

Browse the Book

Technical Architecture Foundation describes the technical foundation of SAP S/4HANA, including a look at the virtual data model and the ABAP RESTful application programming model. This chapter shows the reader the essential parts of a business application and how the semantics function on a technical level.

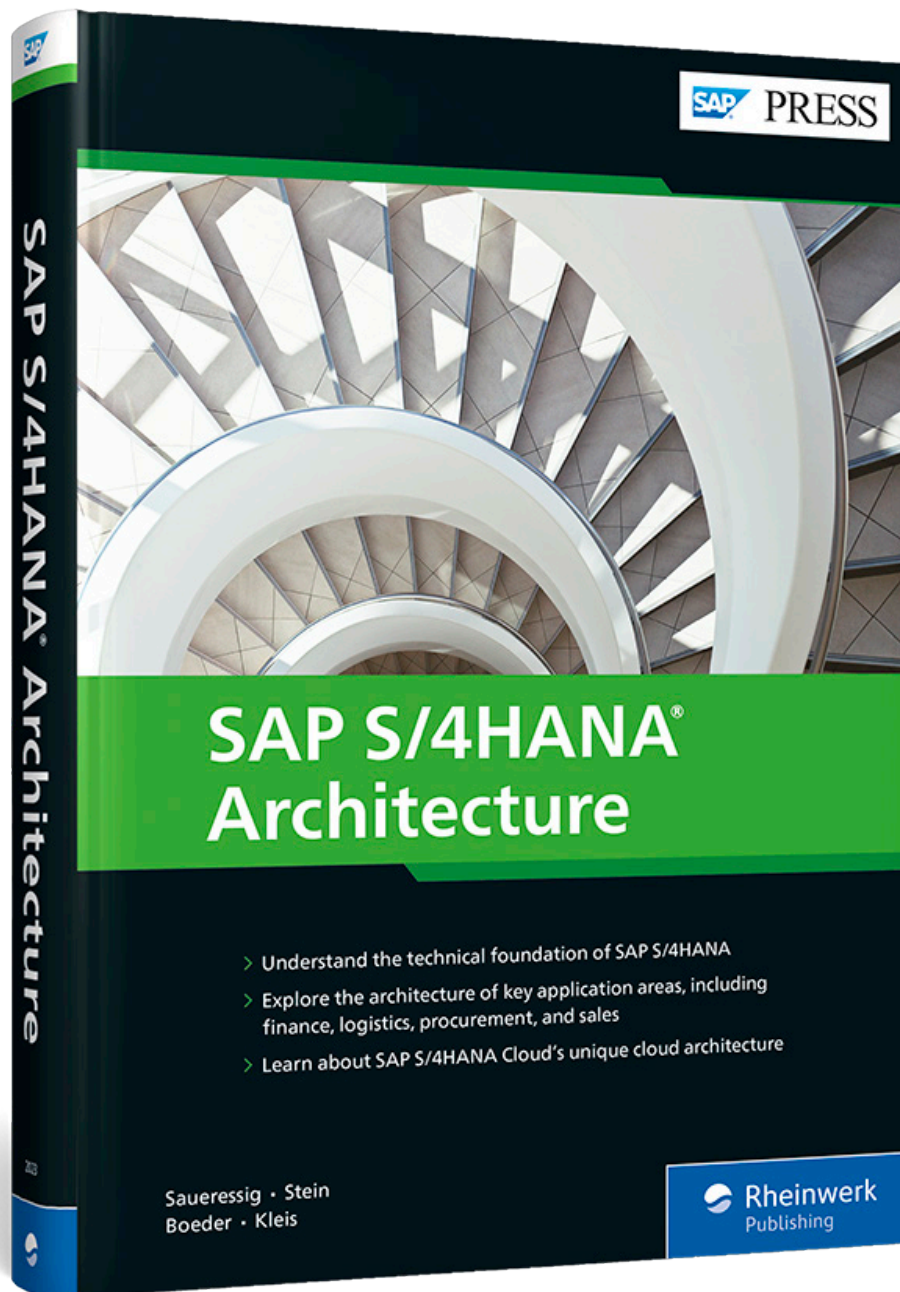
-  **“Technical Architecture Foundation”**
-  **Contents**
-  **Index**
-  **The Authors**

Thomas Saueressig, Tobias Stein, Jochen Boeder, Wolfram Kleis

SAP S/4HANA Architecture

520 Pages, 2021, \$79.95
ISBN 978-1-4932-2023-6

 www.sap-press.com/5189



Chapter 2

Technical Architecture Foundation

Source code and data type definitions contain no information about development or architecture decisions. The virtual data model and the ABAP RESTful application programming model break with this paradigm: they focus on the essential parts of a business application and make their semantics transparent.

Now it's time to get technical. We have mentioned the virtual data model (VDM) already a few times. In this chapter, we explain in detail what it is and how it is built. In particular, we explain how the core entity of transactional ERP applications—the business object—is modeled according to the VDM using CDS views. Then we show how the ABAP RESTful application programming model supports developers in implementing ERP applications using these business objects.

2.1 The Virtual Data Model

The *virtual data model* represents the semantic data model of the business applications in SAP S/4HANA. It aims at exposing all the business data in a way that eases its understanding and consumption. It is called a *virtual* model because it abstracts data from the database tables. This way, existing tables can be transformed into an aligned uniform data model where necessary.

Besides describing the business semantics of the applications, the VDM is also an executable model that provides access to the corresponding data at runtime. The VDM is implemented using specifically classified Core Data Services (CDS) entities, which comply with the VDM rules (Section 2.1.1). Among others, these VDM rules include fundamental naming rules (Section 2.1.2) and rules for structuring the VDM entities (Section 2.1.3). On the one hand, the rules foster the consistency of the models; on the other hand, they allow efficient development of applications and APIs that cover analytical consumption (see Chapter 4, Section 4.1), transactional processing (Section 2.2), and search-related consumption scenarios (see Chapter 3, Section 3.2). In addition to providing the data model for applications, VDM views are also used for other tasks, such as CDS-based extraction (see Chapter 6, Section 6.8.1).

The VDM is not intended only to be used for developing applications at SAP. Instead, released SAP VDM models and released SAP services built on them offer a stable interface with a well-defined lifecycle. As an SAP customer or partner, you can use them to build your own applications and for enhancing SAP applications.

2.1.1 Core Data Services

Core Data Services (CDS) support the definition of semantically rich data models. These models are managed by the Data Dictionary of the ABAP platform and can be executed in the database system.

CDS views represent the most important CDS entity type. They capture select statements in a syntax that is closely related to that of structured query language (SQL). CDS views can be used, for example, as the data source in ABAP select statements. The results of querying a CDS view can be restricted by attaching access control models to the CDS view. This means that the query only returns the data that the current user is authorized to read. These access control models are defined using the CDS *data control language* (DCL). CDS views support the definition of supplementary metadata by means of annotations. Furthermore, CDS views allow modeling associations, which represent directed relations to other CDS entities. Both the annotations and the associations are interpreted by the various consumers of the CDS models. In specific, the ABAP infrastructure uses the corresponding information for deriving additional functionality and services from the CDS models. For example, a CDS view annotated accordingly can be executed by the analytic engine (Section 2.1.4). The analytic engine provides advanced features such as exemption aggregations and hierarchy handling, which are not modeled by the plain select statement of the CDS view itself.

2.1.2 Naming Conventions

The VDM is based on a set of common naming rules. Applying the following rules ensures consistent and self-explanatory naming:

- A name is precise and uniquely identifies a subject. Generic names are avoided. For example, the identifier of the sales order document is named `SalesOrder`, not just `ID` or `Order`.
- A name captures the business semantics of a subject.
- Names being unique implies that different subjects must have different names. This is specifically important for identifiers and codes. Using the same name for two fields implies that their underlying value lists match.
- Names are composed from English terms in camel case notation with an uppercase first letter. Underscores are used in predefined cases only. Abbreviations are avoided.

The naming rules are applied to all CDS entities and their parts—for example, names of fields, associations, parameters, or CDS views.

Figure 2.1 illustrates an example of an SAP Fiori app that uses an OData service, which in turn is based on a VDM view stack. As you can see, the first VDM view (at the bottom) maps the technical field name `MATNR` from the database table onto the semantic name `Product`, which is then used on several layers up to the user-facing SAP Fiori UI.

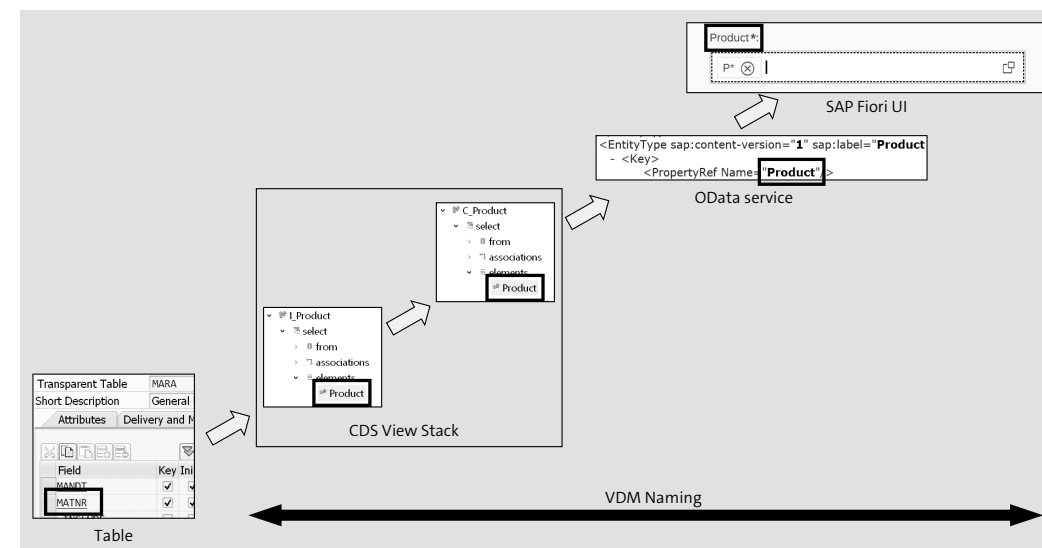


Figure 2.1 Field Names in VDM

2.1.3 Structure of the Virtual Data Model

Within the VDM, CDS entities serve distinct purposes. They are classified accordingly by means of VDM annotations. Note that a CDS entity becomes a VDM entity if it uses VDM annotations and if it adheres to the VDM guidelines.

VDM views are organized in a hierarchical structure following a layered approach. The upper layers select from and define associations to the same or lower layers, but not vice versa. The VDM views are assigned to layers with a special CDS annotation (`@VDM.viewType`).

Figure 2.2 depicts the admissible dependencies between the views as data sources and the different types of views. Along with the dependencies, the typical prefixes of the names of the CDS views are visible here too. Consumption views have name prefix `C_` and remote API views have the prefix `A_`. Basic views and composite views form an interface layer that can be used for building applications. The views of this interface layer have name prefix `I_`. As we will explain in the following sections, basic views and composite views can also be restricted reuse views, indicated by name prefix `R_`.

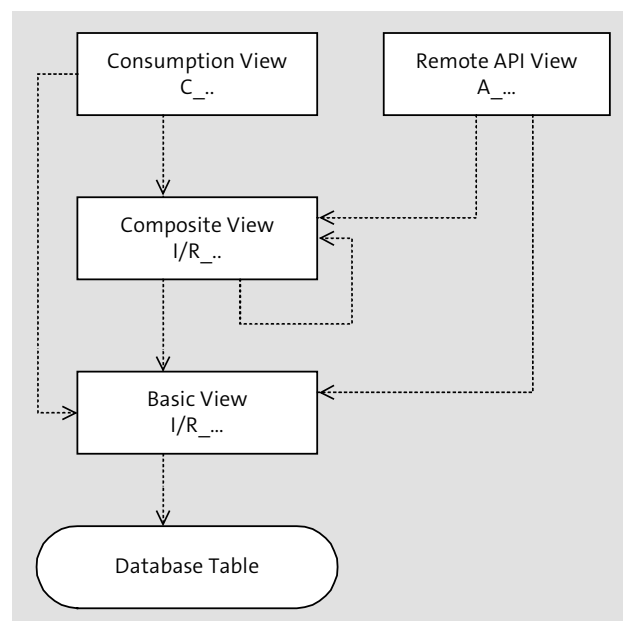


Figure 2.2 View Layering: Select-from Relationships

Basic Views

The lowest level of the VDM view stack is defined by *basic views*. Basic views are the most important constituents of the VDM. They establish what you may call the *entity relationship model* of the applications, from which they get information about data structures, dependencies, and metadata. The main purpose of the basic views is to serve as reusable building blocks for any other nonbasic VDM views. Because the basic views expose all data, there is no need for any other nonbasic view to directly select from the database tables. In fact, accessing database tables from nonbasic VDM views is forbidden in the VDM. This way, basic views provide a complete abstraction from the database tables to higher layers.

