

Jo Weilbach, Mario Herger

SAP[®] xApps

and the Composite Application
Framework

Contents

Preface	9
Acknowledgements	11
Introduction	13
1 SAP xApps—Basic Principles	17
1.1 Market Situation and Opportunities	17
1.2 The Evolution of Business Applications	20
1.2.1 Traditional Development and Architecture	21
1.2.2 Web Services, Service Oriented Architecture (SOA) and Enterprise Services Architecture (ESA)	23
1.2.3 Composite Applications	30
1.3 Characteristics and Definition of SAP xApps	33
1.3.1 Composite Applications from SAP	33
1.3.2 SAP xApps	35
1.3.3 Properties and Requirements of SAP xApps	36
1.3.4 SAP Composite Application Framework	38
1.3.5 Advantages of a Close Integration with SAP NetWeaver	39
1.4 The Developers of SAP xApps	40
1.4.1 SAP as a Developer of SAP xApps	40
1.4.2 The SAP xApp Partner Program	40
1.5 Ideal Procedure for the Creation of an SAP xApp	42
1.6 First Implementation Examples	45
1.6.1 SAP xApp Resource and Portfolio Management (SAP xRPM)	46
1.6.2 SAP xApp Emissions Management (SAP xEM) as an Example of a Partner xApp	49
1.7 Possible Decision Criteria for Developing an SAP xApp or a Custom Composite Application	51
1.7.1 Business Criteria	51
1.7.2 Criteria Determined by the Characteristics of the Business Process	51
1.7.3 Criteria Determined by the Type and Source of the Information	53

2 The Architecture of SAP xApps 55

2.1	SAP NetWeaver Overview	55
2.2	SAP NetWeaver Components	56
2.2.1	Application Platform	57
2.2.2	Process Integration	61
2.2.3	Information Integration	64
2.2.4	People Integration	74
2.2.5	SAP Solution Lifecycle Management	80
2.3	SAP Composite Application Framework	81
2.3.1	Vision	81
2.3.2	Comparison	82
2.3.3	SAP Composite Application Framework—Principles	85
2.3.4	General Structure of the CAF	86
2.4	Architectural Model of SAP xApps	94

3 The SAP Composite Application Framework 97

3.1	Introduction	97
3.2	Scenario xFlights	98
3.2.1	Scenario	98
3.2.2	Modeling	99
3.2.3	Building an Application in the Classical Way	102
3.3	Architecture and Tools of the SAP Composite Application Framework	105
3.3.1	General Remarks	105
3.3.2	Installation	109
3.3.3	Life Cycle of a Composite Application	109
3.3.4	Composite Application Project	110
3.3.5	CORE	112
3.4	External Services	114
3.5	Entity Service	115
3.5.1	Introduction	115
3.5.2	Modeling	116
3.5.3	CORE Services	122
3.5.4	Persistence	126
3.5.5	Generation, Build and Deployment	132
3.5.6	Testing	134
3.5.7	Authorizations	135
3.6	Application Services	138
3.6.1	Introduction	138
3.6.2	Modeling	139
3.6.3	Coding Examples	142
3.6.4	CORE Services	143
3.6.5	Generation, Build, Deployment and Testing	143

3.7	User Interface	144
3.7.1	Web Dynpro	144
3.7.2	Patterns	145
3.7.3	Web Dynpro Foundations	155
3.7.4	xFlights Example	156
3.7.5	Special UI Elements	159
3.8	Process Modeler	162
3.8.1	Introduction	162
3.8.2	Guided Procedures	163
3.9	Business Intelligence Integration	166
3.9.1	Types of Integration	166
3.9.2	Extraction	167
3.9.3	Display	172
3.9.4	Retraction	173
3.9.5	Navigation	173
3.9.6	Guided Procedures	175
3.10	Knowledge Management Integration	175
3.10.1	TREX	176
3.10.2	Discussion	177
3.10.3	Notification & Subscription	177
3.11	Additional Integrations	177
3.11.1	Exchange Infrastructure	177
3.11.2	Replication und Synchronization	178
3.12	Supply and Maintenance	178
3.12.1	Configuration	178
3.12.2	Java Development Infrastructure	178
3.12.3	Projects	180
3.12.4	Testing and Monitoring	183
3.12.5	Debugging	183
3.12.6	Documentation and Help	183
3.12.7	Internationalization	184
3.12.8	Upgrade and Versioning	184
3.13	Modeling Guidelines	185
3.13.1	Data Types	186
3.13.2	Entity Services	188
3.13.3	Application Services	188
3.13.4	User Interface	188
3.13.5	Naming and Other Conventions	189

4 Implementation Examples 191

4.1	SAP xApp Product Definition (SAP xPD)	191
4.1.1	Introduction	191
4.1.2	Business Case	192
4.1.3	SAP xPD as a Solution	200
4.1.4	Summary	221

4.2	SAP xApp Cost and Quotation Management (SAP xCQM)	222
4.2.1	Introduction	222
4.2.2	Business Case	223
4.2.3	SAP xCQM as a Solution	232
4.2.4	Further Development of SAP xCQM and Outlook	253
4.2.5	Summary	254

5 Outlook: Applications and Application Development 257

5.1	Development	257
5.2	Paradigm Shift	259
5.3	Facts	261
5.4	Overall Picture and Solution	264

A Appendix 267

A.1	CORE Objects for CAF 1.0	267
A.2	Operation Types	270
A.3	User Interface Links	270
A.4	Tables	273
A.5	Property Rules for Attributes	275

B Glossary 277

C Sources and Further Reading 283

C.1	SAP Sources	283
C.2	Books and Articles	284
C.3	Internet	284

The Authors 287

Index 289

Preface

In 2003, SAP introduced a revolutionary vision for the future of business applications: Enterprise Services Architecture (ESA). Many of the world's top-performing enterprises adopted ESA in the first year after it was announced, reducing the cost of IT maintenance, and freeing up resources for innovation.

ESA is a radical shift in technology architecture. The last decade was dominated by the client/server-and-database approach to systems, an innovation also pioneered by SAP with the introduction of SAP R/3 in the early 1990s. ESA is based on a service-oriented approach to applications. By eliminating much of the "impedance mismatch" between systems and business processes, ESA can yield much greater business agility while simultaneously achieving substantial reductions in total cost of ownership (TCO).

At the heart of ESA is a new platform. The platform of the 1990s was the client operating system, server operating system, and relational database. Applications were each designed around their own information needs, often with complicated and expensive point-to-point integration across databases. This approach to application integration resulted in integration costs that consumed between 70% and 90% of the IT budget of typical enterprises, according to leading analysts such as Gartner Group.

Client/server applications are designed with a dedicated client and server, each server used by a single application, and with integration performed by point-to-point data synchronization. ESA applications are designed with shared servers (called "components") used by many applications (called "composites"). The components expose a standard set of services, understandable to enterprise business processes. These standard services, such as "create purchase requisition" or "order goods from inventory," are called enterprise services.

To realize this ESA vision, SAP introduced SAP NetWeaver in 2003. A breakthrough platform based around the concept of integration, SAP NetWeaver has had the fastest ramp-up of any platform in history, having been adopted by well over 1,500 of the world's leading enterprises by the time this book went to press. By building integration capabilities into each component of the SAP NetWeaver platform, SAP can dramatically reduce TCO. The resources saved can be invested in innovation, used to catch up on the application backlog, or returned to the business. Adopt-

ing SAP NetWeaver was the first step on the road to ESA for many SAP customers.

One of the most unique capabilities of SAP NetWeaver is called "Composite Application Framework" (CAF), designed to reduce the cost of developing and deploying composite applications. CAF is the basis on which SAP develops and delivers composite applications, including SAP xApps™. SAP xApps are "packaged-innovation" solutions that deliver next-generation business practices with rapid implementation, very low TCO, and competitive differentiation. SAP xApps are all designed using ESA, and are all "powered by SAP NetWeaver."

