Understanding SAP R/3
A Tutorial for Computer Scientists

Florian Matthes Stephan Ziemer

Technical University Hamburg-Harburg, Germany
http://www.sts.tu-harburg.de/~f.matthes

March 1998 (R 1.0)

Abstract

SAP is the market leader for integrated business administration systems, and its SAP R/3 product is a comprehensive enterprise resource planning software system which integrates modules for finance, material management, sales and distribution, etc. Despite its international commercial success, the models, languages and architecture of this system are to a large degree unknown to computer scientists.

The goal of this tutorial is to present the distributed system architecture, the data model, the database programming language, the transaction and process model and the system evolution concepts of SAP R/3 from a computer science perspective and to relate them to established database and distributed system concepts. The presentation will help attendees understand how SAP R/3 relates to their own research and development work and will provide a well-structured foundation for a further study of SAP R/3's innovative system concepts.

Contents

1 SAP R/3: Past, Presence and Future

The Past The company SAP, Walldorf, Germany, was founded in 1972 by five IBM programmers. Their core business idea was to build *one* program suitable for *many* companies instead of writing essentially the same business software again and again for different companies. The system abstracts from the needs of a concrete company and implements the processes of a generic company.

The 'R' in R/3 stands for 'real time' which means that the system is interactive and not based on batch processing. '3' means that it is the third major version of the program, though there has never been a system called 'R/1'. R/3 was introduced in 1992 and by the end of 1995 more than 5,200 R/3 systems were installed worldwide. The latest release of R/3 is 4.0 (Feb 1998). There is still a number of R/2 installations, but SAP intends to move this customer basis to R/3.

R/3 is a package of standard international business applications for areas such as Financial Accounting, Controlling, Logistics and Human Resources. R/3 provides an enterprise solution for all these application areas in a distributed client/server environment. Using R/3, 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 R/3. The system is available on many hardware platforms and for many operating systems (Unix / NT).

To use R/3 for a concrete company, it is necessary to 'customize' R/3 to suit the specific needs. This kind of software is called 'standard business software' or standard ERP software (enterprise resource planning). Ideally, a customized R/3 installation can follow the evolution of R/3 (taking into account new tax or law regulations, new currencies, etc.) without additional development effort.

The Future Hot *technical* topics for the future evolution of R/3 are [SAPHome 1997] *componentization* (customers will be able to install and evolve independent releases of R/3 modules), *business objects* and BAPIs (newly developed client applications written in Java, VisualBasic, Delphi etc. will have stable, semantically-rich APIs to the R/3 backend), the *R/3 Business Engineer* (customization will take place at a higher abstraction level than screens and tables), and OO ABAP (a fully-upward-compatible extension of ABAP/4 with late binding and a form of inheritance).

The Presence R/3 in its current release 4.0 is a massive client/server development and application system which uses relational databases (Oracle, ADABAS-D, Informix, ...) as its back-end. Cooperation between functions takes place by a traditional, tight database integration. In 1994, R/3 already comprised 7.000.000 lines of code, 100.000 function calls, 20.000 distinct functions 21.000 reports, 17.000 menu definitions and 14.000 function modules [Bundesministerium für Forschung und Technologie 1994]. There are 500 developers working on the core R/3 system, its size grows annually by an estimated 10 percent. Each customer has to install the full R/3 application which consumes several GB on disk excluding any actual business data.

The rest of this tuorial gives you an idea of the following technologies underlying R/3 which are also the basis for R/3's future development.

- The integrated R/3 repository
- Enterprise modeling with R/3
- Objects of the R/3 data dictionary
- Programming in the large
- Programming in the small
- Customizing R/3: concepts and techniques
- R/3's process and system architecture

OMT class diagrams (similar to UML class diagrams) are used for conceptual modeling purposes [Rumbaugh *et al.* 1991; Fowler and Scott 1997].

2 The Integrated R/3 Repository

Similar to a DBMS, R/3 relies heavily on meta-data describing its own data structures, functions (modules) and processes (workflows). Consistency between the meta-data and the actual data is preserved either manually or automatically by the development tools. Using the R/3 Repository Information System, developers can access and browse (textual, tabular and graphical represention of) repository objects. This section provides a bird-eye view of the repository and its major conceptual entities (see Fig. 4).

2.1 Integrated Analysis, Design and Implementation

R/3 uses three levels of abstraction to describe the mapping between 'real-world entities' and R/3 system entities. At each level, a process view, a function view and a data view can be distinguished. These views are connected rather loosely.

At the topmost level, the **analysis level**, a description of R/3 is given. Processes are modeled with EPCs (Event-controlled Process Chains) which describe how business processes are carried out using R/3. 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 R/3 modules, each module serving a special functional purpose within the company. For example, the module HR (human resources) helps to manage the staff data.

SAP and R/3 use SERM (Structured Entity Relationship Model) for data modeling. SERM was invented by SAP but is closely related to the common entity relationship model.

At the second level, the **design level**, processes are modeled with *workflows* which are user-defined EPCs. There is no notation to specifiy functions at this level. The data model of the analysis level is mapped to relational tables and foreign keys at this level. A data dictionary and data dictionary tools help to maintain the association between SERM-entities (and relationships) and relational tables.

At the lowest level, the **implementation level**, processes are implicit (transaction sequences consisting of functions working on tables). Functions are represented as ABAP/4 programs and reports. Data is modeled by SQL tables and ABAP/4 variables. ABAP/4 is the programming language of R/3 covered in Sec. 5.

Virtually all parts of the repository are, like any other information in R/3, stored in the underlying SQL database.

Figure 1 shows the concepts and languages at each level. Figure 2 shows the main objects of each level. Figure 3 enumerates the modules shipped with R/3 3.0.

2.2 Coexistence of Multiple R/3 Clients

An R/3 installation is partitioned 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 client used for actual business processing should be in a separate system to avoid side effects by changes of global settings (see also Fig. 5).

2.3 Application and System Evolution

Application and system evolution in the presence of persistent data is a serious problem to be solved by standard business software. Updates have to be performed on a "live" system and customizing adjustments should be preserved across releases. Moreover, name clashes between customization code and newly developed code of the standard system have to be avoided.

In R/3, the namespace is global and flat for each object type. For example, tables must have system-wide unique names, but a program object can have the same name as a table.

