Universität Hamburg
FB Informatik
Vogt-Kölln-Str. 30
22527 Hamburg

SAP R/3
An Overview of its Concepts and Languages

Studienarbeit
vorgelegt von:

Stephan Ziemer
Ebelingstraße 1
21073 Hamburg
2ziemer@informatik.uni-hamburg.de

Betreuer:
Prof. Dr. F. Matthes
TU Hamburg-Harburg
f.matthes@tu-harburg.de

5th of August, 1997

Contents

Motivation and Goals

SAP was founded in 1972 by five IBM-programmers. They had the idea to create one program suitable for many companies instead of writing essentially the same business-software again and again for different companies. They abstracted from the needs of a concrete company and wrote a program called System R which implements the processes of an abstract company. To use System R for a concrete company, it is necessary to 'customize' System R to suit the specific needs. This kind of software is called 'standard business software'.

This text presents an overview of the concepts SAP uses in its R/3-system from a computer scientist's point of view. The focus is on the properties and the intended use of the concepts, technical details are omitted wherever possible. The objective is to show the concepts behind R/3 and to enable the reader to recognize the concepts when they appear in the concrete system. The intention is not to replace a handbook or to give a detailed introduction to a specific issue in R/3.

The first chapter gives an overview of the system and different views of the system. R/3 is divided into the analysis level, the design level and the implementation level. The second chapter deals with the analysis level, the third and fourth chapter deal with the design level and the implementation level, respectively. Chapter five outlines the customizing and the system evolution in time. In the last chapter the technical architecture is stated.

The reader should be familiar with the relational data model. Knowledge about object-oriented modeling is helpful but not necessary.

Chapter 1

Concepts and Languages of R/3

The purpose of this chapter is to give a rough overview of the scope, the history and the concepts and languages of R/3.

1.1 Background and Scope of R/3

SAP's R/3-system is a package of standard international business applications for areas such as Financial Accounting, Controlling, Logistics and Human Resources. The R/3 system provides an enterprise solution for all these application areas in a distributed client/server environment. Using the R/3 system, a company can manage financial accounting around the world, receive and track orders for goods, and organize and retrieve employee information and records, among many other features. Many Fortune 500 companies and high-tech companies (including American Airlines, Chevron, IBM, Mercedes and Microsoft) run their businesses with the R/3 system. [1] The system is available on many hardware platforms and for many operating systems.

The 'R' stands for 'real time' which means that the system is interactive. '3' means that it is the third major version of the program, though there has never been a system called 'R/1'. 'R/1', a retrospective appelation sometimes heard today, refers to a collection of several applications and has never been called so. R/3 was introduced in 1992 and by the end of 1995 more than 5,200 R/3 systems were installed worldwide.

1.2 Integrated Analysis, Design and Implementation

This section describes the view R/3 has got of the 'real world'. 'Real world' in this context means companies and their data.

R/3 uses three levels of abstraction to reflect the real world in the system. At each level, three views can be taken: a process view, a funtion view and a data view.

At the top most level, the *analysis level*, a description of R/3 is given. Processes are modeled with EPCs(Event-controlled Process Chains) which describe how things

[1] taken from [4].

are done. An EPC consists of several states and actions and describes side effects like informing someone that an action has taken place.

Functions are described in terms of SAP modules, each module serving a special functional purpose within the company. E.g., the module HR (human resources) is to manage the staff data.

The data is modeled in a relational way[2]. SAP uses SERM (Structured Entity Relationship Model) as modeling technique. SERM was invented by SAP but is closely related to the common entity relationship model[3]. The model achieved is called 'data model'.

At the second level, the *design level*, processes are modeled with *workflows* which are user-defined EPCs. There is nothing to model functions at this level. The data model of the analysis level is implemented by relational tables and foreign keys on this level. The R/3 System has got its own data dictionary, where all meta data is stored.

At the last level, the *implementation level*, processes are modeled with development classes. A development class is a set of objects that work on the same business objects. Functions are represented as ABAP/4 programs and reports. Data is modeled by SQL tables and ABAP/4 variables[4].

The *repository* contains all three levels. It is the central place where any metadata of the system is stored. In the Repository Information System any object that can be manipulated is accessible. The data dictionary is a part of the repository as well as all development class objects are. See figure 1.4.

Most parts of the repository are, as nearly all information in R/3, saved in one SQL database. Supported databases are, e.g., Informix, Oracle and ADABS D.

Figure 1.1 shows the concepts and languages at each level. Figure 1.2 shows main objects of each level. Figure 1.3 gives the modules shipped with R/3 3.0.

1.3 Coexistence of multiple R/3 Clients

An R/3 system is split into different clients. A client is a business entity like a subsidiary. Clients have separate data for customizing and application data. They share customizing independent data like meta data and global company settings. Only data can be client-dependent. All meta-data, e.g. table defintions, are globally defined. Client-dependent data is achieved by adding a certain field to a table defintion of a table which is to hold client-dependent data. For each row of the table, this field holds the number of the client to which the row belongs.

In a usual R/3 system several clients exist like the default client (number 0000), a development client, a testing client and a customizing client. The actual client used in business should be in a separate system to avoid side effects by changes of global settings. Compare figure 1.5.

1.4 Application and System Evolution

Application and system evolution is a serious problem in practice. Updates have to be done in a very short time and customizing adjustments should not be affected. SAP tries to avoid name clashes by the introduction of naming conventions for objects.

In the R/3 System, the namespace is global and flat for each object type. E.g., tables must have a system-wide unique name, but a program object can have the

[2]For more information about the relational model see [14].

[3]see [14] for details about the entity relationship model.

[4]ABAP/4 is the programming language of the R/3 System. Chapter 4 covers ABAP/4 in more detail.

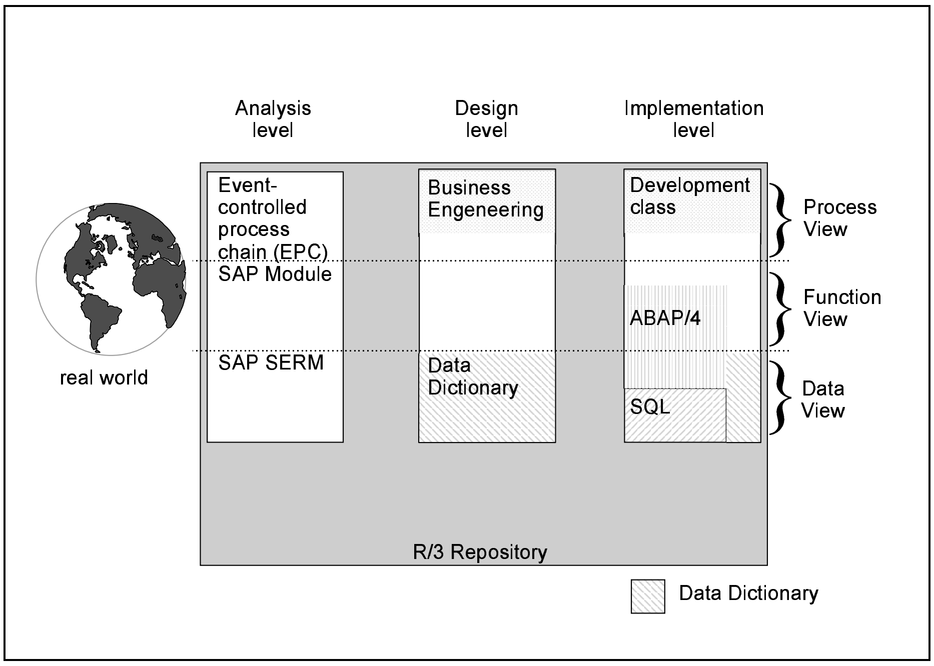


Figure 1.1: Concepts and Languages of the R/3 Repository

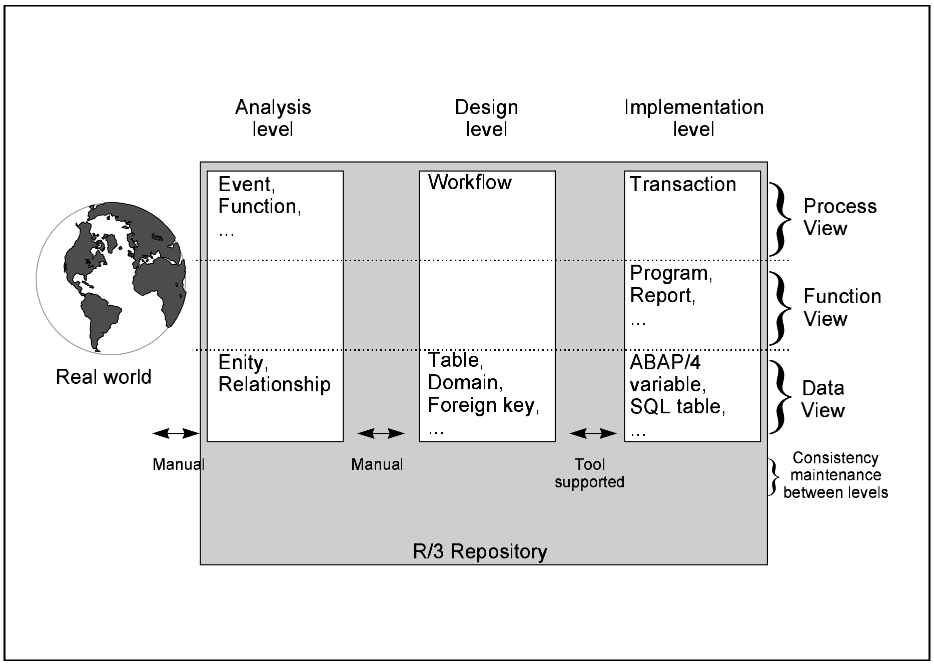


Figure 1.2: Objects of the R/3 Repository

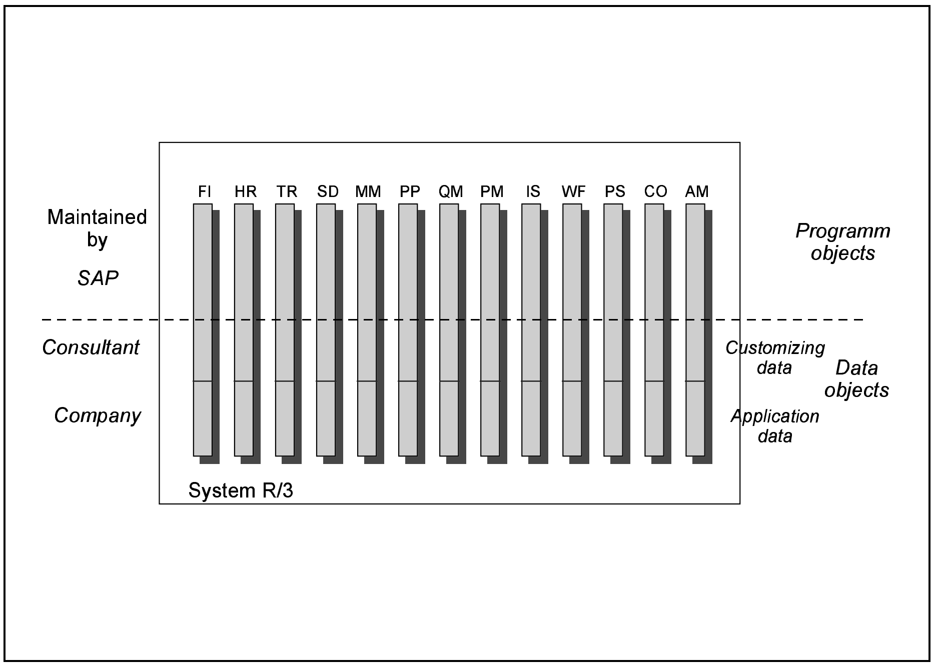


Figure 1.3: Logical Partition of R/3 into Modules

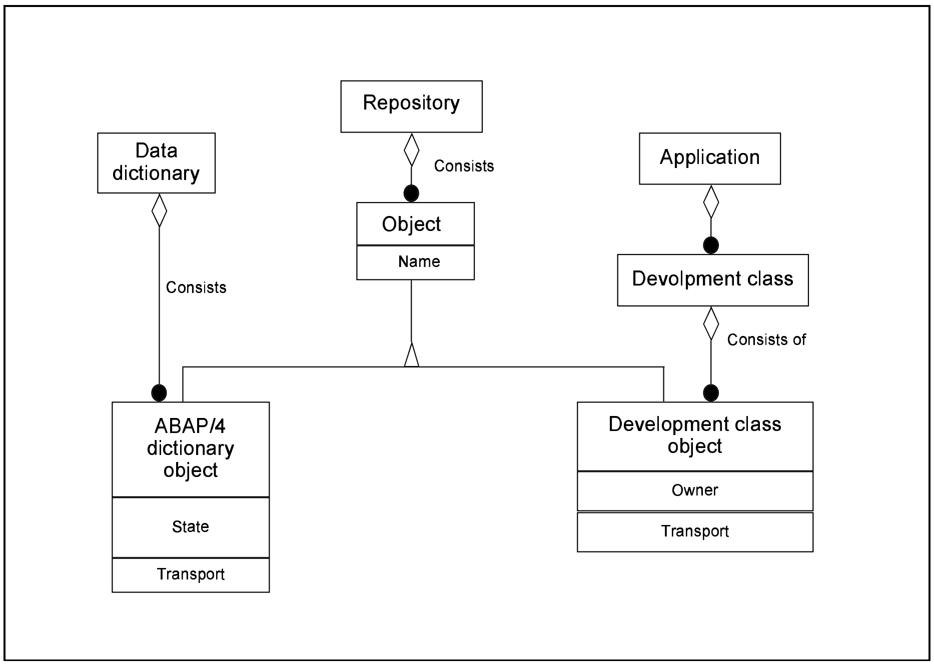


Figure 1.4: The Repository

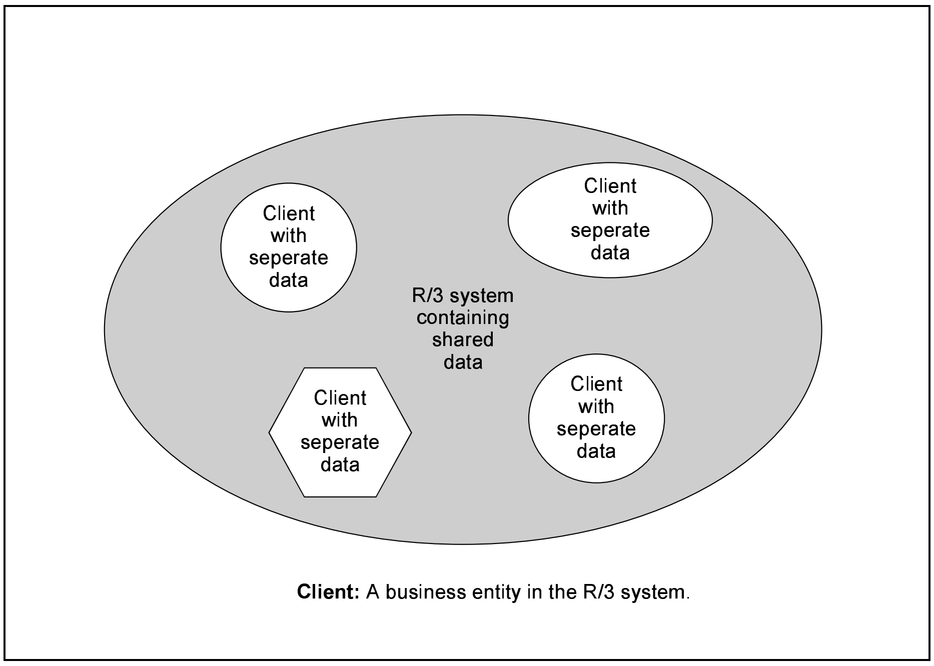


Figure 1.5: Clients in R/3

same name as a table. There are thousands of named objects of all kinds and combined with a strict limitation of length, the names are by no means self-explaining. To worsen the situation, English and German names and abbreviations are mixed.