Composite Views

Composite views comprise additional functionality on top of the basic views. Like the basic views, they are primarily intended to serve as reusable building blocks for other views. However, they can already be defined in such a way that they support a specific consumption domain. For example, they can define analytical cube views, which consolidate data sources for usage in multiple analytical queries.

Transactional Views

Transactional views are a special flavor of composite views. Transactional views define the data model of a business object and act as an anchor for defining its transactional processing-related aspects (Section 2.2.1). Transactional views may contain elements

that support the transactional processing logic like additional fields that help preserving user input temporarily. Therefore, transactional views are only used in the context of transactional processing—for example, when consumed by other transactional views or by consumption views, which delegate their transactional processing logic to the transactional views.

Consumption Views

Regular basic views support any use case. This means they are defined independent of a specific use case. In contrast, *consumption views* are deliberately tailored for a given purpose. Consumption views are expected to be directly used in a specific consumption scenario. For example, a consumption view can provide exactly the data and meta-data (through annotations) that is needed for a specific UI element.

Restricted Reuse Views

By default, basic and composite views define an interface layer (named with the *I_* prefix), which any SAP application may use. Once released, they can also be used by SAP customers and partners. However, development teams sometimes define basic and composite views that are only for local use in their own applications. Such views are not meant to be reused by developers from other application areas. In such a case, a *restricted reuse view* (named with the *R_* prefix) is defined. Examples of restricted reuse views are transactional processing-enabled views, which expose all functions and operations of a business object, including the internal ones.

Enterprise Search Views

Enterprise search uses special CDS views as search models. Enterprise search is described in Chapter 3, Section 3.2.

Remote API Views

Remote API views project the functionality of a single business object for external consumption. They decouple the regular, system-internal VDM model, which can evolve over time, from its external consumers, establishing a stable interface. Based on the remote API views, OData services are defined, which can be consumed by remote applications. The corresponding OData services are published on the SAP API Business Hub (<https://api.sap.com>).

2.1.4 Consumption Scenarios

The most prominent consumption scenarios supported by CDS entities are as follows:

- Analytics
- SAP Fiori UI applications
- Enterprise search
- Remote APIs

Figure 2.3 shows a typical VDM view stack for analytical and transactional processing-enabled applications. Use case-specific views are defined on top of basic and composite views to adapt the data model and functionality to the individual application needs.

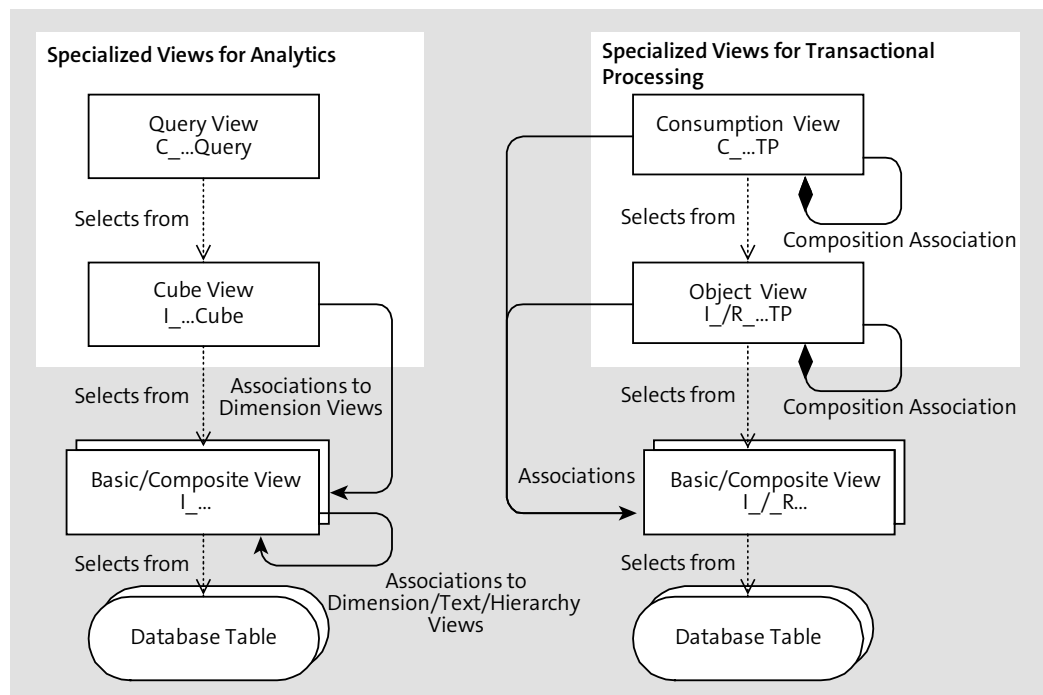


Figure 2.3 View Hierarchies

Analytical Applications

Analytical applications are based on cube views and a network of associated dimension views, which themselves can associate text and hierarchy views. The actual analytical application is defined by an analytical query view, which projects the envisioned functionality from its underlying cube view. Note that unlike other CDS views, the analytical query views themselves are not executed on the database. Instead, the analytic engine interprets their logic and directly executes selections from the cube and dimension views. Analytical applications are covered in detail in Chapter 4, Section 4.1.

Transactional Processing-Enabled Applications

The data model is captured by transactional views, which are related through compositional associations to form an entire business object. Therein the actual data selection logic is defined by the CDS view models. The behavior model specifying the supported

Create, Read, Update, Delete (CRUD) operations is defined in an attached *behavior definition model* and implemented in ABAP classes. The consumption view projects relevant functionality and augments the data model with annotations that provide the metadata for rendering the UI application built on the view. Associated interface views can be used to enrich the data model where appropriate. The recommended way to develop transactional applications is using the ABAP RESTful application programming model, which we discuss next in Section 2.2.

2.2 ABAP RESTful Application Programming Model

The *ABAP RESTful application programming model* is the common foundation on which the programming model of SAP S/4HANA is based. The strategy of SAP S/4HANA is to apply this target architecture across all applications. The model fosters SAP's general architecture goal to clearly separate the database model and database layer, the application logic (business logic) and the user interface. The SAP Fiori technology approach technically ensures the separation of the user interface (and related UX) from the business logic, leveraging OData as the protocol of choice. The ABAP RESTful application programming model addresses the modeling and implementation of application logic (often called *business logic*), including aspects that are common for all applications and the service-specific application logic. In addition, the ABAP RESTful application programming model ensures that the complete application implementation is protocol-agnostic and can also be reused when switching to a different protocol or protocol version.

In Section 2.2.1, we describe the common application model and logic, which we also refer to as a *business object* and the *business object implementation*.

The service-specific model and logic we often call a *business service* and the *business service implementation*. We explain its concept for OData, the main protocol used, in Section 2.2.2.

In Section 2.2.3, we describe the overall runtime architecture of the programming model.

2.2.1 Defining and Developing Business Objects

A business object is defined by a data model and its behavior. Figure 2.4 shows the design-time artifacts for defining and implementing business objects with the ABAP RESTful application programming model. The individual concepts and artifacts are explained ahead.

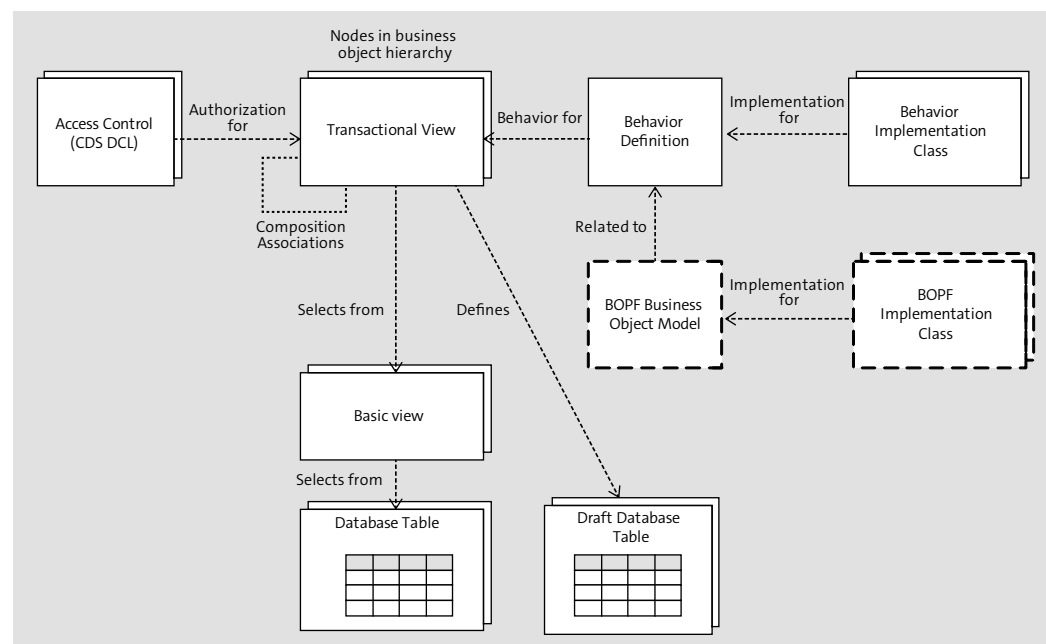


Figure 2.4 Business Object Definition and Implementation

Data Model

The data model of a business object consists of entities, also known as *business object nodes*, which are arranged in a hierarchical compositional tree structure. The data model is defined using CDS entities that are based on basic views of the virtual data model, which we introduced in Section 2.1. In Figure 2.4, the CDS entities for defining business object nodes are called *transactional views*.

In CDS models, *compositions* are a special kind of association that represent a parent-child hierarchical relationship, in which the child is a part of the parent and cannot exist without it. When defining business objects, compositions are especially important because they are needed to construct the business object model from nodes.

An example of a compositional structure is the sales order as a business object that is composed of the sales order header root entity, its sales order item children, and its sales order schedule line grandchildren. Usually we do not distinguish between the business object and its root entity when referring to them, but to avoid misunderstandings, *sales order* refers to all entities within the composition structure, whereas *sales order header* refers to the root entity only.

Other associations to related entities are also relevant and belong to the business object definition but are not part of the business object itself. Such associations can point to other business objects or business object entities—be it for read access, for transactional access, or for the purpose of value helps.

In the example of the sales order, a transactional association within the sales order might be a specialization of the composition, for example `toFreeGoodsSalesOrderItems`. Examples of associations to other business objects include associations to the business partner on the sales order header level or to the product on the sales order item level. Usually, the association to the business partner is also a transactional association; during the sales order entry, a new business partner instance may be created as a customer, or the customer data can be updated—for example, with a new address. The association to the product usually is not transactional because during the sales order entry no products are created or changed.

CDS entities can be accessed at runtime with SQL queries. The read authorization for such a query access is defined and modeled in a declarative way with a related access control model defined in CDS data control language (DCL) for each CDS entity.

The data model also defines the semantics of the fields based on the data types used:

- Because the ABAP type system doesn't support certain types, like `Boolean`, `DateTime`, or the Universal Unique Identifier (UUID), we use dedicated annotations on the field level to indicate that, for example, a character field with length 1 is a `Boolean`. When exposing these entities via OData, this annotation is evaluated because the OData type system is different and supports such corresponding types (i.e., `Edm.Boolean`, `Edm.DateTimeOffset`, and `Edm.Guid`).
- CDS annotations can add additional semantics on top of the pure data type definition. Such annotations can, for example, define that a field with type `DateTime` contains the time when the data was created or last changed, or that another field of type `Date` contains the “from” date on which some business data becomes valid or the “to” date until when business data is valid.

Developers define value helps or references to texts for IDs or codes as part of the data model using CDS annotations too.

Behavior Model and Implementation

The behavior of a business object is modeled with an artifact called *behavior definition*. The behavior definition is based on and complements the underlying data model. Technically, it's assigned to the root view entity that represents the entire business object, and it inherits its name from the root view entity.

The behavior definition defines the transactional behavior of the business object by declaring the available operations for each entity of the business object. This could include common operations such as create, update, and delete, or application-specific actions and functions. In addition, the behavior definition specifies runtime behavior with regard to locking, entity tag (ETag) handling in the case of HTTP-based consumption (typically through OData), authorization, feature control for the provided operations, support of draft handling, and much more.