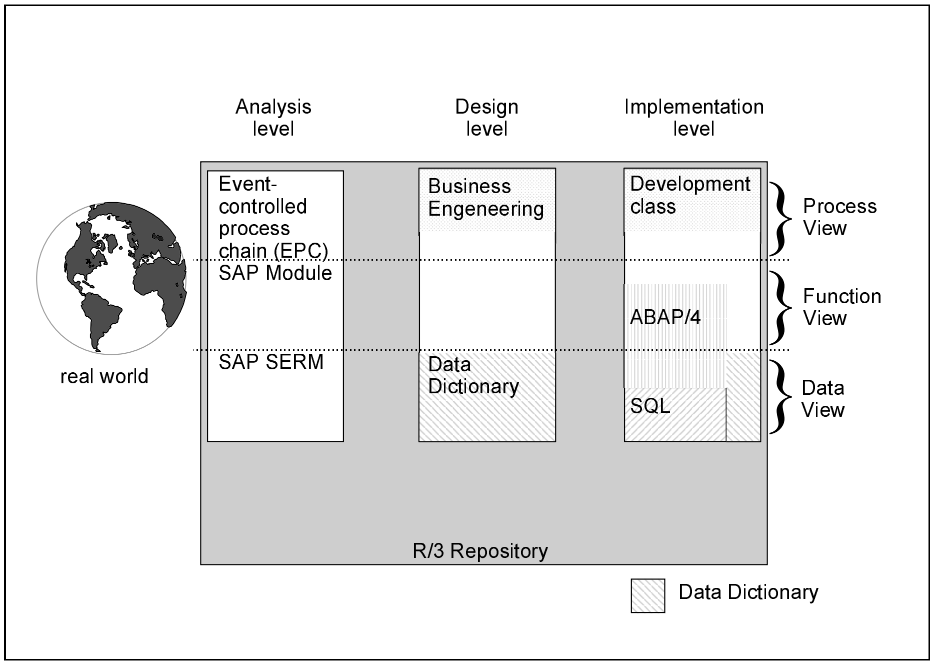


Figure 1: Concepts and Languages of the R/3 Repository

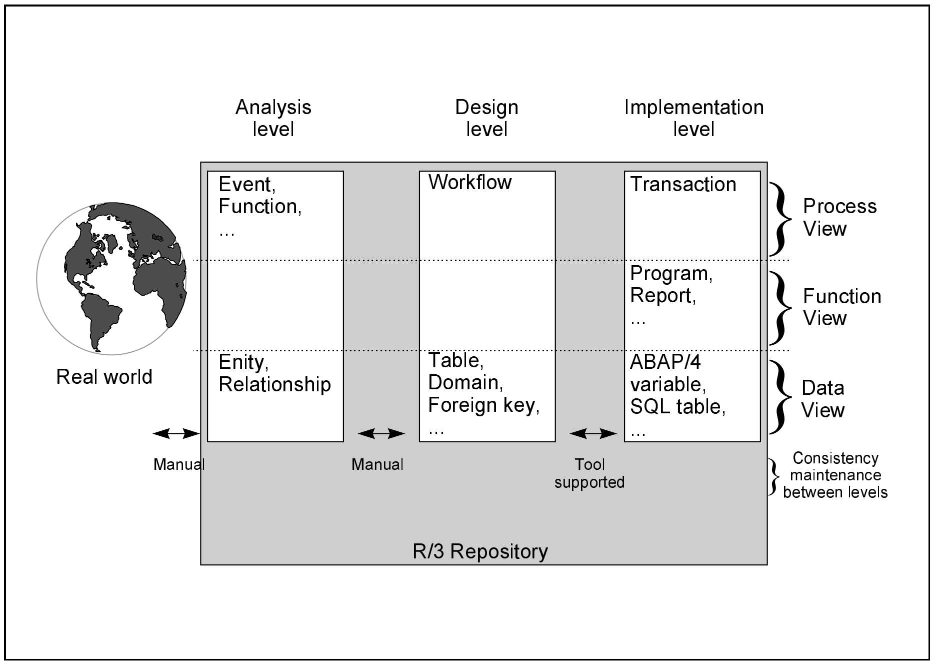


Figure 2: Objects of the R/3 Repository

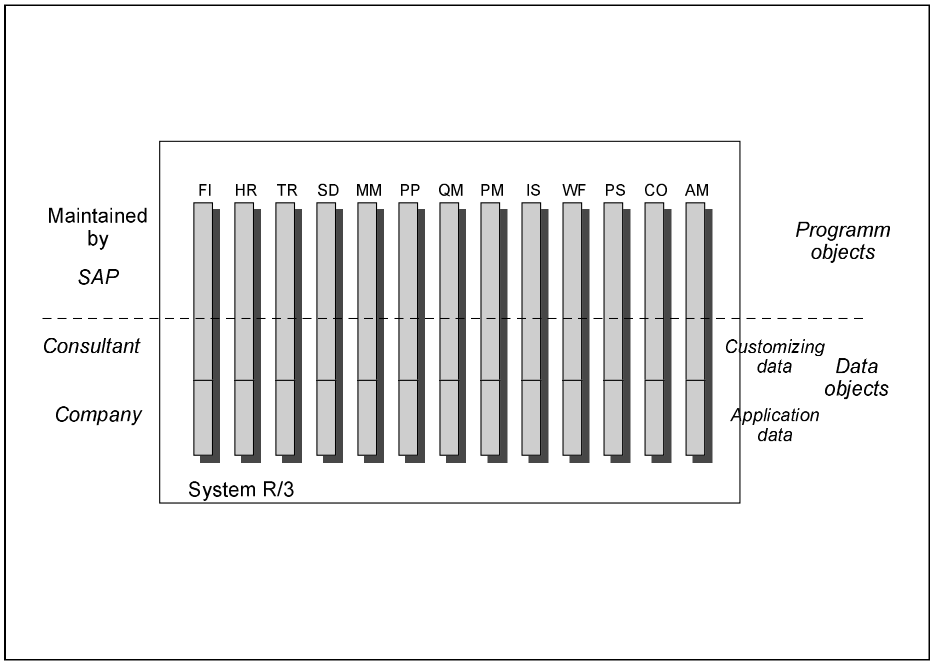


Figure 3: Logical Partition of R/3 into Modules

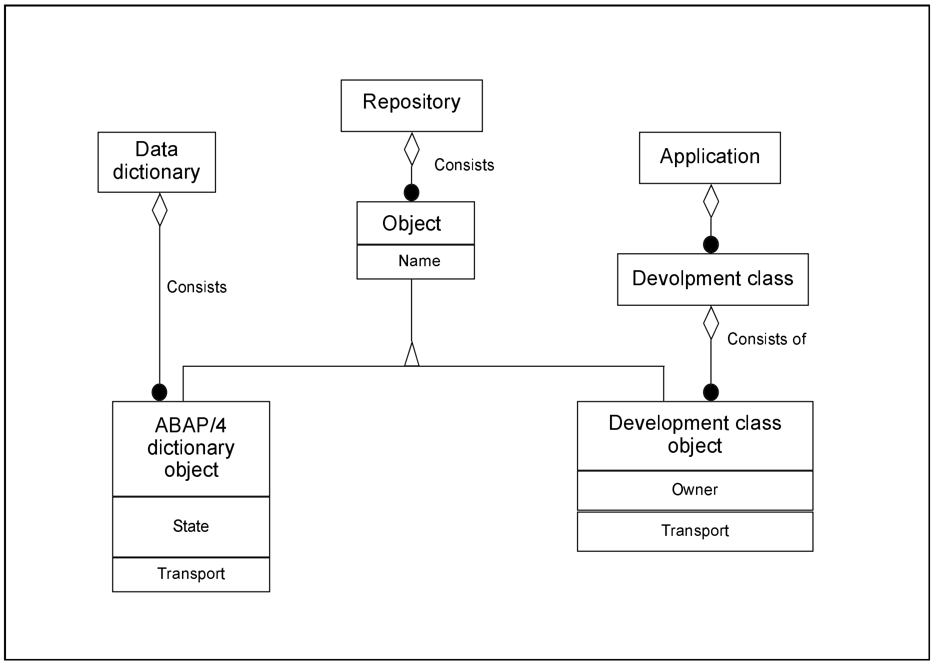


Figure 4: The R/3 Repository

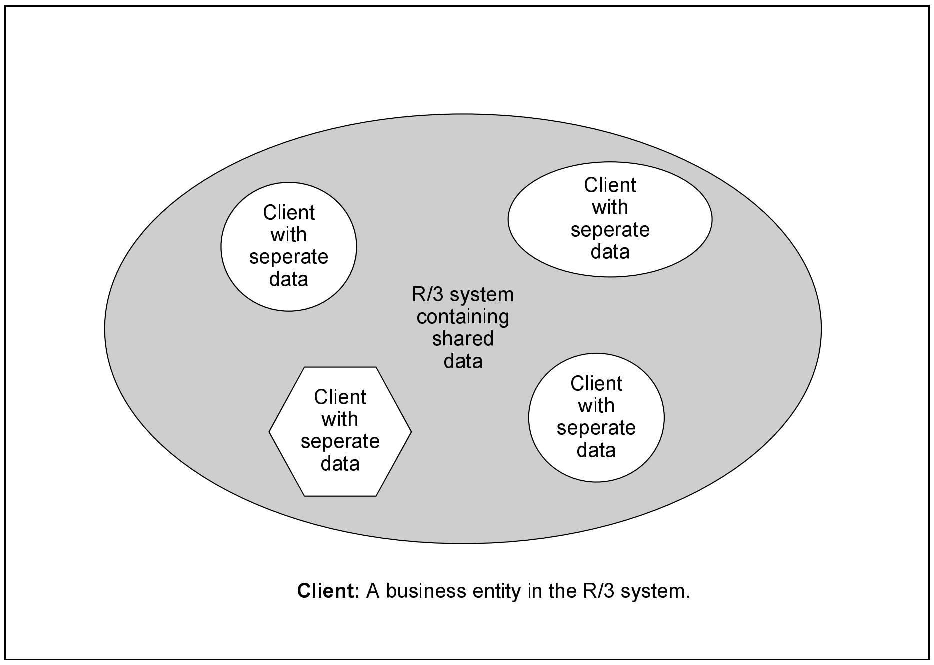

Client with seperate data
Client with seperate data
R/3 system containing shared data
Client with seperate data
Client with seperate data
Client: A business entity in the R/3 system.

Figure 5: The Notion of Clients in R/3

There are thousands of named objects of all kinds and combined with a strict limitation of length (approx. 8 characters), the names are by no means self-explanatory. To worsen the situation, English and German names and abbreviations are mixed.

SAP decided to leave the 'name range' of all objects with a name that starts either with a Y or a Z to customer objects. There are exceptions from this rule leading to some confusion. SAP guarantees that there will be no name clashes with R/3 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.

2.4 Running Example: FM Areas and Funds Centers

Throughout this tutorial, the following tiny fragment of financial management will be used to illustate technical aspects of R/3.

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 [SAP AG 1996].

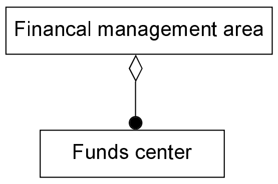
A funds center must be assigned to exactly one FM area, several funds centers can be assigned to the same FM area. A funds center cannot exist without the superordinated FM area. See [SAP AG 1996] for more details.

Figure 6 describes FM area and funds center using the OMT notation.

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

3 Enterprise Modeling with R/3

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



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 6: FM Area and Funds Center in OMT

3.1 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. In R/3 terminology, a 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 constraint 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.
- an '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 a cardinality constraint:

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.

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

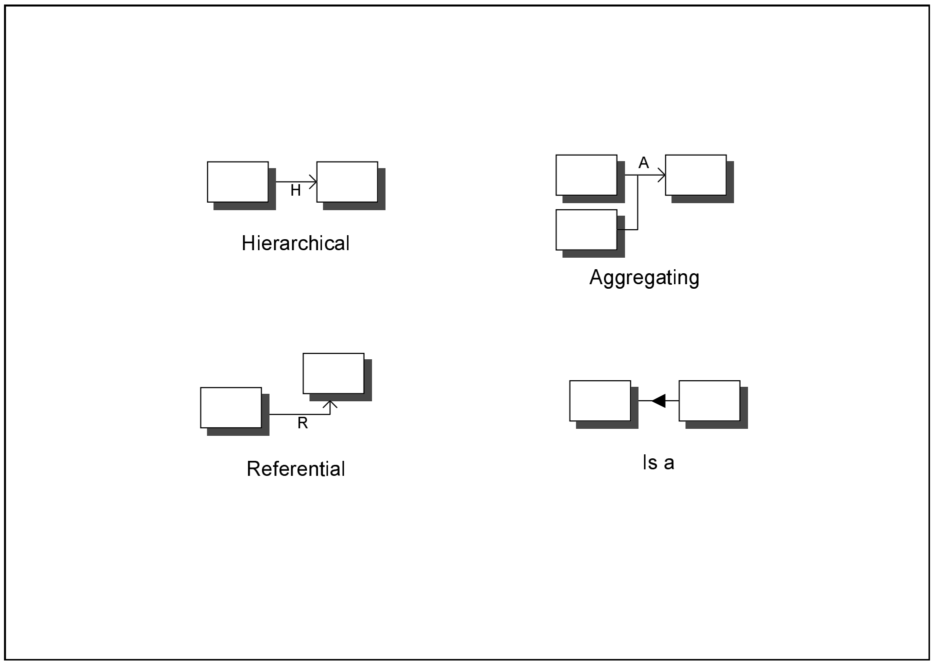


Figure 7: Types of Relationships in the SAP SERM Data Model

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.

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

Entities have some additional attributes, like a unique entity type number and flags to indicate whether the data is changed during customizing or during normal system operation and whether the underling implementation is a table or a view. See Fig. 7 and Fig. 8.

Figure 9 is an excerpt from the data model for FM areas and funds centers. The full R/3 data model documented through this notation is distributed by SAP as huge posters which are several square meters in size.

3.2 Functional Decomposition: R/3 Modules

R/3 consists of large-grain sub-applications, so called *modules* corresponding to the classical functional structure of a company. Examples are Financial Management (FI), Materials Management (MM) and so on. These applications work with the shared data and are highly interrelated which makes it virtually impossible to use or customize one module in isolation.

SAP is currently developing “top down” a so-called business framework based on the notion of business components. A business component exposes its functionality by means of business-objects (with stable BAPIs) to other business components and to third-party client applications.

Business objects will help to close the large semantic gap between the rather low-level ABAP/4 function modules that have evolved “bottom up” over the past decades and a problem-oriented view on R/3 as a collection of logical components that have to be customized, deployed and integrated with non-SAP software components.

