten Anfragen. Die Client-Anfrage muss dem-

nach 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.

[zB]

Vollständige Repräsentation einer Ressource

Eine per HTTP formulierte REST-Anfrage 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 der Anfrage enthalten sein, damit der REST-konforme Server diese 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 Server-Repräsentation 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 Server-Komponenten 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 Server-Bausteine 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*).

[zB]

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 einer REST-basierten 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 Standard-Operationen 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-Kon-

sortium 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.

https://hostname/sap/opu/odata/DMO/UI_TRAVEL_PROC_M_O2/$metadata

```
</edmx:Reference>
−<edmx:DataServices m:DataServiceVersion="2.0">
  −<Schema Namespace="cds_xdmoxui_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. Methodensteuerg" sap:creatable="false"
        sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
        <Property Name="Update_mc" Type="Edm.Boolean" sap:label="Dyn. Methodensteuerg" 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="AgencyNa
        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"/>
```

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 Standard-Operationen 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.

https://hostname/sap/opu/odata/DMO/UI_TRAVEL_PROC_M_O2/

```
−<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

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-Standard-Operationen 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. 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 Personen-Instanz 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, Abfrage-Optionen 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 Service-Metadatendokument 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 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* aus SAP S/4HANA Cloud an.

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 Standard-Software 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-Push-down in die SAP-HANA-Datenbank. Berechnungen können 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 realisieren, 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. Sie 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 Neuerungen des ABAP RESTful Application Programming Model 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 desProgrammiermodells ein und stellen Vorgängertechnologien vor, die einen wesentlichen Einfluss auf dessen Bestandteile haben.

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 Model. 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.

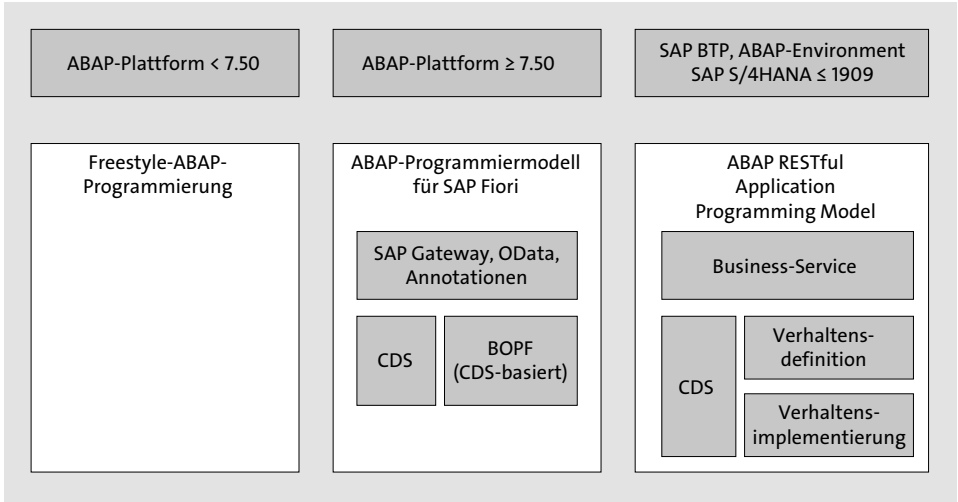

Abbildung 1.3 Evolution ABAP-basierter Programmiermodelle (Quelle: SAP)

Klassische Anwendungsentwicklung mit ABAP

Ursprünglich aus SAP R/3 hervorgegangen, ist der *SAP NetWeaver Application Server 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 Application Server 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 Form-Routinen, Funktionsbausteine oder Klassen und Interfaces zur Verfügung, um Programmlogik zu kapseln. Wir empfehlen Ihnen, Klassen bzw. Interfaces als Mittel der Objektorientierung zur Modularisierung einer 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 Manager 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 Application Server ABAP.

ABAP-Sprachkonstrukte ab Version 7.40

Mit SAP NetWeaver Application Server ABAP 7.40 ist eine wichtige neue Funktionalität in den ABAP-Sprachumfang eingegangen, die zur Anwendungsentwicklung mit dem ABAP RESTful Application Programming Model besonders hilfreich ist. Diese Änderungen umfassen *Konstruktor-* und *Tabellenausdrücke* (ab 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 Model erleichtern Sie die Verhaltensimplementierung und den Umgang mit der 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.

Konzepte von BOPF

Einige zentrale Elemente von BOPF sind:

- **Geschäftsobjekte**
  Sie werden auch *Business-Objekte* genannt und 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 zur Multiinstanz-fähigen Implementierung von Geschäftsobjekten und deren internen 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 Geschäftsobjekt-Funktionalität.

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 Application Server 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 Version 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-Push-down), 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 jeweiligen 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.

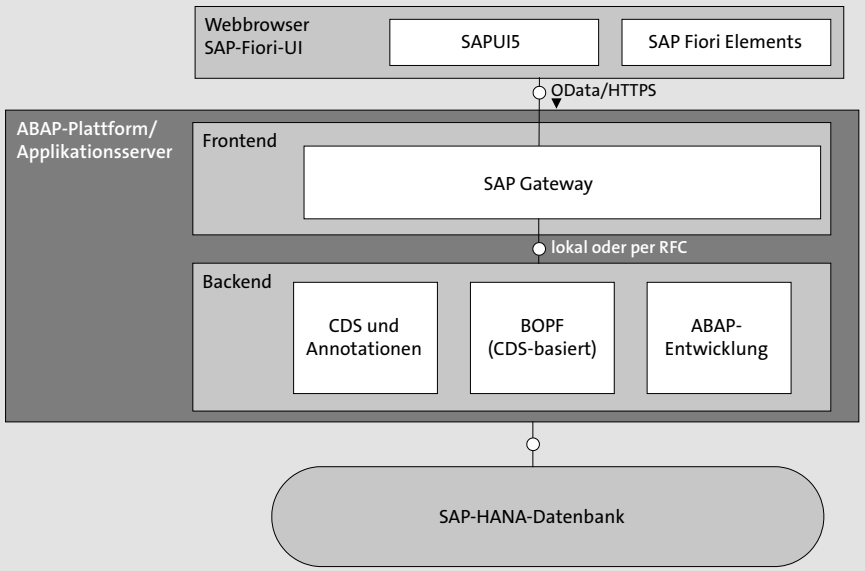

Abbildung 1.4 ABAP-Programmiermodell für SAP Fiori

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 Model

In diesem Abschnitt gehen wir ausführlicher auf die wesentlichen architekturrelevanten Konzepte des ABAP RESTful Application Programming Model 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:

1. Während der *Interaktionsphase* werden Operationen auf einer Geschäftsobjekt-Instanz durchgeführt, z. B. eine Position angelegt oder geändert. Als Ergebnis dieser Operationen werden die Instanzen der jeweiligen CDS-Entitäten im *Transaktionspuffer* vorgehalten.
2. Die anschließende *Speichersequenz* wird durch einen Commit angestoßen. Dabei wird der Zustand des Transaktionspuffers persistent auf die Datenbank geschrieben. Der Zustand des Transaktionspuffers wird nicht über mehrere Requests hinweg vorgehalten, da dies ein wesentliches REST-Prinzip verletzen würde (siehe Abschnitt 1.1.2, »Der REST-Architekturstil«). Die im Transaktionspuffer vorgehaltenen Instanzen von CDS-Entitäten, die angelegt, geändert oder gelöscht wurden (`modify`), können Sie somit als *Logical Unit of Work* (LUW) innerhalb des SAP-Systems betrachten. Im Rahmen der Speichersequenz wird die LUW innerhalb einer Datenbank-LUW persistent auf die Datenbank geschrieben.