This book captures SAP's experiences in developing both composite-application products and custom composite-application projects. The authors designed, developed, and implemented these applications at customer sites. You'll learn everything you need to know about composite applications and SAP xApps in this book, from the greatest experts in the field.

We hope you will enjoy reading this book as much as we enjoyed creating this technology and writing this book. If you'd like to learn more about the topics covered in this book, please visit SAP Developer Network at <http://sdn.sap.com/> you'll have access to the latest information, technologies, and special offers to help you on your journey to composite applications.

Dennis Moore

Senior Vice President, xApps

SAP Labs LLC

Acknowledgements

Although only two people are named on the cover of this book as its authors, it would never have been possible for us to write it without support from and many discussions with our colleagues, partners, customers, and friends. This support went far above and beyond the call of duty, especially since the long distances between the people involved (Palo Alto in California, Sofia in Bulgaria, and Walldorf in Germany) and the resulting time-zone differences meant that telephone conversations had to be held either very early in the morning or very late at night.

It would be impossible to name everybody who contributed to the success of this book. Therefore, we would like to mention just a few individuals and to acknowledge them in gratitude and respect. They are:

Rituparna Reddi, who, despite her already tight schedule, spent many long nights in Walldorf describing user interfaces, helped us to understand the material, and thus regularly missed her last bus home.

Malte Kaufmann, who, without grumbling, provided detailed answers to dozens of e-mails containing modeling questions, and, in several telephone conversations, supplied information and answers to questions regarding functionality, technology, and application.

Jörg Schleiwies, who, without blinking, answered our toughest questions about integrating unstructured information in CAF, and explained all the relevant aspects in detail.

Frank Rakowitz, master of services.

Kalin Komitski, who answered and commented on our modeling questions promptly, in detail, and with Bulgarian flair.

Tim Bussiek, Steffen Kübler, Thomas Anton, Jürgen Kremer and **Werner Aigner**, who gave up some of their scarce free time to proofread this book and give extensive, valuable feedback, and in particular, helped us to see the CAF tree in the SAP NetWeaver forest.

Jürgen Hagedorn and **Katharina Rock**, who gave us the freedom to write this book and were always ready with help and advice.

Gunther Piller and **Andreas Henke**, who always knew the answers to our SAP xPD questions.

Martin Botschek: a quick code example? Martin was always ready. Need a quick test environment? He just seemed to pull it out of a hat.

Stephan Böcker, who was always there for us with answers to our questions about the SAP xApps Partner Program.

Yury Golovenchik, Alexander Efimchik, Alayxey Palayzhay and Ihar Lakhadynau, who fell in love with the xFlights demo.

Natasha and **Katrin**, who kept us motivated, and always provided comfort and encouraging words when writer's block set in.

... And of course, everyone else from the world of CAF and xApps, without whom we would have had nothing to write about.

Mario Herger and Jo Weilbach

Remote Function Calls

RFCs represent the most frequently used method to communicate with an ABAP-based SAP system. RFCs are interfaces that enable a communication with programs (so-called function modules) of an SAP system from outside.

The metadata of RFCs can be imported with the CAF and are then made available to the entity and application services. During import, all methods and their input and output parameters are notified to the CAF and are made available for the CAF services.

Metadata import

ABAP systems also use *transactional RFCs*. These run asynchronously. That means that although data which this RFC receives are written in a database table the transaction can only be completed by an explicit `COMMIT WORK`.

tRFC

Web Services

Web services have become widely accepted in recent times as the interface standard for communication on the World Wide Web across application and system boundaries. Independent of platform, program language, and technology, Web services enable systems and applications to exchange data and to start actions of any type. For the Web services, WSDL (Web Service Description Language) files are imported that can be retrieved from a directory or a UDDI⁴ server.

External services (and entity and application services that we will describe in further details later on) are the cornerstone of an Enterprise Service Architecture and are used to flexibly create business applications.

3.5 Entity Service

3.5.1 Introduction

The Entity Service Modeler is a tool that can be used to model Entity Services, their attributes, methods, relationships with other Entity Services as well as their general properties and behavior. The task of an Entity Service is to save and access application data. The data can be made accessible in a local storage in a backend such as an R/3 system or a Web service. You will find more information on the different persistences in Section 3.5.3.

⁴ You can find information on Universal Description, Discovery and Integration (UDDI) at <http://www.uddi.org/>.

Creating and changing

You can call the Entity Service Modeler in SAP NetWeaver Developer Studio by creating a new or changing an existing Entity Service. Both actions open an editor that provides all properties that can be modeled, as well as attributes and relationships of the Entity Service, in several tabs as well as in the Properties window. In addition, this editor enables you to make changes depending on your selections. The (technical) name of an Entity Service can also be changed at a later stage and it will be registered on the name server during the check-in phase into the Design Time Repository (DTR, see also description of DTR in later chapters) provided the development takes place in the Java Development Infrastructure (JDI).⁵

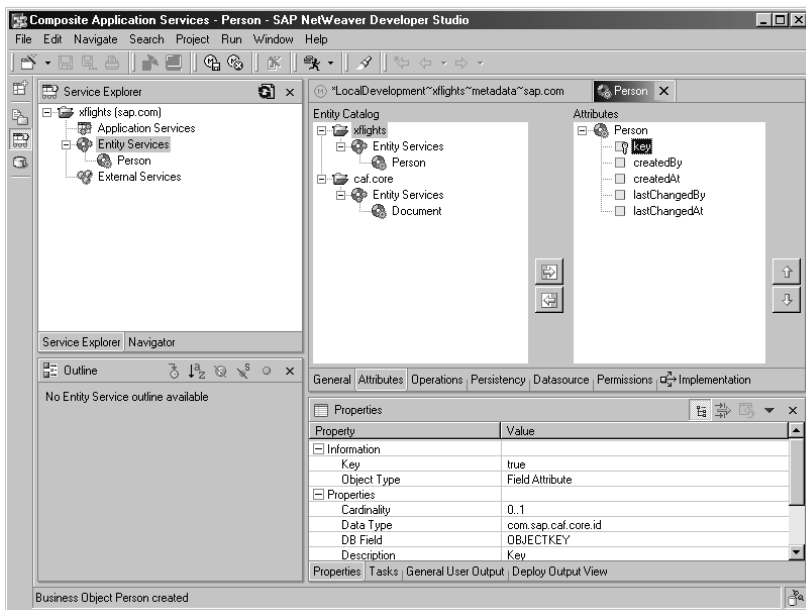


Figure 3.8 Entity Service Modeler

Comparison to ABAP

In the ABAP world, the name server corresponds to the catalog of repository objects (incl. Table TADIR) and the global TADIR (for cross-system and cross-release developments in ABAP systems).

3.5.2 Modeling

Attributes

The first general properties of a new Entity Service are the name, the package, and a short descriptive text. The Entity Service is automatically assigned a unique number ("GUID") and creation and change dates as

⁵ If the name already exists, the check-in is canceled and an error message occurs.

well as creation and change users are logged simultaneously. Thus the Entity Service is uniquely described.

The attributes and properties affect both the number of tables generated and the scope of the coding which is generated as an Enterprise JavaBean (EJB).

Several entities have already been identified for our xFlights scenario, and Table 3.1 shows a list of selected entities.