3.3 Process Modeling: R/3 Reference Model and EPCs

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

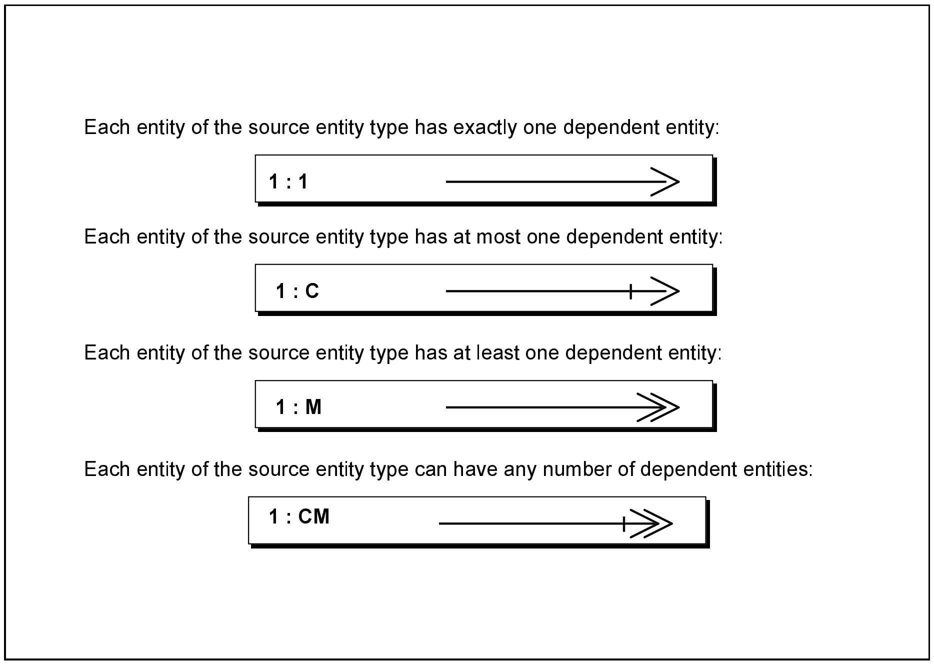

Each entity of the source entity type has exactly one dependent entity:
1 : 1
Each entity of the source entity type has at most one dependent entity:
1 : C
Each entity of the source entity type has at least one dependent entity:
1 : M
Each entity of the source entity type can have any number of dependent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

Figure 8: Cardinalities of Relationships in SAP SERM

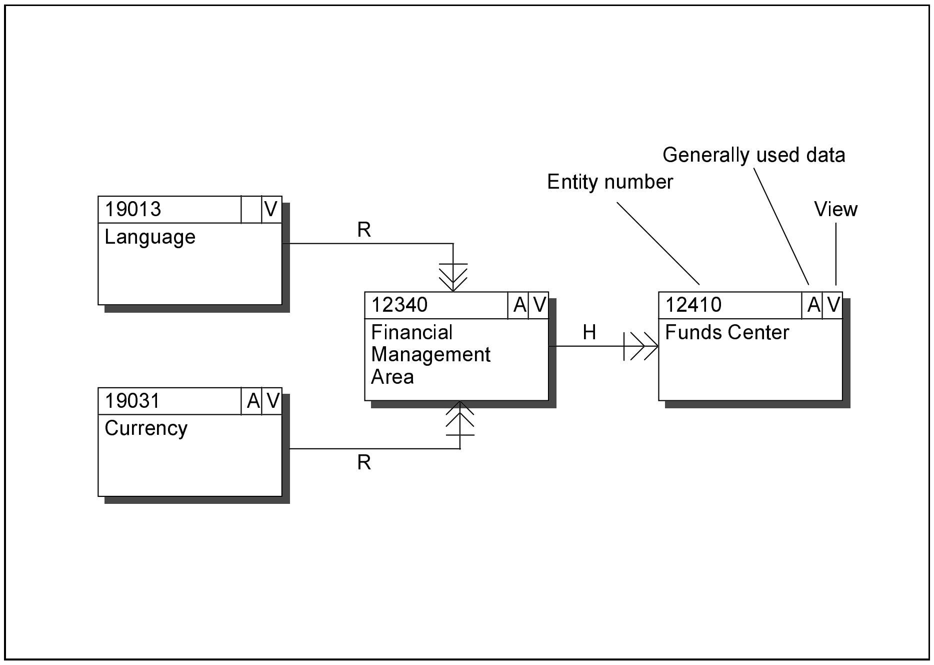


Figure 9: Excerpt form the R/3 Data Model for FM Areas

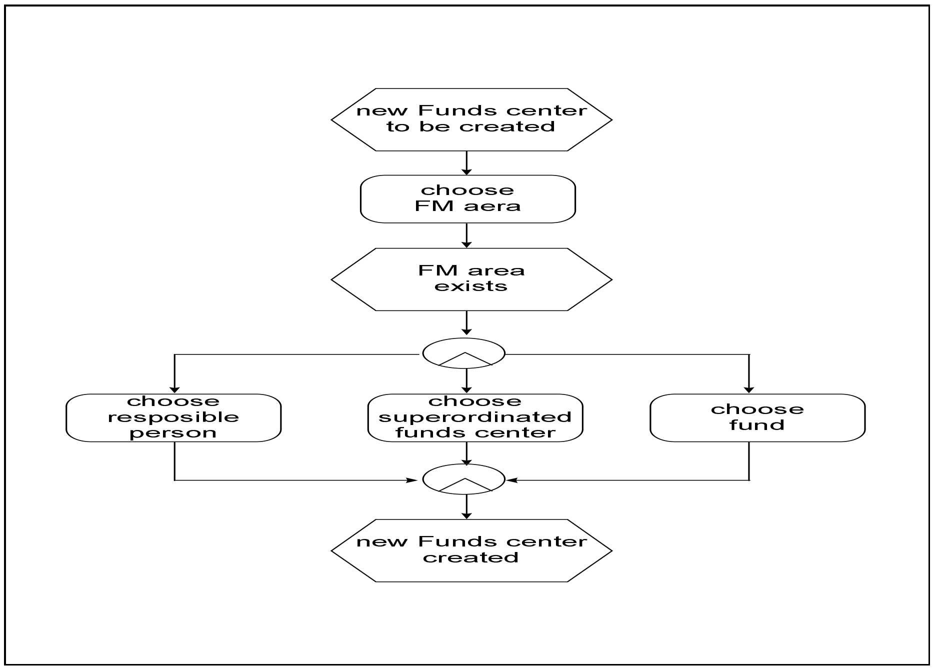


Figure 10: EPC to create a new funds center

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 [SAP AG 1996] using the following graphical elements:

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 10 shows a concrete 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 semantics 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.

4 Objects of the R/3 Data Dictionary

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 analysis model.

The main tool at 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 section 5
- system oriented elements, covered in section 5.

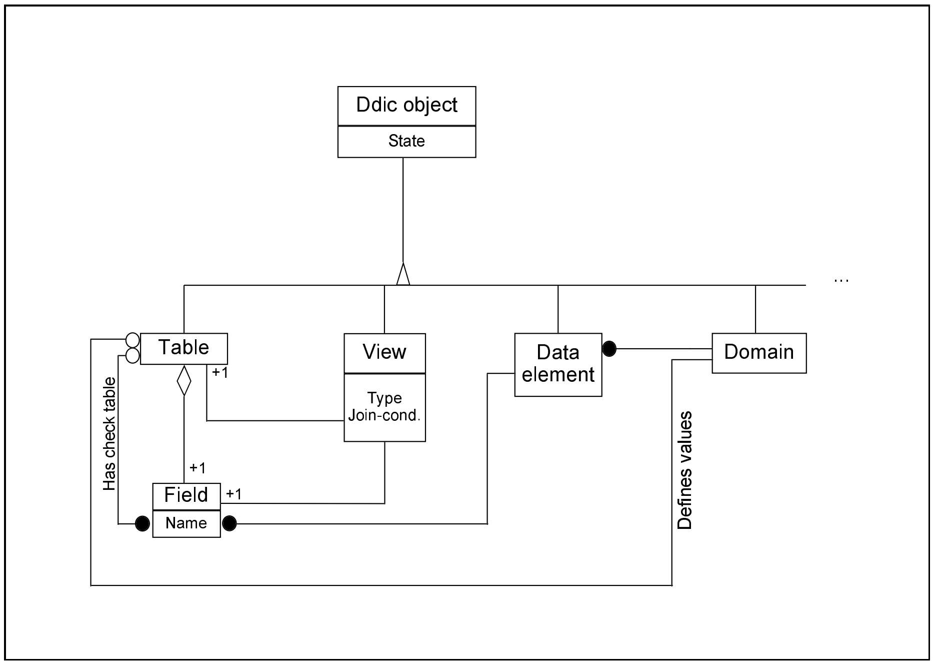


Figure 11: Data Modeling Elements in the R/3 Data Dictionary

The data dictionary itself is conceptually and technically a part of the integrated R/3 repository.

4.1 Data Modeling: Selected Data Dictionary Objects

The R/3 data model adds several concepts to the classical relational data model to better meet the modeling and storage requirements of ABAP/4 applications and to also facilitate schema evolution.

4.1.1 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 semantic domain. For example, 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.

4.1.2 Tables and Structures

The table concept of R/3 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 by enumeration or by reference to the fields of another table or of a structure. A structure is similar to a table, but it does not contain any data (it is a record type, see also Fig. 14). By defining fields through a reference to another structure or table, subsequent schema definitions are propagated automatically from the base tables to application tables and data structures.

A field of a table has a name and an associated data element. A field can refer to a so-called "check table" which defines a foreign key relationship. However, the system does not guarantee referential integrity automatically. The actual R/3 application code 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 (see Fig. 8) can be set, this is shown in the Data Modeler, if appropriate entities for the 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 the structure of standard tables hese are tables predefined in the system as 'hooks' for system extensions.

- Customizing includes are special tables named CI_* (btw. an exception to the rule that the name range for customers is Y* and Z*), which are included by standard tables. This is to offer customers the chance to change a standard table and to ensure that the changes are propagated to subsequent 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.

4.1.3 Views

Views provide a restricted view on data stored in a database. In R/3, a view consists of one primary table and any number of secondary tables.

In an addition to the well-known concept of views in the relational model, 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 (see also Sec. 7).

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 data involved. The data can be maintained by the customizing transactions [SAP AG 1996].

4.2 Process Modeling: Workflows

R/3 allows the user to define business workflows on top of R/3 transactions. The technique used is the same as the one used for EPCs (see Sec. 3).

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 [SAP AG 1996].

5 Programming with ABAP/4 and the DynPro Concept

At the implementation level, the actual business processes are implemented. All applications are written in ABAP/4 (Advanced Business Application Programming Language, 4th generation), 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.