Draft-Handling

Das Draft-Handling im ABAP RESTful Application Programming Model erlaubt es temporär, den Zustand des Transaktionspuffers mit inkonsistenten Anwendungsdaten persistent auf der Datenbank abzulegen. Somit haben Sie die Möglichkeit, die Interaktionsphase auf mehrere eigenständige Requests bzw. Benutzersitzungen zu verteilen, ohne die REST-Prinzipien der zustandslosen Kommunikation zu verletzen. Anwenderinnen und Anwender können ihre Arbeit so unabhängig vom Endgerät zu einem späteren Zeitpunkt fortsetzen. Die Funktionalität des Draft-Handlings wird vollständig von der RAP-Runtime umgesetzt.

Geht das ABAP RESTful Application Programming Model auch stateful?

Das ABAP RESTful Application Programming Model schließt die Realisierung einer zustandsbehafteten Anwendung grundsätzlich nicht aus. Das RAP-Transaktionsmodell umfasst technisch sogar einen Bereich (den Transaktionspuffer), in dem der Anwendungszustand serverseitig vorgehalten werden kann. In diesem Buch werden wir uns jedoch auf REST-basierte Anwendungen konzentrieren.

1.2.2 Implementierungstypen

Managed und unmanaged

Im ABAP RESTful Application Programming Model gibt es das Konzept des *Implementierungstyps*, mit dem Sie festlegen, wer die Implementierung von lesender Funktionalität (Query) und transaktionaler bzw. schreibender Funktionalität (Verhalten) zur Verfügung stellt. Es gibt zwei Implementierungstypen: verwaltet (`managed`) und nicht verwaltet (`unmanaged`). Im verwalteten Fall stellt das Programmiermodell die gewünschte Funktionalität bereit, im nicht verwalteten Fall muss das die Anwendung selbst tun.

Realisierung eines Geschäftsobjekts

Die technische Realisierung lesender und schreibender Zugriffe bzw. des Verhaltens eines Geschäftsobjekts erfolgt durch den *BO-Provider*. Bei der Realisierung des Geschäftsobjekts unterscheiden wir zwischen der Verwendung des verwalteten (*Managed Scenario*) und des nicht verwalteten (*Unmanaged Scenario*):

- **Managed Scenario**
  Im Managed Scenario nutzen Sie eine vom ABAP RESTful Application Programming Model zur Verfügung gestellte, bereits fertige Implementierung des Geschäftsobjekts, den *Managed BO-Provider*. Dieser kümmert sich um die Realisierung der Standard-Operationen zur Anlage (`create`) sowie zum Lesen (`read`), Ändern (`update`) und Löschen (`delete`) von Instanzen der jeweiligen CDS-Entität während der Interaktionsphase und der Speichersequenz. Damit einher geht auch das Handling des Transaktionspuffers. Für die genannten Standard-Operationen wird allgemein die Abkürzung CRUD verwendet.

  Bei der Nutzung des Managed Scenario steht Ihnen ohne eigenen Programmieraufwand ein bereits lauffähiges Geschäftsobjekt zur Verfügung, das die CRUD-Operationen unterstützt. Optional können Sie der Speichersequenz weitere Logik hinzufügen (*Additional Save*) oder selbst implementieren (*Unmanaged Save*).
- **Unmanaged Scenario**
  Im Unmanaged Scenario können Sie die Standard-Funktionalität eines Geschäftsobjekts selbst implementieren. Dies betrifft sowohl die Interaktionsphase mit dem Transaktionspuffer als auch die Speichersequenz. Sie haben somit die Möglichkeit, die API einer Bestandsanwendung zu integrieren und mit RAP-Mitteln zu verschalen. Dieses Szenario wird daher häufig für die sogenannte *Brownfield-Entwicklung* angewendet.

Implementierung eigener Geschäftslogik

Neben den Anwendungsfällen, die durch diese Implementierungstypen umgesetzt werden können, gibt es immer wieder ganz spezifische fachliche Anforderungen, z. B. die Validierung einer bestimmten Datenkonstellation oder die Berechnung bestimmter Werte einer Geschäftsobjekt-Instanz. In

diesen Fällen kann der BO-Provider keine Implementierung bereitstellen. Sie haben jedoch sowohl im Managed als auch im Unmanaged Scenario die Möglichkeit, die jeweilige Geschäftslogik selbst zu implementieren. Das zeigen wir beispielsweise in Abschnitt 9.6, »Anreicherung um eine Ermittlung«, oder Abschnitt 9.7, »Anreicherung um eine Validierung«.

Querys

Lesende Zugriffe werden im ABAP RESTful Application Programming Model *Querys* genannt. Sie können ganz für sich stehend ohne transaktionales Verhalten oder zugehörige schreibende Zugriffe realisiert werden. Eine verwaltete Implementierung eines solchen Lesezugriffs heißt *Managed Query*, eine nicht verwaltete Implementierung wird *Unmanaged Query* genannt. Über die Managed Query stellt das Programmiermodell die Standard-Funktionen für das Lesen von Geschäftsdaten aus der zugrunde liegenden Datenbank zur Verfügung. Dazu gehören beispielsweise auch Funktionen für die Sortierung, Filterung oder Aggregation der gelesenen Daten. Wenn Sie eine Unmanaged Query implementieren, können Sie diese Funktionen selbst realisieren und haben somit die Möglichkeit, beliebige Datenquellen in das Programmiermodell einzubinden (z. B. per Remote-Aufruf von APIs) oder Unterschiede zwischen dem Datenmodell der Datenquelle und dem des Geschäftsobjekt im ABAP RESTful Application Programming Model zu überbrücken.

1.2.3 Entity Manipulation Language

API für den Zugriff auf Geschäftsobjekte

Die *Entity Manipulation Language* (EML) ist eine standardisierte, typsichere API, die dazu dient, auf Daten und Funktionalität eines Geschäftsobjekts zu zugreifen. Sie ist fester Bestandteil des ABAP-Sprachumfangs. Mit EML können Sie schreibend auf Instanzen eines Geschäftsobjekts zugreifen (mit der ABAP-Anweisung `MODIFY ENTITIES`), aber auch einzelne Instanzen aus dem Transaktionspuffer lesen (mit `READ ENTITIES`).

Anwendungsfälle

EML spielt in den folgenden Anwendungsfällen eine zentrale Rolle innerhalb des ABAP RESTful Application Programming Model:

- **Implementierung von Verhalten eines Geschäftsobjekts**
  Sie können EML-Anweisungen nutzen, um das Verhalten eines Geschäftsobjekts in ABAP zu implementieren. In diesem Fall übernimmt Ihre Anwendung die Rolle des BO-Providers.
- **Ausführen von Operationen auf einem Geschäftsobjekt**
  Sie benötigen EML-Anweisungen, wenn Sie eine ABAP-Anwendung programmieren, die als Konsument auf die Funktionalität eines Geschäftsobjekts zugreift, also die Rolle des *BO-Consumers* einnimmt. Auch im

Rahmen von Unit-Tests ist der Einsatz von EML möglich, um die Funktionalität eines Geschäftsobjekts in Ihrer Anwendung zu prüfen.

Eigenes Geschäftsobjekt konsumieren

Wenn Sie EML innerhalb des BO-Providers verwenden, beispielsweise um Daten für eine Validierung zu lesen, verwenden Sie die EML auch in der Rolle des BO-Consumers. Sie lesen damit Daten zum Geschäftsobjekt aus dem Transaktionspuffer, um diese zu validieren, und umgehen dabei beispielsweise Berechtigungsprüfungen, da der Zugriff `IN LOCAL MODE` erfolgt. Sie konsumieren damit die lesende oder schreibende Funktionalität Ihres eigenen Geschäftsobjekts. Neben lesenden und schreibenden Operationen können Sie mit EML auch Transaktionsgrenzen definieren (`COMMIT`- und `ROLLBACK`-Verhalten) und die Speichersequenz anstoßen.