Entity	Selected attributes of the entity
Person	LastName, FirstName, Address
Flight	CarrierID, FlightNumber
Airport	Name, Altitude, Latitude
Plane	PlaneType, MaxSeats, FuelUsage
Address	Streetname, ZIPCode, City, Country
FlightSchedule	DepartureTime, ArrivalTime, FlightNumber
Booking	Person, Flight, Date
Business Partner	lastName

Table 3.1 Selected Entities for xFlights

As already mentioned the entity **Person** is described by last name, first name, title, address, telephone numbers(s) and other attributes. On closer consideration it becomes apparent that these attributes have different characteristics. The person has only one last name but can have several first names. The person can also have several addresses and telephone numbers.

Characteristics of attributes

Addresses and telephone numbers can also be divided into several different attributes. Thus the address consists of country, region, city, zip code, street, and street number. The telephone number can be broken down into country code, area code and extension.

This requires different types of modeling for which it is important to know which options are available to map this data. The CAF provides three additional types of attributes here in addition to the standard attributes which cannot be changed: simple, complex and entity-service attributes.

Standard attributes

When creating an Entity Service five attributes are automatically created. These attributes are: "key," "createdAt," "createdBy," "lastChangedAt" and "lastChangedBy." The attributes cannot be changed or deleted and possess the cardinality 1..1 for "key" and 0..1 for the remaining four. Every Entity Service thus contains a main table with at least five of these attributes, regardless of whether it is locally or remotely persisted.

Meaning The meaning of these fields becomes clear when you consider the names. "Key" which is defined as GUID, is the unique primary key for this Entity Service, "createdAt" and "lastChangedAt" provide the change history of the data records in date fields, and "createdBy" and "lastChangedBy" are used to store the change user as a 255-digit key of the UME.

The "lastChangedAt" attribute is also important because it is used for the delta extraction of data into the BW. You will find a more precise description of the BW extraction in Section 3.9.

Rules for attribute properties The appendix of this book contains a tabular overview of all rules which itemize the property changes to all attribute types and describe the dependencies.

Simple Attributes

Simple attributes are generally simple data fields. In a table they would correspond to a column. In the **Person** example the attribute "last name" would be a simple attribute.

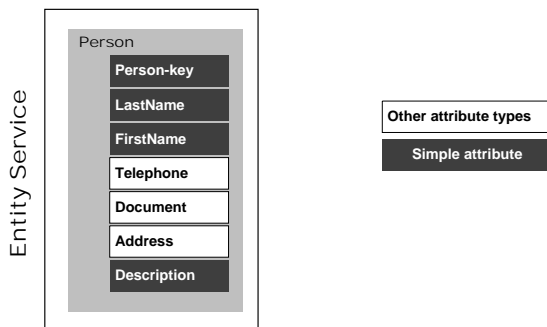


Figure 3.9 Simple Attributes and their Mapping in the Database

Language and time dependency In the CAF world, a simple attribute can also be identified as language or time dependent which entails the creation of an additional language or time-dependency table. However, such an attribute can only be identi-

fied as language dependent if it is of the basic type "string" and has the cardinality 0..1.

Only one language table is created per Entity Service. In the language table, an additional field is used for every language-dependent attribute. This field is no longer available in the main table. In Figure 3.10 this aspect is illustrated using the field "Description."

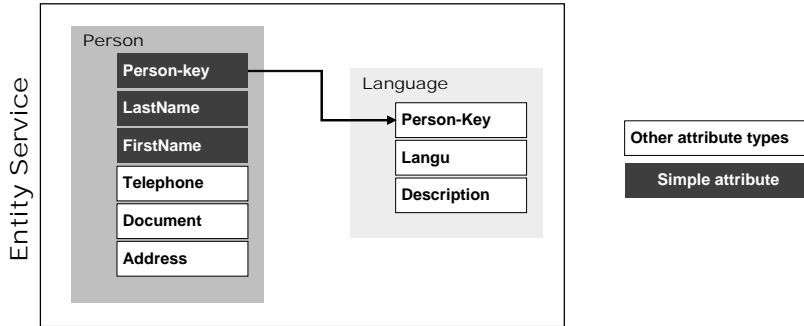


Figure 3.10 Language-Dependent Simple Attributes and their Mapping in the Database

Simple attributes can possess the cardinality 0..1 and 0..n or 1..1 and 1..n respectively if they were entered as mandatory attributes at the same time (the "mandatory" property is set to "true"). Regarding the database, the cardinality 0..n and 1..n means that an additional table is created with a primary key and an attribute. For this and all additional tables which are created for an Entity Service for the persistence, the name can be changed in the **Persistency** tab. In the main table itself a foreign key is then created which is a GUID for the CAF. In Figure 3.11 this is illustrated using the field "FirstName."

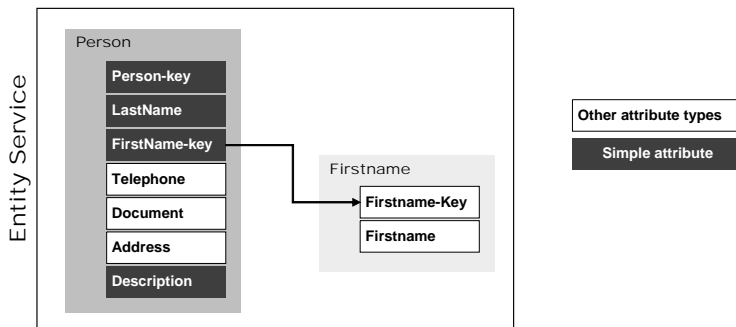


Figure 3.11 Simple Attributes with Cardinality 0..n or 1..n and their Mapping in the Database

Simple attributes can also be defined as keys by selecting the "key" property. This makes them automatically mandatory input fields and the cardinality is set to 1..1. This cannot be changed subsequently. However, in the database these attributes are not key fields but rather receive a unique index. The standard attribute "key" is created as the only key field in the database itself.

Attributes which were defined as mandatory input fields or as keys are automatically transferred as arguments into the `CREATE` method of the Entity Service.

Modeling tip As a rule of thumb, simple attributes are used if the attribute should not be divided into additional sub-attributes and cannot exist independently from the referenced master data record. It should only be identified as language dependent if the values entered have a commentary or description field character. Reusable language-dependent texts such as "color" should be modeled as separate Entity Services.

Specific data types for attributes as are required for using currencies and units will be described in Section 3.13.1.

This attribute type and all other types can be sorted in any way in the **Attributes** tab of the Entity Service Modeler.

Complex Attributes

Complex attributes have several data fields. The data is stored in an additional table in the database. In the **Person** example, the "Telephone" attribute is a complex attribute which consists of the fields "Country code," "Region code," and "Extension."

A complex attribute can also be created with the cardinality 0..n (one person has several telephone numbers). The sub-attributes themselves can only be simple attributes and can have a cardinality of 0..1. A sub-attribute cannot be a mandatory attribute (i.e. the "Mandatory" property cannot be set to "true"). Complex attributes can neither be language-dependent nor defined as mandatory input fields.

Regarding the database, this means that an additional table with several attributes is created. In the attribute of the main table, a foreign key is then stored which is a GUID for the CAF.

Modeling tip As a rule of thumb, complex attributes are used if the attribute is to be divided into additional sub-attributes and cannot exist independently from the referenced master data record. This means that, for instance, a

telephone number makes no sense in most cases without the corresponding person.