5.1 Implementation-Oriented Data Dictionary Objects

5.1.1 Data Types and Type Groups

R/3 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 has to persist must be converted to a corresponding object of an external type. Table 12 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 corresponding 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 database, they serve as buffers.

Internal table type: An internal table is used to store data during the execution of an ABAP/4 program (more precisely: during the execution of an R/3 transaction). Internal tables are not persistent. If the data has to persist it must be inserted into a database table.

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.

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

Figure 12: External Types in R/3

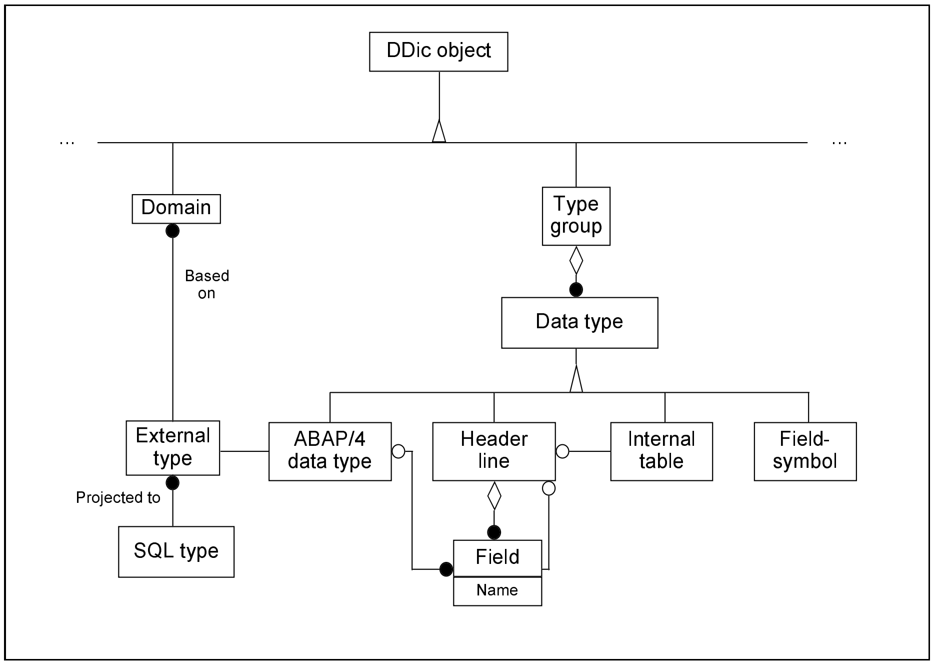


Figure 13: Data Types in the Data Dictionary

5.1.2 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 relationship between (implicit) database locks and explicit lock object operations is explained in Sec. 5.3.5.

The lock mode controls the method by which the users are granted concurrent access to shared data records (identified by their primary key). 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 [SAP AG 1996].

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.

5.1.3 Matchcodes

In the relational data model, tuples are identified by their (key) values. For performance reasons, most R/3 tables have rather short, artificially-created keys, called *id-attributes* in R/3.

Using id-attributes improves system performance but unfortunately they also appear in interactive forms. Users are often requested to identify a tuple by giving its id-attribute, e.g., when processing an incoming order the customer address is requested and the user has to type in the customer-id. The purpose of matchcodes is to help the user find the information by displaying non-identifying attributes, which are useless for the system but meaningful to humans. In the example given, a matchcode could display existing customer-ids and additional information, such as name, street and city.

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

Matchcodes should not be confused with database indices [SAP AG 1996]:

- 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 R/3 help system.

5.2 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, for example, all objects necessary to manipulate funds centers. Figure 15 and 16 explain the concept of development class objects using OMT notation.

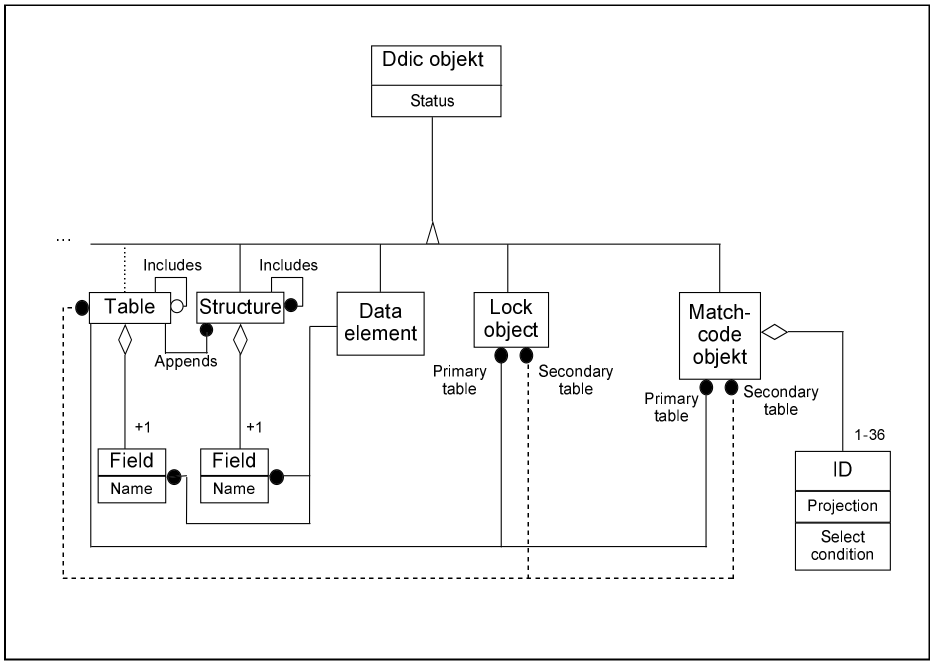


Figure 14: Implementation-Oriented Concepts of the Data Dictionary

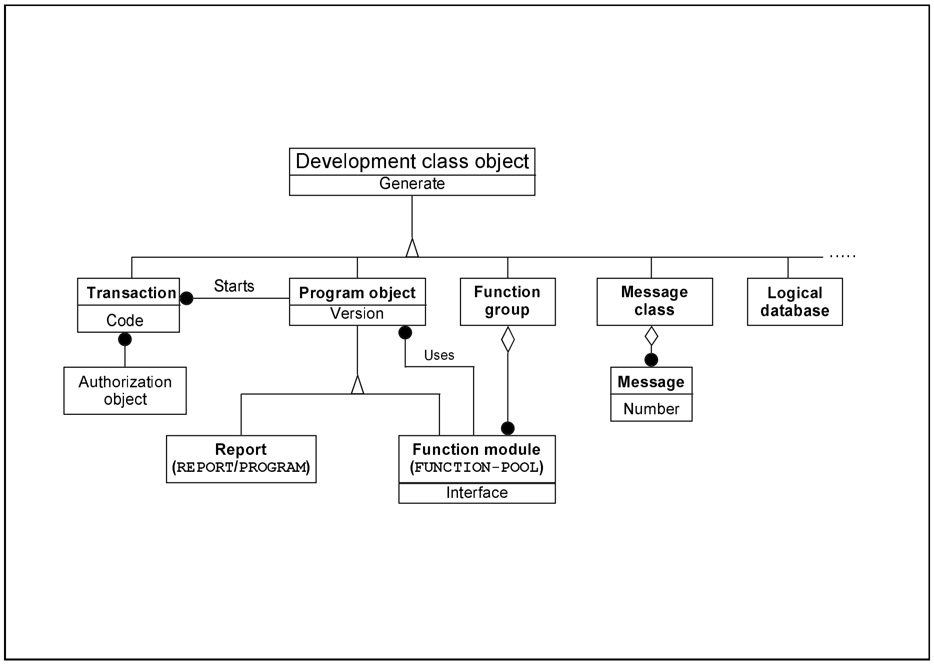


Figure 15: Development Class Objects (1)

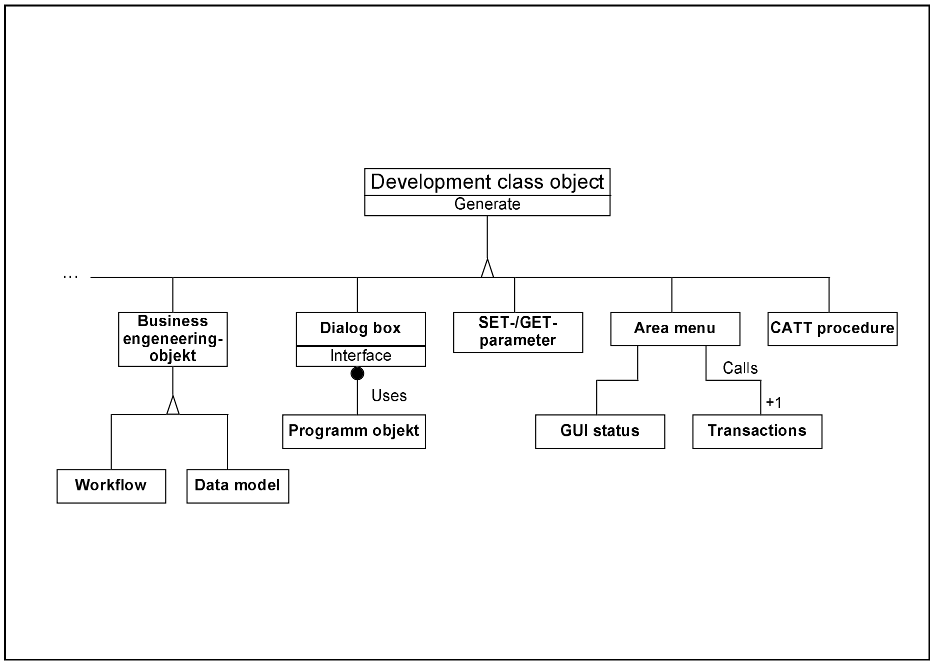


Figure 16: Development Class Objects (2)

5.2.1 R/3 Transactions

An R/3 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 user's point of view 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 [SAP AG 1996].

It is helpful to think of an R/3 transaction as one basic business process which cannot be interrupted or and has to be executed in an all or nothing fashion. R/3 transactions are therefore sometimes called *logical units of work* (LUW). A LUW can involve more than one database transaction (DB-TA).

R/3 transactions can be programmed to comply with the ACID condition (**A**tomity **C**onsistency **I**solation **D**urability). This is achieved by postponing all actual database changes an R/3 transaction has to perform until the end of the R/3 transaction and then performing all changes atomically within one database transaction. Moreover, many R/3 transaction are programmed to work on private in-memory copies of data read atomically at the begin of the R/3 transaction from the database.

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 R/3 transaction is finished and all the changes in the database have been comitted. 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.