1.2.4 Technischer Kontext einer RAP-Anwendung und RAP-Laufzeitumgebung

Zugriff auf RAP-Anwendungen

In Abbildung 1.5 sehen Sie, in welchen technischen Kontext eine mit dem ABAP RESTful Application Programming Model erstellte Anwendung eingebettet ist. Sie gewinnen hier eine Vorstellung davon, auf welchen technischen Wegen und mit welchen Protokollen ein Zugriff auf die Daten und Dienste einer RAP-Anwendung möglich ist. Eine RAP-Anwendung verwendet eine SAP-HANA-Datenbank zur Persistierung von Anwendungsdaten. Durch CDS Custom Entities können Sie mit einer eigens in ABAP programmierten Query ein externes System oder eine andere Datenquelle durch synchrone Kommunikation anbinden.

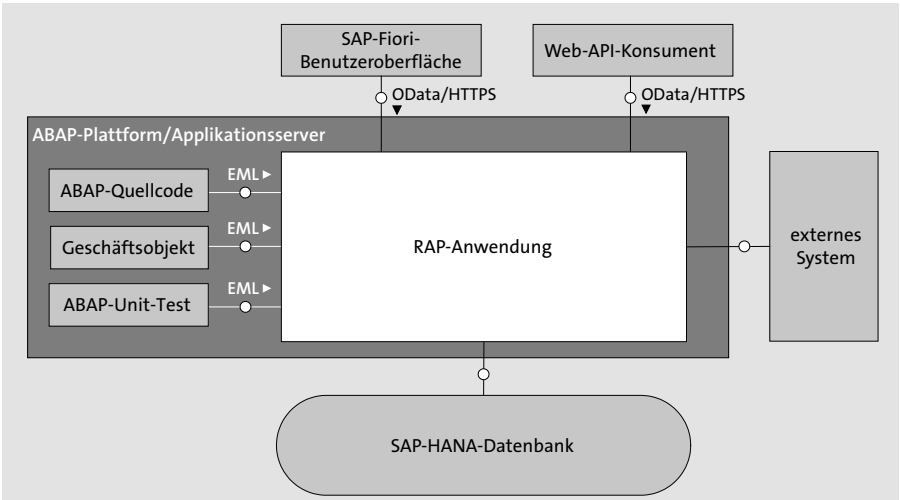

Abbildung 1.5 Technischer Kontext einer RAP-Anwendung

Auch wenn Sie die konkreten technischen Komponenten des ABAP RESTful Application Programming Model nicht im Detail verstehen müssen, damit die RAP-Runtime RAP-Requests verarbeitet, ist es hilfreich, die wesentlichen an diesem Vorgang beteiligten Bausteine zu kennen. Es verbessert nicht nur das generelle Verständnis des Programmiermodells, sondern kann auch bei der Fehleranalyse sehr nützlich sein.

Verarbeitung eines RAP-Request

In Abbildung 1.6 sehen Sie die technischen Komponenten, die an der Verarbeitung eines RAP-Request beteiligt sind. Ein solcher RAP-Request kann beispielsweise die Anlage einer Bestellung mit der CRUD-Operation `create` sein. Ein Request kann von einer externen Anwendung (beispielsweise einer SAP-Fiori-Oberfläche oder einem Web-API-Konsumenten) ausgehen, wobei das OData-Protokoll verwendet wird, oder als lokaler Aufruf (das heißt systemintern) über die EML stattfinden.

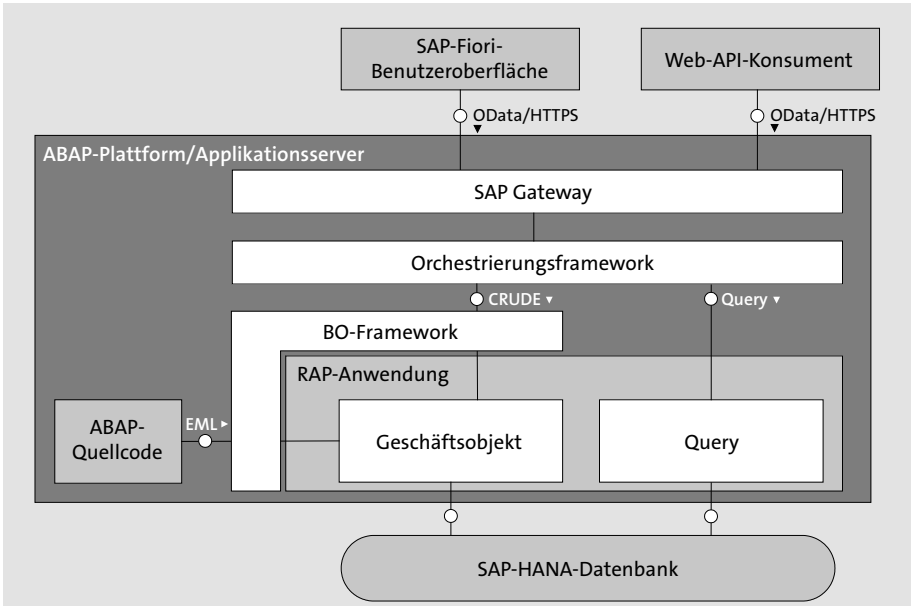

Abbildung 1.6 Komponenten der RAP-Laufzeitumgebung (Quelle: SAP)

Im Folgenden beschreiben wir die einzelnen Komponenten:

- **SAP Gateway**
  SAP Gateway implementiert das OData-Protokoll und dient externen Anwendungen als Eintrittspunkt in die jeweilige Anwendung. Dazu nimmt SAP Gateway den OData-Request entgegen und leitet diesen an das Orchestrierungsframework zur generischen Verarbeitung weiter.

- **Orchestrierungsframework**
  Das *Orchestrierungsframework* bildet den OData-Request auf einen RAP-Request ab und wertet diesen generisch aus. Das Orchestrierungsframework ist auch als *SADL-Framework* (Service Adaptation and Description Language) bekannt. Ein lesender Zugriff wird an die passende Query-Implementierung weitergeleitet, ein schreibender und somit transaktionaler Zugriff wird über das *Business Object Framework* (BO-Framework) verarbeitet. Das Orchestrierungsframework bildet somit die transaktionale Klammer um den RAP-Request und ist beispielsweise für das Anfordern von Sperren, das ETag-Handling und das Commit- und Rollback-Verhalten zuständig. Bei lesenden und schreibenden Zugriffen ermittelt das Orchestrierungsframework, ob die Methoden einer verwalteten oder nicht verwalteten Implementierung aufgerufen werden müssen.
- **Business Object Framework**
  Das BO-Framework steuert die Verarbeitung von Operationen auf einem Geschäftsobjekt. Es sorgt dafür, dass während der Interaktionsphase und Speichersequenz die vorgesehenen Methoden der BO-Provider-API in der verwalteten oder nicht verwalteten Verhaltensimplementierung aufgerufen werden. Während bei einem OData-Request das Orchestrierungsframework die Steuerung von Interaktionsphase und Speichersequenz übernimmt, haben Sie über Ihren ABAP- und den EML-Code die Möglichkeit, selbst zu steuern, welche Operationen innerhalb der Interaktionsphase ablaufen und wann die Speichersequenz angestoßen wird.

Verwendung von SAP Gateway erfolgt im Hintergrund

Das ABAP RESTful Application Programming Model verwendet standardmäßig OData, um den Zugriff auf die Funktionalität der Anwendung zu exponieren, und nutzt dafür SAP Gateway als Framework und Laufzeitumgebung. Für Sie kommt SAP Gateway bei der Entwicklung von RAP-Anwendungen allerdings gar nicht zum Vorschein. Das Programmiermodell generiert die Gateway-Artefakte im Hintergrund, damit diese zur Laufzeit einen OData-Request für das Programmiermodell entgegennehmen können.