Locking, ETag definition, and authorization within a business object usually are not specific to every entity but apply to the complete business object or a compositional subtree. To support this modeling for locking, ETags, and authorization, entities can be defined as `MASTER` or `DEPENDENT`. Thus locking, ETags, and authorization are invoked on the `MASTER` entity even if a `DEPENDENT` entity is accessed. For example, the sales order header is the `LOCK MASTER`, and all other entities are `LOCK DEPENDENT`. If a sales order item in a sales order is changed, the `LOCKING` is on the sales order header.

Feature control refers to the availability of operations (create, update, delete, actions, functions) or the changeability of fields. We distinguish between static and dynamic feature control, while the dynamic feature control is again split into global and instance feature control:

- For operations, *static* means that the operation is or is not available for consumers. For example, if creation of an instance is never allowed directly but only through a factory action, the create operation might not be defined at all or marked as *internal*.
- For fields, *static* means, for example, that a field is read-only and thus cannot be changed by a consumer, but only within the business object implementation.
- *Global* means that the operation is controlled independent of the instance. This is the only option for noninstance operations like create on the root entity level. Global can be applied to instance operations or fields, too—for example, when the action is deactivated based on certain configuration settings.
- *Instance* means that the operation or the field is controlled based on the concrete instance. For example, if a sales order is already released, the release action for that sales order is no longer enabled. At the same time, the delivery address cannot be changed further and is set to read-only dynamically for this sales order.

Some of the information specified in the behavior definition is relevant for the consumer of the business object. It defines, for example, the interface of the business object that can be used from ABAP code with the *entity manipulation language* (EML). EML is integrated into the ABAP language and allows reading and modifying business objects based on the operations defined in the behavior definition. Other aspects of the behavior definition are just implementation details. The implementation type in particular is an implementation detail that defines how the business object provider is implemented.

To ensure that the ABAP RESTful application programming model can be used across all applications in SAP S/4HANA, it supports three different implementation types:

- Managed implementation
- BOPF managed implementation
- Unmanaged implementation

Now we'll explain each type.

Managed Implementation Type

The *managed implementation type* is chosen for greenfield development when new applications are implemented or existing applications are refactored. Here, the infrastructure of the ABAP RESTful application programming model provides the main part of the provider implementation. This includes a full CRUD implementation with a transactional buffer, as well as read and write access to the application database tables. The *transactional buffer* is the buffer that holds the transactional changes that are processed during the transaction. When finalizing the transaction and saving the data, these changes are taken and saved to the database. In a managed implementation, the application only deals with its own application logic. The ABAP RESTful application programming model runtime takes care of buffer handling, locking, and so on.

The application logic is defined using determinations and validations that are declared in the behavior definition and can be implemented with application-specific ABAP code:

- A *determination* is application logic that changes data triggered by other changes. For example, the sales order item amount is calculated based on changes to the order quantity or the unit price. A determination might run immediately on modification or only later on saving.
- A *validation* is application logic that checks the consistency of the data in the transactional buffer to ensure only proper data can be finally stored in the database table. A validation runs only on saving. The infrastructure ensures that only relevant determinations and validations run based on so-called trigger definitions.

Available triggers are create, update, delete, and field changes. So, the determination to calculate the sales order item amount only runs if the order quantity or the unit price changes and only for the sales order item that was changed. For a good user interaction, it is sometimes necessary to invoke certain determinations or validations during the interaction phase. To do so, developers define *determine actions* in the behavior definition and assign the determinations and validations to these actions. The consumer can then call the determine action to invoke the determinations and validations.

Developers also use the managed implementation type when they refactor existing code and, for example, existing database tables need to be reused or need to be kept stable for interoperability. In this case, additional exits can be implemented to affect how changes are saved, to add additional save operations—for example, to write change documents—or to implement the lock method to reuse an existing enqueue object.

BOPF-Managed Implementation Type

The *BOPF-managed implementation type* is quite similar to the managed implementation type but provides the option to reuse and integrate an existing CDS-based BOPF

implementation. The Business Object Processing Framework (BOPF) is a predecessor of the business object framework of the ABAP RESTful application programming model. Before the ABAP RESTful application programming model was introduced, SAP recommended the CDS-based BOPF programming model. With the BOPF-managed option, the corresponding investments are protected and can continue to be used. The functionality of the BOPF-managed implementation type is comparable to the managed implementation type. The transactional buffer handling and further features are provided by BOPF, while applications focus on application logic in determinations and validations. Note that for the BOPF-managed implementation type, the determinations and validations are declared not in the behavior definition, but in the BOPF model as separate design-time artifacts. The link between the behavior definition and the BOPF model is established by naming, as both artifacts have the name of the business object as their name. In Figure 2.4, the artifacts of the BOPF model and the BOPF implementation are shown as dotted boxes because they are specific to the BOPF-managed implementation type. Note that the BOPF implementation replaces the behavior implementation in this case.

Unmanaged Implementation Type

The *unmanaged* implementation type is chosen for brownfield development when an existing application implementation should be reused and a refactoring or reimplementation to a managed implementation is not an option. In this case, the application is responsible for the complete provider implementation. This means that the application developers must implement all supported operations, such as create, delete, update, and save, on their own. This allows developers to reuse their existing application and delegate the work to the existing application APIs. The unmanaged implementation type is also the current option for traditional BOPF implementations. When developing a new application from scratch, the recommendation is to use the managed implementation type instead.

Comparison of ABAP RESTful Application Programming Model Implementation Types

Developers implement the operations and features of a behavior definition with dedicated ABAP classes. These classes refer to the behavior definition and thus have their method signatures inferred. This applies for the managed and the unmanaged implementation type, but developers must implement different parts in each case. In both use cases, developers must implement actions, functions, authority checks, and feature controls for their business object. In the unmanaged case, the application developer also must program the create, update, and delete operations, the transactional buffer handling, the locking, and the save sequence.

For the managed implementation type, the create, update, and delete operations with buffer handling come for free and cannot be implemented at all by application devel-

opers. For locking, the managed scenario offers a generic implementation that can be replaced if needed (with the `UNMANAGED LOCK` option). The ABAP RESTful application programming model framework takes care of saving of data if the database tables correspond to the entity definitions or simple mapping is defined in the behavior definition. If this isn't sufficient, developers can replace the generic implementation (with the `UNMANAGED SAVE` option).

A big benefit is that these ABAP RESTful application programming model features are integrated into the ABAP syntax and thus the signatures are fully typed. With this setup, design-time syntax checks and code completion are supported out of the box.

In the BOPF-managed case, developers implement the operations and features of a behavior definition in the BOPF implementation classes defined in the BOPF model. Because BOPF is a generic framework, the signatures here are only partially typed. The signatures have generic data types only (`TYPE DATA`) because BOPF does not create specific interfaces for each business object.

The Draft Feature

The *draft feature* is a special feature that enables the application to store the user's editing state at any point in time. This feature is also supported from end to end for SAP Fiori UIs implemented with SAP Fiori elements. Thus, for UI-based functionality and services, the draft feature is a core element of the architecture. The draft concept provides the following features:

- Users can interrupt the editing or creation of a business object without losing data.
- Support is provided for switching between devices and browsers during editing.
- In the future, support will probably be available for collaboration of multiple users working on the same business object.

Furthermore, with the draft feature, SAP S/4HANA can follow a stateless paradigm (REST using OData), ensuring cloud qualities such as robustness and scalability, while using an exclusive lock and the invocation of the necessary backend application logic during editing.

The draft implementation usually follows the implementation type of the application. For the managed implementation type, the `with draft` addition enables the draft feature. Then the ABAP RESTful application programming model infrastructure takes care of the draft handling. For the BOPF-managed implementation type, it is similar, with the difference that the BOPF infrastructure is responsible for draft handling.

For the unmanaged implementation type, draft enablement currently only can be defined as managed by the ABAP RESTful application programming model infrastructure (`DRAFT MANAGED`) or by the BOPF infrastructure (`DRAFT BOPF MANAGED`). Thus, the draft handling is always part of the infrastructure. Draft enablement for the unmanaged

implementation type requires that the relevant parts of the application logic are reimplemented so that they can be called by the ABAP RESTful application programming model or BOPF infrastructure for managed draft handling.

With ABAP RESTful application programming model, transactions are split into two phases, the interaction phase and the save phase (for details, Section 2.2.3). The basic architectural idea is that the interaction application logic is the same for active instances and draft instances. The draft reflects a persistent state of the interaction and the transactional buffer.

The application can have additional application logic that is invoked during the save phase and is relevant for nondraft instances only. The draft instances are saved as well, but they do not run through the related application logic of the save phase. This means that draft instances can be saved with missing or inconsistent data, which would be prevented by validations for nondraft instances.

2.2.2 Defining Business Services

Business services provide external access to certain aspects of one or more business objects, either for consumption by the UI layer or as a remote API to be called by other systems. However, you do not define these services directly on top of the transactional business object models. Instead, application developers create service-specific projections of the underlying business objects with the aspects they want to expose through the business service.

Developers can choose entities from the business object's consumer model and project it for a specific service exposure. This includes entities, their fields and associations, operations, actions, functions, ETags, drafts, or restriction with static feature control. Provider-side features like authorization checks, locking, or dynamic feature control, as well as the application logic, cannot be projected, deactivated, or overwritten because the provider is in charge of its own consistency. Typically, you define different services for different purposes, with different service-specific projections. OData services for the UI typically include information such as value helps, texts, and UI annotations that can be evaluated by metadata-driven SAPUI5 controls. In addition, such services support the draft functionality.

OData services for remote APIs usually expose only the entities of one business object—for example, a transactional business document. Related objects, such as codes and their texts or master data objects and their texts, are not included but are exposed by their own services.

Figure 2.5 extends the diagram from Figure 2.4 with the design-time artifacts for defining and implementing business services (included in the grey box). This includes the service-specific data model and implementation and the service definition and binding.

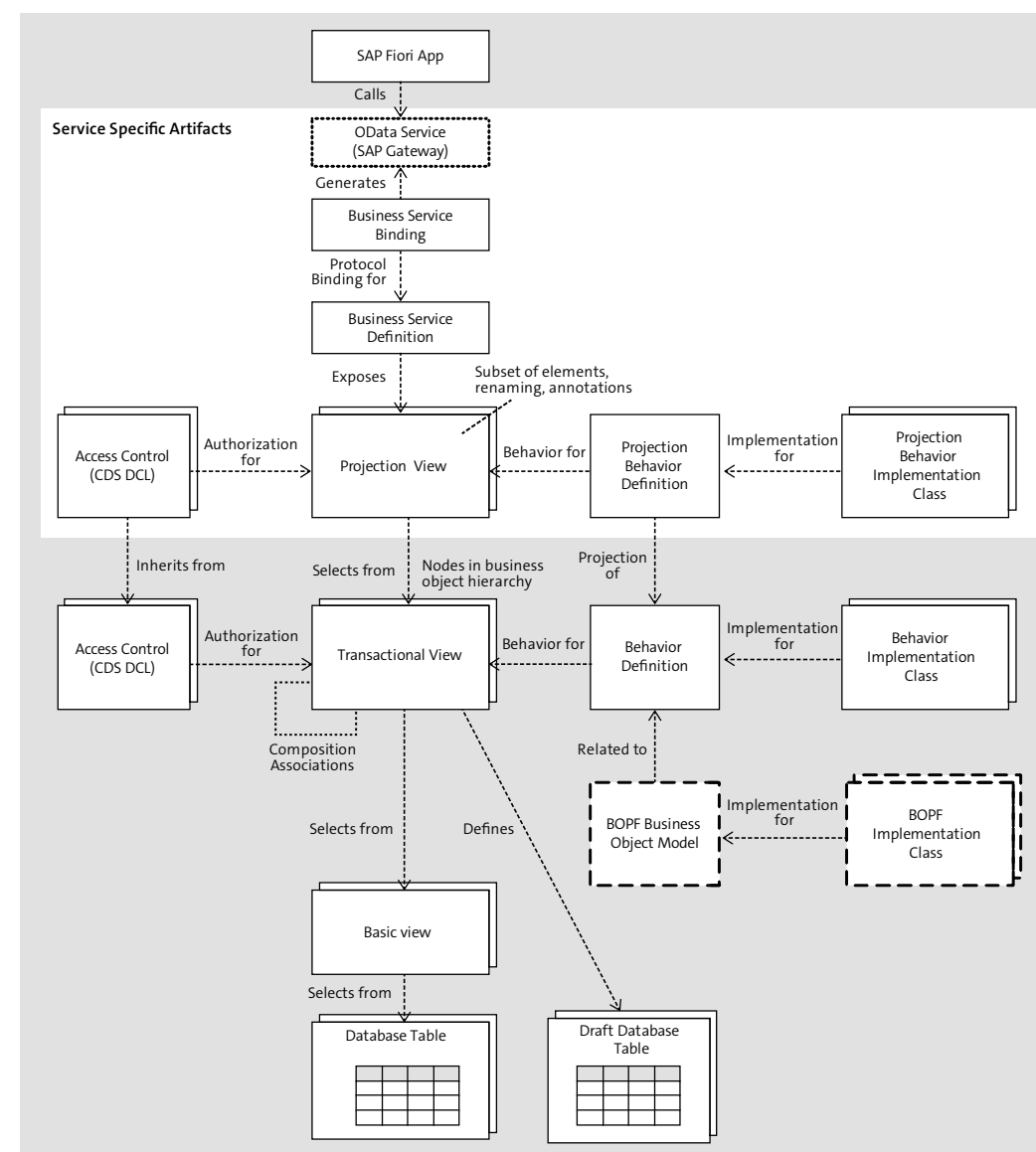


Figure 2.5 Service Definition and Implementation

Service-Specific Data Model

Developers define the business service-specific data model in two steps:

1. Select the entities to be included into the projection.
2. For each entity, decide which fields and associations are included.