5.2.2 Reports

Reports are ABAP/4 programs. They will be explained in more depth in the a following subsection on ABAP/4.

5.2.3 Function Modules

A function module is a routine in ABAP/4. In addition, it has an interface which is stored in the Data Dictionary. The concept of function modules is one of the most important concepts in R/3. There are different kinds 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 (remote function call) from applications outside of the R/3 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 called '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 called 'V1-booking'.

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

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

So called 'batch input' can be provided by the caller of an RFC. This is data stored in special tables and the system interprets the data as interactive input for the system. With batch input everything that can be done interactively can be controlled from client applications.

A *function group* is a set of logically related function modules. A *function library* stores all function modules and allows to search for specific modules.

5.2.4 Messages

A message in R/3 is a string. Messages are prompted to the user in a modal dialog. Dialogs are displayed in separate windows. Every message has a unique number in the message class it is contained in. A message class is a collection of messages which are used in the same program. Nevertheless, it is possible for a program to use multiple message classes.

R/3 supports five 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 whether 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 does not need to be acknowledged.

Abort (A): A critical error has occurred, a re-entering of the information is not possible. The current R/3 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 sensitive, of course. The language is chosen dynamically, based on customization and user login information.

5.2.5 Area Menus

As stated before, ABAP/4 programs can use the full functionality of the R/3 front end, including menus. An area menu is a menu which triggers R/3 transactions. Area menus can call any R/3 transaction defined in the system. Area menus are not assigned to an ABAP/4 program and are invoked by a transaction code.

5.2.6 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 simplify reporting but do not add true expressive power.

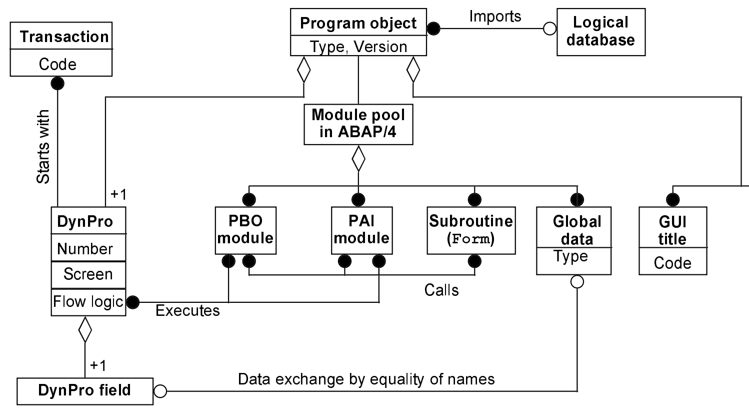


Figure 17: Program Objects in R/3

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 [Kretschmer and Weiss 1997]: Confirmation prompt dialog boxes, dialog boxes for choosing among alternatives, data print dialog boxes, and text display dialog boxes.

SET-/GET-parameters are used to exchange data between R/3 transactions. Their main purpose is to set values for input fields on the screen.

CATT-procedures (Computer Aided Test Tool procedures) are procedures submitted to the R/3 CATT tool which help developers to test newly developed code, customized parts of the system etc. with test data (simulating batch or user input).

5.3 Programming in the Small: Program Objects

A program object (see Fig. 17) 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.

5.3.1 GUI Status

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

The *menu bar* is a container for menus. Two menus are always present: The system menu and the help menu. Menus themselves can contain *menu items*, e.g. ‘Quit’, which trigger

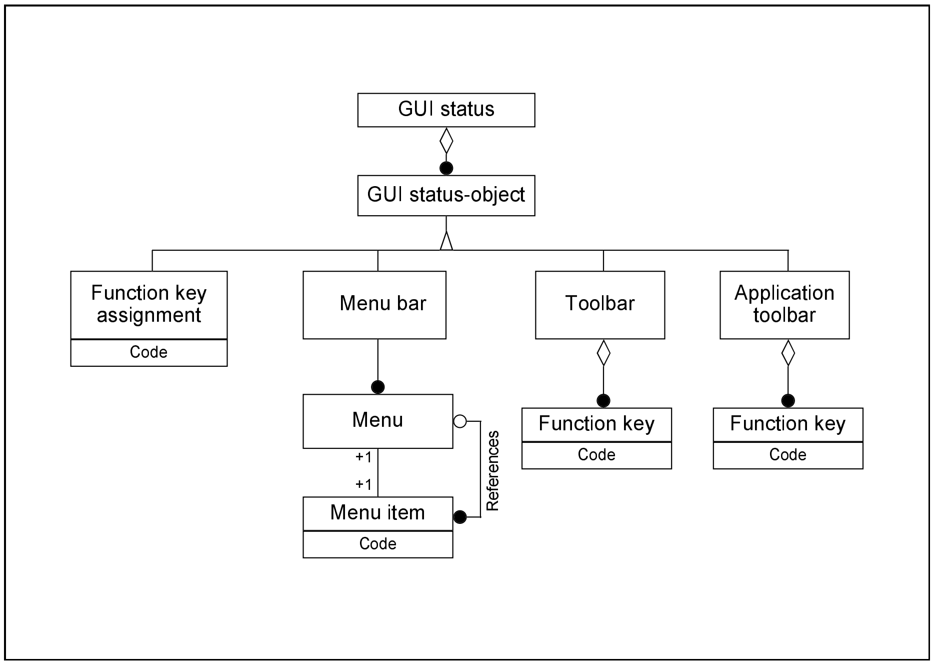


Figure 18: GUI Status Objects in R/3

actions or further submenus, like 'Create Object…', which lead to other menus and menu items. A menu can cascade up to a depth of three levels

The *tool bar* is container for application-independent buttons, the *application tool bar* is a container for application dependent buttons.

5.3.2 Dynpros – Elements of Interactive Transaction

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

The model is essentially screen-oriented, the follwoing cycle is processed for every screen called (DynPro, see Fig. 19 and Fig. 20):

1. At the beginning of the cycle, everything attached to the PBO (Process Before Output) 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 terminated the input,
   - additional information is displayed,
   - another transaction is executed,
   - the EXIT-COMMAND-event is triggered,
   - the PAI (Process After Input) event is triggered, which is the normal case.
4. The actions assigned to the triggered event are performed.

5.3.3 An Example of an R/3 DynPro

To illustrate the 'flavor' of programming interactive transactions in R/3, the DynPro 100 of the function group FM22 will be discussed. 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

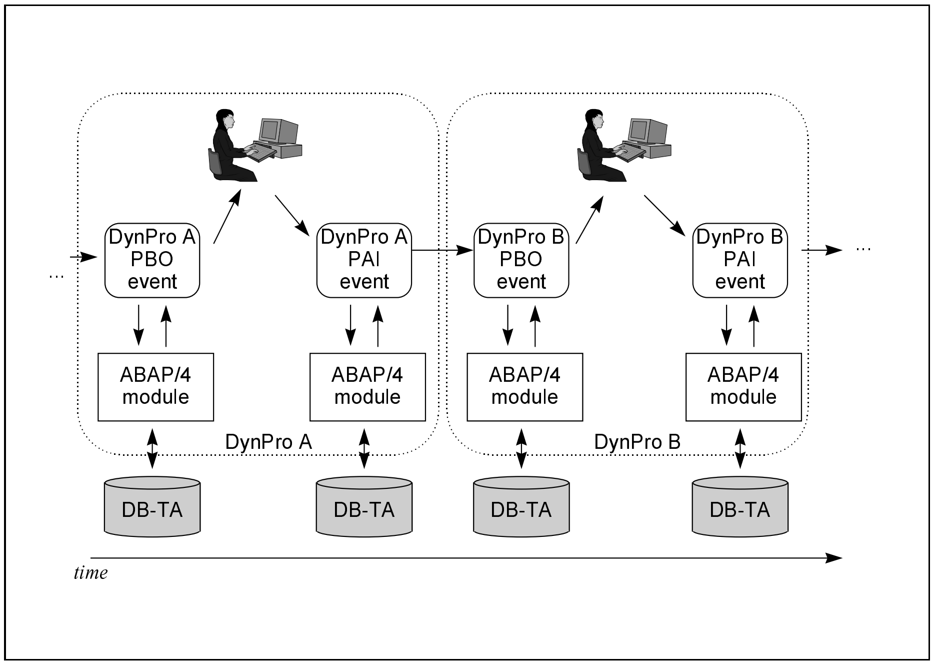


Figure 19: The R/3 Model for Interactive Transactions

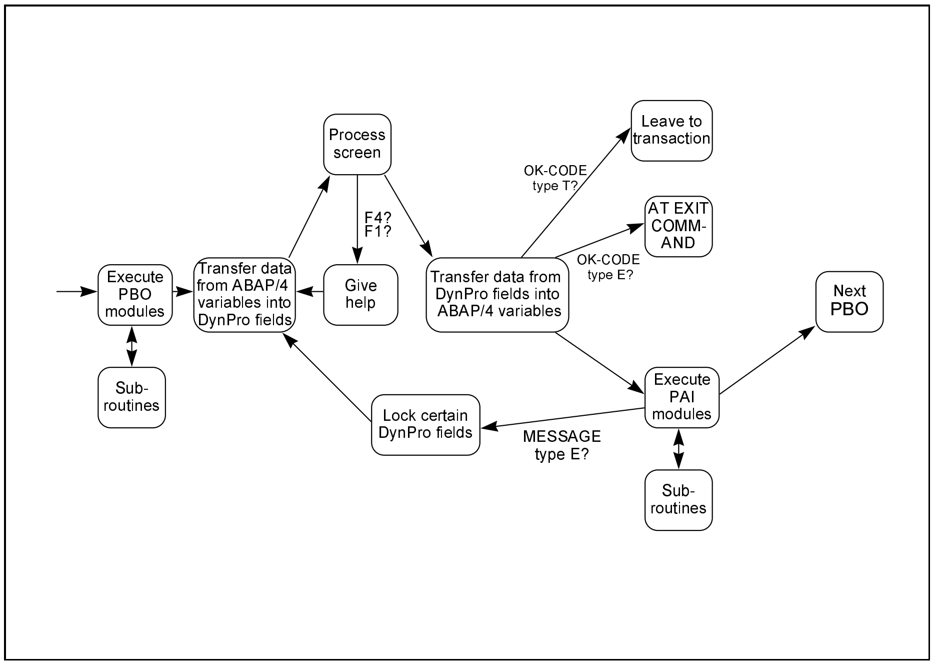
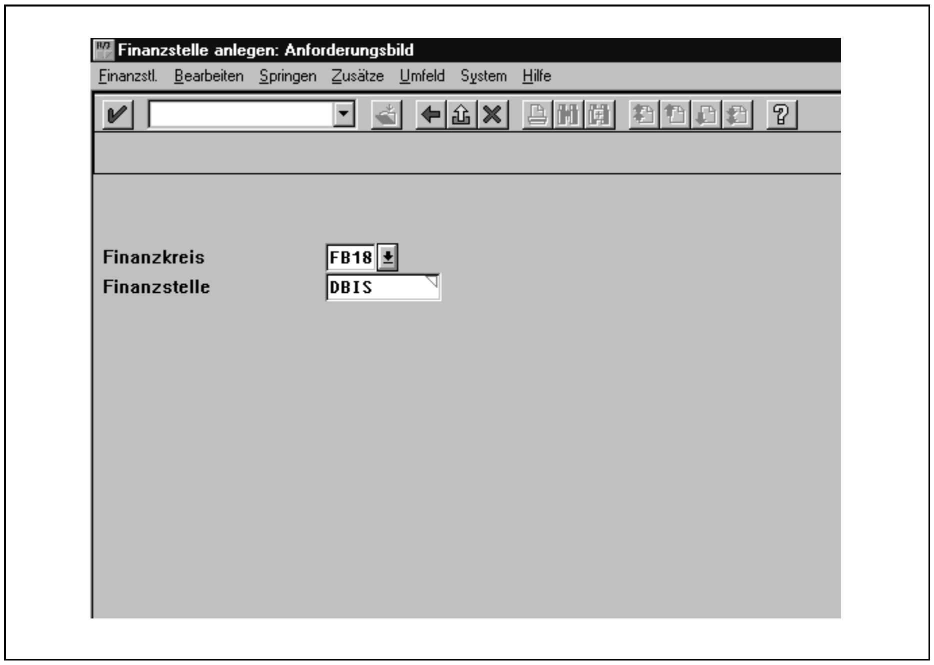