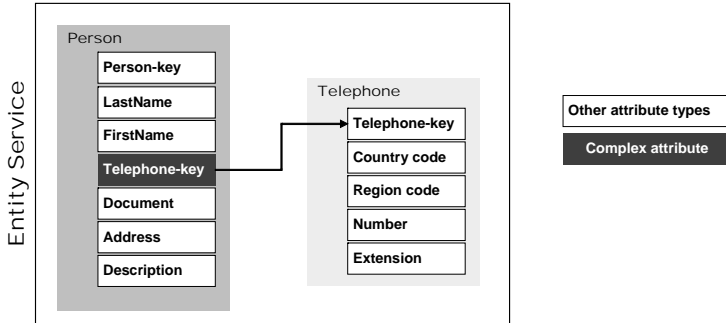


Figure 3.12 Complex Attributes and their Mapping in the Database

Entity Service Attributes

Entity Services can have other Entity Services as attributes. To create such an attribute a user can simply drag an Entity Service from the "Entity catalog" to the attributes in the "Attributes" tab. Technically speaking, this results in a foreign-key relationship. In the **Person** example the "Address" attribute would be defined as a separate Entity Service that can in turn have simple ("Street"), complex ("Geographical coordinates"), and Entity Service ("City," "Country") attributes. This means that one and the same address can therefore be used both separately and in other Entity Services, for instance as a company address and as an address for several employees.

Entity-service attributes have a cardinality of 0..1 or 0..n. They cannot be set as language dependent at this point. This must be defined in an Entity Service, which is used as an attribute. Entity-service attributes cannot also be defined as keys in this Entity Service.

Cardinality

In the database, an additional join table is created as a result. Thus another table is implemented between the main table of the Entity Service and the Entity Services used as attributes. This join table contains three fields: a separate primary key, the foreign key for the Entity Service, and the foreign key for the Entity Service attribute. Although both of the foreign keys alone would be sufficient for uniqueness, it was decided to create an additional key.

As a rule of thumb, entity-service attributes are used if the attribute is to be divided into additional sub-attributes and can exist independently from the referencing master data record.

Modeling tip

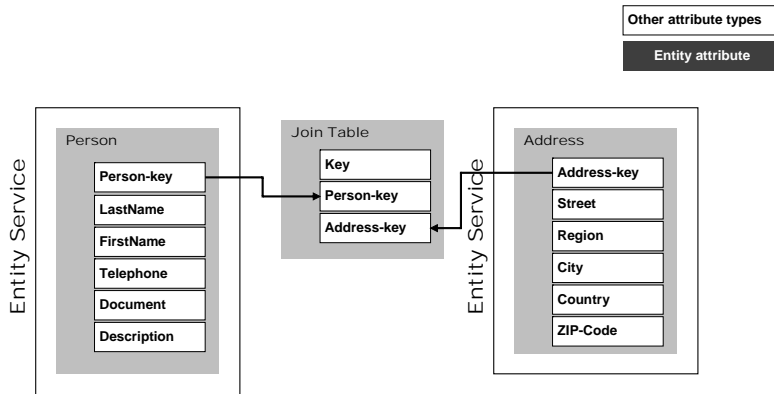


Figure 3.13 Entity Service Attributes and their Mapping in the Database

Cascading deletion

Although an instance of an Entity Service attribute can exist independently of the referencing master data record, a cascading deletion is sometimes necessary. Therefore it is possible to define the Entity Service attribute as an association or composition in the properties section. If you select Composition, referenced data records in the Entity Service attribute are also deleted when carrying out deletions in the referencing Entity Service.

Figure 3.13 shows that when modeling with the CAF, extensive database structures are created very quickly.

Custom Appends

Enhancement project

In the current version of CAF SAP, xApps are supplied as Java archives but not as Java projects with the metadata. In order to be able to perform changes to the application that go beyond the configuration, an enhancement project can be created. In such an enhancement, project, custom appends can be created that enable customers to model additional attributes for the Entity Services supplied.

3.5.3 CORE Services

As already mentioned, with CAF some CORE Entity Services are supplied in the CORE project by default. These services are frequently used and they are cross-application entities that can in general only be changed to a limited extent by application developers. SAP is planning to add an entire range of Entity Services to this project in future versions of CAF.

As a matter of fact, there are already more Entity Services supplied with CAF than those described in this chapter. These mainly involve persis-

tence for business rules, permissions, object metadata etc. required during runtime. These Entity Services are available to the CAF only internally and during runtime. They are installed with the CAF library.

Along with CAF the following additional CORE Entity Services are provided:

**CORE services
supplied**

- ▶ Category
- ▶ CategoryValueSet
- ▶ Document
- ▶ Discussion
- ▶ DiscussionRoom
- ▶ Principal
- ▶ Topic

Category

Instances of entities can be categorized using the Entity Service **Category**. For example, the passengers can be divided into different categories according to "menu requirements," "smokers/non-smokers," "frequent flyers," etc. Several values can exist for each category, and these can be selected via the Entity Service attribute "CategoryValueSet." The category itself is modeled as language dependent.

The assignment of categories to data records is carried out via the UI pattern **Classification Assignment**. Maintaining the value list is carried out either through the *KMIndex UI* or the values are uploaded from SAP Knowledge Management. In either case, the values are stored in the local persistence. You can find the link for this UI in the appendix.

**Assignment to
data records**

CategoryValueSet

This Entity Service is an attribute of **Category** and contains the value list for different categories. If, for example, a category is called "Menu requirements," the lists of values can contain "Vegetarian," "Kosher," "Meat," and "Pasta." The list of values is also modeled as language dependent.

Maintaining the list of values is also carried out through the *KMIndex-UI*. The values are stored in the local persistence. By using this UI, existing taxonomies can also be loaded from SAP Knowledge Management into the local persistence for further use.

Category and **CategoryValueSet** together form a two-level hierarchy that can be used to categorize data records.

Modeling tip In contrast to the **CustomEnumType** (see also Section 3.7.5), which is displayed as a dropdown field in the UI, a UI pattern must be used to categorize a data record. To do this, a more flexible two-level hierarchy is provided.

CategoryService

As one of the last changes, the hierarchy functionality has been added to the CAF support packages. This functionality enables you to create hierarchies with as many levels as you wish. In addition, several hierarchies can also be created for the same values. The assignment to the object instances is carried out in the UI pattern **Classification**.

Document

By using the **Document**, any file types can be checked in to SAP Knowledge Management, where they can be requested from and edited. Typically, these file types are Word documents, text files, PowerPoint files, PDFs, and so forth. If this CORE Entity Service is used as an attribute for another Entity Service, a field of the data type `rid` is created in the main table for the Entity Service. This data type then has a foreign key relationship with the document.

The Entity Service **Document** has the following attributes:

- ▶ `Key` (`String`)
- ▶ `parentFolder` (`String`; 0..1; key attribute)
- ▶ `parentFolder` (`String`; 0..1; key attribute)
- ▶ `title` (`String`; 0..1)
- ▶ `description` (`String`; 0..1)
- ▶ `link` (`String`; 0..1)
- ▶ `contentLength` (`long`; 0..1)
- ▶ `contentType` (`String`; 0..1; contains the MIME type information)
- ▶ `relatedObjectRids` (`String`; 0..n)

In the application the UI pattern **Attachment** is used as the user interface.

Discussion

If **Discussion** is used as an attribute of an Entity Service discussions, users will be able to discuss individual instances. These postings are stored in KM.

DiscussionRoom

In contrast to the Entity Service **Discussion**, **DiscussionRoom** enables you to restrict the access to a discussion. You can grant authorizations to those who are permitted to take part in the discussion. This Entity Service also provides the functionality to send invitations to users to participate in a discussion.

Principal

Principals are used in the Entity Service **DiscussionRoom** and represent the individuals invited, i.e. the users authorized for this **DiscussionRoom**.

Topic

This service is used by the Entity Service **Discussion**. Each discussion can have one or several topics.

Currency

This service is used internally in order to attach currencies to numerical attributes.

ExchangeRate

This service is used internally in order to execute conversions into other currencies on attributes with attached currencies.

UnitOfMeasurement

This service is used internally in order to attach units of measurement to numeric attributes.