In the ABAP RESTful application programming model, the projection is defined with special CDS entity types, called *projection view entities*. This special CDS entity type has

been introduced because not all CDS features are allowed in this kind of view. For example, aggregations, unions, and joins are not allowed. With a special CDS entity type, these restrictions can be enforced with syntax checks.

With projection views, you can not only restrict what entities and entity elements—columns and associations—are exposed. You can also restrict results that are returned by the view by adding a `WHERE` condition.

Sometimes the service layer is not a pure projection, but also needs to add certain additional service-specific elements to the data model. It is possible to add additional fields—currently fields that are calculated in ABAP only, also known as *virtual elements*—or to define new associations to service-specific entities on this service layer.

At runtime, CDS projection view entities can be accessed with SQL. Therefore, developers need to define the read authorization for such a query access and model it for each CDS entity with a related CDS access control definition (specified in data control language). In this case of transactional projection, it's only possible to “inherit” the access control definition of the underlying CDS entity. Otherwise, the authorization of the query access could be different than the authorization checked during the transactional access that is delegated to the underlying layer by definition (see the next section).

Behavior Model

In addition to the behavior definition of the business object, developers can define a projection behavior definition related to the projection root view entity of the projection. The main purpose of the projection behavior definition is to choose which features of the underlying business object behavior definition are exposed as the service-specific behavior. In the projection, only a part of the create/update/delete operations and the functions can be exposed, while others are omitted. In the same way, other features, such as support for draft or ETags, can be included or not. In addition, certain behavior can be restricted, mainly for static feature control. For example, a field that is editable in the business object might be set to read-only in the projection.

Developers can add certain service-specific implementations in the projection layer too. These could include additional checks to prevent certain data from being updated through this projection or additional transformations and augmentations that are not derived automatically by the infrastructure.

All other aspects of the behavior cannot be influenced with the projection behavior definition. It is, for example, not possible to change the implementation type or include or omit aspects like locking or authorization checks or to change or replace the application logic of the underlying business object.

Service Definition and Service Binding

To define an OData service on top of a projection view entity, developers have to add two additional artifacts, the service definition and the service binding:

- The *service definition* defines the scope of the service by listing the entities that are exposed by the service.
- The *service binding* references a service definition and specifies the protocol and protocol version with which the service will be created.

Currently the ABAP platform supports OData V2 and OData V4. The service binding also supports versioning of services. With this, published services can be kept stable by providing a new service version if incompatible changes to the interface are required.

For OData services, the entities of the CDS model are mapped to OData entity sets and entity types. The fields of the CDS entities are mapped to OData properties, and the CDS associations are mapped to OData navigation properties. Whether an OData entity supports POST, PUT, PATCH, or DELETE operations, in addition to GET, also is derived from the declared operations in the projected behavior definition. Furthermore, all exposed actions and functions of the behavior definition lead to OData actions and OData functions (or OData function imports in OData V2).

2.2.3 Runtime Architecture

Now, what happens at runtime? To illustrate the runtime architecture, we take an application with OData services that is built with the ABAP RESTful application programming model. The architecture is shown in Figure 2.6, with ABAP RESTful application programming model abbreviated as RAP, for better readability. The frontend sends OData requests, which are received by the OData service runtime and forwarded to the ABAP RESTful application programming model runtime through a protocol adapter. The runtime calls application-specific handlers—for example, for global authorization, global feature control, and locking of the lock master.

The ABAP RESTful application programming model runtime executes read requests directly using the corresponding views of the data models. For write operations, the runtime invokes the ABAP RESTful application programming model provider. As explained in Section 2.2.1, there are several implementation types for providers: managed, BOPF managed, and unmanaged.

The ABAP RESTful application programming model runtime accepts requests in entity manipulation language. In the ABAP RESTful programming model, EML provides the runtime access to the modeled entities and business objects. Because EML is part of the standard ABAP language and syntax, it allows a fully typed consumption of any ABAP RESTful application programming model-based implementation. EML allows for access to all external operations, like create, update, delete, any action, any function, the invocation of a lock, or the retrieval of data or feature control and authorization

information. With the MODIFY ENTITIES EML statement, several modifying operations of different types (such as create, update, delete, and action) can be bundled within one call—for example, to invoke multiple actions or to create a complete business object in one call and roundtrip.

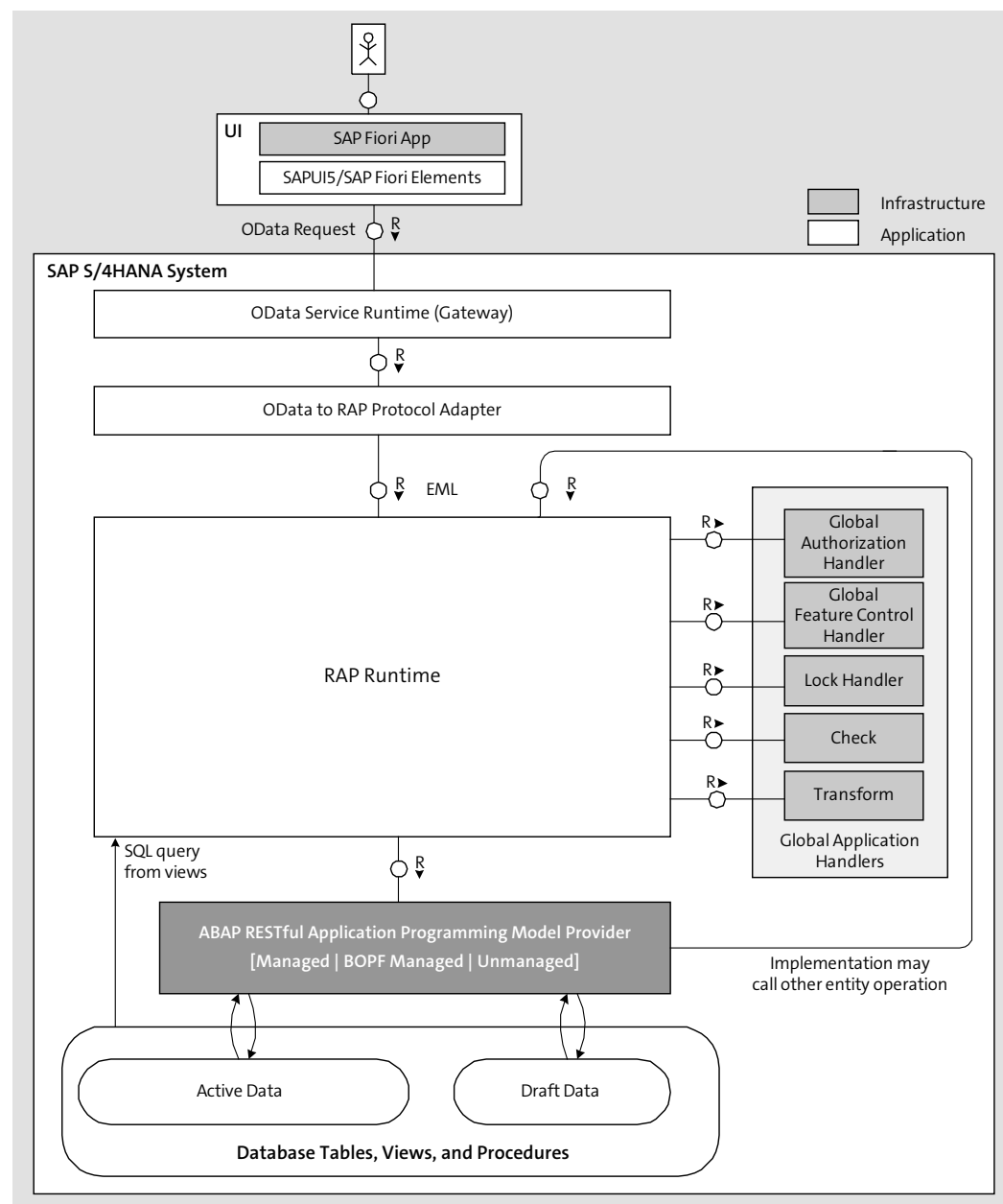


Figure 2.6 ABAP RESTful Application Programming Model Runtime Architecture Overview

Usually other applications or business objects, integration tests, generic consumption tools like data migration, or specific consumption tools like a SOAP API access the business object through EML. For OData, the ABAP platform provides end-to-end support, as mentioned in Section 2.2.2.

Phases of a Transaction in ABAP RESTful Application Programming Model

At runtime, a transaction defines the changes between two consistent saved states. This is also known as a *logical unit of work* (LUW) in the ABAP runtime. ABAP RESTful application programming model splits the transaction into the following two phases:

1. *Interaction*, in which one or multiple entities of one or multiple business objects are changed together
2. *Save*, in which further application logic runs, the data consistency is checked, and the changes are finally committed to the database

If the save phase is rejected due to an inconsistent final state, the transaction returns to the interaction phase to allow adjusting the data to reach a consistent state. Obviously, such an orchestration applies in an end user scenario only; while in machine-to-machine communication, the inconsistent data is not saved and the request is rejected.

Interaction Phase Operation Flow

If an operation is invoked (for example, through EML or OData), the internal processing follows a defined sequence that is partially enforced and provided by the ABAP RESTful application programming model runtime, partially to be implemented by the provider implementation.

Looking at the example of an action call for an instance-bound action, the following steps are performed to invoke this action:

1. Check global authorization. Is the user allowed to invoke this action at all?
2. Check global feature control. Is the action enabled at all?
3. Lock instance. Invoke the enqueue lock of the lock master.
4. Check ETags (only for HTTP requests). Does the provided ETag still match the current ETag?
5. Application-specific (pre)checks.
6. Application-specific transformations and augmentations. Only for projections.
7. Invoke actions of the underlying provider:
 - Check instance authorization. Is the user allowed to invoke this action for this instance?
 - Check instance feature control. Is the action enabled for this instance?
 - Invoke action implementation.
 - Invoke validations and determinations.

In case of a managed provider, the infrastructure executes the provider steps as listed here. Only the implementations of actions, validations, and determinations are application-specific.

In case of an unmanaged provider, the application implementation must perform all the steps. For other operations, the sequence is the same, but not all steps are always relevant, and different application handlers are invoked based on the operation. Static actions, for example, are not related to a specific instance. Consequently, the instance-related steps are not relevant for a static action. Another example is locking, which is not relevant for read access or for functions.

Within the interaction phase, multiple operations can be called that result in changes to the transactional buffer until at some point in time the editing is completed and the data is finally checked and saved to the database.

Save Phase Operation Flow

The COMMIT ENTITIES EML statement finalizes the transaction and the changed data is persisted to the database. For this, the ABAP RESTful application programming model runtime invokes all involved business objects within one transaction in a process called the *save sequence*. This sequence consists of the following steps:

- Finalize (*finalize*)
- Check before save (*check_before_save*)
- Draw numbers (*adjust_numbers*)
- Save (*save*)
- Cleanup (*cleanup*)

Each step is implemented by a corresponding method. The runtime of the ABAP RESTful application programming model calls these methods in sequence for each involved business object.