Figure 20: States of the R/3 DynPro Interpreter



Finanzkreis FB18
Finanzstelle DBIS

Figure 21: Transaction FM2I, DynPro 100

center). It is very common in R/3 to have these three transactions (insert/show/update) for a given object type.

In DynPro 100 (see Fig. 21), 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 means that DynPro 100 is shared by multiple transactions. The full source code can be found in appendix B.

5.3.4 Components and Attributes of DynPros

A DynPro (Dynamic Program) consists of several components [SAP AG 1996]:

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

Screen layout specifies which fields should appear in the DynPro and where

Field attributes are properties of each field in the DynPro.

Flow logic specifies which ABAP/4 routines should be called for the DynPro.

These components will be explained using the example of DynPro 100, Transaction FM2I (insert).

DynPro Attributes

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 next DynPro executed will be the Dynpro 100

```
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 22: Flow Logic of DynPro 100, Program SAPLFM22

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

Field Name	*Type*	*Format*	*Length*	*Remark*
IFMFCTR-FIKRS	Text	CHAR	15	string literal 'Finanzkreis'.
IFMFCTR-FIKRS	I/O	CHAR	4	input field for FM area associated with matchcode FIKRS to assist user-input
IFMFCTR-FICTR	Text	CHAR	15	string literal 'Finanzstelle'
IFMFCTR-FICTR	I/O	CHAR	10	input field for funds center associated with matchcode FIST to assist user-input
OK_CODE	OK			function return code (menu selection, abort, ...)

The OK-field serves a special purpose: Buttons are identified by a function code. Whenever a button is pressed, user input is terminated and the function code of the button is stored in the OK-field. When input is terminated by pressing the return-key, the value of the OK-field is SPACE.

Flow Logic The flow logic consists of keywords identifying the beginning of a section to be processed at an event, module calls and error handling.

Consider the flow logic in Fig. 22: 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.

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

Figure 23: Invocation of a Locking Object

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.

5.3.5 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., [Kretschmer and Weiss 1997], [Curran 1996a] or [Matzke 1996].

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. For example, the ABAP/4 statement '**DATA** G_FIKRS **LIKE** FM01-FIKRS **.**' (include file LFM22DEC) defines a variable called G_FIKRS which has 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 uses 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 not be 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 23 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.

```
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 24: Dynamic Screen Modification

Persistence 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 uses a built-in dialect of SQL, the so called *Open SQL* language. Open SQL is similar to standard SQL, 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, R/3 allows to change the screen mask during execution. This is done in a PBO module like the one shown in Fig. 24.

6 Customizing R/3: Concepts and Techniques

Customizing is a method intended for implementing R/3 (at SAP), enhancing of R/3, undertaking a release upgrade or a system upgrade.

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

6.1 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 [SAP AG 1996]. It is a high-level description of what is to be done.

The procedure model consists of four phases:

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.

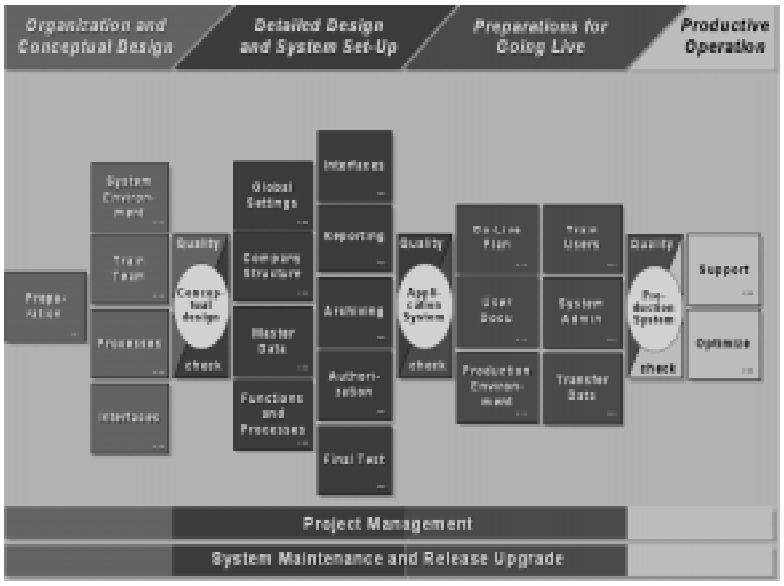


Figure 25: Customizing R/3: The Procedure Model

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.

Figure 25 shows these steps of the procedure model as defined for R/3 3.0C [SAP AG 1996].

6.2 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 [SAP AG 1996]:

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.

IMGs are created by the execution of special R/3 transactions.

6.3 Customizing R/3

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 some 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.

A big problem during R/3 customization is to find out which R/3 modules (FI, TR, HR,...) and functions provide the required functionality. For example, in order to implement budget controlling, free-lance employees are modeled best as suppliers.

The basic top-down-approach to solve customization problems consists of the following steps:

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

Customizing is a time-consuming process which involves a lot of reading and experimentation. The sheer amount of documentation for customizing and the problem of finding out the required information and functions is a major obstacle to quickly introduce R/3 into an enterprise with already established business rules and business practice.

6.4 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 outdated 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 R/3 original objects into client 0, the default client. From there, the changes can be transported to the required destination in the system. Development classes are the basic objects that can be transported.

7 R/3's Process and System Architecture

Despite the fact that R/3 uses relational technology and a client/server architecture, many of its process and distribution concepts clearly show their origin in R/2's centralized mainframe world.

7.1 Client/Server 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. SAPGUI does not adhere to the style guidelines of its host system.

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 26 shows the three layers and the communication between the most important components (example taken from [Will *et al.* 1996]).

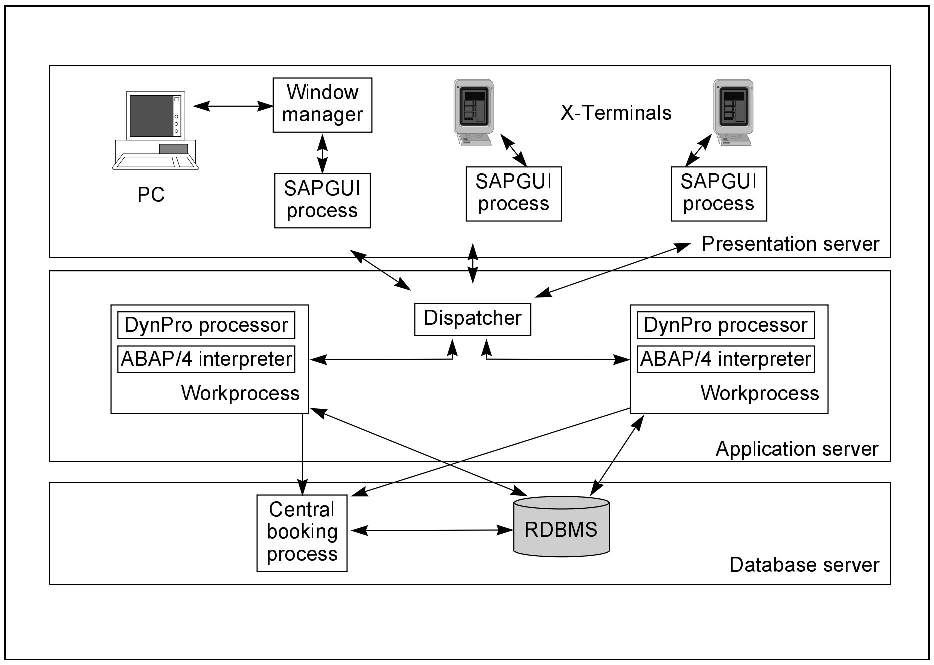


Figure 26: The three-tiered R/3 System Architecture

TCP/IP is used as the communication protocol within R/3. With LU 6.2 it is possible to communicate with IBM mainframes. Fig. 27, adapted from [Buck-Emden and Galimow 1997], shows the use of network protocols by R/3.

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 minimizes the amount of data to be exchanged for a switch from the current screen to the next and during bulk data display.

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

7.2 Process Architecture

As depicted in Fig. 28, 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.

Every application server consists of one message server which handles the communication with other system servers. There can be at most one gateway server in an 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 task handler, 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, 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.