SAP decided to leave the 'name range' Y* and Z*[5] to customer objects. There are exceptions from this rule leading to some confusion. SAP guarantees (so far) that there will be no name clashes with SAP objects when a new release is installed.

The example of the namespace dilemma shows what happens to many systems that have grown in time: Everywhere in the system one can find legacy objects that nobody dares to remove or -in this case- dares to rename for the consequences are largely unpredictable.

1.5 Running Example: FM Areas and Funds Centers

It is not within the scope of this text to describe every feature of R/3. Instead some features will be shown by example and this example is the FM area and the funds center.

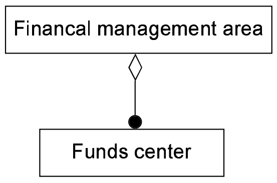
FM area: A financial management area (FM area) is the commercial organizational unit, with which commitment accounting is conducted.

Funds Center: A funds center is a commercial responsibility area to which a budget is assigned. [6]

A funds center must be assigned to exactly one FM area, several funds centers can be assigned to the same FM area. A funds center cannot exist without the superordinated FM area. See [13] for more details.

[5] all objects which names start either with a Y or a Z.

[6] Definitions taken from [13].



Financial management center (FM area):

The FM area is the commercial organizational unit, within which commitment accounting is conducted.

Funds center:

A funds center is a commercial responsibility area, to which a budget is assigned.

Figure 1.6: FM area and funds center in OMT

Figure 1.6 shows FM area and funds center in the OMT notation [7].

In a university context FM areas could be the departsment, like the department of comupter science, and the funds centers represent workgroups who have their own budget.

[7] see Appendix.

Chapter 2

Concepts at the Analysis Level

The purpose of EPCs and the data model is to document the business processes and the data relations R/3 implements.

2.1 Data Modeling: Entities and Relationships

The Data Model is a view on the company from a data point of view. There are two types of objects in the model.

Entity: An entity represents real data like the existing funds centers in a company. In the graphical notation entities are noted as boxes.

Relationship: Describes relationships between entities. The relationship is

- hierarchical, if the key of one entity, the so called *dependent entity*, depends on exactly one other entity, the so called *source entity*. This is the equivalent to a foreign key relation in the relational data model. In the graphical notation this is noted with an arrow from the source entity to the dependent entity. The dependent entity is on the right hand side of the source entity and the arrow points to the left edge.
- aggregating, if the key of the dependent entity depends on more than one source entity.
- referential, if non-key fields of a dependent entity depend on other source entities. This is the equivalent to a foreign key relationship with non-key fields in the relational model. In the graphical notation such a relationship is noted as an arrow pointing to the lower or upper edge of the dependent entity. Again, the dependent entity is on the right hand side of the source entity.
- a 'is a' relationship, if one entity is a special instance of another entity. E.g., the entity 'Bill' is a special instance of the entity 'person'.

A relationship has got a cardinality:

1:1 each entity of the source entity type has exactly one dependent entity. Noted by a single arrow.

1:C each entity of the source entity type has at most one dependent entity, noted by a single arrow with a crossing line.

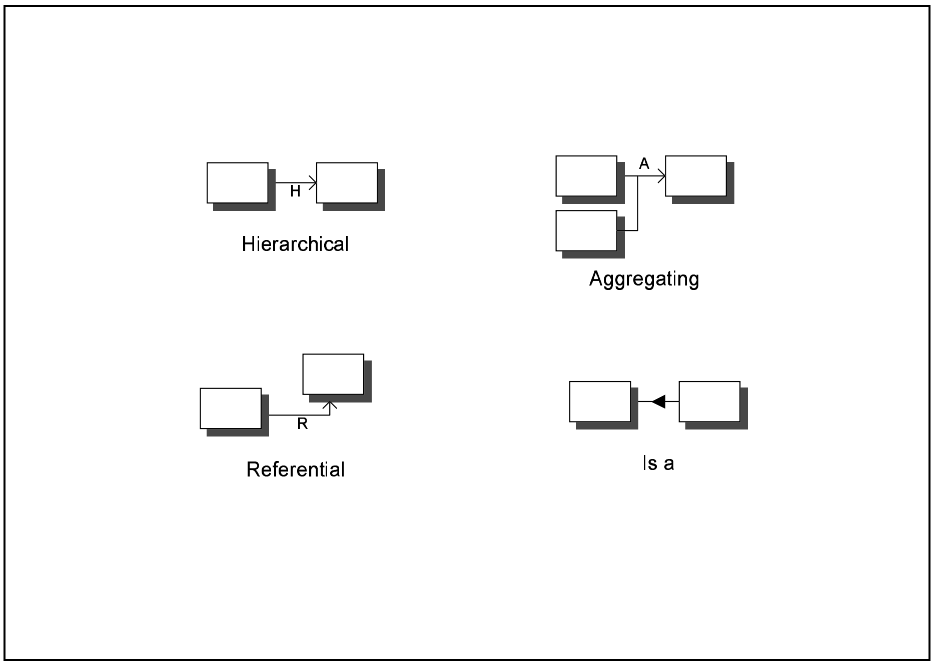


Figure 2.1: Types of relationships in the Data Model

1:N each entity of the source entity type has at least one dependent entity. Noted by a double arrow.

1:CN each entity of the source entity type can have any number of dependent entities. Noted by a double arrow with a crossing line. [1]

There cannot be a direct N:M relationship between entities in the data model.

Entities have some additional attributes, like a unique number and flags to indicate whether the data is changed during customizing or actual use and whether the underling implementation is a table or a view. See figures 2.1 and 2.2.

Figure 2.3 is an excerpt from the data model for FM areas and funds centers.

2.2 Functional Modeling: R/3 Modules

R/3 provides different applications, so called *modules* for the different departments of a company. Examples are the Financial Management (FI), Materials Management (MM) and so on. Theses applications work with the same data and are highly linked to each other. One can hardly use only one module of R/3. One has to use either a big part of the R/3 System or one cannot use it at all.

This is no doubt a strength and a weakness at the same time. On the one hand using much of the system means handling data efficiently for no conversations of data to other programs are needed and the data is handled in a consistent way, on the other hand the system does not scale for smaller or larger companies, making it a bit of an overkill for smaller companies. SAP is well aware of this and is currently looking for possibilities to make the system more scaleable.

[1] taken from [13].

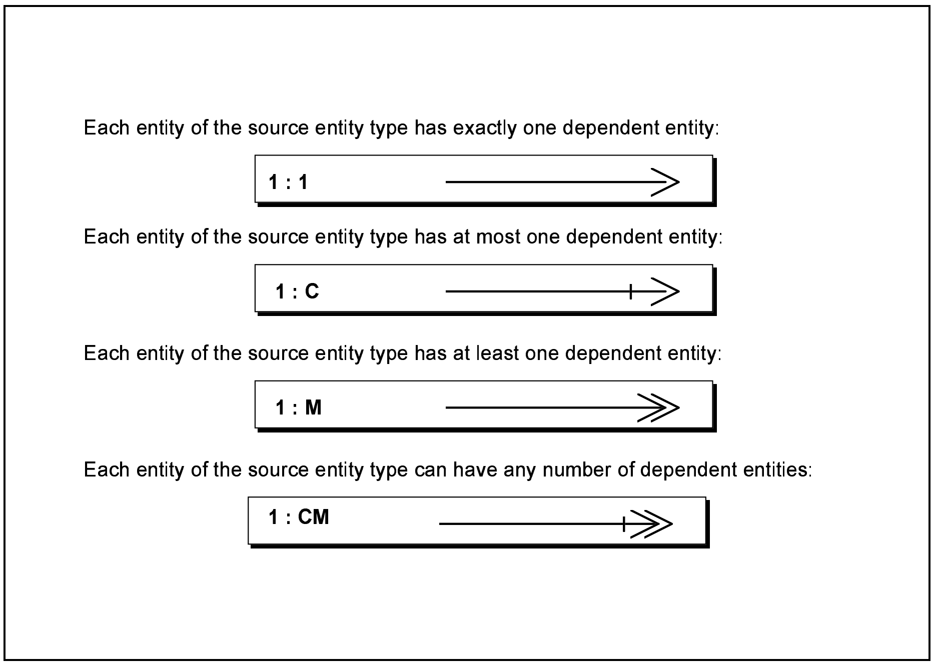

Each entity of the source entity type has exactly one dependent entity:
1 : 1
Each entity of the source entity type has at most one dependent entity:
1 : C
Each entity of the source entity type has at least one dependent entity:
1 : M
Each entity of the source entity type can have any number of dependent entities:
1 : CM

Figure 2.2: Cardinalities of relationships

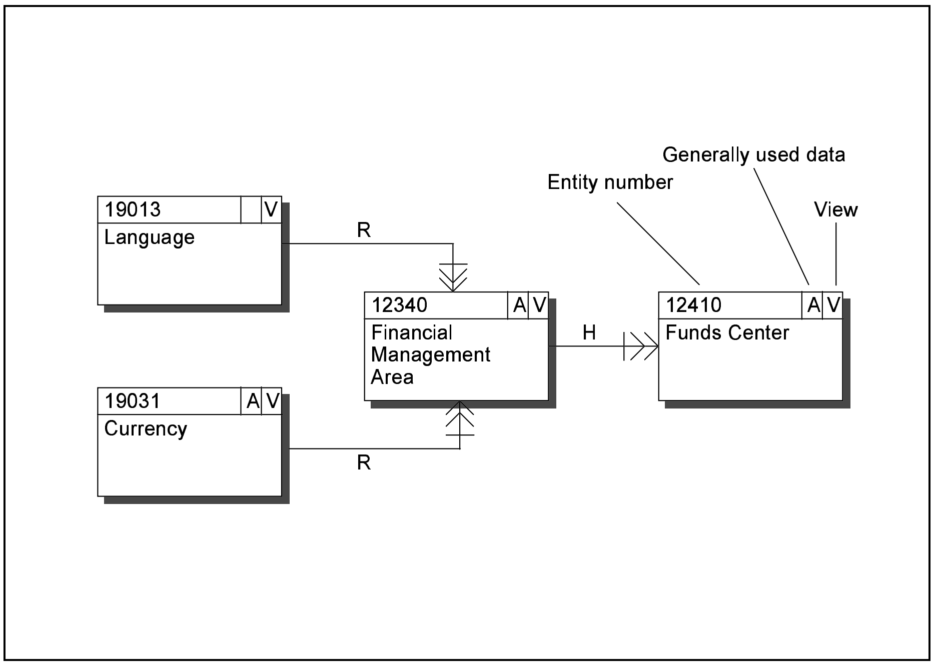


Figure 2.3: Excerpt form the data model

2.3 Process Modeling: R/3 Reference Model and EPCs

The R/3 Reference Model is a representation of the R/3 System using graphical models. It describes the various aspects of the SAP R/3 software, i.e., the supported business processes with their possible variants, data and organizational structures.

In the R/3 Reference Model, business processes that can be executed in the R/3 System are described graphically as *event controlled process chains* (EPCs). An EPC uses events to show the logical and chronological relationships between R/3 System functions. [2]

Four different element types can be used to form an EPC:

Function: describes what is to be done. The symbol is a hexagon.

Events: describing when things are to be done or at which stage the process is so far. The symbol is a rounded box.

Organization unit type: describes who (which part of the company) is involved. The symbol is an ellipse.

Information object: describes what kind of information is needed or produced.

Figure 2.4 shows an example of an EPC to create a new funds center.

EPCs are *not* integrated into the system in a strict sense. They are purely informational. There is no guarantee that a given EPC is implemented at all.

The semantic of EPCs are defined on an informal base only. One can find many contradictions and irregularities in many EPCs published by SAP and others. Efforts are being made to formalize EPCs so they can be checked automatically for inconsistencies. Petri-Nets[3] have been used to check EPCs.

[2] after the glossary in [13].

[3] For details about Petri-Nets see [11].

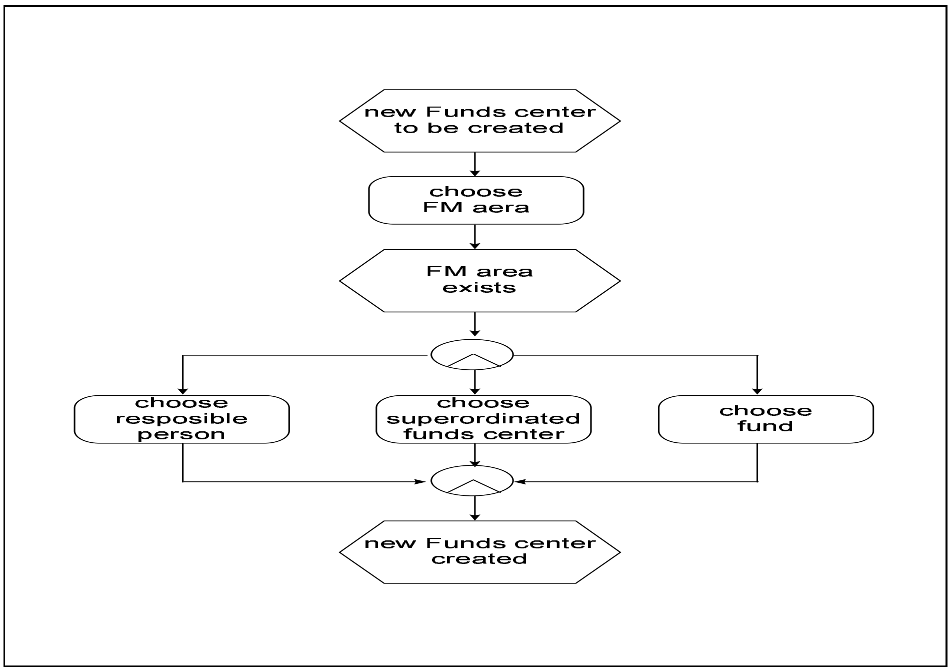


Figure 2.4: EPC to create a new funds center

Chapter 3

Concepts at the Design Level

At the design level the data models of the analysis level are brought to a more technical level. Furthermore, the design level is the link between the actual implementation and the analyzing model. Up to a certain point, it guarantees that different applications work on the same base when they refer to the same analyzing model.