If one method or business object fails—for example, if it isn't in a consistent state to be saved from an application perspective—the runtime rejects the save operation.

During the *finalize* step, the runtime invokes validations, determinations, and application logic. Then during the *check before save* step, everything is checked for consistency and to determine whether it can be saved at all. If after these two phases an error occurs, the transaction moves back to the interaction phase, allowing the consumer to correct the issues and run the save again.

If everything is consistent, then numbers are drawn from number ranges in the *draw numbers* step. This is required for applications that use late numbering—for example, because there are legal requirements for gapless numbering. Finally, in the *save* step the data is saved to the database, usually by registering an update task function module. After the save is complete, the *cleanup* step ensures that transactional buffers are cleaned up and are reset to the proper state. If in these late phases an issue occurs,

the transaction is rolled back. An explicit rollback can be achieved with the ROLLBACK ENTITIES EML statement.

Provider Implementation

The overall architecture and the process for the interaction and save phases are independent of the concrete provider implementation. Every provider needs to fulfill the contract defined by the ABAP RESTful application programming model.

For managed providers, the infrastructure already fulfills most of the contract out of the box and the application developer can focus on the pure application logic. The infrastructure takes care of internal buffer handling of transactional changes, the proper program flow, and even the access to the database (see Figure 2.7, showing the runtime architecture with the managed provider implementation).

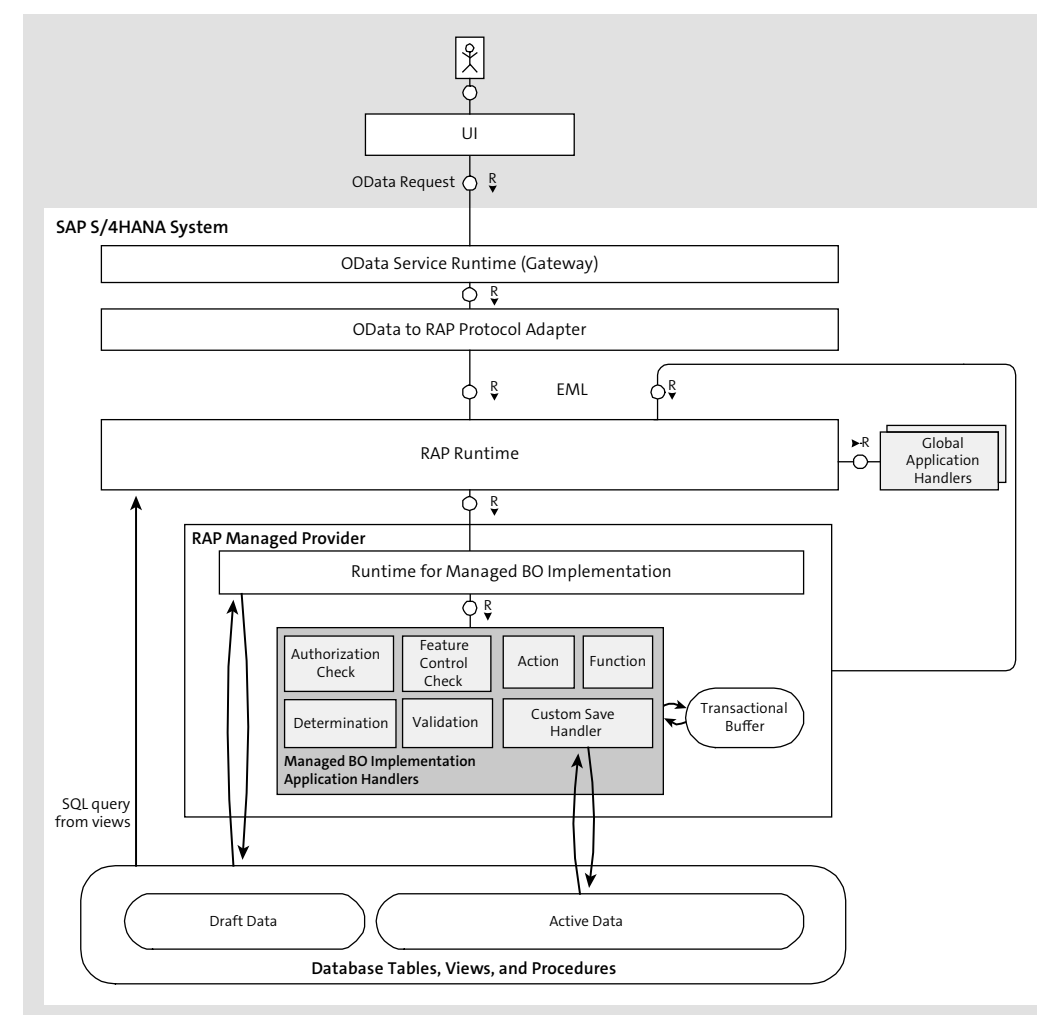


Figure 2.7 Runtime Architecture with Managed Provider

Application developers implement application handlers for different aspects only. Depending on the entity, these can be validations, determinations, entity-specific authorizations, feature controls, actions, and functions. As mentioned, the standard save functionality provided by the infrastructure can be replaced with custom save logic (UNMANAGED SAVE; Section 2.2.1). The custom save handler shown in Figure 2.7 represents such a custom save logic. In most cases, the standard logic provided by the infrastructure is sufficient, and a custom save handler is not needed.

For BOPF-managed providers, the situation is similar (see Figure 2.8). The ABAP RESTful application programming model runtime invokes the BOPF runtime through an adapter. The BOPF runtime takes care of the transactional buffer and the overall flow and calls the application-specific business object implementation handlers for determinations, validations, authorization checks, and so on.

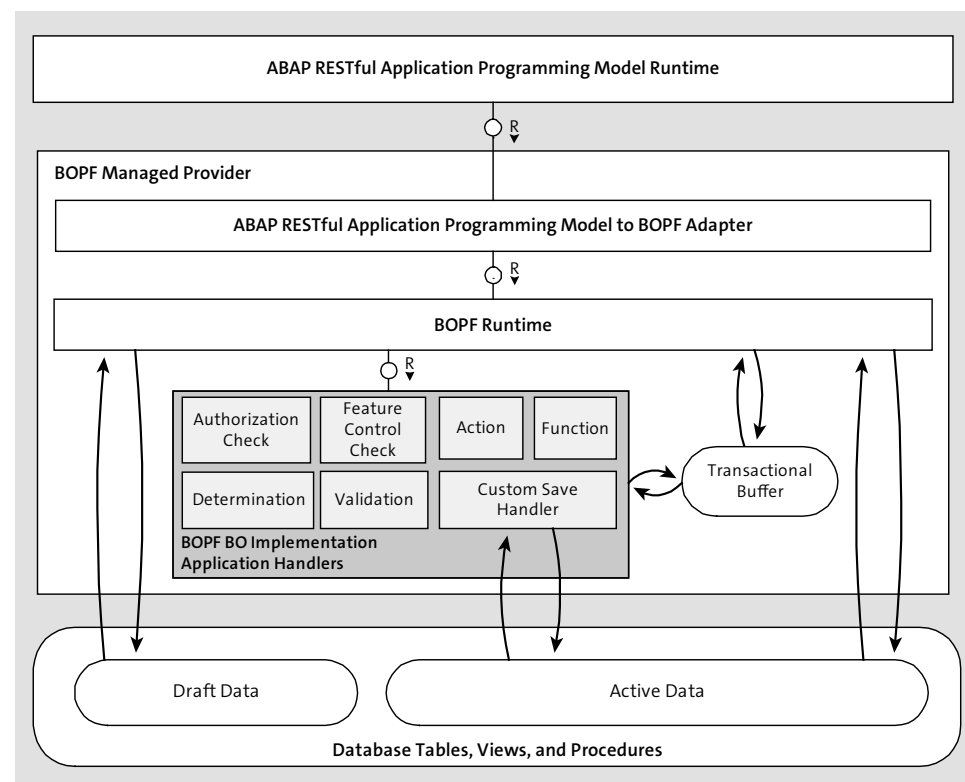


Figure 2.8 BOPF-Managed Provider

In case of an unmanaged provider, developers can reuse the existing application implementation. However, the implementation must fulfill the ABAP RESTful application programming model contract, and the unmanaged provider must implement several aspects that are otherwise provided by the infrastructure. This includes, for example, the create, update, and delete operations, the different transaction phases, the save

sequence, and the transactional buffer. Often the existing application is not structured in a way that fits, for example, with the transaction phases and the save sequence of the ABAP RESTful application programming model, and thus needs to be refactored.

As explained earlier (Section 2.2.1), the draft functionality must be handled either by the ABAP RESTful application programming model infrastructure or by BOPF, even for unmanaged providers. Figure 2.9 illustrates this option for an unmanaged provider with drafts managed by the infrastructure of the ABAP RESTful application programming model.

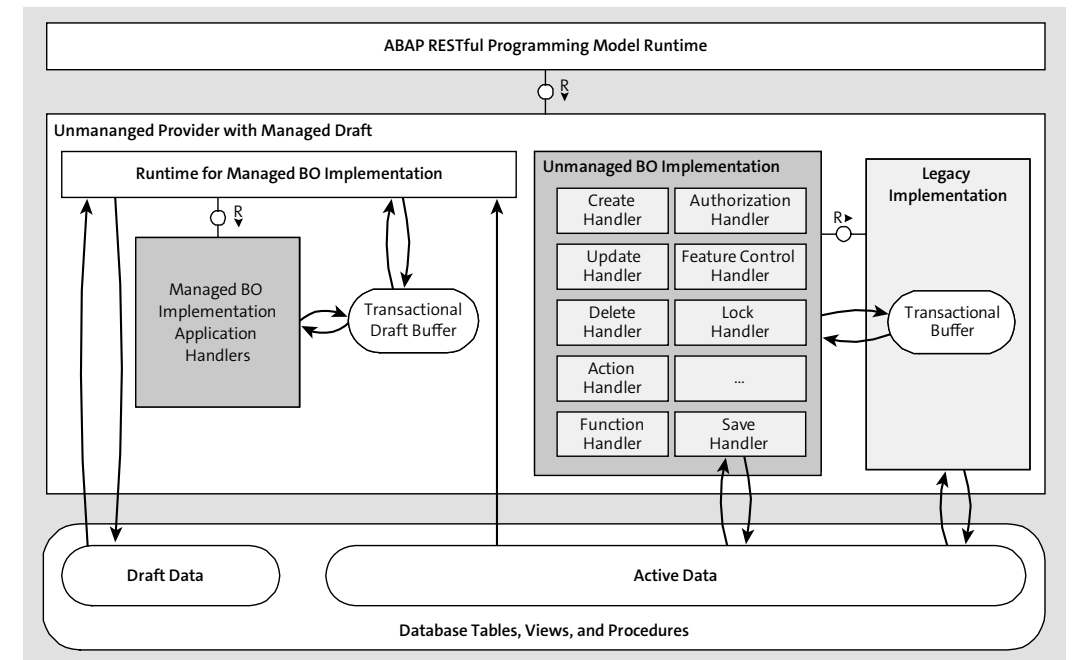


Figure 2.9 Unmanaged Provider with Managed Draft

Consumption through OData

As stated before, the ABAP RESTful application programming model provides full end-to-end support for consuming business objects through an OData service (OData V2 and OData V4). Thus, if an OData service is defined using a service binding (Section 2.2.2), the ABAP platform provides the complete protocol layer and translates any kind of request to either SQL (for query read access) or EML (for transactional access). With this architecture approach, the application implementation does not have to take the protocol and protocol version used into consideration. Thus, supporting additional channels or switching the protocol or protocol version can be achieved with very low efforts. In addition, the application can focus on real application logic instead of dealing with technical protocol topics.

2.3 Summary

To stay compatible with SAP ERP, the data model of SAP S/4HANA uses the same database tables for primary data – while derived data such as aggregates is computed on the fly. However, it shields these database tables using an understandable, comprehensive VDM that uses well-known business terminology, documents the relationships between entities, and enriches the entities with additional business semantics. The VDM is composed from a CDS hierarchy. The data model of a business object consists of CDS views too.

Based on this business object definition, application developers can use the ABAP RESTful application programming model to implement the transactional behavior, especially in CRUD operations.

With the managed, BOPF-managed, and unmanaged implementation types, the programming model supports three flavors. Each has a different level of framework support and solves a different use case, from developing a new application to refactoring an existing application.

By using the ABAP RESTful application programming model, application developers can easily allow business users to store a draft version of their business object.

Now you've seen that the transactional programming model relies on the VDM. In the next chapter, you'll learn how SAP Fiori apps consume the data exposed by the VDM.