Das bedeutet auch, dass Sie mit dem ABAP RESTful Application Programming Model *nicht* auf die Möglichkeiten der Gateway-Entwicklung (das heißt die Transaktion SEGW sowie die Data- und Model-Provider-Implementierung) zugreifen können, auch wenn diese auf dem jeweiligen ABAP-Applikationsserver technisch zur Verfügung stehen. Ein solcher direkter Zugriff würde das Programmiermodell von einer bestimmten Technologie abhängig machen, was den Architekturzielen des Programmiermodells entgegensteht.

1.3 Entwicklungsobjekte des ABAP RESTful Application Programming Model

In diesem Abschnitt vermitteln wir Ihnen einen Überblick über die wesentlichen Artefakte des ABAP RESTful Application Programming Model. Ein Programmiermodell stellt u. a. Entwicklungsobjekte, Sprachen und APIs zur Verfügung und legt deren Zusammenspiel fest, um eine bestimmte Geschäftsfunktionalität zu realisieren.

1.3.1 Datenmodellierung mit Core Data Services

Logisches Datenmodell

Jede Softwareanwendung basiert auf einem Daten- oder Objektmodell, das seine *Entitäten* repräsentiert, diese anhand von Attributen beschreibt und die Beziehungen (Assoziationen) zu anderen Entitäten abbildet. Eine Entität bildet ein Konzept aus der Geschäftswelt ab. Geschäftsprozesse werden in der Regel mit mehreren Entitäten in einer Anwendung abgebildet. Entitäten werden auch *Geschäftsobjekte* genannt. Der Materialstamm ist ein Beispiel für ein solches Geschäftsobjekt, das zu den Stammdaten gehört. Ein Kundenauftrag ist ein weiteres typisches Beispiel für ein Geschäftsobjekt. Es gehört zu den Bewegungsdaten.

[zB]

Datenmodell des Geschäftsobjekts »Kundenauftrag«

Das Geschäftsobjekt »Kundenauftrag« besteht beispielsweise aus den Entitäten »Kundenauftragskopf« und »Auftragsposition«. Beide zusammen bilden die Struktur des Geschäftsobjekts. Die Auftragsposition ist dabei eine vom Kundenauftragskopf abhängige Entität. Sie weist außerdem eine Assoziation zu einem anderen Geschäftsobjekt auf, dem »Material«, das den im Kundenauftrag bestellten Artikel abbildet.

Core Data Services

Um das logische Datenmodell einer Anwendung zu definieren, verwenden Sie CDS. Für jede Entität legen Sie dazu eine CDS-Entität im System an. Typischerweise werden CDS-Entitäten auf Basis von transparenten Tabellen erstellt, CDS Custom Entities können allerdings auch unabhängig von einer solchen Datenquelle definiert werden.

CDS-Wurzel-Entität

Das Konzept des Geschäftsobjekts mit seinen abhängigen CDS-Entitäten ist für das ABAP RESTful Application Programming Model besonders wichtig. Ein RAP-Geschäfts- bzw. Business-Objekt wird in seiner Gesamtheit, also mit allen abhängigen CDS-Entitäten, wiederum als spezielle CDS-Entität abgebildet, die *CDS-Wurzel-Entität* genannt wird (siehe Abbildung 1.7).

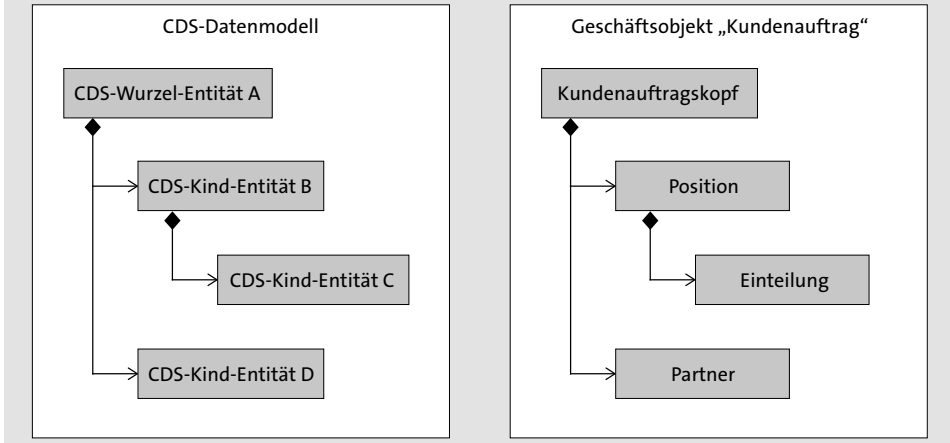

Abbildung 1.7 CDS-Datenmodell mit Beispiel »Kundenauftrag«

1.3.2 Verhaltensdefinition

Verhalten

Das im vorangehenden Abschnitt beschriebene CDS-Datenmodell ist ein erster wichtiger Bestandteil eines Geschäftsobjekts im ABAP RESTful Application Programming Model. Ein weiterer Bestandteil ist das *Verhalten* eines Geschäftsobjekts. Das Programmiermodell fasst schreibende Operationen, damit einhergehende transaktionale Eigenschaften und Geschäftslogik unter diesem Begriff zusammen. Das Verhalten kann also Standard-Operationen (z. B. die Anlage oder Änderung eines Objekts), interne Geschäftslogik (z. B. Prüfungen oder Berechnungen) oder weiteres transaktionales Verhalten (z. B. Sperrverhalten, Berechtigungsprüfungen etc.) umfassen.

Behavior Definition Language

Wenn Sie das Verhalten eines Geschäftsobjekts definieren möchten, legen Sie dazu eine *Verhaltensdefinition* an. Dabei handelt es sich um ein im ABAP RESTful Application Programming Model neu eingeführtes Entwicklungsobjekt. Die Anlage erfolgt mit Bezug auf die CDS-Wurzel-Entität. Eine Verhaltensdefinition gilt somit auch für alle der Wurzel-Entität untergeordneten CDS-Entitäten.

Das gewünschte Verhalten deklarieren Sie über die *Behavior Definition Language* (BDL). Mit den BDL-Schlüsselwörtern `create`, `update` oder `delete` legen Sie z. B. fest, dass Ihre CDS-Entität die entsprechende Standard-Operation zur Anlage, Änderung oder Löschung einer Instanz dieses Geschäftsobjekts unterstützt.

Implementierungstyp festlegen

In der Verhaltensdefinition legen Sie weiterhin fest, welchen Implementierungstyp Sie für die Realisierung der Funktionalität des Geschäftsobjekts

wählen. Mit dem Implementierungstyp verwaltet (`managed`), greifen Sie auf eine Standard-Implementierung von Grundfunktionalitäten zurück. Der Implementierungstyp nicht verwaltet (`unmanaged`) bedeutet, dass Sie selbst die Implementierung des Geschäftsobjekts in ABAP vornehmen. Auf diese Weise können Sie etwa Bestandscoding mit dem ABAP RESTful Application Programming Model verschalen.

1.3.3 Verhaltensimplementierung

Behavior Pool

Haben Sie mit der BDL das Verhalten eines Geschäftsobjekts deklariert, benötigen Sie noch die Programmlogik, die dieses Verhalten zur Laufzeit realisiert. Je nachdem, für welchen Implementierungstyp bzw. welches Szenario Sie sich bei der Anlage der Verhaltensdefinition entschieden haben, kann der Umfang der benötigen Implementierung ganz unterschiedlich ausfallen. Die *Verhaltensimplementierung*, auch *Behavior Pool* genannt, stellen Sie in ABAP bereit.