UnitConversionSimple

This service is used internally in order to be able to perform simple conversions on attributes with units of measurement.

3.5.4 Persistence

Naming conventions

As already seen the modeling of an Entity Service has major effects on the persistence. Deciding on a specific attribute type can be critical for additional tables in the local database. The created tables follow a naming convention, which has the prefix XAP_ and has a generated key as its actual name. Due to length restrictions for databases supported by SAP, the table names cannot be longer than 18 characters. For this and other reasons, a decision was made to assign the name of tables in this form instead of generating any "meaningful" names. Other reasons were the potential conflicts between attribute tables with the same names in different Entity Services, conflicts when installing several xApps with the Entity Services bearing the same names, and also the ever-present temptation among application developers to directly access the tables. Instead of this direct access, the corresponding APIs of the Entity Services should be used.

The tables with the rather cryptic names of XAP_<14-digit ID> can be renamed in the modeler by the developer. The names are registered on the name server during check-in to the DTR. If the name already exists, the check-in procedure cancels with an error.

Other factors affect the table structure. In addition to the attribute type, its data type and the key property, these factors are the language dependency, time dependency, the cardinality, and the decision concerning the use of a local or a remote persistence. A precise explanation of these factors follows.

Language Dependency

Language table

If an attribute is referred to as language dependent, a language table is created that contains the foreign key of the data record, the language, and the corresponding text as fields. For each language-dependent attribute, an additional field is created in this language table. Depending on the user log-on language, the text is selected and displayed in the corresponding language during the application runtime.

An attribute which was identified as language dependent cannot have a cardinality of 0..n or 1..n.

Time Dependency

Information as to when a value was valid can be mapped through the time dependency of attributes. The "last name" or the "address" of a per-

son can be time dependent. This property is stored in an additional table that contains the fields "Key," "Value," "Valid from," and "Valid until." Depending on the period of validity of the request, the corresponding value is selected from the time-dependency table and then displayed.

Cardinality

If there is a 0..n or 1..n relationship between the Entity Service and the attribute, additional tables will only be created in cases where this modeling cannot be mapped by the existing table structures. An attribute that was identified as mandatory ("Mandatory" property is set to "true") can only have cardinalities of 1..1 or 1..n.

Remote and Local Persistence

In an Entity Service with local persistence, at least one table is always created. As already mentioned, the number of tables can vary according to the type of modeling. If the Entity Service was already deployed once on the J2EE server (which means that tables were created in the database) and if subsequent changes are then made to the attributes affecting the persistence (i.e. the tables), the following cases can be differentiated:

Subsequent changes

► Positive changes

Attributes are added. In a new deployment these attributes are generated as additions to the existing tables. The data remains in the tables.

► Negative changes

Attributes are removed. This is not permitted. During deployment at the latest, the J2EE server sends error messages and the deployment cancels.

Subsequent changes that have an impact on the language and time dependencies and the cardinality are only possible to a limited extent. The installation must previously be "undeployed," or deleted on the J2EE server.

So far only the local persistence was taken into consideration. In addition to saving data in the local database on the J2EE server, the CAF also provides the option to "remotely" read the data, to change it, save it, and to delete it. In this context "remote" means that data is stored in a back-end system, i.e. a downstream system. Back-end systems in CAF are all those systems whose data, is separated from the local CAF database, no matter how it is stored.

Remote persistence

Mixed persistence In an Entity Service, a "mixed" persistence, i.e. a local one as well as a remote one can be set as an attribute by using other Entity Services. For example, a **Person** Entity Service can be made completely locally persistent with its simple and complex attributes. On the other hand, the "address" attribute, which is an Entity Service itself, can be made persistent in a remote form in a back-end system. The same applies in reverse: Both the data of the Entity Service and all its attributes can either be stored completely locally or remotely.

In fact, an Entity Service which is identified as remote always has just one local table in which the standard attributes and the information on relationships with other Entity Services are stored. This means no changes have to be performed on the backend systems.

Modeling tip The use of a remote persistence from an Entity Service requires mapping to an external service. For each method (`create`, `read`, `update`, `delete` and the arbitrarily definable `findBy`-methods), an individual external service can be referred to. The methods of the external service are then mapped to the methods and fields of the Entity Service. In the current version of the CAF, there are still some functional limitations. For example, the developer has to ensure that the data types match.

In general, it doesn't make sense to use transactional RFCs in the Entity Service. It is better to use these in the application service, as otherwise another `COMMIT` command (in our specific case the ABAP command is `COMMIT WORK`) must be released.

Distributed commits If you take a closer look at "mixed" persistences, the problem of a "distributed commit" emerges. By this we mean that the data to be written into distributed systems actually arrives there without any interruption. Nothing is worse for data consistency than having the write access to a system canceled due to concurrent accesses, locks set, unavailable systems, or inconsistencies while the relevant data was updated in the other system. If persistence exists in one single system only, you have more or less complete control and therefore always have the option of a rollback. However, to ensure a clean update in distributed systems, all participating systems have to permit a two-phase commit. An example would be to first check in the course of a test update if the data can be written so that it can be actually written in a subsequent update. Unfortunately, most systems do not offer any two-phase commit, and some of the external services used do not even provide rollback functionality.

The CAF, or to be more precise SAP NetWeaver, for these reasons currently provides no options for using generic functionality during a distributed commit. This should not be a reason for ruling out the creation of composite applications with SAP xApps, as in many cases either no distributed commit is required or it can be bypassed by taking other actions.

Access to the Local Persistence

But how do the operations access the local database? This occurs "indirectly" using Java Data Objects (JDO). JDO is an API developed by Sun in order to be able to access data from any data sources through Java. JDO makes access to the data sources transparent for the developers and requires no SQL knowledge. JDO

JDO is not used directly in the Entity Services but via the CAF-specific **DataAccessService**. During creation, two additional files with the endings .jdo and .map are created. The .jdo file contains the relationships between the Java class and the database table and is required when the CAF project is being built. During the build, the **PersistenceCapable** interface is implemented in the Entity Services on the basis of the .jdo file in order to enable the JDO use.⁶ The .map file can then read out these relationships during runtime.

Access to the Remote Persistence

In order to receive data from a back-end system or to save it there, external services are used and mapped to the operations. By mapping we mean the assignment of methods and parameters of the Entity Service to methods and parameters of the external services. External Services

Not all operations of an Entity Service must be mapped to the same one, to a different one, or to any external services. It is conceivable that the read method `READ` is mapped to an external service A, the `FINDBY`-method `XY` to an external service B, and the write method `CREATE` to no external service at all, but that it stores the data locally. A practical example illustrates the mapping of the `FINDBY` methods to the BI-SDK⁷ Web service in order to search purchase order data and line items from SAP Business Intelligence. If this required data is found, it is once again read from the R/3 system and the changes are saved there. The reason for

6 If the class is decompiled, you can see this extension in the code. In the original project code the extension is not available at this point in time.

7 Business Intelligence Software Development Kit. This SDK enables you to access data from SAP NetWeaver Business Intelligence via Web services.

using this solution can be an already heavy load on the R/3 system. Due to the previous filtering performed by BW this load does not increase excessively.