Contents

Foreword	21
SAP S/4HANA: A New ERP Platform	23

PART I Foundation

1 Architecture Challenges of a Modern ERP Solution 29

1.1 Characteristics of a Modern ERP System	30
1.1.1 Even Higher Performance and Scalability	30
1.1.2 Consumer-Grade User Experience	32
1.1.3 Extensible Architecture	33
1.1.4 Intelligent ERP Processes	33
1.1.5 Simplified and Standardized Implementation	34
1.1.6 Cloud and On-Premise Deployment Models	35
1.1.7 Security, Privacy, Compliance, and Data Isolation	36
1.2 SAP S/4HANA Architecture Principles	37
1.2.1 Stable but Flexible Digital Core	37
1.2.2 Simplification with the Principle of One	37
1.2.3 Open for Innovations through Service Orientation	39
1.2.4 Modularization into (Hybrid) Integration Scenarios	40
1.2.5 Cloud First, but Not Cloud Only	40
1.2.6 Semantic Compatibility to Support Evolution with the Least Possible Disruption	41
1.3 Evolving a Cloud ERP System from the Best Possible Origins	42
1.4 Summary	42

2 Technical Architecture Foundation 43

2.1 The Virtual Data Model	43
2.1.1 Core Data Services	44
2.1.2 Naming Conventions	44
2.1.3 Structure of the Virtual Data Model	45
2.1.4 Consumption Scenarios	47

2.2	ABAP RESTful Application Programming Model	49
2.2.1	Defining and Developing Business Objects	49
2.2.2	Defining Business Services	56
2.2.3	Runtime Architecture	59
2.3	Summary	66
3	Simplified Experience	67
3.1	User Experience	67
3.1.1	SAP Fiori	67
3.1.2	User Experience Adoption Strategy	68
3.1.3	SAP Fiori Launchpad	69
3.1.4	SAP Fiori Apps	72
3.1.5	SAP Fiori Elements Apps	74
3.2	Search	77
3.2.1	Search Architecture	77
3.2.2	Enterprise Search Extensibility	80
3.3	Summary	82
4	Intelligence and Analytics	83
4.1	Analytics	83
4.1.1	SAP S/4HANA Embedded Analytics Architecture Overview	84
4.1.2	Embedded Analytical Applications	87
4.1.3	Modeling Analytical Artifacts	88
4.1.4	Analytics Extensibility	89
4.1.5	Enterprise Analytics Applications	91
4.2	Machine Learning	92
4.2.1	Machine Learning Architecture	93
4.2.2	Embedded Machine Learning	94
4.2.3	Side-By-Side Machine Learning Architecture	96
4.2.4	Machine Learning in SAP S/4HANA Applications	97
4.3	Intelligent Situation Handling	100
4.3.1	Example: Contract is Ready as Source of Supply	101
4.3.2	Technological Background	102
4.3.3	Intelligent Situation Handling Concept	104
4.3.4	User Experience	105

4.3.5	Use Case Examples	109
4.3.6	Intelligent Situation Automation	110
4.3.7	Message-Based Situation Handling	111
4.4	Summary	111
5	Extensibility	113
5.1	Key User Extensibility	113
5.1.1	Stability Criteria for Extensibility	114
5.1.2	Principles of In-App Key User Extensibility	116
5.1.3	Field Extensibility	117
5.1.4	Integration Extensibility	119
5.1.5	Custom Business Logic	119
5.1.6	Custom Business Objects	121
5.1.7	Custom CDS views	122
5.1.8	Lifecycle Management	122
5.2	Side-by-Side Extensions	123
5.2.1	Introduction to Cloud-Native Applications	124
5.2.2	SAP Cloud Platform and Programming Models	128
5.2.3	Cloud Foundry Environment	128
5.2.4	Kyma Runtime	128
5.2.5	Integrating with SAP S/4HANA Using the SAP Cloud SDK	129
5.3	Summary	134
6	Integration	137
6.1	SAP S/4HANA Integration Interface Technologies	137
6.1.1	OData Services	137
6.1.2	SOAP Services	138
6.1.3	Remote Function Call	138
6.1.4	BAPIs	138
6.1.5	IDoc	138
6.1.6	SAP S/4HANA API Strategy	139
6.2	SAP API Business Hub	139
6.3	Interface Monitoring and Error Handling with SAP Application Interface Framework	140

6.4	Communication Management in SAP S/4HANA Cloud	142
6.4.1	Communication Scenario	144
6.4.2	Communication User	144
6.4.3	Communication System	145
6.4.4	Communication Arrangement	145
6.4.5	Calling Inbound Services with User Propagation	145
6.5	Cloud Connector	146
6.5.1	Cloud Connector Principles	146
6.5.2	RFC Communication with SAP S/4HANA Cloud	148
6.6	Integration Middleware	148
6.7	Event-Based Integration	150
6.7.1	SAP Cloud Platform Enterprise Messaging	151
6.7.2	Business Events Architecture in SAP S/4HANA	152
6.7.3	Business Events in SAP S/4HANA	153
6.7.4	Event Channels and Topic Filters	154
6.8	Data Integration	154
6.8.1	CDS-Based Data Extraction	155
6.8.2	Data Replication Framework	157
6.8.3	Master Data Integration Services	158
6.9	Summary	159

7 Data Protection and Privacy 161

7.1	Compliance Base Line	162
7.2	Definitions and Principles	162
7.2.1	Basics in SAP S/4HANA	164
7.2.2	Data Subject Rights	165
7.2.3	Technical and Organizational Measures	167
7.3	Summary	169

PART II Application Architecture

8 Master Data 175

8.1	Product Master	175
8.1.1	Product Master Data Model	176

8.1.2	Product Hierarchy	180
8.1.3	Data Migration	182
8.1.4	Product SOAP Service API	183
8.1.5	Product Master Extensibility	183
8.1.6	Self-Service Configuration	184
8.2	Bill of Materials, Characteristics, and Configurations	185
8.2.1	Bill of Materials (BOM)	185
8.2.2	Classification System	187
8.2.3	Variant Configuration	188
8.2.4	Variant Classes	190
8.2.5	Super BOM	190
8.2.6	BOM with Class Items	191
8.2.7	Variant Configuration Profiles	191
8.2.8	Object Dependencies in Variant Configuration	191
8.2.9	User Interface and Grouping	192
8.2.10	Extensibility	192
8.2.11	High-Level and Low-Level Configuration	192
8.2.12	Embedded Analytics for Classification and Configuration Data	193
8.3	Business Partner	195
8.3.1	Architecture of Business Partner Master Data	196
8.3.2	SAP S/4HANA System Conversion Scenarios	201
8.3.3	Data Protection and Policy	203
8.3.4	Extensibility	203
8.3.5	Business Partner APIs	204
8.4	Summary	205

9 Sales 207

9.1	Architecture Overview	207
9.2	Sales Documents Structure	209
9.3	Authorizations	210
9.4	Sales Inquiries and Sales Quotations	211
9.5	Sales Order Processing	212
9.6	Sales Contracts	213
9.7	Sales Scheduling Agreements	214
9.8	Claims, Returns, and Refund Management	215

9.9 Billing	216
9.10 Sales Monitoring and Analytics	217
9.11 Pricing	220
9.12 Integration	221
9.13 Summary	222
10 Service Operations	225
10.1 Architecture Overview	225
10.2 Business Objects and Processes in Service Operations	226
10.2.1 Field Service	227
10.2.2 In-House Repair	228
10.2.3 Service Contracts	228
10.2.4 Solution Business	229
10.2.5 Interaction Center	229
10.3 Master Data and Organizational Model	229
10.3.1 Business Partner	229
10.3.2 Service Products	230
10.3.3 Organizational Units	230
10.3.4 Service Teams	230
10.3.5 Technical Objects	230
10.4 Data Model and Business Transactions Framework	231
10.4.1 Business Transactions Framework	231
10.4.2 Data Model	231
10.4.3 Transaction Type and Item Category	233
10.4.4 Common Functions for Service Transactions	233
10.4.5 Virtual Data Model	234
10.4.6 Public APIs	235
10.5 Integration	235
10.5.1 Data Exchange Manager	235
10.5.2 Backward Integration	236
10.5.3 Integration with SAP Field Service Management	237
10.5.4 User Interface Technology	237
10.6 Summary	238

11 Sourcing and Procurement	239
11.1 Architecture Overview	240
11.2 Procurement Processes	242
11.2.1 Direct Procurement	243
11.2.2 Indirect Procurement	244
11.3 Architecture of a Business Object in Procurement	244
11.4 Central Procurement	245
11.4.1 Backend Integration	248
11.5 APIs and Integration	249
11.5.1 SAP S/4HANA Procurement Integration with SAP Ariba and SAP Fieldglass	249
11.6 Analytics	253
11.7 Innovation and Intelligent Procurement	254
11.8 Summary	256
12 Logistics and Manufacturing	257
12.1 Architecture Overview	258
12.2 Organizational Units	260
12.3 Master Data Objects	261
12.4 Transactional Business Objects	262
12.5 Calculated Business Objects, Engines, and Process Control	265
12.5.1 Inventory	265
12.5.2 Available-to-Promise	267
12.5.3 Material Requirements Planning	271
12.5.4 Demand-Driven Material Requirements Planning	273
12.5.5 Kanban	274
12.5.6 Just-In-Time Processing	275
12.5.7 Predictive Material and Resource Planning	278
12.5.8 Capacity Planning	280
12.5.9 Production Planning and Detailed Scheduling	281
12.6 Cross Functions in Logistics and Manufacturing	281
12.6.1 Batch Management	282
12.6.2 Quality Management	282
12.6.3 Handling Unit Management	285

12.6.4	Serial Number Management	286
12.6.5	Inter-/Intracompany Stock Transport	287
12.7	Logistics Integration Scenarios	287
12.7.1	Warehouse Management	287
12.7.2	Manufacturing Execution Systems	288
12.8	Summary	288
13	Extended Warehouse Management	289
<hr/>		
13.1	Architecture Overview	289
13.2	Organizational Structure	291
13.3	Master Data	292
13.4	Stock Management	292
13.5	Application Components	294
13.6	Monitoring and Reporting	297
13.7	Process Automation	297
13.8	User Interface	298
13.9	Technical Frameworks	299
13.10	Warehouse Automation	300
13.11	Summary	300
14	Finance, Governance, Risk, and Compliance	303
<hr/>		
14.1	Finance Architecture Overview	305
14.2	Accounting	306
14.2.1	General Ledger	309
14.2.2	Fixed Asset Accounting	309
14.2.3	Inventory Accounting	311
14.2.4	Lease Accounting	313
14.2.5	Service and Sales Accounting	314
14.2.6	Group Reporting	318
14.2.7	Financial Closing	324
14.3	Tax and Legal Management	326
14.4	Enterprise Contract Management and Assembly	328

14.5	Financial Planning and Analysis	330
14.5.1	Budgetary Accounting	330
14.5.2	Predictive Accounting	332
14.5.3	Financial Planning	336
14.5.4	Margin Analysis	340
14.5.5	Overhead Cost	342
14.5.6	Production Cost	344
14.6	Payables Management	347
14.6.1	Supplier Invoicing	347
14.6.2	Open Payables Management	348
14.6.3	Automatic Payment Processing	348
14.7	Receivables Management	349
14.7.1	Open Receivables Management	350
14.7.2	Credit Evaluation and Management	354
14.7.3	Customer Invoicing	357
14.7.4	Dispute Resolution	358
14.7.5	Collection Management	360
14.7.6	Convergent Invoicing	361
14.7.7	Contract Accounting	365
14.8	Treasury Management	369
14.8.1	Advanced Payment Management	370
14.8.2	Bank Integration Using SAP Multi-Bank Connectivity	372
14.8.3	Connectivity to Payment Service Providers and Payment Gateways ...	374
14.8.4	Cash Management	375
14.8.5	Treasury and Risk Management	378
14.9	Central Finance	382
14.9.1	Replication	384
14.9.2	Mapping	386
14.9.3	Accounting Views of Logistics Information	387
14.9.4	Central Payment	390
14.9.5	Cross-System Process Control	390
14.10	Finance Extensibility	394
14.11	SAP Governance, Risk, and Compliance	394
14.11.1	Overview of SAP GRC Solutions	395
14.11.2	SAP GRC Solutions and SAP S/4HANA Integration	397
14.11.3	SAP S/4HANA Integration with Enterprise Risk and Compliance	397
14.11.4	SAP S/4HANA Integration with International Trade Management	398
14.11.5	SAP S/4HANA Integration with Access Governance	399
14.11.6	SAP S/4HANA Integration with SAP Privacy Governance	400
14.12	Summary	401