Business Object Provider API

Dazu definieren Sie eine ABAP-Klasse, die mit der *Business Object Provider API* eine Schnittstelle besitzt, um bestimmte Fachlogik, wie Validierungen, Berechnungen, transaktionales Verhalten oder Speicherlogik, zu implementieren. Konkret können Sie die folgenden Operationen implementieren:

- **Standard-Operationen**
  Zu den Standard-Operationen gehören das Anlegen, Lesen, Ändern und Löschen von Geschäftsobjekten. Um diese Operationen zu realisieren, müssen sie im Kontext der Interaktionsphase, des Transaktionspuffers und der Speichersequenz geeignet implementiert werden. Beispielsweise erfordert die Anlage-Operation eine Ablage der Instanz im Transaktionspuffer. Innerhalb der Speichersequenz muss diese im Transaktionspuffer zur Anlage vorgehaltene Instanz dann persistent auf der Datenbank gesichert werden. Sie können entweder eine vom BO-Framework verwaltete Implementierung wählen (Managed BO-Provider) oder die Implementierung vollständig selbst vornehmen (Unmanaged BO-Provider).
- **Spezifische Operationen**
  Zur programmtechnischen Realisierung von Aktionen oder Funktionen gibt es einige spezifische Operationen. Diese benötigen immer eine eigene Verhaltensimplementierung.
- **Interne Geschäftslogik**
  Die programmtechnische Realisierung von Validierungen oder Ermittlungen, die nur innerhalb des Geschäftsobjekts sichtbar sind, wird als interne Geschäftslogik bezeichnet. Hierfür benötigen Sie ebenfalls immer eine eigene Verhaltensimplementierung.
- **Transaktionales Verhalten**
  Auch zur programmtechnischen Realisierung von Berechtigungsprüfungen, Nummernvergaben, Sperren etc. benötigen Sie in der Regel eine eigene Verhaltensimplementierung. Bei der Nummernvergabe können Sie aber auch auf eine fertige Implementierung der Geschäftsobjekt-Runtime verweisen (*Managed Numbering* mit GUIDs). GUIDs sind global eindeutige Identifikatoren, die zur Vergabe von Schlüsselwerten verwendet werden können.

1.3.4 Projektionsschicht

Die Entwicklungsobjekte der *Projektionsschicht* im ABAP RESTful Application Programming Model verwenden Sie, um das CDS-Datenmodell oder das Verhalten des Kern-Geschäftsobjekts auf einen bestimmten Anwendungsfall zuzuschneiden. Generell schränken Sie damit die verfügbaren Attribute und Operationen ein.

Entwicklungsobjekte

Innerhalb der Projektionsschicht gibt es die folgenden Entwicklungsobjekte:

- **CDS Projection View**
  Sie verwenden einen *CDS Projection View*, um relevante Attribute und Assoziationen zu deklarieren und diese gegebenenfalls um Annotationen anzureichern. Diese Annotationen können z. B. zur Definition des UI, einer Suche oder einer Wertehilfe dienen. CDS Projection Views sind vergleichbar mit den *Consumption Views* aus dem ABAP-Programmiermodell für SAP Fiori.
- **Verhaltensprojektion**
  Die *Verhaltensprojektion* ist ein spezieller Implementierungstyp (`projection`) der Verhaltensdefinition. Er schränkt die in der Verhaltensdefinition deklarierten Operationen und Verhalten nachträglich auf eine Auswahl ein. Zusätzliche Operationen können Sie auch in ihrer Verarbeitung im ABAP-Coding ergänzen, ohne die Funktionalität des Kern-Geschäftsobjekts ändern zu müssen. Dazu ist eine zusätzliche Verhaltensimplementierung auf Basis der Verhaltensprojektion notwendig.

Sichtbarkeit deklarieren

Die Projektionsschicht dient auch zur Definition der Sichtbarkeit in Ihrer Anwendung, da Sie darin entscheiden können, welche Daten und welches Verhalten Sie für Konsumenten exponieren möchten. Sie bildet somit die Basis, um über Business-Services Remote-APIs oder APIs für rollenbasierte SAP-Fiori-Benutzeroberflächen zur Verfügung zu stellen.

1.3.5 Business-Services

Entwicklungsobjekte

Die Schicht der *Business-Services* dient dazu, RAP-Geschäftsobjekte mit ihren Daten und Verhalten als Remote-API für Konsumenten zu exponieren. In dieser Schicht stehen Ihnen die folgenden Entwicklungsobjekte zur Verfügung:

- **Servicedefinition**
  Die *Servicedefinition* enthält die zu exponierenden CDS-Entitäten aus dem Datenmodell der Anwendung. Anhand der CDS-Entitäten kann ermittelt werden, welche Daten und welches Verhalten als Service verfügbar gemacht werden. Das kann auf Basis von CDS Projection Views (und Verhaltensprojektionen) geschehen oder, in einfachen Fällen, direkt auf Basis der CDS-Entitäten des Kern-Geschäftsobjekts.
- **Service-Binding**
  Mit dem auf einer Servicedefinition aufsetzenden *Service-Binding* definieren Sie das konkrete technische Protokoll, über das die CDS-Entitäten der Servicedefinition exponiert werden sollen. Das kann z. B. OData in der Version 2 (OData V2) oder der Version 4 (OData V4) sein. Bei diesen Protokollen wird jeweils zwischen einer Variante für UI- oder Web-API-Konsumenten unterschieden. Das Service-Bindung unterstützt die Versionierung der Schnittstelle. Außerdem können hier Standard-Berechtigungen vergeben werden.

1.3.6 Zusammenspiel der Artefakte

In Abbildung 1.8 sehen Sie, wie die in den vorangehenden Abschnitten vorgestellten RAP-Artefakte zusammenspielen und voneinander abhängen. Es sind hier nahezu alle Elemente dargestellt. Lediglich die Möglichkeit, eine eigene Query für lesende Zugriffe (Unmanaged Query) zu programmieren, ist nicht abgebildet.

Optionale Elemente

Beachten Sie, dass die Projektionsschicht grundsätzlich optional ist. Es ist vom jeweiligen Anwendungsfall abhängig, ob Sie diese Elemente benötigen oder nicht.

UI-Entwicklung ist nicht durch das ABAP RESTful Application Programming Model abgedeckt

Wie Sie die Benutzeroberfläche für eine RAP-Anwendung erstellen, ist explizit *nicht* Bestandteil des Programmiermodells. Da Sie mit dem ABAP RESTful Application Programming Model allerdings auf einfache Weise OData-basierte Schnittstellen als APIs exponieren können, bietet es sich an,

SAPUI5 als UI-Framework zu verwenden. Das Programmiermodell schafft außerdem Raum, um CDS-Entitäten um UI-Annotationen anzureichern. Mit diesen Annotationen können Sie die Darstellung und Steuerung von Benutzeroberflächen beeinflussen, die auf SAP Fiori Elements basieren. Wie Sie mit diesen Annotationen arbeiten, erklären wir in Kapitel 7, »Anwendungsoberflächen und SAP Fiori Elements«.