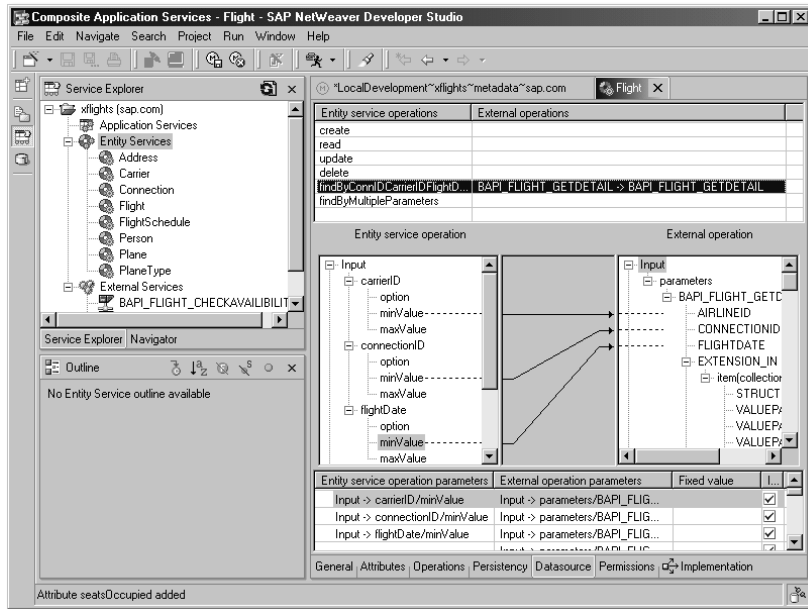


Figure 3.14 Mapping an External Service to an Entity Service

Changes Similar to the local persistence, in a remote persistence the operations can only be changed in so far as the mappings are adaptable to the methods and parameters. If more transformations or adaptations are to be executed, then either the external service itself must be changed correspondingly, or it must be carried out in the application service that is located at the next higher level.

Operations

Operation types The query and storage of data from the persistence in the Entity Services is enabled by five operation types:

1. CREATE
2. READ
3. UPDATE
4. DELETE
5. FINDBY

If the Entity Service is defined with local persistence, the first four operation types (the *CRUD operations*) cannot be changed. In the generated coding, these four methods⁸ are created. For local persistence, adjustments and transformations in the coding can only be executed in the application service.

The arguments of the `CREATE` method can be set in the **Attribute** tabstrip of the Entity Service Modeler. When creating attributes that were defined either as keys or mandatory input fields, the `CREATE` method is also enhanced by the corresponding arguments. When calling the `CREATE` method in order to create a data record, these arguments must be transferred also and cannot be empty.

Attributes

The `FINDBY` operations represent a special case. Depending on the number and type of the attributes, several of these can be created. The search parameters are selected using a dialog wizard. Possible input parameters are standard attributes, simple attributes, the sub-attributes of complex attributes, and attributes of Entity Service attributes. With the exception of the Entity Service attributes, all other attributes referred to here can also be used as output parameters for the search. For the Entity Service attributes, only the respective keys are returned. If attributes of the Entity Service attributes themselves are required as return values, they must be queried separately in the application service.

By default, you can carry out a restricted search with a placeholder in the `FINDBY` operations in the case of local persistence. The character `*` (asterisk) acts as a placeholder. The restriction given by using JDO means you can only run queries of type `"aa*"` or `"*zz"`, i.e. the placeholder symbol can only be the first or the last character of the search term. In the case of a remote persistence, the `FINDBY` operation supports every type of placeholder and position which is supported by the remote operation used.

Search with placeholder

`FINDBY` operations require an object of the type `com.sap.caf.rt.bo1.util.QueryFilter` as a transfer value. It is clear from the nine constructors of this class that several types of query filters are possible, and these can adopt both individual values and intervals and conditions.

Transfer value

A query filter can look as follows:

```
QueryFilter filter1 = new QueryFilter("HelloWorld");
QueryFilter filter2 = new QueryFilter("Hello*");
```

⁸ The terms "method" and "operation" are used as synonyms.

In addition, based on the settings in the Entity Service, operations are created such as

```
searchidxForB0InRelatedDoc
```

This operation belongs to the Entity Service **Document** and runs a search for all data records which contain a reference to a document. The operation then returns a list of results.

During generation, in the complete coding authorization checks, logging and tracing and other calls are automatically created as well.

3.5.5 Generation, Build and Deployment

While the services are being created and changed, the metadata also is being created, links are being created, and coding and interfaces are being generated. For an executable application, Java dictionary objects (DDIC) such as tables, data elements, domains, views and metadata, and program and interface classes are required. During generation, these files and configuration files are once again regenerated and during the final build three archives are stored per CAF project. The archives contain the information on the DDIC objects, the metadata, and the compiled coding.

Consistency check During the build itself, in addition to compilation, all references are also checked once again and the consistency of the archive is ensured. If the consistency and links to references cannot be produced, the build cancels with an error message.

Subsequent action If the build was successful, the archives can be "deployed" on the J2EE server. By "deploying," we mean the process of copying this archive to the server and the execution of subsequent activities. A subsequent action is, for instance, the creation of tables and database structures by calling the information contained in the dictionary archive to execute corresponding SQL statements.

During the deployment, a check is run to see if the specified references are valid. If this is not the case, the deployment is usually canceled with an error message. However, if the deployment took place without error, the services are available to the application. The tables are created in the local database and are waiting to be populated.

The entire process of generation, build, and deployment is executed in one step and transparently for the user.

The comparable functionality in the ABAP Workbench would involve generating or activating objects.

Comparison to
ABAP

Java Dictionary

Tables and their relationships to each other can now be created in the database by using the information in the Java dictionary project.

Metadata

The metadata is stored in the metadata project. In the more than a dozen sub-directories of the Meta Model, the definitions of interfaces, attributes, data objects, fields, mappings, operations, authorizations, properties and tables are stored in XML files. At least two XML files are created for each of the definitions referred to, even small application projects can contain several hundred XML files.

Since these files are regenerated each time you use the CAF Designer to perform changes to the services, modifications executed on XML files with non-CAF tools are lost.

Apart from this, changes to the services, depending on the extent of these changes generally also affect more than half a dozen XML files.

Coding

The generation of the services is based on coding templates and metadata definitions. The coding templates for external, entity and application services and their interfaces can be found in the plug-in directory of the CAF Designer under `com.sap.caf.designer` in the sub-directory `templates`. More than two dozen templates are available there and can be used according to the relevant definition.

Templates

The coding that is generated in such a way is then stored uncompiled in the source directory of the EJB module project, the built archive that is ready for deployment is stored together with the compiled Java classes in the Deploy directory. Coding for Entity Services (and also application services) is generated in the form of sessions beans.

As the Java classes are regenerated each time you use the CAF Designer to perform changes to the services, modifications executed on Java classes with other tools are lost. We would therefore strongly advise you against making changes directly in the generated coding even if the temptation is great.

Note

We would also advise you against making arbitrary changes to the coding templates in the `templates` directory. SAP will not provide any support if these templates are changed by application developers (i.e. developers who are not part of the CAF team). In addition, a future upgrade to a new CAF release can be made more difficult or even impossible, as SAP reserves the right to change these coding templates to make them incompatible, if necessary.

Excursus: Why Are Session Beans Generated?

The purpose is to provide a standardized service interface. Therefore, all services are session beans (entity and application services). By using JDO, the beans are also made persistent and the generation of entity beans would be too much generated coding. However, the generation of other types of coding, depending on the task at hand, is not ruled out for the future.

Four methods are created by default for an Entity Service, namely the `create`, `read`, `update` and `delete` methods. In addition, `findBy` methods are generated, provided that these were generated by the developer on the **Operations** tabstrip. There are also some attributes and set methods that are not available for general use. For example, the `setKey` method cannot be used in order to set a separate key. This method is only used internally.

3.5.6 Testing

ServiceBrowser After successful deployment services can be tested on the J2EE server with the *ServiceBrowser* by calling them from the context menu of the service explorer in the CAF design time. In the subsequent screen you can then see all the services registered on the server.

You can find the link for this and all other tools in the appendix.

Comparison to ABAP In the ABAP world the ServiceBrowser corresponds to Transactions SE37 and SE38 for changing and executing reports and function modules.