15 Localization in SAP S/4HANA	403
15.1 Advanced Compliance Reporting	403
15.2 Document Compliance	406
15.2.1 Motivation	406
15.2.2 Architecture Overview	407
15.2.3 Recent Developments and Future Outlook	409
15.3 Localization Toolkit for SAPS/4HANA Cloud	409
15.3.1 Components of the Toolkit	409
15.3.2 Extensibility Scenario Guides and the Community	410
15.4 Summary	411

PART III SAP S/4HANA Cloud-Specific Architecture and Operations

16 Scoping and Configuration	415
16.1 Configure Your Solution: Scoping and Configuration Today	417
16.1.1 Content Model of SAP Solution Builder Tool	417
16.1.2 Scoping and Deployment	419
16.2 Outlook: SAP Central Business Configuration	419
16.2.1 The Business Adaptation Catalog	420
16.2.2 The Ten Business Adaptation Catalog Commandments	422
16.2.3 Business Processes	423
16.2.4 Constraints	424
16.2.5 From Scoping to Deployment	424
16.3 Summary	424
17 Identity and Access Management	425
17.1 Architecture Concepts of Identity and Access Management	425
17.1.1 ABAP Authorization Concept	426
17.1.2 Authentication	426
17.1.3 Identity and Access Entities and Their Relationships	427
17.1.4 Identity and Access Management Tools	430
17.1.5 SAP Fiori Pages and Spaces	431

17.2 Managing Users, Roles, and Catalogs	432
17.2.1 Communication Arrangements	433
17.2.2 User Types	433
17.2.3 SAP PFCG Roles and Business Catalogs	434
17.2.4 Management of Users, Roles, and Catalogs by Customers	436
17.2.5 Auditors	438
17.3 Summary	440

18 Output Management 441

18.1 Architecture Overview	441
18.2 Printing	442
18.3 Email	444
18.4 Electronic Data Interchange	445
18.5 Form Templates	446
18.6 Output Control	447
18.7 Summary	449

19 Cloud Operations 451

19.1 SAP S/4HANA Cloud Landscape	451
19.2 Data Centers	453
19.3 Multitenancy	454
19.3.1 The System Architecture of SAP S/4HANA	455
19.3.2 Sharing the SAP HANA Database System	456
19.3.3 Sharing of ABAP System Resources	457
19.3.4 The Table Sharing Architecture in Detail	458
19.4 Software Maintenance	461
19.4.1 Maintenance Events	461
19.4.2 Blue-Green Deployment	462
19.5 Built-in Support	463
19.5.1 Support Journey without Built-in Support	464
19.5.2 Built-in Support Architecture	466
19.6 Summary	468

20 Sizing and Performance in the Cloud	471
20.1 Performance-Optimized Programming	471
20.1.1 Minimal Number of Network Round Trips and Transferred Data Volume	472
20.1.2 Content Delivery Networks	474
20.1.3 Buffers and Caches	474
20.1.4 Nonerratic Performance	475
20.2 Sizing	475
20.2.1 Example for Greenfield Sizing	476
20.2.2 Brownfield Sizing	478
20.3 Elasticity and Fair Resource Sharing	478
20.3.1 Dynamic Capacity Management	479
20.4 Sustainability	481
20.5 Summary	483
21 Cloud Security and Compliance	485
21.1 Network and Data Security Architecture	485
21.1.1 Access Levels	486
21.1.2 Resource and Data Separation	487
21.1.3 Resource Sharing	487
21.1.4 Data Security and Data Residency	488
21.1.5 Business Continuity and Disaster Recovery	488
21.2 Security Processes	488
21.2.1 Security Configuration Compliance Monitoring	488
21.2.2 Security Event Management and Incident Response	489
21.2.3 Infrastructure Vulnerability Scanning	489
21.2.4 Malware Management	489
21.2.5 Security Patch Management	489
21.2.6 User Management	490
21.2.7 Hacking Simulation	490
21.3 Certification and Compliance	490
21.3.1 SAP Operations	490
21.3.2 SAP Software Development	491
21.4 Summary	491

22 Outlook	493
The Authors	495
Index	513

Index

A

ABAP authorization concept	167, 426
ABAP CDS reader operator	155
ABAP Dictionary	184, 458, 462
ABAP RESTful application programming	
model	43, 49, 137
<i>BOPF-managed implementation</i>	53
<i>determination</i>	53
<i>draft feature</i>	55
<i>implementation</i>	52
<i>implementation types</i>	54
<i>managed implementation</i>	53
<i>transactional buffer</i>	53
<i>unmanaged implementation</i>	54
<i>validation</i>	53
ABAP-managed database procedures	
(AMDPs)	95, 219
Access	
<i>schema</i>	462
<i>sequence</i>	220
Account assignment	309
Accounting interface	331, 333, 341, 386
Accounting principle	310
Accounting views of logistics information	
(AVL)	387
Active-active	454
Actual costing	312
Adaptation Transport Organizer (ATO)	123
Advanced adapter engine extended	150
Advanced compliance reporting	389, 404
Advanced planning and optimization	176
Advanced variant configurator	188, 234
Alternative-Based Confirmation	270
Analytic engine, runtime objects	86
Analytical applications	48
Analytical list pages	87
Analytical model	
<i>analytical query view</i>	89
<i>cube view</i>	88
<i>dimension view</i>	89
<i>hierarchy view</i>	89
Analytics	83
<i>extensibility</i>	89
Anchor object	102
API consumption	132

Ariba Network	222
Audit	285
Authorization	167, 210, 426
Authorization object extension	435
Automatic Payment Processing	348
Availability group	294
Availability zones	486
Available-to-promise (ATP)	207

B

Backorder processing	269
Backup	488
Backward integration	236
Balance sheet valuation	312
Bandwidth	473
Bank statement processing	352
Basic views	46
Batch	282
Behavior definition	51
<i>model</i>	49
Bill of materials (BOM)	176, 185, 261, 337
Billable items	362
Billing	216
Billing document	216
<i>request</i>	216, 236
Billing due list	217
Blue-green deployment	461
BOM explosion	187, 271
Brownfield	478
Budget availability control	330
Built-in support	463
Business adaptation catalog	420
Business Application Programming	
Interfaces (BAPIs)	138
Business area	421
Business catalog	211, 428, 429
<i>template</i>	434
Business Communication Services	
(BCS)	441, 444
Business event	152, 153
<i>architecture</i>	152
Business object	49, 231, 426
<i>data model</i>	50
<i>design</i>	49
Business object implementation	49

Business object repository	138	Condition	
Business package	421	<i>technique</i>	220, 286
Business partner	175, 195, 208, 292, 313, 355, 396	<i>type</i>	220
<i>employee role</i>	199	Configuration database	192
<i>group</i>	355	Consent	163
<i>master data</i>	196	Consumption views	47
<i>OData API</i>	132	Content Delivery Network (CDN)	70, 474
<i>time-dependency</i>	200	Continuous delivery	126, 127
Business process model and notation	149	Contract accounting	365
Business processes	423	Control object	331
Business role	427, 429	Convergent billing	216
<i>template</i>	211, 429	Convergent invoicing (CI)	361
Business server pages	69	Conversational AI	98
Business service implementation	49	Copy control	212
Business services	56	Core Data Services (CDS)	44
Business topic	421	<i>consumption scenarios</i>	47
Business transactions	231	<i>data control language</i>	51
Business transactions framework	231	<i>entities</i>	45
<i>header</i>	231	Core Data Services (CDS) models	50
<i>item extensions</i>	231	Cost analysis	345
<i>root components</i>	231	Costing-based profitability analysis	340
<i>sets</i>	231	Create, Read, Update, Delete (CRUD)	49, 122
Business user	427, 433, 464	Credit decision support	355
C		Credit events	355
Capacity management	479	Credit management	354
CDS views	117	Cross-System Process Control Framework (CSPC)	390
CDS-based data extraction	155, 254	Cube view	48
CDS-based extraction	155	Custom Analytical Queries application	91
Central procurement	240, 245	Custom business	
Change control	168	<i>logic</i>	119
Change data capture	156	<i>objects</i>	121
Characteristic	340, 394	Customer	
<i>group</i>	192	<i>returns</i>	215
Classification system	187	<i>workspace</i>	424
Cloud BAaI	120, 408	cXML	253
Cloud Connector	146	D	
<i>principles</i>	146	Dashboard	87
Cloud data integration	155	Data	
Cloud Foundry	128	<i>center</i>	453
Cloud-native applications	124	<i>collection</i>	318
Collections management	360	<i>exchange manager</i>	235
Communication		<i>integration</i>	154
<i>channel</i>	449	<i>migration</i>	182
<i>system</i>	144	<i>model</i>	231
<i>user</i>	144, 434	<i>protection</i>	161
Company code	428	<i>replication framework</i>	157, 158
Compliance	490	<i>residency</i>	488
Composite views	46	<i>security</i>	488
Compositions	50	<i>subject</i>	162

Data (Cont.)		Entity (Cont.)	
<i>subject rights</i>	165	<i>relationship model</i>	46
Data Control Language	44	<i>tag</i>	51
Data protection and privacy (DPP)	162, 203	Entity Manipulation Language	52
<i>regulations</i>	92	Event channels	154
Database view	458	Event-based integration	150
Decentralized EWM	289	Exception management	294
Delivery	283	Expense planning	336
Delta extraction	157	Extended warehouse management (EWM)	176, 289
Demand-Driven Material Requirements Planning	273	<i>master data</i>	292
Determine actions	53	<i>monitoring</i>	297
DevOps	126	<i>process automation</i>	297
Direct procurement	242	<i>reporting</i>	297
Direct requirement element	258	<i>technical frameworks</i>	299
Disaster recovery	453, 488	<i>user interface</i>	298
Disclosure control	168	Extensibility	113, 203
Dispute case	358	<i>in-app</i>	113
Dispute management	358	<i>key user</i>	113
Distribution channel	178	<i>side-by-side</i>	123
Division	178	<i>stability criteria</i>	114
Dock appointment scheduling	296	F	
Dunning	351	Failover SAP HANA instance	453
Dynamic Capacity Management	480, 483	Feature control	52
E		<i>global</i>	52
Elasticity	478	<i>instance</i>	52
Electronic Banking Internet		<i>static</i>	52
Communication Standard (EBICS)	372	Field	
Electronic bill presentment and payment (EBPP)	357	<i>extensibility</i>	117
Electronic Data Exchange	138	<i>service</i>	227
Electronic Data Interchange (EDI)	214, 277, 285, 446	Filter	
Embedded analytical applications	87	<i>object</i>	183
Embedded analytics	83, 217	Finance interface	305
Embedded EWM	289	Financial closing	324
Emergency patch (EP)	461	Fixed asset	309
Encryption	169	Foreign currency valuation	324
Enterprise analytics	83, 91	Form template	446
Enterprise Contract Management and Assembly (E-CMA)	328	G	
Enterprise resource planning	23	General Data Protection Regulation (GDPR)	161
Enterprise search	80	<i>technical measures</i>	167
<i>auxiliary views</i>	79	Generally Accepted Accounting Principles (GAAP)	332
<i>personalization</i>	80	Goods	
<i>search scope</i>	80	<i>issue</i>	180, 315, 332, 346
Entitled to dispose	293	<i>receipt</i>	179, 242, 262, 264, 283, 285, 394
Entity		<i>receipt/invoice receipt</i>	109
<i>data model</i>	137		

Greenfield 476
Group reporting 318

H

Handling unit management 285, 294
Handling units 293
High availability (HA) 488
Hotfix collection (HFC) 461
Hybrid analytical applications 87
Hyperscale Providers 453

I

Identity and Access Management (IAM) 425
 architecture 425
 tools 430
Identity Authentication service 452
IDoc 138
iFlows 149
Indirect procurement 244
Information Access (InA) 86
Information retrieval framework 165
Infrastructure vulnerability scanning 489
In-house repair 227, 228
Initial load 474
Inspection lot 283
Integration 137
 middleware 148
Intelligent situation automation 110
Intelligent situation handling 100, 104, 211
International Article Number (EAN) 185
International Bank Account Number (IBAN) 198
International Financial Reporting Standards (IFRS) 307
International Standard on Assurance Engagements (ISAE) 3000 429
Inventory management 179, 311
Inventory valuation methods 312