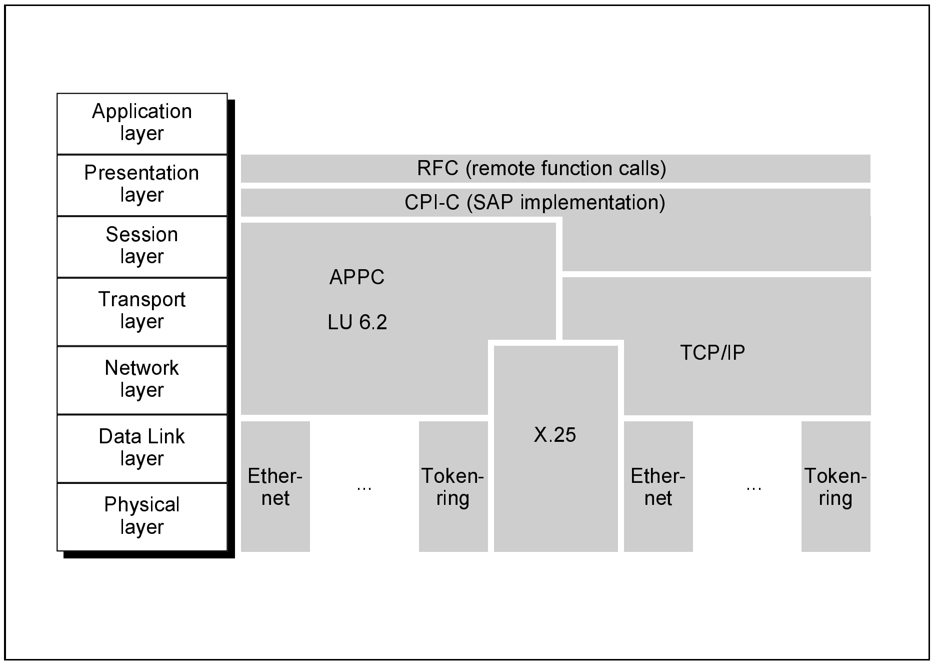


Figure 27: Communication Protocols Supported by R/3

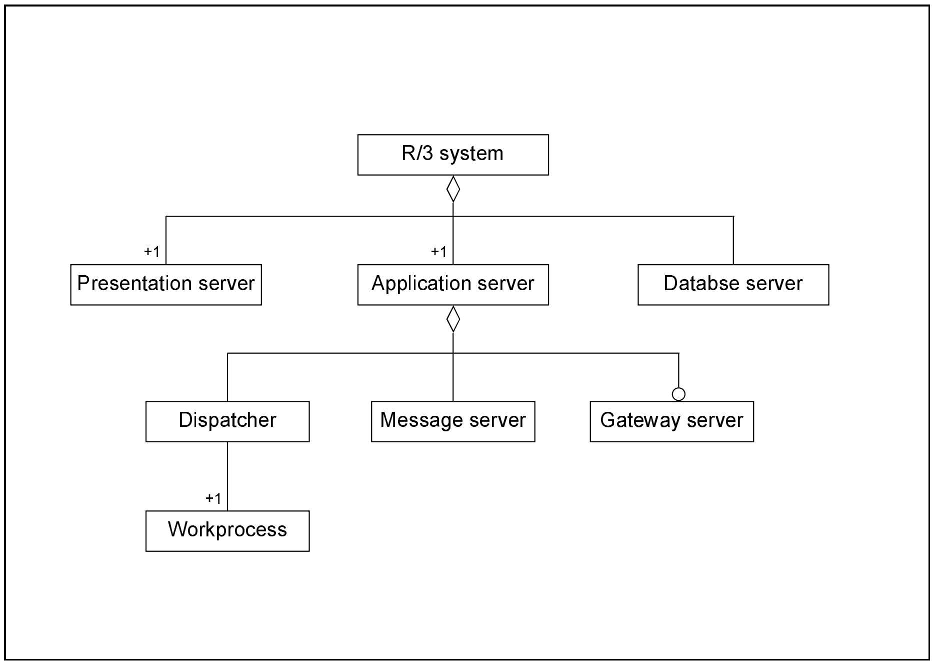


Figure 28: R/3 Process Architecture in OMT Notation

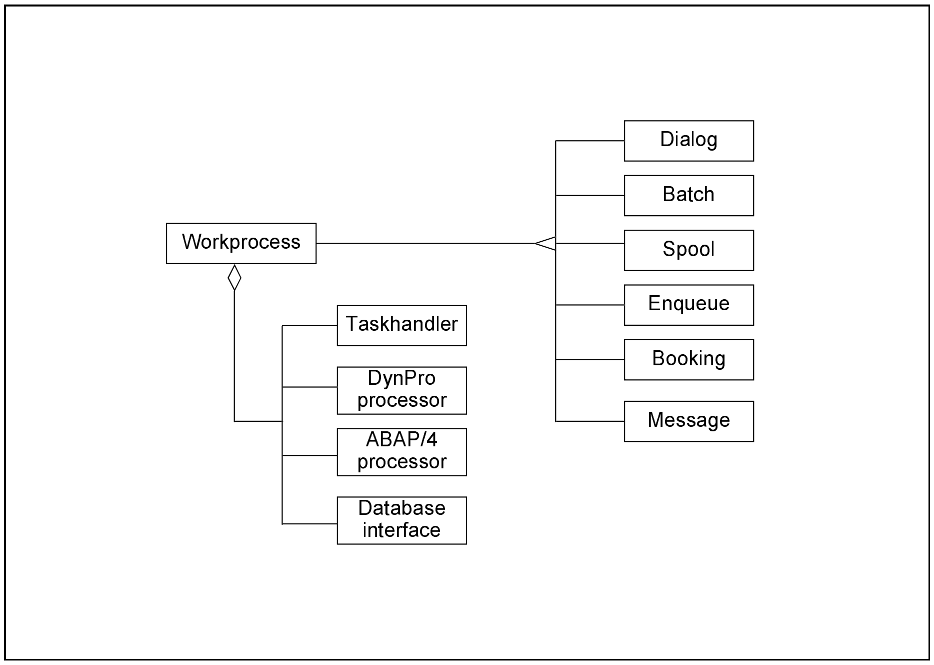


Figure 29: R/3 Work Processes

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.

7.3 External Gateways

R/3 has technical interfaces to external systems at virtually all layers.

- Presentation layer: Intelligent Terminal
- Application layer: Files, CPI-C, RFC, OLE (Windows only), email, EDI, Business API
- 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. BAPIs are nothing else than conventional RFCs.

A Overview: R/3 Terminology and Concepts

Figure 30 summarizes the R/3 client/server architecture by means of an OMT diagram.

Figure 31 gives an overview of all conceptual R/3 objects mentioned in this tutorial and shows their static relationships.

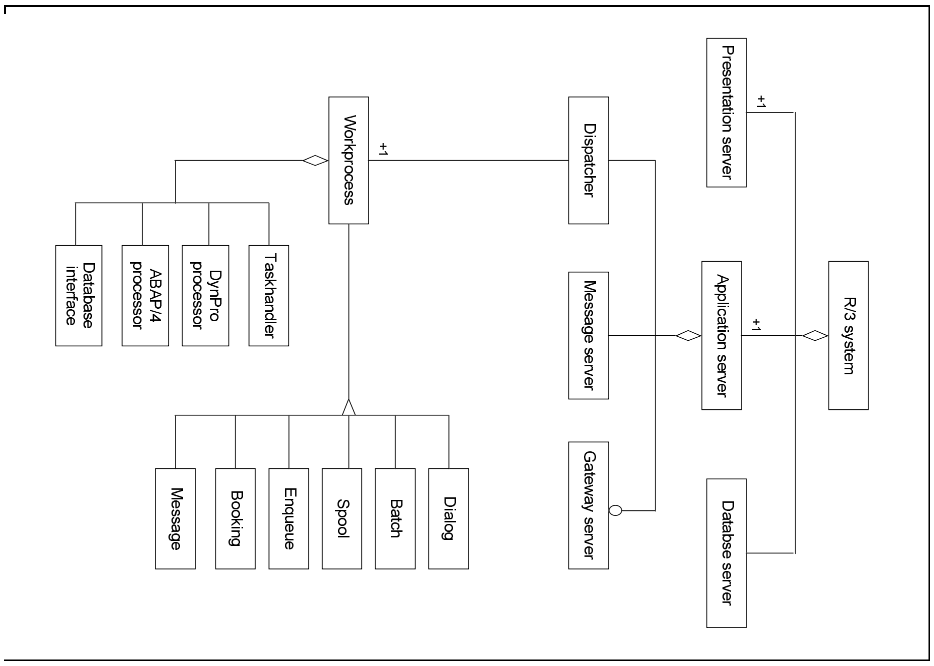


Figure 30: R/3 Architecture

B Source Code of FM22, DynPro 100

The complete source code can be found in the program SAPLFM22 (Release 3.0). The code given here is a subset which is executed by DynPro 100. The reader should get an impression of what ABAP/4 programs look like.

The original comments (and several variable names) are in German and have been translated.

```
* Program header
FUNCTION-POOL FM22 MESSAGE-ID FI.
...
*-----------------------------------------------------------------------
* DEC - include local table data and field-symobol definitions
*-----------------------------------------------------------------------
INCLUDE LFM22DEC.
...
*-----------------------------------------------------------------------
* Interal tables
*-----------------------------------------------------------------------
* Internal table for funds center used for data storage during
* transaction processing and to propagate data from on Dynpro to the
* next
DATA: BEGIN OF I_FMFCTR OCCURS 10.
        INCLUDE STRUCTURE IFMFCTR.
DATA: END OF I_FMFCTR.

* Backup of non-updated value
DATA: BEGIN OF I_OLD_FMFCTR.
        INCLUDE STRUCTURE IFMFCTR.
DATA: END OF I_OLD_FMFCTR.
...

TABLES:
```