The main tool of the Design level is the Data Dictionary, holding all meta-data of the system and the company's data.

The objects stored in the Data Dictionary (DDic) can be grouped into three groups:

- data modeling elements, i.e. tables, views, data elements and domains
- grouped data types, covered in chapter 4
- system oriented elements, covered in chapter 4.

The Data Dictionary is part of the Repository and reflects a view on the underlying database.

3.1 Data Modeling: Some Data Ditctionary Objects.

3.1.1 Domains and Data Elements

A *domain* is a basic elements that hides the technical representation of a piece of data. It is based on an external type. E.g., the domain `FICTR` is a sequence of four characters, the domain `FISTL` is a sequence of sixteen characters. The external type is in both cases the type 'character'.

Domains based on the external types CURR or QUAN are treated in a special way. Domains based on CURR and QUAN describe a currency amount or a quantity. There have to be tables, provided by the system or by the customer, that contain all valid currencies or quantity units, they are called reference tables. A field of a table based on CURR or QUAN must refer to an appropriate field of a reference table, the reference field. The reference field itself is based on either the external type CUKY (currency key) or UNIT. It determines the actually used currency, e.g. dollar or DM, or unit, respectively.

A *data element* describes the business management use of a domain. One can think of a data element as a semantical domain. E.g., the data element `FIKRS` describe FM areas and uses the domain `FICTR`. The data element `FM_FICTR` describes a funds center and uses the domain `FISTL`.

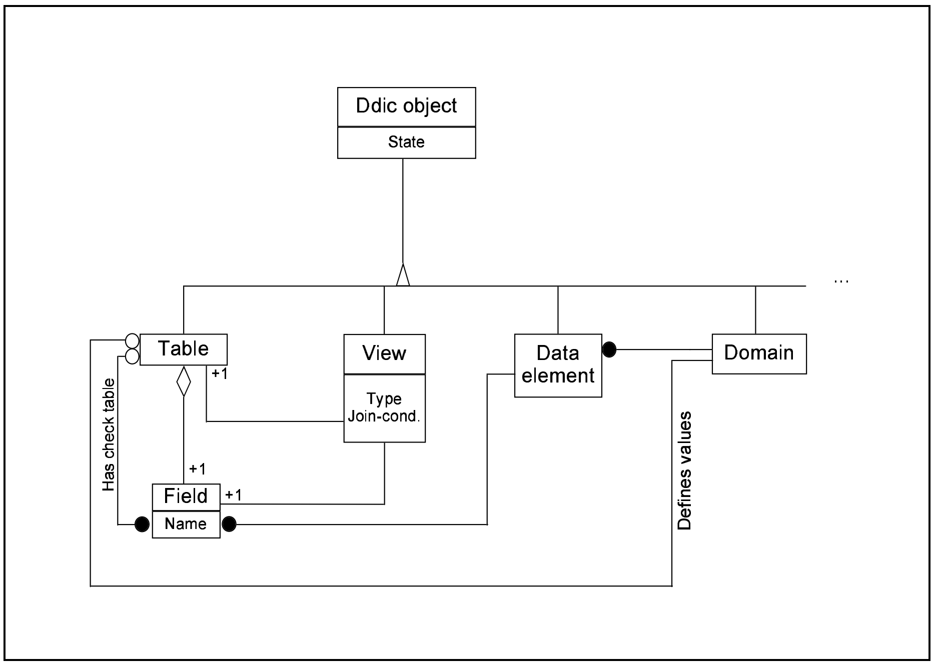


Figure 3.1: Data modeling elements in the DDic

3.1.2 Tables and Structures

The table concept of the R/3 System is similar to the relation concept in the relational model of data. An R/3 table consists of one or more fields. These fields can either be defined directly or in another table or in a structure. A structure is similar to a table, but it does not contain any data. See figure 4.2. The advantage of defining fields not directly but in another structure/table is, that when the included table/structure is altered, the definition of the including table is changed automatically.

A field of a table has got a name and an associated data element. A field can refer to a check table defining a foreign key relationship. The System does not guarantee referential integrity automatically. The actual R/3 application has to take care of that.

Depending on the intended use of a table, it can be realized in different ways in the database. A table can be

- transparent (being realized as an identical SQL table)
- a pool table (every row of the table is saved with other rows of other pool tables in the same SQL table)
- a cluster table (the whole table is saved as one row of an SQL table).

Tables for normal use are transparent, tables with only a few rows, which need not to be accessed from outside the system, can be pool tables. Cluster tables are normally used for language-dependent tables, e.g. a table that holds a description of an object in various languages.

It is also possible to define foreign keys. The referring field must be based on the same domain as the field referred to. The cardinality of a foreign key relationship [1] can be set, this is shown in the Data Modeler, when appropriate entities for the

[1] See figure 2.2

tables have been defined.

Foreign keys are important, because the system often uses so called *primary* and *secondary* tables. A secondary table is linked via a foreign key relationship to a primary table. The referring table is the secondary table, the referred table is the primary table.

The technique of *customizing includes* and *append structures* are used to modify standard tables [2].

- Customizing includes are special tables [3], which are included by standard tables. This is to offer customers the chance to change a standard table and to keep the changes over later upgrades of the system. Customizing includes are delivered empty and can be filled during the customizing process.
- Append structures are structures assigned to exactly one table. The append structure refers to the table, not vice versa. The table does not 'know' that it has a structure assigned. The Data Dictionary knows that and propagates changes to the table when necessary. This technique should be used to change a standard table when no customizing includes are provided for that standard table.

It is not recommended, though possible, to change standard tables directly. The standard tables could be changed with the next update leading to the loss of the changes made by the customer.

This points out that versioning is an important issue in practice. The relational model does not cover this problem, object-oriented systems still have to prove that they perform better on that. SAP's solution to the problem may not be very elegant, but it seems to work.

3.1.3 Views

Views give a special view on a data stored in a database [4]. In R/3, a view consists of one primary table and any number of secondary tables [5].

In an addition to the concept of views in the relational model, SAP R/3 supports four types of views:

Database view : A database view is the equivalent to a view in the relational model of data. If there is more than one table involved, the access is always readonly. Views of this type are used frequently to represent an entity in the Data Modeler.

Projection view : A projection view hides some columns of a table (a projection). The hidden columns will not be sent form the database to the application server, reducing the data volume to be transported .[6]

Help view : A help view shows further fields (columns) of a table, when a user is to specify a value in a foreign key field and requests help for valid inputs. There can be at most one help view per table. Matchcodes are a more advanced technique to achieve the same goal.

Maintenance views : Maintenance views enable a business-oriented approach to looking at data, while at the same time, making it possible to maintain the

[2] These are tables predefined in the system.
[3] named CI_*, an exception to the rule that the name range for customers is Y* and Z*.
[4] see [14] for details.
[5] at most the number of existing secondary tables, of course.
[6] Refer to chapter 6 for more details about the database and application servers of R/3.

data involved. The data can be maintained by the customizing transactions. [7]

3.2 Process Modeling: Workflows

The R/3 system allows the user to define his or her own workflows. The technique used is the same as the one used for EPCs[8].

SAP Business Workflow provides technologies and tools for processing and controlling cross-application processes automatically. This involves primarily the coordination

- of the persons involved
- the work steps required
- the data to be processed (business objects)

Its main aims are to reduce throughput times and the costs involved in managing business processes and to increase transparency and quality.[9].

[7] taken from [13]
[8] see chapter 2.
[9] taken from [13]

Chapter 4

Concepts at the Implementation Level

At the implementation level, the actual business processes are implemented. All applications are written in ABAP/4[1], the R/3 programming language. The user-front end is written in ABAP/4 as well and can be customized and utilized as far as required. The complete functionality of R/3, including the ABAP/4 compiler, is accessible in ABAP/4 programs making, e.g., the generation of programs on demand possible.

4.1 Implementation-Oriented Data Dictionary Objects

4.1.1 Data Types and Type Groups

The R/3 System supports five kinds of data types.

External types : They are the foundation of all types and have an equivalent representation in SQL. Any object that is to be persistent [2], must be converted to a corresponding object of an external type. Table 4.1 shows the existing external types.

ABAP/4 data type : Every ABAP/4 variable is based on an ABAP/4 data type. The types are: `C`(character), `N`(numeric character), `P`(packed numer), `F`(floating point), `I`(integer), `X`(hexadecimal number), `T`(time) and `D`(date).

All external types have a matching definition using ABAP/4 data types.

Header line type : This is an aggregated type. In most cases it is defined by referring directly to a structure, which is the equivalent concept in the Data Dictionary. Header lines are used to define variables which are necassary to exchange data between ABAP/4 and the data base, they serve as buffers.

Internal table type : An internal table is used to store data during the execution of a ABAP/4 program. [3] Internal tables are not persistent. If the data is to be persitent, it must be inserted into a database table.

[1] Advanced Business Application Programming Language, 4th generation.

[2] Persistence means in this context to be stored in the underlying database. For further details about persistence and how it can be achieved see also [8].

[3] Actually not during the execution of an ABAP/4 program, but during the execution of an SAP transaction.

External type	*Description*	*output length*
INT1	1 byte integer, unsigned	3
INT2	2 byte integer, used as length description for LRAW and LCHAR	5
INT4	4 byte integer, signed	10
FLTP	floating point number	16
TMS	time (HHMMSS) as 6 characters [CHAR(6)]	6
DATS	date (YYYYMMDD) as CHAR(8)	8
CLNT	client number	3
ACCP	account period (YYYYMM)	6
CHAR	sequences of characters	<256
NUMC	numerical characters	<256
LCHAR	same as LRAW but with characters	< 65536
RAW	sequence of bytes	<255
LRAW	sequence of bytes beginning with an INT2 as length field	< 65536
DEC	decimal	<18
CURR	currency field, realized as DEC	<18
CUKY	key for a currency	5
LANG	language key	1
QUAN	quantity field	<18
UNIT	key for a quantity field	2 or 3

Table 4.1: External Types in R/3

Type groups are collections of user defined data types or constants in ABAP/4 code. They are stored in the data dictionary for cross-program use.

4.1.2 Lock Objects

Lock objects guarantee the consistency of the database when many users work on the same data. A lock object can lock one primary table and several secondary tables.

The lock mode controls the method by which the users are given access to the same data records at the same time. The system supports the following lock modes:

E (Exclusive, cumulative) : This mode means that locked data may only be displayed or processed by a single user at the same time. The user owning the lock can request the lock again.

S (Shared) : This mode means that several users can simultaneously display the same data. A request for another shared lock is accepted even if it is requested by another user. A call for an exclusive lock is rejected.

X (Exclusive but not cumulative) : A lock of type X can be called only once. Any other call for such a lock is rejected, even if the user holding the lock calls.

The lock mode can be defined separately for each table in the lock object. When a call for a lock occurs, a corresponding entry is inserted into the lock table of the system. [4]

Special ABAP/4 code, a so called *function module*, is generated automatically for every lock object. An ABAP/4 program calls the function module to request a lock. If the lock is rejected, an exception is thrown.

[4] taken form [13].

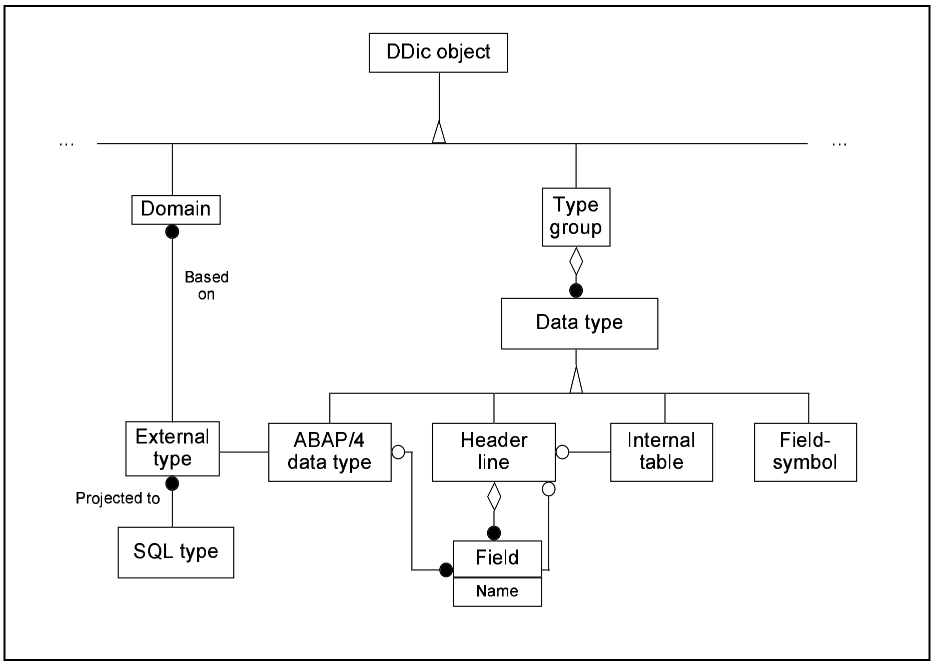


Figure 4.1: Data types in the Data Dictionary

4.1.3 Matchcodes

In the relational model of data, every tuple of a relation is unique and has therefore identifying attributes. In practice it can happen that all or nearly all attributes have to be used to identify a tuple. This means a performance penalty, because a great amount of data must be processed. To perform better, id-attributes like a unique number or a short string are used to identify tuples.

Using id-attributes is good for the performance but bad for human users. Users are often requested to identify a tuple by giving the id-attribute, e.g. when processing an incoming order the customer address is requested and the user has to type in the customer-id, which, e.g., is a number. The purpose of matchcodes is to help the user to find the information by displaying non-identifying attributes, which are useless for the system but meaningful to humans. In the example, a matchcode could display existing customer-ids and additional information, such as name, street and town.