J

Jupyter Notebook 96
Just-in-time (JIT) 275

K

Kanban 274
Key user 114, 464
Kubernetes 128
Kyma environment 128

L

Latency 473
Ledger 307, 343
Legal contract 328
Legal transaction 328
Lifecycle management 122
Localization toolkit 403, 410
Low-level configuration 271

M

Machine learning 92
 architecture 93
 categorization 98
 embedded 94
 matching 97
 prediction 98
 ranking 97
 recommendation 97
 side-by-side architecture 96
Maintenance event 461
Malware 489
Manage KPIs and Reports application 91
Manufacturing execution systems (MES) 288
Mapping 120
Margin analysis 340
Market
 segment 340, 394
Master
 data 175, 257
 data integration 159
Material
 document 258
 flow system 295
 ledger 311
 master 175
 requirements planning 258, 271
 reservation 263
Microservice 124
Migration object 182
Moving average price 180, 312
MRP area 260
MRP type 271
Multibank Connectivity 352
Multidimensional reporting 87
Multitenancy 454
My Situations app 212

O

OData APIs 137
OData service 56, 73
OData service extensions 117
Online transaction processing (OLTP) 31
Open payables management 348
Open receivables management 350
Operational
 data provider 155
 procurement 240
Outbound
 deliveries 208
 delivery 262, 265, 296
Output management 441
Output Management System (OMS) 444

P

Packing instruction 285, 286
Payables management 347
Payment enqueue processing 348
Payment medium workbench (PMW) 349, 410
Peppol 409
Personal data 162
PFCG roles 425, 434
Phantom assemblies 187
Physical inventory document 265
Physical inventory management 294
Picking 213
Planned independent requirement 258
Planned order 259, 262, 280, 288
Plant 260
Postprocessing Framework 298
Predictive accounting 332
Predictive analytics integrator 219
Predictive MRP 278
Price Control Indicator 180
Pricing 220
Pricing procedure 220
Print queue 442
Printing user 434
Privacy 161
Process orders 259, 346
Procurement 241
 processes 242
Product hierarchies 180
Product master 175, 176, 208, 283, 286, 292
Production
 cost 344

Production (Cont.)

order 259, 262, 288, 346
 supply area 260
 version 261
Profit center 309
Profitability analysis 342, 345, 346
Progressive disclosure 107
Pseudonymization 169
Purchase
 order 240, 262, 394
 requisition 240, 244, 246, 259, 262, 271
Purchasing
 contract 240
 group 179
 info record 240
 value key 179
Python 96

Q

Quality
 certificate profile 283
 Info Record in Procurement 283
 Info Record in Sales 282
 inspection 283, 296
Quant 293
Quick Sizer 476
Quota arrangements 240
Quotation 207

R

Read access logging 168
Receivables management 349
Reference content 415
Release for Delivery 269
Release order 214
Remote API views 47
Remote functional call 138
 queued 138
 synchronous 138
 transactional 138
Replenishment elements 275
Requests for quotation 242
Resource 292, 297
Responsibility management 100, 230, 246
Restricted reuse view 47
Restriction type 435
Right of Use asset 314
Routing 261

S

Sales	207
<i>areas</i>	178
<i>inquiry</i>	207, 211
<i>organization</i>	178
<i>quotation</i>	211
<i>scheduling agreement</i>	214
Sales contracts	207, 213
<i>master contract</i>	213
<i>quantity contract</i>	213
<i>value contract</i>	214
Sales document	209
<i>structure</i>	210
<i>types</i>	210
Sales document category	209
Sales document type	209
Sales order	50, 207, 212, 285, 315, 332, 341, 355
<i>header</i>	50
<i>processing</i>	212
SAP Analytics Cloud	74, 83, 452
<i>embedded</i>	86
<i>story designer</i>	91
SAP API Business Hub	39, 132, 139, 140, 235, 409
SAP Application Interface Framework	140
SAP Ariba	249
SAP Ariba Buying	249
SAP Ariba Cloud Integration Gateway	222
SAP Ariba Network	251
SAP Ariba Sourcing	249
SAP Bank Statement Reprocessing Rules	352
SAP Billing and Revenue Innovation Management (SAP BRIM)	364
SAP Business Suite powered by	
SAP HANA	31
SAP Cash Application	99, 353
SAP Central Business Configuration	419
SAP Cloud Platform	128, 129, 407
<i>connectivity</i>	130
<i>enterprise messaging</i>	139, 151, 153
<i>integration suite</i>	139
SAP Cloud Platform Extension Factory,	
Kyma runtime	128
SAP Cloud Platform Forms by Adobe	452
SAP Cloud Platform Integration	250, 407
SAP Cloud Platform Integration service	149
SAP Cloud Platform Master Data	
Integration service	158, 199
SAP Cloud Platform, Cloud Foundry	
environment	128
SAP Cloud Platform, Kubernetes	
environment	96
SAP Cloud Print Manager	442, 444
SAP Cloud SDK	129, 133
SAP Cloud Security Framework	490
SAP Conversational AI	94
SAP Data Intelligence	94, 221
SAP Digital Payments	374
SAP Document Compliance	406
SAP Field Service Management	237
SAP Fieldglass	251
SAP Fiori	32, 67, 431, 474
<i>app reference library</i>	74
<i>apps</i>	73
<i>launchpad</i>	69, 431, 467
<i>libraries</i>	73
<i>library</i>	73
<i>mobile cards</i>	72
<i>roles</i>	71
SAP Fiori elements	73, 74
<i>analytical list page</i>	75
<i>list report</i>	75
<i>object page</i>	75
<i>overview page</i>	75
<i>worklist</i>	75
SAP Fiori UI	45
SAP for Retail	176
SAP GUI	68
SAP HANA	31
<i>search</i>	79
SAP HANA automated predictive library	94
SAP HANA predictive analytics library	94
SAP liveCache	281
SAP Localization Hub	328
SAP Master Data Governance	248
SAP object types	153
SAP One Support Launchpad	463
SAP R/2	23
SAP R/3	23
SAP S/4HANA	36, 74, 94, 239, 249, 347
API	139
<i>architecture</i>	37
<i>authentication</i>	167
<i>business event</i>	152
<i>GDPR</i>	164
<i>integration</i>	137
<i>machine learning application</i>	97
<i>physical access control</i>	167
<i>search</i>	77
<i>search architecture</i>	77
<i>search models</i>	81
<i>system conversion</i>	201

SAP S/4HANA Cloud	86, 130, 137, 139, 142, 211
<i>activity mapping</i>	427
<i>authorization object extension</i>	427
<i>communication arrangement</i>	145
<i>communication management</i>	142
<i>communication scenario</i>	144
<i>communication system</i>	145
<i>communication user</i>	144
<i>performance</i>	471
<i>restriction field</i>	428
<i>restriction type</i>	428
<i>RFC communication</i>	148
<i>sizing</i>	471
<i>user types</i>	433
SAP S/4HANA Embedded Analytics	84
SAP S/4HANA Finance	352
SAP S/4HANA Sales	207, 209
SAP S/4HANA Sourcing and Procurement	239
SAP Smart Business	88
SAP Smart Business app	73
SAP Solution Builder Tool	417
SAP Supply Chain Management	176
SAP Support Portal	463
SAPUI5	70
<i>freestyle application</i>	73
<i>runtime</i>	77
Scalability	478
Schedule line	212
Scheduling agreement	207, 242, 262
Scikit-Learn	94
Security	488
Security Assertion Markup Language (SAML)	426
Security group	487
Segment	309
Segregation of duties	397, 429, 435
Self-Service Configuration	184
<i>UIs (SSC UIs)</i>	416
Serial number	286
Service business transactions	
integration	235
Service confirmation	225, 227
Service contract	227, 229, 316
Service entry sheet	242
Service operations	225
<i>architecture</i>	225
<i>business objects</i>	226
<i>business partner</i>	229
<i>master data</i>	229
<i>organization units</i>	230
Service operations (Cont.)	
<i>processes</i>	226
<i>service product</i>	230
<i>service teams</i>	230
<i>technical objects</i>	230
Service order	225, 227
Service quotation	227
Service transactions	233
<i>advanced variant configuration</i>	234
<i>partner functions</i>	233
<i>pricing</i>	234
<i>status management</i>	233
<i>transaction history</i>	233
Service-specific data model	57
Shipping and receiving	296
Shipping point	213, 260
Side-by-side extensions	123
<i>custom backend application</i>	124
<i>custom user interface</i>	123
Situation	
<i>indication</i>	105
<i>notification</i>	106
<i>page</i>	107
<i>templates</i>	102
Situation handling, message-based	111
Sizing	475
Smart business	217
SOAP services	138
Solution business	227
Source list	240
Source of supply	240
Spool system	442
Standard	
<i>costing</i>	336
<i>price</i>	180, 312
Stock	
<i>management</i>	292
<i>transfer</i>	207
<i>transport orders</i>	287
<i>type</i>	294
Stock-identifying fields	266
Storage	
<i>bin</i>	291
<i>location</i>	260
<i>type</i>	291
Strategic procurement	242
Structured query language (SQL)	44
Super administrator	464
Super BOM	190
Supervisory control and data acquisition (SCADA)	288
Supplier invoicing	347

-
- Supply
 - assignment* 269
 - protection* 269
 - S-user ID 476
 - Sustainability 481
 - System Landscape Transformation
 - Replication Server (SLT) 384
- T**
-
- Tax 326
 - Technical architecture foundation 43
 - Telegram 295, 300
 - Tenant 451
 - Tenant database 455, 462
 - TensorFlow 94
 - Topc filters 154
 - Total cost of implementation (TCI) 415
 - Trading platform integration (TPI) 380, 452
 - Transaction
 - /SAPAPO/MATI* 176
 - /SCWM/MATI* 176
 - DRFIMG* 183
 - MDS_LOAD_COCKPIT* 202
 - MM01* 176
 - MM02* 176
 - MM03* 176
 - MM41* 176
 - MM42* 176
 - MM43* 176
 - Transactional processing-enabled
 - applications 48
 - Transactional view 46, 50
 - Transmission control 168
 - Transportation units 293
 - Trigger object 102
- U**
-
- Unified connectivity 168
 - Units of Measure 178
 - Universal Journal 306, 340, 342, 394
 - Universal key mapping service 248
 - Universal unique identifier 51
 - Update 461
 - Upgrade 461
 - US Generally Accepted Accounting
 - Principles (US GAAP) 307
 - User experience (UX) 67
 - User propagation 145
- V**
-
- Validation 120
 - Valuation area 180
 - Value-added service 296
 - Variant configuration 188, 190
 - object dependencies* 191
 - View Browser application 90
 - Virtual Data Model (VDM) 43, 155, 197, 217, 234
 - structure* 45
 - Virtual elements 58
 - Virtual private cloud 486
 - Virtual private network (VPN) 40
 - Vulnerability announcement service
 - (VAS) 489
- W**
-
- Warehouse
 - automation* 300
 - control unit* 300
 - monitor* 298, 299
 - number* 291
 - request processing* 296
 - Wave management 296
 - Web Client UI 237
 - Web Dynpro for ABAP 68, 119
 - Work breakdown structure (WBS) 331
 - Work center 261



Thomas Saueressig, Tobias Stein, Jochen Boeder, Wolfram Kleis

SAP S/4HANA Architecture

520 Pages, 2021, \$79.95

ISBN 978-1-4932-2023-6

 www.sap-press.com/5189



Thomas Saueressig is a member of the Executive Board of SAP SE. He leads the SAP Product Engineering area and has global responsibility for all business software applications. This includes all functional areas from product strategy and management to product development and innovation as well as product delivery and support.



Tobias Stein is a development senior executive and leads the SAP S/4HANA architecture area. Tobias is a software architecture expert and has more than 22 years of experience with SAP's ERP (SAP Business Suite and SAP S/4HANA) application and solution architectures. As a member of SAP's global leadership team, he is heading the central architecture unit in the SAP S/4HANA development organization. He is a TOGAF-certified Enterprise Architect.



Jochen Boeder is vice president of the SAP S/4HANA architecture area. He has more than 15 years of experience in the architecture of SAP business applications. At SAP, he leads a team of senior architects in the SAP S/4HANA engineering organization. He has co-authored several IT books, most prominently, The Architecture of SAP ERP.



Dr. Wolfram Kleis is a software architect in the SAP S/4HANA architecture area. He joined SAP more than 20 years ago and held several positions as developer and architect. Most recently he engineered cloud applications for Internet of Things before joining the central SAP S/4HANA architecture unit. He has a strong passion for describing software architecture and is the author of the SAP HANA chapter in the book The Architecture of SAP ERP.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.