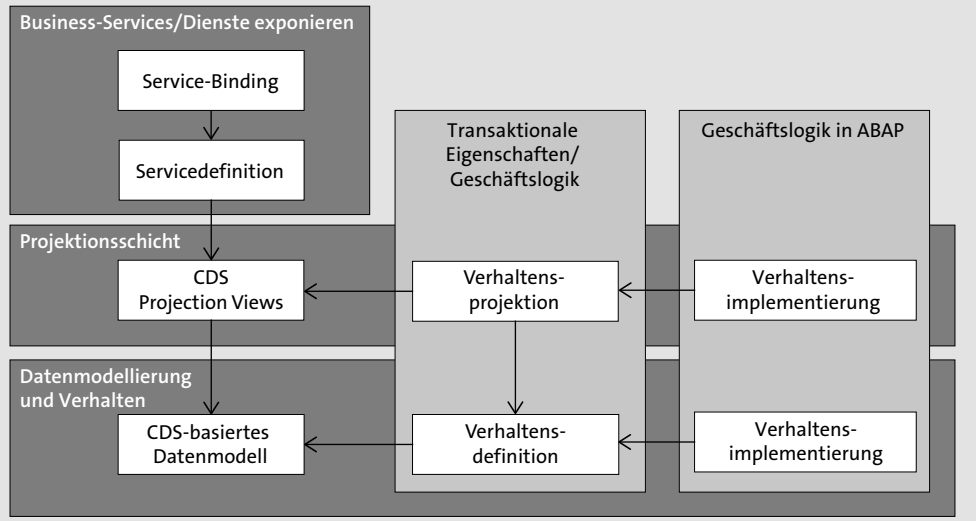

Abbildung 1.8 Artefakte des ABAP RESTful Application Programming Model

1.4 ABAP Development Tools als Entwicklungswerkzeug

Die *ABAP Development Tools* (ADT) sind das Entwicklungswerkzeug der Wahl, wenn es um moderne ABAP-Programmierung und die Erhöhung der Entwicklerproduktivität geht. Das gilt auch, wenn Sie RAP-Anwendungen bauen. Die ADT unterstützen das bekannte serverbasierte Entwicklungsmodell mit einem Anschluss an das Transportwesen des SAP-Systems (*Change and Transport System*, kurz CTS). Technisch ist diese Entwicklungsumgebung jedoch vollkommen unabhängig von den auf einem ABAP-System installierten Geschäftsanwendungen, etwa Anwendungen aus der SAP Business Suite oder SAP S/4HANA oder SAP S/4HANA Cloud. Sie kann ab SAP NetWeaver 7.31 SP04 eingesetzt werden.

Wesentliche Funktionen

Einige wesentliche Funktionen der ADT sind das komfortable Editieren von Quellcode, ein integrierter Code-Assistent, eine Autovervollständigung,

Quick Fixes sowie die Navigation zwischen Entwicklungsobjekten und deren parallele Darstellung. Außerdem wird das Refactoring unterstützt. Die ADT werden mit einer stark anpassbaren Benutzeroberfläche bereitgestellt.

Mehrsystem-fähigkeit

Die ADT basieren auf Eclipse und werden lokal auf Ihrem Rechner installiert. Eine ADT-Installation können Sie mit mehreren ABAP-Systemen verschiedener Releasestände verbinden. Das können On-Pemise-Systeme, Cloud-basierte ABAP-Systeme, die auf der SAP Business Technology Platform (SAP BTP) bereitgestellt werden, oder SAP-S/4HANA-Cloud-Systeme sein. Der Funktionsumfang der Entwicklungsumgebung ist dabei immer abhängig vom Basis-Releasestand des verbundenen Systems. Abbildung 1.9 stellt den Systemkontext dar, in dem sich die ADT bewegen.

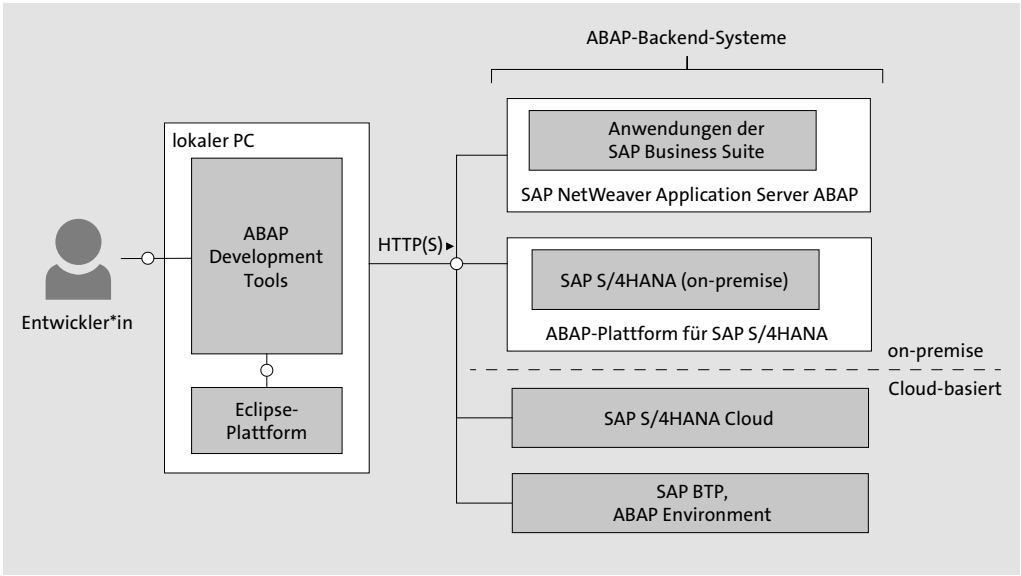

Abbildung 1.9 Systemkontext der ABAP Development Tools

Wenn Sie RAP-Anwendungen bauen, verwenden Sie *ausschließlich* die ADT. Alle notwendigen Werkzeuge zur Entwicklung sowie für Tests und Analysen stehen in dieser Umgebung zur Verfügung. Die ADT werden sowohl zur RAP-Entwicklung für On-Premise- als auch für Cloud-basierte Systeme eingesetzt. Die SAP-GUI-basierten Transaktionen der ABAP Workbench spielen in Zusammenhang mit dem ABAP RESTful Application Programming Model keine Rolle mehr.

Entwicklungswerkzeug für die UI-Entwicklung

Für die Entwicklung von SAP-Fiori-Benutzeroberflächen können Sie die SAP Fiori Tools für *Visual Studio Code* oder das *SAP Business Application Studio* verwenden. Details zu diesen Entwicklungsumgebungen finden Sie in Kapitel 7, »Anwendungsoberflächen und SAP Fiori Elements«.

1.5 Qualitative Eigenschaften des ABAP RESTful Application Programming Model

Nicht-funktionale Anforderungen

In der Softwarearchitektur spielen nicht-funktionale Anforderungen und die qualitativen Eigenschaften der Software eine bedeutsame Rolle. Dazu gehören z. B. Performance, Usability, Änderbarkeit oder die Flexibilität und Anpassbarkeit von Software.

Eine Softwarearchitektur sollte die an sie gestellten qualitativen Anforderungen berücksichtigen. Diese bestimmen die langfristige Nutzbarkeit und Langlebigkeit des Softwareprodukts. So bringt auch die Architektur des ABAP RESTful Application Programming Model bestimmte qualitative Eigenschaften sich, von denen wir einige in diesem Abschnitt näher beleuchten. Für SAP-Entwicklungsteams ergeben sich daraus ebenso wie für SAP-Partner und -Kunden wichtige Konsequenzen, die mit dem Einsatz des Programmiermodells in Verbindung stehen.

In diesem Abschnitt betrachten wir die folgenden, ausgesuchten Qualitätsmerkmale näher:

- Evolutionsfähigkeit
- Entwicklungseffizienz
- Testbarkeit
- Trennung zwischen Fachlichkeit und Technik

1.5.1 Evolutionsfähigkeit

Anpassbarkeit an technologische Trends

Seit jeher strebt die Softwareentwicklung an, evolutionsfähige Software zu erstellen, also Software, die langfristig änderbar ist (und bleibt) und auch neue technische Trends und Entwicklungen berücksichtigen kann. Technische Trends betreffen in der Regel die Infrastruktur einer Geschäftsanwendung, z. B. eine neue UI-Technologie oder eine Art der Datenspeicherung, weniger die Fachlogik der Anwendung. Eine Software ist dann gut an Trends anpassbar, wenn ihr fachlicher Kern nicht mit technischem Coding für die

Infrastruktur vermischt wurde. Es ist auch hilfreich, wenn die Software eine fachlich motivierte API als Zugriffspunkt für Konsumenten anbietet und dadurch in verschiedenen technischen Kontexten verwendbar ist. Gleichermaßen sollte auch der fachliche Kern einer Anwendung gut an zusätzliche Geschäftsanforderungen anpassbar sein. Dies ist der Fall, wenn kein technisches Coding darin enthalten ist.