A matchcode has got a primary table, in which the requested attribute is contained, and it can have secondary tables to show associated attributes (fields) of other tables. Furthermore a matchcode can show different sets of attributes, each set constitutes a so called *matchcode-id.*

Matchcodes are no database indices. Differences are:

- A matchcode can contain fields from several tables. An index contains fields from only one table.
- Matchcodes can be built on the basis of both transparent tables stored in the underlying database and using the special table types pool and cluster.
- The matchcode structure can be restricted by stipulating selection conditions.
- Matchcodes can be used as entry aids in the context of the SAP help system.

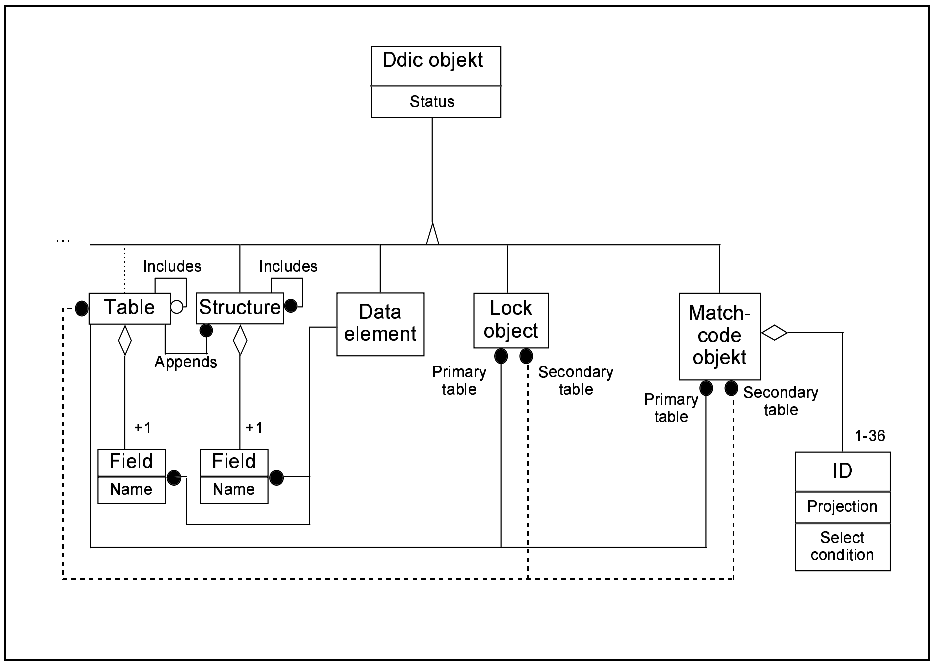


Figure 4.2: System oriented elements

[5]

4.2 Programming in the Large: Development Class Objects

A *development class* is a set of logically related development objects. Such a set of objects could be, e.g., all objects necessary to manipulate funds centers.

Figures 4.3 and 4.4 show the development class objects in OMT notation.

4.2.1 SAP Transactions

A SAP transaction covers a logical process in R/3 (e.g. generating a list of customers, changing the address of a customer, booking a flight reservation for a customer, executing a program). From the point of view of the user, it represents a self-contained unit. In terms of dialog programming, it is a complex object which consists of a module pool, screens, etc. and is called with a transaction code. [6]

It is helpful to think of a SAP transaction as one basic business process, which cannot be interrupted or half fulfilled. SAP transactions are sometimes called *logical units of work*(LUW), a LUW can involve more than one database transaction (DB-TA).

SAP transactions can be programmed to comply with the ACID condition [7]. This is achieved by postponing all actual database changes an SAP transaction wants to perform till the end of the SAP transaction and then doing all changes within one database transaction.

[5] differences taken from [13].
[6] taken from [13].
[7] **A**tomity **C**onsistency **I**solation **D**urability, see [14] for details.

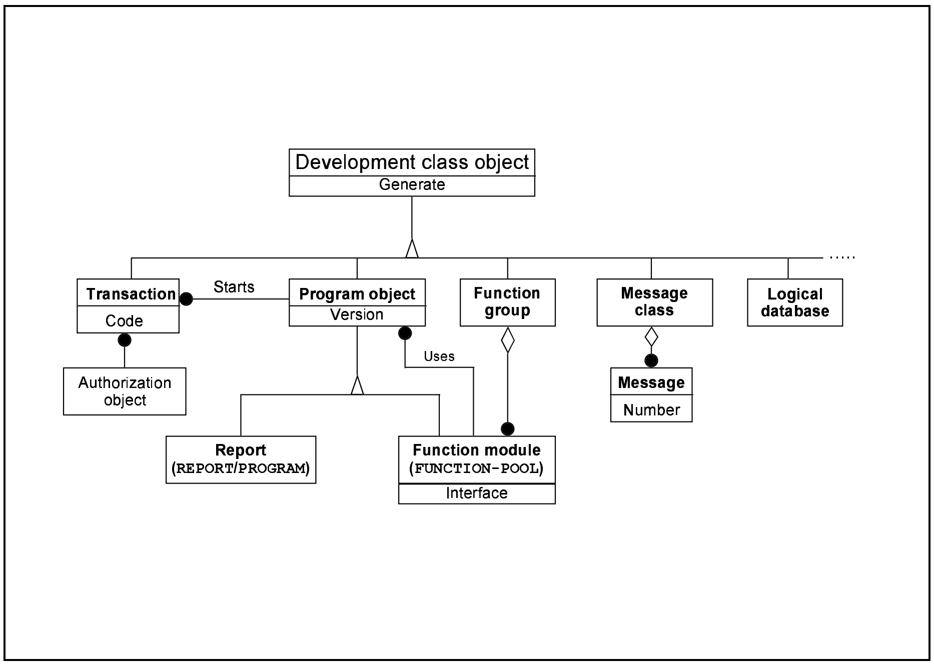


Figure 4.3: Development class objects (1)

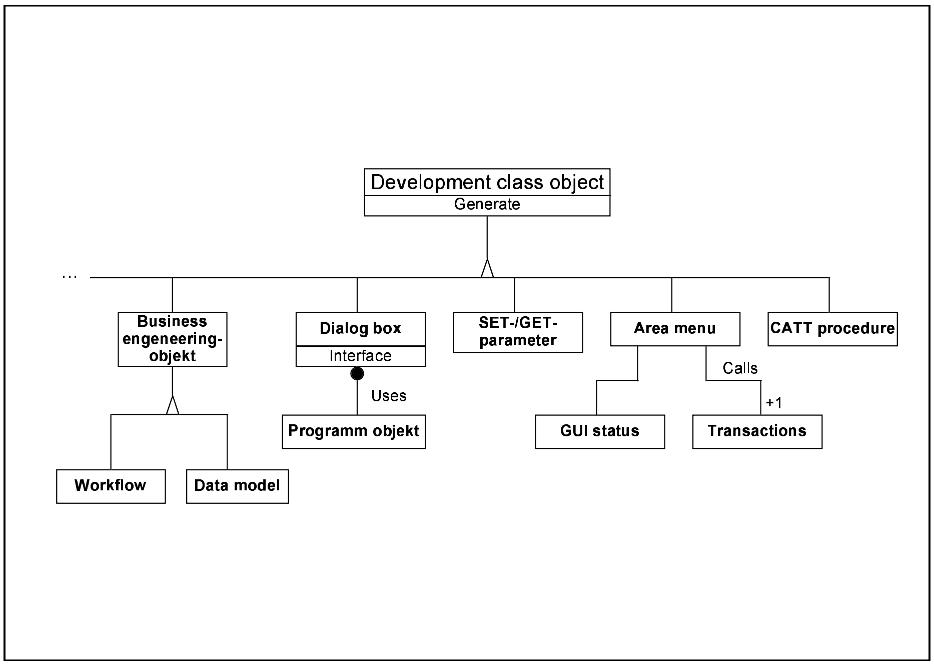


Figure 4.4: Development class objects (2)

To coordinate many users working with the same data, lock objects must be used. Lock objects lock the data the moment they are called and release it by default when the SAP transaction is finished and all the changes in the database have been done. Please note that the data is locked at the time the user works with the data which can be much ahead of the time when the actual change in the database is carried out.

4.2.2 Reports

Reports are ABAP/4 programs. They will be explained in more depth in the ABAP/4 section of this chapter.

4.2.3 Function Modules

A function module is a routine in ABAP/4. In addition, it has got an interface which is stored in the Data Dictionary. The concept of function modules is one of the most important concepts in R/3.

Function modules can

- be called from any other ABAP/4 program,
- be called via RFC[8] from programs outside the system,
- delay all database transactions till the end of the SAP transaction and build by that the fundamental base for fulfilling the ACID condition.

According to the different purposes, there are different types of function modules, called *process types*:

Normal : The function module can be called from inside the system only. Database transactions are performed immediately.

RFC supported : The function module can be called via RFC from outside the system.

Update with start immediately : The database transactions are delayed until the next 'COMMIT WORK' event. Should a database transaction fail, retrying is possible. This is a V1-booking.

Update immediately : The database transactions are delayed until the next 'COMMIT WORK' event. Should a database transaction fail, retrying is *not* possible. This is also a V1-booking.

Update with start delayed : The database transactions are performed (booked) after all V1-bookings have been performed. This is a V2-booking.

Sammellauf : The database transactions can be booked with other V2-bookings at one time. This is a V2-booking.

Function modules are written in normal ABAP/4 code and can use the full functionality of R/3. This is of special interest when a function module is called form outside the system, making it possible to control the system from outside.

So called 'batch input' can be provided. This is data stored in special tables and the system interprets the data as input for the system. With batch input everything that can be done interactively can be simulated.

Function groups are a set of logically related function modules. The *function library* stores all function modules and allows to search for specific modules.

[8]Remote Function Call, see [6] for details.

4.2.4 Messages

A message in R/3 is a string. Messages are prompted to the user in a modal dialog [9]. Every message has got a unique number in the message class it is part of. A message class is a collection of messages which are used in the same program. Nevertheless, it is possible for a program to use many message classes.

The R/3 system supports 5 types of messages:

Error (E) : The user made an invalid input and as soon as he or she has acknowledged the message, is forced to reenter the required information.

Warning (W) : The user may have made possibly invalid input, but can decide, whether to reenter the information or to proceed.

Information (I) : The user has to acknowledge the message and can proceed.

Success (S) : This message is not displayed in a modal dialog, but in the bottom line of the next screen. A message of this type is purely informational and need not be acknowledged.

Abort (A) : A critical error has occurred, a reentering of the information is not possible. The current SAP transaction is aborted. In most cases, technical reasons cause this kind of message to be prompted.

Message classes are maintained with a special tool, in terms of ABAP/4 this tool is an interactive report. Messages should be language dependent, of course.

4.2.5 Area Menus

As stated before, the ABAP/4 programs can use the full functionality of the SAP front end, including the menus. An area menu is a menu which triggers SAP transactions. Area menus can call any SAP transaction defined in the system. Area menus are not assigned to an ABAP/4 program and are invoked by a transaction code.

4.2.6 Other Development Class Objects

Logical databases are used to write reports. They consist of one or more database table(s), which are linked by user-defined conditions. Logical databases make reporting easier, not more powerful.

Dialog boxes are dialogs, which are used quite often in the system. It is possible to define new dialog boxes or to use predefined, standardized dialog boxes. Several kinds of standard dialog boxes are available:

- Confirmation prompt dialog boxes
- Dialog boxes for choosing among alternatives
- Data print dialog boxes
- Text display dialog boxes

[10]

SET-/GET-parameters are used to exchange data between SAP transactions. Their main purpose is to set values for input fields on the screen.

[9] dialogs are displayed in separate windows. A modal dialog blocks the system until the user has reacted to the dialog. Non-modal dialogs allow the user to continue working with the system, the user can react to the dialog whenever he or she wants to.

[10] taken from [4].

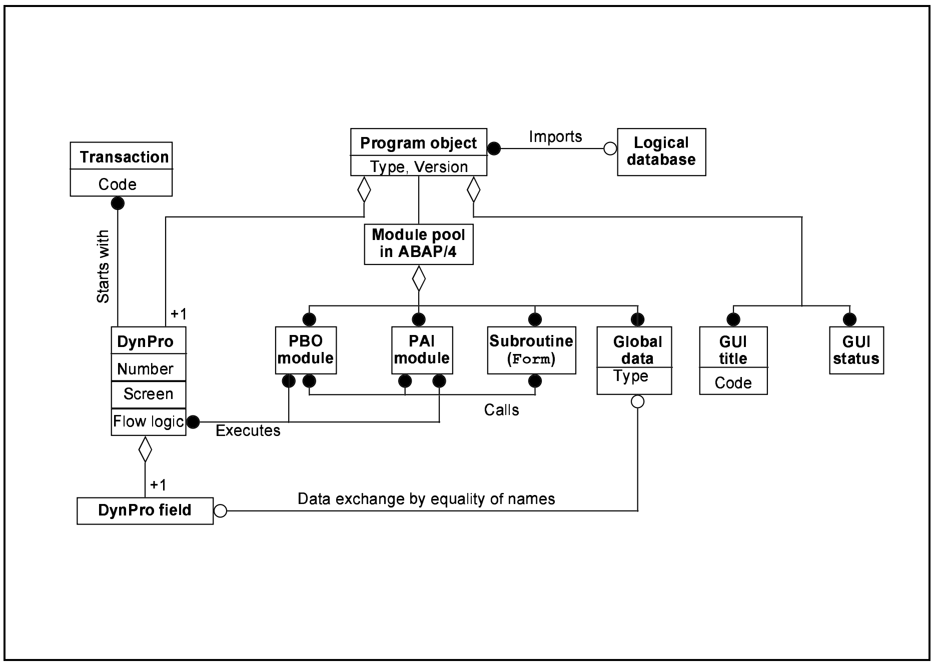


Figure 4.5: Program objects

CATT-procedure : CATT stands for Computer Aided Test Tool. This tool helps the user to test self-written programs or parts of the system functionality with special test data.

4.3 Programming in the Small: Program Objects

Figure 4.5 shows program objects in detail.

A program object consists of a *module pool*, written in ABAP/4, one or more *DynPros*, and some *GUI stati*. A program object can also use at most one *logical database*, the program object will then be a *report*.

The *ABAP/4 modules* are called by the *flow logic* and can call other subroutines, called *forms*, which cannot be invoked directly by the flow logic. Modules and forms can have their own local data and have full access to all global variables. The global variables can be used for internal purposes and are the interface to all database objects.

The data exchange between DynPro fields and ABAP/4 variables is done automatically. Before a screen is displayed, the PBO modules fill ABAP/4 variables, which have the same name as the DynPro fields, with the appropriate data. The DynPro interpreter then transfers the data into the DynPro fields and after input, it transfers the data back into the ABAP/4 variables.