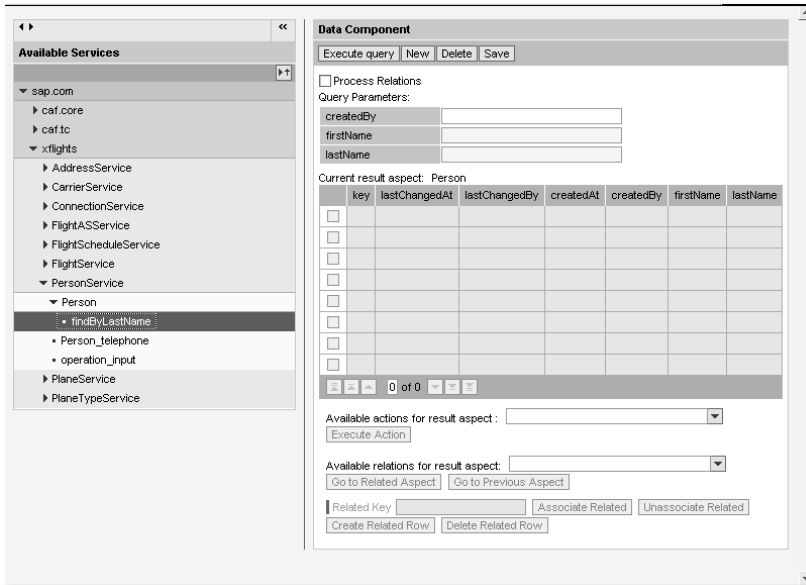


Figure 3.15 ServiceBrowser

3.5.7 Authorizations

The CAF employs the UME supplied with SAP Enterprise Portal to check user authorizations. In the Entity Service Modeler in the **Permissions** tabstrip the authorizations check generally takes place at the object level ("Should an authorization check be carried out for this Entity Service?") and switched on or off at the instance level. The term "instances" refers to the individual data records in this case.

If the authorization check was switched on at the instance level, an access-control list (ACL) is created for each instance. This list contains information on each data record for which authorizations are permitted for which users or roles. The corresponding code sequence can then be executed in the application service. There is an administration interface to maintain the ACLs (see Figure 3.16).

In addition to maintaining ACLs, you can also create rules (business rules) and conditions there. The authorizations created can be imported and exported as well as propagated to referenced Entity Services.

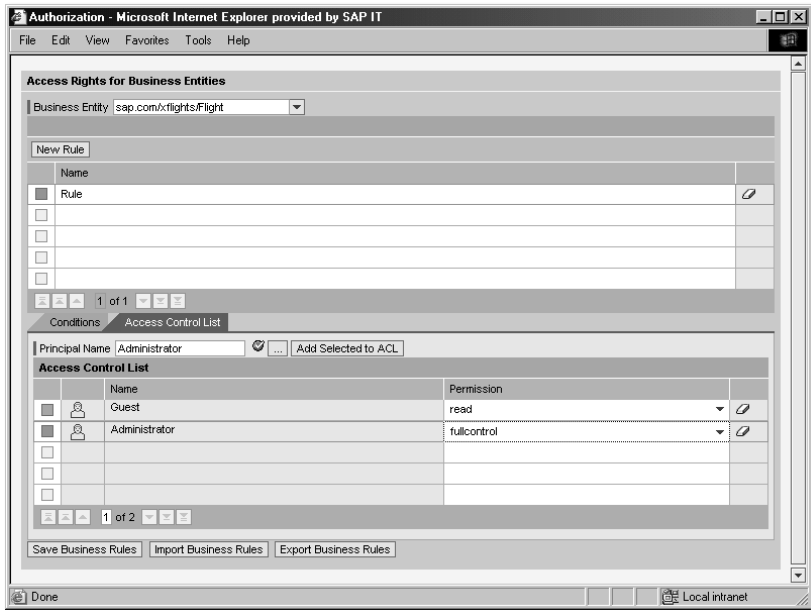


Figure 3.16 Authorization Maintenance for Users, User Groups and Roles

UME actions UME actions must be available so that the UME can work with the authorizations for operations of the application services. You can create these in the CAF Designer using the Application Service Modeler.

The administration tool **Principal Authorization Report** is used in order to be able to view the rule-based authorizations of a user for an Entity Service for all instances (see Figure 3.17).

You can use this tool to display the authorizations but not to change them. The authorization data can also be downloaded locally.

The reverse view of data, namely the display of authorizations for all roles and users on one instance of an Entity Service occurs with the UI illustrated in Figure 3.18.

Calling this UI, does not return any data at first because both the configuration (and hence the Entity Service) and the instance are required. The configuration takes place in the screen displayed in Figure 3.19.

The data cannot be changed here either. In order to transfer the instance, this UI can—for example—be called from the **ObjectSelector** (of the **ObjectList**).

Principal Authorization Report

Business Object Type:

Select business object attributes to be displayed as columns

Attribute Name
<input type="checkbox"/> lastChangedBy
<input type="checkbox"/> lastName
<input type="checkbox"/> primaryKey
<input type="checkbox"/> telephone

7 of 10

Administrator

Business Objects Permissions

PrimaryKey	Instance Level Permissions	Rule: PersonAuthorizations
0b08abc1-26e0-11d9-84d0-000bcd011da3		write, update, owner, delete, remove, read, fullcontrol
54f85170-26e0-11d9-cb58-000bcd011da3		write, update, owner, delete, remove, read, fullcontrol
635ccde0-26db-11d9-89d4-000bcd011da3		write, update, owner, delete, remove, read, fullcontrol
6a324730-26e0-11d9-afba-000bcd011da3		write, update, owner, delete, remove, read, fullcontrol
ed54f230-26df-11d9-ab17-000bcd011da3		write, update, owner, delete, remove, read, fullcontrol

1 of 5

Figure 3.17 Principal Authorization Report

ID	Name	Permissions

0 of 0

Rule Name	ID	Name	Permissions

0 of 0

Figure 3.18 Authorization Report

Figure 3.19 Authorization Report Configuration

3.6 Application Services

3.6.1 Introduction

Business process logic

The Entity Services described up to now primarily deal with generation of code for the relatively simple and redundant, but—provided they are to be programmed manually—coding-intensive tasks of data reading and writing, which are prone to errors. Application services, by contrast, contain the “intelligent” part of the application, namely the business process logic.

Application services are also the only way for entity or external services to get in contact with the UI in the CAF architecture, according to the Model View Controller principle (MVC).

In the xFlights example, the most important application services would mainly be those for booking flights and all others which execute searches for data records in the Entity Services. Figure 3.20 provides an overview of the application services in the sample scenario.