Deklarativer Ansatz

Das ABAP RESTful Application Programming Model unterstützt das Qualitätsmerkmal der Evolutionsfähigkeit durch seinen durchgängig deklarativen Ansatz. Dank CDS sind die Anwendungen generell unabhängig von der darunterliegenden Datenquelle. Das Datenmodell wird ausschließlich auf logischer Ebene deklariert. (UI-)Annotationen sind ebenfalls auf dieser logischen Ebene angesiedelt, denn mit solchen Annotationen legen Sie die gewünschte Funktionalität fest, ohne die konkrete Umsetzung im UI kennen zu müssen. Das ermöglicht es, Annotationen grundsätzlich unverändert in der Anwendung zu belassen, auch wenn Sie deren Interpretation später durch eine andere UI-Technologie vornehmen.

Entkopplung von der Technologie

Auch in der Verhaltensdefinition deklarieren Sie transaktionale Eigenschaften und Geschäftslogik, ohne die konkrete technische Umsetzung kennen zu müssen. Business-Services abstrahieren abermals von der darunterliegenden Laufzeitumgebung, dem SAP-Gateway-System. Bis zur Servicedefinition nehmen Sie noch keinen Bezug auf ein konkretes technisches Protokoll, mit dem die definierten Entitäten exponiert werden. Diesen Bezug stellen Sie erst im Service-Binding her. Das Service-Binding kann daher aus Sicht von SAP als Ankerpunkt dienen, um neue Schnittstellentechnologien und Protokolle zu adaptieren.

Verschalte Bestands-anwendungen

Von dieser Evolutionsfähigkeit profitieren nicht nur Anwendungen, die von Grund auf mit dem ABAP RESTful Application Programming Model entwickelt wurden, sondern auch Bestandsanwendungen, die mit dem ABAP RESTful Application Programming Model verschalt wurden. Solche verschalten Bestandsanwendungen können ebenfalls mit RAP-Mitteln an verschiedene Schnittstellen und Protokolle exponiert werden, ohne dies in der Bestandsanwendung selbst definiert zu haben. Das bedeutet auch, dass bestehende RAP-Anwendungen auch zukünftig von neu unterstützter technischer Funktionalität oder Schnittstellen (z. B. zur Datenbeschaffung für Formulare) profitieren können.

1.5.2 Entwicklungseffizienz

Zu den qualitativen Anforderungen an ein Programmiermodell gehört auch, die Anwendungsentwicklerinnen und -entwickler beim Bau von Anwendungen effizient zu unterstützen. So ist es ein langjähriges Bestreben der Softwareentwicklung, die Entwicklungsteams durch das Angebot von Technologien und Querschnittsdiensten (z. B. UI-Technologien, Applikationsserver etc.) zu entlasten, damit diese sich auf die Umsetzung der fachlichen Anwendungsebene konzentrieren können.

Unterstützende Konzepte

Mit den folgenden Ansätzen unterstützt das ABAP RESTful Application Programming Model die Entwicklungseffizienz:

- **Deklarativer, modellbasierter Ansatz**
  Ein modellbasierter Ansatz stellt eine auf den jeweiligen Anwendungszweck zugeschnittene formale Sprache bereit. Beispielsweise ist die BDL eine formale Sprache, die auf den Zweck der Verhaltensmodellierung zugeschnitten ist. Aus dem auf Basis der Sprache entstandenen Quelltext (dem Modell, das auch ein grafisches Modell sein kann) werden (vom SAP-Standard) Entwicklungsobjekte generiert, oder das Modell wird zur Laufzeit generisch ausgewertet. Sie deklarieren also die gewünschte Funktionalität (das »Was«) und müssen sich um die konkrete technische Realisierung (das »Wie«) keine Gedanken machen.
- **Integration in die ABAP-Sprache**
  Basierend auf dem CDS-Datenmodell ist das ABAP RESTful Application Programming Model fest in die ABAP-Sprache integriert und sorgt dadurch für eine strenge Typisierung RAP-basierter Schnittstellen (durch abgeleitete Datentypen). Das ermöglicht eine statische Syntaxprüfung und eine bessere Werkzeugunterstützung (z. B. durch den Code-Assist und die Element-Info in den ADT). Dadurch werden wiederum Fehleranfälligkeit und Rechercheaufwand reduziert, was der Entwicklungseffizienz zugutekommt.
- **Standard-Implementierungen**
  Das Programmiermodell liefert für bestimmte Aufgaben der Anwendungsentwicklung wie die Implementierung von Standard-Operationen (Anlage, Ändern, Löschen), die Verwaltung des Transaktionspuffers, die Persistenz und das Draft-Handling Standard-Implementierungen aus. Dieses Standard-Implementierungen können Sie durch die Angabe entsprechender Schlüsselwörter in der Verhaltensdefinition für das Geschäftsobjekt nutzen. Auf diese Weise können Sie sehr schnell erste Versionen einer neuen Anwendung entwickeln.

  Die Standard-Implementierung von OData wird durch das im Hintergrund genutzte SAP Gateway bereitgestellt, auf dessen Funktionalität Sie über das Service-Binding zugreifen können. Dazu müssen Sie nicht einmal wissen, dass SAP Gateway als Technologie verwendet wird. SAP-Gateway-Kenntnisse sind somit nicht notwendig, was sich positiv auf die

Anzahl der notwendigen Voraussetzungen auswirkt, die Sie mitbringen müssen.

- **Standard-Vokabular für Geschäftsanwendungen**
  Das Programmiermodell definiert ein Standard-Vokabular für Geschäftsanwendungen und Geschäftslogik. Validierungen (`validation`), Ermittlungen (`determination`) und Aktionen (`action`) sind als explizite Schlüsselwörter in der Verhaltensdefinition nutzbar und damit auch in den ABAP-Sprachumfang eingegangen. Diese Konzepte erleichtern die Kommunikation innerhalb des Entwicklungsteams und mit Fachexpertinnen und -experten, wenn Anforderungen und deren Umsetzung besprochen werden.

1.5.3 Testbarkeit

Änderbarkeit wird seit den Anfängen der agilen Entwicklung vor allem durch automatisierte Tests auf verschiedenen Granularitätsstufen (von einzelnen Modulen bis hin zu vollständigen End-to-End-Anwendungen) gewährleistet. Software ist aber nicht von Haus aus gut testbar. Das Qualitätsmerkmal der *Testbarkeit* bestimmt, wie einfach die Funktionalität einer Anwendung durch eine Test-Suite automatisiert getestet werden kann. Programmcode ist viel leichter und sicherer änderbar, wenn er mit den Mitteln der Testautomatisierung mit zuverlässigen Testfällen abgesichert wurde.

Interaktionsphase und Transaktionspuffer

Wie in Abschnitt 1.2.1, »RAP-Transaktionsmodell«, beschrieben, führt das ABAP RESTful Application Programming Model das Konzept der Interaktionsphase (getrennt von der Speichersequenz) und des Transaktionspuffers ein. Durch die Nutzung des ABAP RESTful Application Programming Model ist es schwierig, an diesem Konzept »vorbei zu programmieren«. Durch das Vorhandensein des (transienten) Transaktionspuffers wird es einfacher, abhängig vom Testfall unterschiedliche Testdaten zu hinterlegen und Testfälle nacheinander automatisiert auszuführen, ohne dass es zu Überscheidungen der Datenbereiche kommt. Tests werden typischerweise durch die Implementierung von Testfällen mithilfe des Frameworks *ABAP Unit* realisiert. ABAP Unit ist ein Test-Framework, mit dem Sie Unit-Tests für Ihre Anwendungen entwickeln und ausführen können.