4.3.1 GUI Status

A GUI status describes which menu bar and buttons should be represented to the user at a certain time.

The *menu bar* holds the menus. Two menus are always present: The system menu and the help menu. Menus themselves can contain *menu items*, e.g. 'Quit', which trigger actions or further submenus, like 'Create Object…', which lead to

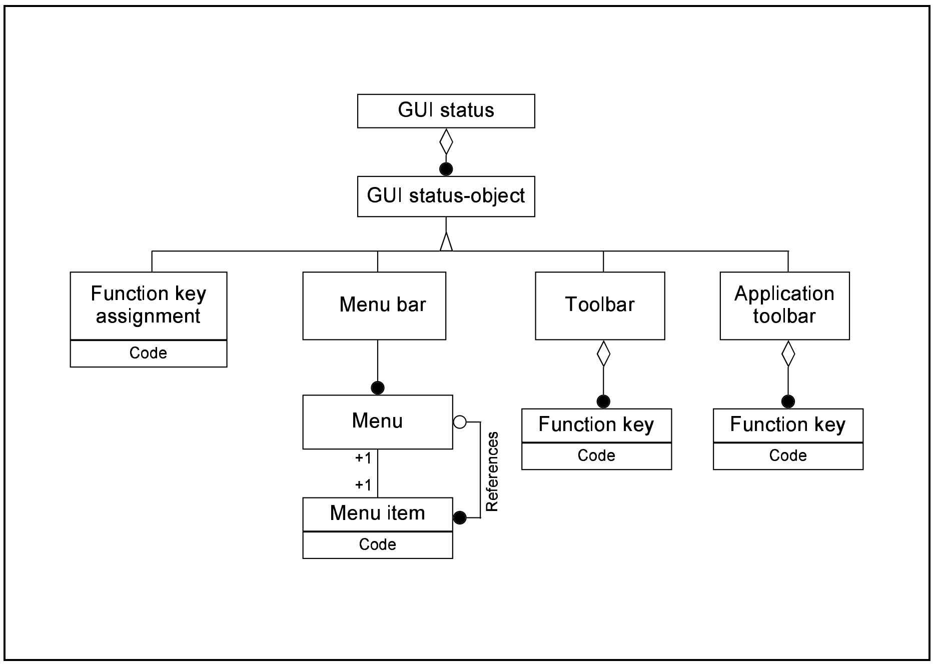


Figure 4.6: GUI status

other menus and menu items. A menu can cascade to a depth of three.

The *tool bar* holds a set of application independent buttons, the *application tool bar* holds a set of application dependent buttons.

4.3.2 The Model for interactive Programs

To understand the programming concept of R/3 it is necessary to understand the model for interactive programs. The R/3 system is mainly event driven, events can be triggered by the system itself or by the user.

The model is screen-oriented, the cycle is processed for every called screen (DynPro).

1. At the beginning of the cycle, everything assigned to the PBO[11] event is executed. In most cases this will be actions to prepare data to be presented to the user.
2. Next, the user does the actual input.
3. Depending on how the user ended the input,
   - additional information is displayed,
   - another transaction is executed,
   - the EXIT-COMMAND-event is triggered,
   - the PAI[12] event is triggered, which is the normal case.
4. The actions assigned to the triggered event are performed.

Figure 4.7 shows a usual basic cycle, figure 4.8 gives the possible execution steps of the DynPro interpreter.

[11]Process Before Output.
[12]Process After Input.

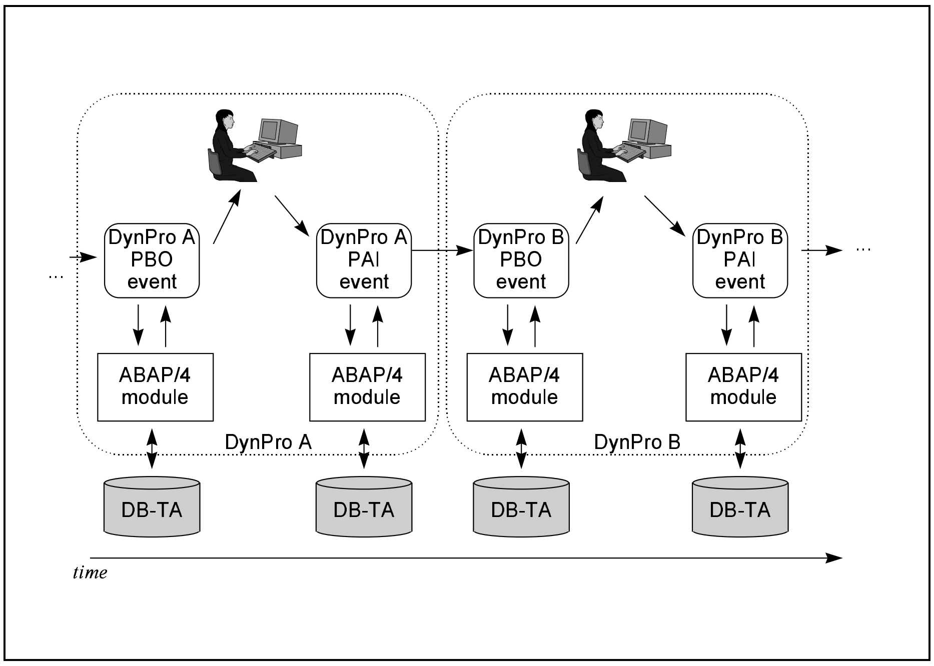


Figure 4.7: The Model for interactive Programs

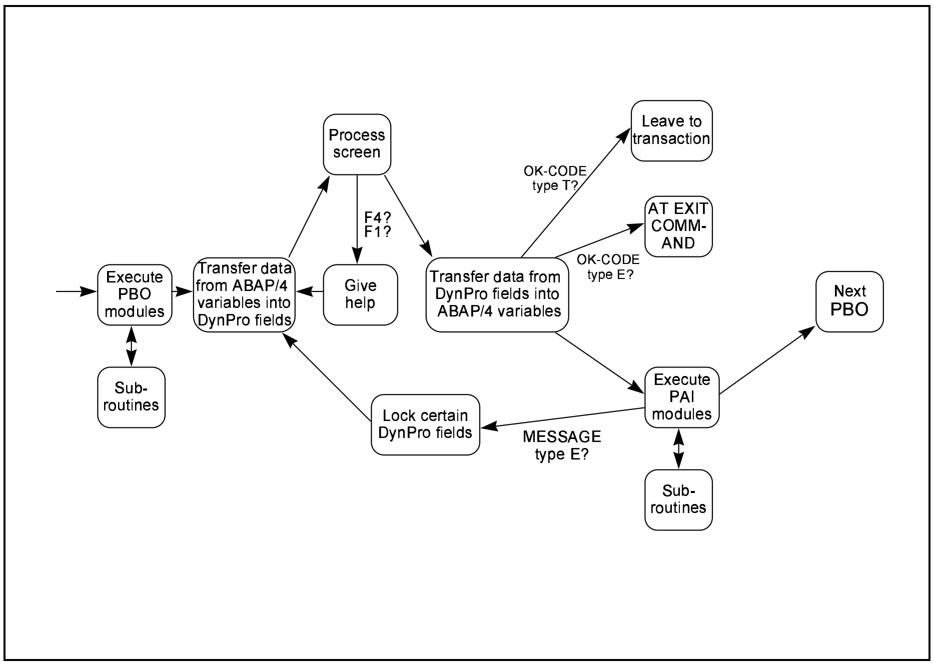
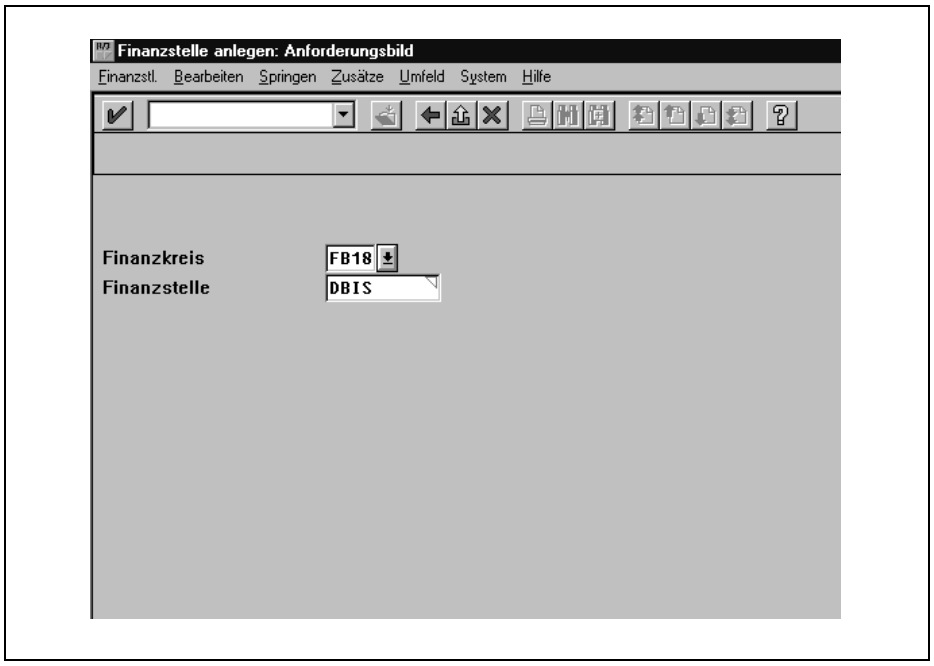


Figure 4.8: Execution steps of the DynPro interpreter


Finanzstelle anlegen: Anforderungsbild
Finanzstl. Bearbeiten Springen Zusätze Umfeld System Hilfe

Finanzkreis FB18
Finanzstelle DBIS

Figure 4.9: Transaction FM2I, DynPro 100

4.3.3 An Example of a SAP R/3 DynPro

This text is not intended to be a handbook for ABAP/4 programming. Instead, some of the features will be shown through an example, typical of R/3 applications.

The example is the DynPro 100 of the function group FM22. The function group provides functionality to maintain funds centers, like the transaction FM2I, which creates a new funds center. Other transactions are FM2S (shows an existing funds center) and FM2U (updates/changes an existing funds center). It is very common in R/3 to have such three transactions (insert/show/update) for one object.

In DynPro 100, shown in figure 4.9, the user has to type in the funds center (Finanzstelle) and the superordinated FM area (Finanzkreis) he or she wants to insert or update or look at. This DynPro will be explained in same depth at the end of this chapter, the source code is given in appendix C.

4.3.4 Components and Attributes of DynPros

A DynPro[13] consists of several components:

DynPro attributes tell the DynPro number, the number of the DynPro that should follow it by default , and some other attributes

Screen layout tells what fields should appear in the DynPro and where

Field attributes tell the properties of each field in the DynPro

Flow logic tells what ABAP/4 routines should be called for the DynPro

14

The single components will be explained by the example of DynPro 100, Transaction FM2I (insert).

[13] Dynamic Program.
[14] taken form [13], slightly modified.

DynPro Attributes

Main attributes are:

Attribute	*Value*	*Explanation*
Progam	SAPLFM22	to every function group there exists a program object named SAPL...
Number	100	this is DynPro number 100
Original Language	D	the language the DynPro has originally been created in, in this case is was created in German
Description	...	short description of what the DynPro does
Type	normal	this is a normal DynPro, no special functionality
Next DynPro	100	by default the succeeding DynPro is number 100, itself.

Screen Layout

The screen layout is designed with a special tool, the screen painter. The required fields are inserted and placed on the screen. This mask will be used later by the DynPro interpreter.

Field Attributes

DynPro 100 has got the following fields:

Field Name	*Type*	*Format*	*Length*	*Remark*
IFMFCTR-FIKRS	Text	CHAR	15	shows the string 'Finanzkreis'.
IFMFCTR-FIKRS	I/O	CHAR	4	in this field the FM area is typed in. It has got the matchcode FIKRS assigned.
IFMFCTR-FICTR	Text	CHAR	15	shows the string 'Finanz-stelle'.
IFMFCTR-FICTR	I/O	CHAR	10	in this field the funds center is typed in. It has got the matchcode FIST assigned.
OK_CODE	OK			returns function code.

The OK-field serves a special purpose: Buttons have got a function code assigned. Whenever a button is pressed, the input is ended and the function code of the pressed button is written into the OK-field. When the input is ended by pressing the return-key, the value of the OK-field is SPACE.

Flow Logic

The flow logic consists of key words marking the beginning of a section to be processed at an event, module calls and error handling.

Consider the flow logic in figure 4.10: the module D0100_INDEPENDENT is called after the PBO event has occurred. The module D0100_EXIT is called when the user wants to exit the current transaction, the module D0100_OK_CODE is called at the PAI event and the input is valid.

When an error message is issued, all fields enumerated by the FIELD-command can be reentered. The CHAIN-ENDCHAIN-command defines a block in which the FIELD-COMMAND is valid.

```
PROCESS BEFORE OUTPUT.
  MODULE D0100_INDEPENDENT.
  MODULE D0100_MODIFY_SCREEN.
  MODULE D0100_SET_PF-STATUS.

PROCESS AFTER INPUT.
  MODULE D0100_EXIT AT EXIT COMMAND.
    CHAIN.
      FIELD: IFMFCTR-FIKRS, IFMFCTR-FICTR.
* check for illegal characters
      MODULE CHECK_SONDERZEICHEN.
* store key of FM area in a gobal variable
      MODULE D0100_DB_KEY_NOTICE.
* is the user entitled to do the transaction?
      MODULE AUTHORITY_CHECK
 * set a lock on the table FMFCTR, holding the funds centers
      MODULE FMFCTR_ENQUEUE.
* read attributes of funds center
      MODULE FMFCTR_LESEN.
    ENDCHAIN.
* set next DynPro to be executed
    FIELD OK_CODE MODULE D0100_OK_CODE.
```

Figure 4.10: Flow Logic of DynPro 100, Program SAPLFM22

4.3.5 Characteristics of ABAP/4

Some of the ABAP/4 charcteristics are:

- ABAP/4 code is interpreted
- the syntax reminds the user of COBOL and BASIC
- the syntax is context-sensitive
- more than 200 key-words in version 3.0C, with an increasing tendency
- little orthogonality.

From a computer scientist's point of view, the language is very old- fashioned and not well designed. Its size and complexity has grown in time and SAP was not able, or did not want to, redesign the language. This has led to a language full of contradictions and irregularities. Nevertheless, in ABAP/4 there are some concepts worth having a closer look at. For a detailed introduction to ABAP/4 see, e.g., [5], [4], [7] or [10].