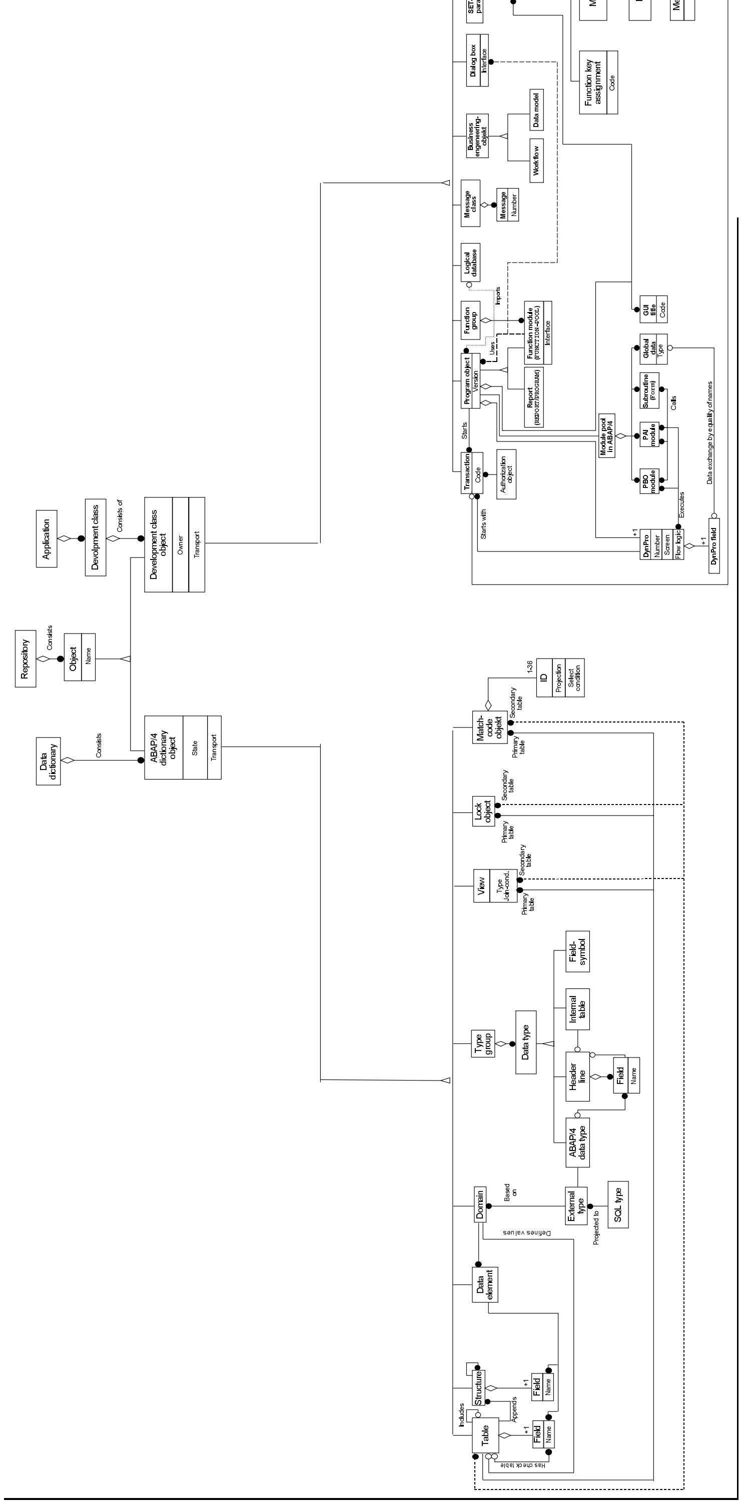


Figure 31: Overview of the R/3 Terminology

```
        "/ FM area
        FM01,
        "/ Text for FM area
        FM01T,
        "/ Funds center
        FMFCTR,
        "/ Internal table for Dynpro fields of the Funds center
        IFMFCTR.

* structure fo a funds center
DATA: BEGIN OF FFMFCTR.
        INCLUDE STRUCTURE IFMFCTR.
DATA: END OF FFMFCTR.

DATA:
      "/ FM area identification
      G_FIKRS           LIKE FM01-FIKRS,
      "/ Funds center identification
      G_FICTR           LIKE FMFCTR-FICTR,
      "/ Transaction code
      G_TCODE           LIKE SY-TCODE,
      "/Flag to signal that the transaction / function module has been
      "/ called indepndently
      FLG_CALLD         LIKE SY-CALLD  VALUE 0.
...

MODULE D0100_MODIFY_SCREEN.
*---------------------------------------------------------------------*
*    Dynamic screen modification for Dynpro 0100:                     *
*    - truncate the output size of the field FM area to 10 characters  *
*    - disable input to the FM area identification field if the       *
*      screen is called from within another transaction               *
*---------------------------------------------------------------------*

"/ Loop over all fields defined for the Dynpro
   LOOP AT SCREEN.
     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
...

MODULE D0100_EXIT.
*---------------------------------------------------------------------*
*      Functions called if the the current processing is terminated    *
*      without executing the checks of the PAI modules.               *
*      LEAVE TO TRANSACTION releases all locks held by the transaction *
*---------------------------------------------------------------------*
  "/ get the return code determined by the user
  "/ indicate that procssing can continue
  SAV_OK_CODE = OK_CODE.
  CLEAR OK_CODE.
  "/ Evaluate the saved return code
  CASE SAV_OK_CODE.
    "/  ENDE = quit
    WHEN 'ENDE'.
         SET SCREEN 0.
         LEAVE SCREEN.
    "/  EINS = branch to transaction "create funds center"
    WHEN 'EINS'.
         "/ branc to this transaction
         LEAVE TO TRANSACTION TR_FICTR_INS.
  ENDCASE.

ENDMODULE. "/ D0100_EXIT
...

* check for special characters:
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.
*  keep key of the FM area
   CONDENSE IFMFCTR-FIKRS NO-GAPS.
   G_FIKRS = IFMFCTR-FIKRS.
   "/ store the FM center
   CONDENSE IFMFCTR-FICTR NO-GAPS.
   G_FICTR = IFMFCTR-FICTR.
   " / if called from another transaction (here a graphical display of an FM area)
   IF ( FLG_CALLD = CON_INDEPENDANT_TA ).
      "/ initialize variables using values passed from the preceding graphical display
      SELECT SINGLE *
      FROM FM01
      WHERE FIKRS = G_FIKRS.
      "/store object number of the FM area
      G_FMA_OBJNR = FM01-OBJNR.
      "/set a flag to force an insertion without a copy in a later
      "/ processing stage
      FLG_COPY = CON_NEIN.
      "/initialize variable (sic!)
      CLEAR G_REF_FICTR.
   ENDIF.
ENDMODULE. "/ D0100_DB_KEY_NOTICE
...

MODULE FMFCTR_ENQUEUE INPUT. "/call a function to lock the funds center
  PERFORM FMCTR_ENQUEUE.
ENDMODULE. "/ FMFCTR_ENQUEUE

FORM FMFCTR_ENQUEUE.
"/ if ( creation or update of a funds center )
```

```
   CHECK ( ( G_TCODE = TR_FICTR_INS )
      OR   ( G_TCODE = TR_FICTR_UPD ) ).
      "/ request a lock for this funds center
      CALL FUNCTION 'ENQUEUE_EFMFCTR'
           EXPORTING
             FIKRS    = G_FIKRS
             FICTR    = G_FICTR
           EXCEPTIONS
             FOREIGN_LOCK   = 1
             SYSTEM_FAILURE = 2.
      CASE SY-SUBRC. "/ exception handling (-> inform user)
        WHEN 1.  "/already locked by another user (FOREIGN_LOCK)
             MESSAGE E641 WITH G_FICTR.
        WHEN 2.  "/ SYSTEM_FAILURE
             MESSAGE A521 WITH G_FICTR.
      ENDCASE. "/ SY-SUBRC
ENDFORM. "/ FMFCTR_ENQUEUE
...

MODULE FMFCTR_LESEN.
  PERFORM FMFCTR_LESEN. "/ call a function to read the funds center
ENDMODULE. "/ FMFCTR_LESEN.

FORM FMFCTR_LESEN.
  DATA: L_FMFCTR_EXISTS LIKE CON_JA. "/Funds center already exists
  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 ). "/ Error message and stay within this Dynpro
    MESSAGE E642 WITH P_FIKRS P_FICTR.
  ...
ENDFORM. "/FMFCTR_LESEN_INS
...

MODULE D0100_OK_CODE INPUT.
*-----------------------------------------------------------------------*
* Evaluate user commands of DynPro 100                                  *
*-----------------------------------------------------------------------*

  "/ get the return code determined by the user
  "/ indicate that procssing can continue
  SAV_OK_CODE = OK_CODE.
  CLEAR OK_CODE.
  "/ evaluate the saved return code
  CASE SAV_OK_CODE.
    "/ ENTER
    WHEN SPACE.
      "/ Branch to successor DynPro 200
      SET SCREEN 200.
      LEAVE SCREEN.
  ENDCASE.

ENDMODULE. "/ D0100_OK_CODE
```

References

Bancroft 1996: Bancroft, N. H. *Implementing SAP R/3.* Prentice Hall, Englewood Cliffs, New Jersey, 1996.

Buck-Emden and Galimow 1997: Buck-Emden, R. and Galimow, J. *The Client/Server Technology of the SAP R/3 System.* Addison-Wesley Publishing Company, 1997.

Bundesministerium für Forschung und Technologie 1994: Bundesministerium für Forschung und Technologie. Initiative zur Frderung der Software-Technologie. In *Wirtschaft, Wissenschaft und Technik*, August 1994.

Curran 1995: Curran, T. A. ABAP/4 Development Workbench. Technical report, TCManagement Inc., Oderstrasse 28, 81677 Munich, Germany, January 1995. Also available through SAP AG, Neurottstrasse 16, 69190 Walldorf, Germany.

Curran 1996a: Curran, T. *Client/Server Development With SAP's ABAP/4 Development Workbench 3.0.* Prentice Hall, Englewood Cliffs, New Jersey, 1996.

Curran 1996b: Curran, T. *Using SAP,s R/3 Client Server Business Process Blueprint Tool.* Prentice Hall, Englewood Cliffs, New Jersey, 1996.

de Brian and other 1996: Brian, Gareth de and other. *Introduction to ABAP/4 Programming for SAP.* Asap World Consultancy, 1996.

Fowler and Scott 1997: Fowler, M. and Scott, K. *UML Distilled - Applying the Standard Object Modeling Language.* Addison-Wesley Publishing Company, 1997.

H. 1997: H., Muneer. *Designing and Implementing SAP R/3.* Sybex, 1997.

Hernandez 1997a: Hernandez, J. *Administering SAP.* AP Professional, 1997.

Hernandez 1997b: Hernandez, J. *The SAP R/3 Handbook.* McGraw Hill, 1997.

Kretschmer and Weiss 1997: Kretschmer, R. and Weiss, W. *Developing SAP's R/3 Applications with ABAP/4.* Sybex, 1997.

Matzke 1996: Matzke, B. *ABAP/4 - Die Programmiersprache des SAP-Systems R/3.* Addison-Wesley Publishing Company, 1996.

Rumbaugh et al. *1991:* Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., editors. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

SAP AG 1996: SAP AG. Online Documentation SAP System R/3, Release 3.0C. Technical report, SAP AG, Neurottstrasse 16, 69190 Walldorf, Germany, March 1996.

SAPHome 1997: WWW Home Page of the SAP AG, Germany. http://www.sap-ag.de/, 1997.

Schuessler 1997: Schuessler, T. *Integrating SAP's R/3 with Visual Basic and OLE.* Sybex, 1997.

Taylor 1996: Taylor, R. *Using SAP R/3 - Special Edition.* Que Corp, 1996.

Will et al. *1996:* Will, L., Hienger, Ch., Strassenburg, F., and Himmer, R. *R/3-Administration.* Addison-Wesley Publishing Company, 1996.