1.5.4 Trennung zwischen Fachlichkeit und Technik

Das Prinzip der Trennung zwischen Fachlichkeit und Technik ist eine Anwendung des Prinzips *Separation of Concerns*. Sie ist notwendig, um unabhängig von äußerer Technologie einen fachlichen Kern herauszubilden, der

technologieunabhängig änderbar und möglichst dauerhaft wiederverwendbar bleibt. Dieses Prinzip ist ein wichtiges Mittel, um bestimmte Qualitätsmerkmale umzusetzen.

Fachlogik und technische Umgebung

Mit Fach- oder Geschäftslogik ist die Umsetzung des Anwendungszwecks betriebswirtschaftlicher Software gemeint, also die Programmlogik, mit der beispielsweise Einkaufs- und Bestellprozesse abgewickelt werden. Zur technischen Umgebung eines solchen fachlichen Kerns gehören etwa die Benutzeroberfläche und die Persistenzschicht (z. B. die verwendete Datenbanktechnologie).

Technologische Änderungen kommen häufiger oder aus anderen Gründen vor als Änderungen in der Anwendungsdomäne. So ist die Berechnung des besten Einkaufspreises unabhängig von der Logik der technischen Umgebung, z. B. welche Code-Artefakte notwendig sind, um diesen Preis auf der Benutzeroberfläche anzuzeigen, oder welche Schnittstelle und welches Nachrichtenformat benötigt werden, um den Preis an ein Fremdsystem zu übertragen. Software lässt sich leichter anpassen, wenn die Fachlogik von der technischen Programmlogik getrennt wird.

Expliziter Raum für Geschäftslogik

Mit dem Konzept des Geschäftsobjekts mit seiner Verhaltensdefinition und -implementierung schafft das ABAP RESTful Application Programming Model einen expliziten Raum für die Fach- oder Geschäftslogik. Es wird schwierig, technisch motiviertes Coding in diese Verhaltenslogik einzubringen, was negative Auswirkungen auf die weiteren Schritte der Anwendungsentwicklung hätte. Das Programmiermodell definiert außerdem einen Ablauf zur Verarbeitung von Standard-Operationen oder Aktionen. Dieser Ablauf gibt explizite Zeitpunkte und Schnittstellen vor, um beispielsweise Berechtigungsprüfungen, eine Nummernvergabe oder das Sichern von Geschäftsdaten zu realisieren.

1.6 Verfügbarkeit des ABAP RESTful Application Programming Model

In diesem Abschnitt erfahren Sie, mit welchem SAP-Produkten das ABAP RESTful Application Programming Model verfügbar ist. Sie erhalten einen kurzen Überblick über die relevanten Eigenschaften der jeweiligen Produkte und lernen, wie sich das Programmiermodell dort jeweils positioniert.

SAP-Produkte

Das ABAP RESTful Application Programming Model steht mit den folgenden Produkten zur Verfügung:

- SAP Business Technology Platform (SAP BTP), ABAP Environment (siehe Kapitel 12 »Besonderheiten im SAP BTP, ABAP Environment«)
- ABAP-Plattform für SAP S/4HANA on-premise

1.6.1 SAP BTP, ABAP Environment

Mit dem SAP BTP, ABAP Environment steht die ABAP-Plattform als Cloud-Service auf der SAP Business Technology Platform (SAP BTP) zur Verfügung. Die erste Version des ABAP Environment, die auf der SAP BTP verfügbar war, war die Version 1808. Das ABAP Environment ist auf logischer Ebene das Cloud-Pendant eines On-Premise-SAP-NetWeaver-Stacks. Es gingen damit allerdings umfangreiche Bereinigungsmaßnahmen einher, um eine Cloud-fähige Version des SAP-NetWeaver-Stacks zu realisieren. SAP-intern wurde für das ABAP Environment der Projektname *Steampunk* verwendet, der sich mittlerweile auch im Sprachgebrauch der Entwicklergemeinde eingebürgert hat.

ABAP-Sprachumfang für die Cloud

Für die Cloud-Entwicklung gibt es eine eigene ABAP-Sprachversion. Der Zugriff auf SAP-Standard-Funktionalität ist nur über offiziell freigegebene APIs möglich (*Whitelisted APIs*). Modifikationen des SAP-Standard-Codes sind nicht erlaubt, und es steht nur noch eine wohl definierte Auswahl an Infrastrukturkomponenten zur Anwendungsentwicklung bereit, beispielsweise das Anwendungslog, die Hintergrundverarbeitung oder der Dienst zur Nummernvergabe. UI-Frameworks wie Web Dynpro ABAP oder der Floorplan Manager werden nicht mehr unterstützt, da auch in der Cloud SAP Fiori das Mittel der Wahl zur Realisierung webbasierter Benutzeroberflächen ist.

ABAP Environment

Zur effizienten Anwendungsentwicklung auf der SAP BTP steht Ihnen das ABAP RESTful Application Programming Model vollumfänglich zur Verfügung. Neue Releases des ABAP Environment werden vierteljährlich veröffentlicht. Damit werden stets auch die neuesten Features des Programmiermodells ausgeliefert. Sie können das ABAP Environment als Plattform zur Realisierung von Stand-alone-Anwendungen nutzen. So können Sie als SAP-Partner Cloud-Anwendungen entwickeln und betreiben. Darüber hinaus können Sie die Side-by-Side-Erweiterungsoption der Anwendungssoftware wie SAP S/4HANA, SAP S/4HANA Cloud oder SAP ERP nutzen. Sie sind dabei nicht unbedingt auf Anwendungssoftware aus dem Hause SAP beschränkt, sondern können letztlich beliebige Anwendungen integrieren, sofern diese geeignete Schnittstellen exponieren.

1.6.2 ABAP-Plattform für SAP S/4HANA on-premise

Das ABAP RESTful Application Programming Model für SAP S/4HANA on-premise war erstmalig mit der ABAP-Plattform für SAP S/4HANA 1909 FPS00 (entspricht SAP NetWeaver 7.54) verfügbar. Das Programmiermodell löste das ABAP-Programmiermodell für SAP Fiori ab, das bereits ab SAP NetWeaver 7.50 unterstützt wird. Das ABAP-Programmiermodell für SAP Fiori wird jedoch auch weiterhin vollumfänglich unterstützt.

Releaseabhängige Informationen zu den Funktionen des ABAP RESTful Application Programming Model

Welche Funktionalität des ABAP RESTful Application Programming Model Ihnen in welchem Release der ABAP-Plattform für SAP S/4HANA zur Verfügung steht, erfahren Sie im SAP Help Portal (*https://help.sap.com*) auf der Seite zum ABAP RESTful Application Programming Model unter dem Produktkontext **ABAP Platform** und den zum Programmiermodell hinterlegten Release Notes.

Verwendung von SAP-Entwicklungsobjekten

Auch wenn für SAP S/4HANA der Grundsatz »Keep the core clean« gilt, haben Sie on-premise die Möglichkeit, alle SAP-Entwicklungsobjekte für Ihre Eigenentwicklungen zu verwenden, auch wenn diese nicht freigegeben sind und die API-Freigabe nicht über eine syntaktische Prüfung durchgesetzt wird. Ebenso ist es technisch weiterhin möglich, Modifikationen am System vorzunehmen. Die ABAP Development Tools stehen dazu als Entwicklungsumgebung zur Verfügung.

Die ABAP-Plattform wird nicht mehr als SAP NetWeaver Application Server ABAP einzeln ausgeliefert, sondern ist Bestandteil einer SAP-S/4HANA-Installation. Somit ist die ABAP-Plattform an den jährlichen Releasezyklus von SAP S/4HANA on-premise gebunden. Neuerungen im ABAP RESTful Application Programming Model stehen Ihnen demnach nur einmal jährlich zur Verfügung.