Consistent Definitions of ABAP/4 variables and DDic Objects

A major problem of every programmable database system is to keep the definitions of program variables consistent with the definitions made in the database, in the case of R/3 the Data Dictionary.

In R/3 variables can be defined with the 'LIKE'-operator, which has the form '*variable* **LIKE** *DDic object*'. This causes the ABAP/4 interpreter to look up the definition of the DDic object and to use that definition for the ABAP/4 variables.

```
CALL FUNCTION 'ENQUE_EFMFCTR'
      EXPORTING
          FIKRS = G_FIKRS
          FICTR = G_FICTR
      EXCEPTIONS
          FOREIGN_LOCK = 1
          SYSTEM_FAILURE = 2
```

Figure 4.11: Invocation of a Locking Object

E.g. the ABAP/4 statement '**DATA** G_FIKRS **LIKE** FM01-FIKRS **.**' [15] defines a variable called G_FIKRS which has got the same definition as the field FIKRS in the table FM01. Since ABAP/4 is interpreted, every time the variable G_FIKRS is defined, it has got the same definition as the field FM01-FIKRS in the Data Dictionary.

The LIKE-operator can be used with any DDic object, especially tables and structures. This is very important, because when new fields are added or the definition of fields are altered, older programs using the table or structure will still work. Otherwise the process of customizing would not only include the altering of tables but also the altering of all applications using these tables. This would obviously be not feasible.

Locking of Database Tables

To guarantee data consistency, database tables must be locked the moment they are used. As described before, this is done by locking objects and the system automatically generates function modules to request (enqueue) and to end a lock (dequeue). Figure 4.11 shows an example for the invocation of a locking object.

In ABAP/4, database tables cannot be locked directly, all locking must be done via locking objects. Again, the indirection pays off when definitions or dependencies in the Data Dictionary are changed. Old programs will still work after a locking object has been changed.

Persistency

The underlying SQL database is the persistent store for ABAP/4. In addition, ABAP/4 can handle files, but this is recommended for temporary data or information interchange with other programs only.

ABAP/4 has got a built-in dialect of SQL, the so called *Open SQL* language. Open SQL is similar to the usual SQL[16], there are some modifications due to the tight integration in ABAP/4.

It is also possible to use the SQL language of the underlying database, the language is called *Native SQL.* It is not wise to use Native SQL, for the applications may not be usable in other R/3 systems.

Dynamic Screen Modification

Different groups of users are interested in the same objects, but they all want to manipulate it from their point of view. Databases take that into account by views,

[15] excerpt from the include LFM22DEC.

[16] for further information about SQL, see, e.g., [9] or [16].

```
MODULE D0100_MODIFY_SCREEN.
    LOOP AT SCREEN.
      "/ if (( Feldname = 'Finanzkreis' ) und ( TA ist abhängig ) )
      IF ( ( SCREEN-NAME = 'IFMFCTR-FIKRS' )
      AND ( FLG_CALLD = CON_DEPENDANT_TA ) ).
      "/ Feld dient nur zur Anzeige
        SCREEN-INPUT = 0. "/'0A' in HEX
        MODIFY SCREEN.
      ENDIF. "/ SCREEN-NAME
    ENDLOOP. "/ SCREEN.
ENDMODULE. "/ D0100_MODIFY_SCREEN
```

Figure 4.12: Dynamic Screen Modification

R/3 allows to change the screen mask during execution. This is done in a PBO module, figure 4.12 shows an example.

Chapter 5

Customizing and System Evolution

Customizing is a method intended for

- implementing the R/3 System
- enhancing of the R/3 System
- undertaking a release upgrade and system upgrade.

Customizing

- provides the *procedure model*, the work breakdown structure for implementation and enhancement of the R/3 System
- provides tools for system configuration and the necessary documentation
- provides the *customizing project* which gives the user tools for management, processing and evaluation of his or her implementation or enhancement projects
- gives configuration recommendations and tools to enable this
- helps to transfer the System configuration from the development environment into the production environment
- gives a set of tools for system upgrades and release upgrades.

[1]

To implement the system means in the customizing context, to introduce R/3 in the actual company.

5.1 Customizing: The Procedure Model

The procedure model is the basic element of customizing. The aim of the procedure model is a structured organization of the R/3 implementation. [2] It is a high level description of what is to be done.

The procedure model consists of four phases:

[1] taken from [13].
[2] taken from [13].

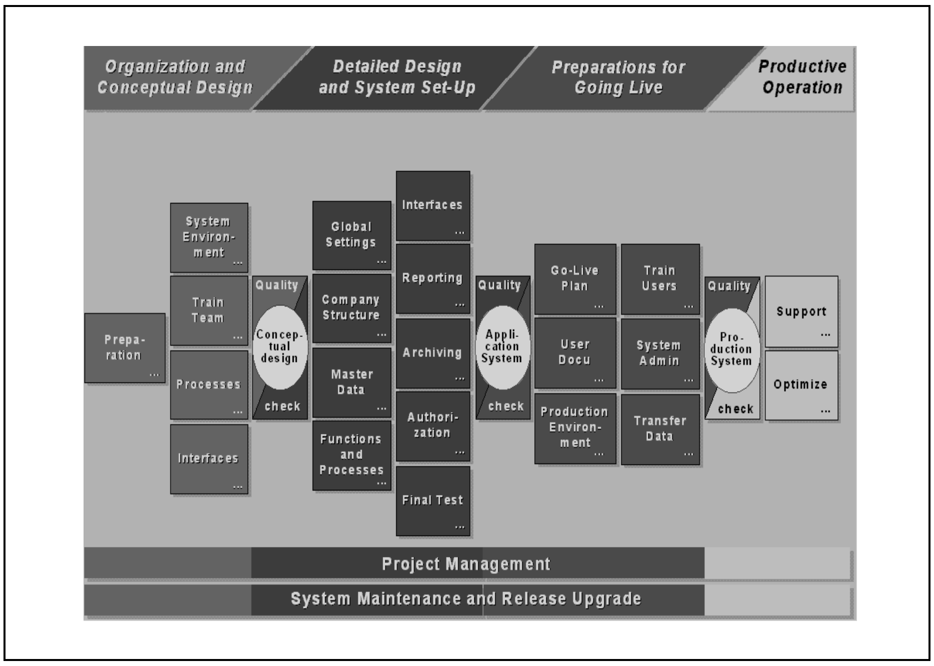


Figure 5.1: The Procedure Model

1. **Organization and conceptual design**. The focus when creating the conceptual design is to use the R/3 reference model to help to work out how the R/3 business application components support the company's processes and functions. Other steps are e.g. to train the project team and to design interfaces and enhancements.

2. **Detailed design and system setup**. The result of the phase 'Detailing and Implementation' is the checked company-specific application system which will be released for phase 3 (production preparation).

3. **Preparations for going live**. The result of the 'Production Preparation' phase is a checked and released production system.

4. **Productive operation**. The result of the phase 'Production' is the organization and execution of a continuous optimization and support of the productive operation.

[3] Figure 5.1 shows the steps of the procedure model in R/3 3.0C.

5.2 Customizing: The Implementation Guide

The implementation guide (IMG) describes what has to be done concretely and is related to the actual customizing project. Furthermore, it contains the necessary sequence of activities and the user can start the appropriate customizing transactions. The IMG is the central element of the customizing process.

Of course, there need to be different IMGs for different companies and projects. SAP introduced four levels of IMGs:

[3] result descriptions are taken from [13].

The SAP Reference IMG contains documentation on all the business application components supplied by SAP.

The Enterprise IMG is a subset of the SAP Reference IMG and contains documentation for those components to be implemented only.

Project IMGs are Enterprise IMG subsets containing just the documentation for Enterprise IMG components to be implemented in particular Customizing projects

Upgrade Customizing IMGs are based either on the Enterprise IMG or on a Project IMG and show, for a given release upgrade, all the documents that are linked to a release note.

[4] IMGs are created by the execution of special SAP transactions.

5.3 Technical Realizing

Customizing is realized by filling in data into certain customizing tables or leaving them with the default data. R/3 standard modules react to customizing data, standard code should not be changed. In same ABAP/4 modules, R/3 provides 'gateways' to user-defined ABAP/4 functions. Obviously this is restricted to foreseen cases only. But within the implementation process it may prove necessary to add additional functionality to the system where no 'gateways' or customizing is provided. The only way is to write programs which are highly linked with standard code, which can lead to trouble when SAP decides to change some of the standard code.

5.4 Common Problems

A big problem when implementing R/3 is to find out which SAP modules [5] and functions provide the required functionality. E.g., employees may be modeled best as suppliers when coming to controlling matters. [6]

A top-down-approach to the problem is

1. to read the functional description of a module in the SAP documentation on a high level
2. to read the documentation 'Functions in Detail', provided by SAP
3. to analyze the EPCs
4. to analyze the data model
5. to check the functionality with appropriate testing data.

Customizing is a long process which involves a lot of reading. The pure amount of documentation for customizing and the problem of finding out the required information and functions shows that this way of customizing a large system is a dead end.

[4] descriptions taken from [13].
[5] FI, TR, HR,...
[6] See [3] for details.

5.5 System Evolution

In an R/3 system, there exists exactly one *original* object and any number of copies. The *Workbench Organizer* takes care, that no copies of an object can be changed. To propagate changes of the original object in the system, the so called *transport system* is used. It replaces old copies of the original object with new copies.

The transport system is used for a system upgrade as well. All changes are transported from the SAP original objects into client 0, the default client. From there, the changes can be transported to the required destinies in the system. Development classes are the basic objects, that can be transported.

Chapter 6

Architectural Aspects of R/3

In this chapter, R/3's technical construction will be sketched briefly. System R was first implemented on mainframes and the resulting structure is still clearly visible.

6.1 Distributed Architecture

Every R/3 System consists of three tiers or layers:

Presentation layer : The graphical user interface (SAPGUI) is run on this layer. No application logic is processed.

Application layer : This layer executes the application logic, like DynPros and ABAP/4 modules. It sends the data to be presented to the user to the presentation layer.

Database layer : This layer holds the system-wide database and the central booking process.

All three layers must exist, but they may be on one computer. In 'normal' R/3 systems, the layers will be on separate computers. Figure 6.1 shows the three layers and the communication between the most important components. [1]

TCP/IP is used as the communication protocol within R/3. With LU 6.2 it is possible to communicate with IBM mainframes. Figure 6.2 shows protocols in R/3.[2]

On top of the communication protocol, a presentation protocol is used for data exchange between the presentation layer and the application layer. This SAP protocol is to minimize the amount of data to be exchanged for a switch from the current screen to the next.

Remote SQL is used to exchange data between the database and the application layer.

6.2 Process Architecture

Figure 6.3 gives a representation of R/3 in OMT.

An R/3 System consists of at least one presentation server, at least one application server and exactly one database server. R/3 does not support distributed databases. SAP claims that distributed databases are not safe enough for practical use.

[1] example taken from [15].

[2] picture taken from [2], translated.