Index

A

- ABAP 58, 277
- ABAP Dictionary 58
- ABAP Workbench 58
- Abstraction 85
- Access Control List 135
- Account manager 236
- ACL 135
- Action 215
- Adapter 277
- Administrator 238
- Advanced Business Application Programming (ABAP) 277
- API 277
- Application 277
- Application Programming Interface (API) 277
- Application Service 94, 209, 277
- Approval procedure 235
- Approved Vendor List (AVL) 223
- Approver 238
- Architectural model 55, 94
- Association 122
- Asterisk 131
- Attributes, transient 83
- Authorizations 135

B

- Backend system 127
- BAPI 277
- BEx Information Broadcasting 66
- BEx Web 67
- BEx Web Analyzer 68
- BEx Web Application Designer 67
- BI Java SDK 68
- BI Meta Model Repository 108
- Bid type 224
- Business Application Programming Interface (BAPI) 277
- Business benefits 232
- Business Content 68
- Business Explorer 66
- Business Intelligence Platform 66
- Business Package 277
- Business Process 277

- Business process logic 138
- Business Scenario 277
- Business Server Pages model 59
- BW Service API 167

C

- CAF 280
- CAF Documents Component 251
- Cascading deletion 122
- Category 123
- CategoryService 124
- Change Management Service (CMS) 58, 61, 184
- ClassificationApplicationService 143
- Client/server architecture 21, 277
- Code generation 85
- Collaboration 277
- Commit
 - distributed 128
 - two-phase 128
- Component 277
- Component Build Service (CBS) 61
- Composite application 30, 31, 33, 34, 277
- Composition 122
- Concept 192
- Configuration 178
- Conflict resolution 244
- Content management 231
- Context 163
- Contracts 241
- CORE 112
- Cost blocks 234
- Cost element 234
- CRUD 131
- Currency 120, 125, 187
- CurrencyConversion 143
- Custom appends 122
- Custom composite applications 39

D

- Dashboard 238
- Data cleansing 224
- DC 179
- Decision criteria 51

- Deploying 132
- Design Time Repository (DTR) 61, 179
- Development class 110
- Development component 179
- Discussion 125
- DiscussionRoom 125
- Dispatch and follow up 235
- DocContent 143
- Document controller 238
- DTR 179
- Dynamic attribute 239

E

- EDI 278
- Electronic Data Interchange (EDI) 278
- Encapsulation 26
- Enqueue Server 169
- Enterprise Portals 280
- Enterprise Service Repository 265
- Enterprise Services 25, 278
- Enterprise Services Architecture (ESA) 27, 278
- Entity Relationship Diagram 100
- Entity Service 93, 100, 278
- ESA 278
- ESA platform 27
- Excel 162, 216
- Excel export components UI 253
- Excel parsing API 251
- Exchange Infrastructure 177
- ExchangeRate 125
- Exploded BOM 242
- Extensible Markup Language (XML) 278
- External Service 94, 278
- Extraction templates 241

F

- Filtering mechanisms 230
- Fuzzy search 176

G

- Guided Procedures 87, 163, 213
 - action 87
 - Guided Procedures Template 88
 - phase 87
 - process template 88
 - process template design time 89

- Guided Procedures Action Interface 164

H

- Hierarchical BOM 242
- Hypertext Markup Language (HTML) 278
- Hypertext Transfer Protocol (HTTP) 278

I

- IBM WebSphere 56
- IDE (Integrated Development Environment) 60
- Implementation process 42
- Information integration 64
- Innovation costs 39
- Innovation risks 39
- Integration Directory 63
- Integration Repository 63
- Integration Server 62
- Interactive forms 90, 164
- Interface 278
- Internationalization 184
- Internet standards 279
- iView 279

J

- J2EE 279
- J2EE Engine 59
- JARM 183
- Java 2 Enterprise Edition (J2EE) 279
- Java Application Response Time Monitoring 183
- Java Community Process 59
- Java Database Connectivity 168
- Java Development Infrastructure (JDI) 58, 116, 178, 184
- Java Dictionary 61
- Java Management Extensions (JMX) 279
- Java Message Service (JMS) 279
- Java Server Pages 279
- JavaDoc 141
- JDBC 168
- JDI 116, 178
- JDO 129, 170
- JMS 279

JMX 279

JSP 279

K

KMIndex UI 123

L

Lightweight Directory Access Protocol (LDAP) 279

Locking mechanism 59

M

Macro 154

Mainframe 21

Manufacturer Part Number (MPN) 224

Market opportunities 18

Market situation 17

Microsoft .NET 56, 279

MMR 106

MMR cache 108

Model View Controller (MVC) 91, 138

Modeling 85

Multi-Channel Access 279

mySAP Business Suite 34, 279

N

Name server 116

O

ODBO 168

OLAP BAPI 172, 173

Online Analytical Processing (OLAP) 279

Open Hub 173

Open SQL 59

Opportunity 239

Original Equipment Manufacturer 223

P

Package 110

Packaged composite applications 35

Pattern 85, 279

People integration 74

People Picker 253

Perspective 97

Placeholder 131

Portal Content Directory (PCD) 77

Portal runtime 77

Price extraction 235

Principal 125

Process Integration 61

Procurement scenarios 235

Product 191

Product definition process 192

Product innovation process 191

Project manager 236

Purchase orders 241

Purchasing info records 241

Q

Query 153

Quotation Worksheet (QWS) 234

Quote package 246

Quote team lead 236

Quote team member 236

R

Release 279

Remote Function Call (RFC) 279

Report-report interface 173

Repository 280

Repository Manager 70, 176

Retraction 172

Reusable components 253

Revenue acquisition 223

Reviewer 238

RFC 279

 transactional 115, 128

RFQ 222

Role 280

S

Sales employee 236

SAP Application Server 57

SAP Business Intelligence 65, 280

SAP Business Process Management 63, 280

SAP BW 280

SAP Collaboration 79

SAP Composite Application

 Framework 38, 42, 280

SAP Content Management 69

SAP Developer Network 172

SAP Developer Studio 58, 60

SAP Enterprise Portal 280

- SAP Exchange Infrastructure 62, 63, 95, 280
- SAP J2EE Engine 60
- SAP Java Connector 60
- SAP Java Development Infrastructure (JDI) 60
- SAP Java Test Tools 61
- SAP Knowledge Management 69, 280
- SAP Master Data Management (MDM) 281
 - Central Master Data Management 73
 - Consolidation 72
 - Harmonization 73
- SAP Mobile 74
- SAP Mobile Client 75
- SAP Mobile Infrastructure 75, 76, 281
- SAP Mobile Server 76
- SAP NetWeaver 35, 38, 55, 280
- SAP NetWeaver Portal 76
- SAP NetWeaver Solution Life Cycle Management (SLCM) 80
- SAP Query 168
- SAP Web Application Server 281
- SAP xApp Cost and Quotation Management 281
- SAP xApp Emissions Management (SAP xEM) 49
- SAP xApp Product Definition 281
- SAP xApp Resource and Portfolio Management (SAP xRPM) 46
 - expertise 48
 - Portfolio 47
 - Program 47
 - resource 47
- SAP xApps 33, 35, 36, 38, 40, 42, 281
- SAP xApps partner program 41
- SAP xCQM 281
- SAP XI 280
- SAP xPD 191, 281
- SCA, Software Component Archive 111
- SDK 168, 172, 173
- SDM 108
- SDN 172
- Search 175
 - fuzzy search 176
- Secure Sockets Layer (SSL) 281
- Service modeler 94
- Service Oriented Architecture (SOA) 27
- ServiceBrowser 134
- Sessions bean 133
- Simple Mail Transfer Protocol (SMTP) 282
- Simple Object Access Protocol (SOAP) 282
- Single Sign-on (SSO) 282
- Software Deployment Manager (SDM) 61, 108
- Software Development Kit 168
- Solution validation 45
- Sourcing bin 243
- Sourcing prep view 243
- Standard and moving average price 241
- Status 163
- Sub-attributes 131

T

- Table naming convention 126
- Taxonomy 70
- Team management 233
- Templates, coding 133
- Testing, service 134
- Topic 125
- Transient attributes 83

U

- UDDI 115
- UI pattern 164, 279
- UML 102
- Unit 120, 188
- UnitConversion 143
- UnitConversionSimple 125
- UnitOfMeasurement 125
- Universal Description, Discovery and Integration (UDDI) 24, 282

V

- Versions 185
- Visual Administrator 108, 183

W

- Web Dynpro 60, 282
 - foundation 164
- Web dynpro patterns 279

Web services 23, 282

Web Services Description Language
(WSDL) 115, 282

WSDL 115

X

xCQM

 backend integration 250

 data model 250

 entity services 250

 excel download 252

 excel upload 251

XI 177

XML 278

XMLA 168