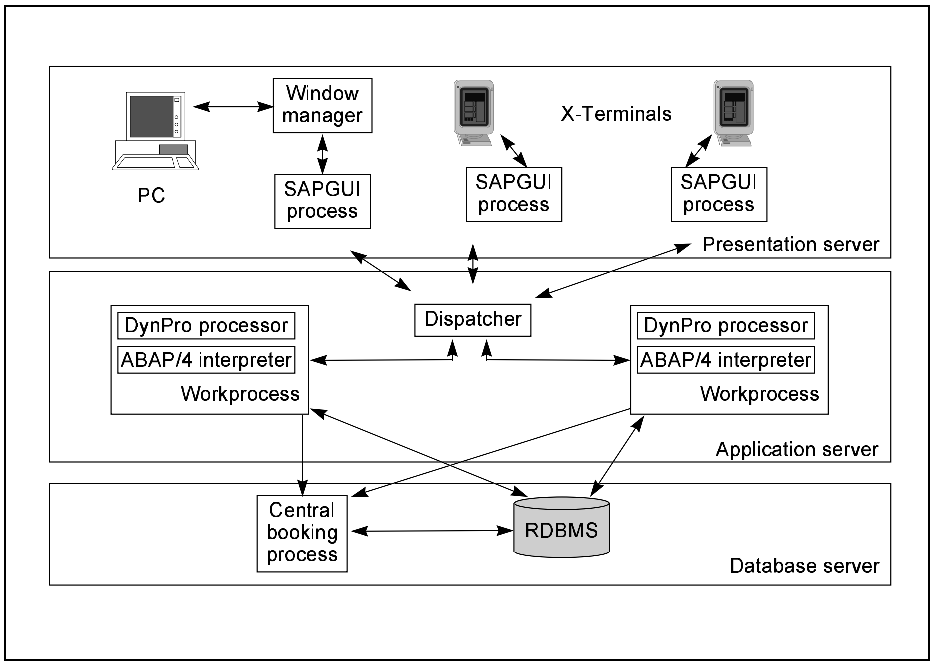


Figure 6.1: Three Tiers in an R/3 System

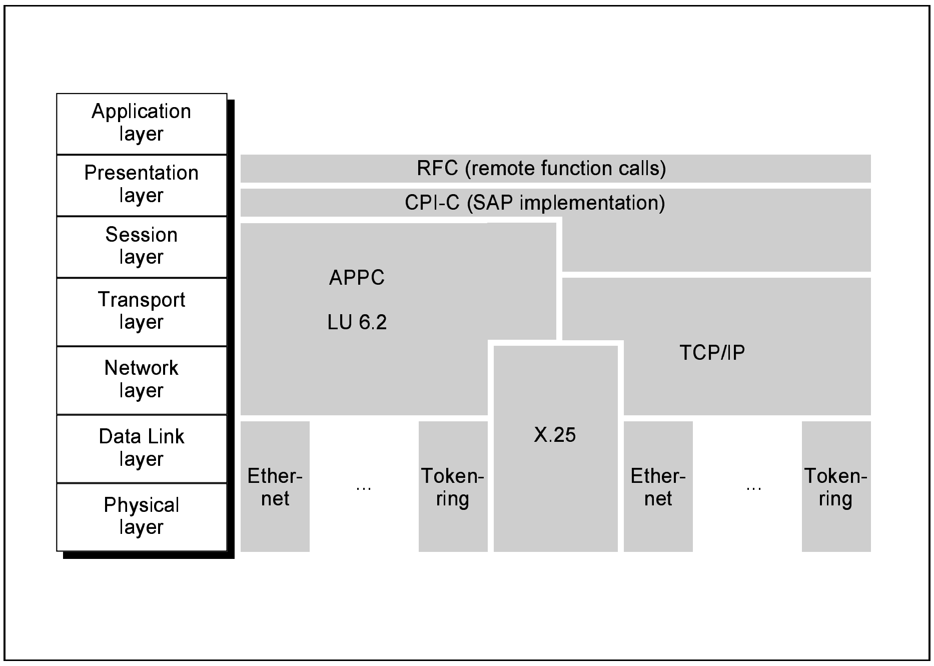


Figure 6.2: Communication Protocols used in R/3

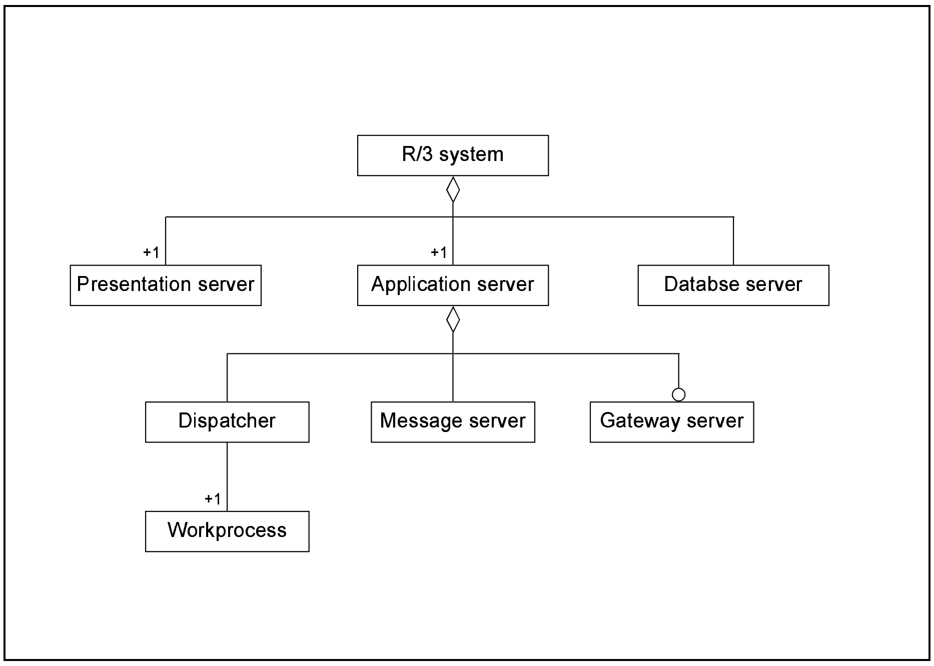


Figure 6.3: R/3 in OMT notation

Every application server has got one message server which handles the communication with other system servers. There can be at most one gateway server in a R/3 System, which handles the communication with other systems, either other R/3 Systems or systems form other vendors.

The central element of an application server is the *dispatcher* which controls the *workprocesses*. The dispatcher assigns the jobs to the appropriate workprocess.

The workprocesses (WP) do the actual work, having their own taskhandler, DynPro processor, ABAP/4 processor and database interface. One can imagine a WP as ‘R/3 in a nutshell’, being specialized on special jobs.

Dialog WPs : Execute DynPros and ABAP/4 modules which are called in the flow logic. At the beginning of a basic cycle [3] the dispatcher assigns the request by the presentation server to an idle Dialog WP, which then does all preparations for screen output, like executing the ABAP/4 modules assigned to the PBO event. After transmitting the new screen layout to the presentation server, the Dialog WP is idle again. When the user has ended the input, the dispatcher will again look for an idle Dialog WP to execute the requested actions.

Batch WPs : Batch WPs are used instead of Dialog WPs when the input is a batch input.

Spool WPs : They do the internal spooling, like printing or transferring data to the database.

Enqueue WPs : An Enqueue WP is specialized on the locking of DDic objects.

Figure 6.4 shows workprocesses in OMT notation.

[3] see chapter 3.

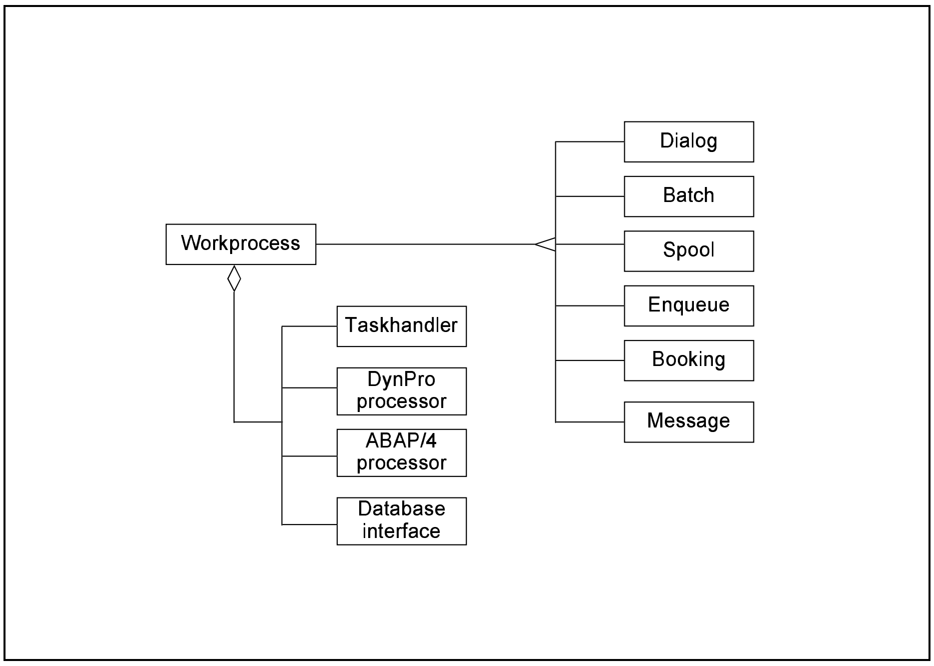


Figure 6.4: Workprocesses

6.3 Gateways to other Systems

R/3 has interfaces to all layers.

- Presentation layer: Intelligent Terminal
- Application layer: Files, CPI-C, RFC, OLE (Windows only), email, EDI, Business API (BAPI, not implemented in R/3 3.0C)
- Database layer: ODBC, Remote SQL.

The interfaces to the application layer have distinct control possibilities, e.g.:

	Data import	Data export	Control from outside	Control of external Software
Files	X	X		
RFC	X	X	X	X
OLE	X	X	X	X
ODBC, remote SQL	X	X		

The best way to interact with R/3 on a program to program base is the RFC mechanism for it has full control and is platform-independent. Function modules can be called via RFCs, they prove again to be a useful concept. [4]

[4] please refer to [6] for further reading about RFCs and R/3.

Conclusion

In R/3, SAP implemented some good concepts and ideas:

- The idea of customizing has proved invaluable to business software. It is a further step in the concept of re-usability of software components, brought to a level that even non-programmers can handle.
- The concept of ACID transactions not only on the level of databases but on the level of business processes overcomes a shortening of the relational model of data. SAP transcations are far nearer to the real world.

But, as always, there less favourable aspects, too:

- The technical realizing of customizing is a dead end.
- The ACID property of SAP transactions is not supported by any high-level programming concept, leaving the fulfillment of the ACID property entirely to the programmer.
- ABAP/4 is irregular and contains many useless concepts.
- R/3 does not scale in a real sense.

Appendix A

Notation: OMT Class Diagrams

In this text OMT[1] class diagrams are used to model R/3 itself. Figure A.1 shows a short summary of OMT symbols.

A class has got a name, attributes and operations. A class **student**,e.g., could have **number** as attribute and **graduate** as operation. An instance would be a concrete student.

The class **person** could be a super-class of the class **student** meaning that every student is a person as well.

The class **student** could be associated with the class **tutors** and the class **student** aggregates over the class **registration data**. Note the difference between an aggregation and an association: When an instance is deleted, associated instances are *not* deleted, aggregated instances *are* deleted. Figure A.2 shows the appropriate OMT class diagram.

See [12] for further details of the OMT Notation.

[1]OMT is a abbreviation for 'Object orientated Modeling Technique'.

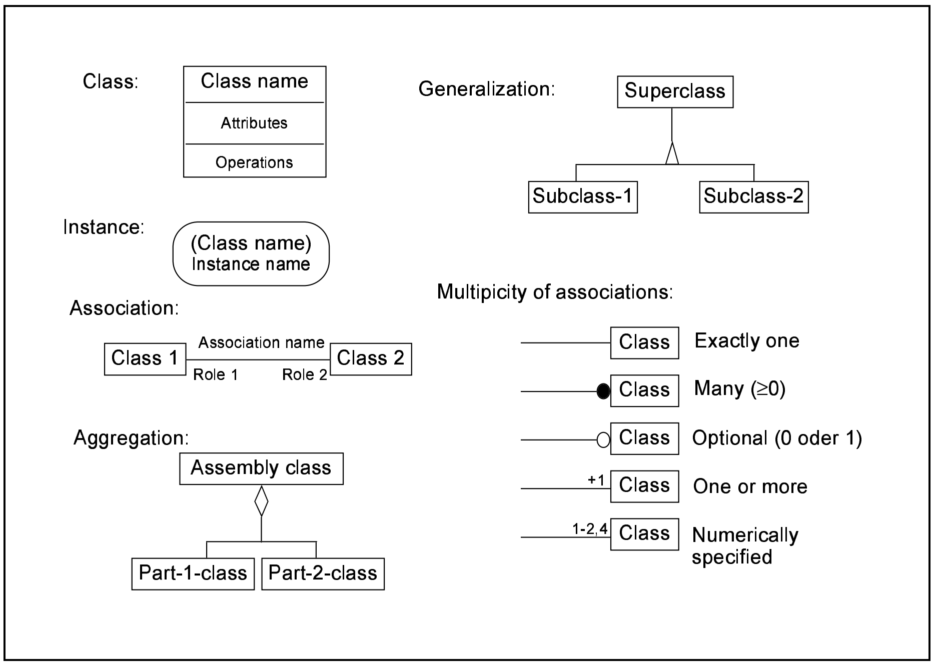


Figure A.1: The OMT notation

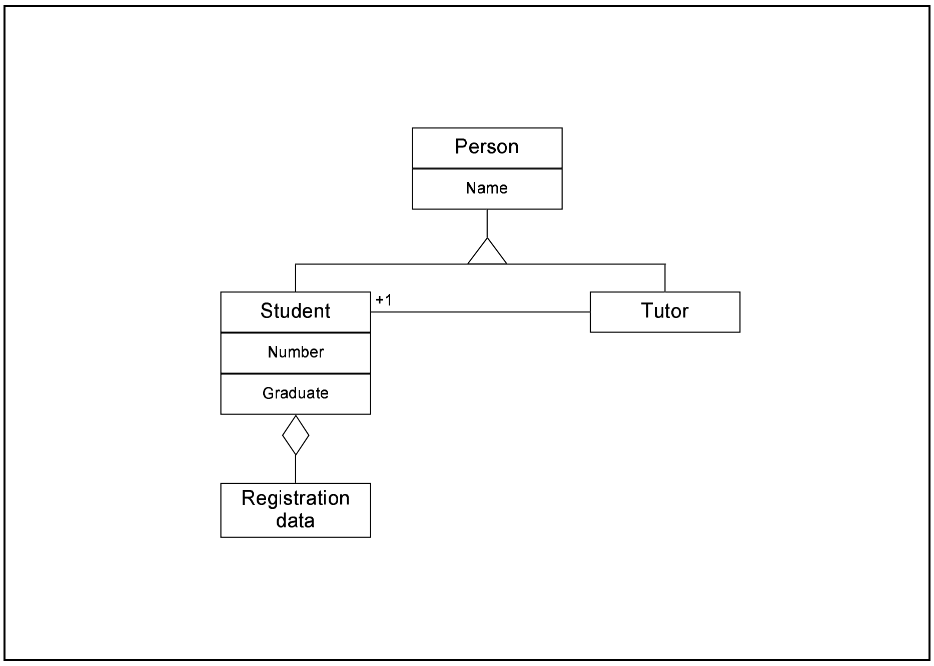


Figure A.2: The Example Screnario

Appendix B

A Meta-Model for R/3

Figure B.1 states the R/3 client/server architecture in an OMT diagram. Figure B.2 shows all R/3 objects mentioned in this text in one diagram.

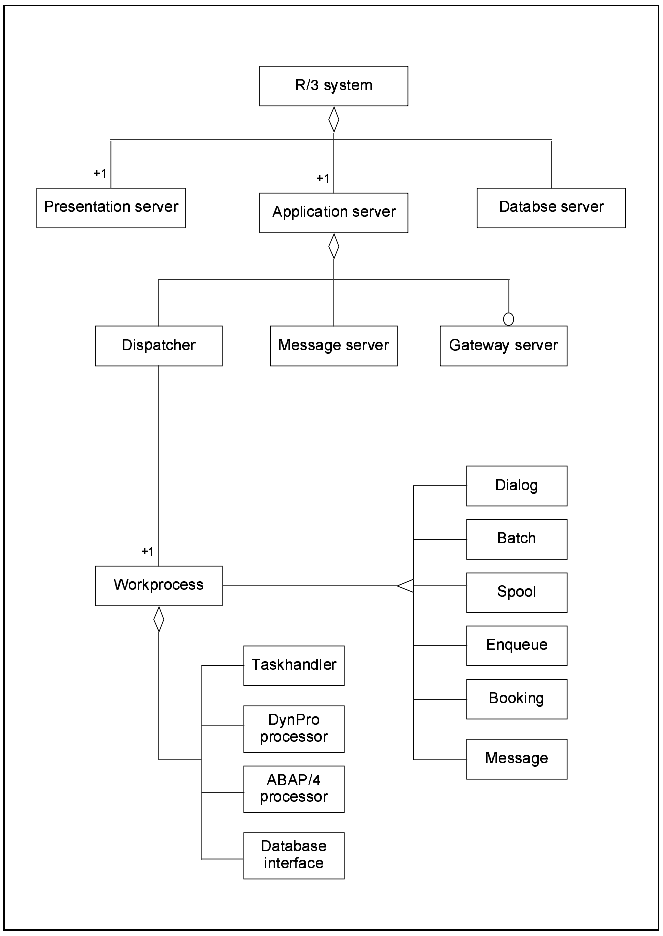


Figure B.1: The R/3 System Architecture

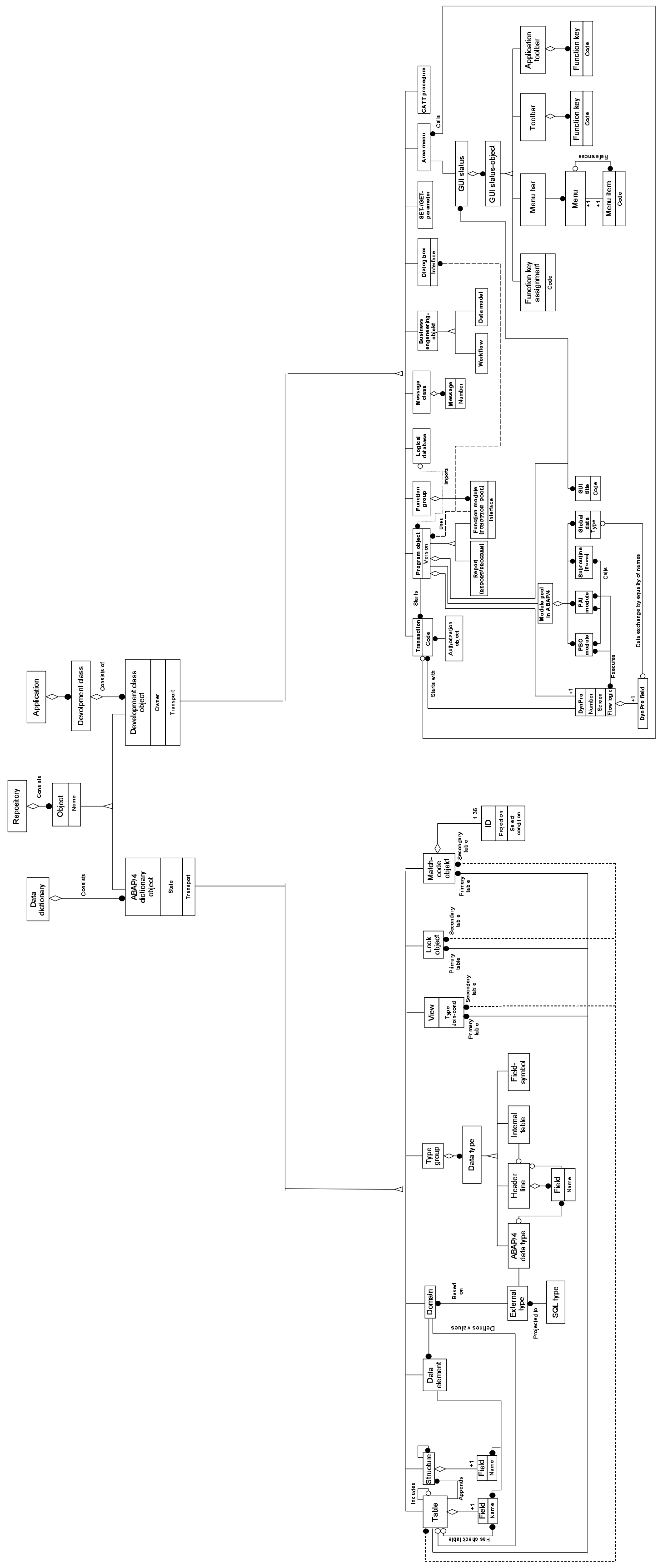


Figure B.2: An Meta-Model for R/3 Objects

Appendix C

Source Code of FM22, DynPro 100

The complete source code can be found in the program SAPLFM22. The code given here is a subset which is executed during DynPro 100. The reader should get an impression of what ABAP/4 programs look like.

```
* Programmkopf
FUNCTION-POOL FM22 MESSAGE-ID FI.
...
*-----------------------------------------------------------------------
* DEC - lokale Datenbeschreibungen Tabellen / Daten / Field-Symbols
*-----------------------------------------------------------------------
INCLUDE LFM22DEC.
...
*-----------------------------------------------------------------------
* Interne Tabellen
*-----------------------------------------------------------------------

* Interne Tabelle fr die Finanzstelle zur Datenhaltung
* und zur Datenbergabe zwischen den Dynpros
DATA: BEGIN OF I_FMFCTR OCCURS 10.
        INCLUDE STRUCTURE IFMFCTR.
DATA: END OF I_FMFCTR.

* Sicherung unbearbeitetes Dynpro
DATA: BEGIN OF I_OLD_FMFCTR.
        INCLUDE STRUCTURE IFMFCTR.
DATA: END OF I_OLD_FMFCTR.
...

TABLES:
        "/ Finanzkreis
        FM01,
        "/ Texte zum Finanzkreis
        FM01T,
        "/ Finanzstelle
        FMFCTR,
        "/ Interne Tabelle fr die Dynprofelder der Finanzstelle
        IFMFCTR.
```

```
* Feldleiste fr die Finanzstelle
DATA: BEGIN OF FFMFCTR.
        INCLUDE STRUCTURE IFMFCTR.
DATA: END OF FFMFCTR.

DATA:
      "/Finanzkreis
      G_FIKRS           LIKE FM01-FIKRS,
      "/Finanzstelle
      G_FICTR           LIKE FMFCTR-FICTR,
      "/Transaktionscode
      G_TCODE           LIKE SY-TCODE,
      "/Flag um zu signalisieren, ob die TA / Function abhngig aufgerufen wurde.
      FLG_CALLD         LIKE SY-CALLD  VALUE 0.
...

MODULE D0100_MODIFY_SCREEN.
*---------------------------------------------------------------------*
*    Dynamische Bildmodifikation fr Dynpro 0100:                     *
*    - Feld 'Finanzstelle' auf Lnge 10 reduzieren                    *
*    - Feld 'Finanzkreis' ist bei Neuanlage aus der Grafik heraus      *
*      nur ein Anzeigefeld                                             *
*---------------------------------------------------------------------*
*---------------------------------------------------------------------*

"/ Schleife ber alle Dynprofelder
   LOOP AT SCREEN.
     "/ if (( Feldname = 'Finanzkreis' ) und ( TA ist abhngig ) )
     IF ( ( SCREEN-NAME = 'IFMFCTR-FIKRS' )
        AND ( FLG_CALLD = CON_DEPENDANT_TA ) ).
        "/ Feld dient nur zur Anzeige
        SCREEN-INPUT  = 0.        "/'OA'  in HEX
        MODIFY SCREEN.
     ENDIF. "/ SCREEN-NAME
   ENDLOOP. "/ SCREEN.

ENDMODULE. "/ D0100_MODIFY_SCREEN
...

MODULE D0100_EXIT.
*---------------------------------------------------------------------*
*      Funktionen, die die aktuelle Bearbeitung beenden               *
*      ohne die PAI-Module, -Prfungen zu aktivieren.                   *
*      LEAVE TO TRANSACTION lst automatisch das Freigeben aller       *
*      Sperren der Transaktion aus!                                    *
*---------------------------------------------------------------------*
  "/ Zwischenspeichern des OK_CODE
  SAV_OK_CODE = OK_CODE.
  CLEAR OK_CODE.
  "/ Auswerten des OK_CODE
  CASE SAV_OK_CODE.
    "/  ENDE = Beenden
    WHEN 'ENDE'.
```

```
         SET SCREEN 0.
         LEAVE SCREEN.
    "/  EINS = In Anlegen-TA verzweigen
    WHEN 'EINS'.
         "/ Rufe Transaktion 'Anlegen'
         LEAVE TO TRANSACTION TR_FICTR_INS.
  ENDCASE.

ENDMODULE. "/ D0100_EXIT
...

MODULE CHECK_SONDERZEICHEN INPUT.
  FIELD-SYMBOLS <F>.

  IF SY-TCODE = TR_FICTR_INS
  OR SY-TCODE = TR_FICTRHI_MNTN.
    IF IFMFCTR-FICTR CA CON_SONDERZEICHEN.
      ASSIGN IFMFCTR-FICTR+SY-FDPOS(1) TO <F>.
      MESSAGE E669 WITH <F>.
    ENDIF.
  ENDIF.
ENDMODULE.                 " CHECK_SONDERZEICHEN  INPUT
...

MODULE D0100_DB_KEY_NOTICE.
*  Schlssel der Finanzstelle in globalen Variablen merken
   "/Finanzkreis
   CONDENSE IFMFCTR-FIKRS NO-GAPS.
   G_FIKRS = IFMFCTR-FIKRS.
   "/Finanzstelle
   CONDENSE IFMFCTR-FICTR NO-GAPS.
   G_FICTR = IFMFCTR-FICTR.
   "/if ( TA ist unabhngig )
   IF ( FLG_CALLD = CON_INDEPENDANT_TA ).
      "/Variablen initialisieren, die bei abhngigem Aufruf aus der Grafik kommen.
      "/Lese den Finanzkreis
      SELECT SINGLE *
      FROM FM01
      WHERE FIKRS = G_FIKRS.
      "/Objektnummer des Finanzkreises bernehmen
      G_FMA_OBJNR = FM01-OBJNR.
      "/Einfgen ohne kopieren
      FLG_COPY = CON_NEIN.
      "/Variable initialisieren
      CLEAR G_REF_FICTR.
   ENDIF.
ENDMODULE. "/ D0100_DB_KEY_NOTICE
...

MODULE FMFCTR_ENQUEUE INPUT. "/Prozedur zum Sperren einer Finanzstelle aufrufen
  PERFORM FMCTR_ENQUEUE.
ENDMODULE. "/ FMFCTR_ENQUEUE

FORM FMFCTR_ENQUEUE.
```

```
"/ if ( Anlegen oder ndern einer Finanzstelle )
   CHECK ( ( G_TCODE = TR_FICTR_INS )
      OR   ( G_TCODE = TR_FICTR_UPD ) ).
      "/ Sperre fr eine Finanzstelle anfordern
      CALL FUNCTION 'ENQUEUE_EFMFCTR'
           EXPORTING
             FIKRS    = G_FIKRS
             FICTR    = G_FICTR
           EXCEPTIONS
             FOREIGN_LOCK   = 1
             SYSTEM_FAILURE = 2.
      CASE SY-SUBRC. "/ Fehlerbehandlung
        WHEN 1.  "/bereits von anderem User gesperrt (FOREIGN_LOCK)
             MESSAGE E641 WITH G_FICTR.
        WHEN 2.  "/ SYSTEM_FAILURE bei einer Sperranforderung
             MESSAGE A521 WITH G_FICTR.
      ENDCASE. "/ SY-SUBRC
ENDFORM. "/ FMFCTR_ENQUEUE
...

MODULE FMFCTR_LESEN.
  PERFORM FMFCTR_LESEN. "/ Aufruf der Prozedur zum Lesen der Finanzstelle
ENDMODULE. "/ FMFCTR_LESEN.

FORM FMFCTR_LESEN.
  DATA: L_FMFCTR_EXISTS LIKE CON_JA. "/Flag Finanzstelle existiert
  IF ( G_TCODE = TR_FICTR_UPD OR G_TCODE = TR_FICTR_SHOW ).
        PERFORM FMFCTR_LESEN_UPD USING    G_FIKRS          "/VALUE
                                          G_FICTR          "/VALUE
                                 CHANGING L_FMFCTR_EXISTS. "/VALUE
  ELSE.
        PERFORM FMFCTR_LESEN_INS USING G_FIKRS       "/VALUE
                                       G_FICTR       "/VALUE
                                       FLG_COPY      "/VALUE
                                       G_REF_FICTR.  "/VALUE
  ENDIF. "/G_TCODE
ENDFORM. "/FMFCTR_LESEN.

FORM FMFCTR_LESEN_INS USING    VALUE(P_FIKRS)
                               VALUE(P_FICTR)
                               VALUE(P_COPY)
                               VALUE(P_REF_FICTR).
  ...
  IF ( L_FMFCTR_EXISTS = CON_JA ). "/ Fehlermeldung und im Dynpro bleiben
    MESSAGE E642 WITH P_FIKRS P_FICTR.
  ...
ENDFORM. "/FMFCTR_LESEN_INS
...

MODULE D0100_OK_CODE INPUT.
*-----------------------------------------------------------------------*
* Auswertung der Benutzerkommandos im Dynpro 100                        *
*-----------------------------------------------------------------------*
```

```
  "/ Zwischenspeichern des OK_CODE
  SAV_OK_CODE = OK_CODE.
  CLEAR OK_CODE.

  "/ Auswerten des OK_CODE
  CASE SAV_OK_CODE.
    "/ ENTER
    WHEN SPACE.
      "/ Folgedynpro 200 aufrufen
      SET SCREEN 200.
      LEAVE SCREEN.
  ENDCASE.

ENDMODULE. "/ D0100_OK_CODE
```

Bibliography

[1] E.F. Codd, 'A Relational Model of Data for Large Shared Data Banks', *Communications of the ACM*, Vol. 13, No.6 (June 1970)

[2] Rüdiger Buck-Emden, Jürgen Galimow, *Die Client/Server-Technologie des Systems R/3*, Addison-Wesley, Bonn 1994

[3] Marko-Andreas Fricke, Stephan Frölich, *Flexible Haushaltsplanung und Mittel-Controlling für einen Modellfachbereich*, Diplomarbeit, Universität Hamburg, 1997.

[4] R. Kretschmer, W. Weiss, *Developing SAP's R/3 Applications with ABAP/4*, Sybex, San Francisco, Paris, Düsseldorf, Soest, 1996

[5] Christian Koch, *Einführung in die ABAP/4 Entwicklungsumgebung*, research paper, 1996

[6] Sebastian Lutz, *Eine polymorph-typisierte Schnittstelle zwischen SAP R/3 und Tycoon*, Master-Theses, Universität Hamburg, 1997.

[7] Florian Matthes, *Betriebswirtschaftliche Informationssysteme am Beispiel SAP R/3*, lecture notes, Technische Universität Hamburg-Harburg, http://www.sts.tu-harburg.de, 1997

[8] Florian Matthes, *Persistente Objektsysteme*, Springer-Verlag, 1993

[9] E. Lynch, *Understanding SQL*, MacMillan, London England, 1990

[10] Bernd Matzke, *ABAP/4: Die Programmiersprache des SAP Systems R/3*, Addison Wesley, Bonn, 1996

[11] J. Peterson, *Petri Net Theory and the Modelling of Systems*, Prentice Hall, Englewood Cliffs, Nj., 1981

[12] Rumbaugh, Blaha, Permerlani, Eddy, Lorensen *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, New Jersey 1992

[13] R/3 System Release 3.0C, *Online Documentation*, March 96

[14] Matthes, Schmidt, 'Datenbankmodelle und Datenbanksprachen' in: Schmidt, Lockemann, *Datenbankhandbuch*, Springer-Verlag, 1997 (to appear)

[15] Liane Will, *R/3-Administration*, Addison-Wesley, Bonn 1995

[16] Stephan Ziemer, *SQL*, 1995, http://www.sts.tu-harburg.de/~people/st.ziemer.